Criando Aplicativos com o CakePHP

Ribamar FS

Fortaleza, 21 de maio de 2018

Sumário

0 – MVC

Ambiente Unix Produtivo

Ambiente com quase tudo numa única janela
Uma boa sugestão para quem usa Linux/Mac é usar o terminator, que pode dividir a tela do terminal em duas ou mais com praticidade
https://www.vivaolinux.com.br/artigo/Terminator-Multiterminais-em-Janela-Unica
Minha disposição preferida:
- Abro o terminator pela linha de comando do terminal ou pelo menu
- Divida a tela ao meio na vertical com Ctrl+Shift+E
- Depois clico na tela da direita (ou teclo em Alt+Seta direita) e a divido em duas na horizontal com Ctrl+Shift+O
- Na área maior da esquerda eu abro o diretório do blog onde estou trabalhando para digitar os comandos do laravel
    /backup/www/laravel/blog (no meu caso)
- Na área superior da direita eu abro a console do mysql e executo
    mysql -uroot -p
- Na área inferior da direita executo o servidor web na pasta do blog
    php artisan serve

Dica: observe que ainda pode ajustar a largura e algura das áreas usando o mouse nas barras de rolagem.
Também podemos maximizar uma região apenas clicando com o botão direito do mouse e Maximizar assim como podemos restaurar as áreas anteriores
Com isso somente preciso sair do terminator para acessar o navegador, com ALT+TAB.
Se vamos codar, vamos aprender a codar na linha de comando não somente o PHP e o Laravel mas também o MySQL e outros SGBDs.
Lembre "Conhecimento é poder" e nos dá poder de criação.
Assim fico mais confortável e produtivo.

Gerenciador de arquivos modo texto. Claro que não é confortável como o nano ou nautilus, mas quebra um galho

E por falar em produtividade existe um ótimo gerenciador de arquivos para o modo texto:
sudo apt install mc

Navegador para o modo texto
sudo apt install lynx
lynx localhost:8000

Pena que não suporte CSS, mas funciona e quebra um galhão algumas vezes
Outra opção bem produtiva é programar usando o Visual Studio Code (https://code.visualstudio.com/) ou similar como Atom (https://atom.io/) ou Bracket (http://brackets.io/). Eles contam com muitos e bons recursos e suportam os principais SO. Também existe o Sublime Text (https://www.sublimetext.com/).

Editor de código com debugger, suporte a diversas linguagens de programação, extensões para muita coisa. Um console para execução de comandos e suporte ao git.

O CakePHP usa o padrão MVC para organizar seu código.

Model

Nesta camada são realizadas as operações de validação, leitura e escrita de dados no banco de dados. É responsável por salvar e receber dados do banco de dados, como também efetua diversos processamentos com os dados.

A camada Model é o cérebro da aplicação. Basicamente qualquer coisa para ler, alterar, salvar ou excluir dados é nesta camada. A camada Model é a camada que sofreu a maior transformação na versão 3.

Nas versões anteriores o Model era representado por classes Model e Behavior.
Na versão 3 agora temos as classes Table, Entity, Behavior e Query.

As classes Table representam os dados armazenados, normalmente uma tabela e são responsáveis por encontrar, salvar e validar nossos dados. Também são responsáveis por manipular grandes massas de dados, como cálculo dos totais de uma coleção de dados.

As classes Entity formam um conceito inteiramente novo e elas representam dados simples, isto é, apenas um registro na tabela. Usando classes Entity permite a você criar campos de dados virtuais muito complexos.

Classes Behavior funcionam exatamente como as da versão anterior, complementando as classes Table e Entity.

A nova classe Query dá ao CakePHP 3 uma incrível e poderosa maneira de interagir com bancos de dados relacionais padrões, através do que é chamada de uma interface fluente.

Exemplo da nova classe Query:

```
/**
 * Assume that $connection and $table have already been set earlier
 * $connection is the database connection (A Cake\Database\Connection object)
 * $table is the table object to run the query on (A Cake\ORM\Table object)
 */
$query = new Cake\ORM\Query($connection, $table);
$query
    ->select([
        'id',
        'first_name',
        'last_name',
        'email_address'
    ])
    ->where([
        'status' => true,
```

```
        'email_address LIKE' => '%@gmail.com'
    ])
    ->order(['last_name' => 'ASC']);
```

A consulta acima deve gerar o SQL seguinte:
SELECT id, first_name, last_name, email_address FROM users WHERE status = 1 AND email_address LIKE '%@gmail.com' ORDER BY last_name ASC

Uma boa prática é trazer para esta camada tudo que diz respeito às regras de negócio, como cálculos ou validações de integridade de dados.
Controller

É o responsável pela integração entre as camadas Model e View. Basicamente a View irá realizar uma solicitação para o Controller como por exemplo uma coleção de dados ou a solicitação de remover algum item do banco e o Controller, por sua vez, irá enviar a instrução para a camada Model executar.

Nesta camada (Controller) também podemos realizar verificações que não se referem às regras de negócio, visto que a boa prática é manter as regras de
negócio no Model.

O Controller é o coração do aplicativo. Esta camada é a que teve a menor quantidade de alterações da versão 2 para a 3. É responsável por controlar o
fluxo de dados da aplicação: recebe requisições da view e as envia para o model. Então recebe do model, efetua algum processamento e envia de volta a resposta para a view.

Deve ter o mínimo de código.

Uma solicitação de entrada do cliente é enviada para uma ação de um controller específico, que tem a lógica necessária para determinar quais dados são
necessários e em qual tabela estariam. O controller verifica se o cliente solicitante tem permissão para acessar tais dados e qual view deve ser usada
para a saída dos dados. A camada controller consiste de classes Controller e classes Component. Um componente tem função parecida com a dos Behavior para os Model, eles auxiliam os Controllers, assim como os Behavior auxiliam os Models. O componente encapsula código que deve ser compartilhado com os Controllers.
Do livro do CakePHP 3. Como funciona uma requisição:
- O dispatche/despachante CakePHP usa o roteador/router para determinar qual ação do controller deve ser chamado para servir a solicitação de entrada
- O controller, então, verificar se há detalhes de autenticação (se necessário) e quaisquer parâmetros de solicitação, tais como IDs, extensões de arquivos, etc.
- A ação/action do controller solicita os dados necessários da camada de modelo
- A camada Model recupera os dados solicitados a partir do armazenamento de dados, manipula e formata conforme necessário e retorna os dados para o controlador
- O controlador então passa os dados para a camada de visão
- A camada View torna os dados em HTML (ou JSON, PDF, XML, etc.)
- Finalmente, o controlador volta a saída da camada View para o dispatcher, que envia para o cliente.

View

É a camada responsável por tudo que é visual, páginas, formulários, listagens, menus, o HTML. Tudo aquilo que interage com o usuário deve estar presente nesta camada.

Representadas por HTML, classes Helpers, View, View Cells

A View não realiza operações diretamente com o banco de dados nem trata diretamente com o Model. Ela as solicita e e exibe através do Controller, que intermedia suas solicitações com o Model.

No CakePHP 3 as views agora ficam no diretório Template

O diretório View contém apenas classes gerais.

Se o Model é o cérebro da nossa aplicação, então a View é a pele.

1 - Instalação do CakePHP 3

Justificativa do CakePHP

Frente à boa quantidade de frameworks em PHP quero justificar a minha adoção do CakePKP. A adoção de uma ferramenta depende da finalidade e de vários aspectos. No meu caso, criar aplicativos de forma produtiva e que implementem com facilidade ACL foram os dois pontos mais importantes para que eu adotasse o CakePHP como meu framework. Estive experimentando os principais: Zend, CodeIgniter, Laravel, Yii entre outros. Para meu caso, o CakePHP foi o mais produtivo, especialmente por conta da espetacular ferramenta de geração de aplicativo “bake” e de sua estrutura de Autenticação e Autorização que me facilitaram a criação do plugin cake-acl-br.

O CakePHP foi concebido para tornar tarefas de desenvolvimento web mais simples e fáceis, pois fornece uma caixa de ferramentas completa para você poder começar.

Pré-requisitos

- Web server como Apache 2 com mod_rewrite
- PHP 5.6 ou superior
- Extensões mbstring e intl
- Um dos 4 SGBDs suportados
- Composer (opcional mas recomendado)

Instalação do Composer

- Se tens a versão 7 ou superior do PHP instale assim:
sudo apt-get install composer

- Para a versão 5.x assim:
sudo su
curl -sS https://getcomposer.org/installer | php

Colocar no path do Linux
mv composer.phar /usr/local/bin/composer

Instalando o CakePHP/Criar Aplicativo

composer create-project --prefer-dist cakephp/app nome_app

Como o comando é grande e pode dar trabalho de memorizar, vamos criar um link (no Windows crie um bat com %1), no Linux para receber parâmetro é $1.

sudo nano /usr/local/bin/comp

Cole a linha
composer create-project --prefer-dist cakephp/app $1

sudo chmod +x /usr/local/bin/comp

Instalar uma versão diferente da atual, a 3.5.13, por exemplo:

composer create-project --prefer-dist cakephp/cakephp:3.5.13 blog

Observação importante

Criei dois scripts sql para que tudo ande como planejado. Use estes scripts caso tenha insegurança do que está fazendo.

Execute assim:

cd /var/www/html

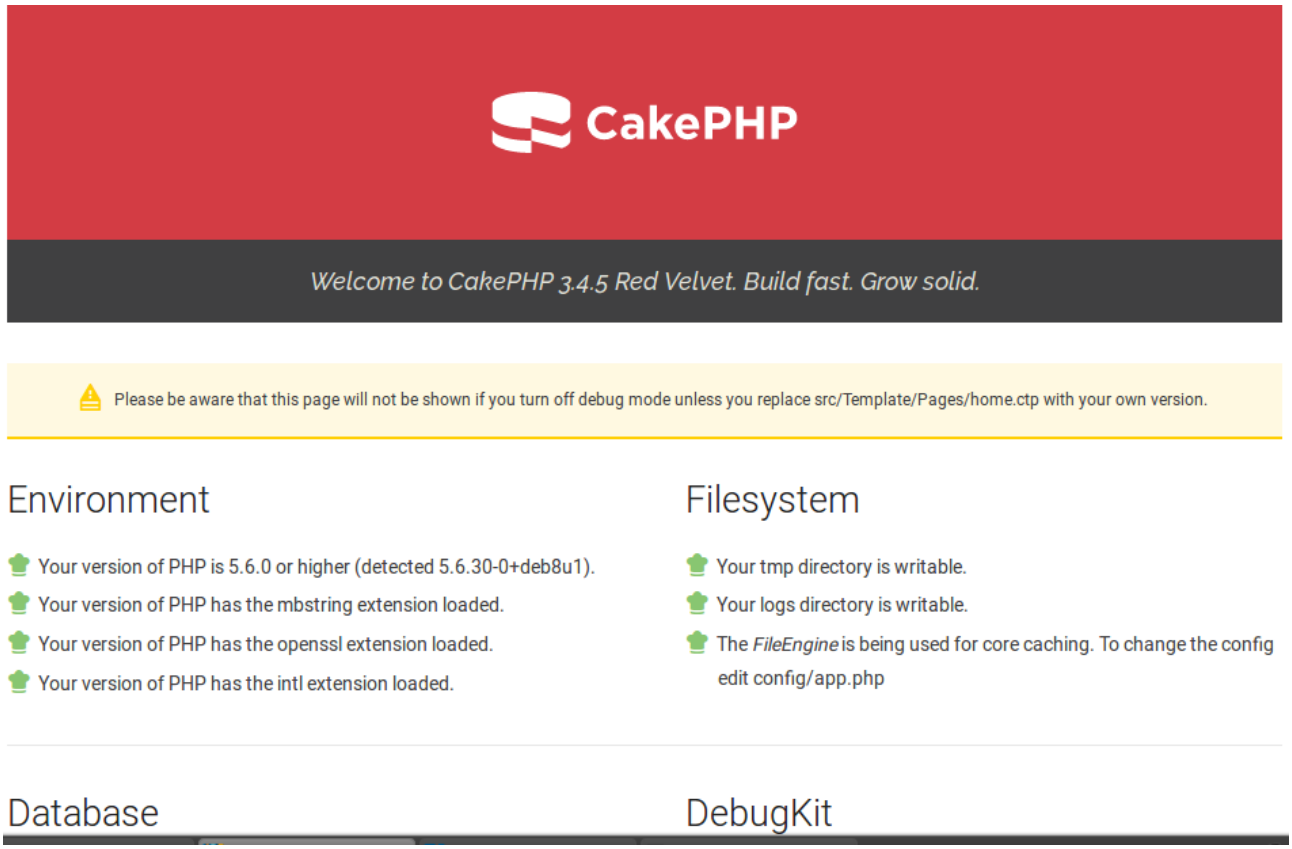
comp cliente

Ao finalizar o Cake já terá trazido todo o seu código para a pasta /var/www/html/cliente.
E já vem com um controller default para que vejamos algo.

Chame pelo navegador

http://localhost/cliente

Aparecerá então a tela abaixo:



Configurações do banco e do routes em config/app.php

Na primeira vez que seguir este roteiro, caso tenha insegurança, crie o banco de dados e importe o script que estou apresentando.

Para criar um aplicativo de testes, apenas para testar este roteiro, estou oferecendo dois script de bancos de dados (clientes), um para mysql e outro para postgresql.
Crie o banco chamado **clientes** usando um destes scripts.

Configuração do Banco

Abra o script config/app.php e faça as alterações abaixo.

Se for usar o MySQL basta alterar estas linhas:

```
'username' => 'root',
'password' => 'mysql',
'database' => 'estoque',
```

Se for usar o PostgreSQL, precisa mudar mais esta linha, que fica acima das 3:

```
'driver' => 'Cake\Database\Driver\Postgres',
```

E adicionar abaixo das 3, mas somente se usar esquemas:

```
'schema'   => 'nome_esquema',
```

Routes

Para que ao abrir o aplicativo automaticamente ele mostra o controller Clientes

Em config/routes.php altere a linha abaixo:

Originalmente é assim a alinha:

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Mudei para

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Geração de Código com o Bake

Depois de configurados o app.php e o routes.php. O script criou 3 tabelas.

cd cliente

```
bin/cake bake all groups
bin/cake bake all users
bin/cake bake all clientes
```

Sabendo a versão do CakePHP instalada pelo seu console

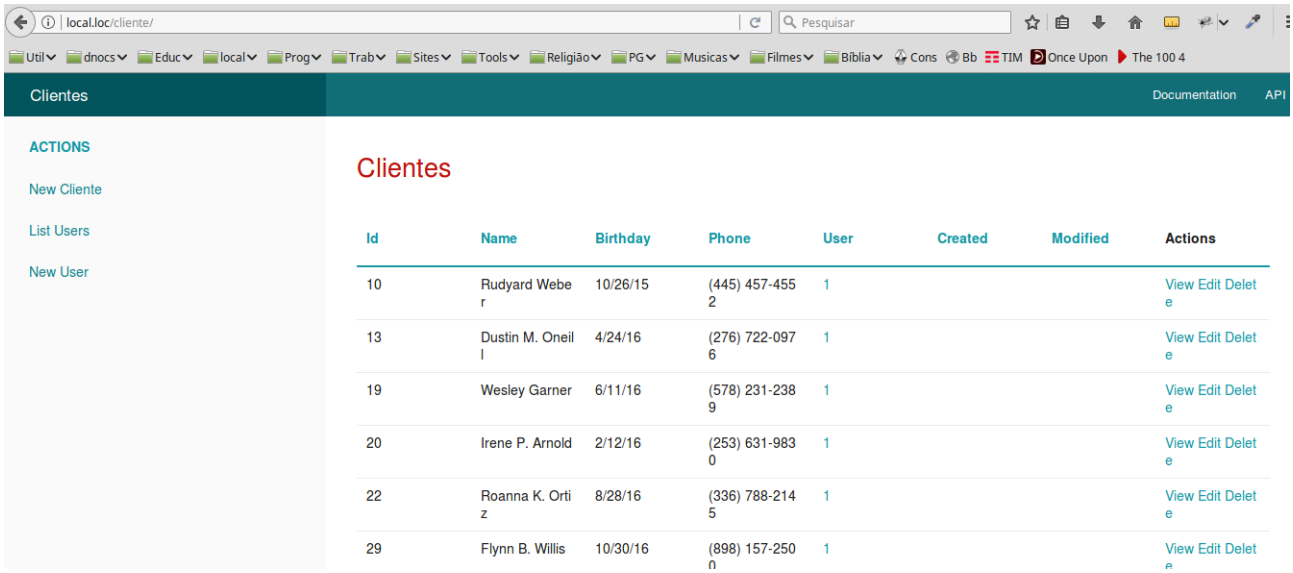
Visualize o arquivo

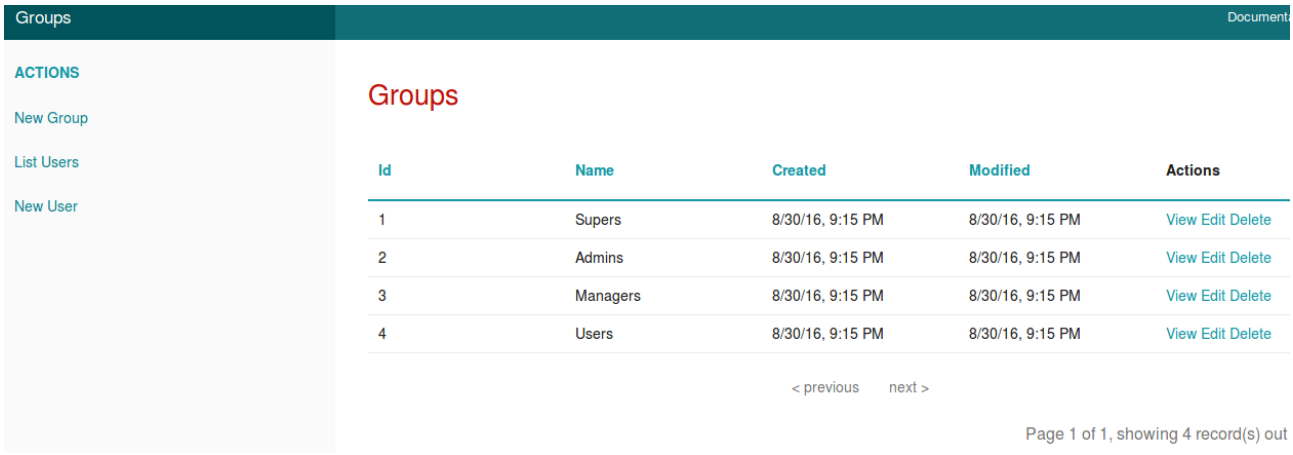
vendor/cakephp/cakephp/VERSION.txt
Acesse pelo Navegador

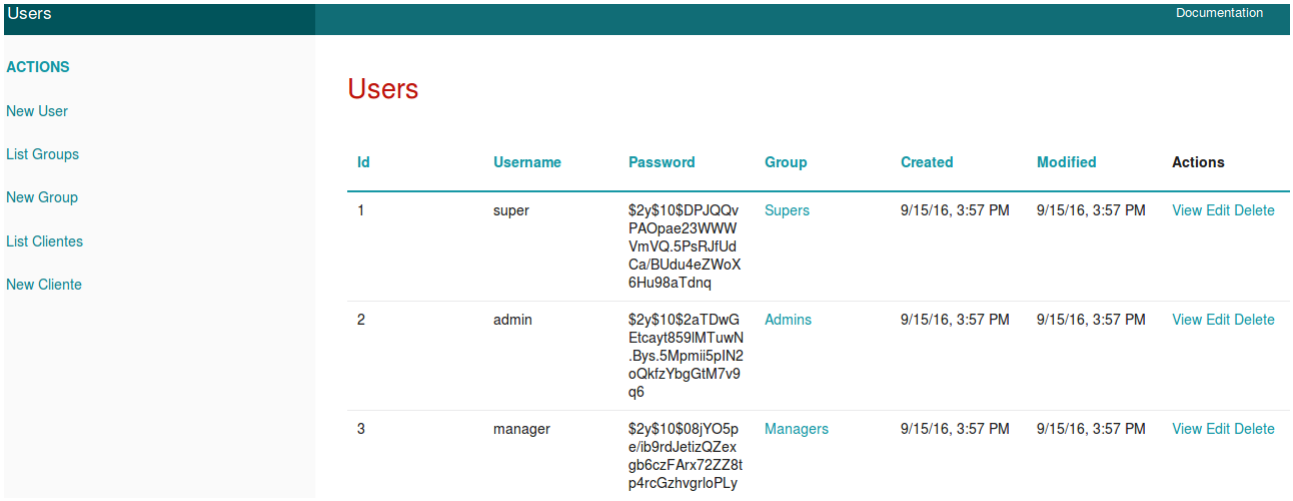
Agora pode acessar seu aplicativo pelo navegador

http://localhost/clientes

Aparece por default o Clientes/index, como configuramos no routes







Agora está pronto para melhorar o aplicativo manualmente ou usando um plugin como o cake-control-br.

Alterando o Template/View

Quando acessar pelo navegador, especificamente users, aparece o campo senha com o hash. Não é interessante que este campo apareça, então vamos removê-lo da view.

Editar então o arquivo:

cd /var/www/html/clientes
cd src/Template/Users/index.ctp

Apagar as duas linhas:

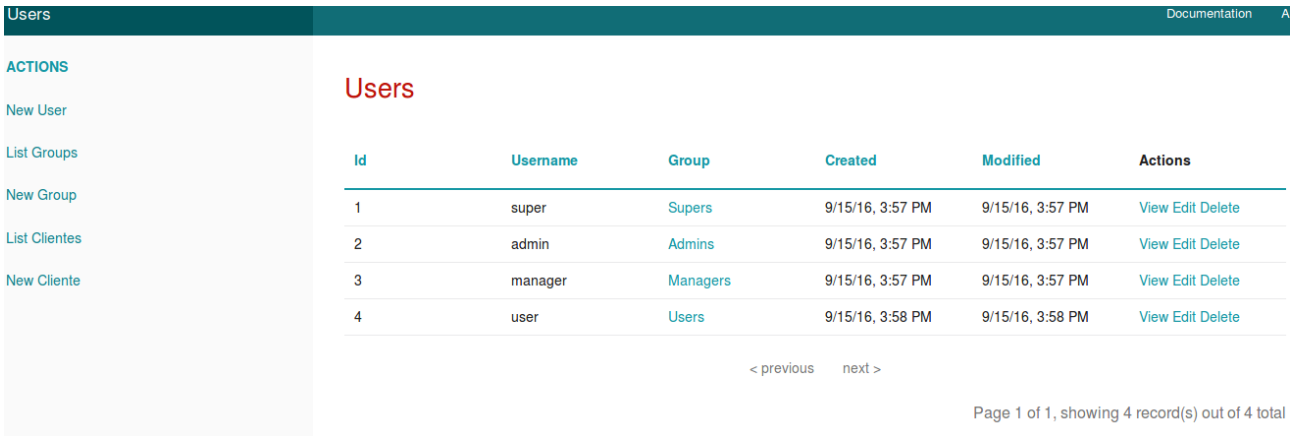
```
<th scope="col"><?= $this->Paginator->sort('password') ?></th>
```

e

```
<td><?= h($user->password) ?></td>
```

Salve e visualize novamente.

Agora está melhor, veja.



2 - Convenções no CakePHP 3

Pode ser um assunto extenso mas compensa a leitura atenta.

A equipe do CakePHP é grande admiradora de convenção sobre configuração. Seguindo convenções você recebe funcionalidades gratuitamente e libera a si mesmo do pesadelo de manter arquivos de configuração.

Aviso Importante: Portanto, antes de começar a trabalhar com CakePHP é muito importante conhecer suas convenções para tirar delas as devidas vantagens. Caso não as use o CakePHP não será de muita ajuda.

Controllers

Nomes de classes tipo Controller devem estar no plural, ser CamelCase e terminarem com o sufixo Controller.

Exemplos de classes controller:
ClientesController, PeopleController e UltimosArtigosController

Actions - Métodos public nos controllers são chamados de actions e se comunicam com views com mesmo nome que eles e extensão .ctp.

Um exemplo: index() - index.ctp
o action Controller/ClientesController/index() é mapeado automaticamente para a view src/Template/Clientes/index.ctp.

Métodos protected ou private não podem ser acessados via Routing.

Considerações sobre URL para nomes de controllers

O ClientesController que está no arquivo ClientesController.php é chamado pelo navegador com:

http://localhost/aplicacao/clientes

Nomes de controllers com palavras compostas podem ficar assim:
• /vermelhaMacas
• /VermelhaMacas
• /Vermelha_macas
• /vermelha_macas

Todos resolverão com o controller VermelhaMacas.

Criar links:

```
$this->Html->link('link-title', [
    'prefix' => 'MyPrefix' // CamelCased
    'plugin' => 'MyPlugin', // CamelCased
    'controller' => 'ControllerName', // CamelCased
    'action' => 'actionName' // camelCased
]
```

Namespaces

Todas as classes do core do CakePHP agora (3.x) usam namespaces e seguem as especificações de autoload (auto-carregamento) do **PSR-4**.

Por exemplo (troca src por Cake e troca as barras)
src/Cache/Cache.php

tem o namespace
Cake\Cache\Cache

Constantes globais e métodos de helpers como __() e debug() não usam namespaces por questões de conveniência.

Nomes de arquivos e Classes:

Classe KissesAndHugsController está no arquivo KissesAnd-HugsController.php
Classe ClientesController está no arquivo ClientesController.php.

Alguns exemplos de classes e seus arquivos:

• O Controller class KissesAndHugsController deve estar no arquivo com nome KissesAndHugsController.php

• O Component class MyHandyComponent deve estar no arquivo com nome MyHandyComponent.php

• A Table class OptionValuesTable deve estar no arquivo com nome OptionValuesTable.php

• A Entity class OptionValue deve estar no arquivo com nome OptionValue.php

• O Behavior class EspeciallyFunkableBehavior deve estar no arquivo com nome EspeciallyFunkableBehavior.php

• A View class SuperSimpleView would deve estar no arquivo com nome SuperSimpleView.php

• O Helper class BestEverHelper deve estar no arquivo com nome BestEverHelper.php

Model e Bancos de Dados

Nomes de classes Table - são no plural e CamelCase

Chave Primária
Toda tabela, obrigatoriamente deve ter uma chave primária e o nome da chave deve ser id para usurfruir das vantagens do CakePHP.

Lembrando que o cake trabalha com nomes em inglês. Ele tem um recurso online importante para mostrar o plural de nomes:
http://inflector.cakephp.org/

Nomes válidos: Clientes, Populacao, GrandePopulacao e RealmenteGrandePopulacao.

Pesquisando no site acima por Clientes e People, vemos que já estão no plural e seu singular é Cliente e Person

Nomes de tabelas são em minúsculas, no plural e palavras compostas separadas por sublinhado. Nomes de tabelas para os acima:

clientes, populacao, grande_populacao e realmente_grande_populacao

A convenção é para usar tabelas e campos com nomes na língua inglesa.

Nomes de campos em minúsculas e compostos são separados por sublinhado: first_name.

Se usarmos os **campos username e password** (com estes nomes) na tabela users, o Cake deve estar apto para auto-configurar algumas coisas para nós, quando implementando o user login.

Obs.: quando usar autenticação use o tamanho do campo password com varchar(255). Também ajuda adicionar um campo chamado role na tabela users.

Relacionamentos
Chave estrangeira nos relacionamentos hasMany, belongsTo ou hasOne são reconhecidas por default com:
nome da tabela relacionada no singular seguida de "_id".

Exemplos: groups e users. Em users o campo group_id para relacionar.
Relacionamento entre articles e users. Em articles adicionar o campo user_id.

Relacionamento Muitos para Muitos
Exemplo: Para relacionamento muitos para muitos das tabelas platillos com cocineros, criaremos uma tabela intermediária para relacionar platillos com cocineros:

```
create table cocineros_platillos(
    id int unsigned auto_increment primary key,
    cocinero_id int(11) not null,
```

```
    platillo_id int(11) not null
);
```

Tipos de Relacionamentos

one to one	hasOne	Um usuario tem um perfil.
one to many	hasMany	Um usuario pode ter múltiplos artigos.
many to one	belongsTo	Muitos artigos pertencem a um usuario
many to many	belongsToMany	Várias Tags pertencem a muitos artigos.

Para uma classe Bakers teremos uma chave estrangeira assim: baker_id.
Para uma tabela como category_types a chave estrangeira será category_type_id.

Nomes de campos especiais, que levam o Cake a tomar iniciativas importantes para nós:
title
name
created
modified

Convenções para as Views

As views tem nomes de arquivos em minúsculas com extensão .ctp.
O método getReady() do conroller PeopleController está associado ao template/view src/Template/People/get_ready.ctp.

Arquivos da Aplicação

Todos os arquivos da aplicação que criamos ficam na pasta src.

Criptografia
Por padrão o CakePHP 3.x usa a criptografia bcrypt para proteger as senhas.

Documentação oficial
http://book.cakephp.org/3.0/pt/intro/conventions.html

Convenções para as Views

O método getReady() do conroller PeopleController está associado ao template/view src/Template/People/get_ready.ctp.

Arquivos da Aplicação
Todos os arquivos da aplicação que criamos ficam na pasta src.

Criptografia
Por padrão o CakePHP 3.x usa a criptografia bcrypt para proteger as senhas.

3 – Projeto

Dicas sobre Projeto de Aplicativos

Primeiro de tudo: DRY

Sempre que usar customização em cada aplicativo procurar criar plugins para isso, contendo:

- Helpers
- Elements
- Components
- Lauouts
- AppController, AppModel, AppHelper, bootstrap, etc

Load classes using App:uses() e Vendor files using App:import()

Caso tenha vários projetos é recomendável extender a classe VENDOR

```
require_once(VENDORS.'my_model.php');
class AppModel extends MyModel {}
```

Assim também extender AppController, AppHelper, bootstrap e similares.

Para Cake 2.0 e superiores recomendo usar Libs para incluir ao invés de App::uses()

Melhor Feedback

Adicione ao AppModel para ser notificado sobre erros em associações/relacionamentos de models:

```
public function __construct($id = false, $table = null, $ds = null) {
    parent::__construct($id, $table, $ds);

    # avoiding AppModel instances instead of real Models without telling you about it
 if (!is_a($this, $this->name) && $this->displayField  !== 'id' && !
Configure::read('Core.disableModelInstanceNotice')) {
        trigger_error('AppModel instance! Expected: '.$this->name);
    }
}
```

Debugar SQL:

```
/**
 * quick sql debug from controller dynamically
 * or statically from just about any other place in the script
 * @param bool $die: TRUE to output and die, FALSE to log to file and continue
 */
function sql($die = true) {
```

```
    if (isset($this->Controller)) {
        $object = $this->Controller->{$this->Controller->modelClass};
    } else {
        $object = ClassRegistry::init(defined('CLASS_USER')?CLASS_USER:'User');
    }

    $log = $object->getDataSource()->getLog(false, false);
    foreach ($log['log'] as $key => $value) {
        if (strpos($value['query'], 'SHOW ') === 0 || strpos($value['query'], 'SELECT
CHARACTER_SET_NAME ') === 0) {
            unset($log['log'][$key]);
            continue;
        }
    }
    # output and die?
 if ($die) {
        debug($log);
        die();
    }
    # log to file then and continue
 $log = print_r($log, true);
    CakeLog::write('sql', $log);
}
```

Executando em um behavior:
CommonComponent::sql();

Sugestões

Remover os arquivos que não serão usados:
build.properties
build.xml
CONTRIBUTING.md
README.md

Remover a pasta:
lib/Cake/Test

Permissões para a pasta app/tmp:
Esta pasta deve oferecer permissão de escrita para o Apache

4 - Criação de um aplicativo tipo CRUD com Facilidade

Usando o gerador de aplicativos Bake do CakePHP.

Vamos começar usando o recurso mais prático do CakePHP, que é o gerador de CRUDs chamado bake.

Iremos criar um simples CRUD mas já é um recurso muito prático e que pode ser utilizado para a criação de aplicativos para nosso dia a dia.
Depois iremos abordar recursos mais avançados.

Observação Estou usando o Linux com diretório web em
/var/www/html

Caso esteja usando outro ambiente ou em outro diretório web altera o caminho abaixo

Também estou usando o nano no terminal do Linux ou o xed no ambiente gráfico do Linux Mint. Caso tenha alguma dúvida sobre que editor usar e esteja usando Windows surigo que use o Notepad++ (https://notepad-plus-plus.org/)

Criar o aplicativo base usando o composer na pasta crud

```
cd /var/www/html
composer create-project --prefer-dist cakephp/app crud
```

Criar o banco de dados no MySQL
Chamarei de 'crud' com apenas uma tabela chamada 'clientes' e contendo 3 campos: id, nome e e-mail

Sugestão de gerenciador de bancos de dados: o gerenciador web em PHP
http://adminer.org

Abra o gerenciador, crie o banco crud e importe o script abaixo:

```
CREATE TABLE IF NOT EXISTS `clientes` (
  `id` int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `nome` char(45) NOT NULL,
  `email` varchar(50) DEFAULT NULL
);
```

Configurar o banco no aplicativo

```
cd /var/www/html/crud
nano config/app.php
```

Como estou usando o mysql basta alterar estar linhas em **Datasource**

```
'username' => 'root',
'password' => '',
'database' => 'crud',
```

Acessar o aplicativo pelo navegador

http://localhost/crud

Ele nos mostra uma página padrão criada automaticamente pelo Cake, mas sem nada do nosso banco

Configurar o routes.php

para que nosso aplicativo abra automaticamente a index de Clientes

nano config/routes.php

Mudar apenas esta linha

cd /var/www/html/crud

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Para

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Gerar todo o código do CRUD com o bake

Execute

cd /var/www/html/crud

bin/cake bake all clientes

Observe que ele mostra mensagens de que criou todo o nosso código: controller, model, view/template entre outros

Chame pelo navegador novamente

http://localhost/crud

Ele nos mostra um aplicativo completo, com todo o CRUD, com CSS, paginação, ordenação dos campos apensa clicando em seus labels, etc.

Clique em New Criante e crie um novo registro.

Entre um nome e entre um e-mail incompleto, apenas o login e clique em SUBMIT para testar.

Ele verificou na tabela o nome do campo "email" e já aplicou validação. Assim ele faz com os campos da tabela e podemos customizar isso.
Uma beleza.

Corrija e clique em SUBMIT

Como a equipe do Cake está se preparando para lançar a versão 4.0, talvez receba a mensagem de erro:

Deprecated (16384): RequestHandlerComponent::beforeRedirect() is deprecated. This functionality will be removed in 4.0.0. Set the `enableBeforeRedirect` option to `false` to disable this warning. - /backup/www/crud/vendor/cakephp/cakephp/src/Event/EventManager.php, line: 353
You can disable deprecation warnings by setting `Error.errorLevel` to `E_ALL & ~E_USER_DEPRECATED` in your config/app.php. [CORE/src/Core/functions.php, line 307]

Caso receba edite novamente o arquivo

config/app.php

E altere a linha

'errorLevel' => E_ALL,

Para
'errorLevel' => E_ALL & ~E_USER_DEPRECATED,

E chame novamente

http://localhost/crud

Veja que temos um CRUD funcional (Add, Edit, View e Delete, além de outrosr ecursos), que podemos usar para o cadastro de clientes, uma agenda e qualquer outra finalidade.

Próximo Passo

Nosso próximo passo será criar um CRUD semelhante ao criado pelo bake mas inteiramente na unha (todo usando código). Para isso criarei este novo aplicativo em três passos: crud1, crud2 e crud3.

Sugestões de leitura:

Uma recomendação é instalar o plugin cake-control-br, que instala um template com o Bootstrap e um controle de acesso para o aplicativo:

https://github.com/ribafs/cake-control-br

O site oficial do CakePHP é uma das grandes forças do framework, com tutoriais de exemplo de aplicativos:
https://book.cakephp.org/3.0/en/tutorials-and-examples.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog/blog.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog-auth-example/auth.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/bookmarks/intro.html

E uma ótima documentação, que pode ser vista online
https://book.cakephp.org/3.0/en/index.html

Ou offline em PDF ou Epub
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.pdf
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.epub

Uma forma bem simples de se criar um bom aplicativo tipo CRUD e muito mais

5 - Gerando Código com Bake no CakePHP 3

http://book.cakephp.org/3.0/en/bake.html

A console bake do CakePHP é um outro esforço para que você esteja rodando o CakePHP rapidamente. A console do bake pode criar qualquer um dos códigos básicos do CakePHP: models, behaviors, views, helpers, controllers, components, test cases, fixtures e plugins. E não estamos falando apenas de esqueletos de classes, o bake pode criar uma aplicação inteiramente funcional em poucos minutos.

Pré-requisitos

A console do bake requer:
- PHP CLI
- extensão intl
- Pelo menos um SGBD instalado e configurado na aplicação atual

Instalação

O bake já vem instalado por padrão, mas se precisar instalar use o comando:

```
composer require --dev cakephp/bake:~1.0

ou

php composer.phar require --dev cakephp/bake:dev-master
```

Comandos disponíveis do bake. Execute:

bin\cake bake

- all
- behavior
- cell
- component
- controller
- fixture
- form
- helper
- model
- plugin
- shell
- template
- test

Usando `cake bake [name]` podemos invocar uma tarefa especificada pelo name.

Exemplo:
bin\cake bake all [name]

Ajuda
bin/cake bake controller --help

O bake é uma ferramenta espetacular. Veja que com apenas um comando geramos o aplicativo.

Com ele podemos fazer muto:

Gerar um controller:
bin/cake bake controller Clientes

Gerar controller sem actions:
bin\cake bake controller --no-actions Clientes

Gerar Componente
bin/cake bake component Calculo

Gerar um Template
bin/cake bake template Clientes

Gerar Helper
bin/cake bake helper MeuForm

Gerar Cell
bin/cake bake cell MinhaCelula

Gerar Form
bin/cake bake form MeuForm

Gerar um Model
bin/cake bake model Clientes

Gerar Behavior
bin/cake bake behavior MeuBehavior

Observe as mensagens para saber em que diretório está sendo criado.

Ajuda
bin/cake bake controller --help

Apenas controller, model e template
bin/cake bake controller clientes
bin/cake bake model clientes
bin/cake bake template clientes

Plugins

Gerando código usando plugin
bin/cake bake all --plugin CakePtbr articles
Criar plugin ContactManager com tabela contacts

cd /backup/www/cake/blog
bin/cake bake plugin ContactManager
bin/cake bake controller --plugin ContactManager Contacts
bin/cake bake model --plugin ContactManager Contacts
bin/cake bake template --plugin ContactManager Contacts

Atualizando
composer dumpautoload

Componente
bin/cake bake component Calculo

Migrate
bin/cake migrations
bin/cake migrations migrate
bin/cake migrations status --plugin PluginName

Existem alguns recursos que não são gerados ou auxiliados pelo bake, como o element.

Gerar o código completo do CRUD para uma tabela:
cd /var/www/cakeapp/app/Console
./cake bake all

Gerar o código para os actions "admin_" em todos os controllers:
./cake bake controller all --admin

cake bake controller ControllerName --plugin PluginNameInCamelCase

Gerar todas as views iniciadas com admin_:
./cake bake view all --admin

Outras formas:
./cake bake model group
./cake bake view group

./cake bake model user
./cake bake view user

./cake bake model
./cake bake controller
./cake bake view

Rodando Interativamente
./cake bake

Outras formas
./cake bake db_config
./cake bake project
./cake bake fixture
./cake bake test
./cake bake plugin plugin_name

Gerar todos os models:
./cake bake model all

Entendendo a geração de código no Cake

Quando geramos um CRUD através do bake, especialmente no que diz respeito ao model gerado, ele cria validação para os campos, mas somente para aqueles que têm alguma restrição (pelo menos NOT NULL) e também armazena no model o respectivo código para os relacionamentos (quando seguimos as convenções do Cake para relacionamentos).

Um exemplo: o model Cliente.php (cuja tabela é relacionada com a tabela pedidos).

Veja a tabela:

```
CREATE TABLE IF NOT EXISTS `clientes` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nome` char(45) NOT NULL,
  `email` varchar(50) DEFAULT NULL,
  `data_nasc` date DEFAULT NULL,
  `cpf` char(11) NOT NULL,
  PRIMARY KEY (`id`)
);
```

Observe que os campos email e data_nasc tem NULL como default, ou seja não exigem qualquer informação. Aceitam qualquer valor para ser salvo, inclusive nenhum valor.

Em pedidos existe um relacionamento com clientes, assim como com funcionários e produtos:

```
CREATE TABLE IF NOT EXISTS `pedidos` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cliente_id` int(11) NOT NULL,
  `funcionario_id` int(11) NOT NULL,
  `produto_id` int(11) NOT NULL,
  `data` date NOT NULL,
  `quantidade` int(11) NOT NULL,
  `preco` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `cliente_id` (`cliente_id`),
  KEY `funcionario_id` (`funcionario_id`),
  KEY `produto_id` (`produto_id`)
```

```
);
```

Agora veja o código gerado pelo Bake para o model Cliente:

```
class Cliente extends AppModel {

	/**
	 * Validation rules
	 *
	 * @var array
	 */
		public $validate = array(
			'nome' => array(
				'notempty' => array(
					'rule' => array('notempty'),
					//'message' => 'Your custom message here',
					//'allowEmpty' => false,
					//'required' => false,
					//'last' => false, // Stop validation after this rule
					//'on' => 'create', // Limit validation to 'create' or 'update' operations
				),
			),
			'cpf' => array(
				'notempty' => array(
					'rule' => array('notempty'),
					//'message' => 'Your custom message here',
					//'allowEmpty' => false,
					//'required' => false,
					//'last' => false, // Stop validation after this rule
					//'on' => 'create', // Limit validation to 'create' or 'update' operations
				),
			),
		);

	/**
	 * hasMany associations
	 *
	 * @var array
	 */
		public $hasMany = array(
			'Pedido' => array(
				'className' => 'Pedido',
				'foreignKey' => 'cliente_id',
				'dependent' => false,
				'conditions' => '',
				'fields' => '',
				'order' => '',
				'limit' => '',
				'offset' => '',
				'exclusive' => '',
				'finderQuery' => '',
```

```
                'counterQuery' => ''
            )
        );
}
```

Veja, que somente tem validação os campos nome e cpf. O código da associação/relacionamento entre clientes e pedidos (hasMany - um cliente tem muitos pedidos).

Já o relacionamento entre pedidos e clientes (veja em Pedido.php) é do tipo belogsTo (pedido pertence a cliente). Veja:

```
    public $belongsTo = array(
        'Cliente' => array(
            'className' => 'Cliente',
            'foreignKey' => 'cliente_id',
            'conditions' => '',
            'fields' => '',
            'order' => ''
        ),
```

Em termos de segurança dos dados no banco, o relacionamento é muito importante. Num banco de dados como o banco acima seria lamentável ter pedidos cadastrados no banco de clientes que já foram excluídos, por exemplo. O relacionamento é muito importante e deve ser implementado inclusive no banco e não somente no Cake.

Geração de todo o código: controllers, models e templates e testes
bin/cake bake all clientes
bin/cake bake all users

Apenas controller, model e template
bin/cake bake controller clientes
bin/cake bake model clientes
bin/cake bake template clientes

Plugins

Criar plugin ContactManager com tabela contacts

cd /backup/www/cake/blog
bin/cake bake plugin ContactManager
bin/cake bake controller --plugin ContactManager Contacts
bin/cake bake model --plugin ContactManager Contacts
bin/cake bake template --plugin ContactManager Contacts

Atualizando
composer dumpautoload

Componente
bin\cake bake component Calculo

Migrate
bin/cake migrations
bin/cake migrations migrate
bin/cake migrations status --plugin PluginName

Criar plugin usando o bake:
O banco precisa ter uma tabela chamada Contacts

cd /backup/www/cake/blog
bin/cake bake plugin ContactManager
bin/cake bake controller --plugin ContactManager Contacts
bin/cake bake model --plugin ContactManager Contacts
bin/cake bake template --plugin ContactManager Contacts

Gerando código para Área Administrativa

Criar aplicativo para cadastro de clientes com área administrativa acessada somente por alguns usuários.

Criar o controller para o Admin:
bin\cake bake controller --no-actions --prefix Admin Home

Criar o controller para o Site
bin\cake bake controller --no-actions --prefix Site Home

Criar um CRUD para o Admin:

Configure o banco de dados no config/app.php

Criar código para gerar classe
bin\cake bake migration CreateUsers nome:string email:string created modified

Criar tabela com código acima
bin\cake migrations migrate

Gerar o Scaffold para Usuários
bin\cake bake all Users --prefix Admin

Criar outros usos para o Bake:

Configurar o banco de dados

Configurar a carga de plugin

Criar:
- model
- controller

```
bin/cake bake controller comments --force
```

- controller sem actions
- action
- view/template
- helper
- element
- component
- criar validações interaticamente
- mudar/criar displayField
- Criar admin

Usar migrações

Criar migrações

bin\cake migrations create CreateUsersTable
bin\cake migrations create CreateGroupsTable

Criar tabelas
bin\cake migrations migrate

Usar Composer

Instalar CakePHP
composer create-project –prefer-dist cakephp/app nopeapp

chmod 550 bin\cake

Adicionando outros actions para controllers no Bake:

Yes, you can bake:

```
cake bake view users custom_func
```

For that you need to create a template in the following directory:

```
YOUR_PROJECT\lib\Cake\Console\Templates\default\views
```

```
"create a custom_func.ctp file"
```

and also create a function in (core controller file):

```
YOUR_PROJECT\lib\Cake\Console\Templates\default\actions
```

```
"add a new function custom_func"
```

Or if you don't want that function to be universal, directly write into the users controller.

Garantir que não use cache:

You will also need to regenerate your model classes and clear out the cache:

bin/cake bake model comments –force
bin/cake bake model issues –force
bin/cake orm_cache clear

Cache do ORM

Reconstruir o cache dos metadados de todas as tabelas da conexão default
bin/cake orm_cache build --connection default

Para reconstruir o cache dos emtadados apenas da tabela artigos
bin/cake orm_cache build --connection default artigos

Remover o cache de todos os metadados:
bin/cake orm_cache clear

Remover o cache de todos os metadados da tabela aratigos
bin/cake orm_cache clear artigos

Plugins e Outros com o Bake

bin\cake bake plugin Admin

Com isso ele cria a pasta plugins\Admin contendo config, src, webroot e composer.json.

Em src ele cria a pasta Controller e dentro o arquivo AppController.php com uma classe AppController vazia.

Carregando Plugin no bootstrap.php com o bake

bin\cake plugin load Admin

Agora podemos continuar e criar controllers, models, etc, como fazemos com aplicativos do CakePHP, assim:

Criando um controller

bin\cake bake controller --plugin Admin Groups
bin\cake bake controller --plugin Admin Users
bin\cake bake controller --plugin Admin Privileges

Alerta: cuidado ao copiar e colar código de processadores de texto. O exemplo acima acaba de me gerar um problema, pois os dois hifens antes de plugin foram para a console como apenas um traço e deu erro.

Criando Componente para o Plugin

bin\cake bake component --plugin Admin AccessControl

Com o comando acima ele cria um controller completo, com todos os actions default. Verifique.

Componentes, Helpers e Behaviors

Um plugin pode conter componentes, helper e behaviors criados da mesma forma que para uma aplicação normal.

Criando Componente com bake
Antes criar o Controler
bin\cake bake component AccessControl

Criando Helper com bake
Antes criar o Template
bin\cake bake helper MeuHelper

Criando Behavior com o bake
Antes criar o Model
bin\cake bake behavior MeuBehavior

Criar o arquivo de migração

```
bin\cake bake migration CreateContacts title:string body:text created modified
```

Com isso ele cria um arquivo em config/Migrations contendo uma classe e uma função com a definição da tabela a ser criada sugerida por nós.

Para criar a tabela execute:

```
bin\cake migrations migrate
```

Pode verificar agora a tabela contacts criada em nosso banco atual.

Criando Model e Template para o Plugin

bin\cake bake model --plugin Admin Groups
bin\cake bake model --plugin Admin Users
bin\cake bake model --plugin Admin Privileges

Criando Templates para o Plugin

bin\cake bake template --plugin Admin Groups
bin\cake bake template --plugin Admin Users
bin\cake bake template --plugin Admin Privileges

Criar somente a view login.ctp para o Plugin

bin\cake bake template --plugin Admin Users login

Agora nosso plugin tem uma estrutura básica completa, inclusive com bancos e podemos inserir registros para ver como se comporta.

Chamando nosso plugin pelo navegador

http://localhost/aplicativo/admin/groups

Ele irá mostrar a index.ctp do controller Contacts do nosso plugin.

É como se nosso plugin Admin fosse outro aplicativo rodando sobre o aplicativo atual e estamos chamando seu controller contacts.

Procurei esclarecer e melhorar um pouco o tutorial oficial do site e está aí, podendo servir para a criação de um plugin bem útil no CakePHP 3.

Também contei com a ajuda do plugin migrations para criar a tabela no tutorial de criação de Blog no site do Cake.

Podemos usar Helpers e Elements em nosso template do Bake.

Os templates do bake usam as tags do Asp <%, <%= e %>

Exemplo de template de um Helper no bake:

```
<?php
namespace &lt;%= $namespace %>\View\Helper;
use Cake\View\Helper;
use Cake\View\View;
/**
 * <%= $name %> helper
 */
class <%= $name %>Helper extends Helper {
/**
 * Default configuration.
 *
 * @var array
 */
  protected $_defaultConfig = [];
}
?>
```

Quando o código acima é interpretada pelo Cake fica assim:

```
<?php
namespace App\View\Helper;
```

```
use Cake\View\Helper;
use Cake\View\View;
/**
 * Post helper
 */
class PostHelper extends Helper {
/**
 * Default configuration.
 *
 * @var array
 */
  protected $_defaultConfig = [];
}
?>
```

Se quizermos customizar o emplate do bake apenas para um certo controler, como o Servidores, para que o bake gere para o mesmo somente os actions: index(), view() e add(), alteramos o código do config/bootstrap_cli.php para algo assim:

```
<?php
use Cake\Event\Event;
use Cake\Event\EventManager;
use Cake\Utility\Hash;
EventManager::instance()->attach(function (Event $event) {
    $view = $event->subject;
    $name = Hash::get($view->viewVars, 'name');
    $isController = strpos($event->data[0], 'Bake/Controller/controller.ctp') !== false;
    if ($isController !== false && $name == 'Servidores') {
        $view->viewVars['actions'] = ['index', 'view', 'add'];
    }
    if ($isController && $name == 'Comentarios') {
        $view->viewVars['actions'] = ['add'];
    }
}, 'Bake.beforeRender');
?>
```

Este código deve:

- Anexar ao evento Bake.beforeRender, que permite modificar qualquer dado indo para o template/view

- Receber o nome do template

- Caso o nome do arquivo termine com Bake/Controller/controller.ctp, o template usado sendo o Servidores, gerará apenas os referidos actions.

- Também adicionamos somente o action add() caso o controller seja Comentários.

Mais detalhes no artigo do José Gonzales em:
http://josediazgonzalez.com/2014/12/03/customizing-bake-and-installing-plugins-with-cakephp-3/

6 – Configurações

Configuração
Embora as convenções eliminem a necessidade de configurar todo o CakePHP, Você ainda precisará configurar algumas coisas, como suas credenciais de banco de dados por exemplo.

Além disso, há opções de configuração opcionais que permitem trocar valores padrão e implementações com as personalizadas para seu aplicativo.

Configurando sua Aplicação

A configuração é geralmente armazenada em arquivos PHP ou INI, e carregada durante a execução do código de inicialização. O CakePHP vem com um arquivo de configuração por padrão. Mas se necessário, você pode adicionar arquivos de configuração adicionais e carregá-los no código de inicialização do aplicativo. `Cake\Core\Configure` é usado para configuração global, e classes como `Cache` providenciam `config()` métodos para tornar a configuração simples e transparente.

Carregando Arquivos de Configurações Adicionais
Se sua aplicação tiver muitas opções de configuração, pode ser útil dividir a configuração em vários arquivos. Depois de criar cada um dos arquivos no seu **config/** diretório, você pode carregá-los em **bootstrap.php**:

```
use Cake\Core\Configure;
use Cake\Core\Configure\Engine\PhpConfig;

Configure::config('default', new PhpConfig());
Configure::load('app', 'default', false);
Configure::load('other_config', 'default');
```

Você também pode usar arquivos de configuração adicionais para fornecer sobreposições específicas do ambiente. Cada arquivo carregado após **app.php** pode redefinir valores previamente declarados permitindo que você personalize a configuração para ambientes de desenvolvimento ou de homologação.

Configuração Geral
Abaixo está uma descrição das variáveis e como elas afetam seu aplicativo CakePHP.

debug
: Altera a saída de depuração do CakePHP. `false` = Modo Produção. Não é exibido nenhuma mensagem de erro e/ou aviso. `true` = Modo de Desenvolvimento. É exibido todas as mensagens de erros e/ou avisos.

App.namespace
: O namespace em que as classes do aplicativo estão.

Ao alterar o namespace em sua configuração, você também precisará atualizar o arquivo ** composer.json ** para usar esse namespace também. Além disso, crie um novo carregador automático executando `php composer.phar dumpautoload`.

App.baseUrl
: Não comentar esta definição se você **não** planeja usar o mod_rewrite do Apache com o CakePHP. Não se esqueça de remover seus arquivos .htaccess também.

App.base
: O diretório base no qual o aplicativo reside. Se `false` isso será detectado automaticamente. Se não `false`, certifique-se de que sua seqüência de caracteres começa com um / e NÃO termina com um /. Por exemplo, */basedir* deve ser uma App.base válida. Caso contrário, o AuthComponent não funcionará corretamente.

App.encoding
: Defina a codificação que seu aplicativo usa. Essa codificação é usada para gerar o charset no layout e codificar entidades. Ele deve corresponder aos valores de codificação especificados para o seu banco de dados.

App.webroot
: O diretório raiz da aplicação web.

App.wwwRoot
: O diretório raiz dos arquivos da aplicação web.

App.fullBaseUrl
: O nome de domínio totalmente qualificado (incluindo o protocolo) para a raiz do aplicativo. Isso é usado ao gerar URLs absolutos. Por padrão, esse valor é gerado usando a variável $_SERVER. Entretanto, Você deve defini-lo manualmente para otimizar o desempenho ou se você está preocupado com as pessoas manipulando o cabeçalho do `Host`. Em um contexto CLI (do Shell) a *fullBaseUrl* não pode ser lido a partir de $_SERVER, como não há servidor envolvido. Você precisará especificá-lo se precisar gerar URLs de um shell (por exemplo, ao enviar e-mails).

App.imageBaseUrl
: O caminho da web para as imagens públicas na webroot da aplicação. Se você estiver usando um CDN, você deve definir este valor para a localização do CDN.

App.cssBaseUrl
: O caminho da web para os arquivos de estilos em cascata(**.css**) públicos na webroot da aplicação. Se você estiver usando um CDN, você deve definir este valor para a localização do CDN.

App.jsBaseUrl
: O caminho da web para os scripts (em JavaScript) públicos na webroot da aplicação. Se você estiver usando um CDN, você deve definir este valor para a localização do CDN.

App.paths
: Configurar caminhos para recursos não baseados em classe. Suporta as subchaves `plugins`, `templates`, `locales`, que permitem a definição de caminhos para plugins, templates e arquivos de locale respectivamente.

Security.salt
: Uma seqüência aleatória usada em hash. Uma seqüência aleatória usada em hash. Este valor também é usado como o sal HMAC ao fazer criptografia simétrica.

Asset.timestamp

Acrescenta um carimbo de data/hora que é a última hora modificada do arquivo específico no final dos URLs de arquivos de recurso (CSS, JavaScript, Image) ao usar assistentes adequados. Valores válidos:

- (bool) `false` - Não fazer nada (padrão)
- (bool) `true` - Acrescenta o carimbo de data/hora quando depuração é `true`
- (string) 'force' - Sempre anexa o carimbo de data/hora.

Configuração do banco de dados
Consulte Database Configuration para obter informações sobre como configurar suas conexões de banco de dados.

Configuração do Cache
Consulte Caching Configuration para obter informações sobre como configurar o cache no CakePHP.

Configuração de manipulação de erro e exceção
Consulte Error and Exception Configuration para obter informações sobre como configurar manipuladores de erro e exceção.

Configuração de log
Consulte Logging Configuration para obter informações sobre como configurar o log no CakePHP.

Configuração de e-mail
Consulte Email Configuration para obter informações sobre como configurar predefinições de e-mail no CakePHP.

Configuração de sessão
Consulte Session Configuration para obter informações sobre como configurar o tratamento de sessão no CakePHP.

Configuração de roteamento
Consulte Routes Configuration para obter mais informações sobre como configurar o roteamento e criar rotas para seu aplicativo.

Caminhos adicionais de classe
Caminhos de classe adicionais são configurados através dos carregadores automáticos usados pelo aplicativo. Ao usar o `Composer` para gerar o seu arquivo de autoload, você pode fazer o seguinte, para fornecer caminhos alternativos para controladores em seu aplicativo:

```
"autoload": {
    "psr-4": {
        "App\\Controller\\": "/path/to/directory/with/controller/folders",
        "App\": "src"
    }
}
```

O código acima seria configurar caminhos para o namespace `App` e `App\Controller`. A primeira chave será pesquisada e, se esse caminho não contiver a classe/arquivo, a

segunda chave será pesquisada. Você também pode mapear um namespace único para vários diretórios com o seguinte código:

```
"autoload": {
    "psr-4": {
        "App\": ["src", "/path/to/directory"]
    }
}
```

Plugin, Modelos de Visualização e Caminhos Locais
Como os plug-ins, os modelos de visualização (Templates) e os caminhos locais (locales) não são classes, eles não podem ter um autoloader configurado. O CakePHP fornece três variáveis de configuração para configurar caminhos adicionais para esses recursos. No **config/app.php** você pode definir estas variáveis

```
return [
    // More configuration
    'App' => [
        'paths' => [
            'plugins' => [
                ROOT . DS . 'plugins' . DS,
                '/path/to/other/plugins/'
            ],
            'templates' => [
                APP . 'Template' . DS,
                APP . 'Template2' . DS
            ],
            'locales' => [
                APP . 'Locale' . DS
            ]
        ]
    ]
];
```

Caminhos devem terminar com um separador de diretório, ou eles não funcionarão corretamente.

Configuração de Inflexão
Consulte Configuração da inflexão para obter mais informações sobre como fazer a configuração de inflexão.

Configurar classe
class `Cake\Core\Configure`

A classe de Configuração do CakePHP pode ser usada para armazenar e recuperar valores específicos do aplicativo ou do tempo de execução. Tenha cuidado, pois essa classe permite que você armazene qualquer coisa nela, para que em seguida, usá-la em qualquer outra parte do seu código: Dando ma certa tentação de quebrar o padrão MVC do CakePHP. O objetivo principal da classe Configurar é manter variáveis centralizadas que podem ser compartilhadas entre muitos objetos. Lembre-se de tentar viver por “convenção sobre a configuração” e você não vai acabar quebrando a estrutura MVC previamente definida.

Você pode acessar o `Configure` de qualquer lugar de seu aplicativo:

```
Configure::read('debug');
```

Escrevendo dados de configuração
static `Cake\Core\Configure::write`(*$key*, *$value*)

Use `write()` para armazenar dados na configuração do aplicativo:

```
Configure::write('Company.name', 'Pizza, Inc.');
Configure::write('Company.slogan','Pizza for your body and soul');
```

O dot notation usado no parâmetro `$key` pode ser usado para organizar suas configurações em grupos lógicos.

O exemplo acima também pode ser escrito em uma única chamada:

```
Configure::write('Company', [
    'name' => 'Pizza, Inc.',
    'slogan' => 'Pizza for your body and soul'
]);
```

Você pode usar `Configure::write('debug', $bool)` para alternar entre os modos de depuração e produção na mosca. Isso é especialmente útil para interações JSON onde informações de depuração podem causar problemas de análise.

Leitura de dados de configuração
static `Cake\Core\Configure::read`(*$key* = *null*)

Usado para ler dados de configuração da aplicação. Por padrão o valor de depuração do CakePHP é importante. Se for fornecida uma chave, os dados são retornados. Usando nossos exemplos de write() acima, podemos ler os dados de volta:

```
Configure::read('Company.name');    // Yields: 'Pizza, Inc.'
Configure::read('Company.slogan');  // Yields: 'Pizza for your body
                                    // and soul'

Configure::read('Company');

//Rendimentos:
['name' => 'Pizza, Inc.', 'slogan' => 'Pizza for your body and soul'];
```

Se $key for deixada nula, todos os valores em Configure serão retornados.

static `Cake\Core\Configure::readOrFail`(*$key*)

Lê dados de configuração como `Cake\Core\Configure::read`, mas espera encontrar um par chave/valor. Caso o par solicitado não exista, a `RuntimeException` será lançada:

```
Configure::readOrFail('Company.name');    // Rendimentos: 'Pizza, Inc.'
Configure::readOrFail('Company.geolocation');  // Vai lançar uma exceção

Configure::readOrFail('Company');
```

```
// Rendimentos:
['name' => 'Pizza, Inc.', 'slogan' => 'Pizza for your body and soul'];
```

Novo na versão 3.1.7: `Configure::readOrFail()` Foi adicionado na versão 3.1.7

Verificar se os dados de configuração estão definidos
static `Cake\Core\Configure::check`(*$key*)

Usado para verificar se uma chave/caminho existe e tem valor não nulo:

```
$exists = Configure::check('Company.name');
```

Excluindo Dados de Configuração
static `Cake\Core\Configure::delete`(*$key*)

Usado para excluir informações da configuração da aplicação:

```
Configure::delete('Company.name');
```

Leitura e exclusão de dados de configuração
static `Cake\Core\Configure::consume`(*$key*)

Ler e excluir uma chave do Configure. Isso é útil quando você deseja combinar leitura e exclusão de valores em uma única operação.

Lendo e escreveendo arquivos de configuração
static `Cake\Core\Configure::config`(*$name*, *$engine*)

O CakePHP vem com dois mecanismos de arquivos de configuração embutidos. `Cake\Core\Configure\Engine\PhpConfig` é capaz de ler arquivos de configuração do PHP, no mesmo formato que o Configure tem lido historicamente. `Cake\Core\Configure\Engine\IniConfig` é capaz de ler os arquivos de configuração no formato ini(.ini). Consulte a documentação do PHP para obter mais informações sobre os detalhes dos arquivos ini. Para usar um mecanismo de configuração do núcleo, você precisará conectá-lo ao Configure usando `Configure::config()`:

```
use Cake\Core\Configure\Engine\PhpConfig;

// Ler os arquivos de configuração da configuração
Configure::config('default', new PhpConfig());

// Ler arquivos de configuração de outro diretório.
Configure::config('default', new PhpConfig('/path/to/your/config/files/'));
```

Você pode ter vários mecanismos anexados para Configure, cada um lendo diferentes tipos ou fontes de arquivos de configuração. Você pode interagir com os motores conectados usando alguns outros métodos em Configure. Para verificar quais aliases de motor estão conectados você pode usar `Configure::configured()`:

```
// Obter a matriz de aliases para os motores conectados.
Configure::configured();

// Verificar se um motor específico está ligado.
Configure::configured('default');
```

static Cake\Core\Configure::drop(*$name*)

Você também pode remover os motores conectados. `Configure::drop('default')` removeria o alias de mecanismo padrão. Quaisquer tentativas futuras de carregar arquivos de configuração com esse mecanismo falhariam:

```
Configure::drop('default');
```

Carregando arquivos de configurações
static Cake\Core\Configure::load(*$key*, *$config* = *'default'*, *$merge* = *true*)

Depois de ter anexado um motor de configuração para o Configure, ficará disponível para poder carregar ficheiros de configuração:

```
// Load my_file.php using the 'default' engine object.
Configure::load('my_file', 'default');
```

Os arquivos de configuração que foram carregados mesclam seus dados com a configuração de tempo de execução existente no Configure. Isso permite que você sobrescreva e adicione novos valores à configuração de tempo de execução existente. Ao definir `$merge` para `true`, os valores nunca substituirão a configuração existente.

Criando ou modificando arquivos de configuração
static Cake\Core\Configure::dump(*$key*, *$config* = *'default'*, *$keys* = *[]*)

Despeja todos ou alguns dos dados que estão no Configure em um sistema de arquivos ou armazenamento suportado por um motor de configuração. O formato de serialização é decidido pelo mecanismo de configuração anexado como $config. Por exemplo, se o mecanismo 'padrão' é `Cake\Core\Configure\Engine\PhpConfig`, o arquivo gerado será um arquivo de configuração PHP carregável pelo `Cake\Core\Configure\Engine\PhpConfig`

Dado que o motor 'default' é uma instância do PhpConfig. Salve todos os dados em Configure no arquivo *my_config.php*:

```
Configure::dump('my_config', 'default');
```

Salvar somente a configuração de manipulação de erro:

```
Configure::dump('error', 'default', ['Error', 'Exception']);
```

`Configure::dump()` pode ser usado para modificar ou substituir arquivos de configuração que são legíveis com `Configure::load()`

Armazenando Configuração do Tempo de Execução

static `Cake\Core\Configure::store`(*$name*, *$cacheConfig* = *'default'*, *$data* = *null*)

Você também pode armazenar valores de configuração de tempo de execução para uso em uma solicitação futura. Como o configure só lembra valores para a solicitação atual, você precisará armazenar qualquer informação de configuração modificada se você quiser usá-la em solicitações futuras:

```
// Armazena a configuração atual na chave 'user_1234' no cache 'default'.
Configure::store('user_1234', 'default');
```

Os dados de configuração armazenados são mantidos na configuração de cache nomeada. Consulte a documentação Caching para obter mais informações sobre o cache.

Restaurando a Configuração do Tempo de Execução

static `Cake\Core\Configure::restore`(*$name*, *$cacheConfig* = *'default'*)

Depois de ter armazenado a configuração de tempo de execução, você provavelmente precisará restaurá-la para que você possa acessá-la novamente. `Configure::restore()` faz exatamente isso:

```
// Restaura a configuração do tempo de execução do cache.
Configure::restore('user_1234', 'default');
```

Ao restaurar informações de configuração, é importante restaurá-lo com a mesma chave e configuração de cache usada para armazená-lo. As informações restauradas são mescladas em cima da configuração de tempo de execução existente.

Criando seus próprios mecanismos de configuração

Como os mecanismos de configuração são uma parte extensível do CakePHP, você pode criar mecanismos de configuração em seu aplicativo e plugins. Os motores de configuração precisam de uma `Cake\Core\Configure\ConfigEngineInterface`. Esta interface define um método de leitura, como o único método necessário. Se você gosta de arquivos XML, você pode criar um motor de XML de configuração simples para sua aplicação:

```
// Em src/Configure/Engine/XmlConfig.php
namespace App\Configure\Engine;

use Cake\Core\Configure\ConfigEngineInterface;
use Cake\Utility\Xml;

class XmlConfig implements ConfigEngineInterface
{

    public function __construct($path = null)
    {
        if (!$path) {
            $path = CONFIG;
        }
        $this->_path = $path;
    }
```

```
    public function read($key)
    {
        $xml = Xml::build($this->_path . $key . '.xml');
        return Xml::toArray($xml);
    }

    public function dump($key, array $data)
    {
        // Code to dump data to file
    }
}
```

No seu **config/bootstrap.php** você poderia anexar este mecanismo e usá-lo:

```
use App\Configure\Engine\XmlConfig;

Configure::config('xml', new XmlConfig());
...

Configure::load('my_xml', 'xml');
```

O método `read()` de um mecanismo de configuração, deve retornar uma matriz das informações de configuração que o recurso chamado `$key` contém.

interface `Cake\Core\Configure\ConfigEngineInterface`

Define a interface usada pelas classes que leem dados de configuração e armazenam-no em `Configure`

`Cake\Core\Configure\ConfigEngineInterface::read`(*$key*)

Parâmetros:
- **$key** (*string*) – O nome da chave ou identificador a carregar.

Esse método deve carregar/analisar os dados de configuração identificados pelo `$key` e retornar uma matriz de dados no arquivo.

`Cake\Core\Configure\ConfigEngineInterface::dump`(*$key*)

Parâmetros:
- **$key** (*string*) – O identificador para escrever.
- **$data** (*array*) – Os dados para despejo.

Esse método deve despejar/armazenar os dados de configuração fornecidos para uma chave identificada pelo `$key`.

Motores de Configuração Integrados

Arquivos de configuração do PHP

class `Cake\Core\Configure\Engine\PhpConfig`

Permite ler arquivos de configuração que são armazenados como arquivos simples do PHP. Você pode ler arquivos da configuração do aplicativo ou do plugin configs diretórios usando sintaxe plugin. Arquivos *devem* retornar uma matriz. Um exemplo de arquivo de configuração seria semelhante a:

```
return [
    'debug' => 0,
    'Security' => [
        'salt' => 'its-secret'
    ],
    'App' => [
        'namespace' => 'App'
    ]
];
```

Carregue seu arquivo de configuração personalizado inserindo o seguinte em **config/bootstrap.php**:

```
Configure::load('customConfig');
```

Arquivos de configuração Ini

class `Cake\Core\Configure\Engine\IniConfig`

Permite ler arquivos de configuração armazenados como arquivos .ini simples. Os arquivos ini devem ser compatíveis com a função `parse_ini_file()` do php e beneficiar das seguintes melhorias.

- Os valores separados por ponto são expandidos em arrays.
- Valores booleanos como ‘on’ e ‘off’ são convertidos em booleanos.

Um exemplo de arquivo ini seria semelhante a:

```
debug = 0

[Security]
salt = its-secret

[App]
namespace = App
```

O arquivo ini acima, resultaria nos mesmos dados de configuração final do exemplo PHP acima. As estruturas de matriz podem ser criadas através de valores separados por pontos ou por seções. As seções podem conter chaves separadas por pontos para um assentamento mais profundo.

Arquivos de configuração do Json

class `Cake\Core\Configure\Engine\JsonConfig`

Permite ler/descarregar arquivos de configuração armazenados como cadeias codificadas JSON em arquivos .json.

Um exemplo de arquivo JSON seria semelhante a:

```
{
    "debug": false,
    "App": {
        "namespace": "MyApp"
    },
```

```
    "Security": {
        "salt": "its-secret"
    }
}
```

Bootstrapping CakePHP
Se você tiver alguma necessidade de configuração adicional, adicione-a ao arquivo **config/bootstrap.php** do seu aplicativo. Este arquivo é incluído antes de cada solicitação, e o comando CLI.

Este arquivo é ideal para várias tarefas de bootstrapping comuns:

- Definir funções de conveniência.
- Declaração de constantes.
- Definição da configuração do cache.
- Definição da configuração de log.
- Carregando inflexões personalizadas.
- Carregando arquivos de configuração.

Pode ser tentador para colocar as funções de formatação lá, a fim de usá-los em seus controladores. Como você verá nas seções Controllers (Controladores) e Views (Visualização) há melhores maneiras de adicionar lógica personalizada à sua aplicação.

Application::bootstrap()
Além do arquivo **config/bootstrap.php** que deve ser usado para configurar preocupações de baixo nível do seu aplicativo, você também pode usar o método `Application::bootstrap()` para carregar/inicializar plugins, E anexar ouvintes de eventos globais:

```
// Em src/Application.php
namespace App;

use Cake\Core\Plugin;
use Cake\Http\BaseApplication;

class Application extends BaseApplication
{
    public function bootstrap()
    {
        // Chamar o pai para `require_once` config/bootstrap.php
        parent::bootstrap();

        Plugin::load('MyPlugin', ['bootstrap' => true, 'routes' => true]);
    }
}
```

Carregar plugins/eventos em `Application::bootstrap()` torna Controller Integration Testing mais fácil à medida que os eventos e rotas serão reprocessados em cada método de teste.

Variáveis de Ambiente
Alguns dos provedores modernos de nuvem, como o Heroku, permitem definir variáveis de ambiente. Ao definir variáveis de ambiente, você pode configurar seu aplicativo CakePHP como um aplicativo 12factor. Seguir as instruções do aplicativo 12factor app instructions é uma boa maneira de criar um app sem estado e facilitar a implantação do seu aplicativo. Isso significa, por exemplo, que, se você precisar alterar seu banco de dados, você precisará modificar uma variável DATABASE_URL na sua configuração de host sem a necessidade de alterá-la em seu código-fonte.

Como você pode ver no seu **app.php**, as seguintes variáveis estão em uso:

- `DEBUG` (`0` ou``1``)
- `APP_ENCODING` (ie UTF-8)
- `APP_DEFAULT_LOCALE` (ie `en_US`)
- `SECURITY_SALT`
- `CACHE_DEFAULT_URL` (ie `File:///?prefix=myapp_&serialize=true&timeout=3600&path=../tmp/cache/`)
- `CACHE_CAKECORE_URL` (ie `File:///?prefix=myapp_cake_core_&serialize=true&timeout=3600&path=../tmp/cache/persistent/`)
- `CACHE_CAKEMODEL_URL` (ie `File:///?prefix=myapp_cake_model_&serialize=true&timeout=3600&path=../tmp/cache/models/`)
- `EMAIL_TRANSPORT_DEFAULT_URL` (ie `smtp://user:password@hostname:port?tls=null&client=null&timeout=30`)
- `DATABASE_URL` (ie `mysql://user:pass@db/my_app`)
- `DATABASE_TEST_URL` (ie `mysql://user:pass@db/test_my_app`)
- `LOG_DEBUG_URL` (ie `file:///?levels[]=notice&levels[]=info&levels[]=debug&file=debug&path=../logs/`)
- `LOG_ERROR_URL` (ie `file:///?levels[]=warning&levels[]=error&levels[]=critical&levels[]=alert&levels[]=emergency&file=error&path=../logs/`)

Como você pode ver nos exemplos, definimos algumas opções de configuração como DSN. Este é o caso de bancos de dados, logs, transporte de e-mail e configurações de cache.

Se as variáveis de ambiente não estiverem definidas no seu ambiente, o CakePHP usará os valores definidos no **app.php**. Você pode usar a biblioteca php-dotenv library para usar variáveis de ambiente em um desenvolvimento local. Consulte as instruções Leiame da biblioteca para obter mais informações.

Desabilitando tabelas genéricas
Embora a utilização de classes de tabela genéricas - também chamadas auto-tables - quando a criação rápida de novos aplicativos e modelos de cozimento é útil, a classe de tabela genérica pode tornar a depuração mais difícil em alguns cenários.

Você pode verificar se qualquer consulta foi emitida de uma classe de tabela genérica via DebugKit através do painel SQL no DebugKit. Se você ainda tiver problemas para diagnosticar um problema que pode ser causado por tabelas automáticas, você pode lançar uma exceção quando o CakePHP implicitamente usa um `Cake\ORM\Table` genérico em vez de sua classe concreta assim:

```
// No seu bootstrap.php
use Cake\Event\EventManager;
// Prior to 3.6 use Cake\Network\Exception\NotFoundException
use Cake\Http\Exception\InternalErrorException;

$isCakeBakeShellRunning = (PHP_SAPI === 'cli' && isset($argv[1]) && $argv[1] ===
'bake');
if (!$isCakeBakeShellRunning) {
    EventManager::instance()->on('Model.initialize', function($event) {
        $subject = $event->getSubject();
        if (get_class($subject === 'Cake\ORM\Table') {
            $msg = sprintf(
                'Missing table class or incorrect alias when registering table
class for database table %s.',
                $subject->getTable());
            throw new InternalErrorException($msg);
        }
    });
}
```

bootstrap.php

Veja as mensagens iniciais que ele pode disparar, acusando que este é um dos arquivos iniciais no boot do CakePHP 3.

Observe que ele já vem com várias configurações prontas. Já vem com alguns plugins carregados, como o Migrations e o DebugKit.

Carregando Plugins

Caso queira carregar um plugin de terceiro precisa adicionar uma linha logo abaixo desta: Plugin::load('Migrations');

Supondo que precise carregar um plugin chamado Acl:

Plugin::load('Acl');

Evite usar loadAll()

Dicas sobre Bancos de dados

Tabelas

Se um campo não for importante não adicionemos na tabela
Mas se for importante precisa ser requerido

Conexão do aplicativo Cake com PostgreSQL

config/app.php

Altere a seção Datasource de acordo com seu banco
Caso use um esquema diferente do public, use como abaixo

```
...
        'driver' => 'Cake\Database\Driver\Postgres',
...
        //'port' => 'non_standard_port_number',
        'username' => 'postgres',
        'password' => 'postgres',
        'database' => 'dn_modelo',
        'schema' => 'basico',
..
```

Relacionamentos

O campo secundário do relacionamento precisa ser NOT NULL na tabela.

Exemplo
users com clientes

Em clientes temos o campo user_id que relaciona as tabelas.
Na tabela clientes o campo user_id precisa ser:

user_id int not null

Caso contrário teremos sérios problemas de cadastramento de todos os registros vazios e fura o relacionamento.

Relacionamentos no Banco e no Cake

O CakePHP tem uma convenção para relacionamento entre tabelas, como visto acima, de forma que nem precisamos implementar o relacionamento pelo SGBD, pois o Cake cuida disso pra nós.

Acontece que no nosso exemplo de script, onde não implementamos o relacionamento pelo banco, temos 100 registros de servidores, todos relacionados com users e ainda não temos nenhum user cadastrado.
Isto não é coerente. Só deveriamos adicionar um servidor, quando antes tivéssemos cadastrado o user respectivo.

Recomendação: sempre implemente os relacionamentos no banco, que fica mais coerente e mais seguro.

Para implementar relacionamento entre servidores e users na tabela servidores faça assim (mas isso no início, antes de criar servidores):

```
CREATE TABLE servidores (
      id SERIAL PRIMARY KEY,
      nome varchar(55) NOT NULL,
      nascimento date default null, -- Requerido default null em campos data peço cake
      cpf char(11),
      fone varchar(14),
      user_id integer not null,
    created timestamp(0) without time zone DEFAULT NULL,
    modified timestamp(0) without time zone DEFAULT NULL,
      observacao text,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE RESTRICT
);
```

Esta sintaxe funciona no PostgreSQL e também no MySQL.

Dicas de CakePHP/Postgresql

Cuidado ao usar campos de senha no PostgreSQL

No MySQL um campo tipo
password char(255) reservará na tabela somente os dígitos digitados, mas no PostgreSQL serão reservados todos os 255.

Uma senha no MySQL assim:
$2y$10$GJrox7HEUNcOLFOEUzwCLu3Omd2HQEzNY7RgFCNGyS0UnEyufrDY6Final(só o resultado)

No PostgreSQL
$2y$10$GJrox7HEUNcOLFOEUzwCLu3Omd2HQEzNY7RgFCNGyS0UnEyufrDY6
Final(255)

Portanto para campos de senha não use char() no PostgreSQL, mas somente varchar().

Configuração do PostgreSQL

config/app.php

```
...
'Datasources' => [
       'default' => [
           'className' => 'Cake\Database\Connection',
           'driver' => 'Cake\Database\Driver\Postgres',
           'persistent' => false,
```

```
        'host' => 'localhost',
        /**
         * CakePHP will use the default DB port based on the driver selected
         * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
         * the following line and set the port accordingly
         */
        'port' => '5432',
        'username' => 'dnocs',
        'password' => 'dnocs.devel',
        'database' => 'intranet',
       'schema'   => 'testes',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'flags' => [],
        'cacheMetadata' => true,
        'log' => false,
…
```

Perigo do Valor Default

Na criação das tabelas não deixe valor defaul em campos.
Caso deixe ele não será criticado, não será exigida uma entrada do usuário.
Ao teclar enter sem ter digitado nada o valor default será armazenado.

Um bom exemplo é o campo user_id no Clientes/add.ctp que por padrão é criado assim pelo Bake

```
        echo $this->Form->input('user_id', ['options' => $users]);
```

Desta forma o primeiro valor é assumido sempre que o usuário não seleciona algo no select.

Idealmente mude para algo assim:

```
        echo $this->Form->input('user_id', ['empty' => true,'options' => $users]);
```

Assim, ao teclar enter, sem ter selecionado nada será exigido que selecione algo.

```
SELECT ordinal_position,
column_name,
data_type,
column_default,
is_nullable,
character_maximum_length,
numeric_precision
FROM information_schema.columns
WHERE table_name = 'clientes' and column_name='nome'
ORDER BY ordinal_position;
```

RECEBER PRIMARY KEY

```
   SELECT kcu.column_name AS pk
     FROM information_schema.table_constraints tc
LEFT JOIN information_schema.key_column_usage kcu
       ON tc.constraint_name = kcu.constraint_name
    WHERE tc.table_name = 'clientes'
    AND constraint_type = 'PRIMARY KEY';
```

RECEBER MAIS INFOS

```
   SELECT tc.constraint_name,
          tc.constraint_type,
          tc.table_name,
          kcu.column_name,
         tc.is_deferrable,
          tc.initially_deferred,
          rc.match_option AS match_type,
          rc.update_rule AS on_update,
          rc.delete_rule AS on_delete,
          ccu.table_name AS references_table,
          ccu.column_name AS references_field
     FROM information_schema.table_constraints tc
LEFT JOIN information_schema.key_column_usage kcu
       ON tc.constraint_catalog = kcu.constraint_catalog
      AND tc.constraint_schema = kcu.constraint_schema
      AND tc.constraint_name = kcu.constraint_name
LEFT JOIN information_schema.referential_constraints rc
       ON tc.constraint_catalog = rc.constraint_catalog
      AND tc.constraint_schema = rc.constraint_schema
      AND tc.constraint_name = rc.constraint_name
LEFT JOIN information_schema.constraint_column_usage ccu
       ON rc.unique_constraint_catalog = ccu.constraint_catalog
      AND rc.unique_constraint_schema = ccu.constraint_schema
      AND rc.unique_constraint_name = ccu.constraint_name
    WHERE tc.table_name = 'clientes';
```

```
select column_name from information_schema.constraint_column_usage where
table_name = '$table' and constraint_name = '{$table}_pkey';
```

MOSTRAR FOREIGN KEY

```
SELECT
    kcu.column_name
FROM
    information_schema.table_constraints AS tc
    JOIN information_schema.key_column_usage AS kcu ON tc.constraint_name =
kcu.constraint_name
    JOIN information_schema.constraint_column_usage AS ccu ON ccu.constraint_name =
tc.constraint_name
```

```
WHERE constraint_type = 'FOREIGN KEY' AND tc.table_name='pedidos';
```

Dicas sobre o SGBD livre PostgreSQL

```
CREATE DATABASE nomebanco;
```

CREATE TABLE

```
CREATE TABLE produtos(
    id SERIAL NOT NULL,
    descricao VARCHAR(40) default '' NOT NULL,
    fornecedor_id INTEGER,
    categoria_id INTEGER,
    quantidade INTEGER,
    preco DECIMAL(12,2),
    estoque INTEGER,
    unidades_pedido INTEGER,
    nivel_pedido INTEGER,
    descontinuado INTEGER default 'f' NOT NULL,
    observacao TEXT,
    data DATE,
    PRIMARY KEY (id)
)Type=MyISAM;

INSERT INTO produtos
(id,descricao,fornecedor_id,categoria_id,quantidade,preco,estoque,unidades_pedido,nivel
_pedido,descontinuado,observacao,data) VALUES (1,'Chai Tea',1,1,'10 boxes x 20
bags',18,39,-7,10,' ','2001-12-01');
```

Scrips

MySQL ou PostgreSQL (este script funciona nos dois)

```
create table clientes
(
    cliente int primary key,
    cpf char(11),
    nome char(45) not null,
    credito_liberado char(1) not null,
    data_nasc date,
    email varchar(50)
);

create table funcionarios
(
    funcionario int primary key,
    cpf char(11),
    nome char(45) not null,
    senha char(32) not null,
    email varchar(50),
```

```
    data_nasc date not null
);

create table pedidos
(
    pedido int primary key,
    cliente_id int not null,
    funcionario_id int not null,
    data_pedido date not null,
    data_confirmacao date not null,
  FOREIGN KEY (cliente_id) REFERENCES clientes(cliente) ON DELETE RESTRICT,
  FOREIGN KEY (funcionario_id) REFERENCES funcionarios(funcionario) ON DELETE
RESTRICT
);
```

Outra forma: constraint pedidos_cli_fk foreign key(cliente_id) REFERENCES clientes (cliente)
ALTER DATABASE

```
ALTER TABLE pedidos ADD CONSTRAINT pedidos_cli_fk FOREIGN KEY (cliente_id)
REFERENCES clientes (cliente);

ALTER TABLE clientes ADD CHECK (length(cpf)=11);

ALTER TABLE estoque ALTER COLUMN data SET DEFAULT CURRENT_DATE;
```

Exemplo rico em recursos:
Devel/SGBD/PostgreSQL/Exercicios/Modulo4Projetos/pessoa.sql

Dicas Avançadas:

```
SELECT CASE WHEN 10*2=30 THEN '30 incorrect'
  WHEN 10*2=40 THEN '40 incorrect'
  ELSE 'Deve ser 10*2=20'
END;

SELECT CASE 10*2
  WHEN 20 THEN '20 correct'
  WHEN 30 THEN '30 incorrect'
  WHEN 40 THEN '40 incorrect'
END;

SELECT CASE WHEN 9>7
            THEN "TRUE"
            ELSE
            "FALSE"
            END
 AS "Resultado";
```

Trazer somente 1 registro posterior ao valor fornecido
SELECT id, title FROM topics WHERE id > $currentTopicId ORDER BY id ASC LIMIT 1

SELECT cliente, nome FROM clientes WHERE cliente > 1 ORDER BY cliente ASC LIMIT 1

Trazer somente 1 registro anterior ao valor fornecido
SELECT id FROM entrada_dados WHERE id < 10 ORDER BY id DESC LIMIT 1
SELECT data,total FROM entrada_dados WHERE data < '2009-07-01' ORDER BY total DESC LIMIT 1

Com outros parâmetros no WHERE
SELECT id, title FROM topics WHERE id > $currentTopicId and visible=1 ORDER BY id ASC LIMIT 1

Último registro
SELECT id FROM tab_pecas WHERE id = (SELECT MAX id from tab_pecas)

SELECT nm_dept FROM departamento ORDER BY codigo DESC LIMIT 1

SELECT entidades FROM entrada_dados ORDER BY id DESC LIMIT 1

select * from employee1 order by empid,name,dob asc limit 1;

ALTER TABLE recursos_humanos.funcionais2 add primary key(id)

SELECT LAST_INSERT_ID()

Datas

SELECT CURRENT_DATE();

7 – Segurança

Dicas sobre o Componente Auth

Identificação, autenticação e autorização de usuários é uma parte comum de quase todas as aplicações web. No CakePHP o AuthComponent fornece uma maneira modular de fazer essas tarefas. O AuthComponent permite combinar objetos de autenticação, autorização e objetos para criar formas flexíveis de identificação e verificação de autorização do usuário.

Este pequeno tutorial é um resumo do oficial (em inglês), uma versão resumida, pois quem fará o principal trabalho de controle de acesso será o componente Acesso.

Alerta: a tabela users não deve ter nenhum usuário cadastrado no início, deve estar vazia. Os usuários devem ser cadastrados após a implementação do Auth e através do controller users.

Caso receba a mensagem:
Erro: Table useres for model User was not found in datasource default.
Isso aconteceu comigo sempre que instalei o o plugin Cakept_br.

Adicione a variável $useTable ao model User
public $useTable = 'users';

Adaptado do tutorial oficial:
http://book.cakephp.org/2.0/en/tutorials-and-examples/blog-auth-example/auth.html

Ajustar AppController.php para isso:

```
class AppController extends Controller {
    public $components = array(
        'Session',
        'Auth' => array(
                'loginAction' => array('controller'=>'users','action'=>'login'),
            'loginRedirect' => array('controller' => 'clientes', 'action' => 'index'),
            'logoutRedirect' => array('controller' => 'users', 'action' => 'login'),
                'authorize' => array('Controller') // Added this line
        )
    );

    public function beforeFilter() {
        $this->Auth->allow('menus','index'); // Estes terão acesso público
    }

      public function isAuthorized($user) {
            if (isset($user['group_id']){
                return true;
            }
            return false;
```

```
    }
}
```

Adicionar ao controller Users:

```
    public function login() {
        if ($this->request->is('post')) {
            if ($this->Auth->login()) {
                $this->redirect($this->Auth->redirect());
            } else {
                $this->Session->setFlash(__('Invalid username or password, try again'));
            }
        }
    }

    public function logout() {
        $this->redirect($this->Auth->logout());
    }
```

Criar a view Users/login.ctp:

```
<div class="users form">
<?php echo $this->Session->flash('auth'); ?>
<?php echo $this->Form->create('User'); ?>
   <fieldset>
      <legend><?php echo __('Please enter your username and password'); ?></legend>
      <?php echo $this->Form->input('username');
      echo $this->Form->input('password');
   ?>
   </fieldset>
<?php echo $this->Form->end(__('Login')); ?>
</div>
```

Início do model User.php:

```
App::uses('AppModel', 'Model');
App::uses('AuthComponent', 'Controller/Component');

class User extends AppModel {

    public function beforeSave($options = array()) {
        if (isset($this->data[$this->alias]['password'])) {
            $this->data[$this->alias]['password'] = AuthComponent::password($this-
>data[$this->alias]['password']);
        }
        return true;
    }
```

Pronto. Estas são as alterações necessárias.

Testando
Agora tente adicionar um cliente, funcionario, produto ou pedido ou editar.
Verá que será redirecionado para o login com a mensagem de que não tem autorização.

Adicionar Usuário
Agora precisamos permitir, por enquanto, que o público use a view "add" em AppController para adicionar um usuário admin.

$this->Auth->allow('menus','index','add');

Acesse então
http://localhost/cakemodelo2/users/

E adicione um usuário "admin" ao grupo "admins".

Após adicionar desfaça então as alterações no AppController:
$this->Auth->allow('menus'); // Deixando para o acesso público somente 'menus'.

Faça o login como user "admin" em:
http://localhost/cakemodelo2/users/login

Agora terá direito a qualquer operação como admin.
Lembre que para fazer logout pode chamar:
http://localhost/cakemodelo2/users/logout

Prontinho, temos nosso aplicativo com controle de acesso implementado via componente Auth.

Copie o arquivo BcryptFormAuthenticate.php para:

cakebcrypt/lib/Cake/Controller/Component/Auth

O Bcrypt aumenta a segurança dos aplicativos e provavelmente virá implementado por padrão nas próximas versões do Cake.

Implementação da autenticação com Bcrypt.

As alterações para que funcionasse com bcrypt foram em:

app/Model/User

No início:
App::uses('BcryptFormAuthenticate', 'Controller/Component/Auth');

No código da classe:
Alterar o beforeSave para este

```
    function beforeSave($options = Array()) {
        if (isset($this->data['User']['password'])) {
```

```
        $this->data['User']['password'] = BcryptFormAuthenticate::hash($this->data['User']
['password']);
      }
      return true;
  }
```

No app/Controller/AppController:

Alteração no $components ...

```
    public $components = array(
        'Session',
        'Auth' => array(
     'authenticate' => array(
            'Blowfish' => array(
                'fields' => array('username' => 'username'))
            ),// https://groups.google.com/forum/#!msg/cake-
php/s8RXjtg3IWE/LjdTOVWH7q0J
            'loginRedirect' => array('controller' => '/clientes', 'action' => 'index'),
            'logoutRedirect' => array('controller' => 'users', 'action' => 'login'),
            'authorize' => array('Controller') // Adicionamos essa linha
        )
    );
```

Agora veja a diferença entre os hashs:
Usando o hash padrão do Auth - 99790b128030b417877948352809548a5d3c55f5
Usando o hash do Bcrypt - $2a$10$DYhG93b0qyJfIxfs2guVoOWUva7s4SWbWRR/RtNzvuZOVDU/jwdP.

Por isso outra alteração obrigatória deve ser aumentar o tamanho do campo password para 60.
Outra exigência do bcrypt: o salt alfanumérico no core.php deve ser sempre maior que 20 dígitos.

Testando

Permita ao público add e adicione o usuário “admin”.

Adicione o usuário admin no grupo admins:
http://localhost/cakebcrypt/users

Altere para não permitir add pelo público.

Acesse
http://localhost/cakebcrypt/logout

Depois
http://localhost/cakebcrypt/users/login
E teste o aplicativo.

Adicionando hash bCrypt para a Senha
Isso é feito no Model, especificamente na classe Entity:

```
// src/Model/Entity/User.php
namespace App\Model\Entity;

use Cake\Auth\DefaultPasswordHasher;
use Cake\ORM\Entity;

class User extends Entity
{

    // Make all fields mass assignable except for primary key field "id".
    protected $_accessible = [
        '*' => true,
        'id' => false
    ];

    // ...

    protected function _setPassword($password)
    {
        return (new DefaultPasswordHasher)->hash($password);
    }

    // ...
}
```

Exemplos de Autenticação

```
    public function initialize() {
        parent::initialize();

        $this->loadComponent('RequestHandler');
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authorize' => [
                'Acl.Actions' => ['actionPath' => 'controllers/']
            ],
            'loginAction' => [
                'plugin' => false,
                'controller' => 'Users',
                'action' => 'login'
            ],
            'loginRedirect' => [
                'plugin' => false,
                'controller' => 'Users',
                'action' => 'index'
            ],
            'logoutRedirect' => [
```

```
            'plugin' => false,
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => [
            'controller' => 'Pages',
            'action' => 'display',
            'prefix' => false
        ],
        'authError' => 'You are not authorized to access that location.',
        'flash' => [
            'element' => 'error'
        ]
    ]);

    // Only for ACL setup
    $this->Auth->allow();
}
```

```
$this->loadComponent('Auth', [
    'authorize' => [
        'Acl.Actions' => ['actionPath' => 'controllers/']
    ],
    'loginAction' => [
        'plugin' => false,
        'controller' => 'Users',
        'action' => 'login'
    ],
    'loginRedirect' => [
        'plugin' => false,
        'controller' => 'Posts',
        'action' => 'index'
    ],
    'logoutRedirect' => [
        'plugin' => false,
        'controller' => 'Users',
        'action' => 'login'
    ],
    'unauthorizedRedirect' => [
        'controller' => 'Users',
        'action' => 'login',
        'prefix' => false
    ],
    'authError' => 'You are not authorized to access that location.',
    'flash' => [
        'element' => 'error'
    ]
]);
```

Component Security no CakePHP 3

class `SecurityComponent`(*ComponentCollection $collection*, *array $config* =[])

O Componente security cria uma maneira fácil de integrar uma segurança mais reforçada em suas aplicações. Ele oferece métodos para várias tarefas como:

- Restringir que métodos HTTP sua aplicação aceita
- Proteção contra adulteração dos formulários
- Requerer que SSL seja usado
- Limitando a comunicação cruzada para controllers

Pode ser configurado através do método beforeFilter.

Usando o Componente Security você automaticamente recebe proteção contra adulteração de formulários. Token oculto deve ser automaticamente inserido nos forms e checados pelo componente Security.

Se você está usando a proteção do Componente Security
Se você estiver usando recursos de proteção de forms do componente Security e outros componentes que processam dados de formulários em seus callback startup(), não se esqueça de colocar componentes Security antes desses componentes em seu método initialize().

Quando usar o componente Security você deve usar o FormHelper para criar seus formulários. Além disso, você não deve sobrescrever qualquer um dos atributos "nome" dos campos. O componente Security procura por certos indicadores que são criados e gerenciados pelo FormHelper (especialmente aqueles created em View\Helper\FormHelper::create() e View\Helper\FormHelper::end()). Dinamicamente alterar os campos que são submetidos em um pedido POST (por exemplo, desabling, deleting ou creating novos campos via JavaScript) é susceptível de causar o request para ser enviado para a callback blackhole. Veja a $validatePost ou o parâmetro de configuração $disabledFields.

Você deve sempre verificar o método HTTP a ser utilizado antes de executar efeitos colaterais. Você deve verificar o método HTTP ou usar Cake\Network\Request::allowMethod() para garantir que o método HTTP correto foi usado.

Usando o Componente Security

O uso do componente Security geralmente é feito no método beforeFilter() dos Controllers. Você deve especificar as restrições de segurança que você deseja e o componente Security deve reforçar então em seu startup():

```
namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\Event;

class WidgetsController extends AppController
```

```
{

    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Security');
    }

    public function beforeFilter(Event $event)
    {
        if (isset($this->request->params['admin'])) {
            $this->Security->requireSecure();
        }
    }
}
```

O exemplo acima deve forçar todos os actions que tem rota admin para exigir seguras request SSL:

```
namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\Event;

class WidgetsController extends AppController
{

    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Security', ['blackHoleCallback' => 'forceSSL']);
    }

    public function beforeFilter(Event $event)
    {
        if (isset($this->params['admin'])) {
            $this->Security->requireSecure();
        }
    }

    public function forceSSL()
    {
        return $this->redirect('https://' . env('SERVER_NAME') . $this->request-
>here);
    }
}
```

Este exemplo poderia forçar todas as ações que tiverem admin routing para exigir solicitações SSL seguras. Quando o pedido é preto furado/black holed, ele irá chamar o callback denominado forceSSL(), que irá redirecionar solicitações não seguras para proteger os pedidos automaticamente.

Proteção CSRF

CSRF ou Cross Site Request Forgery é uma vulnerabilidade comum em aplicações web. Ela permite que um atacante capture e reproduza um pedido anterior, e às vezes enviar solicitações de dados usando tags ou recursos de imagem em outros domínios. Para habilitar os recursos de proteção CSRF usar o *Request Forgery Cross Site*:
http://book.cakephp.org/3.0/en/controllers/components/csrf.html

Desabilitando o Componente Security para Actions Específicos

Pode haver casos em que você deseja desativar todas as verificações de segurança para um action (ex. AJAX requests). Você pode "unlock" essas ações, listando-os em $this->Security->unlockedActions em seu beforeFilter(). A propriedade unlockedActions não afetará outras características do SecurityComponent:

```
namespace App\Controller;

use App\Controller\AppController;
use Cake\Event\Event;

class WidgetController extends AppController
{

    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Security');
    }

    public function beforeFilter(Event $event)
    {
         $this->Security->config('unlockedActions', ['edit']);
    }
}
```

Este exemplo deve desabilitar todos os cheques de segurança para o action edit.

Mais
http://book.cakephp.org/3.0/en/controllers/components/security.html
http://book.cakephp.org/3.0/en/core-libraries/security.html
http://book.cakephp.org/3.0/en/controllers/components/csrf.html

Dicas sobre Segurança

Sanitization

$badString = 'HEY<script>...</script>';
Outputs: HEY<script>...</script>

echo Sanitize::html($badString);

Outputs: HEY...
echo Sanitize::html($badString, array('remove' => true));
Sanitize class has been deprecated and will be removed in 3.0.

- Implementação do algoritmo de criptografia BCrypt

O Bcrypt aumenta a segurança dos aplicativos e provavelmente virá implementado no componente Auth por padrão nas próximas versões do Cake.

app/Model/User

No início:
App::uses('BcryptFormAuthenticate', 'Controller/Component/Auth');

Copiar o código da classe BcryptFormAuthenticate.php, do raiz deste aplicativo para: cakemodelo/lib/Cake/Controller/Component/Auth

No código da classe do model User, alterar o beforeSave para este

```
    function beforeSave($options = Array()) {
        if (isset($this->data['User']['password'])) {
            $this->data['User']['password'] = BcryptFormAuthenticate::hash($this->data['User']
['password']);
        }
        return true;
    }
```

No app/Controller/AppController:

Alteração no $components ...

```
      public $components = array(
            'Session', 'Acesso',
            'Auth' => array(
        'authenticate' => array(
                'Blowfish' => array(
                     'fields' => array('username' => 'username'))
                ),// https://groups.google.com/forum/#!msg/cake-
php/s8RXjtg3IWE/LjdTOVWH7q0J
                'loginRedirect' => array('controller' => '/submenus', 'action' => 'menus'),
                'logoutRedirect' => array('controller' => 'users', 'action' => 'login'),
                'authorize' => array('Controller') // Adicionamos essa linha
            )
      );
```

Agora veja a diferença entre os hashs:
Usando o hash padrão do Auth - 99790b128030b417877948352809548a5d3c55f5
Usando o hash do Bcrypt - $2a$10$DYhG93b0qyJfIxfs2guVoOWUva7s4SWbWRR/RtNzvuZOVDU/jwdP.

Por isso outra alteração obrigatória deve ser aumentar o tamanho do campo password para 60.

Outra exigência do bcrypt: o salt alfanumérico no core.php deve ser sempre maior que 20 dígitos.

Testando

Permita ao público index e edit
Comente o componente em AppController assim

```
    public function beforeFilter() {
        $this->Auth->allow('menus','index','edit'); // Estes terão acesso público

      /*    if($this->action != 'menus'){

                $controller=$this->params['controller'];
                $action=$this->params['action'];
                $this->Acesso->access($controller,$action);

                if($this->Acesso->redir==true){
                    $this->redirect(array('controller' => 'users','action' => 'login'));
                }
            } */
    }
```

http://localhost/cakemodelo2/users
E altere a senha do usuário "admin" para admin e a dos outros também e salve.

Altere novamente para não permitir add pelo público.

Acesse
http://localhost/cakemodelo2/logout

Depois
http://localhost/cakemodelo2/users/login

E teste o aplicativo com os usuários admin, gerente e usuario.

- Plugin Localized validando campos CPF

Adicionando Validação com o Plugin Localized
O plugin Localized contém classes com validação para diversos países, inclusive o Brasil.
Validação Específica do Brasil:
postal, phone, ssn (cpf e cnpj)

Download do plugin:
https://github.com/cakephp/localized
Descompactar e renomear para Localized

Carregar em app/Config/bootstrap.php:

CakePlugin::load('Localized');

Adicionar ao model Cliente:
<?php
App::uses('BrValidation', 'Localized.Validation');

Logo mais abaixo no corpo da classe altere a validação do cpf para ficar assim:

```
    public $validate = array(
        'nome' => array(
            'notempty' => array(
                'rule' => array('notempty'),
                //'message' => 'Your custom message here',
                //'allowEmpty' => false,
                //'required' => false,
                //'last' => false, // Stop validation after this rule
                //'on' => 'create', // Limit validation to 'create' or 'update' operations
            ),
        ),
        'cpf' => array(
                'valid' => array(
                    'rule' => array('ssn', null, 'br'),
                    'message' => 'CPF Inválido!'
                )
            )
    );
```

Pronto. Ao adicionar ou editar um registro com CPF ele será devidamente validado.

- Instalando o componente Acesso (Alternativas: ACL, Usermgmt e plugin Admin)

Esclarecimento: o componente Acesso controla o acesso de usuários ao aplicativo. Ele controla para cada action de cada controller. Mas para isso nós precisamos preencher a tabela privileges através do controller de mesmo nome. Somente após cadastrar cada action de cada controller e seu respectivo usuário, somente então ele funcionará corretamente.

Começe chamando o controller privileges:
http://localhost/cakemodelo2/privileges/

E cadastrando todos os actions que deseja controlar. Deixe de fora somente os que deseja deixar com acesso ao público. Aqui estou deixando somente o menus, login e logout com acesso ao público.

Copie o código em anexo do componente Acesso para app/Controller/Component

Adicione a entrada do componente Acesso ao AppController:

```
    public $components = array(
        'Session', 'Acesso',
            ...
```

```
    }
```

Adicione o beforeFilter ao AppController:

```
    public function beforeFilter() {
        $this->Auth->allow('menus'); // Liberado para o público

        if($this->action != 'menus'){

            $controller=$this->params['controller'];
            $action=$this->params['action'];
            $this->Acesso->access($controller,$action);

            if($this->Acesso->redir==true){
                $this->redirect(array('controller' => 'users','action' => 'login'));
            }
        }
    }
```

Neste caso, estou configurando para que ele dê acesso ao público somente no action "menus".

Altere o método isAuthorized em AppController para ficar assim:

```
    public function isAuthorized($user) {
        // Admin can access every action
        if (isset($user['group_id'])){// && $user['group_id'] == 1) {
            return true;
        }

        // Default deny
        return false;
    }
```

Testando:
http://localhost/cakemodelo2/

Faça login como admin e cadastre os usuários: gerente no grupo gerentes e usuario no usuarios.
Então faça logout.
Faça login como gerente ou usuario e experimente testar os privilégios cadastrados em privileges.

- Instalação e uso do plugin Security

Para efetuar testes no nosso código e dar sugestões para melhorar a segurança.

- Download - https://github.com/nodesagency/cakephp-security
- Descompacte e renomeie para Security
- Copie a pasta Security para /app/Plugin

- Adicione ao app/Config/bootstrap.php:
    CakePlugin::load('Security');
- Executar no terminal
cd /var/www/cadcli/app/Console
./cake Security.check controller ../Controller

Este comando não gerou nenhum comentário/relatório
./cake Security.check view ../View

Este gerou um grande relatório:
2013-06-18 18:27:10 Info: app/View/Clientes/busca.ctp (2 errors)
2013-06-18 18:27:10 Warning: Line 20 - Unsafe variable output for "$cliente". Please wrap in h()
2013-06-18 18:27:10 Warning: Line 22 - Unsafe variable output for "$cliente". Please wrap in h()
2013-06-18 18:27:10 Info: app/View/Pages/home.ctp (3 errors)
2013-06-18 18:27:10 Warning: Line 70 - Unsafe variable output for "$settings". Please wrap in h()
2013-06-18 18:27:10 Warning: Line 85 - Unsafe variable output for "$filePresent". Please wrap in h()
E vários outros warnings. Vale apena

Veja que ele sugere que inclua "h()" em todos os echos.
Após adotar h() as mensagens desaparecem. Exemplo em busca.ctp:
<?php echo h($cliente['Cliente']['id']); ?>

Password Confirm Validate

'passwordequal' => array('rule' => 'checkpasswords' , 'message' => 'Passwords Do Not Match')

```
public function checkpasswords(){

   if(strcmp($this->data['User']['new_password'],$this->data['User']['confirm_password']) == 0 )
   {
      return true;
   }
   return false;
}
```

http://questiontrack.com/password-confirm-validation-cakephp-1046104.html

Outra

two fields in view file

```
echo $this->Form->input('password');
echo $this->Form->input('repass');
```

Model file

```
<?php
class Post extends AppModel {
    public $validate = array(
        'repass' => array(
            'equaltofield' => array(
            'rule' => array('equaltofield','password'),
            'message' => 'Require the same value to password.',
            //'allowEmpty' => false,
            //'required' => false,
            //'last' => false, // Stop validation after this rule
            'on' => 'create', // Limit validation to 'create' or 'update' operations
            )
        )
    );

function equaltofield($check,$otherfield)
    {
        //get name of field
        $fname = '';
        foreach ($check as $key => $value){
            $fname = $key;
            break;
        }
        return $this->data[$this->name][$otherfield] === $this->data[$this->name][$fname];
    }
}?>
```

http://stackoverflow.com/questions/17185246/password-confirm-validation-cakephp

Outra:

```
public $validate = array(
    'password' => array(
        'confirm' => array(
            'rule' => array('password', 'password_control', 'confirm'),
            'message' => 'Repeat password',
            'last' => true
        ),
        'length' => array(
            'rule' => array('password', 'password_control', 'length'),
            'message' => 'At least 6 characters'
        )
    ),
    'password_control' => array(
        'notempty' => array(
            'rule' => array('notEmpty'),
            'allowEmpty' => false,
            'message' => 'Repeat password'
        )
```

```
    )
);

public function password($data, $controlField, $test) {
    if (!isset($this->data[$this->alias][$controlField])) {
        trigger_error('Password control field not set.');
        return false;
    }

    $field = key($data);
    $password = current($data);
    $controlPassword = $this->data[$this->alias][$controlField];

    switch ($test) {
        case 'confirm' :
            if ($password !== Security::hash($controlPassword, null, true)) {
                $this->invalidate($controlField, 'Repeat password');
                return false;
            }
            return true;

        case 'length' :
            return strlen($controlPassword) >= 6;

        default :
            trigger_error("Unknown password test '$test'.");
    }
}
```

http://stackoverflow.com/questions/3760663/cakephp-password-validation

8 – Debug e Erros

https://book.cakephp.org/3.0/en/development/debugging.html

Debug/Depuração

Depuração é uma etapa inevitável e importante de qualquer ciclo de desenvolvimento. Ainda que o CakePHP não forneça nenhuma ferramenta que se conecte com qualquer IDE ou editor de texto, este oferece várias ferramentas que auxiliam na depuração e exibição de tudo que está sendo executado “por baixo dos panos” na sua aplicação.

Depuração Básica

debug(mixed $var, boolean $showHtml = null, $showFrom = true)

A função debug() é uma função de escopo global que funciona de maneira similar a função PHP print_r(). A função debug() exibe os conteúdos de uma variável de diversas maneiras. Primeiramente, se você deseja exibir os dados no formato HTML, defina o segundo parâmetro como true. A função também exibe a linha e o arquivo de onde a mesma foi chamada.

A saída da função somente é exibida caso a variável $debug do core esteja definida com o valor true.

debug($query) Mostra o SQL e os parâmetros incluídos, não mostra resultados.
debug($query->all()) Mostra a propriedade ResultSet retornado pelo ORM.
debug($query->toArray()) Um caminho mais fácil para mostrar todos os resultados.
debug(json_encode($query, JSON_PRETTY_PRINT)) Exemplo em JSON.
debug($query->first()) Primeiro resultado obtido na query.
debug((string)$query->first()) Mostra as propriedades de uma única entidade em JSON.

Tente isto na camada Controller: debug($this->{EntidadeNome}->find()->all());

Tratando erros

Criar um arquivo

src/Templates/Error/pdo_error.ctp

```
<?php
/**
 *
 *
 * CakePHP(tm) : Rapid Development Framework (http://cakephp.org)
 * Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
 *
 * Licensed under The MIT License
 * For full copyright and license information, please see the LICENSE.txt
 * Redistributions of files must retain the above copyright notice.
```

```
 *
 * @copyright     Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
 * @link          http://cakephp.org CakePHP(tm) Project
 * @since         0.10.0
 * @license       http://www.opensource.org/licenses/mit-license.php MIT License
 */
use Cake\Utility\Debugger;
?>
<h2>Database Error</h2>
<p class="error">
    <strong>Error: </strong>
    <?= $message; ?>
</p>
<?php if (!empty($error->queryString)) : ?>
    <p class="notice">
        <strong>SQL Query: </strong>
        <?= h($error->queryString); ?>
    </p>
<?php endif; ?>
<?php if (!empty($error->params)) : ?>
        <strong>SQL Query Params: </strong>
        <?= Debugger::dump($error->params); ?>
<?php endif; ?>
<p class="notice">
    <strong>Notice: </strong>
    <?= sprintf('If you want to customize this error message, create %s', APP_DIR . DS .
'Template' . DS . 'Error' . DS . 'pdo_error.ctp'); ?>
</p>
<?= $this->element('exception_stack_trace'); ?>
```

ou

```
<?php
/**
 * CakePHP(tm) : Rapid Development Framework (https://cakephp.org)
 * Copyright (c) Cake Software Foundation, Inc. (https://cakefoundation.org)
 *
 * Licensed under The MIT License
 * For full copyright and license information, please see the LICENSE.txt
 * Redistributions of files must retain the above copyright notice.
 *
 * @copyright     Copyright (c) Cake Software Foundation, Inc. (https://cakefoundation.org)
 * @link          https://cakephp.org CakePHP(tm) Project
 * @since         0.10.0
 * @license       https://opensource.org/licenses/mit-license.php MIT License
 */
use Cake\Error\Debugger;
$this->layout = 'dev_error';
$this->assign('title', 'Database Error');
$this->assign('templateName', 'pdo_error.ctp');
```

```
$this->start('subheading');
?>
    <strong>Error: </strong>
    <?= h($message); ?>
<?php $this->end() ?>

<?php $this->start('file') ?>
<p class="notice">
    If you are using SQL keywords as table column names, you can enable identifier
    quoting for your database connection in config/app.php.
</p>
<?php if (!empty($error->queryString)) : ?>
    <p class="notice">
        <strong>SQL Query: </strong>
    </p>
    <pre><?= h($error->queryString); ?></pre>
<?php endif; ?>
<?php if (!empty($error->params)) : ?>
        <strong>SQL Query Params: </strong>
        <pre><?= h(Debugger::dump($error->params)); ?></pre>
<?php endif; ?>
<?= $this->element('auto_table_warning'); ?>
<?php $this->end() ?>
```

Dicas sobre erros no Cake

Tratamento de erro é uma missão nobre em aplicativos.
Aplicativos onde o programador prevê erros em todas as situações onde ele imagina que pode haver erro são aplicativos mais robustos. Os softwares de teste como o PHPUnit tem a missão de ajudar com isso.

Mensagens de Erro
Try Catch
Debug
Ferramentas

Inserir no controller que deseja monitorar:

```
      public function appError($method, $messages='Erro no Plugin<br><br>'){
        die('<pre>Mensagem: '.$messages.'Application error: called handler method<br>'.
$method.'</pre>');
      }


Configure::write('Error', array(
    'handler' => 'ErrorHandler::handleError',
    'level' => E_ALL & ~E_DEPRECATED,
    'trace' => true
));
```

Criando seu próprio manipulador de erro

```
//in app/Config/core.php
Configure::write('Error.handler', 'AppError::handleError');

//in app/Config/bootstrap.php
App::uses('AppError', 'Lib');

//in app/Lib/AppError.php
class AppError {
    public static function handleError($code, $description, $file = null, $line = null, $context
= null) {
        echo 'There has been an error!';
    }
}
```

Usando (requer PHP5.3 ou superior:

```
Configure::write('Error.handler', function($code, $description, $file = null, $line = null,
$context = null) {
    echo 'Oh no something bad happened';
});
```

Mudando o comportamento do erro fatal:

```
//in app/Config/core.php
Configure::write('Error.handler', 'AppError::handleError');

//in app/Config/bootstrap.php
App::uses('AppError', 'Lib');

//in app/Lib/AppError.php
class AppError {
    public static function handleError($code, $description, $file = null, $line = null, $context
= null) {
        list(, $level) = ErrorHandler::mapErrorCode($code);
        if ($level === LOG_ERROR) {
            // Ignore fatal error. It will keep the PHP error message only
            return false;
        }
        return ErrorHandler::handleError($code, $description, $file, $line, $context);
    }
}
```

From the PHP documentation (error_reporting):

```
<?php
// Turn off all error reporting
error_reporting(0);
?>
```

Other interesting options for that function:

```
<?php

// Report simple running errors
error_reporting(E_ERROR | E_WARNING | E_PARSE);

// Reporting E_NOTICE can be good too (to report uninitialized
// variables or catch variable name misspellings ...)
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Report all errors except E_NOTICE
// This is the default value set in php.ini
error_reporting(E_ALL & ~E_NOTICE);
// For PHP < 5.3 use: E_ALL ^ E_NOTICE

// Report all PHP errors (see changelog)
error_reporting(E_ALL);

// Report all PHP errors
error_reporting(-1);

// Same as error_reporting(E_ALL);
ini_set('error_reporting', E_ALL);

?>
```

O arquivo abaixo deve ser criado em
src/Template/Errors/pdo_error.ctp

```
<?php
/**
 * src/Templates/Error/pdo_error.ctp
 *
 * CakePHP(tm) : Rapid Development Framework (http://cakephp.org)
 * Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
 *
 * Licensed under The MIT License
 * For full copyright and license information, please see the LICENSE.txt
 * Redistributions of files must retain the above copyright notice.
 *
 * @copyright     Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
```

```
 * @link          http://cakephp.org CakePHP(tm) Project
 * @since         0.10.0
 * @license       http://www.opensource.org/licenses/mit-license.php MIT License
 */
use Cake\Utility\Debugger;
?>
<h2>Erro no Cadastro</h2>
<p class="error">
     <strong>Erro:<br></strong>

     <?php
      // Mensagem para o erro 23503
      if($error->getCode() == 23503){
          print "Este grupo não existe na tabela ContasGerais.<br>Caso tenha realmente
digitado de forma correta<br>e queira adicionar cadastre-o na tabela
ContasGerais<br>primeiro e depois cadastre o material com ele!";
       }

       //print $message;
   ?>
</p>
```

9 – Callbacks

Dicas sobre os CallBacks do Cake

Se você quiser colocar alguma lógica logo antes ou após uma operação de model do CakePHP, use funções tipo callbacks. Estas funções podem ser definidas em classes do model (incluindo o AppModel). Certifique-se de anotar os valores de retorno esperados para cada uma dessas funções especiais.

== CallBacks em models

beforeFind(array $queryData)

afterFind
afterFind(array $results, boolean $primary = false)

beforeValidate
beforeValidate(array $options = array())

beforeSave
beforeSave(array $options = array())

```
public function beforeSave($options = array()) {
    if (!empty($this->data['Event']['begindate']) && !empty($this->data['Event']['enddate'])) {
        $this->data['Event']['begindate'] = $this->dateFormatBeforeSave($this->data['Event']
['begindate']);
        $this->data['Event']['enddate'] = $this->dateFormatBeforeSave($this->data['Event']
['enddate']);
    }
    return true;
}
```

```
    /**
     * Converting the publish date field to mysql date format before save.
     *
     * @return bool
     */
    public function beforeSave() {
        if (isset($this->data[$this->alias]["publish_date"])) {
            $date = date("Y-m-d", strtotime($this->data[$this->alias]["publish_date"]));
            $this->data[$this->alias]["publish_date"] = $date;
        }
        return true;
    }
```

```
public function dateFormatBeforeSave($dateString) {
    return date('Y-m-d', strtotime($dateString));
}
```

afterSave
afterSave(boolean $created)

beforeDelete
beforeDelete(boolean $cascade = true)

```
// using app/Model/ProductCategory.php
// In the following example, do not let a product category be deleted if it still contains
products.
// A call of $this->Product->delete($id) from ProductsController.php has set $this->id .
// Assuming 'ProductCategory hasMany Product', we can access $this->Product in the
model.
public function beforeDelete($cascade = true) {
    $count = $this->Product->find("count", array(
        "conditions" => array("product_category_id" => $this->id)
    ));
    if ($count == 0) {
        return true;
    } else {
        return false;
    }
}
```

afterDelete
afterDelete()

Place any logic that you want to be executed after every deletion in this callback method.

onError
onError()

Called if any problems occur.

CallBacks de Controller

beforeFilter

beforeRender

afterFilter

CallBacks em Controllers

Lembre-se de adicionar os helpers Html e Form padrões se você incluiu o atributo $helpers em seu AppController.

Também lembre de fazer as chamadas de callbacks do AppController nos controllers filhos para obter melhores resultados:

```
function beforeFilter() {
    parent::beforeFilter();
}
```

Os controllers do CakePHP vêm equipados com callbacks que você pode usar para inserir lógicas em torno do ciclo de vida de uma requisição:

Controller::beforeFilter()

Este método é executado antes de cada ação dos controllers. É um ótimo lugar para verificar se há uma sessão ativa ou inspecionar as permissões de um usuário.

O método beforeFilter() será chamado para ações não encontradas e ações criadas pelo scaffold do Cake.

Controller::beforeRender()

Chamada após a lógica da ação de um controller, mas antes da view ser renderizada. Este callback não é usado com frequência mas pode ser preciso se você chamar o método render() manualmente antes do término de uma ação.

Controller::afterFilter()

Chamada após cada ação dos controllers, e após a completa renderização da view. Este é o último método executado do controller.

10 – Constantes e Funções

Constantes e Funções
https://book.cakephp.org/3.0/pt/core-libraries/global-constants-and-functions.html

A maior parte do seu trabalho diário com o CakePHP será feito utilizando classes e métodos do core. O CakePHP disponibiliza funções globais de conveniência que podem ajudar. Muitas dessas funções são usadas em classes do CakePHP (carregando um model ou um component), mas outras tornam mais fácil o trabalho de lidar com arrays ou strings.

Nós também vamos cobrir algumas das constantes existentes em aplicações CakePHP. Constantes essas, que facilitam upgrades e apontam convenientemente para arquivos e diretórios chaves da sua aplicação.
Funções globais

Aqui estão as funções disponíveis globalmente no CakePHP. A maioria delas são wrappers de conveniência para funcionalidades do CakePHP, como por exemplo, debug e localização de conteúdo.

__(string $string_id[, $formatArgs])

Essa função lida com a localização da sua aplicação. O $string_id identifica o ID usado para a tradução. Strings são tratadas seguindo o formato usado no sprintf(). Você pode fornecer argumentos adicionais para substituir placeholders na sua string:

__('Você tem {0} mensagens', $number);

Verifique a seção Internacionalização e Localização para mais informações.

__d(string $domain, string $msg, mixed $args = null)

Permite sobrescrever o domínio atual por uma mensagem simples.

Muito útil ao localizar um plugin: echo __d('PluginName', 'Esse é meu plugin');

__dn(string $domain, string $singular, string $plural, integer $count, mixed $args = null)

Permite sobrescrever o domínio atual por uma mensagem no plural. Retorna a forma corrreta da mensagem no plural identificada por $singular e $plural, pelo contador $count e pelo domínio $domain.

__dx(string $domain, string $context, string $msg, mixed $args = null)

Permite sobrescrever o domínio atual por uma mensagem simples. Também permite a especificação de um contexto.

O contexto é um identificador único para as strings de tradução que a tornam únicas sob um mesmo domínio.

__dxn(string $domain, string $context, string $singular, string $plural, integer $count, mixed $args = null)

Permite sobrescrever o domínio atual por uma mensagem no plural. Também permite a especificação de um contexto. Retorna a forma corrreta da mensagem no plural identificada por $singular e $plural, pelo contador $count e pelo domínio $domain. Alguns idiomas tem mais de uma forma para o plural dependendo do contador.

O contexto é um identificador único para as strings de tradução que a tornam únicas sob um mesmo domínio.

__n(string $singular, string $plural, integer $count, mixed $args = null)

Retorna a forma corrreta da mensagem no plural identificada por $singular e $plural, pelo contador $count e pelo domínio $domain. Alguns idiomas tem mais de uma forma para o plural dependendo do contador.

__x(string $context, string $msg, mixed $args = null)

O contexto é um identificador único para as strings de tradução que a tornam únicas sob um mesmo domínio.

__xn(string $context, string $singular, string $plural, integer $count, mixed $args = null)

Retorna a forma corrreta da mensagem no plural identificada por $singular e $plural, pelo contador $count e pelo domínio $domain. Alguns idiomas tem mais de uma forma para o plural dependendo do contador.

O contexto é um identificador único para as strings de tradução que a tornam únicas sob um mesmo domínio.

collection(mixed $items)

Wrapper de conveniência para instanciar um novo objeto Cake\Collection\Collection, re-passando o devido argumento. O parâmetro $items recebe tanto um objeto Traversable quanto um array.

debug(mixed $var, boolean $showHtml = null, $showFrom = true)

Alterado na versão 3.3.0: Esse método retorna a $var passada para que você possa, por instância, colocá-la em uma declaração de retorno.

Se a variável do core $debug for true, $var será imprimida. Se $showHTML for true, ou for deixada como null os dados serão renderizados formatados para melhor exibição em navegadores. Se $showFrom não for definida como false, o debug começará a partir da linha em que foi chamado. Também veja Depuração

pr(mixed $var)

Alterado na versão 3.3.0: Chamar esse método vai retornar a $var passada, então, você pode, por instância, colocá-la em uma declaração de retorno.

Wrapper de conveniência para print_r() com a adição das tags <pre> ao redor da saída.

pj(mixed $var)

Alterado na versão 3.3.0: Chamar esse método vai retornar a $var passada, então, você pode, por instância, colocá-la em uma declaração de retorno.

Função de conveniência para formatação de JSON, com a adição das tags <pre> ao redor da saída.

Deve ser usada com o intuito de debugar JSON de objetos e arrays.

env(string $key, string $default = null)

Alterado na versão 3.1.1: O parâmetro $default será adicionado.

Recebe uma variável de ambiente de fontes disponíveis. Usada como backup se $_SERVER ou $_ENV estiverem desabilitados.

Essa função também emula PHP_SELF e DOCUMENT_ROOT em servidores não suportados. De fato, é sempre uma boa ideia usar env() ao invés de $_SERVER``ou ``getenv() (especialmente se você planeja distribuir o código), pois é um wrapper completo de emulação.

h(string $text, boolean $double = true, string $charset = null)

Wrapper de conveniência para htmlspecialchars().

pluginSplit(string $name, boolean $dotAppend = false, string $plugin = null)

Divide um nome de plugin que segue o padrão de sintaxe de pontos e o transforma em um nome de classe ou do plugin. Se $name não tem um ponto, então o índice 0 será null.

Comumente usada assim: list($plugin, $name) = pluginSplit('Users.User');

namespaceSplit(string $class)

Divide o namespace do nome da classe.

Comumente usada assim: list($namespace, $className) = namespaceSplit('Cake\Core\App');

Constantes de definição do Core

A maior parte das constantes a seguir referem-se a caminhos da sua aplicação.

constant APP

Caminho absoluto para o diretório de sua aplicação, incluindo a barra final.

constant APP_DIR

Igual a app ou ao nome do diretório de sua aplicação.

constant CACHE

Caminho para o diretório de arquivos de cache. Pode ser compartilhado entre hosts em uma configuração multi-servidores.

constant CAKE

Caminho para o diretório do CakePHP.

constant CAKE_CORE_INCLUDE_PATH

Caminho para o diretório raiz de bibliotecas.

constant CONFIG

Caminho para o diretório de configurações.

constant CORE_PATH

Caminho para o diretório raiz com contra-barra no final.

constant DS

Atalho para o DIRECTORY_SEPARATOR do PHP, que é / no Linux e \\ no Windows.

constant LOGS

Caminho para o diretório de logs.

constant ROOT

Caminho para o diretório raiz.

constant TESTS

Caminho para o diretório de testes.

constant TMP

Caminho para o diretório de arquivos temporários.

constant WWW_ROOT

Caminho completo para o diretório webroot.

Constantes de definição de tempo

constant TIME_START

Timestamp unix em microsegundos como float de quando a aplicação começou.

constant SECOND

Igual a 1

constant MINUTE

Igual a 60

constant HOUR

Igual a 3600

constant DAY

Igual a 86400

constant WEEK

Igual a 604800

constant MONTH

Igual a 2592000

constant YEAR

Igual a 31536000

11 – Ferramentas

11.2 - Faker + CakePHP = gourmet/faker

Some time ago, I did some work on Faker to add built-in support for CakePHP's awesome new ORM. It has just been merged, and I released gourmet/faker to make it even easier to use.

Here's how to install it:

```
$ composer require gourmet/faker
```

At this point, Faker can already be used anywhere in the application. Here's an example for seeding 20 records in the `Posts` table (i.e. in a migration file):

```
<?php
$faker = \Faker\Factory::create();
$entityPopulator = new \Faker\ORM\CakePHP\EntityPopulator('Posts');
$populator = new \Faker\ORM\CakePHP\Populator($faker);
$populator->addEntity($entityPopulator, 20);
$populator->execute(['validate' => false]);
```

It could also be used to auto-generate records in fixtures like so:

```
<?php

namespace App\Test\Fixture;

use Gourmet\Faker\TestSuite\Fixture\TestFixture;

class PostsFixture extends TestFixture {

    public $fields = [
        'id' => ['type' => 'integer'],
        'title' => ['type' => 'string', 'length' => 255, 'null' => false],
        'body' => 'text',
        'published' => ['type' => 'integer', 'default' => '0', 'null' => false],
        'created' => 'datetime',
        'updated' => 'datetime',
        '_constraints' => [
            'primary' => ['type' => 'primary', 'columns' => ['id']]
        ]
    ];

    public $records = [
        [
            'id' => 1,
            'title' => 'First Article',
            'body' => 'First Article Body',
            'published' => '1',
            'created' => '2007-03-18 10:39:23',
            'updated' => '2007-03-18 10:41:31'
        ],
        [
            'id' => 2,
            'title' => 'Second Article',
```

```
            'body' => 'Second Article Body',
            'published' => '1',
            'created' => '2007-03-18 10:41:23',
            'updated' => '2007-03-18 10:43:31'
        ],
        [
            'id' => 3,
            'title' => 'Third Article',
            'body' => 'Third Article Body',
            'published' => '1',
            'created' => '2007-03-18 10:43:23',
            'updated' => '2007-03-18 10:45:31'
        ]
    ];


    public $number = 20;
    public $guessers = ['\Faker\Guesser\Name'];

    public function init() {
        $this->customColumnFormatters = [
            'id' => function () { return $this->faker-
>numberBetween(count($this->records) + 2); },
            'published' => function () { return rand(0,3); }
        ];
        parent::init();
    }
}
```

What's cool about this fixture is that it supports the default `TestFixture` inserting of custom records.

Here's how it works under the hood:

```
<?php
public function create(Connection $db) {
    if (!parent::create($db)) {
        return false;
    }

    $entityPopulator = new EntityPopulator($this->table);
    $populator = new Populator($this->faker);
    array_walk($this->guessers, array($populator, 'addGuesser'));
    $populator->addEntity($entityPopulator, $this->number, $this-
>customColumnFormatters, $this->customModifiers);
    $populator->execute(['validate' => false]);
    return true;
}
```

That's it! Happy faking :)

11.2 – phpDoc

```
<?php
/**
 * @author Um nome <a.name@example.com>
 * @link http://www.phpdoc.org/docs/latest/index.html
 * @package helper
 */
class DateTimeHelper
{
    /**
     * @param mixed $anything Tudo que podemos converter para um objeto \DateTime
     *
     * @return \DateTime
     * @throws \InvalidArgumentException
     */
    public function dateTimeFromAnything($anything)
    {
        $type = gettype($anything);

        switch ($type) {
            // Algum código que tenta retornar um objeto \DateTime
        }

        throw new \InvalidArgumentException(
            "Failed Converting param of type '{$type}' to DateTime object"
        );
    }

    /**
     * @param mixed $date Tudo que podemos converter para um objeto \DateTime
     *
     * @return void
     */
    public function printISO8601Date($date)
    {
        echo $this->dateTimeFromAnything($date)->format('c');
    }

    /**
     * @param mixed $date Tudo que podemos converter para um objeto \DateTime
     */
    public function printRFC2822Date($date)
    {
        echo $this->dateTimeFromAnything($date)->format('r');
    }
}
```

11.3 – Conversão para o CakePHP 3.6

Quando executar

bin/cake plugin load migrations

E ele reclamar:
Your Application class does not have a bootstrap() method. Please add one.

Adicione ao src/Application.php

```
    public function bootstrap()
    {
        parent::bootstrap();
    }
```

Loading a Plugin

If you want to use a plugin's routes, console commands, middleware, or event listeners you will need to load the plugin. Plugins are loaded in your application's bootstrap() function:

```
// In src/Application.php. Requires at least 3.6.0
use Cake\Http\BaseApplication;
use ContactManager\Plugin as ContactManager;

class Application extends BaseApplication {
    public function bootstrap()
    {
        parent::bootstrap();
        // Load the contact manager plugin by class name
        $this->addPlugin(ContactManager::class);

        // Load a plugin with a vendor namespace by 'short name'
        $this->addPlugin('AcmeCorp\ContactManager');
    }
}
```

If you just want to use helpers, behaviors or components from a plugin you do not need to load a plugin.

viewBuilder()->layout()

Para
viewBuilder()->setLayout()

$this->request->data

para

$this->request->getData

$this->response->getType();// Era apenas type() no 3.5

TableSchema::columnType() is deprecated. Use TableSchema::setColumnType() or TableSchema::getColumnType() instead

ServerRequest::session() is deprecated. Use getSession() instead.
$user = $this->request->getSession()->read('Auth.User');

controller, use getParam()
$this->request->getParam('controller');

Accessing routing parameters through `action` will removed in 4.0.0. Use `getParam()` instead.
$action = $this->request->getParam('action');

RequestHandlerComponent::beforeRedirect() is deprecated. This functionality will be removed in 4.0.0. Set the `enableBeforeRedirect` option to `false` to disable this warning.

11.4 – Instalador Web do CakePHP

https://github.com/CakeDC/oven

12 – Model

O Model representa a primeira letra do MVC.

Models (Modelos) são as classes que servem como camada de negócio na sua aplicação. Isso significa que eles devem ser responsáveis pela gestão de quase tudo o que acontece em relação a seus dados, sua validade, interações e evolução do fluxo de trabalho de informação no domínio do trabalho.

No CakePHP seu modelo de domínio da aplicação é dividido em 2 tipos de objetos principais. Os primeiros são repositories (repositórios) ou table objects (objetos de tabela). Estes objetos fornecem acesso a coleções de dados. Eles permitem a você salvar novos registros, modificar/deletar os que já existem, definir relacionamentos, e executar operações em massa. O segundo tipo de objetos são as entities (entidades). Entities representam registros individuais e permitem a você definir comportamento em nível de linha/registro e funcionalidades.

O ORM (MOR - Mapeamento Objeto-Relacional) nativo do CakePHP especializa-se em banco de dados relacionais, mas pode ser extendido para suportar fontes de dados alternativas.

O ORM do Cakephp toma emprestadas ideias e conceitos dos padrões ActiveRecord e Datamapper. Isso permite criar uma implementação híbrida que combina aspectos de ambos padrões para criar uma ORM rápida e simples de utilizar.

Atributos do Model

Indicar nome de configuração do database
public $useDbConfig = 'alternate';

Prefixo das Tabelas
public $tablePrefix = 'alternate_';

Nome do campo Primary Key
public $primaryKey = 'example_id';

Display field
Nomes de campos em tabelas para tabelas relacionadas
Convenção: title ou name
Se diferente indicar no model com:
public $displayField = 'nomedocampo';

Tabela Default
public $useTable = 'users';

Caso receba a mensagem:
Erro: Table useres for model User was not found in datasource default.
Isso aconteceu comigo sempre que instalei o o plugin Cakept_br.

Adicione a variável $useTable contendo o nome da tabela ao model User
public $useTable = 'usuarios';

Acessando Outros Models ou Controllers

Chamando outro controller:

```
App::import('Controller', 'Users');
$Users = new UsersController;
$Users->constructClasses();
```

Estando num model chamar outro model:

```
$Category = ClassRegistry::init("OutroModel");
$category = $Category->findById($underThisCategoryId);
```

Colocando classes personalizadas a disposição do aplicativo em Cake:

- insira a classe em um diretório
- Adicione ao app/Config/bootstrap.php (diretório fora do aplicativo do cake):

```
App::build(array(
    'GlobalUsers' => array(dirname(CAKE_CORE_INCLUDE_PATH).DS.'mydir'.DS)
), App::REGISTER);
```

- Adicione ao AppController.php:

```
App::uses('UsersController','GlobalUsers');
```

UsersController - nome do arquivo atual, no caso UsersController.php
Agora podemos acessar a classe do nosso controller atual.
Crédito: http://www.shahariaazam.com/access-custom-class-inside-cakephp-apps/

Salvando Dados

```
if ($this->ModelName->save($this->request->data)) {
   $this->Session->setFlash('Data Saved!');
}
```

Salvando Muitos

```
$data = array(
   array('title' => 'title 1'),
   array('title' => 'title 2'),
);
$this->ModelName->saveAll($data);
```

Delete

```
$this->Model->delete($this->request->data('Model.id'));
```

Delete all

$this->Model->deleteAll(array('Model.spam' => true), false);

Recebendo dados

find threaded
find neighbors
findAllBy
findBy
query
field
read
More complex CakePHP find examples

CakePHP Model properties, methods, callbacks, and validation

CakePHP Model properties:

$belongsTo = array()
$cacheQueries = true
$data = array()
$displayField = null
$hasAndBelongsToMany = array()
$hasMany = array()
$hasOne = array()
$id = false
$logTransactions = false
$name = null
$primaryKey = null
$recursive = 1
$useDbConfig = 'default'
$useTable = null
$validate = array()
$validationErrors = array()

CakePHP Model methods:

bindModel ($params)
create ()
delete ($id = null, $cascade = true)
escapeField ($field)
execute ($data)
exists ()
field ($name, $conditions = null, $order = null)
find ($conditions = null, $fields = null, $order = null, $recursive = null)
findAll ($conditions = null, $fields = null, $order = null, $limit = null, $page = 1, $recursive = null)

findAllThreaded ($conditions = null, $fields = null, $sort = null)
findCount ($conditions = null, $recursive = 0)
findNeighbours ($conditions = null, $field, $value)
generateList ($conditions = null, $order = null, $limit = null, $keyPath = null, $valuePath = null)
getAffectedRows ()
getColumnType ($column)
getColumnTypes ()
getDisplayField ()
getID ($list=0)
getLastInsertID ()
getNumRows ()
hasAny ($conditions = null)
hasField ($name)
invalidate ($field)
invalidFields ($data = array())
isForeignKey ($field)
loadInfo ()
query ()
read ($fields = null, $id = null)
remove ($id = null, $cascade = true)
save ($data = null, $validate = true, $fieldList = array())
saveField ($name, $value, $validate = false)
set ($one, $two = null)
setDataSource ($dataSource = null)
setSource ($tableName)
unbindModel ($params)
validates ($data=array())
setSource ($tableName)

CakePHP Model callbacks

afterDelete ()
afterFind ($results)
afterSave ()
beforeDelete ()
beforeFind (&$queryData)
beforeSave ()
beforeValidate ()

CakePHP Model validation

'VALID_EMAIL`
'VALID_NOT_EMPTY`
'VALID_NUMBER`
'VALID_YEAR`

Controller::loadModel(string $modelClass, mixed $id)

O método loadModel vem a calhar quando você precisa usar um model que não é padrão do controller ou o seu model não está associado com este.

```
$this->loadModel('Article');
$recentArticles = $this->Article->find('all', array('limit' => 5, 'order' => 'Article.created DESC'));

$this->loadModel('User', 2);
$user = $this->User->read();
```

Indicando o prefixo das tabelas:

```
class Example extends AppModel {
    public $tablePrefix = 'prefx_'; // will look for 'prefx_examples'
}
```

Indicando o nome da primary key (somente quando diferente de id)

```
class Example extends AppModel {
    public $primaryKey = 'example_id'; // example_id is the field name in the database
}
```

Indicando o label de uma tabela: usado pelo gerador em relacionamentos para trazer como título da tabela. Se for name ou title não precisa disso. Apenas quando difefrente:

```
class User extends AppModel {
    public $displayField = 'descricao';
}
```

name
Name do model. Se não for especificadono model então o model seta o nome da classe pelo construtor.

```
class Example extends AppModel {
    public $name = 'Example';
}
```

Usando CallBacks

```
    /**
     * Converting the publish date field to mysql date format before save.
     *
     * @return bool
     */
    public function beforeSave() {
        if (isset($this->data[$this->alias]["publish_date"])) {
            $date = date("Y-m-d", strtotime($this->data[$this->alias]["publish_date"]));
            $this->data[$this->alias]["publish_date"] = $date;
        }
        return true;
```

```
}
```

Algumas funções customizadas

```
/**
 * latest function.
 *
 * @access public
 * @return array
 */
public function latest($limit = 1) {
	$posts = $this->find('all', array(
		'conditions' => array(
			'Post.is_published' => 1
		),
		'order' => array(
			'Post.publish_date' => 'DESC',
			'Post.created' => 'DESC'
		),
		'limit' => $limit
	));
	if (count($posts) === 1 && $limit === 1) {
		$posts = $posts[0];
	}
	return $posts;
}

/**
 * recent function.
 *
 * @access public
 * @return array
 */
public function recent() {
	return $this->find('all', array(
		'conditions' => array(
			'Post.is_published' => 1
		),
		'order' => array(
			'Post.publish_date' => 'DESC'
		),
		'limit' => 5
	));
}
```

12.1 – Validações

Validações

E-mail requerido

```
        $validator
            ->email('email')
            ->notBlank('email')
            ->add('email', 'unique', [
             'rule' => 'validateUnique',
             'provider' => 'table'
            ]);
ou
        $validator
                ->requirePresence('email')
                ->add('email', 'validFormat', [
                        'rule' => 'email',
                        'message' => 'E-mail inválido'
                ]);
```

Nome requerido

```
        $validator
            ->requirePresence('nome', 'create')
            ->notEmpty('nome');
```

No Model/Table/ClientesTable.php

Rule customizada para o campo nascimento

```
    public function validationDefault(Validator $validator)
    {
        $validator
            ->integer('id')
            ->allowEmpty('id', 'create');

        $validator
            ->requirePresence('nome', 'create')
            ->notEmpty('nome');

        $validator
            ->email('email')
            ->notBlank('email')
            ->add('email', 'unique', ['rule' => 'validateUnique', 'provider' => 'table']);

//        $validator
```

```
//          ->date('nascimento')
  //        ->allowEmpty('nascimento');

        $validator->add('nascimento',[
             'notEmptyCheck'=>[
               'rule'=>[$this,'notEmptyNascimento'],
               'provider'=>'table',
               'message'=>'Favor selecionar uma data de nascimento'
             ]
         ]);

        $validator
            ->allowEmpty('cpf');

        $validator
            ->allowEmpty('cnpj');

        $validator
            ->allowEmpty('fone');

        $validator
            ->allowEmpty('observacao');

        $validator
            ->notBlank('user_id');

        return $validator;
    }

    public function notEmptyNascimento($value,$context){
        if(empty($context['data']['nascimento'])) {
            return false;
        } else {
            return true;
        }
    }
```

Dica: O campo group_id não aparecia por padrão na lista de validações e mesmo que não fosse selecionado nenhum grupo
o registro era armazenado. Então adicionei a validação como notBlank e na view add.ctp usei:

```
            echo $this->Form->input('user_id', ['options' => $users,'empty'=>true]);
```

Assim aparece o asterisco vermelho indicando requerido e somente quando alguém esoclhe um grupo na lista é permitodo cadastrar.

Sugestão para lista

```
    public function validationDefault(Validator $validator)
```

```
    {
        return $validator
            ->notEmpty('first_name', 'A username is required')
            ->notEmpty('last_name', 'A username is required')
            ->notEmpty('email', 'A username is required')
            ->notEmpty('username', 'A username is required')
            ->notEmpty('password', 'A username is required')
            ->add('role', 'inList', [
                'rule' => ['inList', ['Employer', 'Job Seeker']],
                'message' => 'Please enter a valid role'
            ]);
    }
```

Como o Cake exige que as datas sejam null por default, uma forma de contornar isso exigindo o preenchimento
é criando uma validação customizada:

No Model/Table/ClientesTable.php

```
    public function validationDefault(Validator $validator)
    {
            ...
        $validator->add('nascimento',[
              'notEmptyCheck'=>[
                'rule'=>[$this,'notEmptyNascimento'],
                'provider'=>'table',
                'message'=>'Favor selecionar uma data de nascimento'
              ]
         ]);
            ...
        return $validator;
    }

    public function notEmptyNascimento($value,$context){
        if(empty($context['data']['nascimento'])) {
            return false;
        } else {
            return true;
        }
    }
```

Sugestão para lista

```
    public function validationDefault(Validator $validator)
    {
        return $validator
            ->notEmpty('first_name', 'A username is required')
            ->notEmpty('last_name', 'A username is required')
            ->notEmpty('email', 'A username is required')
            ->notEmpty('username', 'A username is required')
```

```
            ->notEmpty('password', 'A username is required')
            ->add('role', 'inList', [
                'rule' => ['inList', ['Employer', 'Job Seeker']],
                'message' => 'Please enter a valid role'
            ]);
    }
```

Como o Cake exige que as datas sejam null por default, uma forma de contornar isso exigindo o preenchimento
é criando uma validação customizada:

No Model/Table/ClientesTable.php

```
    public function validationDefault(Validator $validator)
    {
            ...
        $validator->add('nascimento',[
             'notEmptyCheck'=>[
               'rule'=>[$this,'notEmptyNascimento'],
               'provider'=>'table',
               'message'=>'Favor selecionar uma data de nascimento'
             ]
         ]);
            ...
        return $validator;
    }

    public function notEmptyNascimento($value,$context){
        if(empty($context['data']['nascimento'])) {
            return false;
        } else {
            return true;
        }
    }
```

Validação de telefone (falta testar)

```
'fone' => [
        'rule' => array('isValidBRFoneFormat')
    ],

// Formatos aceitos: (99) 99999-9999 e (99) 9999-9999

 /*isValidBRFoneFormat() - Custom method to validate US Phone Number
 * @params Int $phone
 */
 function isValidBRFoneFormat($phone){
 $fone=$fone['fone'];
 $errors = array();
```

```
    if(empty($fone)) {
        $errors [] = "Favor entrar um tenefone válido";
    }
    else if (!preg_match('/^(\(11\) [9][0-9]{4}-[0-9]{4})|(\(1[2-9]\) [5-9][0-9]{3}-[0-9]{4})|(\([2-9]
[1-9]\) [5-9][0-9]{3}-[0-9]{4})$/', $fone)) {
        $errors [] = "Favor entrar um tenefone válido";
    }

    if (!empty($errors))
    return implode("\n", $errors);

    return true;
}
```

Dicas de Validações

Validações

E-mail requerido

```
        $validator
            ->email('email')
            ->notBlank('email')
            ->add('email', 'unique', [
             'rule' => 'validateUnique',
             'provider' => 'table'
            ]);
```

ou

```
        $validator
                ->requirePresence('email')
                    ->add('email', 'validFormat', [
                        'rule' => 'email',
                        'message' => 'E-mail inválido'
                    ]);
```

Nome requerido

```
        $validator
            ->requirePresence('nome', 'create')
            ->notEmpty('nome');
```

No Model/Table/ClientesTable.php

Rule customizada para o campo nascimento

```
    public function validationDefault(Validator $validator)
    {
```

```
        $validator
            ->integer('id')
            ->allowEmpty('id', 'create');

        $validator
            ->requirePresence('nome', 'create')
            ->notEmpty('nome');

        $validator
            ->email('email')
            ->notBlank('email')
            ->add('email', 'unique', ['rule' => 'validateUnique', 'provider' => 'table']);

        $validator->add('nascimento',[
              'notEmptyCheck'=>[
                'rule'=>[$this,'notEmptyNascimento'],
                'provider'=>'table',
                'message'=>'Favor selecionar uma data de nascimento'
              ]
         ]);

        $validator
            ->allowEmpty('cpf');

        $validator
            ->allowEmpty('cnpj');

        $validator
            ->allowEmpty('fone');

        $validator
            ->allowEmpty('observacao');

        $validator
            ->notBlank('user_id');

        return $validator;
    }

    public function notEmptyNascimento($value,$context){
        if(empty($context['data']['nascimento'])) {
            return false;
        } else {
            return true;
        }
    }
```

Dica: O campo group_id não aparecia por padrão na lista de validações e mesmo que não fosse selecionado nenhum grupo

o registro era armazenado. Então adicionei a validação como notBlank e na view add.ctp usei:

```
echo $this->Form->input('user_id', ['options' => $users,'empty'=>true]);
```

Assim aparece o asterisco vermelho indicando requerido e somente quando alguém esoclhe um grupo na lista é permitodo cadastrar.

Sugestão para lista

```
public function validationDefault(Validator $validator)
{
    return $validator
        ->notEmpty('first_name', 'A username is required')
        ->notEmpty('last_name', 'A username is required')
        ->notEmpty('email', 'A username is required')
        ->notEmpty('username', 'A username is required')
        ->notEmpty('password', 'A username is required')
        ->add('role', 'inList', [
            'rule' => ['inList', ['Employer', 'Job Seeker']],
            'message' => 'Please enter a valid role'
        ]);
}
```

Como o Cake exige que as datas sejam null por default, uma forma de contornar isso exigindo o preenchimento
é criando uma validação customizada:

No Model/Table/ClientesTable.php

```
public function validationDefault(Validator $validator)
{
        ...
    $validator->add('nascimento',[
          'notEmptyCheck'=>[
            'rule'=>[$this,'notEmptyNascimento'],
            'provider'=>'table',
            'message'=>'Favor selecionar uma data de nascimento'
          ]
     ]);
        ...
    return $validator;
}

public function notEmptyNascimento($value,$context){
    if(empty($context['data']['nascimento'])) {
        return false;
    } else {
        return true;
    }
```

```
    }
```

Sugestão para lista

```
    public function validationDefault(Validator $validator)
    {
        return $validator
            ->notEmpty('first_name', 'A username is required')
            ->notEmpty('last_name', 'A username is required')
            ->notEmpty('email', 'A username is required')
            ->notEmpty('username', 'A username is required')
            ->notEmpty('password', 'A username is required')
            ->add('role', 'inList', [
                'rule' => ['inList', ['Employer', 'Job Seeker']],
                'message' => 'Please enter a valid role'
            ]);
    }
```

Exemplo

```
<?php
public $validate = array(
    'username' => array(
        'notempty' => array(
            'rule' => array('notempty'),
            'message' => 'Username cannot be empty',
        ),
        'unique' => array(
            'rule' => array('isUnique'),
            'message' => 'That username already exists. Please choose another',
            'on'=>'create',
        ),
        'alphaNumeric' => array(
            'rule' => 'alphaNumeric',
            'required' => true,
            'message' => 'Username can consist of letters and numbers only'
        ),
        'between' => array(
            'rule' => array('between', 5, 15),
            'message' => 'Username must be Between 5 and 15 characters'
        )
    ),
    'email' => array(
        'email' => array(
            'rule' => array('email'),
            'message' => 'Please use a valid email',
        ),
        'notempty' => array(
            'rule' => array('notEmpty'),
```

```
            'message'=> 'Email cannot be empty',
        ),
        'unique' => array(
            'rule' => 'isUnique',
            'message' => 'The email you entered has already been registered',
            'on'=>'create'
        ),
    ),
    'password' => array(
        'notempty' => array(
            'rule' => array('notempty'),
            'message' => 'Password cannot be empty',
        ),
        'mingLength' => array(
            'rule' => array('minLength', 6),
            'message' => 'Mimimum 6 characters long',
        ),
        'maxgLength' => array(
            'rule' => array('maxLength', 30),
            'message' => 'Maximum 30 characters long',
        )
    ),
);
```

Dicas sobre Cake com Bancos de dados

Relacionamentos

O campo secundário do relacionamento precisa ser NOT NULL na tabela.

Exemplo
users com clientes

Em clientes temos o campo user_id que relaciona as tabelas.
Na tabela clientes o campo user_id precisa ser:

user_id int not null

Caso contrário teremos sérios problemas de cadastramento de todos os registros vazios e fura o relacionamento.

Relacionamentos
https://book.cakephp.org/3.0/pt/orm/associations.html

Dicas Sobre Relacionamentos

Ao implementar o relacionamento entre duas tabelas seguindo as convenções do Cake recebemos vários recursos bem úteis:

- O respectivo código para o relacionamento já será criado no model pelo bake. Caso estejamos criando manualmente o código estará disponível.
- Campo da tabela relacionada é transformado em uma combo na view. Se seguirmos outra convenção para os nomes dos campos nas tabelas, o campo tabela_id se transforma em title ou name da tabela relacionada.

Exemplo: temos
groups (id, name) e
users (id, username, password, group_id).
Criando o código dos CRUDs para as duas tabelas pelo bake, ao listar users, onde tem group_id aparecerá o name de groups.

Associações - Conectando tabelas

Definindo a relação entre diferentes objectos na sua aplicação deveria ser um processo natural. Por exemplo, um artigo deve ter muitos comentários, e pertencer a um autor. Autores deve ter muitos artigos e logo muitos comentários. O CakePHP torna fácil a gestão das relações entre os modelos. As quatro tipo de associações no CakePHP são: hasOne, hasMany, belongsTo, and belongsToMany.

Relação	Tipo de associação	Exemplo
um para um	hasOne	Um usuário tem um perfil.
um para muitos	hasMany	Um usuário pode ter vários artigos.
muitos para um	belongsTo	Muitos artigos pertencem a um usuário.
muitos para muitos	belongsToMany	Muitas Tags pertencem a muitos artigos.

Associações são definidas durante o método initialize() do objeto da sua tabela. Métodos que fechem com o tipo de associação permitem a você definir as associações da suaaplicação. Por exemplo se nós quisermos definir uma associação do tipo belongsTo em nosso ArticlesTable:

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->belongsTo('Authors');
    }
}
```

A forma mais simples de definição de qualquer associação é usar o alias do modelo que você quer associar. Por padrão todos os detalhes das associações serão usadas as conveões do CakePHP. Se você quiser customizar a forma como as suas associações são trabalhadas, você pode modificar elas com os setters:

```
class ArticlesTable extends Table
```

```
{
    public function initialize(array $config)
    {
        $this->belongsTo('Authors', [
                'className' => 'Publishing.Authors'
            ])
            ->setForeignKey('authorid')
            ->setProperty('person');
    }
}
```

Você pode também usar arrays para customizar suas associações:

```
$this->belongsTo('Authors', [
    'className' => 'Publishing.Authors',
    'foreignKey' => 'authorid',
    'propertyName' => 'person'
]);
```

Como você pode ver, por especificar a chave da className , é possível utilizar a mesma tabela em diferentes associações para a mesma tabela. Você até mesmo pode criar uma auto associação da tabela, criando uma estrutura de relação pai-filho:

```
class CategoriesTable extends Table
{
    public function initialize(array $config)
    {
        $this->hasMany('SubCategories', [
            'className' => 'Categories'
        ]);

        $this->belongsTo('ParentCategories', [
            'className' => 'Categories'
        ]);
    }
}
```

Você também pode definir associações em massa ao criar uma única chamada, para Table::addAssociations() onde aceita um array contendo o nome das tabelas por associação como um argumento:

```
class PostsTable extends Table
{
    public function initialize(array $config)
    {
      $this->addAssociations([
          'belongsTo' => [
              'Users' => ['className' => 'App\Model\Table\UsersTable']
```

```
            ],
            'hasMany' => ['Comments'],
            'belongsToMany' => ['Tags']
        ]);
    }
}
```

Nós também podemos definir uma relação mais especifica usando os setters:

```
class AddressesTable extends Table
{
    public function initialize(array $config)
    {
        // Prior to 3.4 version, use foreignKey() and joinType()
        $this->belongsTo('Users')
            ->setForeignKey('user_id')
            ->setJoinType('INNER');
    }
}
```

Chaves possíveis para associações belongsTo arrays inclui:

classNAme: o nome da classe do modelo que está sendo associado ao modelo atual. Se você está definindo que a relação 'Profile belongsTo User', a chave da className deveria ser igual a 'Users'.

foreignKey: o nome da chave estrangeira encontrada na tabela atual. Este é especialmente acessível, se você precisar definir multiplas relações belongsTo ao mesmo modelo. O valor padrão para esta chave é o sublinhado, e nome singular do outro modelo, com o sufixo _id.

bindingKey: O nome da coluna da outra tabela, este ser´a usado para o match de foreignKey. Se não especificado, a chave primária (por exemplo o id da tabela Users) será usado.

conditions: como um array para o find(), compativel com as condições ou SQL como ['Users.active' => true]

joinType: o tipo de join a ser usado na query SQL, por padrão é LEFT aos quais não deve atender as suas necessidades para todas as situações, INNER deve ajudar quando você quiser do seu modelo ou associados ou nenhum deles.

propertyName: A propriedade nome deve preenchida com os dados das tabelas associadas e dos resultados. Pos padrão este é o sublinhado e nome singular da associação, então user em nosso exemplo.

strategy: Define uma estratégia de query a ser usada. Por padrão o 'join'. O outro valor valido é 'select', aos quais utiliza uma query separada.

finder: O método finder é usado quando carregamos registros associados.

Uma vez que esta associação é definida, operações to tipo dina para a tabela Addresses pode conter o registro de User se este existir:

```
// No controller ou no metodo de table.
$query = $addresses->find('all')->contain(['Users']);
foreach ($query as $address) {
    echo $address->user->username;
}
```

Abaixo emite um SQL que é similar a:

```
SELECT * FROM addresses LEFT JOIN users ON addresses.user_id = users.id;
```

Nós podemos definir uma associação hasMany em noos modelo de Articles seguindo:

```
class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->hasMany('Comments');
    }
}
```

Podemos definir a associação belongsToMany em ambos os modelos da seguinte forma:

```
// In src/Model/Table/ArticlesTable.php
class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->belongsToMany('Tags');
    }
}

// In src/Model/Table/TagsTable.php
class TagsTable extends Table
{
    public function initialize(array $config)
    {
        $this->belongsToMany('Articles');
    }
}
```

Implementação do displayField()

Tenho duas tabelas relacionadas: users e clientes.
Clientes tem um campo user_id

Em
src/Model/Table/UsersTable.php

Mude o displayField para mostrar não o id mas o username

```
    $this->displayField('username');
```

Aproveite e torne requerido o campo user_id em clientes

src/Model/Table/ClientesTable.php

Adicione

```
    $validator
        ->notBlank('user_id');
```

No Clientes/index.ctp no valor do user_id:

```
            <td><?= $cliente->has('user') ? $this->Html->link($cliente->user->id, ['controller' => 'Users', 'action' => 'view', $cliente->user->id]) : '' ?></td>
```

Entidades\Entity

https://book.cakephp.org/3.0/pt/orm/entities.html

class Cake\ORM\Entity

Enquanto Objetos de tabela representam e fornecem acesso a uma coleção de objetos, entidades representam linhas individuais ou objetos de domínio na sua aplicação. Entidades contêm propriedades persistentes e métodos para manipular e acessar os dados que elas contêm.

Entidades são criadas para você pelo CakePHP cada vez que utilizar o find() em um objeto de Table.

```
// src/Model/Entity/Article.php
namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{
```

```
}
```

Acessando dados de uma entity

```
use App\Model\Entity\Article;

$article = new Article;
$article->title = 'This is my first post';
echo $article->title;
```

Podemos usar get e set

```
$article->set('title', 'This is my first post');
echo $article->get('title');
```

Ao usar set(), você pode atualizar várias propriedades ao mesmo tempo usando um array:

```
$article->set([
    'title' => 'My first post',
    'body' => 'It is the best ever!'
]);
```

Accessors & Mutators

Além da simples interface get/set, as entidades permitem que você forneça métodos acessadores e mutadores. Esses métodos deixam você personalizar como as propriedades são lidas ou definidas.

Acessadores usam a convenção de _get seguido da versão CamelCased do nome do campo.

```
Cake\ORM\Entity::get($field)
```

Eles recebem o valor básico armazenado no array _properties como seu único argumento. Acessadores serão usadas ao salvar entidades, então seja cuidadoso ao definir métodos que formatam dados, já que os dados formatados serão persistido. Por exemplo:

```
namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{
    protected function _getTitle($title)
    {
        return ucwords($title);
    }
}
```

O acessador seria executado ao obter a propriedade através de qualquer uma dessas duas formas:

```
echo $user->title;
echo $user->get('title');
```

Você pode personalizar como as propriedades são atribuidas definindo um mutador:

```
Cake\ORM\Entity::set($field = null, $value = null)
```

Exemplo de Entity:

```
<?php
namespace App\Model\Entity;

use Cake\Auth\DefaultPasswordHasher;
use Cake\ORM\Entity;

class User extends Entity
{

    // Make all fields mass assignable except for primary key field "id".
    protected $_accessible = [
        '*' => true,
        'id' => false
    ];

    // ...

    protected function _setPassword($password)
    {
        return (new DefaultPasswordHasher)->hash($password);
    }

    // ...
}
?>
```

12.2 – ORM

TableRegistry

TableRegsitry
https://book.cakephp.org/3.0/en/orm/table-objects.html

Retornando os nomes dos clientes no src/Controller/ClientesController.php

O bake já faz isso, mas é importante saber fazer isso manualmente:

Editar o controller Clientes e no action index() adicionar ao final

```
$clientes = TableRegistry::get('Clientes');
$query = $clientes->find();
foreach ($query as $row) {
    echo '<h3>'.$row->nome.'</h3>';
}
```

Adicionar ao início do arquivo:

```
use Cake\ORM\TableRegistry;
```

Chame o aplicativo pelo navegador e veja os nomes dos clientes acima
Também podemos retornar qualquer um dos demais campos.

```
$clientes = TableRegistry::get('Clientes');
$query = $clientes->find();

echo '<h3>ID Nome E-mail Usuário</h3>';
foreach ($query as $row) {
    echo '<h4>'.$row->id.'-'.$row->nome.'-'.$row->email.'-'.$row->user_id.'</h4>';
}
```

Num controller

```
$clientes = TableRegistry::get('Clientes');

$query = $clientes
    ->find()
    ->select(['id', 'nome'])
    ->where(['id !=' => 1])
    ->order(['email' => 'DESC']);

foreach ($query as $cliente) {
    debug($cliente->nome);
}
```

```
$resultsIteratorObject = $clientes
    ->find()
    ->where(['id >' => 1])
    ->all();

foreach ($resultsIteratorObject as $cliente) {
    debug($cliente->id);
}

$resultsArray = $clientes
    ->find()
    ->where(['id >' => 1])
    ->toList();

foreach ($resultsArray as $cliente) {
    debug($cliente->id);
}

debug($resultsArray[0]->nome);

$cliente = $clientes
    ->find()
    ->where(['id' => 1])
    ->first();

debug($cliente->nome);

// Use the combine() method from the collections library
// This is equivalent to find('list')
$keyValueList = $clientes->find()->combine('id', 'title');

// An advanced example
$results = $clientes->find()
    ->where(['id >' => 1])
    ->order(['nome' => 'DESC'])
    ->map(function ($row) { // map() is a collection method, it executes the query
        $row->trimmedNome = trim($row->nome);
        return $row;
    })
    ->combine('id', 'trimmedNome') // combine() is another collection method
    ->toArray(); // Also a collections library method

foreach ($results as $id => $trimmedNome) {
    echo "$id : $trimmedNome";
}


$query = $clientes->find();
$query->select(['id', 'nome', 'email']);
foreach ($query as $row) {
```

```
    debug($row->nome);
}


// Results in SELECT id AS pk, title AS aliased_title, body ...
$query = $clientes->find();
$query->select(['pk' => 'id', 'aliased_nome' => 'nome', 'email']);


// Results in SELECT DISTINCT country FROM ...
$query = $articles->find();
$query->select(['country'])
    ->distinct(['country']);
```

Limite e página

```
// Fetch rows 50 to 100
$query = $articles->find()
    ->limit(50)
    ->page(2);

$total = $clientes->find()->where(['nome' => 'Ribamar FS2'])->count();
debug($total);

$query = $articles->find();
$query->select(['Articles.id', $query->func()->count('Comments.id')])
    ->matching('Comments')
    ->group(['Articles.id']);
$total = $query->count();
```

Inserir registro

```
$clientes = TableRegistry::get('Clientes');

$query = $clientes->query();
$query->insert(['nome', 'email'])
    ->values([
        'nome' => 'Elias Ferreira',
        'email' => 'meu@amigo.com'
    ])
    ->execute();


Vários
$query = $articles->query();
$query->insert(['title', 'body'])
    ->values([
        'title' => 'First post',
```

```
        'body' => 'Some body text'
    ])
    ->values([
        'title' => 'Second post',
        'body' => 'Another body text'
    ])
    ->execute();
```

Unindo select com insert

```
$select = $articles->find()
    ->select(['title', 'body', 'published'])
    ->where(['id' => 3]);

$query = $articles->query()
    ->insert(['title', 'body', 'published'])
    ->values($select)
    ->execute();
```

Update

```
$id=2;
$query = $clientes->query();
$query->update()
    ->set(['email' => 'teste@tes.com'])
    ->where(['id' => $id])
    ->execute();
```

Delete

```
$query = $articles->query();
$query->delete()
    ->where(['id' => $id])
    ->execute();
```

Cake 3.6

```
// Prior to 3.6.0 use association() instead.
$matchingComment = $articles->getAssociation('Comments')->find()
    ->select(['article_id'])
    ->distinct()
    ->where(['comment LIKE' => '%CakePHP%']);

$query = $articles->find()
    ->where(['id IN' => $matchingComment]);
```

Query Builder

https://book.cakephp.org/3.0/pt/orm/query-builder.html

```
use Cake\ORM\TableRegistry;

$clientes = TableRegistry::get('Clientes');

$query = $clientes
    ->find()
    ->select(['id', 'nome'])
    ->where(['id !=' => 1])
    ->order(['email' => 'DESC']);

foreach ($query as $cliente) {
    debug($cliente->nome);
}

debug($clientes->find()->where(['id' => 1]));
```

O Query Builder do CakePHP funciona basicamente em dois passos. O primeiro passo você monta a query, no segundo passo você executa a query e ela pode lhe retornar um objeto da sua entity ou uma collection.

No exemplo que você apresentou, você está executando apenas o primeiro passo, que é montar a query. Quando você dá um echo na sua view, ele está excutando o método toString do Query Builder, e não executando a query que ele montou.

Para executar a query e retorna um objeto com os dados do seu banco de dados, você pode usar um dos seguintes métodos: all(), toArray(), first(), extract() e para métodos que façam insert, delete ou update o método execute(). Recomendo que teste-os para ver o funcionamento e seu comportamento.

No seu caso, você poderia fazer da seguinte maneira:

```
use Cake\ORM\TableRegistry;

public function index()
{
   $contratacoes = TableRegistry::get('Contratacoes');
   $query = $contratacoes->find();
   $soma_financiamento = $query
      ->select(['sum' =>$query->func()->sum('vlr_financiamento')])
      ->first();

   // Descomente a linha abaixo para ver o resultado da query
   // debug($soma_financiamento);

   $this->set(['soma_financiamento' => $soma_financiamento->sum]);
}
```

Existe outras maneiras de fazer isso. O Query Builder armazena internamente uma colletion com os dados da sua query. Ela é populada quando é chamado um método da colletion com o estado atual do seu Query Builder.

Sendo assim outra possível solução é:

```
use Cake\ORM\TableRegistry;

public function index()
{
   $contratacoes = TableRegistry::get('Contratacoes');
   $query = $contratacoes->find();

   $this->set(['soma_financiamento' => $query->sumOf('vlr_financiamento')]);
}
```

Referencias e leitura recomendada da documentaçãodo CakePHP:

https://book.cakephp.org/3.0/en/orm/query-builder.html
https://book.cakephp.org/3.0/en/core-libraries/collections.html#aggregation

Retornando Dados

https://book.cakephp.org/3.0/pt/orm/retrieving-data-and-resultsets.html

Num controler

```
use Cake\ORM\TableRegistry;

$clientes = TableRegistry::get('Clientes');

$query = $clientes->find('all');
$results = $query->all();

foreach($results as $result){
debug($result->nome);
}


// No controller
$query = $clientes->find('all', [
    'conditions' => ['Clientes.nome >' => 'Ribamar FS2'],
    'contain' => ['Users'],
    'limit' => 10
]);
```

```
foreach($query as $q){
debug($q->nome);
}
```

Opções suportadas por find() são:

- conditions provê acesso direto na cláusula Where.
- limit Limite o número de resultados.
- offset Uma página que você quer. Use page para cálculo simplificado.
- contain defina uma associação para carregar.
- fields Quais campos você deseja carregar somente? Quando carregar somente alguns campos o lembre-se dos plugins, callbacks.
- group adicione um GROUP BY. muito usado para funçoes agregadas.
- having adicionar HVAING.
- join Defina um Join específico.
- order Ordenar resultados por.

Retornar o primeiro

```
// No controller
$query = $clientes->find('all', [
    'order' => ['Clientes.nome' => 'DESC']
]);
$row = $query->first();

//Ex: Retorne todos os artigos, mais quero somente o primeiro.

foreach($query as $q){
debug($q->nome);
}
```

Contar registros

```
// No controller ou table.
$query = $clientes->find('all', [
    'conditions' => ['Clientes.nome LIKE' => '%Riba%']
]);
$number = $query->count();
print_r($number);

$clientes = TableRegistry::get('Clientes');

// Select id & title from articles, but all fields off of Users, Comments and Tags.
$query = $clientes->find();
$query->select(['id', 'nome'])
    ->contain(['Users'])
    ->enableAutoFields(true)
    ->contain(['Users' => function($q) {
```

```
        return $q->enableAutoFields(true);
    }]);
foreach($query as $q){
debug($q->nome);
}
```

Ordenando

```
$clientes = TableRegistry::get('Clientes');

$query = $clientes->find();
$query->contain([
    'Users' => [
        'sort' => ['Users.username' => 'DESC']
    ]
]);

foreach($query as $q){
debug($q->nome);
}
```

Recebendo o primeiro e o último registros

```
$clientes = TableRegistry::get('Clientes');

$result = $clientes->find('all')->all();

// Get the first and/or last result.
$fir = $result->first();
$las = $result->last();

print 'Primeiro registro: '.$fir->nome;
print '<br>Último registro: '.$las->nome;
```

Salvando Dados

https://book.cakephp.org/3.0/pt/orm/saving-data.html

Depois que carregamos os dados é hora de atualizar e salvar as alterações.

Inserindo dados

A maneira mais fácil de inserir dados no banco de dados é criando uma nova entidade e passando ela pro método save() na classe Table:

```
use Cake\ORM\TableRegistry;

$clientesTable = TableRegistry::get('clientes');
$cliente = $clientesTable->newEntity();
```

```
$cliente->nome = 'Fátima EF';
$cliente->email = 'fatima@gmail.com';

if ($clientesTable->save($cliente)) {
    // The $cliente entity contains the id now
    $id = $cliente->id;
}
```

Update

```
use Cake\ORM\TableRegistry;

$clientesTable = TableRegistry::get('Clientes');
$cliente = $clientesTable->get(2); // Return cliente with id 12

$cliente->nome = 'Este era o Tiago';
$clientesTable->save($cliente);
```

Salvando com Associações

Por padrão o método save() também salvará associações de um nível:

```
$clientes = TableRegistry::get('Clientes');

$clientesTable = TableRegistry::get('clientes');
$user = $clientesTable->Users->findByUsername('super')->first();

$cliente = $clientesTable->newEntity();
$cliente->nome = 'Um cliente';
$cliente->email = 'novo@novo.com';
$cliente->user_id = $user;

if ($clientesTable->save($cliente)) {
    // A chave estrangeira foi atribuída automaticamente
    echo $cliente->user_id;
}
```

O método save() também é capaz de criar novos registros para associações:

```
$firstComment = $clientesTable->Comments->newEntity();
$firstComment->body = 'The CakePHP features are outstanding';

$secondComment = $clientesTable->Comments->newEntity();
$secondComment->body = 'CakePHP performance is terrific!';
```

```
$tag1 = $clientesTable->Tags->findByName('cakephp')->first();
$tag2 = $clientesTable->Tags->newEntity();
$tag2->name = 'awesome';

$cliente = $clientesTable->get(12);
$cliente->comments = [$firstComment, $secondComment];
$cliente->tags = [$tag1, $tag2];

$clientesTable->save($cliente);
```

Excluindo Registros

https://book.cakephp.org/3.0/pt/orm/deleting-data.html

class Cake\ORM\Table

Cake\ORM\Table::delete(Entity $entity, $options = [])

Depois que você carregou uma entidade, você pode excluir ela chamando o o método delete da tabela de origem:

```
// Num a controller.
$entity = $this->Articles->get(2);
$result = $this->Articles->delete($entity);
```

Ao excluir entidades algumas coisas acontecem:

As delete rules serão aplicadas. Se as regras falharem, a exclusão será impedida.
O evento Model.beforeDelete é disparado. Se esse evento for interrompido, a exclusão será cancelada e o resultado do evento será retornado.
A entidade será excluída.
Todas as associações dependentes serão excluídas. Se as associações estão sendo excluídas como entidades, eventos adicionais serão disparados.
Qualquer registro da tabela de ligação para associação BelongsToMany serão removidos.
O evento Model.afterDelete será disparado.

Por padrão, todas as exclusões acontecem dentro de uma transação. Você pode desativar a transação com a opção atomic:

```
$result = $this->Articles->delete($entity, ['atomic' => false]);
```

Exclusão em Cascata

Ao excluir entidades, os dados associados também podem ser excluídos. Se suas associações HasOne e HasMany estão configurados como dependent, as operações de exclusão serão ‘cascate’ para essas entidades também. Por padrão entidades em tabelas

associadas são removidas usando Cake\ORM\Table::deleteAll(). Você pode optar que o ORM carregue as entidades relacionadas, para então excluir individualmente, definindo a opção cascadeCallbacks como true:

```
// No método initialize de alguma modelo Table
$this->hasMany('Comments', [
    'dependent' => true,
    'cascadeCallbacks' => true,
]);
```

Configurando cascadeCallbacks para true, resulta em exclusões consideravelmente mais lentos quando comparado com exclusão em masa. A opção cascadeCallbacks apenas deve ser ativada quando sua aplicação tem trabalho importante manipulado por event listeners.

Exclusão em Massa

Cake\ORM\Table::deleteAll($conditions)

Pode ter momentos em que excluir linhas individualmente não é eficiente ou útil. Nesses casos, é mais eficiente usar uma exclusão em massa para remover várias linhas de uma vez só:

```
// Exclui todos oss spam
function destroySpam()
{
    return $this->deleteAll(['is_spam' => true]);
}
```

Uma exclusão em massa será considerada bem-sucedida se uma ou mais linhas forem excluídas.

deleteAll não dispara os eventos beforeDelete/afterDelete. Se você precisa deles, você precisa, primeiro carregar uma coleção de registros e então excluí-las.

Dicas sobre o CakePHP 3

```
use Cake\Datasource\ConnectionManager;

$connection = ConnectionManager::get('default');
$results = $connection->execute('SELECT * FROM articles')->fetchAll('assoc');

ou
$results = $connection->execute('SELECT * FROM articles WHERE id = :id', ['id' => 1])-
>fetchAll('assoc');
```

Insert

```
use Cake\Datasource\ConnectionManager;

$connection = ConnectionManager::get('default');
$connection->insert('articles', [
    'title' => 'A New Article',
    'created' => new DateTime('now')
], ['created' => 'datetime']);
```

Update

```
use Cake\Datasource\ConnectionManager;
$connection = ConnectionManager::get('default');
$connection->update('articles', ['title' => 'New title'], ['id' => 10]);
```

Delete

```
use Cake\Datasource\ConnectionManager;
$connection = ConnectionManager::get('default');
$connection->delete('articles', ['id' => 10]);
```

```
$stmt = $conn->query('UPDATE posts SET published = 1 WHERE id = 2');
```

The query() method does not allow for additional parameters. If you need additional parameters you should use the execute() method, which allows for placeholders to be used:

```
$stmt = $conn->execute(
    'UPDATE posts SET published = ? WHERE id = ?',
    [1, 2]
);

$stmt = $conn->execute(
    'UPDATE posts SET published_date = ? WHERE id = ?',
    [new DateTime('now'), 2],
    ['date', 'integer']
);

$stmt->execute();

// Read one row.
$row = $stmt->fetch('assoc');

// Read all rows.
$rows = $stmt->fetchAll('assoc');
```

```
// Read rows through iteration.
foreach ($rows as $row) {
    // Do work
}

$rowCount = count($stmt);
$rowCount = $stmt->rowCount();
```

Insert

```
$query = $articles->query();
$query->insert(['title', 'body'])
    ->values([
        'title' => 'First post',
        'body' => 'Some body text'
    ])
    ->execute();

$select = $articles->find()
    ->select(['title', 'body', 'published'])
    ->where(['id' => 3]);

$query = $articles->query()
    ->insert(['title', 'body', 'published'])
    ->values($select)
    ->execute()
```

Update

```
$query = $articles->query();
$query->update()
    ->set(['published' => true])
    ->where(['id' => $id])
    ->execute();
```

Delete

```
$query = $articles->query();
$query->delete()
    ->where(['id' => $id])
    ->execute();
```

Outras

```
$matchingComment = $articles->association('Comments')->find()
    ->select(['article_id'])
    ->distinct()
    ->where(['comment LIKE' => '%CakePHP%']);
```

```
$query = $articles->find()
    ->where(['id' => $matchingComment]);
```

Criar uma nova conexão de banco de dados
https://book.cakephp.org/3.0/pt/orm/database-basics.html#gerenciando-conexoes

Configuring Connections

By default all table instances use the default database connection. If your application uses multiple database connections you will want to configure which tables use which connections. This is the defaultConnectionName() method:

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public static function defaultConnectionName() {
        return 'replica_db';
    }
}
```

Usando TableRegistry para a conexão

Configuring Table Objects

static Cake\ORM\TableRegistry::get($alias, $config)

When loading tables from the registry you can customize their dependencies, or use mock objects by providing an $options array:

```
$articles = TableRegistry::get('Articles', [
    'className' => 'App\Custom\ArticlesTable',
    'table' => 'my_articles',
    'connection' => $connectionObject,
    'schema' => $schemaObject,
    'entityClass' => 'Custom\EntityClass',
    'eventManager' => $eventManager,
    'behaviors' => $behaviorRegistry
]);
```

Suponha que queremos conectar a um outro banco de dados diferente do atual
Podemos criar outras conexões usando o arquivo config/app.php (veja ao final)
Também via código como abaixo, podemos criar quantas conexões com outros bancos quizermos

Vamos fazer isso no model clientes

Adicionar ao início do arquivo
use Cake\Datasource\ConnectionManager;

Inserir no intialize()

```
$dsn = 'mysql://root:password@localhost/my_database';
ConnectionManager::config('outro', ['url' => $dsn]);
```

Ao invés de config() agora do cake 3.6 usa-se setConfig()

Driver - mysql
User - root
Senha - root
Banco - crud

```
$dsn = 'mysql://root:root@localhost/crud';
ConnectionManager::config('outro', ['url' => $dsn]);

$connection = ConnectionManager::get('outro');
```

Agora vamos retornar algo do outro banco

```
$results = $connection->execute('SELECT * FROM clientes')->fetchAll('assoc');
debug($results);exit;
```

Podemos consultar também com:

```
$results = $connection
    ->execute('SELECT * FROM clientes WHERE id = :id', ['id' => 1])
    ->fetchAll('assoc');
```

Com query builder

```
$results = $connection
    ->newQuery()
    ->select('*')
    ->from('clientes')
    ->where(['email' => 'ribafs@gmail.com'])
    ->order(['nome' => 'DESC'])
    ->execute()
    ->fetchAll('assoc');
```

Inserindo registros

```
$connection->insert('clientes', [
    'nome' => 'Tiago EF',
    'email' => 'tiago@tiago.com']);
```

Update

```
$connection->update('clientes', ['nome' => 'Tiago EF2'], ['id' => 2]);
```

Delete

```
$connection->delete('clientes', ['id' => 2]);
```

```
use Cake\Datasource\ConnectionManager;

ConnectionManager::config('terceiro', [
    'className' => 'Cake\Database\Connection',
    'driver' => 'Cake\Database\Driver\Mysql',
    'persistent' => false,
    'host' => 'localhost',
    'username' => 'root',
    'password' => 'root',
    'database' => 'cadastro',
    'encoding' => 'utf8',
    'timezone' => 'UTC',
    'cacheMetadata' => true,
]);
```

Usando o config/app.php

```
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            //'port' => 'non_standard_port_number',
            'username' => 'root',
            'password' => 'root',
            'database' => 'cake36_code',
            'timezone' => 'UTC',
            'flags' => [],
            'cacheMetadata' => true,
            'log' => false,

            'quoteIdentifiers' => false,

            'url' => env('DATABASE_URL', null),
        ],

        'segundo' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
```

```
            'host' => 'localhost',
            //'port' => 'non_standard_port_number',
            'username' => 'root',
            'password' => 'root',
            'database' => 'crude',
            'timezone' => 'UTC',
            'flags' => [],
            'cacheMetadata' => true,
            'log' => false,

            'quoteIdentifiers' => false,

            'url' => env('DATABASE_URL', null),
        ],

        /**
         * The test connection is used during the test suite.
         */
        'test' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            //'port' => 'non_standard_port_number',
            'username' => 'my_app',
            'password' => 'secret',
            'database' => 'test_myapp',
            //'encoding' => 'utf8mb4',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
            'quoteIdentifiers' => false,
            'log' => false,
            //'init' => ['SET GLOBAL innodb_stats_on_metadata = 0'],
            'url' => env('DATABASE_TEST_URL', null),
        ],
    ],
```

Executando consultas

Cake\Database\Connection::query($sql)

```
$stmt = $conn->query('UPDATE articles SET published = 1 WHERE id = 2');

$stmt = $conn->execute(
    'UPDATE articles SET published = ? WHERE id = ?',
    [1, 2]
);

$stmt = $conn->execute(
    'UPDATE articles SET published_date = ? WHERE id = ?',
```

```
    [new DateTime('now'), 2],
    ['date', 'integer']
);

Cake\Database\Connection::newQuery()

$query = $conn->newQuery();
$query->update('articles')
    ->set(['published' => true])
    ->where(['id' => 2]);
$stmt = $query->execute();

$query = $conn->newQuery();
$query->select('*')
    ->from('articles')
    ->where(['published' => true]);

foreach ($query as $row) {
    // Faz alguma coisa com a linha.
}
```

Transações

```
$conn->begin();
$conn->execute('UPDATE articles SET published = ? WHERE id = ?', [true, 2]);
$conn->execute('UPDATE articles SET published = ? WHERE id = ?', [false, 4]);
$conn->commit();

Cake\Database\Connection::transactional(callable $callback)

$conn->transactional(function ($conn) {
    $conn->execute('UPDATE articles SET published = ? WHERE id = ?', [true, 2]);
    $conn->execute('UPDATE articles SET published = ? WHERE id = ?', [false, 4]);
});
```

Contagem de registros

```
$rowCount = count($stmt);
$rowCount = $stmt->rowCount();
```

Código e mensagem de erro

```
$code = $stmt->errorCode();
$info = $stmt->errorInfo();
```

Quoting

```
$conn->driver()->autoQuoting(true);
```

Criando banco de dados

```
$dsn = 'mysql://root:password@localhost/';

$connection->query("CREATE DATABASE IF NOT EXISTS my_database");
```

13 – View

Documentação oficial
http://book.cakephp.org/3.0/en/views.html

As Views representam o V de MVC. Elas são responsáveis por gerar a saída específica para a requisição do usuário. Geralmente a saída é em HTML, mas também pode ser em XML, JSON, PDF e outras.

O CakePHP conta com uma grande quantidade de classes embutidas para manipulação o mais comum dos cenários a renderizar:
- Para criar webservers XML ou JSON que podemos usar nas views
- Para servir arquivos protegidos, ou arquivos gerados dinamicamente
- Para criar múltiplas views com templates

A AppView

A classe AppView é a classe View default. Em si ela estende a classe View e é definida em src\View\AppView.php assim:

```
<?php
namespace App\View;

use Cake\View\View;

class AppView extends View
{
}
```

Podemos usar AppView para carregar Helper (no método initialize()), mas geralmente os helpers criados e adicionados a src\View\Helper já ficam automaticamente disponíveis para as views dos templates.

Views Template

A camada de views do CakePHP é a forma de nos comunicarmos com o usuário da aplicação.

A extensão default das views é .ctp (**C**ake**P**HP **T**emplate) e utiliza uma sintaxe alternativa em PHP para estruturas de controle e saída. Estes arquivos contem a lógica necessária para preparar os dados recebidos do controller no formato de apresentação que está pronto para a nossa audiência.

Alternativa ao echo

Mostrando uma variável com echo ou print:
<?php echo $variavel; ?>

Usando a tag curta:
<?= $variavel ?>

Alternativa de Estrutura de Controle

Estruturas de controle como if, for, foreach, switch e while podem ser escritas de forma simplificada, sem chaves, ao invés disso dois pontos e end; ao final, assim:

foreach ($todo as $item):
    <?= $item ?>
endforeach;
Nas views geralmente aparece assim:

<?php foreah ($todo as $item): ?>
    <?= $item ?>
<?php endforeach; ?>

Outro exemplo, veja o ; no endif;

```
<?php if ($username === 'sally'): ?>
   <h3>Hi Sally</h3>
<?php elseif ($username === 'joe'): ?>
   <h3>Hi Joe</h3>
<?php else: ?>
   <h3>Hi unknown user</h3>
<?php endif; ?>
```

Arquivos de templates são armazenados em:
src\Template

Nomeado após os controllers que usam os arquivos e após os actions correspondentes. Por exemplo, o arquivo de view para o action view() do controller ProdutosController chama-se
src\Template\Produtos\view.ctp

Partes de uma View

A camada View no CakePHP pode ser formada por algumas partes. Cada parte tem diferentes usos e será coberta neste tutorial:

- **views**: Templates são a parte da página que é única para o action começar a rodar; Elas são a parte principal da resposta do aplicativo.

- **elements**: pequenos, reusáveis pedaços de código da view. Elements geralmente são renderizados dentro da view. Documentação oficial:
http://book.cakephp.org/3.0/en/views.html#elements

- **layouts**: arquivos de templates que contém código de apresentação que envolve muitas interfaces na sua aplicação. Muitas views são renderizadas dentro de um layout.
http://book.cakephp.org/3.0/en/views.html#layouts

- **helpers**: estas classes encapsulam lógica de views que são necessárias em muitos lugares na camada view. Entre outras coisas os helpers no CakePHP podem ajudar você a construir forms, criar funcionalidade AJAX, paginar modelo de dados, etc. http://book.cakephp.org/3.0/en/views/helpers.html

- **cells**: estas classes oferecem uma miniatura tipo as características dos controllers para a criação de componentes UI auto contidos. Veja a documentação para mais informações: http://book.cakephp.org/3.0/en/views/cells.html

Variáveis das Views

Qualquer variável setada nos controllers com o método set() deve estar disponível nas views e layout que o action renderizou. Como também as variaveis setadas com o método set() estarão disponíveis nos elements. Caso precise passar variáveis adicionais de uma view para um layout você pode chamar o método set() na view do template ou usar um bloco – http://book.cakephp.org/3.0/en/views.html#view-blocks.

Devemos lembrar de sempre "escapar" qualquer dado recebido de usuário antes de exibir de volta, pois o CakePHP não usa escapes automaticamente. Podemos "escapar" com a função h():
<?= h($user->bio) ?>

Configurando Variáveis de Views

As Views têm um método set() que é análogo ao set() encontrado nos objetos Controllers. Usando set() nos arquivos de view podemos adicionar as variáveis para o layout e elements que devem ser renderizados depois. Veja para mais detalhes ao usar set(): http://book.cakephp.org/3.0/en/controllers.html#setting-view-variables

Nos seus arquivos de view você pode fazer:
```
$this->set('activeMenuButton', 'posts');
```

Então no seu leyout, a variável $activeMenuButton deve estar disponível e conterá 'posts'.

Dicas sobre Views

Título do Aplicativo

- Adicionar ao AppController.php:

```
    function beforeFilter(){
        $this->set('title_for_layout','Controle de Finanças Pessoais');
    }
```

- Será usado no Layout/default.ctp, aqui assim:

```
    <title>
        <?php echo $title_for_layout .' - '. ucfirst($this->params['controller']).'/'.$this-
>params['action']; ?>
    </title>
```

Isso mostrará no title do navegador a variável $title_for_layout definida no AppController mais o controller e o action atuais.

Melhorando Tradução

Melhorar tradução de botão na view edit:

```
<li><?php echo $this->Form->postLink(__('Delete').(' Despesa'), array('action' => 'delete', $this->Form->value('Despesa.id')), null, __('Tem certeza de que deseja excluir # %s?', $this->Form->value('Despesa.id'))); ?></li>
```

Adicionei: .(' Despesa')

E mais abaixo na div header do layout:

```
<div id="header">
    <h3><?php echo '  '.$title_for_layout .' - '. ucfirst($this->params['controller']).'/'.$this->params['action'];?></h3>
    <?php echo $this->element('menutopo2');?>
</div>
```

Alteração no CSS

Copiei o app/webroot/css/cake.generic.css para app/webroot/css/cake.financas.css
Editei e alterei a cor do H#:

```
h3 {
    color: #B2F7EC;
    font-family:'Gill Sans','lucida grande', helvetica, arial, sans-serif;
    font-size: 165%;
}
```

Alterei o View/Layout/default.ctp, mudando o css padrão para o financas, que criei:

```
echo $this->Html->css('cake.financas');
```

Trocando campo do tipo text por select:

```
$options = array(''=>'Selecione','colegioelias' => 'Colégio do Elias', 'eliasmesada'=>'Elias Mesada','eliasmerenda'=>'Elias Merenda','meire' => 'Meire Salário', 'coelce'=>'COELCE', 'condominio'=>'Condomínio Ferreira', 'gvt'=>'GVT', 'carnes'=>'Carnes','mercantil'=>'Mercantil','almoco'=>'Almoço Trabalho','passagens'=>'Passagens','remedio'=>'Remédio');

        echo $this->Form->input('descricao', array('type'=>'select','label' => 'Descrição da Despesa','options' => $options,'default'=>'0'));
```

13.1 – Element

`Cake\View\View::element`(*string $elementPath*, *array $data*, *array $options* =[])

Alguns aplicativos tem pequenos blocos de código de apresentação que se repetem de página para página, algumas vezes em diferentes lugares do layout. O CakePHP pode ajudar você a repetir partes do seu site que precisam ser reutilizadas. Estas partes reusáveis são chamadas de Elements. Ads, help boxes, navigational controls, extra menus, login forms e callouts são implementados em CakePHP como elements.

Um element é basicamente uma mini-view que pode ser incluída em outros pontos da view, em layouts, e até mesmo dentro de outros elements. Os elements podem ser usados para fazer uma view mais legível, colocando a prestação de elements repetidos em seu próprio arquivo. Eles também podem ajudá-lo a reutilizar fragmentos de conteúdo em sua aplicação.

Elements ficam na pasta
src\Template\Element

E tem extensão .ctp

Mostrando com o método element da view:

```
echo $this->element('helpbox');
```

Passando Variáveis em Elementos

Podemos passar dados para um elemento através do segundo argumento do método element():

```
echo $this->element('helpbox', [
    "helptext" => "Oh, this text is very helpful."
]);
```

Dentro do elemento podemos usar a variável $helptext de forma semelhante ao método set() dos controllers:

```
// Inside src/Template/Element/helpbox.ctp
echo $helptext; // Outputs "Oh, this text is very helpful."
```

O método element() também suporta options para o element.

```
echo $this->element('helpbox', [
        "helptext" => "This is passed to the element as $helptext",
        "foobar" => "This is passed to the element as $foobar",
    ],
    [
        // uses the "long_view" cache configuration
        "cache" => "long_view",
        // set to true to have before/afterRender called for the element
        "callbacks" => true
    ]
);
```

Criando seu Próprio Element

Criaremos um pequeno Element que adicionará um menu para a região <nav> do layout.

Criar o arquivo

src\Template\Element\topmenu.ctp

Contendo:

```
<?php
    $loggedno = 'Não Logado';
    if(!$loguser) $loguser=$loggedno;

    if($loguser == 'admin'){
        echo $this->Html->link(__('Clientes'),
array('plugin'=>null,'controller'=>'Clientes','action'=>'index'));
        echo ' '.$this->Html->link(__('Funcionários'),
array('plugin'=>null,'controller'=>'Funcionarios','action'=>'index'));
        echo ' '.$this->Html->link(__('Grupos'),
array('plugin'=>null,'controller'=>'Groups','action'=>'index'));
        echo ' '.$this->Html->link(__('Usuários'),
array('plugin'=>null,'controller'=>'Users','action'=>'index'));
        echo ' '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'));
    }elseif($loguser == 'manager'){
        echo $this->Html->link(__('Clientes'),
array('plugin'=>null,'controller'=>'Clientes','action'=>'index'));
        echo ' '.$this->Html->link(__('Funcionários'),
array('plugin'=>null,'controller'=>'Funcionarios','action'=>'index'));
        echo ' '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'));
    }elseif($loguser == 'user'){
        echo $this->Html->link(__('Clientes'),
array('plugin'=>null,'controller'=>'Clientes','action'=>'index'));
        echo ' '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'));
        /* Modelo
        echo $this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout'),['class' => 'button', 'target' =>
'_blank']); */
    }
    echo '   Logado como: '. $loguser;
```

Usando

Para usar adicione a linha no src/Template/Layout/default.ctp, como indicado abaixo:

```
<body>
  <nav class="top-bar expanded" data-topbar role="navigation">
        <?php echo $this->element('topmenu');?>
```

Adicionar ao AppController

E adicione estas 3 linhas ao src/Controller/AppController.php no método initialize(), abaixo de parent::initialize();:

```
    public function initialize()
    {
        parent::initialize();

            $loguser = $this->request->session()->read('Auth.User');
            $loguser = $loguser['username'];
            $this->set('loguser',$loguser);
```

Dicas sobre Element no Cake

Muitas aplicações possuem pequenos blocos de código de apresentação que precisam ser repetidos a cada página, às vezes em diferentes lugares no layout. O CakePHP ajuda você a repetir partes do seu website que precisam ser reutilizados. Estas partes reutilizáveis são chamadas de Elements (ou Elementos). Propagandas, caixas de ajuda, controles de navegação, menus extras, formulários de login e chamadas geralmente são implementadas como elements. Um element é básicamente uma mini-view que pode ser incluída em outras views, layouts e até mesmo em outros elements. Elements podem ser usados para criar uma view mais legível, colocando o processamento de elementos repetidos em seu próprio arquivo. Eles também podem ajudá-lo a re-usar conteúdos fragmentados pela sua aplicação.

Element - são pequenos trechos de código que podem ser usados nas views para qualquer utilidade.

Um element pode ser acessado por qualquer view.

Diretório - /app/View/Elements/

No local da view onde queremos que apareça o código do elemento inserimos:
echo $this->element('espacos');

Podemos passar informações para um element através do seu segundo parâmetro:

```
echo $this->element('helpbox', array(
    "textoajuda" => "Oh, this text is very helpful."
));
echo $textoajuda;
```

Criando um element simples e útil:

Criar o arquivo:
/app/View/Elements/menutopo.ctp

```
<ul class="actions">
```

```
    <?php
        $valor = $this->Session->read('Auth.User');
        $usuario=$valor['username'];
        $grupo=$valor['group_id'];

    echo $this->Html->link('Produtos', '/submenus/menus');
    if($grupo==1){
        echo $this->Html->link('Administração', '/submenus/admin');
    }

    if($usuario=='Não logado!'){
                echo $this->Html->link(__('Login'),
array('controller'=>'users','action'=>'login'));
    }else{
                echo $this->Html->link(__('Logout'),
array('controller'=>'users','action'=>'logout'));
    }
        if(!$usuario) $usuario='Não logado!';
        echo '   Logado como: '. $usuario;

    ?>
</ul>
```

Chamar numa view qualquer com:
echo $this->element('menutopo');

https://book.cakephp.org/3.0/en/views.html#elements

Exemplos

Espaços em Branco

```
<?php
// Element para mostrar 25 espaços em branco um ao lado do outro
    for($x=0;$x<25;$x++){
        print " ";
    }
?>
```

Menu

```
<ul>
    <h3>Menu Principal</h3>
    <?php
    echo $this->Menu->item($this->Html->link('Clientes', array('controller' => 'clientes',
'action' => 'index')));
    echo $this->Menu->item($this->Html->link('Funcionários', array('controller' =>
'funcionarios', 'action' => 'index')));
    echo $this->Menu->item($this->Html->link('Example', '/example/add'));
```

```
    echo $this->Menu->item($this->Html->link('Example', array('controller' => 'example',
'action' => 'index')));
    echo $this->Menu->item($this->Html->link('Example', '/examplealternate'));
    echo $this->Menu->item($this->Html->link('About', '/about'), array('class' =>
'specialClass'));
    echo $this->Menu->item($this->Html->link('Contact', array('controller' => 'contact',
'action' => 'index')));
    ?>
</ul>
```

Criando um Element topmenu2

Adicionar ao método initialize do AppController.php

```
      $loguser = $this->request->session()->read('Auth.User');
      $loguser = $loguser['username'];
      $this->set('loguser',$loguser);
```

= Código do element contendo:
src\Template\Element\topmenu2.ctp

```
<?php

    if($loguser == 'admin'){
        $controllers = array('Clientes','Funcionarios','Produtos','Pedidos','Groups','Users');

        foreach($controllers as $controller){
            echo '<div class="button">'.$this->Html->link(__($controller),
array('plugin'=>null,'controller'=>$controller,'action'=>'index')).'</div>';
        }
    }elseif($loguser == 'manager'){
        $controllers = array('Clientes','Funcionarios','Produtos','Pedidos');

        foreach($controllers as $controller){
            echo '<div class="button">'.$this->Html->link(__($controller),
array('plugin'=>null,'controller'=>$controller,'action'=>'index')).'</div>';
        }
    }elseif($loguser == 'user'){
        $controllers = array('Clientes');

        foreach($controllers as $controller){
            echo '<div class="button">'.$this->Html->link(__($controller),
array('plugin'=>null,'controller'=>$controller,'action'=>'index')).'</div>';
        }
    }
    echo '<div class="button"> '.$this->Html->link(__('Sair'),
array('plugin'=>null,'controller'=>'Users','action'=>'logout')).'</div>';
```

```
      echo '<div align="right" class="logado">   Logado como: '.
$loguser.'</div>';
```

= Editar o webroot\css\cake.css e adicionar ao início do arquivo:

```
a:hover {
    color: #000000; /* Remova ou comente o a:hover existente */
}

.button{
      font-weight:normal;
      padding: 4px 8px;
      background: #dcdcdc;
      background-image: -moz-linear-gradient(top, #fefefe, #dcdcdc);
      color:#333;
      border:1px solid #bbb;
      border-radius: 4px;
      text-decoration: none;
      text-shadow: #fff 0px 1px 0px;
      min-width: 0;
      box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.3), 0px 1px 1px rgba(0, 0, 0, 0.2);
}

.logado{
      font-weight:normal;
      float: right;
      padding: 4px 8px;
      background: #dcdcdc;
      background-image: -moz-linear-gradient(top, #fefefe, #dcdcdc);
      color:#333;
      border:1px solid #bbb;
      border-radius: 4px;
      text-decoration: none;
      text-shadow: #fff 0px 1px 0px;
      min-width: 0;
      box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.3), 0px 1px 1px rgba(0, 0, 0, 0.2);
      display: inline-block;
}
```

= Editar o src\Template\Layout\default.ctp Altere como segue:

```
            <div class="top-bar-section">
          <ul class="right">
             <li><b><?= $this->Html->link(__('Sistema de Cadastro de Produtos'), ['controller'
=> 'Produtos', 'action' => 'index']) ?></a></b></li>
          </ul>
       </div>
    </nav>
      <?php echo $this->element('topmenu');?>
```

13.2 – Layouts

Layout contém código de apresentação que é incorporado na view. Qualquer coisa que você deseje ver em todas as suas views deve ser colocada em um layout.

No CakePHP 3 o arquivo default de layout fica em
src\Template\Layout\default.ctp

Se você deseja mudar a aparência da sua aplicação (cores, fontes e posições, imagens, etc) então o layout é o lugar para mexer.

Outros arquivos de layout devem ser criados em
src\Template\Layout\

Aqui está o layout default.ctp do CakePKP 3.2.8:

```
<?php
// Comentários
$cakeDescription = 'CakePHP: the rapid development php framework';
?>
<!DOCTYPE html>
<html>
<head>
    <?= $this->Html->charset() ?>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
        <?= $cakeDescription ?>:
        <?= $this->fetch('title') ?>
    </title>
    <?= $this->Html->meta('icon') ?>

    <?= $this->Html->css('base.css') ?>
    <?= $this->Html->css('cake.css') ?>

    <?= $this->fetch('meta') ?>
    <?= $this->fetch('css') ?>
    <?= $this->fetch('script') ?>
</head>
<body>
    <nav class="top-bar expanded" data-topbar role="navigation">
        <ul class="title-area large-3 medium-4 columns">
            <li class="name">
                <h1><a href=""><?= $this->fetch('title') ?></a></h1>
            </li>
        </ul>
        <div class="top-bar-section">
            <ul class="right">
                <li><a target="_blank"
href="http://book.cakephp.org/3.0/">Documentation</a></li>
```

```
            <li><a target="_blank" href="http://api.cakephp.org/3.0/">API</a></li>
        </ul>
      </div>
    </nav>
    <?= $this->Flash->render() ?>
    <div class="container clearfix">
        <?= $this->fetch('content') ?>
    </div>
    <footer>
    </footer>
</body>
</html>
```

Podemos setar o bloco title dentro da view:

```
$this->assign('title', 'View Usuários Ativos');
```

Podemos criar tantos leyouts quantos desejarmos. Apenas os crie em: src\Template\Layout\

E chame no action respectivo com:

```
$this->viewBuilder()->layout('admin');
```

Na view use:

$this->layout = 'loggedin';

Exemplo de uso

Uma aplicação seria quando o site precisa de um banner no topo.

Para que esteja disponível para todos os controller então nosso AppController deve ficar com:

```
namespace App\Controller;

class UsersController extends AppController
{
    public function view_active()
    {
        $this->set('title', 'View Active Users');
        $this->viewBuilder()->layout('default_small_ad');
    }

    public function view_image()
    {
        $this->viewBuilder()->layout('image');
        // Output user image
    }
}
```

13.3 – Helper

Helpers no CakePHP 3

Os helpers contém código de apresentação que é compartilhado entre muitas views, elements ou layouts.

O Cake já traz alguns bem úteis Helpers:
- Flash
- Form
- Html
- Number
- Paginator
- Text
- Time
- Url

Carregamos Helpers no Cake declarando em classes views.

```
Class AppView extends View
{
    public function initialize()
    {
        parent::initialize();
        $this->loadHelper('Html');
        $this->loadHelper('Form');
        $this->loadHelper('Flash');
    }
}
```

Carregando Helpers em Plugins

$this->loadHelper('ContactManager.Comentario');

Podemos carregar helper condicionalmente

```
class AppView extends View
{
    public function initialize()
    {
        parent::initialize();
        if ($this->request->action === 'index') {
       $this->loadHelper('Message');
        }
    }
}
```

Usando o Helper

$this->Form->create($article);
Alguns helpers são carregados pelo Cake automaticamente, como é o caso do Form.

Também podemos usar o método beforeRender() dos controllers para carregar Helpers.

```
class ArticlesController extends AppController
{
    public function beforeRender(Event $event)
    {
        parent::beforeRender($event);
        $this->viewBuilder()->helpers(['MyHelper']);
    }
}
```

Criando Helpers

Podemos criar classes Helper para usar em aplicações ou em plugins.

Os Helpers têm algumas convenções que ajudam se seguirmos:
- Arquivos de classe Helper devem ficar em src/View/Helper. Exemplo:
    src/View/Helper/LinkHelper.php
- Classes helpers devem ser sufixadas com Helper. Exemplo:
    LinkHelper
- Quando referenciar classes helper deve omitir o sufixo Helper:
    $this->loadHelper('Link');

Criando o esqueleto de um Helper chamado Message com o bake:

cd c:\xampp\htdocs\aplicativo
bin\cake bake helper Message

Editar o src\View\Helper\MessageHelper.php e adicionar as funções, assim:

```
<?php
namespace App\View\Helper;

use Cake\View\Helper;
use Cake\View\View;

/**
 * Table helper
 */
class TableHelper extends Helper
{

    /**
     * Default configuration.
     *
     * @var array
     */
      protected $_defaultConfig = [];

      public function msg($msg)
      {
```

```
        return '<h2>'.$msg.'</h1>';
    }
}
```

Carregar na classe AppView

Editar src/View/AppView.php e adicionar o loadHelper()

```
class AppView extends View
{
    public function initialize()
    {
        parent::initialize();
        if ($this->request->action === 'index') {
       $this->loadHelper('Message');
        }
    }
}
```

```
Atente para o fato de que na classe acima ele restringe para que somente no
action index.ctp o helper apareça. Para mostrar em todas as actions use:
```

Observação – pelos meus testes nem precisa carregar na AppView e o helper ficará disponível automaticamente.

```
    public function initialize()
    {
       $this->loadHelper('Message');
    }
```

Usando o Helper
Editar a view index.ctp e adicionar:

<?=$this->Message->msg('Minha Mensagem Helper') ?>

Ao chamar no navegador verá a mensagem com título h2.
Um helper bem simples para mostrar o caminho das pedras.

Documentação oficial:
http://book.cakephp.org/3.0/en/views/helpers.html

Os helpers contém código de apresentação que é compartilhado entre muitas views, elements ou layouts.

Carregamos Helpers no Cake declarando em classes views.

```
Class AppView extends View
{
    public function initialize()
    {
        parent::initialize();
        $this->loadHelper('Html');
```

```
        $this->loadHelper('Form');
        $this->loadHelper('Flash');
    }
}
```

Carregando Helpers em Plugins

```
$this->loadHelper('ContactManager.Comentario');
```

Podemos carregar helper condicionalmente

```
class AppView extends View
{
    public function initialize()
    {
        parent::initialize();
        if ($this->request->action === 'index') {
            $this->loadHelper('Message');
        }
    }
}
```

Usando o Helper

```
$this->Form->create($article);
```

Alguns helpers são carregados pelo Cake automaticamente, como é o caso do Form.
Também podemos usar o método beforeRender() dos controllers para carregar Helpers.

```
class ArticlesController extends AppController
{
    public function beforeRender(Event $event)
    {
        parent::beforeRender($event);
        $this->viewBuilder()->helpers(['MyHelper']);
    }
}
```

Criando Helpers

Podemos criar classes Helper para usar em aplicações ou em plugins.

Os Helpers têm algumas convenções que ajudam se seguirmos:
- Arquivos de classe Helper devem ficar em src/View/Helper. Exemplo:
  src/View/Helper/LinkHelper.php
- Classes helpers devem ser sufixadas com Helper. Exemplo:
  LinkHelper
- Quando referenciar classes helper deve omitir o sufixo Helper:
  $this->loadHelper('Link');

Criando o esqueleto de um Helper chamado Message com o bake:

```
cd c:\xampp\htdocs\aplicativo
bin\cake bake helper Message
```

Editar o src\View\Helper\MessageHelper.php e adicionar as funções, assim:

```
<?php
namespace App\View\Helper;

use Cake\View\Helper;
use Cake\View\View;

/**
 * Table helper
 */
class TableHelper extends Helper
{

    /**
     * Default configuration.
     *
     * @var array
     */
      protected $_defaultConfig = [];

      public function msg($msg)
      {
            return '<h2>'.$msg.'</h1>';
      }
}
```

Carregar na classe AppView

Editar src/View/AppView.php e adicionar o loadHelper()

```
class AppView extends View
{
    public function initialize()
    {
        parent::initialize();
        if ($this->request->action === 'index') {
            $this->loadHelper('Message');
        }
    }
}
```

Atente para o fato de que na classe acima ele restringe para que somente no action index.ctp o helper apareça. Para mostrar em todas as actions use:

Observação – pelos meus testes nem precisa carregar na AppView e o helper ficará disponível automaticamente.

```
    public function initialize()
    {
            $this->loadHelper('Message');
    }
```

Usando o Helper
Editar a view index.ctp e adicionar:

```
<?=$this->Message->msg('Minha Mensagem Helper') ?>
```

Ao chamar no navegador verá a mensagem com título h2.
Um helper bem simples para mostrar o caminho das pedras.

Exemplo de criação e uso de helper

Colocar um valor de um determinado objeto disponível em sua view no layout geral.

Este helper tem como função servir de título da página.

views/layouts/default.ctp

```
<div id="container">
  <?php $session->flash(); ?>

  <?php if ($h->show_title()): ?>
    <h1><?php echo $h->show_title(); ?></h1>
  <?php endif; ?>

  <div id="content">
    <?php echo $content_for_layout; ?>
  </div>
</div>
```

views/pages/show.ctp

```
<?php $h->title($page["Page"]["name"]); ?>

<div class="content_cms">
  <?php echo $page["Page"]["body"] ?>
</div>
```

views/helpers/h.php

```
<?php
class HHelper extends Helper
{
  var $title;
```

```
  public function title($title='')
  {
    $this->title = $title;
  }

  public function show_title()
  {
    return $this->title;
  }
}
?>
```

Máscaras no CakePHP

- Máscara na view index (CPF)

Enquanto não descubro uma função do Cake que faz isso...

O CPF é um campo texto mas constituido somente de números. A listagem (index.ctp) mostra a relação de números sem nenhuma formatação. Ajudaria se formatassemos usando uma máscara adequada para CPF, ajudaria a visualizar. Vamos fazer isso agora para a view Clientes/index.ctp:

Substitua a linha

```
        <td><?php echo h($cliente['Cliente']['cpf']); ?> </td>
```

Por este código:

```
<?php
$cpfmask1 = substr($cliente['Cliente']['cpf'], 0,3);
$cpfmask2 = substr($cliente['Cliente']['cpf'], 3,3);
$cpfmask3 = substr($cliente['Cliente']['cpf'], 6,3);
$cpfmask4 = substr($cliente['Cliente']['cpf'], 9,2);
$cpfmask = $cpfmask1.'.'.$cpfmask2.'.'.$cpfmask3.'-'.$cpfmask4
 ?>
        <td><?php echo h($cpfmask); ?> </td>
```

Com isso a listagem mostrará o CPF com a sua máscara.

- Máscara em formulários com jQuery

Adicionando Máscaras com o plugin maskedinput do jQuery:

- Baixar o jQuery - http://jquery.com/
  Maskedinput - http://digitalbush.com/projects/masked-input-plugin/
- Renomear jQuery para jquery.js e maskedinput para jquery.maskedinput.js
- Copiar ambos para app/webroot/js

- Criar a referência para ambos em app/View/Layout/default.ctp ou outro layout, na seção head, assim:

```
echo $this->Html->script(array('jquery','jquery.maskedinput'));
```

- Editar a view onde deseja a máscara e adicionar ao final:

```
<script type="text/javascript">
   jQuery(document).ready(function($){
      $("#cpf").focus();
      $("#cpf").mask("999.999.999-99");
   });
</script>
```

Ainda na mesma view edite o campo onde ficará a máscara:

```
echo $this->Form->input('cpf',array('label'=>'CPF','id'=>'cpf'));
```

Crétidos: http://linguagensdeprogramacao.wordpress.com/2011/11/16/mascaras-com-cakephp/

Quando customizamos um campo para exibir máscara, precisamos remover a máscara antes de mandar para o banco, especialmente campos numéricos.

Dicas sobre data e hora

Para o CakePHP os campos do tipo data, datatime ou timestamp obrigatoriamente devem usar DEFAULT NULL:

nascimento date DEFAULT NULL

E também não podemos alterar sua validação para exigir preenchimento com notEmpty ou notBlank. Caso contrário
o Cake não reconhece e não adiciona o registro.

src/Template/Clientes/add.ctp ou edit.ctp

Ano mínimo sendo 13 anos antes do atual e máximo sendo 100 anos antes do atual, ou seja,
como estou em 2016 de 1916 até 2003

```
echo $this->Form->input('nascimento',['label' => 'Nascimento',
  'dateFormat'    => 'DMY',
  'minYear'       => date('Y') - 100,
  'maxYear'       => date('Y') - 13,
  'empty'         => [
     'day'       => 'Dia',
     'month'     => 'Mês',
     'year'      => 'Ano'
  ]
]);
```

13.3.1 – Forms

Forms no CakePHP 3

Tradução simplificada e resumida do original em:
http://book.cakephp.org/3.0/en/views/helpers/form.html

class `Cake\View\Helper\FormHelper`(*View $view*, *array $config* =[])

O FormHelper faz a maior parte do trabalho pesado na criação de formulários. O FormHelper foca na criação de formulários de forma rápida, de uma forma que irá agilizar a validação, re-população e layout. O FormHelper também é flexível - ele vai fazer quase tudo por você, utilizando convenções, ou você pode usar métodos específicos para obter apenas o que você precisa.

Iniciando um Formulário

`Cake\View\Helper\FormHelper::create`(*mixed $model* = *null*, *array $options* =[])

O primeiro método que você deve usar para tirar vantagem do FormHelper é o create(). Este método mostra uma tag de abertura de formulário.

Todos os parâmetros são opcionais. Caso create() seja chamado sem nenhum parâmetro, ele assume que você está construindo um form que submete para o controller atual, via URL atual. O método default para a submissão do form é o POST. Caso você chame o create dentro da view para UsersController::add(), você deve ver a seguinte saída na view renderizada:

```
<form method="post" action="/users/add">
```

O argumento $model é usado como forma de contexto. Existem vários contextos embutidos para formulário e você pode adicionar seu próprio, o que nós vamos cobrir na próxima seção. O provedores internos mapeiam para os seguintes valores de $model:

- Uma instância de entidade ou um mapa de iterator para o EntityContext, neste contexto permite FormHelper trabalhar com os resultados do ORM embutido.
- Um array que contém a chave de esquema, mapeia para ArrayContext que permite a criação de estruturas de dados simples para construir forms contra.
- null e false mapeiam para o NullContext, esta classe de contexto simplesmente satisfizer a interface que a FormHelper requer. Este contexto é útil se você quiser construir um pequeno formulário que não requer persistência ORM.

Para criar um form para uma entity faça o seguinte:

```
// If you are on /articles/add
// $article should be an empty Article entity.
echo $this->Form->create($article);
```

Isto irá postar os dados do formulário para o action add () de ArticlesController. No entanto, você também pode usar a mesma lógica para criar um formulário de edição. O FormHelper utiliza o objeto de entidade para detectar automaticamente se criará um form de adicionar ou editar. Se a entidade fornecido não é "novo", o formulário será criado como um formulário de edição. Se for novo será add.

Por exemplo, se navegar para
http://example.org/articles/edit/5, nós devemos fazer o seguinte:

```
// src/Controller/ArticlesController.php:
public function edit($id = null)
{
    if (empty($id)) {
        throw new NotFoundException;
    }
    $article = $this->Articles->get($id);
    // Save logic goes here
    $this->set('article', $article);
}

// View/Articles/edit.ctp:
// Since $article->isNew() is false, we will get an edit form
<?= $this->Form->create($article) ?>
```

Mudando o Método para o Form

```
echo $this->Form->create($article, ['type' => 'get']);
```

Quando existir algum campo do tipo file, precisará ser assim:

```
echo $this->Form->create($article, ['type' => 'file']);
```

Configurando a URL para o Form

```
echo $this->Form->create($article, ['url' => ['action' => 'login']]);
```

Caso o desejado form action não seja do controller atual, podemos especificar a completa URL para a action do form:

```
echo $this->Form->create(null, [
    'url' => ['controller' => 'Articles', 'action' => 'publish']
]);
```

Ou pode apontar para uma URL externa:

```
echo $this->Form->create(null, [
    'url' => 'http://www.google.com/search',
    'type' => 'get'
]);
```

Criando Inputs para Forms

`Cake\View\Helper\FormHelper::input`(*string $fieldName*, *array $options* =[])

O método input() deixa você gerar inputs completos para forms. O método input() deve incluir um invólucro/wrapping
div,
label
input widget
validação de erros se necessário

Relação de tipos de campos no input():

Column Type - Resulting Form Field

string, uuid (char, varchar, etc.) - text
boolean, tinyint(1) - checkbox
decimal -number
float -number
integer - number
text - textarea
text, with name of password, passwd - password
text, with name of email - email
text, with name of tel, telephone, or phone - tel
date - day, month, and year selects
datetime, timestamp - day, month, year, hour, minute, and meridian selects
time - hour, minute, and meridian selects
binary - file

O segundo parâmetro ($options), permite que mudemos o tipo de input:

```
echo $this->Form->input('published', ['type' => 'checkbox']);
```
No caso: ['type' => 'checkbox']

Exemplo de form:

```
echo $this->Form->create($user);
// Text
echo $this->Form->input('username');
// Password
echo $this->Form->input('password');
// Day, month, year, hour, minute, meridian
echo $this->Form->input('approved');
// Textarea
echo $this->Form->input('quote');

echo $this->Form->button('Add');
echo $this->Form->end();
```

Campo data:

```
echo $this->Form->input('birth_dt', [
    'label' => 'Date of birth',
    'minYear' => date('Y') - 70,
```

```
    'maxYear' => date('Y') - 18,
]);
```

Podemos especificar qualquer opção para o tipo do input e qualquer atributo HTML.

Se você deseja criar um campo select enquanto usando uma relação belogTo ou hasOne você pode adicionar o seguinte para seu UsersController (assumindo que User belongTo Group):

```
$this->set('groups', $this->Users->Groups->find('list'));
```

Então na view do Template:
```
echo $this->Form->input('group_id', ['options' => $groups]);
```

Para um select para uma associação belongsToMany Groups você pode adicionar o seguinte para seu UsersController:

```
$this->set('groups', $this->Users->Groups->find('list'));
```

Na view:
```
echo $this->Form->input('groups._ids', ['options' => $groups]);
```

Se o nome do model consiste em duas ou mais palavras, por exemplo, "UserGroup", ao passar os dados usando set() você deve nomear os seus dados com um formato pluralizado e camelCase como se segue:

```
$this->set('userGroups', $this->UserGroups->find('list'));
```

Não usar input() para gerar submits

Para isso usar o método \View\Helper\FormHelper::submit().

Options do Input

$options['type']

```
echo $this->Form->input('field', ['type' => 'file']);
echo $this->Form->input('email', ['type' => 'email']);
```

$options['label']

```
echo $this->Form->input('name', [
    'label' => 'The User Alias'
]);
```

Outros:
```
echo $this->Form->input('name', [
    'label' => [
        'class' => 'thingy',
        'text' => 'The User Alias'
    ]
]);
```

Tipo select:

```
$sizes = ['s' => 'Small', 'm' => 'Medium', 'l' => 'Large'];
echo $this->Form->select('size', $sizes, ['default' => 'm']);
```

$options['value']

```
echo $this->Form->time('close_time', [
    'value' => '13:30:00'
]);

echo $this->Form->select('rooms', [
    'multiple' => true,
    // options with values 1 and 3 will be selected as default
    'default' => [1, 3]
]);
```

$options['empty']

```
echo $this->Form->select(
    'field',
    [1, 2, 3, 4, 5],
    ['empty' => '(choose one)']
);
```

$options['datetime']

```
echo $this->Form->input('time', [
    'type' => 'time',
    'interval' => 15
]);
```

Outros

```
echo $this->Form->text('username', ['class' => 'users']);

echo $this->Form->password('password');

echo $this->Form->hidden('id');

echo $this->Form->textarea('notes');

echo $this->Form->textarea('notes', ['escape' => false]);
// OR....
echo $this->Form->input('notes', ['type' => 'textarea', 'escape' => false]);

echo $this->Form->textarea('textarea', ['rows' => '5', 'cols' => '5']);

echo $this->Form->checkbox('done');

echo $this->Form->checkbox('done', ['value' => 555]);

echo $this->Form->radio(
    'favorite_color',
    [
        ['value' => 'r', 'text' => 'Red', 'style' => 'color:red;'],
        ['value' => 'u', 'text' => 'Blue', 'style' => 'color:blue;'],
        ['value' => 'g', 'text' => 'Green', 'style' => 'color:green;'],
    ]
);
```

```
$options = ['M' => 'Male', 'F' => 'Female'];
echo $this->Form->select('gender', $options);

$options = [
   'Group 1' => [
      'Value 1' => 'Label 1',
      'Value 2' => 'Label 2'
   ],
   'Group 2' => [
      'Value 3' => 'Label 3'
   ]
];
echo $this->Form->select('field', $options);

echo $this->Form->select('field', $options, ['multiple' => true]);


echo $this->Form->datetime('released', [
    'year' => [
        'class' => 'year-classname',
    ],
    'month' => [
        'class' => 'month-class',
        'data-type' => 'month',
    ],
]);

echo $this->Form->time('released', [
    'interval' => 15,
    'hour' => [
        'class' => 'foo-class',
    ],
    'minute' => [
        'class' => 'bar-class',
    ],
]);

echo $this->Form->year('purchased', [
    'minYear' => 2000,
    'maxYear' => date('Y')
]);

echo $this->Form->month('mob');

echo $this->Form->month('mob', ['monthNames' => false]);

echo $this->Form->day('created');

echo $this->Form->hour('created', [
    'format' => 12
]);
echo $this->Form->hour('created', [
    'format' => 24
]);


echo $this->Form->minute('created', [
    'interval' => 10
]);
```

```
echo $this->Form->submit();


echo $this->Form->submit('ok.png');

echo $this->Form->button('A Button');
echo $this->Form->button('Another Button', ['type' => 'button']);
echo $this->Form->button('Reset the Form', ['type' => 'reset']);
echo $this->Form->button('Submit Form', ['type' => 'submit']);// Evitar
```

Todos os campos do form:

```
echo $this->Form->allInputs(
    [
        'name' => ['label' => 'custom label']
    ],
    null,
    ['legend' => 'Update your post']
);
```

Se estivéssemos editar um artigo com suas associações carregadas poderíamos criar as seguintes entradas:

```
$this->Form->create($article);

// Article inputs.
echo $this->Form->input('title');

// Author inputs (belongsTo)
echo $this->Form->input('author.id');
echo $this->Form->input('author.first_name');
echo $this->Form->input('author.last_name');

// Author profile (belongsTo + hasOne)
echo $this->Form->input('author.profile.id');
echo $this->Form->input('author.profile.username');

// Tags inputs (belongsToMany)
echo $this->Form->input('tags.0.id');
echo $this->Form->input('tags.0.name');
echo $this->Form->input('tags.1.id');
echo $this->Form->input('tags.1.name');

// Multiple select element for belongsToMany
echo $this->Form->input('tags._ids', [
    'type' => 'select',
    'multiple' => true,
    'options' => $tagList,
]);

// Inputs for the joint table (articles_tags)
echo $this->Form->input('tags.0._joinData.starred');
echo $this->Form->input('tags.1._joinData.starred');

// Comments inputs (hasMany)
echo $this->Form->input('comments.0.id');
echo $this->Form->input('comments.0.comment');
echo $this->Form->input('comments.1.id');
echo $this->Form->input('comments.1.comment');
```

Mais detalhes em:
http://book.cakephp.org/3.0/en/views/helpers/form.html

Select com valor default

```
$options = ['A'=>'Val1',
            'B'=>'Val2',
            'C'=>'Val3',
            'D'=>'Val4',
            'E'=>'Val5'];
 print $this->Form->select('category', $options,['multiple'=>'checkbox', 'required'=>'false',
'label'=>'Category','class'=>'col-md-12','value'=>'E']);
```

Select múltiplo (permite selecionar várias opções)

```
      print $this->Form->input('pilot_ratings',[
            'type' => 'select',
            'class' => 'listbox',
            'size' => 5,
            'id' => 'pilot_ratings',
            'multiple' => 'multiple',
            'options' => [
                ['name' => 'Habilitación de Vuelo Nocturno Local', 'value' => '1'],
                ['name' => 'Habilitación Cat. II / Cat. III', 'value' => '2'],
                ['name' => 'Habilitación de Remolque de Planeador', 'value' => '5']
            ]
      ]);
```

src/Template/Clientes/add.ctp ou edit.ctp

Ano mínimo sendo 13 anos antes do atual e máximo sendo 100 anos antes do atual, ou seja,
como estou em 2016 de 1916 até 2003

```
            echo $this->Form->input('nascimento',['label' => 'Nascimento',
              'dateFormat'    => 'DMY',
              'minYear'       => date('Y') - 100,
              'maxYear'       => date('Y') - 13,
              'empty'         => [
                  'day'       => 'Dia',
                  'month'     => 'Mês',
                  'year'      => 'Ano'
            ]
       ]);
```

Mudar largura de campo de form:

```
<?= $this->Form->input('username',['style'=>'width: 200px']) ?>
```

Select com valor default

```
$options = ['A'=>'Val1',
            'B'=>'Val2',
            'C'=>'Val3',
            'D'=>'Val4',
            'E'=>'Val5'];
 print $this->Form->select('category', $options,['multiple'=>'checkbox', 'required'=>'false',
'label'=>'Category','class'=>'col-md-12','value'=>'E']);
```

Select múltiplo (permite selecionar várias opções)

```
      print $this->Form->input('pilot_ratings',[
            'type' => 'select',
            'class' => 'listbox',
            'size' => 5,
            'id' => 'pilot_ratings',
            'multiple' => 'multiple',
            'options' => [
                ['name' => 'Habilitación de Vuelo Nocturno Local', 'value' => '1'],
                ['name' => 'Habilitación Cat. II / Cat. III', 'value' => '2'],
                ['name' => 'Habilitación de Remolque de Planeador', 'value' => '5']
            ]
      ]);
```

src/Template/Clientes/add.ctp ou edit.ctp

Ano mínimo sendo 13 anos antes do atual e máximo sendo 100 anos antes do atual, ou seja,
como estou em 2016 de 1916 até 2003

```
            echo $this->Form->input('nascimento',['label' => 'Nascimento',
              'dateFormat'    => 'DMY',
              'minYear'       => date('Y') - 100,
              'maxYear'       => date('Y') - 13,
              'empty'         => [
                  'day'       => 'Dia',
                  'month'     => 'Mês',
                  'year'      => 'Ano'
            ]
       ]);
```

Criando dois selects estáticos para os campos controller e action em Permissions/add.ctp

```
$options =
['Customers'=>'Customers','Groups'=>'Groups','Users'=>'Users','Permissions'=>'Permissio
ns','Products'=>'Products','ProductItems'=>'ProductItems','value'=>'Selecione'];
echo $this->Form->input('controller',['options'=>$options,'required'=>'false', 'class'=>'col-
md-12','empty'=>'Selecione']);

$options2 = ['index'=>'index','add'=>'add','edit'=>'edit','view'=>'view','delete'=>'delete'];
                    echo $this->Form->input('action', ['options'=>$options2,'required'=>'false',
'class'=>'col-md-12', 'empty'=>'Selecione']);
```

São estáticos, portanto sempre que precisar adicionar um novo controller ou action, precisa alterar este código.

Para que Html->Link permita CSS:

```
<?= $this->Html->Link('<span class="glyphicon glyphicon-plus"></span> Novo',
['controller'=>'Bookmaarks','action'=>'add'],
    ['class'=>'btn tbn-primary pull-right']);
```

Assim ele mostrará o código <span...

Para que permita o CSS, use:

```
<?= $this->Html->link('<span class="glyphicon glyphicon-plus"></span> Novo',
['controller'=>'Bookmaarks','action'=>'add'],
    ['class'=>'btn tbn-primary pull-right', 'escape'=>false]);
```

Observação:
classe Html, método link. Classe com inicial máiúscuça e método tudo em minúsculas.

Mudar o tipo de um campo

O Cake gerou um campo com o tipo textarea em um form.
Para mudar podemos fazer isso:

```
print $this->Form->input('url',['label', 'URL']);
```

Mudar para tipo text (campo texto) assim:

```
print $this->Form->input('url',['type'=>'text','label', 'URL']);
```

https://book.cakephp.org/3.0/pt/core-libraries/form.html

Formulários sem Models ##########################m

class Cake\Form\Form

Muitas vezes você precisará ter formulários associados ao ORM entities e ORM tables ou outras persistência de dados, mas há vezes quando você precisará validar um campo de usuário e então realizar uma ação se o dado é válido. O exemplo mais comum para esta situação é o formulário de contato.
Criando o Formulário

Geralmente quando se usa a classe Form será necessário utilizar uma sub-classe para definir seu formulário. Isso facilita o teste, e permite o reuso do formulário. Formulários ficam dentro de src/Form e comumente tem Form como sufixo da classe. Por exemplo, um simples formulário de contato poderia ficar assim:

```
// em src/Form/ContactForm.php
namespace App\Form;

use Cake\Form\Form;
use Cake\Form\Schema;
use Cake\Validation\Validator;

class ContactForm extends Form
{

    protected function _buildSchema(Schema $schema)
    {
        return $schema->addField('name', 'string')
            ->addField('email', ['type' => 'string'])
            ->addField('body', ['type' => 'text']);
    }

    protected function _buildValidator(Validator $validator)
    {
        return $validator->add('name', 'length', [
                'rule' => ['minLength', 10],
                'message' => 'A name is required'
            ])->add('email', 'format', [
                'rule' => 'email',
                'message' => 'A valid email address is required',
            ]);
    }

    protected function _execute(array $data)
    {
        // Envie um email.
        return true;
    }
```

```
}
```

No exemplo acima vemos os 3 métodos hooks que o formulário fornece:

_buildSchema é usado para definir o esquema de dados usado pelo FormHelper para criar um formulário HTML. Você pode definir o tipo de campo, tamanho, e precisão.
_buildValidator Pega uma instância do Cake\Validation\Validator que você pode você juntar com os validadores.
_execute permite definir o comportamento desejado quando o execute() é chamado e o dado é válido.

Você sempre pode adicionar métodos públicos a sua maneira.

Processando Requisição de Dados

Uma vez definido o formulário, é possível usá-lo em seu controller para processar e validar os dados:

```
// No Controller
namespace App\Controller;

use App\Controller\AppController;
use App\Form\ContactForm;

class ContactController extends AppController
{
    public function index()
    {
        $contact = new ContactForm();
        if ($this->request->is('post')) {
            if ($contact->execute($this->request->getData())) {
                $this->Flash->success('We will get back to you soon.');
            } else {
                $this->Flash->error('There was a problem submitting your form.');
            }
        }
        $this->set('contact', $contact);
    }
}
```

No exemplo acima, usamos o método execute() para chamar o nosso método _execute() do formulário apenas quando o dado é válido, e definimos as mensagens flash adequadas. Poderíamos também ter usado o método validate() apenas para validar a requisição de dados:

```
$isValid = $form->validate($this->request->getData());
```

Definindo os Valores do Formulário

Na sequência para definir os valores para os campos do formulário sem modelo, basta apenas definir os valores usando $this->request->getData(), como em todos os outros formulários criados pelo FormHelper:

```
// Em um controller
namespace App\Controller;

use App\Controller\AppController;
use App\Form\ContactForm;

class ContactController extends AppController
{
    public function index()
    {
        $contact = new ContactForm();
        if ($this->request->is('post')) {
            if ($contact->execute($this->request->getData())) {
                $this->Flash->success('Retornaremos o contato em breve.');
            } else {
                $this->Flash->error('Houve um problema ao enviar seu formulário.');
            }
        }

        if ($this->request->is('get')) {
            //Values from the User Model e.g.
            $this->request->getData('name', 'John Doe');
            $this->request->getData('email','john.doe@example.com');
        }

        $this->set('contact', $contact);
    }
}
```

Valores devem apenas serem definidos se a requesição é do tipo GET, caso contrário você sobreescreverá os dados anteriormente passados via POST que de certa forma poderiam estar incorretos e não salvos.
Pegando os Erros do Formulário

Uma vez sido validado, o formulário pode recuperar seus próprios erros:

```
$errors = $form->errors();
/* $errors contains
[
    'email' => ['A valid email address is required']
]
*/
```

Criando o HTML com FormHelper

Uma vez sido criado uma class Form, Once you've created a Form class, você provavelmente vai querer criar um formulário HTML para isso. FormHelper compreende objetos Form apenas como entidades ORM:

```
echo $this->Form->create($contact);
echo $this->Form->input('name');
echo $this->Form->input('email');
echo $this->Form->input('body');
echo $this->Form->button('Submit');
echo $this->Form->end();
```

O código acima criar um formulário HTML para o ContactForm definidos anteriormente. Formulários HTML criados com FormHelper usará o esquema definido e validador para determinar os tipos de campos, tamanhos máximos, e validação de erros.

Form Helper

Form
$this->Form->control() - usado para criar elementos com o mesmo nome. Tem dois parâmetros:
Primeiro - nome do campo
Segundo - opcional, permite usar arrays com múltiplas opções

Para o CakePHP os campos do tipo data, datatime ou timestamp obrigatoriamente devem usar DEFAULT NULL:

nascimento date DEFAULT NULL

E também não podemos alterar sua validação para exigir preenchimento com notEmpty ou notBlank. Caso contrário o Cake não reconhece e não adiciona o registro.

src/Template/Clientes/add.ctp ou edit.ctp

Ano mínimo sendo 13 anos antes do atual e máximo sendo 100 anos antes do atual, ou seja, como estou em 2016 de 1916 até 2003

```
    echo $this->Form->input('nascimento',['label' => 'Nascimento',
      'dateFormat'    => 'DMY',
      'minYear'       => date('Y') - 100,
      'maxYear'       => date('Y') - 13,
      'empty'         => [
        'day'       => 'Dia',
        'month'     => 'Mês',
        'year'      => 'Ano'
    ]
]);
```

Select com valor default

```
$options = ['A'=>'Val1',
            'B'=>'Val2',
            'C'=>'Val3',
            'D'=>'Val4',
            'E'=>'Val5'];
 print $this->Form->select('category', $options,['multiple'=>'checkbox', 'required'=>'false',
'label'=>'Category','class'=>'col-md-12','value'=>'E']);
```

Select múltiplo (permite selecionar várias opções)

```
    print $this->Form->input('pilot_ratings',[
        'type' => 'select',
        'class' => 'listbox',
        'size' => 5,
        'id' => 'pilot_ratings',
        'multiple' => 'multiple',
        'options' => [
            ['name' => 'Habilitación de Vuelo Nocturno Local', 'value' => '1'],
            ['name' => 'Habilitación Cat. II / Cat. III', 'value' => '2'],
            ['name' => 'Habilitación de Remolque de Planeador', 'value' => '5']
        ]
    ]);
```

src/Template/Clientes/add.ctp ou edit.ctp

Ano mínimo sendo 13 anos antes do atual e máximo sendo 100 anos antes do atual, ou seja,
como estou em 2016 de 1916 até 2003

```
        echo $this->Form->input('nascimento',['label' => 'Nascimento',
          'dateFormat'    => 'DMY',
          'minYear'       => date('Y') - 100,
          'maxYear'       => date('Y') - 13,
          'empty'         => [
              'day'       => 'Dia',
              'month'     => 'Mês',
              'year'      => 'Ano'
          ]
      ]);
```

Form

Mudar o tipo de um campo

O Cake gerou um campo com o tipo textarea em um form.
Para mudar podemos fazer isso:

print $this->Form->input('url',['label', 'URL']);

Mudar para tipo text (campo texto) assim:

print $this->Form->input('url',['type'=>'text','label', 'URL']);

Dicas sobre data e hora

Para o CakePHP os campos do tipo data, datatime ou timestamp obrigatoriamente devem usar DEFAULT NULL:

nascimento date DEFAULT NULL

E também não podemos alterar sua validação para exigir preenchimento com notEmpty ou notBlank. Caso contrário
o Cake não reconhece e não adiciona o registro.

src/Template/Clientes/add.ctp ou edit.ctp

Por padrão o Cake mostra apenas os anos de 2011 até 2021 na combo Ano.

Vamos alterar para que o ano mínimo seja 13 anos antes do atual
e máximo seja 100 anos antes do atual, ou seja, como estou em 2016,
que mostre de 1916 até 2003, mas isso deve ser pensado para atender ao requisito
da tabela/aplicativo. No nosso caso, do DNOCS, devemos usar 18 anos antes,
ou mais para o primeiro?
E 100 antes do atual ou mais para o segundo.
Precisamos saber da legislação e usar a favor da segurança, deixando uma margem.

```
        echo $this->Form->input('nascimento',['label' => 'Nascimento',
          'dateFormat'    => 'DMY',
          'minYear'       => date('Y') - 100,
          'maxYear'       => date('Y') - 13,
          'empty'         => [
              'day'       => 'Dia',
              'month'     => 'Mês',
              'year'      => 'Ano'
        ]
    ]);
```

Criar o select controller e o action

```
<?php
    $controls = ['Groups'=>'Groups', 'Users'=>'Users', 'Permissions'=>'Permissions',
'Customers'=>'Customers'];
    $actions = ['index'=>'index','add'=>'add','edit'=>'edit','delete'=>'delete'];
    echo $this->Form->input('group_id', ['options' => $groups,
'empty'=>'Grupo','class'=>'col2']);
    echo $this->Form->input('controller',
['options'=>$controls,'class'=>'col2','empty'=>'Controller']);
    echo $this->Form->input('action',
['options'=>$actions,'class'=>'col2','empty'=>'Action']);
?>
```

13.3.2 - HtmlHelper no CakePHP 3

class Cake\View\Helper\HtmlHelper(*View $view*, *array $config* =[])

O papel do HtmlHelper no CakePHP é fazer opções relacionados a HTML mais fácil, mais rápido e mais resistente à mudança. Usando este helper permitirá que sua aplicação seja mais luz em seus pés, e mais flexível de onde ele é colocado em relação à raiz de um domínio.

Muitos métodos HtmlHelper incluem um parâmetro $attributes, que lhe permitem aderência em quaisquer atributo extras em suas tags. Aqui estão alguns exemplos de como usar o parâmetro $attributtes:

```
Desired attributes: <tag class="someClass" />
Array parameter: ['class' => 'someClass']

Desired attributes: <tag name="foo" value="bar" />
Array parameter:  ['name' => 'foo', 'value' => 'bar']
```

Inserindo Elementos Bem Formatados

A tarefa mais importante que o HtmlHelper realiza é a criação de marcação bem formado. Esta seção irá cobrir alguns dos métodos do HtmlHelper e como usá-los.

Criando a tag Charset

Cake\View\Helper\HtmlHelper::charset(*$charset=null*)

Usado para criar a meta tag especificando o charset do documento. O default valor é UTF-8. Veja um exemplo:

echo $this->Html->charset();

Uma alternativa:

echo $this->Html->charset('ISO-8859-1');

Linkando para Arquivos CSS

Cake\View\Helper\HtmlHelper::css(*mixed $path*, *array $options* =[])

Cria um link para um CSS. Caso o bloco option seja configurado para true, a tag link é adicionada para o bloco CSS que você pode imprimir dentro da tag head do documento.

Este método de inclusão do CSS assume que o CSS especificado reside dentro do webroot/css e não inicia com '/':

```
echo $this->Html->css('forms');
```

Deve gerar:
```
<link rel="stylesheet" href="/css/forms.css" />
```

O primeiro parâmetro deve ser um array para incluir múltiplos arquivos:

```
echo $this->Html->css(['forms', 'tables', 'menu']);
```

Saída:
```
<link rel="stylesheet" href="/css/forms.css" />
<link rel="stylesheet" href="/css/tables.css" />
<link rel="stylesheet" href="/css/menu.css" />
```

Adicionando CSS de um Plugin:

```
echo $this->Html->css('DebugKit.toolbar.css');
```

Criando Meta tags

```
<?= $this->Html->meta(
    'favicon.ico',
    '/favicon.ico',
    ['type' => 'icon']
);
?>

<?= $this->Html->meta(
    'keywords',
    'enter any meta keyword here'
);
?>

<?= $this->Html->meta(
    'description',
    'enter any meta description here'
);
?>
```

Linkando para Imagens

`Cake\View\Helper\HtmlHelper::image`(*string $path*, *array $options* =[])

```
echo $this->Html->image('cake_logo.png', ['alt' => 'CakePHP']);
```

Gera:
```
<img src="/img/cake_logo.png" alt="CakePHP" />
```

```
echo $this->Html->image("recipes/6.jpg", [
    "alt" => "Brownies",
    'url' => ['controller' => 'Recipes', 'action' => 'view', 6]
]);
```

Plugin:
```
echo $this->Html->image('DebugKit.icon.png');
```

Criando Links

```
echo $this->Html->link(
    'Enter',
    '/pages/home',
    ['class' => 'button', 'target' => '_blank']
);

echo $this->Html->link(
    'Dashboard',
    ['controller' => 'Dashboards', 'action' => 'index', '_full' => true]
);

echo $this->Html->link(
    'Delete',
    ['controller' => 'Recipes', 'action' => 'delete', 6],
    ['confirm' => 'Are you sure you wish to delete this recipe?'],
);

echo $this->Html->link('View image', [
    'controller' => 'Images',
    'action' => 'view',
    1,
    '?' => ['height' => 400, 'width' => 500]
]);
```

Linkando Vídeos

```
<?= $this->Html->media('audio.mp3') ?>

 // Output
 <audio src="/files/audio.mp3"></audio>

 <?= $this->Html->media('video.mp4', [
     'fullBase' => true,
     'text' => 'Fallback text'
 ]) ?>

<?= $this->Html->media(
     ['video.mp4', ['src' => 'video.ogg', 'type' => "video/ogg; codecs='theora,
vorbis'"]],
     ['autoplay']
 ) ?>
```

Linkando JavaScript

```
echo $this->Html->script('scripts');

echo $this->Html->script('http://code.jquery.com/jquery.min.js');

echo $this->Html->script('/otherdir/script_file');
```

Plugin:

```
echo $this->Html->script('DebugKit.toolbar.js');
```

Criando Cabeçalhos de Tabelas

`Cake\View\Helper\HtmlHelper::tableHeaders`(*array $names*, *array $trOptions = null*, *array $thOptions = null*)

```
echo $this->Html->tableHeaders(['Date', 'Title', 'Active']);

echo $this->Html->tableHeaders(
    ['Date','Title','Active'],
    ['class' => 'status'],
    ['class' => 'product_table']
);

echo $this->Html->tableHeaders([
    'id',
    ['Name' => ['class' => 'highlight']],
    ['Date' => ['class' => 'sortable']]
]);
```

Criando Células de Tabelas

`Cake\View\Helper\HtmlHelper::tableCells`(*array $data*, *array $oddTrOptions = null*, *array $evenTrOptions = null*, *$useCount = false*, *$continueOddEven = true*)

```
echo $this->Html->tableCells([
    ['Jul 7th, 2007', 'Best Brownies', 'Yes'],
    ['Jun 21st, 2007', 'Smart Cookies', 'Yes'],
    ['Aug 1st, 2006', 'Anti-Java Cake', 'No'],
]);

echo $this->Html->tableCells([
    ['Jul 7th, 2007', ['Best Brownies', ['class' => 'highlight']] , 'Yes'],
    ['Jun 21st, 2007', 'Smart Cookies', 'Yes'],
    ['Aug 1st, 2006', 'Anti-Java Cake', ['No', ['id' => 'special']]],
]);

echo $this->Html->tableCells(
    [
        ['Red', 'Apple'],
        ['Orange', 'Orange'],
        ['Yellow', 'Banana'],
    ],
    ['class' => 'darker']
);
```

Criando Trilhas Breadcrumb com HtmlHelper

`Cake\View\Helper\HtmlHelper::addCrumb`(*string $name*, *string $link = null*, *mixed $options = null*)
`Cake\View\Helper\HtmlHelper::getCrumbs`(*string $separator = '»'*, *string $startText = false*)
`Cake\View\Helper\HtmlHelper::getCrumbList`(*array $options =*, []*$startText = false*)

Muitas aplicações usam trilhas tipo breadcrumbs para tornar mais fácil a navegação dos usuários. Você pode criar uma trilha tipo breadcrumb na sua aplicação usando o HtmlHelper. Primeiro adicione ao seu layout default.ctp a seguinte linha, logo após a linha:

```
</nav>
```
Adicionar:
```
<?php echo $this->Html->getCrumbs(' > ', 'Início');?>
```

Alternativamente uma imagem como link:

```
<?php echo $this->Html->getCrumbs(' > ', [
    'text' => $this->Html->image('home.png'),
    'url' => ['controller' => 'Articles', 'action' => 'index'],
    'escape' => false
]);?>
```

Mais detalhes em:
http://book.cakephp.org/3.0/en/views/helpers/html.html
http://book.cakephp.org/3.0/en/views/helpers/paginator.html
http://book.cakephp.org/3.0/en/views/helpers/form.html
http://book.cakephp.org/3.0/en/views.html
http://book.cakephp.org/3.0/en/core-libraries/form.html
http://book.cakephp.org/3.0/en/core-libraries/email.html
http://book.cakephp.org/3.0/en/views/helpers.html

HTML

Para que Html->Link permita CSS:

```
<?= $this->Html->Link('<span class="glyphicon glyphicon-plus"></span> Novo',
['controller'=>'Bookmaarks','action'=>'add'],
     ['class'=>'btn tbn-primary pull-right']);
```

Assim ele mostrará o código <span...

Para que permita o CSS, use:

```
<?= $this->Html->link('<span class="glyphicon glyphicon-plus"></span> Novo',
['controller'=>'Bookmaarks','action'=>'add'],
     ['class'=>'btn tbn-primary pull-right', 'escape'=>false]);
```

Observação:
classe Html, método link. Classe com inicial máiúscuça e método tudo em minúsculas.

Link para Imagens

Cake\View\Helper\HtmlHelper::image(string $path, array $options = [])

Creates a formatted image tag. The path supplied should be relative to webroot/img/.

```
echo $this->Html->image('cake_logo.png', ['alt' => 'CakePHP']);
```

Irá mostrar:

```
<img src="/img/cake_logo.png" alt="CakePHP" />
```

To create an image link specify the link destination using the url option in $attributes.

```
echo $this->Html->image("recipes/6.jpg", [
    "alt" => "Brownies",
    'url' => ['controller' => 'Recipes', 'action' => 'view', 6]
]);
```

Will output:

```
<a href="/recipes/view/6">
    <img src="/img/recipes/6.jpg" alt="Brownies" />
</a>
```

If you are creating images in emails, or want absolute paths to images you can use the fullBase option:

```
echo $this->Html->image("logo.png", ['fullBase' => true]);
```

Will output:

```
<img src="http://example.com/img/logo.jpg" alt="" />
```

You can include image files from any loaded plugin using plugin syntax. To include plugins/DebugKit/webroot/img/icon.png You could use the following:

```
echo $this->Html->image('DebugKit.icon.png');
```

If you want to include an image file which shares a name with a loaded plugin you can do the following. For example if you had a Blog plugin, and also wanted to include webroot/img/Blog.icon.png, you would:

```
echo $this->Html->image('Blog.icon.png', ['plugin' => false]);
```

CSS CakePHP

Customizar CSS do bootstrap numa view

```
    echo $this->Form->input('grupo',['style'=>'width: 200px']);
```

Tipos de Campos

```
input[type="text"],
input[type="password"],
input[type="email"],
input[type="tel"],
input[type="select"] {
    max-width: 280px;
}
```

Usando num template login.ctp

```
<div class="users form">
<?= $this->Flash->render('auth') ?>
    <?= $this->Form->create() ?>
    <div align="center">
    <fieldset>
        <legend><?= __('<h3><b>Favor entrar seu com login e senha</b></h3>') ?
></legend>
        <?= $this->Form->input('username', ['label'=>'Login', 'class'=>'col4']) ?>
        <?= $this->Form->input('password',['label'=>'Senha', 'class'=>'col4']) ?>
    </fieldset>
    <?= $this->Form->button(__('Acessar')); ?>
    <?= $this->Form->end() ?>
    </div>
</div>
```

No arquivo
webroot/css/custom.css

```
/* Para a customização da largura dos campos de form: echo $this->Form->input('grupo',
['class'=>'col2']); */
.col1{
    width:50px;
}

.col2{
    width:100px;
}

.col3{
```

```
    width:150px;
}

.col4{
    width:200px;
}

.col5{
    width:250px;
}

.col6{
    width:300px;
}

.col7{
    width:350px;
}

.col8{
    width:400px;
}

.col9{
    width:950px;
}
```

13.3.3 – DateTime

Date & Time
https://book.cakephp.org/3.0/en/core-libraries/time.html

class Cake\I18n\Time
Data e Hora com CakePHP 3

class Cake\I18n\Time

Se você precisa das funcionalidades do TimeHelper fora de uma View, então use a classe Time:

```
use Cake\I18n\Time;

class UsersController extends AppController
{
    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Auth');
    }

    public function afterLogin()
    {
        $time = new Time($this->Auth->user('date_of_birth'));
        if ($time->isToday()) {
            // Greet user with a happy birthday message
            $this->Flash->success(__('Happy birthday to you...'));
        }
    }
}
```

CakePHP usa Chronos para dar poder ao seu utilitário Timer. Qualquer coisa que você possa fazer com Chronos e DateTime você pode fazer com Time e Date:

https://github.com/cakephp/chronos
http://api.cakephp.org/chronos/1.0/

Antes da versão 3.2 o CakePHP usava Carbon:
https://github.com/briannesbitt/Carbon

Criando Instâncias de Time

Existem algumas maneiras de criar instâncias de Time:

```
use Cake\I18n\Time;
```

```
// Create from a string datetime.
$time = Time::createFromFormat(
    'Y-m-d H:i:s',
    $datetime,
    'America/Fortaleza'
);

// Create from a timestamp
$time = Time::createFromTimestamp($ts);

// Get the current time.
$time = Time::now();

// Or just use 'new'
$time = new Time('2014-01-10 11:11', 'America/New_York');

$time = new Time('2 hours ago');
```

Mais:

```
// Fixate time.
$now = new Time('2014-04-12 12:22:30');
Time::setTestNow($now);

// Returns '2014-04-12 12:22:30'
$now = Time::now();

// Returns '2014-04-12 12:22:30'
$now = Time::parse('now');
```

Manipulação

Depois de criada a instância podemos manipular as instâncias de Time usando métodos setters.

```
$now = Time::now();
$now->year(2016)
      ->month(08)
      ->day(31);
```

Também podemos usar os métodos providos pelo PHP embutidos na classe DateTime:

```
$now->setDate(2016, 08, 31);
```

Datas podem ser modificdas através da subtração e adição de seus componentes:

```
$now = Time::now();
$now->subDays(5);
$now->addMonth(1);

// Using strtotime strings.
$now->modify('+5 days');
```

Você pode receber os componentes internos da data acessando suas propriedades:

```
$now = Time::now();
echo $now->year; // 2014
echo $now->month; // 5
echo $now->day; // 10
echo $now->timezone; // America/Fortaleza
```

Também é permitido assinalar diretamente essas propriedades para modificar a data:

```
$time->year = 2015;
$time->timezone = 'Europe/Paris';
```

Formatando

static Cake\I18n\Time::setJsonEncodeFormat($format)

O método configura o formato default usado quando convertendo um objeto para json:

```
Time::setJsonEncodeFormat('yyyy-MM-dd HH:mm:ss');
```

Este método precisa ser chamado estaticamente.

Cake\I18n\Time::i18nFormat($format = null, $timezone = null, $locale = null)

Algo bem comum a fazer com instâncias de Time é imprimir as datas formatadas. Veja:

```
$now = Time::parse('2014-10-31');

// Prints a localized datetime stamp.
echo $now;

// Outputs '10/31/14, 12:00 AM' for the en-US locale
$now->i18nFormat();

// Use the full date and time format
$now->i18nFormat(\IntlDateFormatter::FULL);

// Use full date but short time format
$now->i18nFormat([\IntlDateFormatter::FULL, \IntlDateFormatter::SHORT]);

// Outputs '2014-10-31 00:00:00'
$now->i18nFormat('yyyy-MM-dd HH:mm:ss');
```

Configurando o Locale Padrão e o Formato de String

```
// The same method exists on Date, FrozenDate, and FrozenTime
Time::setDefaultLocale('pt-BR');
```

Datas

A classe Date no CakePHP implementa a mesma API e métodos que o Time\ I18n\. A principal diferença entre hora e data é que Data não acompanha componentes de tempo, e está sempre em UTC. Como um exemplo:

```
use Cake\I18n\Date;
```

```
$date = new Date('2015-06-15');

$date->modify('+2 hours');
// Outputs 2015-06-15 00:00:00
echo $date->format('Y-m-d H:i:s');

$date->modify('+36 hours');
// Outputs 2015-06-15 00:00:00
echo $date->format('Y-m-d H:i:s');
```

Mais em:
http://book.cakephp.org/3.0/en/views/helpers/time.html
http://book.cakephp.org/3.0/en/core-libraries/time.html
http://book.cakephp.org/3.0/en/core-libraries/internationalization-and-localization.html#parsing-localized-dates

Máscaras no CakePHP

- Máscara na view index (CPF)

Enquanto não descubro uma função do Cake que faz isso...

O CPF é um campo texto mas constituido somente de números. A listagem (index.ctp) mostra a relação de números sem nenhuma formatação. Ajudaria se formatassemos usando uma máscara adequada para CPF, ajudaria a visualizar. Vamos fazer isso agora para a view Clientes/index.ctp:

Substitua a linha

```
        <td><?php echo h($cliente['Cliente']['cpf']); ?> </td>
```

Por este código:

```
<?php
$cpfmask1 = substr($cliente['Cliente']['cpf'], 0,3);
$cpfmask2 = substr($cliente['Cliente']['cpf'], 3,3);
$cpfmask3 = substr($cliente['Cliente']['cpf'], 6,3);
$cpfmask4 = substr($cliente['Cliente']['cpf'], 9,2);
$cpfmask = $cpfmask1.'.'.$cpfmask2.'.'.$cpfmask3.'-'.$cpfmask4
 ?>
        <td><?php echo h($cpfmask); ?> </td>
```

Com isso a listagem mostrará o CPF com a sua máscara.

- Máscara em formulários com jQuery
Adicionando Máscaras com o plugin maskedinput do jQuery:

- Baixar o jQuery - http://jquery.com/
        Maskedinput - http://digitalbush.com/projects/masked-input-plugin/

- Renomear jQuery para jquery.js e maskedinput para jquery.maskedinput.js
- Copiar ambos para app/webroot/js
- Criar a referência para ambos em app/View/Layout/default.ctp ou outro layout, na seção head, assim:

```
echo $this->Html->script(array('jquery','jquery.maskedinput'));
```

- Editar a view onde deseja a máscara e adicionar ao final:

```
<script type="text/javascript">
   jQuery(document).ready(function($){
      $("#cpf").focus();
      $("#cpf").mask("999.999.999-99");
   });
</script>
```

Ainda na mesma view edite o campo onde ficará a máscara:

```
echo $this->Form->input('cpf',array('label'=>'CPF','id'=>'cpf'));
```

Crétidos: http://linguagensdeprogramacao.wordpress.com/2011/11/16/mascaras-com-cakephp/

Quando customizamos um campo para exibir máscara, precisamos remover a máscara antes de mandar para o banco, especialmente campos numéricos.

JavaScript

No topo da view

```
echo $this->Html->script(array('ajaxupload.3.5.js'));

echo $this->Html>css(array('new_layout.css'));


<script type="text/javascript">
function filldetails()
{
   document.getElementById('FirstName').value  = "hjshjsh";
}
</script>
echo $this->Form->select('grupos',['onchange' =>grupo());
```

Exemplo de Helper

Criar na pasta

src/View/Template/SelectHelper.php

```
<?php
namespace App\View\Helper;

use Cake\View\Helper;
use Cake\View\View;

/**
 * Select helper
 */
class SelectHelper extends Helper
{

    /**
     * Exemplo:
     * Chamando assim
     * $options = array('admin'=>'Administrator','user'=>'Usuário');
       * echo $this->Form->input('title', $this->Select->sel($options,'user','Tipo'));
     * @var array
     */
    protected $_defaultConfig = [];

      public function sel($options,$default,$label)
      {
        $sel = array('type'=>'select', 'default'=>$default,'label'=>$label,'options'=>$options,
'empty'=>'Nenhum');
            return $sel;
      }
}
```

Usando o helper acima

No add.ctp

```
<?php
    <?= $this->Form->create($article) ?>
    <fieldset>
        <legend><?= __('Add Article') ?></legend>
        <?php
                $options = array('admin'=>'Administrator','user'=>'Usuário');
            echo $this->Form->input('title',$this->Select->sel($options,'user','Tipo'));
            echo $this->Form->input('body');
        ?>
    </fieldset>
    <?= $this->Form->button(__('Submit')) ?>
```

```
<?= $this->Form->end() ?>
```

14 – Controller

Os controllers correspondem ao ‘C’ no padrão MVC. Após o roteamento ter sido aplicado e o controller correto encontrado, a ação do controller é chamada. Seu controller deve lidar com a interpretação dos dados de uma requisição, certificando-se que os models corretos são chamados e a resposta ou view esperada seja exibida. Os controllers podem ser vistos como intermediários entre a camada Model e View. Você vai querer manter seus controllers magros e seus Models gordos. Isso lhe ajudará a reutilizar seu código e testá-los mais facilmente.

Mais comumente, controllers são usados para gerenciar a lógica de um único model. Por exemplo, se você está construindo um site para uma padaria online, você pode ter um RecipesController e um IngredientsController gerenciando suas receitas e seus ingredientes. No CakePHP, controllers são nomeados de acordo com o model que manipulam. É também absolutamente possível ter controllers que usam mais de um model.

Os controllers da sua aplicação são classes que estendem a classe CakePHP AppController, a qual por sua vez estende a classe Controller do CakePHP. A classe AppController pode ser definida em /app/Controller/AppController.php e deve conter métodos que são compartilhados entre todos os seus controllers.

Os controllers fornecem uma série de métodos que são chamados de ações. Ações são métodos em um controller que manipulam requisições. Por padrão, todos os métodos públicos em um controller são ações e acessíveis por urls.

Os atributos e métodos criados em AppController vão estar disponíveis para todos os controllers da sua aplicação. Este é o lugar ideal para criar códigos que são comuns para todos os seus controllers. Componentes (que você vai aprender mais tarde) são a melhor alternativa para códigos que são usados por muitos (mas não obrigatoriamente em todos) controllers.

Enquanto regras normais de herança de classes orientadas à objetos são aplicadas, o CakePHP também faz um pequeno trabalho extra quando se trata de atributos especiais do controller. A lista de componentes (components) e helpers usados no controller são tratados diferentemente. Nestes casos, as cadeias de valores do AppController são mescladas com os valores de seus controllers filhos. Os valores dos controllers filhos sempre sobrescreveram os do AppController.

O CakePHP mescla as seguintes variáveis do AppController em controllers da sua aplicação:

- $components
- $helpers
- $uses

Lembre-se de adicionar os helpers Html e Form padrões se você incluiu o atributo $helpers em seu AppController.

Também lembre de fazer as chamadas de callbacks do AppController nos controllers filhos para obter melhores resultados:

```
function beforeFilter() {
   parent::beforeFilter();
}
```

Usando dois ou mais models em um controller

Indicar model padrão:

```
function beforeFilter() {
    $this->Auth->userModel = 'Usuario';
}
```

Para usar em um controller mais de um model indique no controller:

```
public $uses = array('User', 'Permissao', 'Cliente');
```

Acessando Outros Models ou Controllers

Chamando outro controller:

```
App::import('Controller', 'Users');
$Users = new UsersController;
$Users->constructClasses();
```

Estando num model chamar outro model:

```
$Category = ClassRegistry::init("OutroModel");
$category = $Category->findById($underThisCategoryId);
```

Colocando classes personalizadas a disposição do aplicativo em Cake:

- insira a classe em um diretório
- Adicione ao app/Config/bootstrap.php (diretório fora do aplicativo do cake):

```
App::build(array(
    'GlobalUsers' => array(dirname(CAKE_CORE_INCLUDE_PATH).DS.'mydir'.DS)
), App::REGISTER);
```

- Adicione ao AppController.php:

```
App::uses('UsersController','GlobalUsers');
```

UsersController - nome do arquivo atual, no caso UsersController.php
Agora podemos acessar a classe do nosso controller atual.
Crédito: http://www.shahariaazam.com/access-custom-class-inside-cakephp-apps/

Recebendo informações sobre campos com facilidade

Estando em um controller:

```
$this->Post->id = 22;
echo $this->Post->field('name');
```

CakePHP Controller properties, methods, callbacks

CakePHP controller properties:

$action = null
$autoLayout = true
$autoRender = true
$base = null
$beforeFilter = null
$cacheAction = false
$components = array()
$data = array()
$helpers = array()
$here = null
$layout = 'default'
$name = null
$output = null
$pageTitle = false
$params = array()
$persistModel = false
$plugin = null
$uses = false
$view = 'View'
$viewPath = null
$webroot = null
$_viewClass = null
$_viewVars = array()

CakePHP controller methods

cleanUpFields ()
constructClasses ()
disableCache()
flash(string $message, string $url, integer $pause, string $layout)
flashOut ($message, $url, $pause = 1)
generateFieldNames ($data = null, $doCreateOptions = true)
loadModel(string $modelClass, mixed $id)
paginate()
postConditions ($data, $operator = '', $bool = 'AND', $exclusive = false)
redirect(mixed $url, integer $status, boolean $exit)
referer(mixed $default = null, boolean $local = false)

render ($action = null, $layout = null, $file = null)
requestAction(string $url, array $options)
set (string $var, mixed $value)
setAction ($action)
validate ()
validateErrors ()

CakePHP controller callbacks

afterFilter ()
beforeFilter ()
beforeRender ()

Vide callbacks.txt para detalhes.

https://book.cakephp.org/3.0/pt/controllers.html

Exemplo de Controller Comentado

```
<?php
// O comando abaixo faz o include da classe AppController que encontra-se na pasta
app/Controller
App::uses('AppController', 'Controller');
/**
 * Posts Controller
 *
 * @property Post $Post
 */
// Observe que o controller PostsController extende a classe AppController
class PostsController extends AppController {

/**
 * index method
 *
 * @return void
 */
      public function index() {
/*
```

A propriedade recursive define qual a profundidade que o CakePHP deve ir para procurar modelos associados de dados via métodos find() e read().

Imagine uma aplicação com Groups e cada Group contém muitos Users que por sua vez, cada User tem muitos Articles. Você pode setar $recursive para vários valores baseado no total de dados que você deseja de volta de uma chamada $this->Group->find():

-1 Cake busca somente dados de Group, sem joins.
0 Cake busca dados de Group e seu domínio

1 Cake busca um Group, seu domínio e seus Users associados
2 Cake busca um Group, seu domínio, seus Users associados e os Articles assoaiados aos Users

Não defina $recursive mais do que você precisa. Quando você tem dados buscados pelo CakePHP você não vai deixar sua aplicação lenta desnecessariamente. Observe também que o nível recursivo padrão é 1.
*/
```
        $this->Post->recursive = 0;
/*
O método set() tem como objetivo enviar informações para a View.
Com a linha abaixo a view index.ctp poderá chamar o array $posts, que contém todos os posts paginados.
*/
        $this->set('posts', $this->paginate());
    }

/**
 * view method
 *
 * @throws NotFoundException
 * @param string $id
 * @return void
 */
    public function view($id = null) {
        // Verifica com o model Post, se existe o $id recebido
        if (!$this->Post->exists($id)) {
            throw new NotFoundException(__('Invalid post'));
        }
        // Opções para filtrar o model Post que tem o $id como chave primária
        $options = array('conditions' => array('Post.' . $this->Post->primaryKey => $id));
        // Passa para a view a variável $post com o primeiro Post encontrado
        $this->set('post', $this->Post->find('first', $options));
    }

/**
 * add method
 *
 * @return void
 */
    public function add() {
        // Verifica se a requisição é do tipo POST
        if ($this->request->is('post')) {
            // Cake PHP recomenda chamar a função create() para re-inicializar um model antes de salvar
            // um novo registro. Criar - create(). Salvar/Atualizar - save().
            $this->Post->create();
            // O método save() salva o registro e também valida
            if ($this->Post->save($this->request->data)) {
                // setFlash mostra uma mensagem
```

```
                $this->Session->setFlash(__('The post has been saved'));
                // redirect redireciona para o action index. Também existem outros
parâmetros
                $this->redirect(array('action' => 'index'));
            } else {
                $this->Session->setFlash(__('The post could not be saved. Please, try
again.'));
            }
        }
    }

/**
 * edit method
 *
 * @throws NotFoundException
 * @param string $id
 * @return void
 */
    public function edit($id = null) {
        // Verifica se existe o $id no model Post
        if (!$this->Post->exists($id)) {
            throw new NotFoundException(__('Invalid post'));
        }
    //echo 'CONTROLLER - Enviando para o Model';
    //exit;
        // Verifica se houve requisição do tipo POST ou PUT
        if ($this->request->is('post') || $this->request->is('put')) {
            if ($this->Post->save($this->request->data)) {
    //echo 'CONTROLLER - Vindo do Model';
    //exit;
                $this->Session->setFlash(__('The post has been saved'));
                $this->redirect(array('action' => 'index'));
            } else {
                $this->Session->setFlash(__('The post could not be saved. Please, try
again.'));
            }
        } else {
            $options = array('conditions' => array('Post.' . $this->Post->primaryKey =>
$id));
            $this->request->data = $this->Post->find('first', $options);
        }
    }

/**
 * delete method
 *
 * @throws NotFoundException
 * @param string $id
 * @return void
 */
```

```
    public function delete($id = null) {
        // Guarda em $this->Post->id o $id passado via parâmetro
        $this->Post->id = $id;
        if (!$this->Post->exists()) {
            throw new NotFoundException(__('Invalid post'));
        }
        // CakeRequest::onlyAllow($methods) - adicionado no Cake 2.3. Substituiu o
método
        // requireDelete() das versões anteriores.
        $this->request->onlyAllow('post', 'delete');
        // Exclui o registro
        if ($this->Post->delete()) {
            $this->Session->setFlash(__('Post deleted'));
            $this->redirect(array('action' => 'index'));
        }
        $this->Session->setFlash(__('Post was not deleted'));
        $this->redirect(array('action' => 'index'));
    }
}
```

Repassando Informações Entre Controllers

Como faço para repassar informações entre dois controllers, exemplo:

Eu tenho no meu banco de dados duas tabelas, uma de usuários e outra de livros, aí eu seleciono um livro que está na tabela livros, pego seu id e tenho que atualizar essa informação na tabela do usuário..

Eu tenho dois controllers, um de usuários e outro de livros, eu tenho que repassar o id do livro para o controller de usuários..

Como passo essa informação entre controllers
Como seleciono o id do usuário logado no sistema
Qual o jeito correto de fazer essa atualização

No cookbook você pode ver um método chamado loadModel(), com ele é possível vocês utilizar o Model de usuários dentro do seu controller livros.

Exemplo:

```
$this->loadModel('User');
$users = $this->User->find('all');
```

Caso queira carregar um dado específico de outro Model você pode usar:

```
$this->loadModel('Livro', 2);
$livro = $this->Livro->read();
```

Caso você tenha feito o relacionamento corretamente entre as tabelas e criado os Models corretamente você pode usar o sequinte para manipular os dados relacionados

```
$users = $this->User->Livro->find('all');
$livros = $this->Livro->User->find('all');
```

Para obter os dados do usuário logado no cakephp você pode utilizar a seção ou o component Auth
Auth

```
$this->Auth->user('id');
```

Session

```
$this->Session->read('User.id');
```

Passando dados entre controllers

No cookbook você pode ver um método chamado loadModel(), com ele é possível vocês utilizar o Model de usuários dentro do seu controller livros.
Exemplo:

```
$this->loadModel('User');
$users = $this->User->find('all');
```

Caso queira carregar um dado específico de outro Model você pode usar:

```
$this->loadModel('Livro', 2);
$livro = $this->Livro->read();
```

Caso você tenha feito o relacionamento corretamente entre as tabelas e criado os Models corretamente você pode usar o sequinte para manipular os dados relacionados

```
$users = $this->User->Livro->find('all');
$livros = $this->Livro->User->find('all');
```

Para obter os dados do usuário logado no cakephp você pode utilizar a seção ou o component Auth
Auth

```
$this->Auth->user('id');
```

Session

```
$this->Session->read('User.id');
```

14.1 – Componentes

O que é um Component segundo a doc do CakePHP
Os componentes são pacotes de lógica que são compartilhados entre controllers. O CakePHP vem com um conjunto fantástico de componentes que você pode usar para ajudar em várias tarefas comuns. Você também pode criar seus próprios componentes. Se você copia e cola parte de código entre controllers, você deve considerar criar seu próprio Component para conter essa funcionalidade. A criação de Components no CakePHP mantém o código dos controllers mais limpos e permite que você reutilize código entre controllers diferentes.

Documentação: https://book.cakephp.org/3.0/en/controllers/components.html#components

Criando um Component
Para nosso exemplo, vamos criar um Component para criação de senhas com caracteres aleatórios. Para isso precisamos criar uma classe no diretório

`src/Controller/Component/`, um nome de component seguindo a convenção do cake seria algo como: `MinhaClasseComponent.php` que extende da classe `Component`. Como umas das filosofias do nosso Framework favorito é facilitar o trabalho, podemos usar o PHP CLI(command line inteface) ou vulgarmente conhecido como Bake.

Para executar o console do cake, precisamos simplesmente executar o código abaixo.

Com esse comando teremos uma saída no terminal parecido com isso:

```
Welcome to CakePHP v3.3.14 Console
---------------------------------------------------------------
App : src
Path: /var/www/cakephp.dev/src/
PHP: 7.0.16
---------------------------------------------------------------
Current Paths:
App: src
Path: /var/www/cakephp.dev/src
Core: /var/www/cakephp.dev/vendor/cakephp/cakephp

Available Shells:

[Bake] bake

[DebugKit] benchmark, whitespace

[Migrations] migrations

[CORE] cache, i18n, orm_cache, plugin, routes, server

[app] console

To run an app or core command, type `cake shell_name [args]`
To run a plugin command, type `cake Plugin.shell_name [args]`
To get help on a specific command, type `cake shell_name --help`
```

Note que o shell do cake nos oferece algumas possibilidades para geraçao de código, debug, migrations e etc, a possibilidade é infinita.

No nosso caso usaremos o `bake`, agora se executarmos o seguinte código:

Teremos a seguinte saída no nosso terminal:

```
Welcome to CakePHP v3.3.14 Console
---------------------------------------------------------------
App : src
Path: /var/www/cakephp.dev/src/
PHP : 7.0.16
---------------------------------------------------------------
The following commands can be used to generate skeleton code for your
application.

Available bake commands:

- all
- behavior
- cell
- component
- controller
- fixture
- form
- helper
- mailer
- migration
- migration_diff
- migration_snapshot
- model
- plugin
- seed
- shell
- shell_helper
- task
- template
- test

By using `cake bake [name]` you can invoke a specific bake task.
```

Note que as possibilidades aumentaram, e podemos gerar códigos para controller, models, plugins, inclusive executar consultas no próprio shell.

Finalmente vamos criar nosso component, esta foi uma pequena introdução ao shell do cakePHP. Em seu terminal execute o seguinte comando:

```
bin/cake bake component UserGeneratePass
```

A saída no terminal será algo como:

```
Creating file
/var/www/cakephp.dev/src/Controller/Component/TestComponentTest.php
Wrote `/var/www/cakephp.dev/src/Controller/Component/TestComponentTest.php`

Creating file
/var/www/cakephp.dev/tests/TestCase/Controller/Component/TestComponentTest.php
```

```
Wrote
`/var/www/cakephp.dev/tests/TestCase/Controller/Component/TestComponentTest.php`
```

Note que nosso Bake já gera um arquivo de test, para executarmos testes uniários, para este tutorial não utilizaremos este arquivo, pode inclusive ficar uma pendência para um proximo post.

Enfim, Nosso componente foi criado, agora vamos criar o método responsável pela criação de senha:

```
<?php
namespace App\Controller\Component;

use Cake\Controller\Component;
use Cake\Controller\ComponentRegistry;

class UserGeneratePassComponent extends Component
{
    protected $_defaultConfig = [];

    // método responsável pela criação da senha.
    public function generatePass($length = 10, $uppercase = true, $number =
true, $symbol = false)
    {
        $lmin           = 'abcdefghijklmnopqrstuvwxyz';
        $lmai           = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
        $num            = '1234567890';
        $simb           = '!@#$%*-';
        $return         = '';
        $caracters      = '';
        $caracters      .= $lmin;

        if ($uppercase) $caracters .= $lmai;
        if ($number) $caracters .= $num;
        if ($symbol) $caracters .= $simb;

        $len = strlen($caracters);

        for ($n = 1; $n <= $length; $n++) {
             $rand = mt_rand(1, $len);
             $return .= $caracters[$rand-1];
        }

        return $return;
    }
}
```

Exemplo de uso
No seu controller `php UserController.php` por exemplo, você pode chamar o Component da seguinte forma:

```
<?php

class UserController extends Controller
{
        public function initialize()
    {
        parent::initialize();
```

```
        // importando o nosso component para ficar acessível ao nosso
controller.
        $this->loadComponent('UserGeneratePass');
    }

    public function add()
    {
        $user = $this->Users->newEntity();
        if ($this->request->is('post')) {

            // Gera uma senha com 15 carecteres de números, letras e símbolos
            if (!$user['password']){
                // aqui começa a mágica.
                $user['password'] = $this->UserGeneratePass->generatePass(15,
true, true, true);
            }

            $user = $this->Users->patchEntity($user, $this->request->data);
            if ($this->Users->save($user)) {
                $this->Flash->success('Usuário adicionado com sucesso.',
['class' => 'alert alert-info']);
                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('Não foi possível adicionar este usuário, por
favor tente novamente.'));
        }

        $this->set('user', $user);
    }
}
```

Conclusão
Esperamos que este tutorial simples ajude você no seu desenvolvimento, essa é uma dica bem básica mas que pode salvar vidas (risos).

Se você tiver dúvidas ou encontrar nossos erros no tutorial acima, deixe um comentário abaixo para nos informar.

Caso tenha ficado alguma dúvida, ou tenha outra solução, fique a vontade para deixar seu comentário, ou criar um PR com seu artigo, tenho certeza que a comunidade agradece.

Componentes são classes auxiliares dos controllers. Os Components podem ser considerados como formas de criar pedaços reutilizáveis de código relacionado a controllers com uma característica específica ou conceito. Os components também podem ligar-se ao evento do ciclo de vida do controller e interagir com a sua aplicação.

Criação de Componentes

Criar o arquivo

CalcularComponent.php, contendo:

```
<?php
namespace App\Controller\Component;

use Cake\Controller\Component;

class CalcularComponent extends Component
{
    public function soma($valor1, $valor2)
    {
        $s =$valor1 + $valor2;
            return $s;
    }
}
```

E copiar para a pasta src/Controller/Component

Adicionar uma chamada no src/Controller/AppController

```
    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Auth');
        $this->loadComponent('RequestHandler');
        $this->loadComponent('Flash');
            $this->loadComponent('Calcular');
    }
```

Executar num action, como o src/Controller/ClientesController.php

```
    public function index()
    {
            print 'Componente Soma: '.$this->Calcular->soma(2,3);
```

Estando no AppController ele será visto por todos os demais controllers e seus métodos estarão disponíveis logo que cada controller seja executado.

Caso queira que apenas um controller tenha acesso ao mesmo, insira uma chamada somente neste controller, assim:

```
$this->loadComponent('Calcular');

$soma = $this->Calcular->soma(2,4);
print 'Calcular: '.$soma;
```

Exemplos de Componentes Simples

```
<?php
/**
 * Component Calculo
 * For CakePHP 3.x
 * Autor: Ribamar FS
 *
 * This component allow access control to each action from application with web interface.
 *
 * Licenced with The MIT License
 * Redistributions require keep copyright below.
 *
 * @copyright     Copyright (c) Ribamar FS (http://ribafs.org)
 * @link          http://ribafs.org
 * @license       http://www.opensource.org/licenses/mit-license.php MIT License
 */


namespace App\Controller\Component;
use Cake\Controller\Component;

class CalculoComponent extends Component{

      private $um;
      private $dois;

      public function soma($um,$dois){
            return ($um + $dois);
      }

      public function subtracao($um,$dois){
            return ($um - $dois);
      }

      public function multiplicacao($um,$dois){
            return ($um * $dois);
      }

      public function divisao($um,$dois){
            return ($um / $dois);
      }

}
/*Usando
Salvar na pasta src/Controller/Component

Em um dos actions:
      public $components = array('Calculo');
```

```
    $soma = $this->Calculo->soma(2,3);
    $multiplicacao = $this->Calculo->multiplicacao(2,3);
    $this->set('soma',$soma);
    $this->set('multiplicacao',$multiplicacao);

No Template correspondente:
<?php print 'A soma é '.$soma;?>
<?php print '<br>O produto é '.$multiplicacao;?>
*/
```

Paginação

```
class ArticlesController extends AppController
{
    public $paginate = [
        'fields' => ['Articles.id', 'Articles.created'],
        'limit' => 25,
        'order' => [
            'Articles.title' => 'asc'
        ]
    ];

    public function initialize()
    {
        parent::initialize();
        $this->loadComponent('Paginator');
    }
}
```

Dicas sobre Redirecionamentos

```
Controller::redirect(mixed $url, integer $status, boolean $exit)

$this->redirect(array('controller' => 'clientes', 'action' => 'add'));

// Quando no próprio controller, basta citar o action
$this->redirect(array('action' => 'add'));

// Com path
$this->redirect('/clientes/index'));

// Com URL
$this->redirect('http://www.example.com');

// Passando parâmetro: $id
$this->redirect(array('action' => 'edit', $id));
```

```
//Voltar para a página que fez a requisição:
$this->redirect($this->referer());

$this->redirect($this->referer(array('action' => 'index')));

public function autoRedirect($whereTo, $useReferer = true) {
        if ($useReferer && $this->Controller->referer() != '/' . $this->Controller->params['url']
['url']) {
            $this->Controller->redirect($this->Controller->referer($whereTo, true));
        } else {
            $this->Controller->redirect($whereTo);
        }
}

public function postRedirect($whereTo, $status = 302) {
        $this->Controller->redirect($whereTo, $status);
    }

Redirect to 404 not found:
throw new NotFoundException('404 Error - Page not found');

// Conteúdo movido ou deletado
if (empty($manufacturer)) {
    $this->Session->setFlash(__('Invalid Manufacturer', true));
    $this->redirect(array('action'=>'index'), 301);
}
```

Redirecionar para si mesmo, ou seja, não sair da página atual

https://book.cakephp.org/3.0/en/controllers.html#redirecting-to-other-pages

14.2 - Request e Response em CakePHP 3

Os objetos request e response provem uma abstração para request e response HTTP. O objeto request em CakePHP permite que você inspecione um request de entrada, enquanto que um objeto response permite a você sem esforço criar responses HTTP de seus controllers.

Request

class Cake\Network\Request

Request é o objeto de request default usado em CakePHP. Ele centraliza algumas características para interrogar e interagir com dados de request. Em cada request um Request é criado e então passado por referência para as várias camadas de uma aplicação que usa dados de request. Por default o request é atribuído para $this->request e está disponível em Controllers, Cells, Views e Helpers. Você também pode acessar eles nos Components usando a referência do controller. Alguns dos deveres que o Request executa incluem:

- Processamento de arrays GET, POST e FILES em estruturas de dados que você já está familiarizado.

- Fornecendo ambiente de introspecção pertencentes ao request. Informações como os cabeçalhos enviados, o endereço IP do cliente, e os nomes domínio/subdomínio do servidor onde sua aplicativo está sendo executada.

- Proporcionar o acesso aos parâmetros do request, ambos como arrays indexados e propriedades do objeto.

Parâmetros do Request

Request Pedido traz várias interfaces para acessar parâmetros de requests.

```
$this->request->params['controller'];
$this->request->param('controller');
```

Ambos devem acessar o mesmo valor. Todos os Route Elements são acessados através desta interface.
http://book.cakephp.org/3.0/en/development/routing.html#route-elements

Em adição você também deve acessar os Argumentos Passados. Estes estão ambos disponíveis no objeto request veja.
http://book.cakephp.org/3.0/en/development/routing.html#passed-arguments

```
// Passed arguments
$this->request->pass;
$this->request['pass'];
$this->request->params['pass'];
```

Parâmetros de Query Strings

`Cake\Network\Request::query(`*$name*`)`

```
// URL is /posts/index?page=1&sort=title
$this->request->query('page');
```

Você pode também acessar diretamente a propriedade da query ou você pode usar o método query() para ler a URL query array em forma de error-free. Qualquer chave que não exista deve retornar null:

```
$foo = $this->request->query('value_that_does_not_exist');
// $foo === null
```

Dados do corpo do Request

`Cake\Network\Request::data(`*$name*`)`

Todos os dados POST podem ser acessados usando Cake\Network\Request::data(). Qualquer dados de formulário que contém um prefixo de dados terá o prefixo de dados removido. Por exemplo:

```
// An input with a name attribute equal to 'MyModel[title]' is accessible at
$this->request->data('MyModel.title');
```

Qualquer chave que não exista deve retornar null:

```
$foo = $this->request->data('Value.that.does.not.exist');
// $foo == null
```

Você pode também acessar os dados do array, como um array:

```
$this->request->data['title'];
$this->request->data['comments'][1]['author'];
```

Dados PIT, PATCH ou DELETE

`Cake\Network\Request::input(`*$callback*`[, `*$options*`])`

Variáveis de Ambiente (de $_SERVER e $_ENV)

`Cake\Network\Request::env(`*$key*`, `*$value* = *null*`)`

```
// Get a value
$value = $this->request->env('HTTP_HOST');

// Set a value. Generally helpful in testing.
$this->request->env('REQUEST_METHOD', 'POST');
```

Checando Condições do Request

Cake\Network\Request::is(*$type*)

O objeto request provém uma maneira fácil de inspecionar certas condições em um dado request. Usando o método is() você pode checar um número de condições comuns, bem como inspecionar outros critérios de uma aplicação específica:

$this->request->is('post');

Receber nome de Host e Domínio

Cake\Network\Request::domain(*$tldLength = 1*)

```
// Prints 'example.org'
echo $request->domain();
```

Cake\Network\Request::subdomains(*$tldLength = 1*)

Retornar o subdomínio

```
// Returns ['my', 'dev'] for 'my.dev.example.org'
$request->subdomains();
```

Cake\Network\Request::host()

retornar o host da applicação:

```
// Prints 'my.dev.example.org'
echo $request->host();
```

Retornar o IP do visitante:
Cake\Network\Request::clientIp()

Receber uma lista de idiomas aceitos:

```
$this->request->acceptLanguage();
```

Checar por um específico idioma aceito:

```
$this->request->acceptLanguage('pt-BR');
```

Response

class Cake\Network\Response

É a classe response default do CakePHP. Ela encapsula várias características e funcionalidades gerando responses HTTP em sua aplicação. Ela também colabora com os testes, como isto pode ser mocked/stubbed permitindo que você inspecione cabeçalhos que devem ser enviados. Como Request, Response também consolida vários

métodos previamente encontrados no Controller, no RequestHandlerComponent e no Dispatcher. O antigo método está em desuso em favor de usar Cake\Network\Response.

Response provê uma interface que incorpora as tarefas comuns relacionadas como:
- Enviar cabeçalhos para redirect
- Enviar conteúdo tipo cabeçalho
- Enviar quaquer cabeçalho
- Enviar o corpo do response

Mudando a Classe Response

CakePHP usa Response por default. Response é uma classe flexível e tansparente. Caso precise sobrescrever ela com sua própria classe de aplicação específica, você pode sobrescrever Response em webroot/index.php.

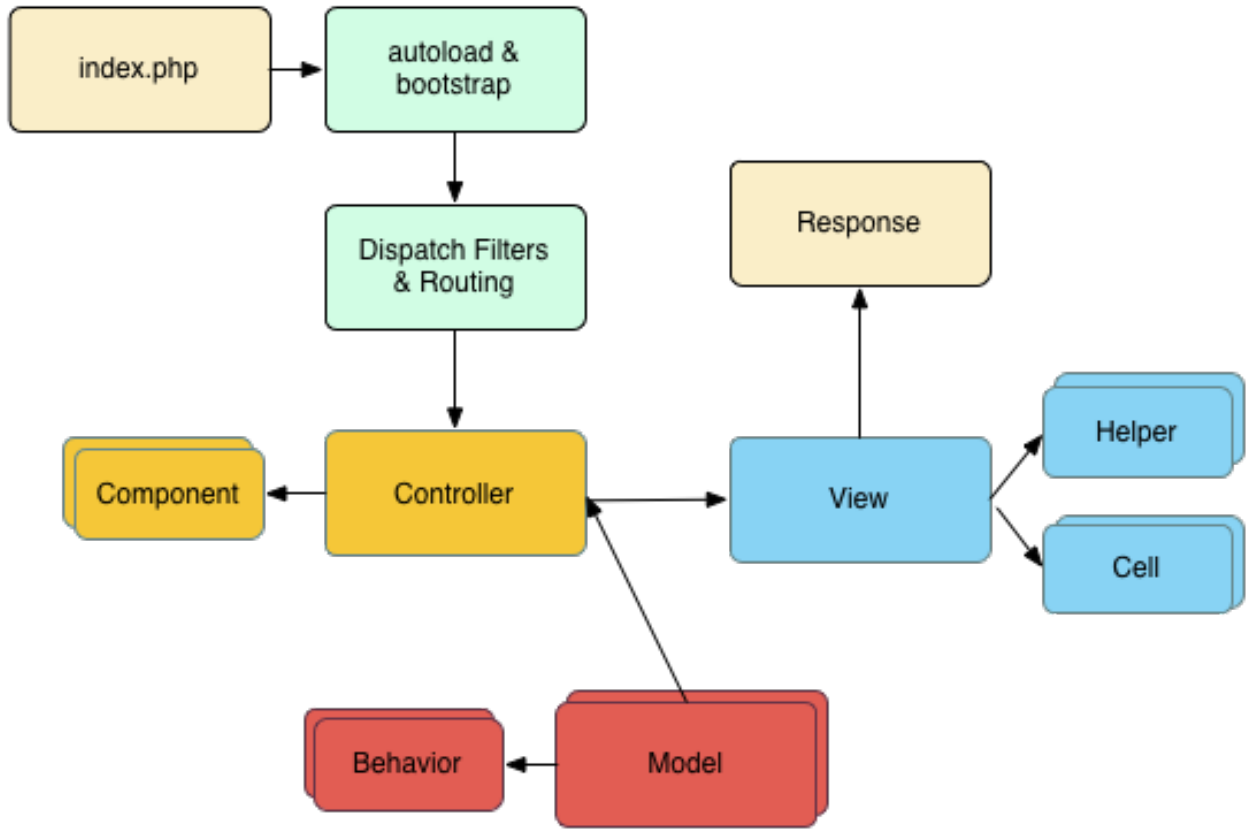
Configurando o Conjunto de Caracteres

```
$this->response->charset('UTF-8');
```

Desabilitando o Cache do Navegador

```
public function index()
{
    // Do something.
    $this->response->disableCache();
}
```

Ciclo do Request no CakePHP



Me parece que este diagrama precisa de alguns ajustes. Veja:
- Entre Controller e Componente a seta deve ser dupla, pois existe comunicação de mão dupla entre ambos.
- Entre Controller e Model da mesma forma
- Entre Model e Behavior do mesmo jeito, mão dupla
- Entre Controller e View me parece que também é mão dupla

Ciclo de Request no CakePHP

No CakePHP este ciclo começa quando o usuário requisita uma página ou recurso em sua aplicação. Cada request segue os seguintes passos:

- O usuário clica num link e o navegador solicita este link do servidor web
- O servidor web reescreve regras diretamente e faz um request para webroot/index.php
- Seu autoloader da aplicação e arquivo de bootstrap são executados
- Qualquer filtro Dispatcher que estiver configurado pode manipular o request e opcionalmente gerar uma resposta/response
- O dispatcher seleciona um apropriado Controller e action baseado nas rules do routing
- O action do Controller é chamado e o Controller interage com o requerido Model e Component
- O Controller delega responsabilidade de criação para a View para gerar a saída resultante dos dados do Model
- A View usa Helpers e Cells para gerar o corpo e cabeçalho da resposta

- A resposta é enviada de volta para o usuário

Mais, bem mais, em:
http://book.cakephp.org/3.0/en/controllers/request-response.html
http://book.cakephp.org/3.0/en/controllers/components/request-handling.html
http://book.cakephp.org/3.0/en/controllers/components/csrf.html

14.3 – Routes

Introdução às Rotas no CakePHP 3

http://book.cakephp.org/3.0/en/development/routing.html
http://book.cakephp.org/3.0/en/development/dispatch-filters.html
http://book.cakephp.org/3.0/en/console-and-shells/routes-shell.html
http://book.cakephp.org/3.0/en/contents.html

class Cake\Routing\Router

Routing oferece a você ferramentas que mapeiam URLs para actions de controllers. Definindo rotas você pode separar como sua aplicação é implementada de como as URLs são estruturadas.

Roteamento em CakePHP também incorpora a ideia de rotas reversas, onde um array de parâmetros pode ser transformado numa string de URL. Usando rotas reversas você pode refatorar a estrutura das URLs da sua aplicação sem ter de atualizar todo o seu código.

O arquivo de configuração das rotas é:
config\routes.php

Veja alguns exemplos de rotas:

```
use Cake\Routing\Router;

// Using the scoped route builder.
Router::scope('/', function ($routes) {
    $routes->connect('/', ['controller' => 'Articles', 'action' => 'index']);
});

// Using the static method.
Router::connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Router oferece duas interfaces para conectar rotas. O método connect é compatível com versões anteriores e o scope oferece mais concisão na sintaxe quando construindo múltiplas rotas e oferece uma melhor performance.

Algumas vezes você precisa de rotas dinâmicas que devam aceitar múltiplos parâmetros, este deve ser o caso, por exemplo de uma rota para visualizar o conteúdo de artigos:

```
Router::connect('/articles/*', ['controller' => 'Articles', 'action' =>
'view']);
```

A rota acima deve aceitar qualquer URL tipo /articles/15 que deve invocar o método view(15) no ArticlesController. Caso você deseje poderá restringir alguns parâmetros de acordo com uma expressão regular:

```
Router::connect(
    '/articles/:id',
```

```
    ['controller' => 'Articles', 'action' => 'view'],
    ['id' => '\d+', 'pass' => ['id']]
);
```

A expressão regular \d+ restringe para somente dígitos.

Acessando Plugins

```
Router::scope('/blog', ['plugin' => 'Blog'], function ($routes) {
    $routes->connect('/', ['controller' => 'Articles']);
});
```

A rota acima deve casar /blog com `Blog\Controller\ArticlesController::index()`.

Conectando Rotas

static `Cake\Routing\Router::connect`(*$route*, *$defaults* =, []*$options* =[])

Para manter seu código DRY você deve usar **scope**. Scope não somente deixa seu código DRY, como também ajuda Router otimizar suas operações. Para criar um scope e conectar algumas rotas devemos usar o método scope():

```
// In config/routes.php
Router::scope('/', function ($routes) {
    $routes->fallbacks('DashedRoute');
});
```

Exemplos práticos:

Veja o caso dos exemplos abaixo, caso existam os dois métodos mostrados, então se chamarmos:

http://localhost/blog

Será chamado o segundo, Articles do blog.

```
$routes->connect('/pages/*', ['controller' => 'Pages', 'action' => 'display', 'home']);

$routes->connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Mas se for assim:

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);

$routes->connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Será chamado o primeiro, Pages, default.

Ou seja, será chamado o primeiro método encontrado, que tenha condições satisfeitas.

Muito mais em:
http://book.cakephp.org/3.0/en/development/routing.html

Routes no CakePHP3

Rota para a pasta raiz do aplicativo

```
    $routes->connect('/', ['controller' => 'Clientes', 'index']);
```

Criando um Controllers

Criar um controller vazio, sem actions:

```
      bin/cake bake controller users --no-actions
```

Assim ele cria uma classe vazia:

```
namespace App\Controller;

use App\Controller\AppController;

class UsersController extends AppController
{
}
```

Criando um router para users:

Editar o config/routes.php

Ao final adicionar:

```
...
    $routes->fallbacks(DashedRoute::class);
});

// Adidionar aqui abaixo:
Router::connect('/users/index', ['controller'=>'Users', 'action' => 'index']);

/**
 * Load all plugin routes.  See the Plugin documentation on
 * how to customize the loading of plugin routes.
 */
Plugin::routes();
```

Criar o action index no controller Users:

```
class UsersController extends AppController
{
      public function index()
```

```
    {
        echo "<h1>Lista de usuários</h1>";
        exit(); // Sem esta linha ele reclama da view respectiva
    }
}
```

Chamar pela url:

pocake/users/index

Veremos
Lista de usuários

Criar rotas usando o método scope, com uma rota principal e outras secundárias:

```
Router::scope('/users', function($routes){
    $routes->connect('index', ['controller'=>'Users', 'action' => 'index']);
    $routes->connect('view', ['controller'=>'Users', 'action' => 'view']);
    $routes->connect('add', ['controller'=>'Users', 'action' => 'add']);
});
```

Passando parâmetros para as rotas:

Na rota

```
    $routes->connect('view/*', ['controller'=>'Users', 'action' => 'view']);
```

no Action

```
    public function view($nome)
    {
        echo "Detalhe de usuário ".$nome;
        exit(); // Sem esta linha ele reclama da view respectiva
    }
```

Pela URL
users/view/ribafs

Rotas customizadas

```
    $routes->connect('lista', ['controller'=>'Users', 'action' => 'index']);
```

Detalhes:
http://book.cakephp.org/3.0/en/development/routing.html

Routes e Controllers

Rotas são pontos de entrada para usuários à nossa aplicação e em geral este ponto de entrada vai representar nossa aplicação

Cada rota aponta a um controller para que futuramente interaja com seus correspondentes actions,
que finalmente gerará uma resposta para o usuário e será no formato HTML com as Views/Templates ou em outro formato.

Rota default:

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

No caso acima temos um action chamado display() e uma view chamada home.

Quando o action chama-se display e a view também chama-se display, temos um route assim:
```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display']);
```

Vejamos como se comunicam as rotas com os controllers.

Editar o routes.php e adicionar ao final, antes de Plugin::routes();
```
Router->connect('/users/index', ['controller' => 'Users', 'action' => 'index']);
```

Criar Controller sem actions:
```
bin\cake bake controller Users --no-actions
```

Então editar o UsersController e adicionar o método index:

```
public function index()
{
    echo "Lista de usuários";
    exit();
}
```

Agora chamamos no navegador:

http://localhost/cliente/users/index

Nos mostrará:
Lista de usuários

Adicionando rota para a View view:

Editar o routes.php e adicionar ao final, antes de Plugin::routes();
Agora vamos usar o método scope de Router

```
Router::scope('/users', function ($routes)
{
```

```
    $routes->connect('/index',(controller' => 'Users', 'action' => 'index'));
    $routes->connect('/view',(controller' => 'Users', 'action' => 'view'));
}
```

Criar Controller sem actions:
bin\cake bake controller Users --no-actions

Então editar o UsersController e adicionar o método index:

```
public function view()
{
    echo "Detalhes de usuário";
    exit();
}
```

Agora chamamos no navegador:

http://localhost/cliente/users/view

Nos mostrará:
Detalhes de usuário

Também podemos passar parâmetros para as rotas:

```
    $routes->connect('/view/*',(controller' => 'Users', 'action' => 'view'));
```

```
public function view($nome)
{
    echo "Detalhes de usuário: ".$nome;
    exit();
}
```

http://localhost/cliente/users/view/ribafs

Detalhes de usuário: ribafs

Documentação
http://book.cakephp.org/3.0/en/development/routing.html

O Cake tem rotas default.
Caso excluamos as duas rotas criadas para inde e view e acessarmos:

http://localhost/cliente/users/index
http://localhost/cliente/users/view/ribafs

Ainda assim funcionará, pois assume as rotas default do Cake.

Assim, basta adicionar o action e chamar pelo navegador:

```
public function add()
{
    echo "Cadastro de usuário";
```

```
    exit();
}
```

http://localhost/cliente/users/add

Passando variáveis pela URL para uma view

Na URL:
localhost/cake-control-demo/pages/home?lang=en&temp=default

No AppController

```
    public function initialize()
    {
        parent::initialize();

        $this->loadComponent('RequestHandler');
        $this->loadComponent('Flash');

            $lang=$this->request->query('lang');
            $temp=$this->request->query('temp');

            $this->set('lang',$lang);
            $this->set('template',$temp);
    }
```

Na View :

```
    <div id="content">
        <div class="row">
            <div class="columns large-12 ctp-warning checks">
                <?php print "<h1>Idioma: ".$lang." Template: ".$template."</h1>"; ?>
```

Links:

http://ribafs.org/cakephp/cake-control-demo/users/login?lang=en&temp=default
http://ribafs.org/cakephp/cake-control-demo/users/login?lang=en&temp=bootstrap
http://ribafs.org/cakephp/cake-control-demo/users/login?lang=pt&temp=default
http://ribafs.org/cakephp/cake-control-demo/users/login?lang=en&temp=bootstrap

15 – Plugins

No CakePHP 3 o plugin é a forma mais adequada de reutilizar código e compartilhar com a comunidade nossos melhores recursos para o Cake.

Basicamente um plugin pode conter um apliicativo a ser usado dentro do Cake como um mini ou sub aplicativo.

Ele pode conter
Controllers
Models
Views/Templates

E dentro destes:
Componentes, Helpers, Behaviors, etc

Então é um super recurso do Cake.

Empacotamos nosso código, hospedamos no GitHub gratuitamente e publicamos no Packagist para que possa ser usado por nós e por toda a comunidade com grande facilidade.

Depois podemos atualizar via git ou no próprio GitHub com os ótimos recursos do mesmo.

Plugin Assets

Os recursos da web de um plugin (mas não arquivos PHP) podem ser atendidos através do plugin no diretório webroot, assim como os assets da aplicação principal:

/vendor/meuVendor/NomePlugin/webroot/
css/
js/
img/
flash/
pdf/

Linking to Assets in Plugins

Você pode usar o sintaxe plugin ao vincular aos recursos do plugin usando o View\Helper\HtmlHelper script, image ou css methods:

```
// Gera a URL /contact_manager/css/styles.css
echo $this->Html->css('ContactManager.styles');

// Gera a URL  /contact_manager/js/widget.js
echo $this->Html->script('ContactManager.widget');
```

```
// Gera a URL /contact_manager/img/logo.jpg
echo $this->Html->image('ContactManager.logo');
```

Chamar Element no layout do plugin

/vendor/ribafs/cake-control-br/src/Template/Layout/default.ctp

```
$this->element('CakeControlBr.topmenu');
```

Carregando um Plugin

Depois de instalar um plugin e configurar o autoloader, você deve carregar O plugin. Você pode carregar plugins um a um, ou todos eles com um único método:

```
// In config/bootstrap.php
// Or in Application::bootstrap()

// Carrega um único plugin
Plugin::load('ContactManager');

// Carrega um plugin com um namespace no nível superior.
Plugin::load('AcmeCorp/ContactManager');

    public function initialize()
    {
        parent::initialize();

        $this->loadComponent('RequestHandler');
        $this->loadComponent('Flash');
            $this->loadComponent('CakeControlBr.Control');

// Carrega todos os plugins de uma só vez
Plugin::loadAll();
```

loadAll() carrega todos os plugins disponíveis, permitindo que você especifique determinadas configurações para plugins. load() funciona de forma semelhante, mas apenas carrega o Plugins que você especifica explicitamente.

Recomenda-se não usar o loadAll().

Plugin::loadAll() não irá carregar os plugins no namespace vendor que não são definidos em vendor/cakephp-plugins.php.

Há também um comando de shell acessível para habilitar o plugin. Execute a seguinte linha:

bin/cake plugin load ContactManager

Isso colocará o plugin Plugin::load('ContactManager'); no bootstrap para você.

Autoloading Plugin Classes

Ao usar bake para criar um plugin ou quando instalar um plugin usando o Composer, você normalmente não precisa fazer alterações em seu aplicativo para faça com que o CakePHP reconheça as classes que vivem dentro dele.

Em qualquer outro caso, você precisará modificar o arquivo do composer.json do seu aplicativo. Para conter as seguintes informações:

```
    "autoload": {
        "psr-4": {
            "CakeControlBr\\": "src"
        }
    }


"psr-4": {
    (...)
    "MyPlugin\\": "./plugins/MyPlugin/src",
    "MyPlugin\\Test\\": "./plugins/MyPlugin/tests"
}
```

Components, Helpers and Behaviors

Um plugin pode ter Components, Helpers e Behaviors, como uma aplicação CakePHP normal. Você pode até criar plugins que consistem apenas em Componentes, Helpers ou Behaviors que podem ser uma ótima maneira de construir componentes reutilizáveis que pode ser lançado em qualquer projeto.

Construir esses componentes é exatamente o mesmo que construí-lo dentro de uma aplicacao normal, sem convenção de nome especial.

Referir-se ao seu componente de dentro ou fora do seu plugin requer apenas que você prefixa o nome do plugin antes do nome do componente. Por exemplo:

```
// Component definido no 'ContactManager' plugin
namespace ContactManager\Controller\Component;

use Cake\Controller\Component;

class ExampleComponent extends Component
{
}

// Dentro de seus controllers
```

```
public function initialize()
{
    parent::initialize();
    $this->loadComponent('ContactManager.Example');
}
```

A mesma técnica se aplica aos Helpers e Behaviors.

Configuração do Plugin

Os métodos load() e loadAll() podem ajudar na configuração do plugin E roteamento. Talvez você queira carregar todos os plugins automaticamente enquanto especifica Rotas personalizadas e arquivos bootstrap para determinados plugins:

```
// No config/bootstrap.php,
// ou in Application::bootstrap()

// Usando loadAll()
Plugin::loadAll([
    'Blog' => ['routes' => true],
    'ContactManager' => ['bootstrap' => true],
    'CakeAclBr' => ['bootstrap' => true],
    'WebmasterTools' => ['bootstrap' => true, 'routes' => true],
]);
```

Ou você pode carregar os plugins individualmente:

```
// Carregando apenas o blog e inclui rotas
Plugin::load('Blog', ['routes' => true]);

    // Inclua o arquivo configuration/initializer do bootstrap.
Plugin::load('CakeAclBr', ['bootstrap' => true]);
```

Plugin Controllers

Os Controllers para o nosso plug-in do ContactManager serão armazenados em plugins/ContactManager/src/Controller/. Como a principal coisa que vamos estar fazendo gerenciar contatos, precisaremos de um ContactsController para este plugin.

Então, colocamos nosso new ContactsController em plugins/ContactManager/src/Controller e parece ser assim:

```
// plugins/ContactManager/src/Controller/ContactsController.php
namespace ContactManager\Controller;

use ContactManager\Controller\AppController;

class ContactsController extends AppController
```

```
{

    public function index()
    {
        //...
    }
}


<?php
namespace CakeAclBr\Controller\Component;

use Cake\Controller\Component;
use Cake\Controller\ComponentRegistry;
use Cake\Datasource\ConnectionManager;

class ControlComponent extends Component
{
    protected $_defaultConfig = [];
```

Criação de Plugin para o CakePHP 3

Irei relatar como criei o plugin twbs-cake-css:
https://github.com/ribafs/twbs-cake-css

Eu criei um plugin que controla a autenticação em aplicativos do Cake, que é o
https://github.com/ribafs/cake-control-br

E neste plugin, para melhorar o aspecto, adicionei o plugin twbs-cake-plugin:
https://github.com/elboletaire/twbs-cake-plugin

Acontece que o twbs-cake-plugin usa less ao invés de css puro e é meio lento. Então resolvi editar o mesmo e criar um que ficasse como eu queria, somente com CSS, mais simples e mais rápido. Resolvi usar o Bootstrap, apenas com CSS.

Baixei o Bootstrap:
http://getbootstrap.com/docs/4.0/getting-started/download/
Eu ainda usei a versão 3, que vinha com as fontes glyphicons.

Baixei o twbs-cake-plugin, que tem a seguinte estrutura:

```
/config
    bootstrap.php
/src
    /Template
        /Bake
            /Element
```

```
            form.ctp
         /Template
            add.ctp
            edit.ctp
            index.ctp
            view.ctp
      /Layout
         default.ctp
/webroot
   /fonts
      glyphicons-halflings-regular.eot
      glyphicons-halflings-regular.svg
      glyphicons-halflings-regular.ttf
      glyphicons-halflings-regular.woff
      glyphicons-halflings-regular.woff2
   /js
      affix.js
      alert.js
      button.js
      carousel.js
      collapse.js
      dropdown.js
      modal.js
      popover.js
      scrollspy.js
      tab.js
      tooltip.js
      transition.js
   /less
      43 arquivos .less e duas pastas: cakephp e mixins contendo diversos arquivos .less
composer.json
LICENSE
readme.md
```

Agora veja como mudei o plugin acima para deixar como eu queria:

Comecei eliminando a pasta config com o arquivo bootstrap.php. Ele chama dois outros plugins que não usarei.

Todo o conteúdo da pasta /src eu usarei como está sem nenhuma modificação, pois ela contém o template para que o bake gere o conteúdo com o CSS, apenas mudarei o layout default.ctp.

Veja como ficou o layout default.ctp:

```
<?php
/**
 * CakePHP(tm) : Rapid Development Framework (http://cakephp.org)
 * Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
```

```
 *
 * Licensed under The MIT License
 * For full copyright and license information, please see the LICENSE.txt
 * Redistributions of files must retain the above copyright notice.
 *
 * @copyright     Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
 * @link          http://cakephp.org CakePHP(tm) Project
 * @since         0.10.0
 * @license       http://www.opensource.org/licenses/mit-license.php MIT License
 */

$cakeDescription = 'CakePHP: the rapid development php framework';
?>
<!DOCTYPE html>
<html>
<head>
    <?= $this->Html->charset() ?>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
        <?= $cakeDescription ?>:
        <?= $this->fetch('title') ?>
    </title>
    <?= $this->Html->meta('icon') ?>

    <?= $this->Html->css('base.css') ?>
    <?= $this->Html->css('bootstrap.min.css') ?>
    <?= $this->Html->script('bootstrap.min') ?>

    <?= $this->fetch('meta') ?>
    <?= $this->fetch('css') ?>
    <?= $this->fetch('script') ?>
</head>
<body>
<style>
/* Remover as duas linhas abaixo, caso o texto do site fique com letras muito grandes */
html { font-size: 14px; }
body { font-size: 14px; }
</style>

    <nav class="top-bar expanded" data-topbar role="navigation">
        <ul class="title-area large-3 medium-4 columns">
            <li class="name">
                <h1><a href=""><?= $this->fetch('title') ?></a></h1>
            </li>
        </ul>
        <div class="top-bar-section">
            <ul class="right">
                <li><a target="_blank"
href="http://book.cakephp.org/3.0/">Documentation</a></li>
                <li><a target="_blank" href="http://api.cakephp.org/3.0/">API</a></li>
```

```
            </ul>
        </div>
    </nav>
    <?= $this->Flash->render() ?>
    <div class="container clearfix"> <!-- Assim <div class="clearfix"> ocupará toda a tela-->
        <?= $this->fetch('content') ?>
    </div>
    <footer>
    </footer>
</body>
</html>
```

Agora vejamos como ficou o conteúdo da pasta /webroot:

```
/css
    bootstrap.min.css
/fonts
    glyphicons-halflings-regular.ttf
/js
    bootstrap.min.js
```

Um arquivo muito importante é o composer.json, que contém informações para a instalação pelo composer:

```
{
    "name": "ribafs/twbs-cake-css",
    "authors": [
        {
            "name": "Ribamar FS",
            "email": "ribafs@gmail.com",
            "homepage": "http://racotecnic.com",
            "role": "Developer"
        }
    ],
    "description": "Twitter Bootstrap Plugin for CakePHP 3",
    "type": "cakephp-plugin",
    "keywords": ["cakephp", "bootstrap", "plugin", "template"],
    "license": "MIT",
    "support": {
        "issues": "https://github.com/ribafs/twbs-cake-css/issues",
        "source": "https://github.com/ribafs/twbs-cake-css.git"
    },
    "homepage": "https://github.com/ribafs/twbs-cake-css",
    "require": {
        "cakephp/cakephp": "^3.3"
    },
    "autoload": {
        "psr-4": {
            "Bootstrap\\": "src"
```

```
    }
  }
}
```

Outro arquivo importante, que no GitHub funciona como o index.html em sites, é o README.md. Ele geralmente contém o help ensinando a instalar e outras informações.

README.md

Simple plugin to implement Bootstrap in CakePHP 3
==========

This plugin is a fork of the Twitter Bootstrap Plugin
https://github.com/elboletaire/twbs-cake-plugin

This plugin only use CSS, dont use Less.

It also contains bake templates that will help you starting *twitter-bootstraped*
CakePHP webapps.

General Features

- Bake templates.
- Generic Bootstrap layout.

Installation

Adding the plugin

You can easily install this plugin using composer as follows:

```bash
composer require ribafs/twbs-cake-css
```

Enabling the plugin

After adding the plugin remember to load it in your `config/bootstrap.php` file:

```php
bin/cake plugin load Bootstrap
```

This will load the CSS for you.

Add Template to src/Controller/AppController.php
```php
```

```
    public function beforeRender(Event $event)
    {
        $this->viewBuilder()->theme('Bootstrap');
    ...
```

Baking views

You can bake your views using the twitter bootstrap templates bundled with this plugin. To do so, simply specify the `bootstrap` template when baking your files:

```bash
bin/cake bake all amigos --theme Bootstrap
```

License

The MIT License (MIT)

Faltou apenas a licença, que mantive a mesma: MIT.

Nos resta agora a pasta /src/, que de fato contém o código do plugin que será usado pelo Cake.

Ele contém os seguintes arquivos e pastas:

src/Template/Bake/Element/form.ctp

src/Template/Bake/Template/add.ctp
src/Template/Bake/Template/edit.ctp
src/Template/Bake/Template/index.ctp
src/Template/Bake/Template/view.ctp

src/Template/Layout/default.ctp

Os cinco primeiro mantive intocados. Apenas o layout default eu substitui por um que adaptei do original do Cake, que mostro abaixo:

```
<?php
// Removi o cabeçalho original apenas aqui
$cakeDescription = 'CakePHP: the rapid development php framework';
?>
<!DOCTYPE html>
<html>
<head>
    <?= $this->Html->charset() ?>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>
        <?= $cakeDescription ?>:
        <?= $this->fetch('title') ?>
    </title>
    <?= $this->Html->meta('icon') ?>

    <?= $this->Html->css('base.css') ?>
    <!-- Abaixo estão as duas linhas do css e js que introduzimos -->
    <?= $this->Html->css('bootstrap.min.css') ?>
    <?= $this->Html->script('bootstrap.min') ?>

    <?= $this->fetch('meta') ?>
    <?= $this->fetch('css') ?>
    <?= $this->fetch('script') ?>
</head>
<body>
<style>
/* Remover estas linhas de style, caso o texto do site fique com letras muito grandes */
html { font-size: 14px; }
body { font-size: 14px; }
</style>

    <nav class="top-bar expanded" data-topbar role="navigation">
        <ul class="title-area large-3 medium-4 columns">
            <li class="name">
                <h1><a href=""><?= $this->fetch('title') ?></a></h1>
            </li>
        </ul>
        <div class="top-bar-section">
            <ul class="right">
                <li><a target="_blank"
href="http://book.cakephp.org/3.0/">Documentation</a></li>
                <li><a target="_blank" href="http://api.cakephp.org/3.0/">API</a></li>
            </ul>
        </div>
    </nav>
    <?= $this->Flash->render() ?>
    <div class="clearfix"> <!-- Alterei esta linha. Caso queira volte ao original, que era
assim: <div class="container clearfix">-->
        <?= $this->fetch('content') ?>
    </div>
    <footer>
    </footer>
</body>
</html>
```

Assim concluímos nosso plugin. Como me pareceu que pode ser útil, abriguei no GitHub e Packagist e está em:
https://github.com/ribafs/twbs-cake-css

DebugKit

https://book.cakephp.org/3.0/pt/debug-kit.html

DebugKit é um plugin suportado pelo time principal do CakePHP que oferece uma barra de ferramentas para auxiliar na depuração de aplicações do CakePHP.

Já vem instalado por default no CakePHP 3

Para usar clique no botão abaixo e à direita, para ver as opções:

Cache
Environment
History
Include
Log
Mail
Packages
Request
Routes
Session
Sql
Log
Timer
Variables

Clique numa opção para ver detalhes

Exemplo: Sql, mostra as consultas que levaram ao que aparece na tela

Plugin Models

Os models para o plugin são armazenados em plugins/ContactManager/src/Model. Nós já definimos um ContactsController para este plugin, então vamos criar a tabela e a entidade para esse controlador:

```
// plugins/ContactManager/src/Model/Entity/Contact.php:
namespace ContactManager\Model\Entity;

use Cake\ORM\Entity;

class Contact extends Entity
{
}

// plugins/ContactManager/src/Model/Table/ContactsTable.php:
namespace ContactManager\Model\Table;
```

```
use Cake\ORM\Table;

class ContactsTable extends Table
{
}
```

Você pode usar TableRegistry para carregar suas tabelas de plugins usando o familiar sintaxe plugin:

```
use Cake\ORM\TableRegistry;

$contacts = TableRegistry::get('ContactManager.Contacts');
```

Alternativamente, a partir de um contexto de controller, você pode usar:

```
$this->loadModel('ContactsMangager.Contacts');
```

No CakePHP 3 temos duas opções de plugin:

aplicativo/plugins
aplicativo/vendor/vendorname/plugin

Instalando um plugin pelo composer

```
cd aplicativo
composer require vendorname/plugin-name
```

Criação de Plugins no CakePHP

Eles ficam na pasta:

app/Plugin

Pode também ficar em aplicativo/plugin

Criação de um plugin bem simples:

app/Plugin/Ola

app/Plugin/Ola/Controller
app/Plugin/Ola/Model
app/Plugin/Ola/View

Para acessar o controller 'teste' do plugin Ola usamos:

http://localhost/clientes/ola/teste

Criando o plugin pela console com:

./cake bake plugin Ola

Ele criará toda a estrutura, mas vazia.
Vamos adicionar conteúdo apenas para mostrar como funciona.

Habilitando o Plugin

Editar o app/Config/bootstrap.php e adicionar ao final:

CakePlugin::load('Ola');

Agora chame com:
http://localhost/clientes/ola/teste

Criação de plugins para o CakePHP3

comp app_control

Criar um plugin:
cd app_control
bin/cake bake plugin AdminApp

Carregar o plugin no bootstrap.php:
bin/cake plugin load AdminApp

Estrutura de um plugin:

```
/src
/plugins
   /ContactManager
      /config
      /src
         /Controller
            /Component
         /Model
            /Table
            /Entity
            /Behavior
         /View
            /Helper
         /Template
            /Layout
      /tests
         /TestCase
         /Fixture
      /webroot
```

Criando um controller para o plugin:

bin/cake bake controller --plugin AdminApp Groups
bin/cake bake controller --plugin AdminApp Users
bin/cake bake controller --plugin AdminApp Permissions

Criando um componente:
bin/cake bake component --plugin AdminApp Control

Criar o banco e configurar no app.php e o routes para Groups/index

Criando um model para o plugin:
bin/cake bake model --plugin AdminApp Groups
bin/cake bake model --plugin AdminApp Users
bin/cake bake model --plugin AdminApp Permissions

Criando templates para o plugin:
bin/cake bake template --plugin AdminApp Groups
bin/cake bake template --plugin AdminApp Users
bin/cake bake template --plugin AdminApp Permissions

Criando a view login.ctp para o plugin:
bin/cake bake template --plugin AdminApp Users login

Chamando o plugin pelo navegador:
http://localhost/app_control/admin-app/groups

Dica: Sempre que criar um plugin regere o autoloader:
php composer.phar dumpautoload

O routes é criado automaticamente.

Criar um helper para o plugin:
bin/cake bake helper --plugin MeuHelper

Criando um Behaviour para o plugin:
bin/cake bake behavior --plugin MeuBehavior

Routes
```
Router::scope('/adminapp', ['plugin' => 'AdminApp'], function ($routes) {
    $routes->fallbacks();
});
```

```
<?php
use Cake\Routing\Router;

Router::plugin('Taxonomy', function($routes) {
      $routes->fallbacks();
```

});

Chamando Plugins

Você pode fazer referência aos controllers, models, components, behaviors, e helpers, prefixando o nome do plugin antes

Por exemplo, vamos supor que você queria usar o plugin do ContactManager ContactInfoHelper para produzir algumas informações de contato legítimas em uma das suas opiniões. No seu controller, o $helpers array poderia ficar assim:

public $helpers = ['ContactManager.ContactInfo'];

Esse nome de classe separado por pontos é denominado sintaxe plugin.

Você poderia então acessar o ContactInfoHelper como qualquer outro helper em sua view, como:

echo $this->ContactInfo->address($contact);

```
if($loguser == 'user' || $loguser == 'manager'){
    $this->viewBuilder()->layout('CakeAclBr.default');
}else{
    $this->viewBuilder()->layout('CakeAclBr.admin');
}
```

Plugin Views

As views se comportam exatamente como ocorrem em aplicações normais. Basta colocá-los na pasta plugins/[PluginName]/src/Template/. Para nós o plugin ContactManager, precisamos de uma view para o nosso ContactsController::index() action, então incluamos isso também:

```
// plugins/ContactManager/src/Template/Contacts/index.ctp:
<h1>Contacts</h1>
<p>Following is a sortable list of your contacts</p>
<!-- A sortable list of contacts would go here....-->
```

Os plugins podem fornecer seus próprios layouts. Para adicionar layouts em plugins, coloque seus arquivos de template dentro plugins/[PluginName]/src/Template/Layout. Para usar um layout de plug-in em seu controller você pode fazer o seguinte:

public $layout = 'ContactManager.admin';

Plugin com controle de acesso para CakePHP 3

Tutorial detalhado em:
http://ribafs.org/portal/cakephp/plugins/cake-control/introducao

Plugin para download no Github
https://github.com/ribafs/cake-control

Crie um novo aplicativo com o cakephp 3 usando o composer:

```
composer create-project --prefer-dist cakephp/app control1
```

Para instalar a versão *estável* do plugin cake-control, execute:

```
cd control1
composer require ribafs/cake-control
```

Habilitar o plugin

cd control1

```
bin/cake plugin load CakeControl --bootstrap
```

Criar banco e configurar.

Assim como as rotas

1) Datas no formato usado no Brasil e bake gerando Users já com login e logout

Faça o download do plugin em
https://github.com/ribafs/cake-control

Descompacte e acesse a pasta docs. Então faça as seguintes cópias:

bootstrap-pt_BR.php para a control1/config/bootstrap.php
bootstrap_cli.php para a control1/config
AppController2.php para control1/src/Controller/AppControler.php
ControlComponent.php para control1/src/Controller/Component

2) Gerar os CRUDs

Edite o src/AppController.php e altere a variável $layout para 'bootstrap.

cd control1

bin/cake bake all groups --theme CakeControl --force
bin/cake bake all users --theme CakeControl --force
bin/cake bake all permissions --theme CakeControl --force
bin/cake bake all customers --theme CakeControl --force

bin/cake bake all products --theme CakeControl --force
bin/cake bake all product_items --theme CakeControl --force

Acesse o aplicativo pela web

Já existem 4 usuários cadastrados como abaixo.

Grupo	Usuário	Senha
Supers	user	super
Admins	admin	admin
Managers	manager	manager
Users	user	user

Internacionalização

Criar a pasta Locale em seu src e dentro dela a pasta pt_BR, assim:
control1/src/Locale/pt_BR

Configurar o locale default:
config/bootstrap.php

Alterar a linha 150 (comente e crie a linha abaixo):
//ini_set('intl.default_locale', Configure::read('App.defaultLocale'));

Crie esta:
ini_set('intl.default_locale', 'pt_BR');

Copiar o arquivo do plugin
vendor/cakephp/localized/src/Locale/pt/cake.po

Para
src/Locale/pt_BR/default.po

Limpar cache do banco

```
bin/cake orm_cache clear
```

Caso seja necessário limpe o cache do navegador.

Exemplo de um Componente para Controle de Acesso/ACL no plugin cake-acl-br:

https://github.com/ribafs/cake-acl-br

16 – Dicas sobre o CakePHP 3

CakePHP 3 CheatSheet
http://cake3.codaxis.com/

Controllers
Controller actions

```
use Cake\Controller\Controller;
class PostsController extends Controller {
    public function view($id) {
        //Your code here
    }
}
```

Before filter

```
use Cake\Event\Event;
public function beforeFilter(Event $event) {
    parent::beforeFilter($event);
}
```

Setting view variables

```
// Setting one variable at a time
$this->set('color', 'pink');
$this->set('color', $color);

// Setting via compact()
$color1 = 'pink';
$color2 = 'red';
$this->set(compact('color1', 'color2'));
```

Retrieving post data

```
// Using $request->data array
$this->request->data['field'];

// Using $request->data() getter
$this->request->data('field');
$this->request->data('data.subfield');
```

Load additional model

```
$this->loadModel('Articles');
```

Redirecting

```
return $this->redirect([
    'controller' => 'myController',
    'action' => 'myAction'
]);
```

Pass variables to the action

```
return $this->redirect([
    'controller' => 'myController',
    'action' => 'myAction',
    $id
]);
```

Redirect to the referer page

```
return $this->redirect($this->referer());
```

View layouts

```
public function index() {
    $this->viewBuilder()->layout('admin'); // New in 3.1
    // or
    $this->layout = 'admin'; // Before 3.1

    //To load a layout from a plugin
    $this->viewBuilder()->layout('Admin.admin'); // New in 3.1
    // or
    $this->layout = 'Admin.admin'; // Before 3.1
}
```

Change the layout of the entire controller

```
//use Cake\Event\Event;
public function beforeFilter(Event $event) {
    parent::beforeFilter($event);
    $this->viewBuilder()->layout('admin'); // New in 3.1
    // or
    $this->layout = 'admin'; // Before 3.1
}
```

Helpers & Components

```
class RecipesController extends AppController {
    public $helpers = ['Form'];
    public $components = ['RequestHandler'];
}
```

Views
View variables

```
//Output a variable
echo $color;
echo $color->name;
```

Setting variables

```
//Views have a set method of their own
$this->set('activeMenuButton', 'posts');
echo $activeMenuButton;
```

View elements

```
echo $this->element('helpbox');

//Passing variables to the view element
echo $this->element('helpbox', [
    'helptext' => 'This is the helptext'
]);

//Load elements from a plugin
echo $this->element('Contacts.helpbox');
```

Page title

```
$this->assign('title', 'Your title');

//In your layout
$this->fetch('title');
```

Models

Tables

```
namespace App\Model\Table;
use Cake\ORM\Table;

class SitiosTable extends Table {
    //Code...
}
```

Attributes

```
class SitiosTable extends Table {
    public function initialize(array $config) {
        $this->table('sitios');
        $this->displayField('myField');
        $this->primaryKey('my_id');
```

```
        $this->tablePrefix('prefix_');
    }
}
```

Associations

```
class SitiosTable extends Table {
    public function initialize(array $config) {
      $this->belongsTo('Authors', [
            'className' => 'Authors',
            'foreignKey' => 'author_id',
            'joinType' => 'INNER',
        ]);
        $this->hasMany('Tags');
        $this->hasOne('Categories');
        $this->hasAndBelongsToMany('Topics');
    }
}
```

Retrieve single row

```
$article = $this->Articles->get($id);
```

Find all

```
$query = $this->Articles->find('all');
```

Filter data

```
$query = $this->Articles->find('all');
$filtered = $query->where(['category_id' => 2);
```

Count results

```
$num = $filtered->count();
```

Full example

```
$articles = $this->Articles->find('all')
        ->where(['category_id' => 1)
        ->order(['created' => 'DESC'])
        ->limit(10);
```

Find first

```
$query = $this->Articles->find('all', [
    'order' => ['Articles.created' => 'DESC']
]);
$row = $query->first();
```

Conditions to contain

```
$query = $articles->find()->contain([
    'Comments' => function ($q) {
       return $q
           ->select(['body', 'author_id'])
           ->where(['Comments.approved' => true]);
    }
]);
```

Validating

```
use Cake\Validation\Validator;
class ArticlesTable extends Table {

    public function validationDefault(Validator $validator) {
       $validator
           ->add('title', [
                'unique' => [
                     'rule' => ['validateUnique', ['scope' => 'title']],
                     'provider' => 'table',
                     'message' => 'You already have an article with that title!'
                ]
               ])
           ->notEmpty('body', 'The body of your article cannot be empty')
           ->notEmpty('title', 'Give your article a title!');
       return $validator;
    }
}
```

Saving data

```
// In your controller
$article = $this->Articles->newEntity($this->request->data);
if ($this->Articles->save($article)) {
    // ...
}
```

Modify a field

```
$article = $this->Articles->find('all')->where(['id' => 2])->first();

$article->title = 'My new title';
$this->Articles->save($article);
```

Bulk update

```
$this->updateAll(['published' => true], ['published' => false]);
```

Deleting data

```
$entity = $this->Articles->find('all')->where(['id' => 2]);
$result = $this->Articles->delete($entity);

//Delete all
$this->Articles->deleteAll(['is_spam' => true]);
```

Emails

Load class

```
use Cake\Network\Email\Email;
```

Basic usage

```
$email = new Email('default');
$email->from(['me@example.com' => 'My Site'])
    ->to('you@example.com')
    ->subject('About')
    ->send('My message');
```

Configuring transport

```
// Sample smtp configuration.
Email::configTransport('gmail', [
    'host' => 'ssl://smtp.gmail.com',
    'port' => 465,
    'username' => 'my@gmail.com',
    'password' => 'secret',
    'className' => 'Smtp'
]);
```

Templated Emails

```
//src/Template/Email/html/welcome.ctp
//src/Template/Layout/Email/html/fancy.ctp
$email->template('welcome', 'fancy')

//Using templates from plugins
$email->template('Blog.new_comment', 'Blog.auto_message');
```

View vars

```
$email->viewVars(['value' => 12345]);

//in your email templates
<p>Here is your value: <b>echo $value</b></p>
```

Attachments

```
$Email->attachments([
    'photo.png' => [
        'file' => '/full/some_hash.png',
        'mimetype' => 'image/png',
        'contentId' => 'my-unique-id'
    ]
]);
```

Quick emails

```
Email::deliver('you@example.com', 'Subject', 'Message', ['from' => 'me@example.com']);
```

Forms

Creating a form

```
// $article should be an empty Article entity if you are in articles/add.
// If you are in articles/edit it should contain the article to be edited.
echo $this->Form->create($article);
```

Finishing a form

```
echo $this->Form->end();
```

Form method - Get

```
echo $this->Form->create($article, ['type' => 'get']);
```

Form method - File

```
// This is still a Post method, but Cake automatically includes proper enctype
echo $this->Form->create($article, ['type' => 'file']);
```

Form action

```
echo $this->Form->create($article, ['action' => 'login']);
```

Form action outside controller

```
// Used when the desired action is outside of the controller
echo $this->Form->create(null, [
    'url' => ['controller' => 'Articles', 'action' => 'publish']
]);
```

Input password

```
echo $this->Form->input('password', ['type' => 'password']);
```

Input file

```
echo $this->Form->input('file', ['type' => 'file']);
```

Input hidden

```
echo $this->hidden('id');
```

Input birthday

```
echo $this->Form->input('birth_dt', [
    'label' => 'Date of birth',
    'dateFormat' => 'DMY',
    'minYear' => date('Y') - 70,
    'maxYear' => date('Y') - 18,
]);
```

Input textarea

```
echo $this->Form->input('notes', ['type' => 'textarea', 'cols' => 10, 'rows' => 2]);
```

Input button

```
echo $this->Form->button('Add', ['class' => 'yourClass']);
```

Input checkbox

```
echo $this->Form->checkbox('published', ['hiddenField' => false]);
```

Input select

```
echo $this->Form->select('field',
    [1, 2, 3, 4, 5],
    ['empty' => '(choose one)']
);
```

Input radio button

```
$options = ['M' => 'Male', 'F' => 'Female'];
$attributes = ['legend' => false];
echo $this->Form->radio('gender', $options, $attributes);
```

Select year

```
echo $this->Form->year('purchased', [
    'minYear' => 2000,
    'maxYear' => date('Y')
]);
```

Select months

```
// Autopopulated with the months of the year. You can pass your own months via the
attribute 'monthNames'
echo $this->Form->month('mob');
```

Select days

```
// Autopopulated with numerical values representing the days of the month
echo $this->Form->day('created');
```

Select hours

```
echo $this->Form->hour('created', [
    'format' => 12
]);

echo $this->Form->hour('created', [
    'format' => 24
]);
```

Select minutes

```
echo $this->Form->minute('created', [
    'interval' => 10
]);
```

HTML Helper

Charset UTF-8

```
echo $this->Html->charset();
```

Charset ISO-8859-1

```
echo $this->Html->charset('ISO-8859-1');
```

Linking to CSS

```
//By default looks for files in webroot/css/
echo $this->Html->css('forms');

//Loading css from plugins
echo $this->Html->css('DebugKit.toolbar.css');
```

Linking to multiple CSS

```
echo $this->Html->css(['forms', 'tables', 'menu']);
```

Favicon

```
echo $this->Html->meta(
    'favicon.ico',
    '/favicon.ico',
    ['type' => 'icon']
);
```

Meta Comments

```
echo $this->Html->meta(
    'Comments',
    '/comments/index.rss',
    ['type' => 'rss']
);
```

Meta Keywords

```
echo $this->Html->meta(
    'keywords',
    'enter any meta keyword here'
);
```

Meta Description

```
echo $this->Html->meta(
    'description',
    'enter any meta description here'
);
```

Doctype

```
echo $this->Html->docType();
```

Linking to images

```
echo $this->Html->image('cake_logo.png', ['alt' => 'CakePHP']);

//To load images from a plugin
echo $this->Html->image('DebugKit.icon.png');
```

Linking to images with full path

```
echo $this->Html->image('logo.png', ['fullBase' => true]);
```

Images as links

```
echo $this->Html->image('recipes/6.jpg', [
    'alt' => 'Brownies',
    'url' => ['controller' => 'Recipes', 'action' => 'view', 6]
```

```
]);
```

Generating links

```
echo $this->Html->link(
    'Enter',
    '/pages/home',
    ['class' => 'button', 'target' => '_blank']
);
```

Generating links w/absolute URL

```
echo $this->Html->link(
    'Dashboard',
    ['controller' => 'Dashboards', 'action' => 'index', '_full' => true]
);
```

Generating links w/confirm message

```
echo $this->Html->link(
    'Delete',
    ['controller' => 'Recipes', 'action' => 'delete', 6],
    ['confirm' => 'Are you sure you wish to delete this recipe?'],
);
```

Generating links w/especial chars

```
echo $this->Html->link(
    $this->Html->image('recipes/6.jpg', ['alt' => 'Brownies']),
    'recipes/view/6',
    ['escape' => false]
);
```

Linking to media

```
echo $this->Html->media('audio.mp3')
//<audio src="/files/audio.mp3"></audio>

echo $this->Html->media('video.mp4', [
     'fullBase' => true,
     'text' => 'Fallback text'
 ])
// Outputs:
// <video src="http://www.somehost.com/files/video.mp4">Fallback text</video>

$this->Html->media(
     ['video.mp4', ['src' => 'video.ogg', 'type' => "video/ogg; codecs='theora, vorbis'"]],
     ['autoplay']
 )
/*
```

```
<video autoplay="autoplay">
    <source src="/files/video.mp4" type="video/mp4"/>
    <source src="/files/video.ogg" type="video/ogg;
       codecs='theora, vorbis'"/>
 </video>
*/
```

Linking to javascript

```
$this->Html->script('scripts');
```

Linking to multiple javascript

```
echo $this->Html->script(['jquery', 'wysiwyg', 'scripts']);
```

Javascript to specific block

```
echo $this->Html->script('wysiwyg', ['block' => 'scriptBottom']);
//in your layout
echo $this->fetch('scriptBottom');
```

Nested Lists

```
$list = [
   'Languages' => [
      'English' => [
         'American',
         'Canadian',
         'British',
      [,
      'Spanish',
      'German',
   ]
];
echo $this->Html->nestedList($list);
/*
<ul>
   <li>Languages
      <ul>
         <li>English
            <ul>
               <li>American</li>
               <li>Canadian</li>
               <li>British</li>
            </ul>
         </li>
         <li>Spanish</li>
         <li>German</li>
      </ul>
   </li>
```

```
</ul>

*/
```

Table headers

```
echo $this->Html->tableHeaders(['Date', 'Title', 'Active']);
/*
<tr>
    <th>Date</th>
    <th>Title</th>
    <th>Active</th>
</tr>
*/

echo $this->Html->tableHeaders(
    ['Date','Title','Active'],
    ['class' => 'status'],
    ['class' => 'product_table']
);
/*
<tr class="status">
     <th class="product_table">Date</th>
     <th class="product_table">Title</th>
     <th class="product_table">Active</th>
</tr>
*/

echo $this->Html->tableHeaders([
    'id',
    ['Name' => ['class' => 'highlight']],
    ['Date' => ['class' => 'sortable']]
]);
/*
<tr>
    <th>id</th>
    <th class="highlight">Name</th>
    <th class="sortable">Date</th>
</tr>
*/
```

Creating table cells

```
echo $this->Html->tableCells([
    ['Jul 7th, 2007', 'Best Brownies', 'Yes'],
    ['Jun 21st, 2007', 'Smart Cookies', 'Yes'],
    ['Aug 1st, 2006', 'Anti-Java Cake', 'No'],
]);
/*
<tr><td>Jul 7th, 2007</td><td>Best Brownies</td><td>Yes</td></tr>
```

```
<tr><td>Jun 21st, 2007</td><td>Smart Cookies</td><td>Yes</td></tr>
<tr><td>Aug 1st, 2006</td><td>Anti-Java Cake</td><td>No</td></tr>
*/

echo $this->Html->tableCells([
    ['Jul 7th, 2007', ['Best Brownies', ['class' => 'highlight']] , 'Yes'],
    ['Jun 21st, 2007', 'Smart Cookies', 'Yes'],
    ['Aug 1st, 2006', 'Anti-Java Cake', ['No', ['id' => 'special']]],
]);
/*
<tr>
    <td>
        Jul 7th, 2007
    </td>
    <td class="highlight">
        Best Brownies
    </td>
    <td>
        Yes
    </td>
</tr>
<tr>
    <td>
        Jun 21st, 2007
    </td>
    <td>
        Smart Cookies
    </td>
    <td>
        Yes
    </td>
</tr>
<tr>
    <td>
        Aug 1st, 2006
    </td>
    <td>
        Anti-Java Cake
    </td>
    <td id="special">
        No
    </td>
</tr>
*/
```

Paginator

Setting up

```
class ArticlesController extends AppController {
    public $components = ['Paginator'];
    public $paginate = [
        'limit' => 25,
        'order' => [
            'Articles.title' => 'asc'
        ]
    ];
}
```

Finding data

```
public function index() {
    $this->set('articles', $this->paginate());
}
public function index() {
    $query = $this->Articles->find('popular')->where(['author_id' => 1]);
    $this->set('articles', $this->paginate($query));
}
```

Using the paginator directly

```
$articles = $this->Paginator->paginate($articleTable, $config);
```

Maxlimit

```
// Limit the maximum number of rows that can be fetched (default: 100)
public $paginate = [
    'maxLimit' => 10
];
```

Paginator Helper - Sort Links

```
echo $this->Paginator->sort('user_id', 'User account');
```

Paginator Helper - Page numbers

```
echo $this->Paginator->numbers();
```

Paginator Helper - Page numbers limit

```
// In the example, display links to the first 2 pages and the last 2
echo $this->Paginator->numbers(['first' => 2, 'last' => 2]);
```

Paginator Helper - Previous, Next

```
echo $this->Paginator->prev(' << ' . __('previous'));
echo $this->Paginator->next(' >> ' . __('next'));
```

Paginator Helper - First, Last page

```
echo $this->Paginator->first('< first');
echo $this->Paginator->last('> last');
```

Routing

Basic usage

```
Router::connect('/articles/*', ['controller' => 'Articles', 'action' => 'view']);
```

Passing variables to the controller

```
Router::connect(
    '/articles/:id',
    ['controller' => 'Articles', 'action' => 'view'],
    ['id' => '\d+', 'pass' => ['id']]
);
```

Onde usar os inserts de CSS e JavaScript?

No topo da view

```
echo $this->Html->script(array('ajaxupload.3.5.js'));

echo $this->Html>css(array('new_layout.css'));

<script type="text/javascript">
function filldetails()
{
    document.getElementById('FirstName').value  = "hjshjsh";
}
</script>
echo $this->Form->select('grupos',['onblur' =>grupo());
```

Foco com HTML5

Foco automático em um campo

```
echo $this->Form->input('grupo',['style'=>'width: 100px', 'type'=>'text', 'autofocus']);
```

JavaScript

```
<script>
    (function(){
        var forms = document.forms || [];
        for(var i = 0; i < forms.length; i++){
            for(var j = 0; j < forms[i].length; j++){
                if(!forms[i][j].readonly != undefined && forms[i][j].type != "hidden" && forms[i]
[j].disabled != true && forms[i][j].style.display != 'none'){
                    forms[i][j].focus();
                    return;
                }
            }
        }
    })();
</script>
```

Atualização de Aplicativo

Após instalar um aplicativo podemos atualizar o cake e algum plugin com:

Acessar o diretório e executar

composer update

Implementando Busca em Forms do CakePHP 3

Adaptação de:
http://www.tayron.com.br/blog/121/criando-um-formulario-de-pesquisa-com-cakephp3

O exemplo aqui apresentado foi com um aplicativo criado através do bake.
Apenas duas tabelas:

```
CREATE TABLE IF NOT EXISTS `groups` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL,
  `created` datetime DEFAULT NULL,
  `modified` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) NOT NULL,
  `password` char(255) NOT NULL,
  `group_id` int(11) NOT NULL,
  `created` datetime DEFAULT NULL,
  `modified` datetime DEFAULT NULL,
```

```
  PRIMARY KEY (`id`),
  UNIQUE KEY `username` (`username`)
);
```

Banco - busca
Diretório - busca

Os dados do formulário deve ser enviado via query string (get).
E a action do formulário deve sempre apontas para o método index, exemplo:
www.site.com.br/users/index.

View Users/index.ctp

Entendido as duas regras acima vamos criar nosso formulário na nossa view
antes da tag <table>:

```
<?php
    echo $this->Form->create(null, ['type' => 'get']);

    echo $this->Form->input('search',
      ['class' => 'form-control', 'label' => false,
      'placeholder' => 'Digite o username',
      'value' => $this->request->query('search')]);

    echo $this->Form->button('Pesquisar');
    echo $this->Form->end();
?>

<hr />
```

Substitua o método index() existente do Controller/UserController.php por:

```
public function index()
{
    $this->paginate = [
        'contain' => ['Groups'],
        'conditions' => ['and' => [
            //'Users.people like' => '%' . $this->request->query('search') . '%',
            'Users.username like' => '%' . $this->request->query('search') . '%'
        ]],
        'order' => ['Users.id' => 'DESC' ]
    ];

    $this->set('users', $this->paginate($this->Users));
    $this->set('_serialize', ['users']);
}
```

```
<?php
namespace App\Model\Table;

use Cake\ORM\Query;
use Cake\ORM\RulesChecker;
use Cake\ORM\Table;
use Cake\Validation\Validator;

class UsersTable extends Table
{
    public function initialize(array $config)
    {
        parent::initialize($config);

        $this->table('users'); // Name of the table in the database, if absent convention
assumes lowercase version of file prefix
        $this->displayField('full_name'); // field or virtual field used for default display in
associated models, if absent 'id' is assumed
        $this->primaryKey('id'); // Primary key field(s) in table, if absent convention assumes
'id' field

        $this->addBehavior('Timestamp'); // Allows your model to timestamp records on
creation/modification
    }
}
```

Entity com suporte Hash Bcrypt de Senha

```
<?php
namespace App\Model\Entity;

use Cake\ORM\Entity;
use Cake\Auth\DefaultPasswordHasher;

class User extends Entity
{
    // Fields that can be mass assigned using newEntity() or patchEntity();
    protected $_accessible = [
        '*' => true,
        'id' => false
    ];
    protected $_hidden = [ 'password' ]; // Fields excluded from JSON versions of the entity
    // Hash the user's password before saving and before validation
    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($value);
    }
}
```

Customizar paginação

```
class UsersController extends AppController
{
    private $roles = ['User' => 'User', 'Admin' => 'Admin', 'Disabled' => 'Disabled'];
    public $paginate = [
        'limit' => 30,
      'order' => ['first_name' => 'ASC', 'last_name' => 'ASC']
    ];
    public function index() {
        $users = $this->paginate($this->Users);
        $this->set(compact('users'));
        $this->set('_serialize', ['users']);
    }
```

config/bootstrap.php

```
Cake\I18n\Time::setToStringFormat('Y-MM-dd hh:mm a');
Cake\I18n\FrozenTime::setToStringFormat('Y-MM-dd hh:mm a');
Cake\I18n\Date::setToStringFormat('Y-MM-dd');
Cake\I18n\FrozenDate::setToStringFormat('Y-MM-dd');
```

Add Password and Reset Functions

When a user forgets their password, we need a method for them to reset it. We can verify them by email, send them an email to that verified address, and allow them to reset their password.

At the top of the Users Controller add:

```
use Cake\Routing\Router;
use Cake\Mailer\Email;
```

Then add the following functions:

```
    public function password()
    {
        if ($this->request->is('post')) {
            $query = $this->Users->findByEmail($this->request->data['email']);
            $user = $query->first();
            if (is_null($user)) {
                $this->Flash->error('Email address does not exist. Please try again');
            } else {
                $passkey = uniqid();
                $url = Router::Url(['controller' => 'users', 'action' => 'reset'], true) . '/' . $passkey;
                $timeout = time() + DAY;
                 if ($this->Users->updateAll(['passkey' => $passkey, 'timeout' => $timeout], ['id'
=> $user->id])){
                    $this->sendResetEmail($url, $user);
```

```
                $this->redirect(['action' => 'login']);
            } else {
                $this->Flash->error('Error saving reset passkey/timeout');
            }
        }
    }
}

private function sendResetEmail($url, $user) {
    $email = new Email();
    $email->template('resetpw');
    $email->emailFormat('both');
    $email->from('no-reply@naidim.org');
    $email->to($user->email, $user->full_name);
    $email->subject('Reset your password');
    $email->viewVars(['url' => $url, 'username' => $user->username]);
    if ($email->send()) {
        $this->Flash->success(__('Check your email for your reset password link'));
    } else {
        $this->Flash->error(__('Error sending email: ') . $email->smtpError);
    }
}

public function reset($passkey = null) {
    if ($passkey) {
        $query = $this->Users->find('all', ['conditions' => ['passkey' => $passkey, 'timeout >'
=> time()]]);
        $user = $query->first();
        if ($user) {
            if (!empty($this->request->data)) {
                // Clear passkey and timeout
                $this->request->data['passkey'] = null;
                $this->request->data['timeout'] = null;
                $user = $this->Users->patchEntity($user, $this->request->data);
                if ($this->Users->save($user)) {
                    $this->Flash->set(__('Your password has been updated.'));
                    return $this->redirect(array('action' => 'login'));
                } else {
                    $this->Flash->error(__('The password could not be updated. Please, try
again.'));
                }
            }
        } else {
            $this->Flash->error('Invalid or expired passkey. Please check your email or try
again');
            $this->redirect(['action' => 'password']);
        }
        unset($user->password);
        $this->set(compact('user'));
    } else {
```

```
            $this->redirect('/');
        }
    }
```

2: Add Reset and Password Views

To utilize these methods we need the accompanying views.

src/Template/Users/password.ctp displays a simple form for the user to enter their email.

```
<?php $this->assign('title', 'Request Password Reset'); ?><div class="users content">
      <h3><?php echo __('Forgot Password'); ?></h3>
      <?php
      echo $this->Form->create();
       echo $this->Form->input('email', ['autofocus' => true, 'label' => 'Email address',
'required' => true]);
            echo $this->Form->button('Request reset email');
      echo $this->Form->end();
      ?>
</div>
```

src/Template/Users/reset.ctp displays a simple form for the user to change their password after their passkey and token have been verified.

```
<?php $this->assign('title', 'Reset Password'); ?>
<div class="users form large-9 medium-8 columns content">
    <?php echo $this->Form->create($user) ?>
    <fieldset>
        <legend><?php echo __('Reset Password') ?>
    <?php
        echo $this->Form->input('password', ['required' => true, 'autofocus' => true]); ?>
        <p class="helper">Passwords must be at least 8 characters and contain at least 1
number, 1 uppercase, 1 lowercase and 1 special character</p>
    <?php
        echo $this->Form->input('confirm_password', ['type' => 'password', 'required' =>
true]);
    ?>
    </fieldset>
      <?php echo $this->Form->button(__('Submit')); ?>
    <?php echo $this->Form->end(); ?>
</div>
```

3: Create Email Templates

For the sendResetEmail() function we need to create the email templates that display the two variables we are sending, and since we're sending both plain text and html formats, we need to create a template for each.

src/Template/Email/html/resetpw.ctp

```
<p>Your username is <?php echo $username; ?></p>

<p>Click on the link below to Reset Your Password.</p>

<p><a href="<?php echo $url; ?>">Click here to Reset Your Password</a></p>

<pre>or Visit this Link</pre><br/>
<p><a href=""><?php echo $url; ?></a></p>
```

src/Template/Email/text/resetpw.ctp

```
Your username is <?php echo $username; ?>

Click on the link below or copy and paste it into your web browser to reset your password:

<?php echo $url; ?>
```

http://www.naidim.org/cakephp-3-tutorial-9-reset-password

Dicas de CakePHP

action index resumido

```
public function index()
{
    $this->set('posts', $this->paginate($this->Posts));
}
```

ou

```
$this->set('categories', $this->Despesas->find('all'));
```

action view

```
public function view($id = null)
{
    $post = $this->Posts->get($id);
    $this->set('posts', $posts);
}
```

action add

```
  function add() {
    if (!empty($this->data)) {
      if ($this->Category->save($this->data)) {
        $this->Session->setFlash('Your category has been saved.');
        $this->redirect(array('action' => 'index'));
      }
    }
  }
```

Alterar combo
Que mostra id para mostrar o campo descrição
No Caso o campo Roles do Users/add, relacionado com Roles

Apenas acessar o RolesTable.php
e mudar o DisplayField() de id para 'role'.

Autenticação de e-mail
https://www.youtube.com/watch?v=cEwf9PpbMcQ

config/app.php

```
...
    'EmailTransport' => [
        'default' => [
            'className' => 'Mail',
            // The following keys are used in SMTP transports
            'host' => 'localhost',
            'port' => 25,
            'timeout' => 30,
            'username' => 'user',
            'password' => 'secret',
            'client' => null,
            'tls' => null,
            'url' => env('EMAIL_TRANSPORT_DEFAULT_URL', null),
        ],
            'gmail' => [
                'className' => 'Smtp',
                'host' => 'ssl://smtp.gmail.com',
                'port' => 465,
                'timeout' => 30,
                'username' => 'ribafs@gmail.com',
                'password' => 'zmxn1029g',
                'client' => null,
                'tls' => null,
                'url' => env('EMAIL_TRANSPORT_DEFAULT_URL', null)
    ],
```

Profile

```
'Email' => [
        'default' => [
            'transport' => 'default',
            'from' => 'you@localhost',
            //'charset' => 'utf-8',
            //'headerCharset' => 'utf-8',
        ],
            'ribaportal' => [
                'transport' => 'gmail',
```

```
            'from' => ['ribafs@gmail.com'=>'Portal do RibaFS'],
        ],
    ],
```

Criar classe mailler usando o bake

bin/cake bake mailer user

Criar classes no controller e no model

Abrir src/Controller/Mailer/UserMailer.php

Adicionar após a linha
static ...

```
    public function welcome($user)
    {
        $this->to($user->email)
        ->profile('ribaportal')
        ->emailFormat('html')
        ->template('welcome_email_template')
        ->layout('default')
        ->viewVars(['nome' => $user->name])
        ->subject(sprintf('Bem vindo, %s', $user->name));
    }
```

Criar o template para o e-mail

src/Template/Email/html/welcome_email_template.ctp

```
<h1>Seja bem-vindo(a), <?php echo $nome;?></h1>
<p>
    Apenas um teste de envio de e-mail pelo CakePHP.
</p>
```

Para testar criei um novo usuário e veja se ele enviou o e-mail.

Recuperação de senha

https://www.youtube.com/watch?v=cEwf9PpbMcQ

Exemplos Práticos com o CakePHP 3

https://www.tutorialspoint.com/cakephp/index.htm

Exemplo 1 - Routing

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/', ['controller' => 'Tests', 'action' => 'index']);
    $routes->connect('/pages/*', ['controller' => 'Pages', 'action' => 'display']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/TestsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class TestsController extends AppController{
    public function index(){
    }
  }
?>
```

src/Template/Tests/index.ctp

This is CakePHP tutorial and this is an example of connecting routes.

http:/localhost/exemplos

Passando argumentos

Adicionar a linha ao routes.php:
$routes->connect('tests/:arg1/:arg2', ['controller' => 'Tests', 'action' =>'index'],['pass' => ['arg1', 'arg2']]);

Visite:

http:/localhost/exemplos/tests/value1/value2

Exemplo 2 - Passando argumentos

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('tests/:arg1/:arg2', ['controller' => 'Tests', 'action'=>'index'],['pass'
=>['arg1', 'arg2']]);

    $routes->connect('/pages/*', ['controller' => 'Pages', 'action' => 'display']);
    $routes->fallbacks('DashedRoute');
  });

  Plugin::routes();
```

src/Controller/TestsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class TestsController extends AppController{
    public function index($arg1,$arg2){
      $this->set('argument1',$arg1);
      $this->set('argument2',$arg2);
    }
  }
?>
```

src/Template/Tests/index.ctp

```
This is CakePHP tutorial and this is an example of Passed arguments.<br />
Argument-1: <?=$argument1?><br />
Argument-2: <?=$argument2?><br />
```

Visite
http:/localhost/exemplos/tests/Ribamar/Sousa

Exemplo 3 - Gerando URLs
config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
```

```
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes){
    $routes->connect('/generate',['controller'=>'Generates','action'=>'index']);
  });

  Plugin::routes();
```

src/Controller/GeneratesController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;

  class GeneratesController extends AppController{
    public function index(){
    }
  }
?>
```

src/Template/Generates/index.ctp

This is CakePHP tutorial and this is an example of Generating URLs.

Visite
http://localhost/exemplos/generate

Exemplo 4 - redirect Routing

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/generate2', ['controller' => 'Tests', 'action' => 'index']);
    $routes->redirect('/generate1','http://tutorialspoint.com/');
    $routes->connect('/generate_url',['controller'=>'Generates','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

Visite:

URL 1 — http://localhost/exemplos/generate_url
URL 2 — http://localhost/exemplos/generate1
URL 3 — http://localhost/exemplos/generate2

Exemplo 5 - Controllers

src/Controller/AppController.php

```
<?php
  namespace App\Controller;
  use Cake\Controller\Controller;
  use Cake\Event\Event;

  class AppController extends Controller{
    public function initialize(){
      parent::initialize();
      $this->loadComponent('RequestHandler');
      $this->loadComponent('Flash');
    }
    public function beforeRender(Event $event){
      if (!array_key_exists('_serialize', $this->viewVars) &&
        in_array($this->response->type(), ['application/json', 'application/xml'])) {
        $this->set('_serialize', true);
      }
    }
  }
```

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/redirect-controller',['
      controller'=>'Redirects','action'=>'action1']);

    $routes->connect('/redirectcontroller2',['
      controller'=>'Redirects','action'=>'action2']);

    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/RedirectsController.php

```
<?php
```

```
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;

  class RedirectsController extends AppController{
    public function action1(){
    }
    public function action2(){
      echo "redirecting from action2";
      $this->setAction('action1');
    }
  }
?>
```

src/Template/Redirects/action1.ctp

This is an example of how to redirect within controller.

Visite
http://localhost/exemplos/redirect-controller

http://localhost/exemplos/redirect-controller2

Exemplo 6 - Views

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('template',['controller'=>'Products','action'=>'view']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/ProductsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class ProductsController extends AppController{
    public function view(){
      $this->set('Product_Name','XYZ');
```

```
    }
  }
?>
```

src/Template/Products/view.ctp

```
Value of variable is: <?php echo $Product_Name; ?>
```

Visite

http://localhost/exemplos/template

Exemplo 7 - Extendendo Views

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('extend',['controller'=>'Extends','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/ExtendsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class ExtendsController extends AppController{
    public function index(){
    }
  }
?>
```

src/Template/Extends/header.ctp

```
<div align = "center"><h1>Common Header</h1></div>
<?= $this->fetch('content') ?>
```

src/Template/Extends/index.ctp

```
<?php $this->extend('header'); ?>
This is an example of extending view.
```

Visite
http://localhost/exemplos/extend

Exemplo 8 - Element

src/Template/Element/helloworld.ctp

```
<p>Hello World</p>
```

src/Template/Elems/index.ctp

```
Element Example: <?php echo $this->element('helloworld'); ?>
```

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/element',['controller'=>'Elems','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/ElemsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;

  class ElemsController extends AppController{
    public function index(){
    }
  }
?>
```

Visite
http://localhost/exemplos/element-example

Operações com Registros

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
)
ENGINE = InnoDB AUTO_INCREMENT = 7 DEFAULT CHARSET = latin1
```

Exemplo 9 - Inserir um registros

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/users/add', ['controller' => 'Users', 'action' => 'add']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/controller/UsersController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;
  use Cake\Auth\DefaultPasswordHasher;

  class UsersController extends AppController{

    public function add(){
      if($this->request->is('post')){
        $username = $this->request->data('username');
        $hashPswdObj = new DefaultPasswordHasher;
        $password = $hashPswdObj->hash($this->request->data('password'));
        $users_table = TableRegistry::get('users');
        $users = $users_table->newEntity();
        $users->username = $username;
        $users->password = $password;

        if($users_table->save($users))
        echo "User is added.";
```

```
      }
    }
  }
?>
```

src/Template/Users/add.ctp

```
<?php
  echo $this->Form->create("Users",array('url'=>'/users/add'));
  echo $this->Form->input('username');
  echo $this->Form->input('password');
  echo $this->Form->button('Submit');
  echo $this->Form->end();
?>
```

Visite
http://localhost/exemplos/users/add

Exemplo 10 - Visualizar registro

Para visualizar um registro antes precisamos registrar o objeto TableRegistry.

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/users', ['controller' => 'Users', 'action' => 'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/controller/UsersController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;

  class UsersController extends AppController{

    public function index(){
      $users = TableRegistry::get('users');
      $query = $users->find();
```

```
        $this->set('results',$query);
      }
    }
?>
```

src/Template/Users/index.ctp

```
<a href = "Users/add">Add User</a>
<table>
  <tr>
    <td>ID</td>
    <td>Username</td>
    <td>Password</td>
    <td>Edit</td>
    <td>Delete</td>
  </tr>

  <?php
    foreach ($results as $row):
      echo "<tr><td>".$row->id."</td>";
      echo "<td>".$row->username."</td>";
      echo "<td>".$row->password."</td>";
      echo "<td><a href = '".$this->Url->build
      (["controller" => "Users","action"=>"edit",$row->id])."'>Edit</a></td>";

      echo "<td><a href = '".$this->Url->build
      (["controller" => "Users","action"=> "delete",$row->id])."'>Delete</a></td></tr>";
    endforeach;
  ?>
</table>
```

Visite
http://localhost/exemplos/users

Exemplo 11 - Atualizar registro

Para atualizar um registro antes precisamos registrar o objeto TableRegistry.

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/users/edit', ['controller' => 'Users', 'action' => 'edit']);
    $routes->fallbacks('DashedRoute');
  });
```

```
  Plugin::routes();
```

src/controller/UsersController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;

  class UsersController extends AppController{
    public function index(){
      $users = TableRegistry::get('users');
      $query = $users->find();
      $this->set('results',$query);
    }
    public function edit($id){
      if($this->request->is('post')){
        $username = $this->request->data('username');
        $password = $this->request->data('password');
        $users_table = TableRegistry::get('users');
        $users = $users_table->get($id);
        $users->username = $username;
        $users->password = $password;

        if($users_table->save($users))
        echo "User is udpated";
        $this->setAction('index');
      } else {
        $users_table = TableRegistry::get('users')->find();
        $users = $users_table->where(['id'=>$id])->first();
        $this->set('username',$users->username);
        $this->set('password',$users->password);
        $this->set('id',$id);
      }
    }
  }
?>
```

src/Template/Users/index.ctp

```
<a href = "Users/add">Add User</a>
<table>
  <tr>
    <td>ID</td>
    <td>Username</td>
    <td>Password</td>
    <td>Edit</td>
    <td>Delete</td>
  </tr>
```

```
  <?php
    foreach ($results as $row):
      echo "<tr><td>".$row->id."</td>";
      echo "<td>".$row->username."</td>";
      echo "<td>".$row->password."</td>";
      echo "<td><a href = '".$this->Url->build
        (["controller" => "Users","action" => "edit",$row->id]).
        "'>Edit</a></td>";
      echo "<td><a href = '".$this->Url->build
        (["controller" => "Users","action" => "delete",$row->id]).
        "'>Delete</a></td></tr>";
    endforeach;
  ?>
</table>
```

src/Template/Users/edit.ctp

```
<?php
  echo $this->Form->create("Users",array('url'=>'/users/edit/'.$id));
  echo $this->Form->input('username',['value'=>$username]);
  echo $this->Form->input('password',['value'=>$password]);
  echo $this->Form->button('Submit');
  echo $this->Form->end();
?>
```

Visite
http://localhost/exemplos/users

Exemplo 12 - Excluir um registro

Para excluir um registro antes precisamos registrar o objeto TableRegistry.

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/users/delete', ['controller' => 'Users', 'action' => 'delete']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/controller/UsersController.php

```
<?php
```

```
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;

  class UsersController extends AppController{
    public function index(){
      $users = TableRegistry::get('users');
      $query = $users->find();
      $this->set('results',$query);
    }
    public function delete($id){
      $users_table = TableRegistry::get('users');
      $users = $users_table->get($id);
      $users_table->delete($users);
      echo "User deleted successfully.";
      $this->setAction('index');
    }
  }
?>
```

src/Template/Users/index.ctp

```
<a href = "Users/add">Add User</a>
<table>
  <tr>
    <td>ID</td>
    <td>Username</td>
    <td>Password</td>
    <td>Edit</td>
    <td>Delete</td>
  </tr>

  <?php
    foreach ($results as $row):
      echo "<tr><td>".$row->id."</td>";
      echo "<td>".$row->username."</td>";
      echo "<td>".$row->password."</td>";
      echo "<td><a href='".$this->Url->build(["controller" => "Users","action" => "edit",$row-
>id])."'>Edit</a></td>";

      echo "<td><a href='".$this->Url->build(["controller" => "Users","action" -> "delete",
$row->id])."'>Delete</a></td></tr>";
    endforeach;
  ?>
</table>
```

Visite
http://localhost/exemplos/users

Exemplo 13 - Serviços - FormAuthentication

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/auth',['controller'=>'Authexs','action'=>'index']);
    $routes->connect('/login',['controller'=>'Authexs','action'=>'login']);
    $routes->connect('/logout',['controller'=>'Authexs','action'=>'logout']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/AppController.php

```
<?php
  namespace App\Controller;
  use Cake\Controller\Controller;
  use Cake\Event\Event;
  use Cake\Controller\Component\AuthComponent;

  class AppController extends Controller{
    public function initialize(){
      parent::initialize();

      $this->loadComponent('RequestHandler');
      $this->loadComponent('Flash');
      $this->loadComponent('Auth', [
        'authenticate' => [
          'Form' => [
            'fields' => ['username' => 'username', 'password' => 'password']
          ]
        ],
        'loginAction' => ['controller' => 'Authexs', 'action' => 'login'],
        'loginRedirect' => ['controller' => 'Authexs', 'action' => 'index'],
        'logoutRedirect' => ['controller' => 'Authexs', 'action' => 'login']
      ]);

      $this->Auth->config('authenticate', [
        AuthComponent::ALL => ['userModel' => 'users'], 'Form']);
    }

    public function beforeRender(Event $event){
      if (!array_key_exists('_serialize', $this=>viewVars) &&
      in_array($this->response=>type(), ['application/json', 'application/xml'])) {
```

```
            $this->set('_serialize', true);
        }
      }
    }
```

src/Controller/AuthexsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\ORM\TableRegistry;
  use Cake\Datasource\ConnectionManager;
  use Cake\Event\Event;
  use Cake\Auth\DefaultPasswordHasher;

  class AuthexsController extends AppController{
    var $components = array('Auth');
    public function index(){
    }
    public function login(){
      if($this->request->is('post')){
        $user = $this->Auth->identify();

        if($user){
          $this->Auth->setUser($user);
          return $this->redirect($this->Auth->redirectUrl());
        } else
        $this->Flash->error('Your username or password is incorrect.');
      }
    }
    public function logout(){
      return $this->redirect($this->Auth->logout());
    }
  }
?>
```

src/Template/Authexs/login.ctp

```
<?php
  echo $this->Form->create();
  echo $this->Form->input('username');
  echo $this->Form->input('password');
  echo $this->Form->button('Submit');
  echo $this->Form->end();
?>
```

src/Template/Authexs/logout.ctp

```
You are successfully loggedout.
```

src/Template/Authexs/index.ctp

```
You are successfully logged in.
<?php echo
  $this->Html->link('logout',["controller" => "Authexs","action" => "logout"]);
?>
```

Visite
http://localhost/exemplos/auth

Exemplo 14 - Errors and Exception Handling

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/exception/:arg1/:arg2',[
      'controller'=>'Exps','action'=>'index'],['pass' => ['arg1', 'arg2']]);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/ExpsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\Core\Exception\Exception;

  class ExpsController extends AppController{
    public function index($arg1,$arg2){
      try{
        $this->set('argument1',$arg1);
        $this->set('argument2',$arg2);

        if(($arg1 < 1 || $arg1 > 10) || ($arg2 < 1 || $arg2 > 10))
        throw new Exception("One of the number is out of range[1-10].");
      }catch(\Exception $ex){
        echo $ex->getMessage();
      }
    }
  }
?>
```

src/Template/Exps/index.ctp

This is CakePHP tutorial and this is an example of Passed arguments.
Argument-1: <?=$argument1?>
Argument-2: <?=$argument2?>

Visite
http://localhost/exemplos/exception/5/0

Exemplo 15 - Logging

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('logex',['controller'=>'Logexs','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/LogexController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\Log\Log;

  class LogexsController extends AppController{
    public function index(){
      /*The first way to write to log file.*/
      Log::write('debug',"Something didn't work.");

      /*The second way to write to log file.*/
      $this->log("Something didn't work.",'debug');
    }
  }
?>
```

src/Template/Logexs/index.ctp

Something is written in log file. Check log file logs\debug.log

Visite
http://localhost/exemplos/logex

Exemplo 16 - Manupulação de Forms

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('register',['controller'=>'Registrations','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/RegistrationsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class RegistrationsController extends AppController{
    public function index(){
      $country = array('India','United State of America','United Kingdom');
      $this->set('country',$country);
      $gender = array('Male','Female');
      $this->set('gender',$gender);
    }
  }
?>
```

src/Template/Registrations/index.ctp

```
<?php
  echo $this->Form->create("Registrations",array('url'=>'/register'));
  echo $this->Form->input('username');
  echo $this->Form->input('password');
  echo $this->Form->input('password');
  echo '<label for="country">Country</label>';
  echo $this->Form->select('country',$country);
  echo '<label for="gender">Gender</label>';
  echo $this->Form->radio('gender',$gender);
  echo '<label for="address">Address</label>';
  echo $this->Form->textarea('address');
  echo $this->Form->file('profilepic');
  echo '<div>'.$this->Form->checkbox('terms').
    '<label for="country">Terms &Conditions</label></div>';
  echo $this->Form->button('Submit');
  echo $this->Form->end();
```

```
?>
```

Visite
http://localhost/exemplos/register

Exemplo 17 - E-mail

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/email',['controller'=>'Emails','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/EmailsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\Mailer\Email;

  class EmailsController extends AppController{
    public function index(){
      $email = new Email('default');
      $email->to('abc@gmail.com')->subject('About')->send('My message');
    }
  }
?>
```

config/app.php

```
'Emailtransport' => [
    'default' => [
        'className' => 'Mail',
        // as seguintes chaves são usadas no SMTP
        'host' => 'localhost',
        'port' => 25,
        'timeout' => 30,
        'username' => 'users',
        'password' => 'senha',
        'client' => null,
        'tls' => null,
```

```
            'url' => env('EMAIL_TRANSPORT_DEFAULT_URL', null),
        ],
        'gmail' => [
            'host' => 'ssl://smtp.gmail.com',
            'port' => 465,
            'username' => 'GMAIL USERNAME',
            'password' => 'APP PASSWORD',
            'className' => 'Smtp'
        ],
],
```

Visite
http://localhost/exemplos/email

Exemplo 18 - Session

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('/sessionobject',
      ['controller'=>'Sessions','action'=>'index']);
    $routes->connect('/sessionread',
      ['controller'=>'Sessions','action'=>'retrieve_session_data']);
    $routes->connect('/sessionwrite',
      ['controller'=>'Sessions','action'=>'write_session_data']);
    $routes->connect('/sessioncheck',
      ['controller'=>'Sessions','action'=>'check_session_data']);
    $routes->connect('/sessiondelete',
      ['controller'=>'Sessions','action'=>'delete_session_data']);
    $routes->connect('/sessiondestroy',
      ['controller'=>'Sessions','action'=>'destroy_session_data']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/SessionsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class SessionsController extends AppController{
    public function retrieveSessionData(){
      //create session object
```

```
        $session = $this->request->session();

        //read data from session
        $name = $session->read('name');
        $this->set('name',$name);
    }
    public function writeSessionData(){
        //create session object
        $session = $this->request->session();

        //write data in session
        $session->write('name','Virat Gandhi');
    }
    public function checkSessionData(){
        //create session object
        $session = $this->request->session();

        //check session data
        $name = $session->check('name');
        $address = $session->check('address');
        $this->set('name',$name);
        $this->set('address',$address);
    }
    public function deleteSessionData(){
        //create session object
        $session = $this->request->session();

        //delete session data
        $session->delete('name');
    }
    public function destroySessionData(){
        //create session object
        $session = $this->request->session();

        //destroy session
        $session->destroy();
    }
  }
?>
```

src/Template/Sessions/write_session_data.ctp

The data has been written in session.

src/Template/Sessions/retrieve_session_data.ctp

Here is the data from session.
Name: <?=$name;?>

src/Template/Sessions/check_session_data.ctp

```
<?php if($name): ?>
name exists in the session.

<?php else: ?>
name doesn't exist in the database

<?php endif;?>
<?php if($address): ?>
address exists in the session.

<?php else: ?>
address doesn't exist in the database

<?php endif;?>
```

src/Template/Sessions/delete_session_data.ctp

```
Data deleted from session.
```

src/Template/Sessions/destroy_session_data.ctp

```
Session Destroyed.
```

Visite
http://localhost/exemplos/session-write

http://localhost/exemplos/session-read

http://localhost/exemplos/sessioncheck

http://localhost/exemplos/sessiondelete

http://localhost/exemplos/sessiondestroy

Exemplo 19 - Cookies

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('cookie/write',['controller'=>'Cookies','action'=>'write_cookie']);
    $routes->connect('cookie/read',['controller'=>'Cookies','action'=>'read_cookie']);
    $routes->connect('cookie/check',['controller'=>'Cookies','action'=>'check_cookie']);
```

```
    $routes->connect('cookie/delete',['controller'=>'Cookies','action'=>'delete_cookie']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/Cookies/CookiesController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;
  use Cake\Controller\Component\CookieComponent;

  class CookiesController extends AppController{
    public $components = array('Cookie');

    public function writeCookie(){
      $this->Cookie->write('name', 'Virat');
    }
    public function readCookie(){
      $cookie_val = $this->Cookie->read('name');
      $this->set('cookie_val',$cookie_val);
    }
    public function checkCookie(){
      $isPresent = $this->Cookie->check('name');
      $this->set('isPresent',$isPresent);
    }
    public function deleteCookie(){
      $this->Cookie->delete('name');
    }
  }
?>
```

src/Template/Cookie/write_cookie.ctp

```
The cookie has been written.
```

src/Template/Cookie/read_cookie.ctp

```
The value of the cookie is: <?php echo $cookie_val; ?>
```

src/Template/Cookie/check_cookie.ctp

```
<?php
  if($isPresent):
?>
The cookie is present.

<?php
  else:
?>
```

The cookie isn't present.

```
<?php
  endif;
?>
```

src/Template/Cookie/delete_cookie.ctp

The cookie has been deleted.

Visite
http://localhost/exemplos/cookie/write
http://localhost/exemplos/cookie/read
http://localhost/exemplos/cookie/check
http://localhost/exemplos/cookie/delete

Exemplo 20 - Segurança
O CakePHP tem alguns recursos que melhoram a segurança do aplicativo.

config/routes.php

```
<?php
  use Cake\Core\Plugin;
  use Cake\Routing\RouteBuilder;
  use Cake\Routing\Router;

  Router::defaultRouteClass('DashedRoute');
  Router::scope('/', function (RouteBuilder $routes) {
    $routes->connect('login',['controller'=>'Logins','action'=>'index']);
    $routes->fallbacks('DashedRoute');
  });
  Plugin::routes();
```

src/Controller/LoginsController.php

```
<?php
  namespace App\Controller;
  use App\Controller\AppController;

  class LoginsController extends AppController{
    public function initialize(){
      parent::initialize();
      $this->loadComponent('Security');
    }
    public function index(){
    }
  }
?>
```

src/Template/Logins/index.ctp

```
<?php
   echo $this->Form->create("Logins",array('url'=>'/login'));
   echo $this->Form->input('username');
   echo $this->Form->input('password');
   echo $this->Form->button('Submit');
   echo $this->Form->end();
?>
```

Visite
http://localhost/exemplos/login

Resumo do CakePHP

Geralmente controller são usados para gerenciar a lólica de um único model.

Controllers são classes que estendem a classe AppController. AppController deve conter métodos que são compartilhados entre todos os controllers de sua aplicação.

Os controllers fornecem uma série de métodos que lidam com requisições. Estes são chamados de actions.

Components - são a melhor alternativa sobre código usado por muitos controllers.

Os controllers e componentes contam com um método initilize() que é invocado ao final do cconstrutor do controller.

Qualquer trecho de código ou componente:
- No AppController fica acessível por todos os controllers.
- Num certo controller fica acessível somente para este

Redirecionamento para outras páginas

controller/action
$this->redirect(['controller' => 'Clientes', 'action' => 'index'])

URL
$this->redirect('http://ribafs.org')

Action do controller atual
$this->redirect(['action' => 'edit', $id]);
ou
$this->setAction('index')

Para o próprio link de onde veio
$this->redirect($this->referer());

Carregar model que não é o default

$this->LoadModel('Clientes');

Componente Flash
Uma forma eficiente de enviar mensagens do controller para as views
$this->Flash->msg

msg =
success
greatSuccess
set
error
warning

Layout - contém código de apresentação da view. Tudo que vemos nas views está num layout.

Estão em src/Template/Layout

O cake já vem com um layout padrão, que é o default.ctp.

Como setar outro layout no controller:

```
public function initialize()
{
    $this-viewBuilder()-setLayout('admin');
}
```

Element - são pedaços de código de apresentação reutilizáveis para as views

Como chamar um element (declarar onde deseja que apareça):
echo $this->element('nome_element');

Acesso a Banco de Dados

O trabalho com bancos de dados no Cake é feito com dois objetos Tables (lida com coleções da dados, tabela, por exemplo) e Entities (lida com apenas um registro).

Para trabalhar com Tabelas num controller

Carregar o objeto Table
use Cake\ORM\TableRegistry;

Carregar o respectivo objeto
$clientes = TableRegistry::get('Clientes');

Agora pode trabalhar com seu conteúdo.

```
use Cake\Datasource\ConnectionManager;

$dsn = 'mysql://root:password@localhost/my_database';
ConnectionManager::config('default', ['url' => $dsn]);
$conn = ConnectionManager::get('default');

$results = $conn->execute('SELECT * FROM articles')->fetchall('assoc');
```

Também podemos usar query builder.

Usando transações

```
$conn->begin();
$conn->execute('UPDATE articles SET published = ? WHERE id = ?', [true, 2]);
$conn->execute('UPDATE articles SET published = ? WHERE id = ?', [false, 2]);
$conn->commit();
```

Cake na Prática

Controller e Views

No controller src/Controller/ClientesController.php

```
   public function index()
   {
      $this->paginate = [
         'contain' => ['Users']
      ];
      $this->set('clientes', $this->paginate($this->Clientes));
      $this->set('_serialize', ['clientes']);

     // Testes
     $this->set('cor', 'Verde');
          $data = [
               'color' => 'pink',
               'type' => 'sugar',
               'base_price' => 23.95
          ];

          // Make $color, $type, and $base_price available to the view:
          $this->set($data);

          $this->set($data);
          $cor1='verde';
          $cor2='azul';
          $this->set(compact('cor1','cor2'));
   }
```

Na view src/Template/Clientes/index.ctp

```
<?php
echo 'Cor: '.$cor.'<br>';
echo $color.'<br>';
echo $type.'<br>';
echo $base_price.'<br><br>';

echo $cor1.'<br>';
echo $cor2.'<br><br>';

?>
```

Redirect

Em um action:

```
    return $this->redirect(['conttroller'=>'Clientes', 'action'=>'add']);
```

Na versão 3 do Cake, os Table Objects contém:
queries
callbacks
validations
events

Carregamento de

Componentes
Helpers
Plugins
Behaviors

Plugins

Depois de instalar um plugin e configurar o autoloader, você deve carregar
O plugin. Você pode carregar plugins um a um, ou todos eles com um único
método::

```
// In config/bootstrap.php
// Or in Application::bootstrap()

// Carrega um único plugin
Plugin::load('ContactManager');

// Carrega um plugin com um namespace no nível superior.
Plugin::load('AcmeCorp/ContactManager');

// Carrega todos os plugins de uma só vez
Plugin::loadAll();
```

``loadAll()`` carrega todos os plugins disponíveis, permitindo que você especifique determinadas
configurações para plugins. ``load()`` funciona de forma semelhante, mas apenas carrega o
Plugins que você especifica explicitamente.

.. note::

    ``Plugin::loadAll()`` não irá carregar os plugins vendor namespaced que não são
    Definido em **vendor/cakephp-plugins.php**.

Há também um comando de shell acessível para habilitar o plugin. Execute a seguinte linha:

.. code-block:: bash

    bin/cake plugin load ContactManager

Isso colocará o plugin ``Plugin::load('ContactManager');`` no bootstrap para você.

.. _autoloading-plugin-classes:

Deployment/Implantação

https://book.cakephp.org/3.0/pt/deployment.html

Antes de implantar um aplicativo em produção faça as seguintes verificações/alterações:

- Atualizar o arquivo config/core.php
    debug = false;

Ao desabilitar o debug altera o seguinte:

    Mensagens de depuração criadas com pr() e debug() serão desabilitadas.
    O cache interno do CakePHP será descartado após 999 dias ao invés de ser a cada 10 segundos como em desenvolvimento.

    Views de erros serão menos informativas, retornando mensagens de erros genéricas.
    Erros do PHP não serão mostrados.

    O rastreamento de stack traces (conjunto de exceções) será desabilitado.

- Checar a segurança:

    Certifique-se de utilizar o Cross Site Request Forgery.
    Você pode querer habilitar o Security. Isso pode prevenir diversos tipos de adulteração de formulários e reduzir a possibilidade de overdose de requisições.

Certifique-se que seus models possuem as regras Validação de validação habilitadas.

Verifique se apenas o seu diretório webroot é visível publicamente, e que seus segredos (como seu app salt, e qualquer chave de segurança) são privados e únicos também.

- Aprimorar a eprformance do aplicativo
Execute
php composer.phar dumpautoload -o

Ler e gravar em variáveis de configuração

```
Configure::load('bootstrap','intl.default_locale');//, 'intl.default_locale');
//Configure::write('intl.default_locale','en_US');

$configValue = Configure::read('intl.default_locale');
echo $configValue;
exit;
```

Dicas sobre o uso de bancos de dados no Cake

Conexão com o PostgreSQL quando usar esquema

```
    public $default = array(
        'datasource' => 'Database/Postgres',
        'persistent' => false,
        'host' => 'localhost',
        'login' => 'postgres',
        'password' => 'postgres',
        'database' => 'banco',
        'schema' => 'esquema',
        'prefix' => '',
        //'encoding' => 'utf8',
    );
```

Query no Model
```
$this->query("SELECT * FROM clientes;");
```

Query no Controller
```
$this->Cliente->query("SELECT * FROM clientes;");
```

Salvando Dados
```
if ($this->ModelName->save($this->request->data)) {
   $this->Session->setFlash('Data Saved!');
}
```

Salvar Muitos

```
$data = array(
   array('title' => 'title 1'),
   array('title' => 'title 2'),
);
$this->ModelName->saveAll($data);
```

Excluir

```
$this->Model->delete($this->request->data('Model.id'));
```

Delete all

```
$this->Model->deleteAll(array('Model.spam' => true), false);
```

Uso do LIKE

```
$keyword =$this->params['named']['keyword'];
$this->paginate['conditions'] = array("Cliente.nome LIKE" => "%$keyword%");

$this->Post->find('first', array (
   "Author.name" => "Bob",
   "OR" => array (
      "Post.title LIKE" => "%magic%",
      "Post.created >" => date('Y-m-d', strtotime("-2 weeks"))
   )
));

$this->Post->find("all",array('condition'=>array('Author LIKE'=>"ad%")));

$this->Post->find("all",array('condition'=>array("OR"=>array('Author LIKE'=>"ad%",'Author
LIKE'=>"bo%"))));
```

Complexas conditions

```
// Diferente
array("Post.title !=" => "This is a post")

array (
   "Post.title" => array("First post", "Second post", "Third post"),
   "Post.created >" => date('Y-m-d', strtotime("-2 weeks"))
)

// Ou
array("OR" => array(
   "Post.title" => array("First post", "Second post", "Third post"),
   "Post.created >" => date('Y-m-d', strtotime("-2 weeks"))
))
```

```
array(
    "Author.name" => "Bob",
    "OR" => array(
        "Post.title LIKE" => "%magic%",
        "Post.created >" => date('Y-m-d', strtotime("-2 weeks"))
    )
)

array('OR' => array(
    array('Post.title LIKE' => '%one%'),
    array('Post.title LIKE' => '%two%')
))

// BETWEEN
array('Post.read_count BETWEEN ? AND ?' => array(1,10))

// Group By
array(
    'fields' => array(
        'Product.type',
        'MIN(Product.price) as price'
    ),
    'group' => 'Product.type'
)
```

// Datasource
Should you need even more control over your queries, you can make use of prepared statements. This allows you to talk directly to the database driver and send any custom query you like:

Se você precisar de mais controle sobre suas consultas, você pode fazer uso de instruções preparadas. Isso permite que você se comunique diretamente com o driver de banco de dados e enviar qualquer consulta personalizada que você queira:

```
$db = $this->getDataSource();
$db->fetchAll(
    'SELECT * from users where username = ? AND password = ?',
    array('jhon', '12345')
);
$db->fetchAll(
    'SELECT * from users where username = :username AND password = :password',
    array('username' => 'jhon','password' => '12345')
);
```

Outro exemplo:
```
$query = "SELECT * FROM user WHERE id=:user_id"
$data = $this->getDataSource()->fetchAll($query, array("usery_id" => $user_id),
array("cache" => false));
```

Funções e Rotinas úteis

```
// Devolve nomes de todos os controllers
pr(App::objects('Controller'));

Retornar todos os controllers para uma view
public function controllers(){
      $controllers = App::objects('Controller');

      // Evitar AppController e PagesController
      $value = 'AppController';
      $key = array_search($value, $controllers);
      unset($controllers[$key]);
      $value = 'PagesController';
      $key = array_search($value, $controllers);
      unset($controllers[$key]);

      $this->set('controllers',$controllers);

}

// Devolve os nomes de todos os models
pr(App::objects('Model'));

Para receber os actions de um controller:
      public function actions($controller){
       $parentClassMethods =
get_class_methods(get_parent_class(Inflector::camelize($controller).'Controller'));
       $subClassMethods    =
get_class_methods(Inflector::camelize($controller).'Controller');
       $classMethods       = array_diff($subClassMethods,$parentClassMethods);

       foreach($classMethods as $method) {
          if($method{0} <> "_") {
             $classMethodsCleaned[] = $method;
          }
       }
           return $classMethodsCleaned;
      }

      public function controles(){
           $this->Privilege = ClassRegistry::init('Privilege');
           $controllers = App::objects('Controller');

           $act = $this->ControllerList->getList();

           foreach($controllers as $controller){
                if($controller != 'AppController' && $controller !='PagesController'){
                     $actions = $act[$controller]['actions'];
                     foreach($actions as $action){
```

```
                        $controller = strtolower(substr($controller,0,-10));
                        $act2 = $this->Privilege->find('first', array(
                            'conditions'=>array('Privilege.controller'=>$controller,'Privilege.acti
on'=>$action),
                            'fields'=>array('Privilege.action','Privilege.controller')
                        ));

                        if(isset($act2['Privilege']['action'])){
                            $act2 = $act2['Privilege']['controller'].'-'.$act2['Privilege']['action'];
                            print $act2;
                        }else{
                            $act3 = $this->Privilege->query("select action,controller from
privileges where action='$action'");
                        }
                        if(isset($act3['Privilege']['action'])){
                            $act3 = $act3['Privilege']['action'].'-'.$act3['Privilege']['action'];
                            print $act3;
                        }else{
                            return false;
                        }
                    }
                }
            }

    }


No appcontroller
$this->AccessControl->controles();


// Devolve nomes de todas as tabelas
// Adicionar ao AppController
public function tabelas($banco, $model){
    $tabelas = $this->{$model}->query('SHOW TABLES');// Cliente pode ser qq Model do
aplicativo
    $tab = array();
    $bd="Tables_in_{$banco}";
    for($x=0;$x<count($tabelas);$x++){
        $tab[] = $tabelas[$x]['TABLE_NAMES'][$bd];
    }
    pr($tab);
}
// Chamar no beforeFilter em um Controller:
    public function beforeFilter() {
        parent::beforeFilter();
        $this->tabelas('cake_admin', 'Cliente');
        // Outros códigos que precisar ficam aqui
    }
```

```
// Controlando a action inicial de cada grupo
if($this->Auth->user('group_id') == 1){
	$this->Auth->loginRedirect = array('controller' => 'users', 'action' => 'index');
}else{
	$this->Auth->loginRedirect = array('controller' => 'posts', 'action' => 'index');
}
```

Dicas de PHP

Dicas que são úteis em qualquer aplicativo.

Operador Ternário/Expressão Condicional

```
$retorno = isset ($x) ? $x : 0;
```

Silimar a:

```
if(isset ($x)) {
	$retorno = $x;
}else{
	$retorno = 0;
}
```

Identificador único

```
$idunico = md5(uniqid(microtime(), 1)).getmypid();
print $idunico;
```

Retorna: e24077859a91baddc537cd9d69449c941532

Criar número realmente aleatório

```
mt_srand((double)microtime()*1000000);
print mt_rand();
print mt_rand();
// 1369582106372919427
```

Filtro de Caracteres Especiais

Importante antes de armazenar no banco e antes de exibir, vindo do banco.

addslashes($str) - Adiciona barras invertidas às aspas de a uma string

```
$str = "Is your name O'reilly?";
echo addslashes($str);
// Saída: Is your name O\'reilly?
```

quotemeta — Adiciona uma barra invertida antes dos meta caracteres:
. \ + * ? [^] ($)

Exemplo:
$str = "Hello world. (can you hear me?)";
echo quotemeta($str);
// Saída: Hello world\. \(can you hear me\?\)

nl2br — Insere quebras de linha HTML antes de todas newlines em uma string
echo nl2br("foo isn't\n bar");

str_replace — Substitui todas as ocorrências da string de procura com a string de substituição
Exemplo:
$string = "Apenas uma vez será suficiente";
$trocar = "será";
$por = "não será";

print str_replace($trocar, $por, $string);

trim — Retira espaço no ínicio e final de uma string
$semespacos = trim($texto);

is_int — Informa se a variável é do tipo inteiro

if (is_int(23))

is_ array
is_ bool
is_ callable
is_ double
is_ float
is_ int
is_ integer
is_ long
is_ null
is_ numeric
is_ object
is_ real
is_ resource
is_ scalar
is_ string

substr — Retorna uma parte de uma string
string substr (string $string , int $start [, int $length])

start
- Pode iniciar do começo para o final da string (valor positivo)

- Do final para o começo (valor negativo)

length - quando negativo, retiramos a parte negativa do todo da string

Exemplos:
print substr("abcdef", 0, 3); // retorna "abc", comece em 0 e pegue 3
print substr("abcdef", -2, 2); // retorna "ef", comece do segundo de traz para a frente e traga 2
print substr("abcdef", 0, -2); // retorna "abcd", ou seja, começa em 0 e não traz os 2 últimos

strlen — Retorna o tamanho de uma string
$str = 'abcdef';
echo strlen($str); // 6

number_format — Formata um número com os milhares agrupados
string number_format (float $number , int $decimals , string $dec_point , string $thousands_sep)
// Notação Brasileira
$valor = '1235.99';
$nombre_format_br = number_format($valor, 2, ',', '.');
print $nombre_format_br;
// 1.234,56

PHP para Matemática

log — Logaritmo natural
print log(10);

rad2deg — Converte o número em radianos para o equivalente em graus
$pi = 3.14;
echo rad2deg($pi);

deg2rad — Converte o número em graus ao equivalente em radianos
echo deg2rad(45); // 0.785398163397

checkdate — Valida uma data Gregoriana
bool checkdate (int $month , int $day , int $year)
Exemplo:
$data = '2013-08-21';
$d = explode('-',$data);

$ano = $d[0];
$mes = $d[1];
$dia = $d[2];

```
$vdata = checkdate ( $mes , $dia , $ano );

if($vdata){
    print 'Data válida';
}else{
    print 'Data inválida';
}
```

explode — Divide uma string em strings
array explode (string $delimiter , string $string [, int $limit])

```
$data = '2013-08-21';
$d = explode('-',$data);

$ano = $d[0];
$mes = $d[1];
$dia = $d[2];
```

== Adicionar um array a outro

array_merge — Funde dois ou mais arrays

```
$array1 = array("cor" => "vermelho", 2, 4);
$array2 = array("a", "b", "cor" => "verde", "forma" => "trapezoide", 4);
$result = array_merge($array1, $array2);
print '<pre>';
print_r($result);
print '</pre>';
```

Observe que cor ficou apenas o último (vermelho), o primeiro foi sobrescrito.

array_diff — Analisa as diferenças entre arrays

```
$array1 = array("a" => "verde", "vermelho", "azul", "vermelho");
$array2 = array("b" => "verde", "amarelo", "vermelho");
$result = array_diff($array1, $array2);
print_r($result);// Mostrará: Array ( [1] => azul ) // Somente o que tem de diferente entre ambos
```

array_diff_assoc — Computa a diferença entre arrays com checagem adicional de índice

```
$array1 = array("a" => "verde", "b" => "marrom", "c" => "azul", "vermelho");
$array2 = array("a" => "verde", "amarelo", "vermelho");
$result = array_diff_assoc($array1, $array2);
print_r($result);
```

Mostrará:

```
Array
(
```

```
    [b] => marrom
    [c] => azul
    [0] => vermelho
)
```

array_intersect — Calcula a interseção entre arrays

```
$array1 = array("a" => "verde", "vermelho", "azul");
$array2 = array("b" => "verde", "amarelo", "vermelho");
$result = array_intersect($array1, $array2);
print_r($result);
```

Mostrará:

```
Array
(
    [a] => verde
    [0] => vermelho
)
```

array_fill — Preenche um array com valores

```
array array_fill ( int $start_index , int $num , mixed $value )
```

```
$a = array_fill(5, 6, 'banana');
$b = array_fill(-2, 2, 'pear');
print_r($a);
print_r($b);
?>
```

O exemplo acima irá imprimir:

```
Array
(
    [5]  => banana
    [6]  => banana
    [7]  => banana
    [8]  => banana
    [9]  => banana
    [10] => banana
)
Array
(
    [-2] => pear
    [0] => pear
)
```

array_key_exists — Checa se uma chave ou índice existe em um array

```
bool array_key_exists ( mixed $key , array $search )
```

```
$busca_array = array("primeiro" => 1, "segundo" => 4);
if (array_key_exists("primeiro", $busca_array)) {
    echo "O elemento 'primeiro' está no array!";
```

```
}
```

array_keys — Retorna todas as chaves de um array
array array_keys (array $input [, mixed $search_value [, bool $strict]])

array_map — Aplica uma função em todos os elementos dos arrays dados

```
function cubo($n)
{
    return $n*$n*$n;
}

$a = array(1, 2, 3, 4, 5);
$b = array_map("cubo", $a);
print_r($b);
?>
```

E programa acima faz com que $b tenha:

```
Array
(
    [0] => 1
    [1] => 8
    [2] => 27
    [3] => 64
    [4] => 125
)
```

array_pad — Expande um array para um certo comprimento utilizando um determinado valor
array array_pad (array $input , int $pad_size , mixed $pad_value)

```
$input = array(12, 10, 9);

$result = array_pad($input, 5, 0); // 5 é o novo tamanho
// $result é array(12, 10, 9, 0, 0)

$result = array_pad($input, -7, -1);
// $result é array(-1, -1, -1, -1, 12, 10, 9)

$result = array_pad($input, 2, "noop");
// Não será expandido
```

array_pop — Retira um elemento do final do array
mixed array_pop (array &$array)

```
$cesta = array("laranja", "banana", "melancia", "morango");
$fruta = array_pop($cesta);
print_r($cesta);
?>
```

Depois disso, $cesta terá 3 elementos:

```
Array
(
    [0] => laranja
    [1] => banana
    [2] => melancia
)
```

array_product — Calcula o produto dos valores de um array
number array_product (array $array)

```
$a = array(2, 4, 6, 8);
echo "product(a) = " . array_product($a) . "\n";
```

array_push — Adiciona um ou mais elementos no final de um array
int array_push (array &$array , mixed $var [, mixed $...])

```
$cesta = array("laranja", "morango");
array_push($cesta, "melancia", "batata");
print_r($cesta);
```

array_rand — Retorna um ou mais elementos aleatórios de um array
mixed array_rand (array $input [, int $num_req])

array_reduce — Reduz um array para um único valor através de um processo iterativo utilizando uma função

array_search — Procura por um valor em um array e retorna sua chave correspondente caso seja encontrado
mixed array_search (mixed $needle , array $haystack [, bool $strict])

```
$array = array(0 => 'blue', 1 => 'red', 2 => 'green', 3 => 'red');

$key = array_search('green', $array); // $key = 2;
$key = array_search('red', $array);   // $key = 1;
```

array_shift — Retira o primeiro elemento de um array
mixed array_shift (array &$array)

array_slice — Extrai uma parcela de um array

array_splice — Remove uma parcela do array e substitui com outros elementos

array_sum — Calcula a soma dos elementos de um array

```
$a = array(2, 4, 6, 8);
echo "soma(a) = ".array_sum($a)."\n";

$b = array("a" => 1.2, "b" => 2.3, "c" => 3.4);
echo "soma(b) = ".array_sum($b)."\n";
```

array_unique — Remove os valores duplicados de um array

```
$input = array("a" => "verde", "vermelho", "b" => "verde", "azul", "vermelho");
$result = array_unique($input);
print_r($result);
```

array_values — Retorna todos os valores de um array
array array_values (array $input)

```
$array = array("tamanho" => "G", "cor" => "dourado");
print_r(array_values ($array));
```

asort — Ordena um array mantendo a associação entre índices e valores
bool asort (array &$array [, int $sort_flags])

compact — Cria um array contendo variáveis e seus valores
array compact (mixed $varname [, mixed $...])

```
$cidade = "Sao Paulo";
$estado = "SP";
$evento = "SIGGRAPH";

$localidade = array("cidade", "estado");

$result = compact("evento", "nada_aqui", $localidade);
print_r($result);
?>
```

O exemplo acima irá imprimir:

```
Array
(
    [evento] => SIGGRAPH
    [cidade] => Sao Paulo
    [estado] => SP
)
```

count — Conta o número de elementos de uma variável, ou propriedades de um objeto
int count (mixed $var [, int $mode])

sizeof — Sinônimo de count()

current — Retorna o elemento corrente em um array
mixed current (array &$array)
Todo array tem um ponteiro interno para o elemento "atual", o qual é inicializado para apontar para o primeiro elemento inserido em um array.

list — Cria variáveis como se fossem arrays

Variáveis Variáveis

```
<?php
```

```
$reciboextra2=array(0=>'2a', 1=>'2b', 2=>'2c', 3=>'2d');
$reciboextra3=array(0=>'3a', 1=>'3b', 2=>'3c', 3=>'3d');
$reciboextra4=array(0=>'4a', 1=>'4b', 2=>'4c', 3=>'4d');

$numero=0;
echo "<pre>";
for($x=2;$x<5;$x++){

//Aqui você transforma a string em nome de variável
 $var = ${"reciboextra$x"};

echo "Var:reciboextra$x\tNumero:$x\tValor:" . $var[$numero] . '<br>';
echo "--------------------------------\n";

$numero++;
}

echo "</pre>";
?>
```

Testar online http://www.compileonline.com/execute_php_online.php

Um Resumo da codificicação do CakePHP

```
class ArticlesController extends AppController
{

    public function index()
    {
        $this->set('articles', $this->Articles->find('all'));
    }

    public function view($id = null)
    {
        $article = $this->Articles->get($id);
        $this->set(compact('article'));
    }
}
```

Repare que você está usando get() ao invés de find('all') porquê nós queremos a informação de apenas um artigo.

Repare que nossa action recebe um parâmetro: o ID do artigo que gostariamos de visualizar.
Se o usuário requisitar /articles/view/3, então o valor '3' é passado como $id para a action.

Para receber segurança com strings nos forms, usar a função h():

```
<!-- File: src/Template/Articles/view.ctp -->

<h1><?= h($article->title) ?></h1>
<p><?= h($article->body) ?></p>
```

Testar se uma requisição é realmente post:

```
if ($this->request->is('post')) {
```

Debugar

```
degub($variavel);exit;
```

Métodos success e error do FlashComponent para definir mensagens

```
if ($this->Articles->save($article)) {
            $this->Flash->success(__('Seu artigo foi salvo.'));
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Não é possível adicionar o seu artigo.'));
```

Validação dos Forms

Para tirar proveito dos recursos de validação do Cake, você vai precisar usar o Form helper em suas views.
Basta usar $this->Form->.

Exemplo de view add:

```
<!-- File: src/Template/Articles/add.ctp -->
<h1>Add Article</h1>
<?php
   echo $this->Form->create($article);
   echo $this->Form->input('title');
   echo $this->Form->input('body', ['rows' => '3']);
   echo $this->Form->button(__('Salvar artigo'));
   echo $this->Form->end();
?>
```

Regras de validação são definidas no Model, exemplo

```
   public function validationDefault(Validator $validator)
   {
```

```
        $validator
            ->notEmpty('title')
            ->notEmpty('body');

        return $validator;
    }

    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->getData());
```

Em seguida, a action verifica se a requisição é POST ou PUT e caso seja, os dados são usados para atualizar a entidade de artigo em questão ao usar o método patchEntity().

Recebendo o nome de um action

```
$this->request->getParam('action')
```

Recebendo nome de Controller

```
$controllerName = $this->request->getParam('controller');
```

Camada Model

```
use Cake\ORM\TableRegistry;

$users = TableRegistry::get('Users');
$query = $users->find();
foreach ($query as $row) {
    echo $row->username;
}
```

Se nós quiséssemos criar um usuário e salvá-lo (com validação) fariamos algo assim:

```
use Cake\ORM\TableRegistry;

$users = TableRegistry::get('Users');
$user = $users->newEntity(['email' => 'mark@example.com']);
$users->save($user);
```

A camada view

```
// No arquivo view, nós renderizaremos um 'elemento' para cada usuário.
<?php foreach ($users as $user): ?>
    <div class="user">
        <?= $this->element('user', ['user' => $user]) ?>
```

```
    </div>
<?php endforeach; ?>
```

Controller
Um controller pode ser visto como um gerente que certifica-se que todos os recursos necessários para completar uma tarefa sejam delegados aos trabalhadores corretos.

Ele aguarda por petições dos clientes, checa suas validades de acordo com autenticação ou regras de autorização, delega requisições ou processamento de dados da camada Model, selecciona o tipo de dados de apresentação que os clientes estão aceitando, e finalmente delega o processo de renderização para a camada View.

Enviar informações do controller para uma view:

```
$this->set('valores', $valores);
```

Somente acaitar DELETE ou POST

```
    $this->request->allowMethod(['post', 'delete']);
```

Enviar informações da view para o controller

in your view you can use get method to send the form data. and in your controller's action you can access the passed arguments with $this->params['url'];

in your view

```
  $this->Form->create('Model', array('type' => 'get', 'action' => 'search'));
  $this->Form->input('select_tfield_id', array('type' => 'select'));
  $this->Form->input('value'));
  $this->Form->end('submit');
```

In your controller

```
  function search() {
     $url = $this->params['url'];
     $id = $url['select_tfield_id'];
     $value = $url['value'];
  }
```

Pelo form
<?= $this->Form->hidden('name', ['value' => $this->Form->select('name')]) ?>
Não passa nada, apenas null

Carregar componentes

```
$this->loadComponent('Flash');
$this->loadComponent('Auth');
```

Alguns actions típicos:

```
  public function login()
  {
       if ($this->request->is('post')) {
          $user = $this->Auth->identify();
          if ($user) {
             $this->Auth->setUser($user);
             return $this->redirect($this->Auth->redirectUrl());
          }
          $this->Flash->error('Your username or password is incorrect.');
       }
  }

  public function logout()
  {
       $this->Flash->success('You are now logged out.');
       return $this->redirect($this->Auth->logout());
  }

  public function index()
{
   $this->set('articles', $this->Articles->find('all'));
}

public function view($id)
{
   $article = $this->Articles->get($id);
   $this->set(compact('article'));
}

public function add()
{
   $article = $this->Articles->newEntity();
   if ($this->request->is('post')) {
      $article = $this->Articles->patchEntity($article, $this->request->getData());
      if ($this->Articles->save($article)) {
         $this->Flash->success(__('Your article has been saved.'));
         return $this->redirect(['action' => 'index']);
      }
      $this->Flash->error(__('Unable to add your article.'));
   }
   $this->set('article', $article);
}
```

```
    // src/Controller/ArticlesController.php

    public function edit($id = null)
    {
        $article = $this->Articles->get($id);
        if ($this->request->is(['post', 'put'])) {
            $this->Articles->patchEntity($article, $this->request->getData());
            if ($this->Articles->save($article)) {
                $this->Flash->success(__('Your article has been updated.'));
                return $this->redirect(['action' => 'index']);
            }
            $this->Flash->error(__('Unable to update your article.'));
        }

        $this->set('article', $article);
    }

    // src/Controller/ArticlesController.php

    public function delete($id)
    {
        $this->request->allowMethod(['post', 'delete']);

        $article = $this->Articles->get($id);
        if ($this->Articles->delete($article)) {
            $this->Flash->success(__('The article with id: {0} has been deleted.', h($id)));
            return $this->redirect(['action' => 'index']);
        }
    }
```

Algumas views típicas:

```
<!-- File: src/Template/Articles/add.ctp -->

<h1>Add Article</h1>
<?php
    echo $this->Form->create($article);
    echo $this->Form->control('title');
    echo $this->Form->control('body', ['rows' => '3']);
    echo $this->Form->button(__('Save Article'));
    echo $this->Form->end();
?>

<!-- File: src/Template/Articles/edit.ctp -->

<h1>Edit Article</h1>
<?php
    echo $this->Form->create($article);
    echo $this->Form->control('title');
```

```
    echo $this->Form->control('body', ['rows' => '3']);
    echo $this->Form->button(__('Save Article'));
    echo $this->Form->end();
?>

<!-- File: src/Template/Articles/view.ctp -->

<h1><?= h($article->title) ?></h1>
<p><?= h($article->body) ?></p>
<p><small>Created: <?= $article->created->format(DATE_RFC850) ?></small></p>


<!-- File: src/Template/Articles/index.ctp  (edit links added) -->

<!-- File: src/Template/Articles/index.ctp (delete links added) -->

<h1>Blog articles</h1>
<p><?= $this->Html->link('Add Article', ['action' => 'add']) ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
        <th>Actions</th>
    </tr>

<!-- Here's where we loop through our $articles query object, printing out article info -->

    <?php foreach ($articles as $article): ?>
    <tr>
        <td><?= $article->id ?></td>
        <td>
            <?= $this->Html->link($article->title, ['action' => 'view', $article->id]) ?>
        </td>
        <td>
            <?= $article->created->format(DATE_RFC850) ?>
        </td>
        <td>
            <?= $this->Form->postLink(
                'Delete',
                ['action' => 'delete', $article->id],
                ['confirm' => 'Are you sure?'])
            ?>
            <?= $this->Html->link('Edit', ['action' => 'edit', $article->id]) ?>
        </td>
    </tr>
    <?php endforeach; ?>

</table>
```

```
<!-- File: src/Template/Users/login.ctp -->

<div class="users form">
<?= $this->Flash->render() ?>
<?= $this->Form->create() ?>
    <fieldset>
        <legend><?= __('Please enter your username and password') ?></legend>
        <?= $this->Form->control('username') ?>
        <?= $this->Form->control('password') ?>
    </fieldset>
<?= $this->Form->button(__('Login')); ?>
<?= $this->Form->end() ?>
</div>
```

Mudando a route:

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

para

```
$routes->connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Select num Controller

```
use Cake\ORM\TableRegistry;

        $query = TableRegistry::get('Articles')->find();

        foreach ($query as $article) {
            debug($article->title);
        }
```

Por default uma consulta traz todos os campos de uma tabela

```
$query = $articles->find();
$articles->find()->select();
```

Para trazer apenas id e title:

```
$articles->find()->select(['id', 'title']);
```

Retornar todos os title

```
use Cake\ORM\TableRegistry;
        $articles = TableRegistry::get('Articles');
        $allTitles = $articles->find()->extract('title');
        foreach ($allTitles as $title) {
            echo $title.'<br>';
        }
```

Par chave-valor:
```
$list = $articles->find('list');

foreach ($list as $id => $title) {
    echo "$id : $title"
}
```

Retornando os title
```
$query = TableRegistry::get('Articles')->find();

foreach ($query as $article) {
    debug($article->title);
}
```

Retornando um único registro de uma tabela

```
$articles = TableRegistry::get('Articles');
$article = $articles
    ->find()
    ->where(['id' => 1])
    ->first();

debug($article->title);
```

Mostrar apenas um campo (title)
```
$articles = TableRegistry::get('Articles');
$query = $articles->find();
$query->select(['id', 'title', 'body']); // ou apenas $query->select();
foreach ($query as $row) {
    debug($row->title);
}
```

Limitar registros e páginas:
```
// Fetch rows 50 to 100
$query = $articles->find()
    ->limit(50)
    ->page(2);
```

WHERE CASE
```
$query = $cities->find()
    ->where(function ($exp, $q) {
        return $exp->addCase(
            [
                $q->newExpr()->lt('population', 100000),
                $q->newExpr()->between('population', 100000, 999000),
```

```
            $q->newExpr()->gte('population', 999001),
        ],
        ['SMALL',  'MEDIUM', 'LARGE'], # values matching conditions
        ['string', 'string', 'string'] # type of each value
      );
  });
# WHERE CASE
#   WHEN population < 100000 THEN 'SMALL'
#   WHEN population BETWEEN 100000 AND 999000 THEN 'MEDIUM'
#   WHEN population >= 999001 THEN 'LARGE'
#   END

$query = $cities->find()
  ->where(function ($exp, $q) {
     return $exp->like('name', '%A%');
  });
# WHERE name LIKE "%A%"
```

Retornar todo o primeiro registro:

```
$articles = TableRegistry::get('Articles');
$query = $articles->find('all', [
  'order' => ['Articles.created' => 'DESC']
]);
$row = $query->first();
debug($row);
```

Dinâmico encontrar

```
$query = $this->Users->findByUsername('usuario');
$query = $this->Users->findAllByUsername('usuario');
```

Encontrar relacionados

```
$query = $articles->find()->contain([
  'Authors' => ['Addresses'], 'Comments' => ['Authors']
]);

$query = $products->find()->contain([
  'Shops.Cities.Countries',
  'Shops.Managers'
]);
```

Inserir um novo registro

```
$articlesTable = TableRegistry::get('Articles');
$article = $articlesTable->newEntity();
```

```
$article->title = 'A New Article';
$article->body = 'This is the body of the article';

Atualizar
$articlesTable = TableRegistry::get('Articles');
$article = $articlesTable->get(12); // Return article with id 12

$article->title = 'CakePHP is THE best PHP framework!';
$articlesTable->save($article);
```

Comunicação entre Model e Controller

Criar function no model:

```
    public function teste()
    {
        return 'Funciona';
    }
```

Chamar no Controller:

```
use Cake\ORM\TableRegistry;

        $articles = TableRegistry::get('Articles');
        $func=$articles->teste();// Redebe do Model
        $this->set('func',$func); // Envia para a view
```

Comunicação entre Controller e View/Template

No controller/action:

```
$this->set('func',$func);
```

Na view

```
print $func
ou
debug($func)
```

View outras recebem objetos

```
foreach($clientes as $cliente){
    print $cliente->id;
    print $cliente->name;
}
```

Contagem e outros

```
$query = $this->Model->find()
   ->where([
```

```
        'category_id'=>1,
        'ativo'=>1
    ])->order('created DESC');

debug($query->all());
debug($query->first());
debug($query->count());


$query = $this->Model->find()
    ->where([
        'category_id'=>1,
        'ativo'=>1
    ])
    ->order('created DESC')
    ->contain([
        'Category'
    ]);

//ou com get()
$data = $this->Model->get(1, [
    'contain'=>['Category']
]);
```

17 – Aplicativos de Exemplo

17.1 – CakePHP Olá

Instalação e Teste de Aplicativo Mínimo no CakePHP 3.1.6

Criar a estrutura básica do aplicativo (de dentro do diretório web)

cd /backup/www/cake
composer create-project --prefer-dist cakephp/app ola

Obs.: dessa forma, com o composer, é mais prático para a criação da estrutura básica. Ainda mais prático que fazendo o download manual e descompactando no diretório web.

Chamar pelo navegador
http://localhost/cake/ola

Classes customizadas:

Controller
Ola

src/Controller/OlaController.php:

```
    $routes->connect('/', ['controller' => 'Ola', 'action' => 'index']);
```

```
<?php
namespace App\Controller;
use App\Controller\AppController;

class OlaController extends AppController
{
    public function index()
    {

    }
}
```

View
Template/OlaController/index.ctp:

```
<h1>Olá Mundo do Cake</h1>
```

Configurações
config/routes.php Ajustar o controller inicial para:

```
    $routes->connect('/', ['controller' => 'OlasController', 'action' => 'index']);
```

Este action diz que quando o usuário chamar o aplicativo pelo raiz o action index do OlasController será chamado.

config/app.php

Customizando as views

Mudando o CSS
Edite o base.css ou cake.css em webroot/css inserindo no início:

```
#menu-box{
width:150px;
float:left;
border-right:1px solid #CCCCCC;
}
#content-box{
margin-left:10px;
width:700px;
float:left;
border:1px solid #CCCCCC;
padding:10px;
background-color:#F3F3F3;
}
```

Na view troque as classes pelo nosso id, assim:

```
<div id="menu-box">
    <ul class="side-nav">
        <li class="heading"><?= __('Actions') ?></li>
        <li><?= $this->Html->link(__('New Cliente'), ['action' => 'add']) ?></li>
        <li><?= $this->Html->link(__('List Pedidos'), ['controller' => 'Pedidos', 'action' =>
'index']) ?></li>
        <li><?= $this->Html->link(__('New Pedido'), ['controller' => 'Pedidos', 'action' =>
'add']) ?></li>
    </ul>
</div>
```

17.2 – Blog

Criando um Simples Blog em CakePHP 3

Pré-Requisitos

PHP 5.5.9 com pdo_mysql, intl e mbstring (no terminal digite php -v)
Apache 2
MySQL 5

Caso tenha o cRUL instalado use:
curl -s https://getcomposer.org/installer | php
sudo mv composer.phar /usr/local/bin/composer

Instalar no Windows
http://book.cakephp.org/3.0/en/installation.html

Receber/criar uma nova aplicação do Cake:
composer create-project --prefer-dist cakephp/app cakeblog
O comando acima criou a estrutura de arquivos e diretórios básicos de um aplicativo chamado cakeblog.

Permissões
Os diretórios /tmp e /logs juntamente com seus subdiretórios são usados internamente pelo Cake para operações internas.
Eles precisam ter permissão de escrita para o usuário do servidor web.

No terminal acesse a pasta do aplicativo cakephp e execute:
chown -R www-data tmp
chown -R www-data logs

Criar o banco cake_blog com:

```
/* Primeiro, criamos a tabela articles: */
CREATE TABLE articles (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(50),
    body TEXT,
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);

/* Então inserimos articles para testes: */
INSERT INTO articles (title,body,created)
    VALUES ('The title', 'This is the article body.', NOW());
INSERT INTO articles (title,body,created)
    VALUES ('A title once again', 'And the article body follows.', NOW());
INSERT INTO articles (title,body,created)
    VALUES ('Title strikes back', 'This is really exciting! Not.', NOW());
```

Configurar o banco em config/app.php

```
return [
    // Mais configurações acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'root',
            'password' => '',
            'database' => 'cake_blog',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Mais configurações abaixo.
];
```

Ainda no app.php altere o salt:

```
'Security' => [
    'salt' => 'algum valor longo contendo uma mistura aleatória de valores.',
],
```

Configure seu Apache para que tenha suporte ao mod_rewrite.

Model
Após criar um model (modelo) no CakePHP, nós teremos a base necessária para interagirmos com o banco de dados e executar operações.

Os arquivos de classes, correspondentes aos models, no CakePHP estão divididos entre os objetos Table e Entity. Objetos Table provêm acesso à coleção de entidades armazenada em uma tabela e são alocados em src/Model/Table.

Configurando rotas:

config/routes.php

```
    $routes->connect('/', ['controller' => 'Articles', 'action' => 'index']);
```

Gerando código com bake:

bin/cake bake all Articles

Executar para criar um arquivo de migração:
bin/cake bake migration CreateArticles title:string body:text category_id:integer created modified

Criar uma tabela de categorias:
bin/cake bake migration CreateCategories parent_id:integer lft:integer[10] rght:integer[10] name:string[100] description:string created modified

Criar as tabelas
bin/cake migrations migrate

Abra o arquivo src/Model/Table/ArticlesTable.php e adicione o seguinte:

```
// src/Model/Table/ArticlesTable.php
namespace App\Model\Table;
use Cake\ORM\Table;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
        // Just add the belongsTo relation with CategoriesTable
        $this->belongsTo('Categories', [
            'foreignKey' => 'category_id',
        ]);
    }
}
```

Gerar código
bin/cake bake all Categories

17.3 – Aplicativo com Bootstrap

Criação de Layouts

Esta versão cria 3 novos layouts e os aplica respectivamente aos usuários:
super
admin
manager

- Baixei os arquivos:
bootstrap.css, jquery.min.js e bootstrap.css e alterei o plugin TwitterBootstrap para que os use offline

https://github.com/CakePHPBrasil/TwitterBootstrap

composer require cakephp-brasil/twitter-bootstrap

bin/cake plugin load CakephpBrasil

Em src/View/AppView.php adicionar dentro de initialize(), desta forma:

```
public function initialize()
{
    $this->loadHelper('Form', ['className' => 'TwitterBootstrap.Form']);
}
```

No AppController.php
$this->viewBuilder()->theme('TwitterBootstrap');

$this->set('project_name', 'Título que você quer');
Para alterar o menu superior direito

Crie um arquivo
src/Template/Element/nav-bar-right.ctp
Você pode copiar o modelo dentro de
vendor/cakephp-brasil/twitter-bootstrap/src/Template/Element

Para alterar o menu superior esquerdo

Crie um arquivo
src/Template/Element/nav-bar-left.ctp
Você pode copiar o modelo dentro de
vendor/cakephp-brasil/twitter-bootstrap/src/Template/Element

Gerar o código
bin\cake bake all MyModel --theme TwitterBootstrap

Para testar:

$ cd path-to-project
$ bin/cake TwitterBootstrap.publish

Ou:

$ cd path-to-project
$ bin/cake TwitterBootstrap.publish all

- Copia do arquivo app/webroot/css/bootstrap.css para bootstrap.super.css, bootstrap.admin.css e
bootstrap.manager.css (todos no webroot/css)
Editei cada um dos que criei e apenas alterei a cor de fundo do body na linha do background-color abaixo:

```
body {
  font-family: "Helvetica Neue", Helvetica, Arial, sans-serif;
  font-size: 14px;
  line-height: 1.42857143;
  color: #333;
  background-color: #B78999;
}
```

- Copiei o arquivo
/vendor/cakephp-brasil/twitter-bootstrap/src/Template/Layout/default.ctp
para super.ctp, admin.ctp e manager.ctp

- Editei cada um dos 3 criados (super.ctp, admin.ctp e manager.ctp) e alterei a linha:
```
    <?= $this->Html->css('bootstrap');?>
```
para
```
    <?= $this->Html->css('bootstrap.super');?>
    <?= $this->Html->css('bootstrap.admin');?>
    <?= $this->Html->css('bootstrap.manager');?>
```
- Adicionei ao initialize() do AppController.php:

```
        $this->set('project_name', 'Título do Aplicativo');
        if($loguser == 'super'){
            //$this->viewBuilder()->theme('TwitterBootstrap');
            $this->viewBuilder()->layout('TwitterBootstrap.super');
        }elseif($loguser == 'admin'){
            $this->viewBuilder()->layout('TwitterBootstrap.admin');
        }elseif($loguser=='manager'){
            $this->viewBuilder()->layout('TwitterBootstrap.manager');
        }elseif($loguser=='user'){
            $this->viewBuilder()->layout('TwitterBootstrap.default');
        }
```
Assim, ao logar cada usuário terá seu próprio ambiente.
Claro que falta fazer um bom acabamento nos layouts, nas cores.

17.4 – Aplicativo na Unha

Criando aplicativo no CakePHP quase na Unha, inteiramente na linha de código

Para aprender mais sobre a codificação e como as informações caminham numa aplicação, iremos criar um pequenino aplicativo inteiramente digitando o código, na unha.

Criaremos apenas um Controller (Users) e as ViewsTemplates: index e view.

Vamos começar criando o Controller.

No Cake 2.x um modelo era composto por apenas uma classe Tables. No Cake 3 temos uma classe Table e uma Entity,
cada uma em seu respectivo diretório (Table e Entity).
Uma classe Table guarda informações sobre uma tabela, associações, validações e behaviors e uma Entity guarda informações sobre entidades/registros.

Bem, para não dizer que não tivemos ajuda criemos o Controller com o Bake, mas, sem os actions, somente a classe vazia.

Instalação do Cake

Instalar Cake criando aplicatino scratch

Criando comp.bat para criar aplicativos

Para tornar a coisa mais prática eu criei um arquivo .bat com o comando do composer para criar aplicativo.

Apenas criei um arquivo comp.bat e coloquei dentro esta linha:

```
php ..\composer.phar create-project --prefer-dist cakephp/app %1
```

Salvei na pasta c:\xampp\htdocs\cake, que é a pasta onde crio os aplicativos cake.
Como o composer.phar está no htdocs ficou com os ..
Como quero sempre passar o nome do aplicativo que quero criar como parametro usei o %1.

Para criar um aplicativo, por exemplo, uso:
cd\xampp\htdocs\cake
comp scratch

Criar e Configurar banco scratch

```
CREATE TABLE `users` (
  `id` int(10) UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `role` varchar(20) DEFAULT NULL,
  `created` datetime DEFAULT NULL,
```

```
  `modified` datetime DEFAULT NULL
) ENGINE=InnoDB;

INSERT INTO `users` (`id`, `username`, `password`, `role`, `created`, `modified`) VALUES
(1, 'ribafs', 'ribafs', 'admin', NULL, NULL),
(2, 'fatima', 'fatima', 'user', NULL, NULL),
(3, 'tiago', 'tiago', 'admin', NULL, NULL),
(4, 'elias', 'elias', 'user', NULL, NULL);
```

Criar banco e configurar aplicativo para o banco, contendo apenas a tabela users.

Configurar banco em config/app.php

Criar Controller sem actions

```
cd\xampp\htdocs\cake\scratch
bin\cake bake controller Users --no-actions
```

Com o comando acima o bake criou apenas isto:

```
<?php
namespace App\Controller;
use App\Controller\AppController;

class UsersController extends AppController
{
}
```

Se chamarmos no navegador com
http://localhost/cake/scratch/

Veremos apenas a view Page mostrada pelo action Display.

Se chamarmos assim:
http://localhost/cake/scratch/users/

Ele vai nos avisar que falta o action index().

Criar um método público index() (action)

Então vamos criar o action index() dentro da classe UsersController.
Nossa classe deverá ficar assim:

```
class UsersController extends AppController
{
    public function index()
    {
        echo "Action index()";
        exit();
    }
```

```
}
```

Se chamarmos novamente no navegador
http://localhost/cake/scratch/users/

Agora nos mostrará a frase do echo.

Se chamarmos assim:
http://localhost/cake/scratch/users/index
Será semelhante

Criar o action view()

Adicionar abaixo do index():

```
    public function view()
    {
        echo "Action view()";
        exit();
    }
```

Chamar pela URL
http://localhost/cake/scratch/users/view

Agora vamos criar o Template para mostrar as informações. Ao invés de mostrar mensagens com echo diretamente no controller vamos fazer isso pelas views do Template, que serão chamadas pelos actions. Exemplo: o usuário digitou o endereço de um action de um controller, então este action chama a respectiva view do template.
Criar a view index.ctp (ctp - Cake Template Page)
Criar a pasta Users na pasta Template (deve ser o mesmo nome do controller)
Criar dentro da pasta Users o arquivo index.ctp e apenas digitar nele:
<h1>Index</h1>

Vá até o action index() do controller e comente suas duas linhas, o echo e o exit();

Agora acesse pelo navegador:
http://localhost/cake/scratch/users/
ou
http://localhost/cake/scratch/users/index

Nos mostra Index e a página já vem com um cabeçalho e rodapé.

Veja que automaticamente o action index chama a views index.ctp.

Outra forma para view():

```
   public function view($id = null)
   {
      $this->viewBuilder()->layout('layout_admin');
      $user = $this->Users->get($id, [
```

```
            'contain' => []
        ]);
        $this->set('user', $user);
    }
```

Agora façamos algo mais interessante:

No action index() remova as linhas existentes e adicione estas duas:

```
    // Recuperar todos os usuários cadastrados
    $users = $this->Users->find('all');
    $this->set('users',$users);
```

Assim:

```
public function index()
{
    // Recuperar todos os usuários cadastrados, do método Users do model
    $users = $this->Users->find('all');
    $this->set('users',$users);   // O método set deixa o objeto $users disponível na view
index.ctp
}
```

O método set é destinado a armazenar o objeto $users na variável que será recebida pela respectiva view, no caso a index.ctp.

No CakePHP 2 trabalhávamos com arrays e no Cake 3 trabalhamos com objetos.

Uma alternativa o index() em apenas uma linha, mas o mesmo código:

```
public function index()
{
    $this->set('users', $this->Users->find('all'));
}
```

Paginação de Resultados

Para paginar os resultados no Cake apenas altere a primeira linha do action index() para:

```
    $users = $this->paginate($this->Users);
```

Para configurar a paginação na view usaremos o bake para gerar o código para nós.
Antes elimine o diretório Template\Users.
bin\cake bake all Users

Assim ele gerou um aplicativo básico mas completo.
Poderemos ver no Template\Users\index.ctp ao ser visualizado no navegador que ele implementou a paginação.
Mesmo que tenha poucos registros, ainda assim poderá ver abaixo os links <previous e next>.

Agora nós podemos ver um código completo do aplicativo.

Veja que ele implementou uma grande quantidade de bons recursos: paginação, ordenamento de colunas, template com CSS, menu lateral, etc.

Criação de view index.ctp

Crie o diretório src/Template/Users e dentro dele a index.ctp

Código bem simples:

```
public function index()
{
    $this->set('users', $this->paginate($this->Users));
}
```

Agora vamos melhorar a view src/Template/Users/index.ctp para receber as informações do index(), encluindo a linha digitada e adicionando estas:

```
<ul>
    <?php foreach($users as $user){?>
    <li>
        <?php print $user->username; ?>
    <?php }?>
    </li>
</ul>
```

Outra

```
<table>
    <?php foreach($users as $user){?>
    <tr>
        <td><?php print $user->username; ?></td><td><?php print $user->role; ?></td>
    </tr>
    <?php }?>
</table>
```

Versão melhorada:

```
<table>
   <?php foreach ($users as $user): ?>
   <tr>
      <td><?= $this->Number->format($user->id) ?></td>
      <td><?= h($user->username) ?></td>
      <td><?= $user->has('group') ? $this->Html->link($user->group->name, ['controller' =>
'Groups', 'action' => 'view', $user->group->id]) : '' ?></td>
          <td><?= h($user->role) ?></td>
      <td><?= h($user->created) ?></td>
      <td><?= h($user->modified) ?></td>
      <td>
         <?= $this->Html->link(__('View'), ['action' => 'view', $user->id]) ?>
         <?= $this->Html->link(__('Edit'), ['action' => 'edit', $user->id]) ?>
         <?= $this->Form->postLink(__('Delete'), ['action' => 'delete', $user->id], ['confirm'
=> __('Tem certeza de que deseja excluir # '.$user->username.'?', $user->id)]) ?>
```

```
        </td>
    </tr>
    <?php endforeach; ?>
</table>
```

Versão a versão criada pelo bake (remova password, created e modified).

Criar o action add()

Adicionar abaixo do view() no controller UsersController:

```
    public function add()
    {
        $user = $this->Users->newEntity();
        $this->set(compact('user'));
    }
```

Adicionar a view add.ctp ao src\Template\Users contendo:

Usaremos o Helper Form e alguns métodos: create, input, button e end

```
<h2>Novo Usuário</h2>

<?php
    echo $this->Form->create($user);
    echo $this->Form->input('username');
    echo $this->Form->input('password');
    echo $this->Form->input('role',
['options'=>['admin'=>'Administrador','user'=>'Usuário']]);
    echo $this->Form->input('created');
    echo $this->Form->input('modified');
    echo $this->Form->button('Criar Usuário');
    echo $this->Form->end();
```

Outra alternativa (do tutorial Blog):

```
<div class="users form">
<?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Adicionar Usuário') ?></legend>
        <?= $this->Form->input('username') ?>
        <?= $this->Form->input('password') ?>
        <?= $this->Form->input('role', ['options' => ['admin' => 'Admin', 'author' => 'Author']]) ?
>
        <?= $this->Form->input('created') ?>
        <?= $this->Form->input('modified') ?>
    </fieldset>
<?= $this->Form->button(__('Adicionar')); ?>
<?= $this->Form->end() ?>
</div>
```

Melhorando add()

Voltando ao action add(), vamos adicionar uma linha entre as duas existentes:

debug($this->request->data);

Ficará assim:

```
    public function add()
    {
        $user = $this->Users->newEntity();
        debug($this->request->data);
        $this->set(conpact('user'));
    }
```

Se executar agora o método add, preencher o formulário e clicar no submit verá um array acima com todos os campos e seus valores, resultante
da função debug.

Comentemos a linha do debug e criemos outra logo abaixo, deixando assim:

```
    public function add()
    {
      $user = $this->Users->newEntity();
      if ($this->request->is('post')) {
          $user = $this->Users->patchEntity($user, $this->request->data);
          if ($this->Users->save($user) && ($user->username != null)) {
              $this->Flash->success(__('O usuário foi criado corretamente.'));
              return $this->redirect(['action' => 'index']);
          }
          else
          {
           $this->Flash->error(__('O usuário não pode ser criado. Username, senha e role
são requeridos. Tente novamente!'));
          }
       }
       $this->set('user', $user);
   }
```

O método patchEntity valida os dados do registro
Assim nosso novo usuário pode ser cadastrado corretamente, ou quase, visto que falta criptografar sua senha.

bin\cake bake template Users add

Para ver como é o código padrão.

Adicionando Criptografia nas Senhas no Model Entity
Isso será feito no Model, especificamente na classe em
src\Model\Entity\User.php

Vamos usar o bake para criar:
bin\cake bake model Users

Edite o arquivo acima e...
logo abaixo da linha:
use Cake\ORM\Entity;

Digite
use Cake\Auth\DefaultPasswordHasher;

A versão 3 do Cake usa o algoritmo bCrypt para a criptografia de senhas.
Adicionar ao final da classe User o método abaixo:

```
    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($value);
    }
```

Assim nossos usuários serão corretamente salvos com senha criptografada.

As mensagens mostradas nas views do Cake são criadas pela classe flash e seus métodos. Seu código fica no diretório:
src/Template/Element/Flash

Podemos criar mensagens personalizadas, para tanto criar um arquivo no diretório Flash acima com as nossas preferências.
Podemos copiar um dos arquivos error.ctp ou success.ctp e renomear para o nosso e mudar seu conteúdo e passar a usar na chamada da classe Flash.

Action view() simples:

```
  public function view($id)
  {
    $user = $this->Users->get($id);
    $this->set(compact('user'));
  }
```

View view.ctp

Simples

```
<h1>Usuário</h1>
  <table class="vertical-table">
        <tr><td><?= $this->Number->format($user->id) ?></td></tr>
    <tr><td><?= h($user->username) ?></td></tr>
    <tr><td><?= h($user->created) ?></td></tr>
    <tr><td><?= h($user->modified) ?></td></tr>
  </table>
```

Agora crie uma view pelo bake e sobrescreva a existente para comparar os códigos.
bin\cake bake template Users view

Geré o código do menu lateral dentro da tag <nav> e abaixo mostrando os campos dentro da tabela).

Action edit()

Código simples:

```
public function edit($id = null)
{
    $user = $this->Users->get($id);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $user = $this->Users->patchEntity($user, $this->request->data);
    }
    $this->set(compact('user'));
}
```

Código gerando com o bake:
bin\cake bake controller Users

```
public function edit($id = null)
{
    $user = $this->Users->get($id, [
        'contain' => []
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $user = $this->Users->patchEntity($user, $this->request->data);
        if ($this->Users->save($user)) {
            $this->Flash->success(__('The user has been saved.'));
            return $this->redirect(['action' => 'index']);
        } else {
            $this->Flash->error(__('The user could not be saved. Please, try again.'));
        }
    }
    $this->set(compact('user'));
    $this->set('_serialize', ['user']);
}
```

Criação da view edit.ctp

Código simples

```
<?= $this->Form->create($user) ?>
<fieldset>
    <legend><?= __('Edit User') ?></legend>
    <?php
```

```
            echo $this->Form->input('username');
            echo $this->Form->input('password');
            echo $this->Form->input('role');
        ?>
    </fieldset>
    <?= $this->Form->button(__('Submit')) ?>
    <?= $this->Form->end() ?>
```

Código gerado pelo bake:

```
<nav class="large-3 medium-4 columns" id="actions-sidebar">
    <ul class="side-nav">
        <li class="heading"><?= __('Actions') ?></li>
        <li><?= $this->Form->postLink(
                __('Delete'),
                ['action' => 'delete', $user->id],
                ['confirm' => __('Are you sure you want to delete # {0}?', $user->id)]
            )
        ?></li>
        <li><?= $this->Html->link(__('List Users'), ['action' => 'index']) ?></li>
    </ul>
</nav>
<div class="users form large-9 medium-8 columns content">
    <?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Edit User') ?></legend>
        <?php
            echo $this->Form->input('username');
            echo $this->Form->input('password');
            echo $this->Form->input('role');
        ?>
    </fieldset>
    <?= $this->Form->button(__('Submit')) ?>
    <?= $this->Form->end() ?>
</div>
```

Criando o action delete():

```
    public function delete($id = null)
    {
        $this->request->allowMethod(['post', 'delete']);
        $user = $this->Users->get($id);
        if ($this->Users->delete($user)) {
            $this->Flash->success(__('The user has been deleted.'));
        } else {
            $this->Flash->error(__('The user could not be deleted. Please, try again.'));
        }
        return $this->redirect(['action' => 'index']);
    }
```

Para delete não temos view, será usado apenas um link na view index.ctp para cada registro.

17.5 – Aplicativo via Código

Criar um aplicativo para exemplo de interação do codigo do CakePHP entre controllers, models e templates.

cake3_app_code

Criar um banco no postgresql com duas tabelas relacionadas

groups e users

Configurar o banco no app.php e a rota para apontar para Users e index

Testar

http://localhost/cake3_app_code

Criar um novo método com apenas uma mensagem/string

Criar um método novo no UsersTable.php

```
    public function olaModel(){
        return "Olá model do CakePHP3";
    }
```

No UsersController adicione ao início

```
use Cake\ORM\TableRegistry;
```

E crie o método ola

```
    public function ola()
    {
        $msg = TableRegistry::get('Users');
        $ola = $msg->olaModel();
        $this->set('ola',$ola);
    }
```

No Template Users cria o arquivo ola.ctp contendo

```
<h3>Mensagem criada no método olaModel Model UsersTable.php e enviada para a view ola.ctp pelo action ola() do Controller UsersController.php</h3>
<?= $ola ?>
```

Chamar pelo navegador assim:

http://localhost/cake3_app_code/users/ola

Alternativamente criar um link para a view ola na view index.ctp assim:

Abaixo de New Group

```
<li><?= $this->Html->link(__('Olá'), ['controller' => 'Users', 'action' => 'ola']) ?></li>
```

Criar um método que pega um registro da tabela users

No UsersTable.php:

Adicione ao início

```
use Cake\ORM\TableRegistry;

public function umUser()
{
  $users = TableRegistry::get('Users');
  $query = $users->find();
  $user=$query->where(['username' => 'super']);
  return $user;
}
```

No UsersController.php

```
public function umUser()
{
      $user = TableRegistry::get('Users');
      $super = $user->umUser();
      $this->set('super',$super);
}
```

No Template/Users criar um_user.ctp contendo

```
<?php
//
?>
<h3>Usuário recebido do model através do controller</h3>
<?php
foreach($super as $campos){
   print ' ID - '.$campos->id.'<br>';
   print ' Grupo - '.$campos->group_id.'<br>';
   print ' Username - '.$campos->username.'<br>';
}
```

Testar

http://localhost/cake3_app_code/users/um_user

Adicionar o link abaixo para o action Users/index.ctp

```
<li><?= $this->Html->link(__('Um usuário'), ['controller' => 'Users', 'action' =>
'um_user']) ?></li>
```

17.6 - CakePHP3 Treinamento

Criação de alguns aplicativos

Introdução

Nosso principal objetivo aqui é o de criar aplicativos de forma simples e prática e também que eles tenham um bom controle de acesso, permitindo apenas aos usuários determinados acessar determinadas partes do aplicativo.

Iremos fazer isso por etapas, de forma organizada.

Usaremos principalmente o Framework CakePHP em sua versão 3. O CakePHP é um framework bem popular entre os demais existentes e ajuda muito na criação de aplicativos. Ele conta com muitos e bons recursos e usa a linguagem PHP, um servidor Web como o Apache ou outros e um SGBD como o MySQL ou outros. Também usa orientação a objetos e alguns padrões de projeto especialmente o MVC e o ORM. O foco do CakePHP é nas convenções. Você segue as convenções que ele determina e ele facilita sua vida.

Pré-requisitos

De conhecimentos
Desejáveis – conhecimento da linguagem PHP, de orientação a objetos e de padrões de projeto.

Obrigatório – boa motivação e vontade de aprender.

De Hardware
Praticamente qualquer computador que esteja funcionando pode ser usado para nosso treinamento.

De Software
Praticamente qualquer sistema operacional pode ser utilizado.
Precisamos basicamente do Apache, PHP e MySQL mais alguns módulos/extensões.

Servidor

O servidor precisa ter Apache, PHP e MySQL no mínimo. Eu prefiro criar este servidor em meu computador desktop, no Linux (preferido) ou no Windows, mas podem ser usados outros SOs.

Podemos usar um servidor online daqueles que já vem com editor e todo o ambiente para programação, como é o caso do Cloud9 (http://c9.io). Este exige conhecimento sobre a linha de comando do Linux, mas mostrarei um guia em seguida. Caso queira experimentar, ele oferece um plano free:
https://c9.io/signup

Podemos usar o Xampp (http://xampp.sf.net), o Wampp, o EasyPHP ou outro no Windows.

No Linux prefiro usar os pacotes da distribuição.

Para começar você precisa já ter um ambiente pronto e configurado com Apache, PHP e MySQL e extensões. Caso não tenha e não saiba como fazer provavelmente este treinamento não é para você Ainda, a não ser que seja daqueles corajosos que tenham uma grande motivação para aprender e queira experimentar. Se for seu caso, procure um tutorial de como instalar o Xampp no Windows ou outro.

Pré-requisitos para Instalar o CakePHP 3.2.9

- Apache2 ou outro servidor web
- PHP 5.5.9 ou superior
- Módulos do Apache: mod_rewrite
- Extensões: mcrypt, php5-mcrypt, intl, mbstring

Instalação do Ambiente

Use o Xampp no Windows e os pacotes no Linux.

Instalação do CakePHP pelo Composer

No Linux
curl -sS https://getcomposer.org/installer | php

sudo mv composer.phar /usr/local/bin/composer

sudo nano /usr/local/bin/comp

Adicionar a linha:
composer create-project --prefer-dist cakephp/app $1

Execute
sudo chmod +x /usr/local/bin/comp

Criando o aplicativo:
cd /var/www/html/treinamento
comp app1

Acessando
http://localhost/treinamento/app1

No Windows
Download - https://getcomposer.org/Composer-Setup.exe
Instale

Veja detalhes sobre como adicionar o php ao path.

Criar o arquivo bat comp.bat na pasta c:\xampp\htdocs, contendo a linha:

composer create-project --prefer-dist cakephp/app %1
Criando o Aplicativo
Abrir o prompt
cd c:\xampp\htdocs\treinamento
comp app1

Acessando
http://localhost/treinamento/app1

Primeiro Aplicativo

Este primeiro aplicativo apenas instala o CakePHP 3.2.9 através do composer e mostra o aplicativo padrão que já vem com com o Cake. Vem um controller, um action e um template.

Customizando o Terminal/Prompt

Eu gosto de configurar para que ele já abra no diretório onde trabalho.

No Windows
Clique com o botão direito – propriedades – Iniciar em c:\xampp\htdocs\treinamento
Aproveita e entra com uma combinação de teclas tipo Ctrl+Alt+C

No Linux
Crie um atalho para o terminal assim:
Configurações do sistema – Teclado – Atalhos de teclado – Atalhos personalizados
Clicar no sinal de +
Nome – Terminal

Comando - gnome-terminal –working-directory=/var/www/html/treinamento
Teclar Enter
Adicionar o atalho – Ctrl+Alt+C

Agora sempre que teclar Ctrl+Alt+C ele abre nosso ambiente já no respectivo diretório.

Alterando o document root do Apache no Linux

Em meu computador desktop no Linux eu gosto de mudar do /var/www/html para /backup/www, backup é minha partiçãod e backup e como ribafs sou o dono dela e assim fica mais confortável para trabalhar.

Criando o aplicativo app1

Abrir o terminal e
comp app1

Com este comando, aguardamos um pouco e ele traz o CakePHP atualizado para sua última versão estável juntamente com todas as dependências.

Vejamos. Execute o comando:
cd app1
bin/cake

Mude a barra para \ caso esteja usando o Windows.
Ele mostra algumas informações importantes, inclusive a versão do cake, a do php e alguns diretórios:

Welcome to CakePHP v3.2.9 Console

Path: /backup/www/treinamento/app1/src/
PHP : 5.6.11-1ubuntu3.4
---
* app: src – neste subdiretório fica todo o código do aplicativo
Abaixo fica todo o código do framework
* core: /backup/www/treinamento/app1/vendor/cakephp/cakephp

Acesse o aplicativo padrão embutido no Cake pelo navegador:

http://localhost/treinamento/app1

Já veremos uma página padrão criada pelo controller padrão e seu template.
Como podemos ver que arquivos são os responsáveis por isso? São muitos, mas os principais são
src/Controller/PagesController.php
src/Template/home.ctp

Este aplicativo não tem um banco de dados, usa apenas página estática.

Convenções do CakePHP 3

Sempre que for encontrando oportunidade irei abordando as convenções necessárias e importantes, mas caso queira se antecipar veja a respectiva seção ou o site oficial:
http://book.cakephp.org/3.0/en/intro/conventions.html

Segundo Aplicativo

Nome de banco de dados – tuto em minúsculas e no singular. Palavras compostas separadas por sublinhado.
Nome de tabela – tudo em minúsculas e no plural.

Para este primeiro aplicativo útil criaremos um banco chamado cadastro contendo apenas uma tabela chamada clientes. Depois de configurar o banco no aplicativo geraremos o código do aplicativo usando o bake, que gera um CRUD completo e com bons recursos.

Banco de Dados

Banco – cadastro
Tabela – clientes

Criar o banco no phpmyadmin e colar o script abaixo:

```
CREATE TABLE IF NOT EXISTS `clientes` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nome` char(45) NOT NULL,
  `email` varchar(50) DEFAULT NULL,
  `data_nasc` date DEFAULT NULL,
  `cpf` char(11) NOT NULL,
  PRIMARY KEY (`id`)
);

INSERT INTO `clientes` (`id`, `nome`, `email`, `data_nasc`, `cpf`) VALUES
(1, 'Erin Pate Skinner', 'dolor.vitae.dolor@mollisvitaeposuere.ca', '2013-10-07',
'74426302285'),
(2, 'Leonard Martinez Hays', 'dignissim.magna.a@dolorvitae.org', '2012-08-22',
'75278965048'),
(3, 'Aladdin Curry French', 'eu.augue@eutemporerat.org', '2012-10-28', '10376915676'),
(4, 'Chloe Macdonald Dalton', 'parturient.montes@Mauris.ca', '2013-05-12',
'64444679077'),
(5, 'Mallory Sweet Strong', 'lorem@fringillaporttitor.ca', '2013-05-19', '15687101505'),
(6, 'Jermaine Pierce Woodward', 'mi.pede.nonummy@molestiearcu.ca', '2013-03-22',
'36712502261'),
(7, 'Bell Raymond Pruitt', 'dignissim.tempor.arcu@nuncac.org', '2013-03-09',
'64629428663'),
(8, 'Lydia Bell Whitfield', 'Sed@semper.com', '2013-12-02', '41962749289'),
(9, 'Tad Mason Graham', 'elit.erat@vestibulum.edu', '2012-06-08', '05642745964'),
(10, 'Felix Bradshaw Mccray', 'dui@elitCurabitursed.edu', '2013-09-16', '82071617437'),
(11, 'Idona Jensen Garrett', 'sem@Crasvulputate.com', '2014-01-08', '07941004794'),
(12, 'Wayne Ray Padilla', 'luctus.felis.purus@nonjustoProin.org', '2014-04-03',
'60934465323'),
(13, 'Nelle Finch Cantu', 'placerat.eget@Donec.ca', '2012-05-29', '64704574060'),
(14, 'Maite Emerson Best', 'dui.augue@quisdiam.com', '2014-04-01', '04531857574'),
(15, 'Jada Holman Wilkins', 'dolor@tristiquealiquet.com', '2013-01-11', '88994190741'),
(16, 'Beverly Lane Lindsay', 'et.euismod@ametfaucibusut.com', '2013-10-22',
'40194697135'),
```

(17, 'Hayden Clayton Foreman', 'enim@aliquamenimnec.edu', '2013-04-16', '72583040904'),
(18, 'Hadassah Leonard Key', 'dui.quis@augueidante.com', '2013-04-07', '72626859924'),
(19, 'Adrian Ballard Peters', 'enim.Curabitur@faucibus.com', '2012-07-13', '50918748283'),
(20, 'Phyllis Richmond Wynn', 'eget.laoreet@justoeuarcu.org', '2013-07-01', '62712888794'),
(21, 'Amelia Baird Barrera', 'id.ante@dignissim.org', '2012-06-09', '12106836368'),
(22, 'Whitney Mack Lamb', 'quam.Curabitur.vel@PraesentluctusCurabitur.org', '2012-06-26', '52403407001'),
(23, 'Myra Mcmahon Valentine', 'ac.mi@fringillami.edu', '2012-07-27', '42961419194');

ALERTA – tome cuidado ao copiar e colar de processadores de texto como o Word ou Writer, pois algumas vezes eles alteram o texto. Um exemplo é este: --, que o writer gosta de mudar para –.

Criando o aplicativo

Ctrl+Alt+C
comp app2

Configurações

Configurar o banco de dados. Editar o arquivo app2/config/app.php

Alterar apenas 3 linhas:

Em
```
    'Datasources' => [
        'default' => [
```

Altere as linhas:

```
            'username' => 'root',
            'password' => '',
            'database' => 'cadastro',
```

Salve e feche

Configure as rotas. Editar o arquivo app2/config/routes.php

Copiar e linha:
```
    $routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Colar logo acima e alterar assim:
```
    $routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Com isso estamos dizendo que o raiz do nosso aplicativo será respondido pelo action index() do controller Clientes, assim, ao abrir http://localhost/treinamento/app2 ele mostrará a view index.ctp do template clientes, chamada pelo action index() do controller Clientes.

Agora, chamando pelo navegador:
http://localhost/treinamento/app2

Como ele procura o controller Clientes e não encontra, então mostra a mensagem de erro e já ajuda com a mesma. Então vamos criar o código do nosso aplicativo: controllers, models e views.

Gerando o Código do Aplicativo

Abra o terminal, acesse o diretório app2 e execute o comando abaixo:

bin/cake bake all clientes

Observe as mensagens e veja que com isso o bake gerou o código básico de um CRUD, com o controller, o model e a view, criando assim um aplicativo simples mas funcional.

Acesse novamente e veja como ficou. Navegue pelas seções do site.

Veja que já aplicou paginação, CSS e todo um layout adequado para a gente trabalhar.

Experimente adicionar, editar, remover e visualizar registros. Como também mudar para a segunda página e reordenar as colunas de campos clicando no título e bem mais. E mais ainda existe a nossa disposição no framework.

Mais detalhes sobre a geração de código com Bake, veja a seção respectiva ou o site oficial:
http://book.cakephp.org/3.0/en/bake/usage.html

Terceiro Aplicativo

Agora vamos adicionar 3 tabelas: groups, users e privileges. A grande intenção destas tabelas é criar uma área restrita para usuários, ou melhor, criar um controle de acesso, liberando apenas certos usuários para certas seções do aplicativo.

Criar o aplicativo

Ctrl+Alt+C
comp app4

Banco de Dados

Banco – app3
Tabelas: clientes, groups e users.
Crie o banco e importe o script app3.sql

Configurações

app3/config/app.php

Altere:

```
'username' => 'root',
'password' => '',
'database' => 'app3',
```

app3/config/routes.php

Adicione a linha, acima da existente:

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Gerando o código

cd /backup/www/treinamento

bin/cake bake all clientes
bin/cake bake all groups
bin/cake bake all users

Acesse pelo navegador

http://localhost/treinamento/app3

Experimente acessar
http://localhost/treinamento/app3/users/login
Já estão lá, os actions login() e logout() e a view login.ctp.

Navegue entre as tabelas/controllers assim:
http://localhost/treinamento/app3 – clientes
http://localhost/treinamento/app3/users
http://localhost/treinamento/app3/groups
Mais a frente adicionaremos um menu para tornar isso mais prático. O Cake tem um recurso chamado Element, que pode ser usado para adicionar um menu ao aplicativo, bastando apenas que adicionemos o arquivo e um link ao layout.

Quarto Aplicativo

Agora é a hora de conhecer melhor o CakePHP, conhecer seu código. Vamos começar a criação de um código para controle de acesso. Para isso vamos criar uma função no controller AppController. Este controller fica em:
src/Controller/AppController.php

Depois de pronto o código que iremos implementar, moveremos para um componente. Mais a frente criaremos um plugin contendo o código das tabelas groups, users e privileges e também o componente e algo mais. Este plugin controlará o acesso ao nosso aplicativo de forma bem flexível. Poderemos definir que ations cada usuários poderá acessar, alterar, criar e excluir.

Criar Aplicativo

Acessar o terminal e criar o aplicativo app4:

Ctrl+Alt+C
comp app4

Customizar o bake para facilitar a geração de código do controller Users.

Por padrão o bake gera somente os actions e views default, que são: index, edit, add, delete e view. Mas quando for o caso do controller Users precisamos de mais dois actions: login e logout e de uma view login.

Altere o arquivo app3/config/bootstrap_cli.php para que fique assim:

```
<?php
use Cake\Core\Configure;
use Cake\Core\Exception\MissingPluginException;
use Cake\Core\Plugin;
// Adicionar os 3 abaixo
use Cake\Event\Event;
use Cake\Event\EventManager;
use Cake\Utility\Hash;

// Set logs to different files so they don't have permission conflicts.
Configure::write('Log.debug.file', 'cli-debug');
Configure::write('Log.error.file', 'cli-error');

try {
    Plugin::load('Bake');
} catch (MissingPluginException $e) {
    // Do not halt if the plugin is missing
}
// Adicionar daqui pra frente
EventManager::instance()->on(
    'Bake.beforeRender.Controller.controller',
    function (Event $event) {
        $view = $event->subject();
        if ($view->viewVars['name'] == 'Users') {
            // add the login and logout actions to the Users controller
            $view->viewVars['actions'] = [
                'login',
                'logout',
                'index',
                'view',
                'add',
                'edit',
                'delete'
            ];
        }
```

```
    }
);
```

Criando modelo de aplicativo

Também podemos criar um aplicativo com o composer e logo que tenha sido criado copiar seu diretório. Exemplo: criar o **app4** e copiar para **modelo**. A partir de agora iremos copiar o aplicativo **modelo** para gerar o **app5** e seguintes.

Com algum tempo nosso aplicativo pode ficar com o CakePHP desatualizado, então executaremos:

```
Ctrl+Alt+C
cd app4
composer update
```

Assim o composer atualiza nosso framework para a última versão estável.

Banco de Dados

Criar o banco app4. Importe o script app4.sql.

Usaremos o mesmo banco do app3, com mais a tabela privileges já com alguns registros cadastrados.

Agora precisamos que cada tabela adicionada, como clientes, funcionarios, produtos ou qualquer outra, contenha o campo user_id por conta do uso do componente Auth, que precisa relacionar as tabelas com users.

Configuração

Configure o app.php para o banco app4 e o routes.php da mesma forma que o app3, sendo Clientes nosso raiz.

Estrutura de diretórios

Para conhecer a estrutura de arquivos e diretórios do CakePHP 3 percorra seu app4.
Para saber detalhes sobre a mesma veja:
http://book.cakephp.org/3.0/en/installation.html

Implementando a criptografia bcrypt para as senhas dos usuários

Adicionando hash bCrypt para a Senha

Isso é feito no Model, especificamente na classe Entity User.php:

Adicione:

```
use Cake\Auth\DefaultPasswordHasher;
```

Ao final da classe adicione a função abaixo:

```
protected function _setPassword($password)
{
    return (new DefaultPasswordHasher)->hash($password);
}
```

Criar Grupos e Usuários

Vamos criar 3 usuários, um para cada grupo:

Groups	Users	Password
Admins	admin	admin
Managers	manager	manager
Users	user	user

Acesse
http://localhost/treinamento/app4/groups/add

E cadastro os 3 grupos acima.

Acesse
http://localhost/treinamento/app4/users/add

E cadastre os 3 usuários acima com as respectivas senhas.
Após cadastrar os usuários vemos um hash bem grande criado pelo algoritmo do bcrypt.
É interessante editar as views src/Template/Users/index.ctp, view.ctp e edit.ctp removendo o campo senha do form.

Criação de Funções para o Controle de Acesso

Por enquanto, para fazer as coisas mais simples, iremos criar as funções uma de cada vez, para ir testando e depois juntaremos todas em um componente. Aliás, aqui eu já mostro o código todo debugado, com os erros que encontrei corrigidos. A intenção é criar a coisa de forma modular para facilitar.

Depois criaremos o componente e moveremos as 3 funções para ele. Um componente guarda código que colabora com os controllers.

Depois então criaremos um plugin e moveremos o componente, assim como o código do groups, users e privileges para o plugin. Um plugin praticamente é um aplicativo que roda dentro do nosso aplicativo, já trazendo controllers, models, templates, configurações, etc.

Iremos criar a função populate() no src/Controller/PrivilegesController.php que cadastrará todos os actions de um grupo para uma tabela.
Ela tem como objetivo cadastrar os actions de todos os controllers para os grupos das tabelas que sejam diferentes de users, groups e privileges, para clientes e outras que adicionarmos.

Edite o arquivo:
src/Controller/PrivilegesController.php

Adicione ao seu início:
use Cake\Datasource\ConnectionManager;
use Cake\ORM\TableRegistry;

E adicione o código abaixo:

```
    public function populate($group, $controller){
        // Somente se nenhum action tiver sido cadastrado na tabela privileges para o
referido grupo
        $conn = ConnectionManager::get('default');
        //$tables = $conn->query("show tables");

        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'index')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'view')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'add')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'edit')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'delete')");
    }
```

Adicionar as linhas abaixo para o método beforeFilter:

```
$this->populate('Admins', 'Clientes');
$this->populate('Managers', 'Clientes');
```

Então chame o aplicativo no navegador apenas uma vez e comente as duas linhas acima. Caso execute novamente o aplicativo sem antes comentar as linhas verá erro de violação de integridade.

Este método/função precisa ser chamado para que seja executado, visto que não é um dos métodos nativos do CakePHP.

O beforeFilter() ficará assim:

```
   public function beforeFilter(\Cake\Event\Event $event) {
        parent::beforeFilter($event);

    $this->populate('Admins', 'Clientes');
    $this->populate('Managers', 'Clientes');
}
```

Caso adicionemos uma nova tabela, produtos por exemplo, então adicionamos a tabela, geramos o código para ela e adicionamos a linha:

```
$this->populate('Admins', 'Produtos');
```

Para o beforeFilter() do controller Privileges e executamos apenas uma vez para comentar a linha.

Estou refletindo agora e acho que podemos melhorar esta função populate(), de forma que não precisemos ficar comentando e descomentando. Podemos fazer um select inicialmente na tabela privileges de forma que verifique se o action está cadastrado e somente faça o insert se não existir.
Vejamos como ficará:

```
public function populate($group, $controller){
    $conn = ConnectionManager::get('default');

    $actions = $conn->execute("SELECT action FROM `privileges` WHERE
group_name = '$group' and controller = '$controller' and action='index'")->fetchAll();

    if(!$actions){
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'index')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'view')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'add')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'edit')");
        $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'delete')");
    }
}
```

Como está esta função somente irá cadastrar os 5 actions, caso ainda não tenhamos cadastrado nenhum action desse controller.
Importante:

Lembre de adicionar a linha abaixo ao UsersController.php:
use Cake\Event\Event;

Função Group

Função que recebe o group_name de privileges, de acordo com o group_id do usuário logado.

Retorna o group_name, se existir
Ou false se não existir

```
public function group_priv($controller,$action,$group){
    //$controller = $this->request->controller;
    //$action = $this->request->action;
```

```
        $conn = ConnectionManager::get('default');
        $group = $conn->execute("select group_name from privileges where controller =
'$controller' and action ='$action' and group_name='$group'")->fetchAll();

        if($group){
            $grouppriv = $group[0][0];
            return $grouppriv;
        }else{
            return false;
        }
    }
```

Ela será chamada de beforeFilter() por enquanto:

```
        // Chamar a função group_priv()
        if($username == 'admin'){
            $this->group_priv($controller, $action,'Admins');
        }elseif($username == 'manager'){
            $this->group_priv($controller, $action,'Managers');
        }elseif($username == 'user'){
            $this->group_priv($controller, $action,'Users');
        }
```

O beforeFilter() agora deve ficar assim:

```
    public function beforeFilter(\Cake\Event\Event $event) {
        parent::beforeFilter($event);

        $controller = $this->request->controller;
        $action = $this->request->action;
        $user = $this->request->session()->read('Auth.User');
        $username=$user['username'];

        $this->Auth->allow(['view', 'index','edit']);

        if($username == 'admin'){
            $group='Admins';
        }elseif($username == 'manager'){
            $group='Managers';
        }elseif($username == 'user'){
            $group='Users';
        }

        $denied='Acesso Negado!';
        $privilege = 'Privilégio não Cadastrado!';
        if($action != 'login' && $action != 'logout'){
            if($this->go($controller,$action,$group)==false){
                $this->Flash->error(__($denied));
                // Voltar para onde veio, ou seja, nem sair de onde está
```

```
                return $this->redirect($this->referer());
                // return $this->redirect(['controller' => 'Users','action' => 'login']);
                // $this->redirect($this->Auth->logout());
            }else{
                if($this->msg == 'priv'){
                    $this->Flash->set(__($this->privilege));
                }
            }
        }
    }
```

E nossa função go() assim:

```
    public function go($controller,$action,$groupo){
        $user = $this->request->session()->read('Auth.User');
        $username=$user['username'];
        if($username == 'admin'){
            $groupo='Admins';
        }elseif($username == 'manager'){
            $groupo='Managers';
        }elseif($username == 'user'){
            $groupo='Users';
        }

        $conn = ConnectionManager::get('default');
        $group = $conn->execute("select group_name from privileges where controller =
'$controller' and action ='$action' and group_name='$groupo'")->fetchAll();

        if($group){
            $group = $group[0][0];
        }

        if($username == 'admin' && $group=='Admins'){
            return true;
        }elseif($username == 'manager' && $group == 'Managers' && $controller ==
'Clientes'){
            return true;
        }elseif($username == 'user' && $group == 'Users' && $controller == 'Clientes' &&
($action == 'index' || $action == 'view')){
            return true;
        }else{
            return false;
        }
    }
```

Controle de Acesso
Precisamos lembrar que devemos cadastrar o acesso no controller Privileges e também ajustar o acesso devido no módulo go(). No caso, o usuário user está com acesso somente ao action index e ao view.

Quinto Aplicativo

Criando um Componente para Controlar o Acesso

Agora iremos criar o esqueleto de um componente usando o bake e mover para ele a função populate() e a função go(), para ficar algo mais profissional.

Criando o Aplicativo

Agora, para criar o aplicativo faremos uma cópia da pasta modelo para app5.

Criação do banco de dados

Criaremos o banco app5 e importaremos o script app5.sql, que contem as tabelas groups, users e privileges, além de clientes.

Configurações

Configurar o banco e as rotas.

Geração do Código do Aplicativo

cd /backup/www/treinamento/app5

bin/cake bake all groups
bin/cake bake all users
bin/cake bake all privileges
bin/cake bake all clientes

Experimente navegar pelo aplicativo

http://localhost/treinamento/app5

Adicionando o Bcrypt

Fazer como fizemos no aplicativo anterior.

Criar Grupos e Usuários

Vamos criar 3 usuários, um para cada grupo:

Groups	Users	Password
Admins	admin	admin
Managers	manager	manager
Users	user	user

Acesse
http://localhost/treinamento/app4/groups/add

E cadastro os 3 grupos acima.

Acesse
http://localhost/treinamento/app4/users/add

E cadastre os 3 usuários acima com as respectivas senhas.
É interessante editar as views src/Template/Users/index.ctp, view.ctp e edit.ctp removendo o campo senha do form.

Cadastro dos Privilégios

Os privilégios de acesso dos usuários aos actions de cada controller é você quem define. Para isso basta cadastrar no controller Privileges, antes de implementar o controle de acesso e fazer um pequneo ajuste no módulo go() se necessário.

Por padrão o controle que já vem na tabela privileges é:

Grupo	Controller	Actions
Admins	Groups	Todos
Admins	Users	Todos
Admins	Privileges	Todos

Falta Cadastrar o acesso para

Admins	Clientes	Todos
Manager	Clientes	Todos
Users	Clientes	index

Para Admins e Manager usaremos o método populate().
Faremos o cadastro do acesso de Users ao controller Clientes e ao action index pelo navegador, através do controller Privileges, antes de implementar o controle de acesso.

Criando nosso Componente Control

Usando o bake para isso:

cd /backup/www/treinamento/app5

bin/cake bake component Control

Com isso o bake gera o esqueleto do nosso
src/Controller/Component/ControlComponent.php

Vamos começar adicionando as duas linhas abaixo:
use Cake\Datasource\ConnectionManager;
use Cake\ORM\TableRegistry;

Adicione logo abaixo de:
use Cake\Controller\ComponentRegistry;

Abaixo da propriedade adicione a função populate:

```
    public function populate($group, $controller){
        $conn = ConnectionManager::get('default');

        $actions = $conn->execute("SELECT action FROM `privileges` WHERE
group_name = '$group' and controller = '$controller' and action='index'")->fetchAll();

        if(!$actions){
            $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'index')");
            $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'view')");
            $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'add')");
            $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'edit')");
            $conn->execute("insert into privileges (group_name,controller,action) values
('$group', '$controller', 'delete')");
        }
    }
```

Logo abaixo da função populate() adicione a função go():

```
    public function go($controller,$action,$groupo){
        $user = $this->request->session()->read('Auth.User');
        $username=$user['username'];
        if($username == 'admin'){
            $groupo='Admins';
        }elseif($username == 'manager'){
            $groupo='Managers';
        }elseif($username == 'user'){
            $groupo='Users';
        }

        $conn = ConnectionManager::get('default');
        $group = $conn->execute("select group_name from privileges where controller =
'$controller' and action ='$action' and group_name='$groupo'")->fetchAll();

        if($group){
            $group = $group[0][0];
        }

        if($username == 'admin' && $group=='Admins'){
            return true;
        }elseif($username == 'manager' && $group == 'Managers' && $controller ==
'Clientes'){
            return true;
        }elseif($username == 'user' && $group == 'Users' && $controller == 'Clientes' &&
($action == 'index' || $action == 'view')){
```

```
            return true;
        }else{
            return false;
        }
    }
```

Ajustando o src/Controller/AppController.php

Removi os comentários

```
<?php
namespace App\Controller;

use Cake\Controller\Controller;
use Cake\Event\Event;

class AppController extends Controller
{
     public $msg;

   public function initialize()
   {
      parent::initialize();

      $this->loadComponent('RequestHandler');
      $this->loadComponent('Flash');
          $this->loadComponent('Control');

      $this->loadComponent('Auth', [
         'loginRedirect' => [
            'controller' => 'Clientes',
            'action' => 'index'
         ],
         'logoutRedirect' => [
            'controller' => 'Users',
            'action' => 'login'
         ],
               'unauthorizedRedirect' => [
                     'controller' => 'Users',
                     'action' => 'login',
                     'prefix' => false
               ],
            'authError' => 'Faça login antes',
            'authenticate' => [
               'Form' => [
                  'fields' => ['username' => 'username']
               ]
            ],
            'storage' => 'Session'
      ]);
```

```
    }

    public function beforeFilter(\Cake\Event\Event $event) {
        parent::beforeFilter($event);

            $controller = $this->request->controller;
            $action = $this->request->action;

            //$this->Control->populate('Admins', 'Clientes');
            //$this->Control->populate('Managers', 'Clientes');

            $user = $this->request->session()->read('Auth.User');
            $username=$user['username'];

            $this->Auth->allow(['view', 'index']);//,'add'

            if(isset($username) && $username == 'admin'){
                $group='Admins';
            }elseif($username == 'manager'){
                $group='Managers';
            }elseif($username == 'user'){
                $group='Users';
            }

            $denied='Acesso Negado!';
            $privilege = 'Privilégio não Cadastrado!';
            if(isset($username) && $action != 'login' && $action != 'logout'){
                if($this->Control->go($controller,$action,$group)==false){
                    $this->Flash->error(__($denied));
                    // Voltar para onde veio, ou seja, nem sair de onde está
                    return $this->redirect($this->referer());
                    //return $this->redirect(['controller' => 'Users','action' => 'login']);
                    // $this->redirect($this->Auth->logout());
                }else{
                    if($this->msg == 'priv'){
                        $this->Flash->set(__($this->privilege));
                    }
                }
            }
      }

    public function beforeRender(Event $event)
    {
        if (!array_key_exists('_serialize', $this->viewVars) &&
            in_array($this->response->type(), ['application/json', 'application/xml'])
        ) {
            $this->set('_serialize', true);
        }
    }
}
```

Antes de começar a testar vamos efetuar os devidos registros para Admins, Managers e Users.

Após executar o aplicativo, o método populate() irá cadastrar Clientes para Admins e Managers.

Falta somente abrir o aplicativo como admin e cadastrar index em Clientes para Users.

Agora podemos testar o aplicativo com nosso componente.

http://localhost/treinamento/app5

Faça login como admin, depois logout e login como manager e logout então login como user e navegue pelos vários controllers e actions para ver como reage.

Veja que nenhum action está liberado ao público. Ao acessar o raiz do aplicativo será rdirecionado para o login. Faça o login e experimente.

O aplicativo demo já tem todos estes registros cadastrados e está pronto para ser testado. Apenas crie o banco e configure para testar.

Se tentar logar como usuário **user** terá acesso negado, pois não cadastrou ainda nenhum action para o **user**.

Acesse como admin e vá em privileges e cadastre o controller Clientes e action index para o usuário user. Faça logout e agora login como user. Agora poderá acessar o index de Clientes.

Tente acessar qualquer outro action que será barrado. Libere agora o action view de Clientes para o user e experimente.

Veja a versão melhorada deste componente no próximo aplicativo (app5a).

Aplicativo app5a

Neste aplicativo apenas melhorarei o componente Control.

Fiz uma cópia do diretório para app5a e copiei também o banco para app5a.

Após copiar diretório e banco e configurar, experimente e veja a diferença. Em termos de funcionalidades para o usuário não mudou tanto mas em termos de código eu procurei otimizar ao máximo e melhorei algumas coisas, confira.

Grupos de Usuários

Agora são 4:

Supers - poder total, acessam tudo do aplicativo
Admins -acesso total às tabelas administrativas: groups, users e permissions (agora mudou de privileges para permissions)

Managers - acessam tudo que Admins não acessam: todas as tabelas diferentes de groups, users e permissions

Users - inicialmente tem acesso somente ao login e logout. Para que possa acessar qualquer outro action precisa que seja cadastrado nos actions respectivos. Basta acessar como um Supers ou Admins e cadastrar suas permissões.

Observe que como está os registros na tabela permissions somente serão cadastrados quando for feito o primeiro login no sistema.

Após o primeiro login e execução do sistema, pode comentar as duas linhas respectivas de execução da função populate() no AppController.

Crie o aplicativo fazendo uma cópia do app5b para app5c.

Crie o banco pelo app5c.sql.

Configure o banco, pois as rotas já estão ok.

Abra o aplicativo:

http://localhost/treinamento/app5c

Descomente a linha para permitir add e index no AppController e acesse:
http://localhost/treinamento/app5c/users

Então cadastre 4 usuários, um usuário para cada grupo:

Usuário	**Senha**	**Grupo**
super	super	Supers
admin	admin	Admins

manager	manager	Managers
user	user	Users

Então comente novamente a linha que permite add e index.

Faça login no sistema com cada um dos usuários e teste o acesso ao aplicativo.

Sexto Aplicativo
Agora iremos adicionar ao aplicativo um element e um layout. O element criará um menu que facilitará a navegação através dos controllers e o layout deverá diferenciar a área administrativa. Quando o usuário **admin** fizer login ele encontrará um layout diferente dos demais usuários. Podemos criar um layout específico para cada usuário.

Adicionando um Element e um Layout

Devemos customizar o ambiente de trabalho do administrador, usando para isso um element e um layout.

Adicionaremos ao beforeFilter() do AppController:

```
$this->set('title_for_app','Meu Aplicativo');
$this->set('title_for_admin','Administração do Aplicativo');
$this->set('loggedyes','Logado como');
$this->set('loggedno','Não logado!');


$this->set('current_user', $user);
if($user == 'admin'){
    $this->layout = 'admin';
}
```

Criaremos o Elemento topmenu.ctp e o layout_admin.ctp.

17.7 – Aplicativo Finanças Pessoais

Este aplicativo tem como objetivo principal efetuar operações entre os elementos do MVC (Controller, Model e View).

Procurarei simplificar outras operações, focando apenas na codificação

Tabelas

receitas
id
descricao
valor

despesas
id
descricao
quantidade
receita_id int
valor

Relatório

saldo = receitas - despesas

Começar apenas com a tabela despesas:

```
create table receitas(
	id int primary key auto_increment,
	descricao varchar(100),
	mes char(7),-- 05/2017
	valor int,
	created date null,
	modified date null
);

create table despesas(
	id int primary key auto_increment,
	descricao varchar(100),
	valor int,
	mes char(7),
	created date null,
	modified date null,
	receita_id int not null
);
```

Valores em int para facilitar

Banco de dados - model/table - controller - template

O model vai buscar no banco de dados
O controller vai buscar no model
O template vai buscar no controller

Chamar model no Controller:

```
use Cake\ORM\TableRegistry;

        $articles = TableRegistry::get('Articles');
        $func=$articles->teste();// Redebe do Model
```

Model

```
    public function cageceTotal($mes){
      $cagece = $this->find('first', [
          'conditions'=>array('Despesa.mes'=>$mes),
          'fields'=>array('Despesa.valor')
      ]);
          if(isset($cagece['Despesa']['valor'])){
              return $cagece['Despesa']['valor'];
          }
    }
```

```
use Cake\ORM\TableRegistry;

    public function cagece(){
        $despesas = TableRegistry::get('Despesas');
      $cagece = $despesas->cageceTotal(8);
        $this->set('cagece',$cagece);
    }
```

Model/Table

Receitas

```
    public function receitasTotal(){
        $query = $this->find('all'); //fetch the record
        $res = $query->select(['total_val' =>$query->func()->sum('valor')])->first();
//perform the sum operation
        $total = $res->total_val;
        return $total;
    }
```

Controller

```
use Cake\ORM\TableRegistry;

    public function total()
    {
        $despesas = TableRegistry::get('Despesas');
        $total = $despesas->despesasTotal();
        $this->set('total',$total);
    }

    public function saldo()
    {
        $this->LoadModel('Receitas');
        $receitas = $this->Receitas->receitasTotal();

        $despesas = TableRegistry::get('Despesas');
        $total = $despesas->despesasTotal();

        $saldo = $receitas - $total;
        $this->set('saldo',$saldo);
    }
```

Template

total

```
<?php
/**
  * @var \App\View\AppView $this
  */
print $total;
?>
```

saldo

```
<?php
/**
  * @var \App\View\AppView $this
  */
print $saldo;
?>
```

Adicionar ao index.ctp:

```
    <li><?= $this->Html->link(__('Total de Despesas'), ['action' => 'total']) ?></li>
    <li><?= $this->Html->link(__('Saldo'), ['action' => 'saldo']) ?></li>
```

Versão do código anterior com o mês e forms

Models

Despesas

```
    public function despesasMes($mes){
        $connection = ConnectionManager::get('default');
        $results = $connection->execute("SELECT sum(valor) FROM despesas WHERE
mes = '$mes'")->fetchAll();
        return $results[0][0];
    }
```

Receitas

```
    public function receitaMes($mes){
        $connection = ConnectionManager::get('default');
        $results = $connection->execute("SELECT sum(valor) FROM receitas WHERE
mes = '$mes'")->fetchAll();
        return $results[0][0];
    }
```

Controllers

```
    public function despesasMes()
    {
        $mes = $this->request->data('mes');
        $despesas = TableRegistry::get('Despesas');
        $total = $despesas->despesasMes($mes);
        $this->set('total',$total);
    }

    public function saldoMes()
    {
        $mes = $this->request->data('mes');
        $this->LoadModel('Receitas');
        $receitas = $this->Receitas->receitaMes($mes);

        $despesas = TableRegistry::get('Despesas');
        $total = $despesas->despesasMes($mes);

        $saldo = $receitas - $total;
        $this->set('saldo',$saldo);
    }
```

Template

Despesas/index

```
<nav class="large-3 medium-4 columns" id="actions-sidebar">
   <ul class="side-nav">
      <li class="heading"><?= __('Actions') ?></li>
      <li><?= $this->Html->link(__('New Despesa'), ['action' => 'add']) ?></li>
```

```
        <li><?= $this->Html->link(__('List Receitas'), ['controller' => 'Receitas', 'action' =>
'index']) ?></li>
        <li><?= $this->Html->link(__('New Receita'), ['controller' => 'Receitas', 'action' =>
'add']) ?></li>
        <li><?= $this->Html->link(__('Saldo'), ['action' => 'saldo']) ?></li>
<b>Despesas</b>
<?php
   echo $this->Form->create("Despesas",array('url'=>'/despesas/despesas_mes'));
   echo $this->Form->input('mes');
   echo $this->Form->button('Submit');
   echo $this->Form->end();
?>
<b>Saldo</b>
<?php
   echo $this->Form->create("Saldo",array('url'=>'/despesas/saldo_mes'));
   echo $this->Form->input('mes');
   echo $this->Form->button('Submit');
   echo $this->Form->end();
?>
```

despesas_mes

```
<?php
print $total;
?>
```

saldo_mes

```
<?php
print $saldo;
?>
```

Os forms enviam dados para o controller e o controller devolve a view respectiva.

Para adicionar um novo código: model, controller e view:
Quero ver numa view o total das despesas

model:

```
    public function despesasTotal($mes){
      $despesas = $this->find('all', array(
          'conditions'=>array('Despesa.data'=>$mes),
          'fields'=>array('sum(Despesa.valor) as total')
      ));

          if(isset($despesas[0][0]['total'])){
              return $despesas[0][0]['total'];
          }
    }
```

controller:

```
    public function total() {
        $mes = $this->request->data['Despesa']['mes'];
        $total=$this->Despesa->despesasTotal($mes);

        if($total > 0){
            $this->Session->setFlash(__('Despesas do mês de '.$mes));
            $this->set('total', $total);
        }else{
            $this->Session->setFlash(__('Nenhuma despesa cadastrada no mês de '.
$mes));
            return $this->redirect(array('action' => 'index'));
        }
    }
```

view:

```
<div class="despesas view">
<h2><?php echo __('Total de Despesas do Mês'); ?></h2>
    <dl>
        <dt><?php echo __('Valor'); ?></dt>
        <dd>
            <?php echo h($total); ?>
             
        </dd>
    </dl>
</div>
<div class="actions">
    <h3><?php echo __('Actions'); ?></h3>
</div>
```

Quero ver na view index o total das despesas:

model:

```
    public function despesasTotal($mes){
      $despesas = $this->find('all', array(
         'conditions'=>array('Despesa.data'=>$mes),
         'fields'=>array('sum(Despesa.valor) as total')
      ));

        if(isset($despesas[0][0]['total'])){
            return $despesas[0][0]['total'];
        }
    }

    public function index() {
        $this->Despesa->recursive = 0;
        $this->set('despesas', $this->Paginator->paginate());
```

```
        // Total
        if(isset($this->request->data['Despesa']['mes'])){
                $mes = $this->request->data['Despesa']['mes'];
                $total=$this->Despesa->despesasTotal($mes);

                if($total > 0){
                    $this->set('total', $total);
                }
        }
    }
```

view:

Adicionar ao final da index.ctp, na div actions, abaixo do link Nova Despesa:

```
<div class="actions">
    <h3><?php echo __('Actions'); ?></h3>
...

    <?php echo $this->Form->create('Despesa', array('action' => 'index')); ?>
    <?php echo $this->Form->input('mes', array('label'=>'Total deste mês')); ?>
    <?php echo $this->Form->end(__('Enviar')); ?>

    <?php
    if(isset($total)){?>
    <h2><?php echo __('Total do Mês'); ?></h2>
        <dl>
            <dt><?php echo __('Valor'). ' - ' .h($total); ?></dt>
        </dl>
    <?php
    }
    ?>
</div>
```

Controllers

```
    public function total() {
        $mes = $this->request->data['Despesa']['mes'];
        $total=$this->Despesa->despesasTotal($mes);

        if($total > 0){
            $this->Session->setFlash(__('Despesas do mês de '.$mes));
            $this->set('total', $total);
        }else{
            $this->Session->setFlash(__('Nenhuma despesa cadastrada no mês de '.
$mes));
            return $this->redirect(array('action' => 'index'));
        }
    }
```

```
    public function recibo($id) {

        if (!$this->Recibo->exists($id)) {
            throw new NotFoundException(__('Invalid recibo'));
        }
        $options = array('conditions' => array('Recibo.id' => $id));
        $this->set('recibo', $this->Recibo->find('first', $options));
        $rec=$this->Recibo->find('first', $options);

        $mes = $rec['Recibo']['mes'];

        $mesquery = $this->Relatorio->query("select numero from relatorios where mes =
'$mes'");
        $nr=$mesquery[0]['relatorios']['numero'];

        if($nr=='101'){
            $this->Session->setFlash(__('Este relatório já está cadastrado!'));
            $this->redirect(array('action' => 'index'));
        }

        $cag = $this->Despesa->cageceTotal($mes);
        if($cag == 0){
            $this->Session->setFlash(__('Despesas não cadastradas para este mês!'));
            $this->redirect(array('action' => 'index'));
        }

        if($cag > 0){
            $vencimento = '10/'.$mes;
            $pessoasTotal = $this->Unidade->pessoasTotal();
            $unidadesTotal = $this->Unidade->unidadesTotal();
            $despesasCondTotal = $this->Despesa->despesasCondTotal($mes);
            $cotaCond = round(($despesasCondTotal/$unidadesTotal),2);
            $cageceTotal = $this->Despesa->cageceTotal($mes);

            // Todos os aptos
            for ($x=1;$x<13;$x++){
                if($x < 6)    $numero = 100+$x;
                if($x >= 6 && $x <10)$numero = 195+$x;
                if($x==10) $numero = '536A';
                if($x==11) $numero = '538';
                if($x==12) $numero = '540';

                $inquilino = $this->Unidade->inquilino($numero);
                $pessoasApto = $this->Unidade->pessoasApto($numero);
                $cotaAgua = round(($cageceTotal/$pessoasTotal)*$pessoasApto,2);
                $multa = $this->Extra->multa($numero,$mes);
                if(!$multa) $multa=0;
                $observacao = $this->Extra->observacao($numero,$mes);
```

```
                $total = $cotaAgua + $cotaCond + $multa;
                $rel=$this->Relatorio->query("insert into relatorios

(numero,mes,vencimento,inquilino,pessoas,cagece,condominio,multa,total,observacao)
values

('$numero','$mes','$vencimento','$inquilino','$pessoasApto','$cotaAgua','$cotaCond','$mult
a','$total','$observacao')");
            }

            $this->Session->setFlash(__('Recibos cadastrados com sucesso! Veja o
relatório!'));
            $this->redirect(array('controller'=>'relatorios','action' => 'index'));
        }else{
            $this->Session->setFlash(__('As despesas deste mês não foram cadastradas
ainda!'));
            $this->redirect(array('action' => 'index'));
        }
    }
```

Models

Codificação nos Models

```
    public function __dataBeforeSave($data){
        return implode("-",array_reverse(explode("/", $data)));
        return date('Y-m-d', strtotime($data));
    }

    public function beforeSave($options = array()) {
        if(!empty($this->data['Cliente']['aniversario'])){
            $this->data['Cliente']['aniversario'] = $this->__dataBeforeSave($this->data['Cliente']
['aniversario']);
        }
        return true;
    }

      public function beforeSave($options = array()){
            // Toda a lógica deste callback é aplicada após a validação e antes de salvar
            // Para continuar deve retornar true

            if($this->data['Despesa']['descricao']=='meire'){
                // Salvar o 'valor' será acrescido de 20% do INSS a cada add ou edit
                // Cuidado: cada a cada alteração o valor será acrescido de 20%
                if(!empty($this->data['Despesa']['valor'])){
                    $this->data['Despesa']['valor'] = $this->data['Despesa']['valor']*(1.2);
                    return true;
                }
```

```
        }
    }

    public function despesasTotal($mes){
      $despesas = $this->find('all', array(
          'conditions'=>array('Despesa.data'=>$mes),
          'fields'=>array('sum(Despesa.valor) as total')
      ));

          if(isset($despesas[0][0]['total'])){
              return $despesas[0][0]['total'];
          }
    }

    public function receitasTotal($mes){
      $receitas = $this->find('all', array(
          'conditions'=>array('Receita.data'=>$mes),
          'fields'=>array('sum(Receita.valor) as total')
      ));
          if(isset($receitas[0][0]['total'])){
              return $receitas[0][0]['total'];
          }
    }
```

Validação com expressões regulares

```
        'data' => array(
            'soalgarismos' => array(
                'rule' => array('custom', '/[0-9\/]{7,7}$/i'),
                'message' => 'Mês com apenas algarismos e a barra. Exatamente 7
caracteres'
            ),
        ),
```

Este aplicativo pode ser baixado de:

https://github.com/ribafs/cake-financas

Versão com CakePHP 3.6.1

Criar um Aplicativo com o CakePHP 3.6.1
Para gerenciamento de finanças pessoais.
Apenas com as tabelas
despesas e receitas

Importante: este código customizado funciona apenas no CakePHP 3.6 ou superior

Criando behaviors para executar o código da camada de negócios nos models

Criando components para executar o código da camada de negócios nos controllers

- Criar o aplicativo

cd /var/www/html
composer create-project --prefer-dist cakephp/app financas

- Criar o banco de dados

Usaremos como SGBD o MySQL

Criar o banco financas contendo:

```
CREATE TABLE `despesas` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `descricao` varchar(100) COLLATE utf8_unicode_ci DEFAULT NULL,
  `valor` int(11) DEFAULT NULL,
  `mes` char(8) COLLATE utf8_unicode_ci DEFAULT NULL,
  `created` date DEFAULT NULL,
  `modified` date DEFAULT NULL,
  `receita_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO `despesas` (`id`, `descricao`, `valor`, `mes`, `created`, `modified`,
`receita_id`) VALUES
(1, 'Mercantil', 500, '05/2017', '2017-05-11', '2017-05-15', 1),
(2, 'Condomínio', 100, '05/2017', '2017-05-11', '2017-05-15', 1),
(3, 'Teste', 300, '06/2017', '2017-05-11', '2017-05-11', 2);

CREATE TABLE `receitas` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `descricao` varchar(100) COLLATE utf8_unicode_ci DEFAULT NULL,
  `mes` char(7) COLLATE utf8_unicode_ci DEFAULT NULL,
  `valor` int(11) DEFAULT NULL,
  `created` date DEFAULT NULL,
  `modified` date DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO `receitas` (`id`, `descricao`, `mes`, `valor`, `created`, `modified`) VALUES
(1, 'Salário', '05/2017', 3000, '2017-05-11', '2017-05-15'),
(2, 'Extra', '05/2017', 100, '2017-05-11', '2017-05-15');
```

Observe acima que o mês é tipo texto para que seja inserido algo com mes/ano

- Configurar o banco de dados em

cd /var/www/html/financas
config/app.php

e o
config/routes.php para Despesas/index
- Gerar o código dos dois CRUDs com o Bake

cd /var/www/html/financas

bin/cake bake all despesas

bin/cake bake all receitas

Veja pelo navegador

http://localhost/financas

Já temos um CRUD básico para as despesas e para as receitas

Mas o importante de um aplicativo deste tipo são o código personalidado, chamado de lógica de negócio, que é por exemplo somar todas as despesas de um mês e subtrair as receitas do mês do total de despesas. Isso o CakePHP não faz e precisamos fazer. Também precisamos criar código nos tempaltes adicionando um formulário

- Criar os dois Behaviors: DespesasBehavior.php e ReceitasBehavior.php
- Criar os dois Components: DespesasComponent.php e ReceitasComponent.php

Agora a parte interessante

O código do negócio

Observe que o mês tem um formato próprio, texto sendo dia/ano (dd/aaaa).

Vamos criar uma função em DespesasTable.php, que retorne a soma das despesas do mês:

Adicionar ao início:

use Cake\Datasource\ConnectionManager;

Criar

```
public function despesasMes($mes){
    $connection = ConnectionManager::get('default');
    $results = $connection->execute("SELECT sum(valor) FROM despesas WHERE mes =
'$mes'")->fetchAll();
    return $results[0][0];
}
```

Criar uma função no DespesasController.php, que retorne o resultado da função acima:

Adicionar ao DespesasController

```
use Cake\ORM\TableRegistry;

public function despesasMes()
{
   $mes = $this->request->getData('mes');
   $despesas = TableRegistry::get('Despesas');
   $despesa = $despesas->despesasMes($mes);
   $this->set('despesa',$despesa);
}
```

Agora criaremos uma função no ReceitasTable.php, que retorne a soma das receitas do mês:

Adicionar ao início:

```
use Cake\Datasource\ConnectionManager;
```

Criar

```
public function receitaMes($mes){
   $connection = ConnectionManager::get('default');
   $results = $connection->execute("SELECT sum(valor) FROM receitas WHERE mes =
'$mes'")->fetchAll();
   return $results[0][0];
}
```

Veja que existe grande semelhança entre esta e a das despesas.

Vamos trabalhar com dois models com o mesmo controller.

Para isso vamos usar a função LoadModel(). Criaremos outra função no DespesasController.php

Vamos criar uma função que retorne o saldo do mês entre receitas e despesas

```
public function saldoMes()
{
      $mes = $this->request->getData('mes');
      $this->LoadModel('Receitas');

      $receitas = $this->Receitas->receitaMes($mes);
      $despesas = TableRegistry::get('Despesas');
```

```
        $despesas = $despesas->despesasMes($mes);
        $saldo = $receitas - $despesas;

        $this->set('saldo',$saldo);
}
```

Vamos customizar a src/Template/Despesas e criar duas views novas:

src/Template/Despesas/despesas_mes.ctp:

```
<div align="center"><h3><?= $despesa ?></h3></div>
<br>
<br>
<br>
<li><?= $this->Html->link(__('Voltar'), ['action' => 'index']) ?></li>
```

src/Template/Despesas/saldo_mes.ctp:

```
<div align="center"><h3><?= $saldo ?></h3></div>
<br>
<br>
<li><?= $this->Html->link(__('Voltar'), ['action' => 'index']) ?></li>
```

Agora vejamos a customização do index.ctp

Logo abaixo da linha:

```
<li><?= $this->Html->link(__('New Receita'), ['controller' => 'Receitas', 'action' => 'add']) ?
></li>
```

Adicione os dois pequenos forms:

```
<b>Despesas de um mês</b>
<?php
   echo $this->Form->create("Despesas",array('url'=>'/despesas/despesas_mes'));
   echo $this->Form->input('mes');
   echo $this->Form->button('Submit');
   echo $this->Form->end();
?>
<b>Saldo de um mês</b>
<?php
   echo $this->Form->create("Saldo",array('url'=>'/despesas/saldo_mes'));
   echo $this->Form->input('mes');
   echo $this->Form->button('Submit');
   echo $this->Form->end();
?>
```

Depois de pronto podemos efetuar as operações através dos pequenos forms.
- O form chama o action do controller
- O controller chama o model para que devolva o resultado de uma função
- Recebendo o resultado o controller devolve para a view/template respectiva

Testando

Agora acesse pela web

http://localhost/financas

Pesquise o total das despesas do mês de 05/2017 ou outro que você tenha cadastrado.

Pesquise qual o saldo de um mês

Isso dá para começar a mexer no código e ir em frente.

Exemplo Simplificado do Fluxo de Informações no CakePHP

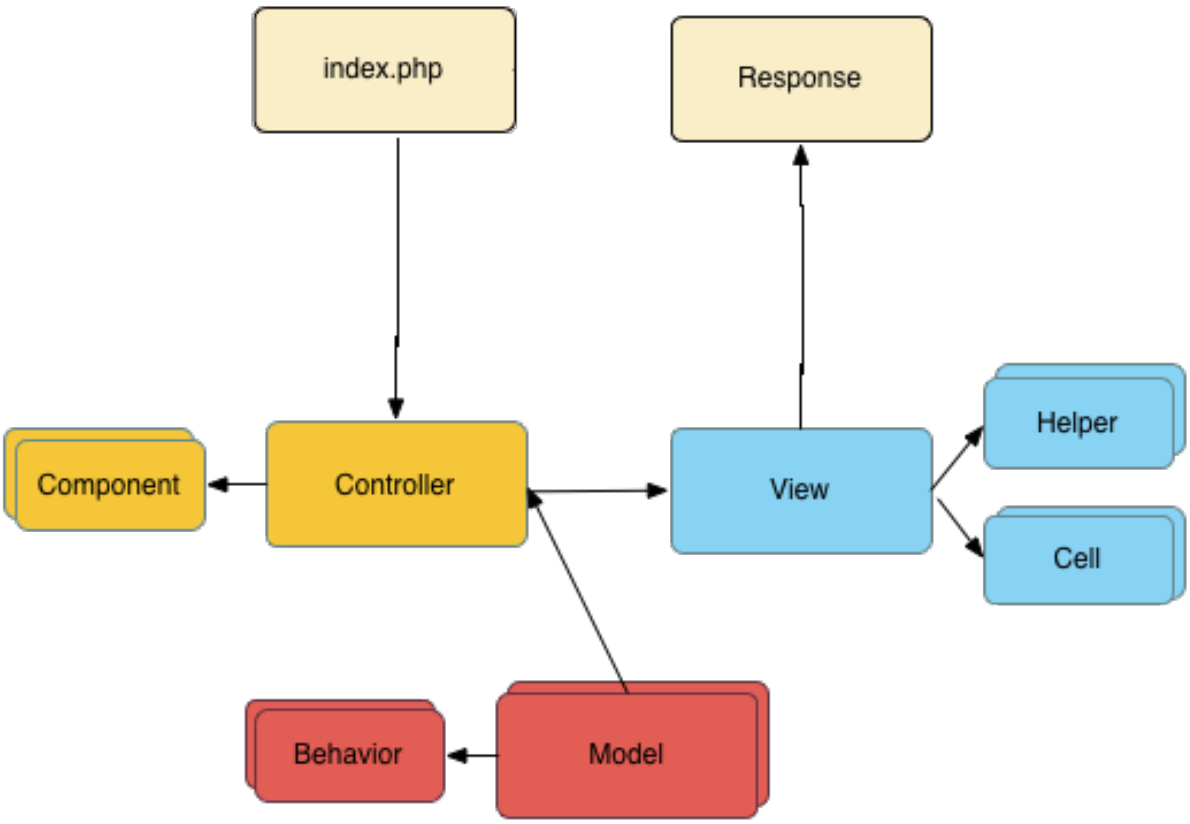
A foram mais simples seria a requisição de index.ctp pelo usuário, que receberia a listagem dos registros

- Usuário abre a URL do aplicativo que por padrão chama o método/action index de um Controller
- O controller recebe a solicitação e depois de analisar, se for o caso, ele manda para a respectiva view
- O usuário recebe de volta o resultado na listagem da view

- (Usuário)http://localhost/crud -> src/ClientesController.php -> src/Template/Clientes/index.ctp(Usuário)

- Mas existem diversas variações deste fluxo:
    - O controller pode solicitar ajuda de um componente quando for o caso
    - O controller poderá solicitar informações do respectivo model, que devolverá ou não as informações para o controller
    - O model poderá solicitar informações de um behavior quando for o caso para então devolver ao controller
    - O controller depois de receber do model enviaria para a a respectiva view e a view poderá solicitar ajuda de um helper, de uma cell, de um element, etc
    - A view então envia para o navegador quando o usuário terá acesso

Detalhes - https://book.cakephp.org/3.0/pt/intro.html



17.8 - Fluxo de Informações entre controllers, models e view/templates

Temos o aplicativo clientes

UsersController.php
UsersTable.php
Template/Users/index.ctp

Criar em UsersTable.php a função:

```
    public function teste(){
        $query = $this->find('all', [
            'order' => ['Users.id' => 'ASC']
        ]);
        $row = $query->first(); // Ou ->last()
        print "Model<br>";
        return $row->username;
    }
```

Chamar no index() do UsersController.php:

```
  public function index()
  {
     $clientes = $this->paginate($this->Users);

     $this->set(compact('users'));
     $this->set('_serialize', ['users']);

        print "Controller<br>";

        // Mostrar o primeiro username:
        print $this->Users->teste();//exit;

  }
```

Chamar pela web

http://localhost/clientes/users/index

Mostrará

Controller
Model
admin

Listagem de Users

View

Veja a ordem:
1) O controller recebe a requisição do usuário para mostrar o endereço:
http://localhost/clientes/users/index

2) O Controller envia para o Model pedindo o primeiro username e a listagem de users

3) O Model processa e devolve

4) Então o controller envia para a view o username e a listagem de users

Passando variáveis pela URL para uma view

Na URL:
localhost/cake-control-demo/pages/home?lang=en&temp=default

No AppController

```
    public function initialize()
    {
        parent::initialize();

        $this->loadComponent('RequestHandler');
        $this->loadComponent('Flash');

            $lang=$this->request->query('lang');
            $temp=$this->request->query('temp');

            $this->set('lang',$lang);
            $this->set('template',$temp);
    }
```

Na View :

```
    <div id="content">
        <div class="row">
            <div class="columns large-12 ctp-warning checks">
                <?php print "<h1>Idioma: ".$lang." Template: ".$template."</h1>"; ?>
```

17.9 – CRUD em 4 Partes

Criação de um aplicativo tipo CRUD com o CakePHP3

Vamos começar usando o recurso mais prático do CakePHP, que é o gerador de CRUDs chamado bake.
Iremos criar um simples CRUD mas já é um recurso muito prático e que pode ser utilizado para a criação de aplicativos para nosso dia a dia.
Depois iremos abordar recursos mais avançados.

Estou usando o Linux com diretório web em
/var/www/html

Caso esteja usando outro ambiente ou em outro diretório web altera o caminho abaixo

- Criar o aplicativo base usando o composer na pasta crud
```
cd /var/www/html
composer create-project --prefer-dist cakephp/app crud
```

- Criar o banco de dados no MySQL chamado 'crud' com apenas uma tabela chamada 'clientes'

Sugestão de gerenciador de bancos de dados: o gerenciador web em PHP
http://adminer.org

```
CREATE TABLE IF NOT EXISTS `clientes` (
  `id` int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `nome` char(45) NOT NULL,
  `email` varchar(50) DEFAULT NULL
);
```

- Configurar o banco no aplicativo
```
cd /var/www/html/crud
nano config/app.php
```
Como usaremos o mysql basta alterar estar linhas em Datasource
```
        'username' => 'root',
        'password' => '',
        'database' => 'crud',
```

- Acessar o aplicativo pelo navegador
```
http://localhost/crud
```

Ele nos mostra uma página padrão criada automaticamente pelo Cake, mas sem nada do nosso banco

- Configurar o routes.php para Clientes com index
```
nano config/routes.php
```

Mudar apenas esta linha
cd /var/www/html/crud
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
Para
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);

- Gerar o CRUD com o bake

Execute

cd /var/www/html/crud
bin/cake bake all clientes

Observe que ele mostra mensagens de que criou todo o nosso código: controller, model, view/template entre outros

Chame pelo navegador

http://localhost/crud

Ele nos mostra um aplicativo completo, com todo o CRUD, com CSS, paginação, ordenação dos campos apensa clicando em seus labels, etc.

Clique em New Criante e crie um.

Entre um nome e entre um e-mail incompleto, apenas o login e clique em SUBMIT.
Ele já traz validação para o e-mail. Ele verificou na tabela o nome do campo "email" e já aplicou validação.

Uma beleza.

Corrija e clique em SUBMIT

Como a equipe do Cake está se preparando para lançar a versão 4.0, talvez receba a mensagem de erro:

Deprecated (16384): RequestHandlerComponent::beforeRedirect() is deprecated. This functionality will be removed in 4.0.0. Set the `enableBeforeRedirect` option to `false` to disable this warning. - /backup/www/crud/vendor/cakephp/cakephp/src/Event/EventManager.php, line: 353
You can disable deprecation warnings by setting `Error.errorLevel` to `E_ALL & ~E_USER_DEPRECATED` in your config/app.php. [CORE/src/Core/functions.php, line 307]

Caso receba etirre novamente o arquivo

config/app.php

E altere a linha

'errorLevel' => E_ALL,

Para
'errorLevel' => E_ALL & ~E_USER_DEPRECATED,

E chame novamente

http://localhost/crud

Veja que temos um CRUD funcional (Add, Edit, View e Delete, além de outrosr ecursos), que podemos usar para o cadastro de clientes, uma agenda e qualquer outra finalidade.

Sugestões de leitura:

Uma recomendação é instalar o plugin cake-control-br, que instala um template com o Bootstrap e um controle de acesso para o aplicativo:

https://github.com/ribafs/cake-control-br

O site oficial do CakePHP é uma das grandes forças do framework, com tutoriais de exemplo de aplicativos:
https://book.cakephp.org/3.0/en/tutorials-and-examples.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog/blog.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog-auth-example/auth.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/bookmarks/intro.html

E uma ótima documentação, que pode ser vista online
https://book.cakephp.org/3.0/en/index.html

Ou offline em PDF ou Epub
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.pdf
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.epub

Uma forma bem simples de se criar um bom aplicativo tipo CRUD e muito mais

Criação de um aplicativo tipo CRUD com o CakePHP3

O objetivo maior deste tutorial é criar um aplicativo tipo com o Cake, mas agora apenas codificando, sem a ajuda do bake.

Vamos seguir as dicas do CakePHP para ir criando nosso aplicativo.
Este aplicativo será apenas para mostrar como funcionam alguams coisas no CakePHP. Ele não faz nada de útil mas conhecer estas coisas no Cake é muito importante. Quando mais você dominar o funcionamento mais poderá fazer com a ferramenta.

- Criar o aplicativo base usando o composer na pasta crud0
  cd /var/www/html
  composer create-project --prefer-dist cakephp/app crud0

- Vamos usar o mesmo banco 'crud' criado no tutorial anterior com a tabela clientes

- Configurar o banco no aplicativo
  cd /var/www/html/crud0
  nano config/app.php
Como usaremos o mysql basta alterar estar linhas em Datasource
        'username' => 'root',
        'password' => '',
        'database' => 'crud',

- Acessar o aplicativo pelo navegador
  http://localhost/crud0

Ele nos mostra uma página padrão criada automaticamente pelo Cake com diversas informações úteis, mas sem nada do nosso banco

- Vamos tentar acessar nossa tabela de clientes
  http://localhost/crud0/clientes

Veja os erros:
Error: ClientesController could not be found.

Error: Create the class ClientesController below in file:
src/Controller/ClientesController.php

```
<?php
namespace App\Controller;

use App\Controller\AppController;

class ClientesController extends AppController
{

}
```

Ele diz que precisamos ter um controller chamado Clientes e nos indica uma estrutura básica. Então vamos seguir suas dicas

cd /var/www/html/crud0

Lembre que a pasta src guarda todo o código do aplicativo em CakePHP. Todo o código que adicionamos diretamente para o aplicativo deve ficar nesta pasta.
Crie o arquivo:

nano src/Controller/ClientesController.php
Contendo

```
<?php
namespace App\Controller;
```

```
use App\Controller\AppController;

class ClientesController extends AppController
{

}
```

Chamemos agora novamente pelo navegador

http://localhost/crud0/clientes

Veja os erros agora

The action index is not defined in ClientesController

Error: Create ClientesController::index() in file: src/Controller/ClientesController.php.

```
<?php
namespace App\Controller;

use App\Controller\AppController;

class ClientesController extends AppController
{

    public function index()
    {

    }
}
```

Ele está dizendo que precisamos criar o action index() no nosso controller, então vamos criar como indicado dentro da classe

```
    public function index()
    {

    }
```

Chamemos novamente pelo navegador

http://localhost/crud0/clientes

E agora os erros são diferentes:

Error: The view for ClientesController::index() was not found.

Confirm you have created the file: "Clientes/index.ctp" in one of the following paths:

/backup/www/crud/src/Template/Clientes/index.ctp

Ele nos pede uma view chamada index.ctp em src/Template/Clientes

```
mkdir /var/www/html/crud/src/Template/Clientes
nano /var/www/html/crud/src/Template/Clientes/index.ctp
```

Criando apenas o arquivo index.ctp vazio já atende e não mais veremos erro.
Mas a tela abrirá limpa, somente com o título Clientes

Adicione para o index.ctp:

```
<h1>Meu primeiro aplicativo</h1>
```

Chamemos novamente pelo navegador

```
http://localhost/crud0/clientes
```

Porque ao chamar clientes ele abre o action index() que chama o index.ctp correspondente?
É o padrão do Cake, ao chamar um controller, ele chama por padrão o action index().
E ao chamar um action ele chama o template correspondente, no caso o index.ctp.

Agora nada de erro, apenas a mensagem que adicionamos.

Veja toda a sequência:
- Criar o controller
- Criar no controller o método/action ClientesController/index()
- Criar uma view/template correspondente ao método do controller, no caso o Tempalte/Clientes/index.ctp

Veja como o Cake ajuda na criação. Em algumas etapas ele diz exatamente o que precisamos fazer.

- Configurar o routes.php para que nosso aplicativo acesse automaticamente o action index() do controller Clientes

```
cd /var/www/html/crud0
nano config/routes.php
```

Mudar apenas esta linha

```
$routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Para

```
$routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Fazendo isso podemos chamar nosso aplicativo assim:

```
http://localhost/crud0
```

E ele automaticamente chamará

```
http://localhost/crud0/clientes/index
```

Pois definimos o controller Clientes e o action index como defaults.

O que vem a seguir?

Veja que nosso aplicativo crud0 não é funcional. Seu código não é usável. Não trabalha com o banco de dados.
Criaremos o aplicativo crud1 e então trabalharemos com o banco de dados.

Leituras recomendadas:

https://github.com/ribafs/cake-control-br

O site oficial do CakePHP é uma das grandes forças do framework, com tutoriais de exemplo de aplicativos:
https://book.cakephp.org/3.0/en/tutorials-and-examples.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog/blog.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/blog-auth-example/auth.html
https://book.cakephp.org/3.0/en/tutorials-and-examples/bookmarks/intro.html

E uma ótima documentação, que pode ser vista online
https://book.cakephp.org/3.0/en/index.html

Ou offline em PDF ou Epub
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.pdf
https://book.cakephp.org/3.0/_downloads/en/CakePHPCookbook.epub

Uma forma bem simples de se criar um bom aplicativo tipo CRUD e muito mais

Criação de um aplicativo tipo CRUD com o CakePHP3

O objetivo maior deste tutorial é criar um aplicativo tipo CRUD com o Cake, mas agora apenas codificando, sem a ajuda do bake.
Vamos seguir de onde paramos no tutorial anterior sobre o crud0.

- Criar o aplicativo base usando o composer na pasta crud1
```
cd /var/www/html
composer create-project --prefer-dist cakephp/app crud1
```

- Vamos usar o mesmo banco 'crud' criado no tutorial anterior com a tabela clientes

- Configurar o banco no aplicativo
```
cd /var/www/html/crud1
nano config/app.php
```
Como usaremos o mysql basta alterar estar linhas em Datasource
```
'username' => 'root',
'password' => '',
'database' => 'crud',
```

Aproveite e altere a linha com

```
        'errorLevel' => E_ALL,
```
Para
```
        'errorLevel' => E_ALL & ~E_USER_DEPRECATED,
```

Vamos inserir alguns registros na tabela clients para ficar mais prático:

```
insert into clientes (nome, email) values
('Ribamar FS', 'ribafs@gmail.com'),
('João Britu Cunha', 'joao@gmail.com'),
('Pedro Álvares', 'pedro@gmail.com');
```

- Configurar o routes.php para que nosso aplicativo acesse automaticamente o action index() do controller Clientes

```
    cd /var/www/html/crud1
    nano config/routes.php
```

Mudar apenas esta linha
```
    $routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```
Para
```
    $routes->connect('/', ['controller' => 'Clientes', 'action' => 'index']);
```

Fazendo isso podemos chamar nosso aplicativo assim:

```
    http://localhost/crud1
```

E ele chamará
http://localhost/crud1/clientes/index

Nos mostrará erros dizendo que não temos o controller clientes.

Criação do controller ClientesController.php com o action index()

O CakePHP é um framework que segue o padrão MVC. Quando criamos um aplicativo sem criar explicitamente o model ele cria um model para nós.
Vejamos. Vamos criar o controller

src/Controller/ClientesController.php

Contendo:

```
<?php
namespace App\Controller;
use App\Controller\AppController;

class ClientesController extends AppController
{
    public function index()
    {
        $clientes = $this->Clientes;
        $this->set(compact('clientes'));
```

```
    }

}
```

Veja que é o método index() criado pelo bake, mas sem os comentários para simplificar.

Este método usa a função set() para enviar a variável clientes, que contém todos os registros da tabela clientes para a view chamada pelo método index, que no caso é src/Template/Clientes/index.ctp

Criação da View/Template Clientes/index.ctp

Então criemos o diretório

src/Template/Clientes

E dentro dele o template

index.ctp

Contendo

```
<?php
foreach ($clientes as $cliente){
    echo $cliente->id.'-'.$cliente->nome.'-'.$cliente->email.'<br>';
}
```

Melhorando um pouco a aparência do nosso template index.ctp:

```
<table>
<?php
print '<tr><td>ID</td><td>Nome</td><td>E-mail</td></tr>';
foreach ($clientes as $cliente){
    print '<tr><td>'.$cliente->id.'</td><td>'.$cliente->nome.'</td><td>'.$cliente-
>email.'</td></tr>';
}
```

Aqui recebemos a variável $clientes e varremos a mesma com o laço foreach().

Ele mostrará todos os campos de todos os registros.

Veja que fizemos tudo isso apenas com o controller e o template/view, não tocmaos em model. O cake cuidou disso para nós.

Criação do action add() e do tempalte add.ctp

Adicionar o método/action add() ao nosso controller clientes

```
    public function add()
    {
        $cliente = $this->Clientes->newEntity();

        // o método patchEntity(), como o newEntity() validará os dados antes de ser salvo na
entidade
        $cliente = $this->Clientes->patchEntity($cliente, $this->request->getData());

        // Salvar o registro cliente
        $this->Clientes->save($cliente);

        // Armazena os dados do registro clientes na variável cliente e envia para o add.ctp
        $this->set(compact('cliente'));

        // Redireciona para a index
        return $this->redirect(['action' => 'index']);
    }
```

É o método criado pelo bake mas simplificado.

Se chamar pelo navegador agora com:

http://localhost/crud1/clientes/add

Ele reclamará do tempalte add.ctp correspondente

Criar o template src/Template/Clientes/add.ctp

Contendo este código abaixo, que é uma simplificação do criado pelo bake

```
<?= $this->Form->create($cliente) ?>
<legend><?= __('Add Cliente') ?></legend>
    <?php
        echo $this->Form->control('nome');
        echo $this->Form->control('email');
    ?>
<?= $this->Form->button(__('Submit')) ?>
<?= $this->Form->end() ?>
```

Criando o action a o template view

Adicionar ao controller o método abaixo

```
    public function view($id = null)
```

```
    {
        $cliente = $this->Clientes->get($id);
        $this->set(compact('cliente'));
    }
```

Agora chame pelo navegador com

http://localhost/crud1/clientes/view/3

Ele informará que o view.ctp não existe

Então crie o arquivo src/Template/Clientes/view.ctp

Contendo

```
<table class="vertical-table">
    <tr>
        <th>Nome</th>
        <td><?= $cliente->nome ?></td>
    </tr>
    <tr>
        <th>E-mail</th>
        <td><?= $cliente->email ?></td>
    </tr>
    <tr>
        <th>Id</th>
        <td><?= $cliente->id ?></td>
    </tr>
</table>
```

Criar o action e o template edit

```
    public function edit($id = null)
    {
        $cliente = $this->Clientes->get($id);
        if ($this->request->is(['patch', 'post', 'put'])) {
            $cliente = $this->Clientes->patchEntity($cliente, $this->request->getData());
            if ($this->Clientes->save($cliente)) {
                return $this->redirect(['action' => 'index']);
            }
        }
        $this->set(compact('cliente'));
    }
```

```
<?= $this->Form->create($cliente) ?>
    <h3>Editar Cliente<>
    <?php
        echo $this->Form->control('nome');
        echo $this->Form->control('email');
```

```
    ?>
<?= $this->Form->button(__('Submit')) ?>
<?= $this->Form->end() ?>
```

Agora chame pelo navegador com:

http://localhost/crud1/clientes/edit/3

Ele mostrará um formulário com o registro 3 para ser editado.

Agora o interessante é a criação de uma grid contendo a listagem de todos os registros (index), um botão para adicioanr novos registros e cada registro contendo um link para editar, visualizar e excluir, como fez o Bake.

Mas ainda falta muito para ficar parecido com o código gerado pelo bake:
- Paginação
- Ordenação dos campos
- Estilo com CSS
- ...

O que vem a seguir?

Veja que nosso aplicativo crud1 não está completo.
Criaremos o aplicativo crud2 que mostrará links para editar, visualizar e excluir os registros no index.ctp.

Uma forma bem simples de se criar um bom aplicativo tipo CRUD e muito mais

Criação de um aplicativo tipo CRUD com o CakePHP3 mas na unha

O objetivo maior deste tutorial é criar um aplicativo tipo CRUD com o Cake, mas agora apenas codificando, sem a ajuda do bake.
Vamos seguir de onde paramos no tutorial anterior sobre o crud1.

Desta vez não criaremos outro aplicativo iremos apenas copiar a pasta do crud1 apra crud2 aproveitando tudo que fizemos

Criar os links para editar, ver e excluir no index.ctp

Chame pelo navegador com:

http://localhost/crud2

Veja que ele lista todos os registros, com respectivos id, nome e email.
Vamos adicionar uma coluna/label chamada Ações após o email e ela conterá em cada registro os três links: editar, ver e excluir, como faz o Bake.

Vamos para isso editar o src/Template/Clientes/index.ctp

Atualmente ele está assim:

```
<table>
<?php
print '<tr><td>ID</td><td>Nome</td><td>E-mail</td></tr>';
foreach ($clientes as $cliente){
    print '<tr><td>'.$cliente->id.'</td><td>'.$cliente->nome.'</td><td>'.$cliente-
>email.'</td></tr>';
}
```

Vamos alterar (aproveitando o código gerado pelo Bake no app crud):

```
<table>
<?php
print '<tr><td>ID</td><td>Nome</td><td>E-mail</td><td>Ações</td></tr>';
foreach ($clientes as $cliente){
   print '<tr><td>'.$cliente->id.'</td><td>'.$cliente->nome.'</td><td>'.$cliente->email.'</td>
   <td class="actions">
      <?= $this->Html->link(__('View'), ['action' => 'view', $cliente->id]) ?>
      <?= $this->Html->link(__('Edit'), ['action' => 'edit', $cliente->id]) ?>
      <?= $this->Form->postLink(__('Delete'), ['action' => 'delete', $cliente->id] ?>
   </td>
</tr>';
}
?>
</table>
```

No crud1 faltou a criação do método delete() para o controller Clientes

Adicione ao controller ClientesController.php

```
   public function delete($id = null)
   {
      $this->request->allowMethod(['post', 'delete']);
      $cliente = $this->Clientes->get($id);
      $this->Clientes->delete($cliente);
      return $this->redirect(['action' => 'index']);
   }
```

Veja que mais uma vez eu simplifiquei o código, removendo validações e mensagens de erro ou acerto apenas para facilitar o entendimento.
Mas o ideal é fazer como faz o Bake, com tudo que tem direito para tornar o código mais amigável e mais seguro.

Agora podemos chamar novamente pelo navegador e testar todas as operações do crud:

http://localhost/crud2

Ops, ainda falta a opção na index para criar um novo registro, ou seja, um link para o add.ctp

Vamos adicioná-lo acima da listagem, logo abaixo da tag <table>.

<?= $this->Html->link(__('New Cliente'), ['action' => 'add']) ?>

Assim nosso CRUD está completo. Todo criado linha a linha do código.

O que vem a seguir?

Veja que nosso aplicativo crud2 não está completo mas já está funcional, inserindo, editando, visutalizando e excluindo registros.

Falta muita coisa para que fique decente. Mas nesta linha pararemos por aqui.
Daqui pra frente estudaremos as partes mais avançadas do código do Cake:
- Como validar informações usando o model
- Como programar nas três camadas, model, crontroller e view e fazer as informações conversagem entre si, mostrando todo o fluxo, onde ela começa e onde ela termina. Desde a solicitação do usuário ao clicar em um link, em um botão ou chamar um link até a resposta final da informação voltando para o usuário. Com isso precisaremos usar funções que são próprias do model, outras que são próprias do controller e outras que são próprias da view.

Existe muita coisa para aprendermos.

Onde obter ajuda?

O site do Cake é a melhor fonte de consulta que conheço. Com toda a documentação fornecida online. Também é oferecida em forma de pdf e epub.

Mas não existe documentação perfeita. Seria praticamente impossível chegar perto. Devemos contar com outras fontes de pesquisa que ajudam:

Lista oficial em inglês, onde podemos tirar dúvidas - https://discourse.cakephp.org/

Forum sobre CakePHP em inglês no StackOverflow - https://stackoverflow.com/tags/cakephp/new

Forum do StackOverflow em pt-br sobre o CakePHP 3 - https://pt.stackoverflow.com/questions/tagged/cakephp-3

Claro que também temos os instrumentos de busca caso falte algo.

17.10 – Aplicativo Controle de Estoque

É um controle de estoque bem simplificado, apenas para mostrar como se usa código customizado no CakePHP.

Aplicativo tipo Controle de Estoque

Usando o Framework CakePHP 3.6.1
Com o plugin cake-acl-br para implementar ACL

Este aplicativos será criado em 3 versões e usará um script de banco de dados do CEP do Ceará, meio desatualizado:

Todos iniciando com a versão 3.5.13 e migrando para a atual

- Usando o banco com MySQL
- Usando o banco com PostgreSQL com esquema public
- Usando o banco com PostgreSQL com esquema sc_estoque e usuário us_estoque

Validar CEP, CPF e CNPJ com a validação do Cake

Gerar diagrama do Banco com o dbVisualzier

== Aplicativo com o MySQL

Criar o banco e importar o script estoque_my_acl_cepce.sql

Por conta das mudanças no CakePHP 3.6::

- Devemos primeiro instalar o CakePHP 3.5.13

https://github.com/cakephp/cakephp/archive/3.5.13.zip

Descompactar
Renomear a pasta para estoque_my

Obs.: isso somente por enquanto o cake-acl-br está incompatível com o CakePHP 3.6. Alias, não estou conseguindo instalar nenhum plugin nesta versão, inclusive os da equipe como o migrations.

- Instalar e carregar o plugin cake-acl-br

cd /var/www/html/estoque_my

composer require ribafs/cake-acl-br
bin/cake plugin load CakeAclBr --bootstrap

- Gerar todo o código com o Bake

bin/cake bake all groups -t CakeAclBr
Repetir para todas as demais

- Migrar para a versão 3.6.1 com o plugin migrations

composer require --update-with-dependencies "cakephp/cakephp:3.6.*"

Atualizar o debug_kit

composer require cakephp/debug_kit

Acessar

http://localhost/estoque_my

Algo interssante a fazer é organizar manualmente a ordem dos itens do menu:

Na primeira linha: users, groups, permissions e ceps
Na segunda: empresas, clientes, funcionarios, produtos, armazens
Na terceira: bonus, comissoes, compras, comprasitens, pedido, pedidoitens, estoque

Algo fortuito é que o menu como está está em ordem alfabética. Então vou deixar como está.

Ao efetuar o logout apareceu

Deprecated (16384): RequestHandlerComponent::beforeRedirect() is deprecated. This functionality will be removed in 4.0.0. Set the `enableBeforeRedirect` option to `false` to disable this warning. - /backup/www/estoque_my/vendor/cakephp/cakephp/src/Event/EventManager.php, line: 353
You can disable deprecation warnings by setting `Error.errorLevel` to `E_ALL & ~E_USER_DEPRECATED` in your config/app.php. [CORE/src/Core/functions.php, line 307]

Então editei
vendor/cakephp/cakephp/src/Event/EventManager.php

E na linha 353 não encontrei enableBeforeRedirect

Então fiz uma busca recursiva pela string 'enableBeforeRedirect' no diretório vendor/cakephp/cakephp/src usando a ferramenta

E encontrei no arquivo
vendor/cakephp/cakephp/src/Controller/Component/RequestHandlerComponent.php na linha 83

'enableBeforeRedirect' => true

Então mudei para

'enableBeforeRedirect' => false

Como sugere a ótima mensagem de erro do Cake

Acessei novamente pelo navegador e a mensagem desapareceu

Agora estamos usando a versão 3.6.1 do CakePHP.

Como consultar a versão do CakePHP 3?

Tá no arquivo
vendor/cakephp/cakephp/VERSION.txt

Aplicativo com o PostgreSQL no esquema default public

Aplicativo com o PostgreSQL no esquema sc_estoque e usuário us_estoque

Agora façamos alguns ajustes para melhorar nosso aplicativo:

1) Aterar a largura de campos nos forms.
Para isso usaremos as classes criadas no plugin cake-acl-br para o framework Bootstrap, que estão no arquivo webroot/css/custom.css

Implementar a busca nas Permissões

- Editar o arquivo src/Controller/PermissionsController.php

- Remover o action index() ativo
- Mover use Cake\Datasource\ConnectionManager; para baixo da linha use App\Controller\AppController;
- Remover o final do comentário que está abaixo do index()
- Ajustar o index() que estava comentado para que fique assim, para que possamos pesquisar pelo group e pelo controller:

```
public function index()
{
        $conn = ConnectionManager::get('default');
        $driver = $conn->config()['driver']; // Outros: database, etc.

        if($driver == 'Cake\Database\Driver\Postgres'){
            $this->paginate = [
                'contain' => ['Users'],
```

```
                'conditions' => ['or' => [
                    'Permissions.group ilike' => '%' . $this->request->query('search') . '%',
                    'Permissions.controller ilike' => '%' . $this->request->query('search') . '%'
                ]],
                'order' => ['Permissions.id' => 'DESC' ]
            ];
        }elseif($driver=='Cake\Database\Driver\Mysql'){
            $this->paginate = [
                'contain' => ['Users'],
                'conditions' => ['or' => [
                    'Permissions.group like' => '%' . $this->request->query('search') . '%',
                    'Permissions.controller like' => '%' . $this->request->query('search') . '%'
                ]],
                'order' => ['Permissions.id' => 'DESC' ]
            ];
        }else{
              print '<h2>Driver database dont supported!';
              exit;
        }

          $this->set('permissions', $this->paginate($this->Permissions));
        $this->set('_serialize', ['permissions']);
    }
```

- Agora ajustemos i src/Template/Permissions/index.ctp. Veja que o trecho de código da busca está comentado no início. Descomente e deixe assim:

```
    <?php
        echo $this->Form->create(null, ['type' => 'get']);
        echo $this->Form->label('Busca');
        echo $this->Form->input('search', ['class' => 'form-control', 'label' => false,
'style'=>'width:170px;',
          'placeholder' => 'Entre um group ou controller', 'value' => $this->request-
>query('search')]);
        echo $this->Form->button('Busca');
        echo $this->Form->end();
    ?>
```

- Agora já pode acessar novamente Permissions e efetuar uma busca para testar

- Vamos trocar os tipos de campos controller e action do form src/Template/Permissions/add.ctp de text para select, já trazendo os respectivos

Action

Comente a linha do campo action e adicione esta abaixo:

```
$options = ['index'=>'index','add'=>'add','edit'=>'edit','view'=>'view','delete'=>'delete'];
echo $this->Form->input('action', ['options'=>$options,'required'=>'false', 'class'=>'col-md-
12', 'empty'=>'Selecione']);
```

Controller apenas a sugestão, se fossem apenas os campos originais seria assim:

```
$options2 = ['Customers'=>'Customers','Groups'=>'Groups', 'Users'=>'Users',
'Permissions'=>'Permissions', 'Products'=>'Products', 'ProductItems'=>'ProductItems',
'value'=>'Selecione'];
echo $this->Form->input('controller',['options'=>$options2,'required'=>'false', 'class'=>'col-
md-12','empty'=>'Selecione']);
```

- Mudar a largura do campo group e do campo controller usando a classe col4

- Remover a coluna Password do src/Template/Users/index.ctp

Remover as linhas:

```
            <th><?= $this->Paginator->sort('password') ?></th>
```
e
```
            <td><?= h($user->password) ?></td>
```

- Mostrar username ao invés de id na index.ctp de todas as tabelas

Trocar a linha por
```
                <?= $produto->has('user') ? $this->Html->link($produto->user->username,
['controller' => 'Users', 'action' => 'view', $produto->user->id]) : '' ?>
```

Especificamente troque
$produto->user->id

por
$produto->user->username

- Adicionar 'Selecione' para a combo de User em add.ctp de todas as tabelas,

A linha deve ficar assim:

```
        echo $this->Form->input('user_id', ['options' => $users, 'empty'=>'Selecione']);
```

- Aproveite e mude a largura do campo para col4
```
        echo $this->Form->input('user_id', ['options' => $users, 'empty'=>'Selecione',
'class'=>'col4']);
```

- Mostrar nome do cliente e do funcionario na index de pedidos

src/Tempalte/Pedidos/index.php

Mude nas linhas:

```
                <?= $pedido->has('cliente') ? $this->Html->link($pedido->cliente->nome,
['controller' => 'Clientes', 'action' => 'view', $pedido->cliente->id]) : '' ?>
            </td>
```

```
        <td>
            <?= $pedido->has('funcionario') ? $this->Html->link($pedido->funcionario->nome, ['controller' => 'Funcionarios', 'action' => 'view', $pedido->funcionario->id]) : '' ?>
```

Mudar
$pedido->cliente->nome
$pedido->funcionario->nome

Observe que em compra_itens ele não mostrou o estoque_id como combo
Então vamos ajustar isso

Editar o CompraItensController.php

Vamos ajustar o action add()

Adicone a linha abaixo:

```
$estoque = $this->CompraItens->Estoque->find('list', ['limit' => 200]);
```

E altere esta

```
$this->set(compact('compraIten', 'compras', 'produtos', 'estoque'));
```

Alterar src/Template/CompraItens/add.ctp

Apenas a linha

```
echo $this->Form->input('estoque_id', ['options' => $estoque]);
```

Agora com edit

Adicione

```
$estoque = $this->CompraItens->Estoque->find('list', ['limit' => 200]);
```

Altere

```
$this->set(compact('compraIten', 'compras', 'produtos','estoque'));
```

No action edit()

Altere apenas a linha:

```
echo $this->Form->input('estoque_id', ['options' => $estoque]);
```

Adicionar Estoque para o action index() na linha

```
'contain' => ['Compras', 'Produtos', 'Estoque']
```

Agora na view edit.ctp

```
echo $this->Form->input('estoque_id', ['options' => $estoque]);
```

No index.ctp adicione logo abaixo da linha

```
<td><?= $this->Number->format($compraIten->preco_unitario) ?></td>
```

Estas:

```
    <td>
```

```
                <?= $compraIten->has('estoque') ? $this->Html->link($compraIten-
>estoque_id, ['controller' => 'Estoque', 'action' => 'view', $compraIten->estoque_id]) : '' ?>
            </td>
```

E remova a linha existente

```
            <td><?= $this->Number->format($compraIten->estoque_id) ?></td>
```

Vamos mostrar a descricao do produto ao inves do id:

Alterre de id para descricao em index.ctp

```
<?= $compraIten->has('produto') ? $this->Html->link($compraIten->produto->descricao,
['controller' => 'Produtos', 'action' => 'view', $compraIten->produto->id]) : '' ?>
```

Boa parte dos formulários agora recebbem o foco no primeiro campo ao abrir, inclusive o login usando a propriedade autofocus do HTML5:

```
'autofocus' => 'true'
```

Nesta etapa estaremos codificando nosso aplicativo para adicionar a camada de negócios do estoque:

Precisamos tomar um grande cuidado quando estivermos modelando um banco de dados, ao criar astabelas, seus campos, suas constraints, chaves primárias, chaves estrangeiras. Esta fase do projeto é vital. Erros aqui se propagam e prejudicam o projeto. Algo que não pode acontecer é uma tabela A ter um relacionametno com uma tabela B e também a tabela B ter o relacionamento com a tabela A. Cuidado para não cometer estes descuidos. Claro que precisamos ter bem claros os conceitos de bancos de dados e das formas normais em mente.1

Cadastros primários que não têm relacionamentos, exceto com users e ceps e também as tabelas administrativas, groups, users e permissions:

Chamadas Tabelas primárias, pois outras se relacionam com ela e estas não guardam chaves estrangeiras

- empresas
- ceps
- armazens
- clientes
- fornecedores
- produtos

Tabelas secundárias, que contém relacionamentos, ou seja, um campo que relaciona com a tabela principal e a chave estrangeira:

- funcionarios
- compras

- compra_itens
- pedidos
- pedido_itens
- bonus
- comissoes
- estoque

São num total 14 tabelas

Tipos de dados

Estoque, comprar, pedidos devem ter um tipo de dados datetime ou timestamp para permitir mais de um por dia. Aqui é uma simplificação

Sequência de Cadastro

- Cadastramos sempre as informações nas tabelas primárias. Não posso cadastrar uma empresa sem o CEP (isso se CEP for obrigatório). Não posso cadastrar um pedido para um cliente se o cliente ainda não foi cadastrado. Primeiro temos que cadastrar o cliente.

- Antes de cadastrar um funcionario precisamos cadastrar a empresa

- Antes de cadastrar uma compra precisamos cadastrar os fornecedores

- Assim sucessivamente, cadastrar primeiro a tabela primária depois a secundária/estrangeira. Para o usuário que está cadastrando, basta abrir o add e olhar os campos. Se existir alguma combo que traz ou trará informações de outra tabela ele precisa cadastrar primeiro esta tabela da combo. Exemplo, ele abre Produtos para cadastrar um novo produto. Sem problema, pois não existem combos. Todas as tabelas cujo nome tem Itens, precisa cadastrar a principal antes (compras, pedidos). Mas se ele quizer cadastrar um novo estoque. Veja que tem uma combo Armazen. Se ele for cadastrar em um dos armazens da combo tá beleza, mas se for em um que não está na relação, ele deve fechar o estoque e abrir Armazens para cadastrar o novo armazen. Ainda no estoque ele percebe que não existe CompraItens cadastrado. Então terá que abrir CompaItens e talvez compras e cadastrar primeiro a compra e depois voltar para Estoque. Também acontece se existirem compraitens cadasrados mas não existir a compra iten que ele deseja. Precisará cadastrar antes a compra e somente depois voltar para o Estoque.

Operações que precisamos fazer:

As operações devem ficar dentro de uma transação

- Na compra precisamoms consultar os armazens, com as indicações de armários e prateleiras e também informações dos fornecedores e também consultar o estoque_min e o estoque_max em estoque através do relacionamento

- Quando um pedido for feito precisamos atualizar a quant_estoque na tabela estoque

- Subtraindo a quantidade existente em quant_estoque da quantidade do pedido_itens
- Precisamos registrar na tabela comissoes o valor da comissão deste pedido, que é de 10% do valor do pedido e este é a soma dos (quantidade X preco_venda) do pedido_itens
  Veja que em comissoes tem o codigo do pedido, de onde pegaremos o funcionario_id
- Para pagar as comissoes dos funcionarios pegaremos o funcionario_id do pedidos e o total de pedido_itens, que é o somatório de (quantidade x preco_venda)

- Encontrar uma forma de avisar quando o estoque mínimo for atingido