



# **CakePHP Cookbook Documentation**

*Versão 3.x*

**Cake Software Foundation**

18 September, 2015



---

# Conteúdo

---

<b>1</b>	<b>CakePHP num piscar de olhos</b>	<b>1</b>
	Convenções Sobre Configuração . . . . .	1
	A camada Model . . . . .	1
	A camada View . . . . .	2
	A camada Controller . . . . .	2
	Ciclo de Requisições do CakePHP . . . . .	3
	Apenas o Começo . . . . .	4
	Leitura adicional . . . . .	4
<b>2</b>	<b>Guia de Início Rápido</b>	<b>11</b>
	Tutorial - Criando um Bookmarker - Parte 1 . . . . .	11
	Tutorial - Criando um Bookmarker - Parte 2 . . . . .	17
<b>3</b>	<b>3.0 Guia de migração</b>	<b>25</b>
<b>4</b>	<b>Tutoriais &amp; Exemplos</b>	<b>27</b>
	Tutorial - Criando um Bookmarker - Parte 1 . . . . .	27
	Tutorial - Criando um Bookmarker - Parte 2 . . . . .	33
	Tutorial - Criando um Blog - Parte 1 . . . . .	41
	Tutorial - Criando um Blog - Parte 2 . . . . .	44
	Tutorial - Criando um Blog - Parte 3 . . . . .	55
	Tutorial - Criando um Blog - Autenticação e Autorização . . . . .	56
<b>5</b>	<b>Contribuindo</b>	<b>57</b>
	Documentação . . . . .	57
	Tickets . . . . .	57
	Código . . . . .	57
	Padrões de código . . . . .	58
	Guia de retrocompatibilidade . . . . .	58

<b>6</b>	<b>Instalação</b>	<b>59</b>
	Requisitos . . . . .	59
	Instalando o CakePHP . . . . .	60
	Permissões . . . . .	61
	Servidor de Desenvolvimento . . . . .	61
	Produção . . . . .	62
	Aquecendo . . . . .	62
	Reescrita de URL . . . . .	63
<b>7</b>	<b>Configuração</b>	<b>69</b>
<b>8</b>	<b>Roteamento</b>	<b>71</b>
	Filtros do Dispatcher . . . . .	71
<b>9</b>	<b>Objetos de requisição e resposta</b>	<b>73</b>
<b>10</b>	<b>Controllers (Controladores)</b>	<b>75</b>
	O App Controller . . . . .	75
	Fluxo de requisições . . . . .	76
	Métodos (actions) de controllers . . . . .	77
	Redirecionando para outras páginas . . . . .	80
	Carregando models adicionais . . . . .	81
	Paginando um model . . . . .	82
	Configurando components para carregar . . . . .	82
	Configurando helpers para carregar . . . . .	83
	Ciclo de vida de callbacks em uma requisição . . . . .	83
	Mais sobre controllers . . . . .	84
<b>11</b>	<b>Views (Visualização)</b>	<b>87</b>
	More About Views . . . . .	87
<b>12</b>	<b>Models (Modelos)</b>	<b>93</b>
	Exemplo rápido . . . . .	93
	Mais informação . . . . .	95
<b>13</b>	<b>Authentication</b>	<b>101</b>
<b>14</b>	<b>Bake Console</b>	<b>103</b>
	Instalação . . . . .	103
<b>15</b>	<b>Caching</b>	<b>105</b>
<b>16</b>	<b>Console e Shells</b>	<b>107</b>
	O Console do CakePHP . . . . .	107
	Criando uma Shell . . . . .	108
	Tasks de Shell . . . . .	110
	Invocando outras Shells a partir da sua Shell . . . . .	112
	Recenendo Input de usuários . . . . .	112
	Criando Arquivos . . . . .	112

Saída de dados do Console . . . . .	113
Opções de configuração e Geração de ajuda . . . . .	115
Roteamento em Shells / CLI . . . . .	123
Métodos enganchados . . . . .	123
Mais tópicos . . . . .	123
<b>17 Debugging</b>	<b>127</b>
<b>18 Implantação</b>	<b>129</b>
Atualizar config/app.php . . . . .	129
Checar a segurança . . . . .	130
Definir a raiz do documento . . . . .	130
Aprimorar a performance de sua aplicação . . . . .	130
<b>19 Email</b>	<b>131</b>
<b>20 Erros &amp; Exceções</b>	<b>133</b>
<b>21 Sistema de eventos</b>	<b>135</b>
<b>22 Internacionalização e Localização</b>	<b>137</b>
<b>23 Logging</b>	<b>139</b>
<b>24 Formulários sem models</b>	<b>141</b>
<b>25 Pagination</b>	<b>143</b>
<b>26 Plugins</b>	<b>145</b>
<b>27 REST</b>	<b>147</b>
<b>28 Segurança</b>	<b>149</b>
Segurança . . . . .	149
Cross Site Request Forgery . . . . .	149
Security . . . . .	150
<b>29 Sessions</b>	<b>151</b>
<b>30 Testando</b>	<b>153</b>
<b>31 Validação</b>	<b>155</b>
<b>32 App Class</b>	<b>157</b>
<b>33 Collections (Coleções)</b>	<b>159</b>
<b>34 Arquivos &amp; Pastas</b>	<b>161</b>
<b>35 Hash</b>	<b>163</b>

<b>36</b>	<b>Http Client</b>	<b>165</b>
<b>37</b>	<b>Inflector</b>	<b>167</b>
<b>38</b>	<b>Número</b>	<b>169</b>
<b>39</b>	<b>Objetos de Registro</b>	<b>171</b>
<b>40</b>	<b>Texto</b>	<b>173</b>
<b>41</b>	<b>Tempo</b>	<b>175</b>
<b>42</b>	<b>Xml</b>	<b>177</b>
<b>43</b>	<b>Constantes e Funções</b>	<b>179</b>
<b>44</b>	<b>Debug Kit</b>	<b>181</b>
	Instalação . . . . .	181
	Armazenamento do DebugKit . . . . .	181
	Uso da barra de ferramentas . . . . .	182
	Usando o painel History . . . . .	182
	Desenvolvendo seus próprios painéis . . . . .	182
<b>45</b>	<b>Migrations</b>	<b>185</b>
<b>46</b>	<b>Apêndices</b>	<b>187</b>
	Guia de Migração para a versão 3.0 . . . . .	187
	Guia de Migração para a versão 3.1 . . . . .	188
	Informações Gerais . . . . .	188
	<b>PHP Namespace Index</b>	<b>189</b>
	<b>Índice</b>	<b>191</b>

---

# CakePHP num piscar de olhos

---

O CakePHP é concebido para tornar tarefas de desenvolvimento web mais simples e fáceis. Por fornecer uma caixa de ferramentas completa para você poder começar, o CakePHP funciona bem em conjunto ou isoladamente.

O objetivo desta análise é introduzir os conceitos gerais presentes no CakePHP, e lhe dar uma rápida visão geral de como estes conceitos são implementados. Se você está ávido para começar um projeto, você pode *começar com o tutorial*, ou mergulhar na documentação.

## Convenções Sobre Configuração

O CakePHP provê uma estrutura organizacional básica que cobre nomenclaturas de classes, nomenclaturas de arquivos, nomenclaturas de banco de dados, e outras convenções. Apesar das convenções levarem algum tempo para serem assimiladas, ao segui-las o CakePHP evita configuração desnecessário e cria uma estrutura de aplicação uniforme que faz trabalhar com vários projetos uma tarefa suave. O *capítulo de convenções* cobre as variadas convenções que o CakePHP utiliza.

## A camada Model

A camada Model representa a parte da sua aplicação que implementa a lógica de negócio. Ela é responsável por recuperar dados e convertê-los nos conceitos significativos primários na sua aplicação. Isto inclui processar, validar, associar ou qualquer outra tarefa relacionada à manipulação de dados.

No caso de uma rede social, a camada Model deveria tomar cuidado de tarefas como salvar os dados do usuário, salvar as associações entre amigos, salvar e recuperar fotos de usuários, localizar sugestões para novos amigos, etc. Os objetos de modelo podem ser pensados como “Friend”, “User”, “Comment”, ou “Photo”. Se nós quiséssemos carregar alguns dados da nossa tabela `users` poderíamos fazer:

```
use Cake\ORM\TableRegistry;

$users = TableRegistry::get('Users');
$query = $users->find();
```

```
foreach ($query as $row) {  
    echo $row->username;  
}
```

Você pode notar que não precisamos escrever nenhum código antes de podermos começar a trabalhar com nossos dados. Por usar convenções, o CakePHP irá utilizar classes padrão para tabelas e entidades ainda não definidas.

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

A View renderiza uma apresentação de dados modelados. Estando separada dos objetos da Model, é responsável por utilizar a informação que tem disponível para produzir qualquer interface de apresentação que a sua aplicação possa precisar.

Por exemplo, a view pode usar dados da model para renderizar uma página HTML que os conhtenha, ou um resultado formatado como XML:

```
// 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; ?>
```

A camada View provê alguma variedade de extensões como *view-elements* e *View Cells (Células de Visualização)* para permitir que você reutilize sua lógica de apresentação.

A camada View não está limitada somente a HTML ou apresentação textual dos dados. Ela pode ser usada para entregar formatos de dado comuns como JSON, XML, e através de uma arquitetura encaixável qualquer outro formato que você venha precisar.

## A camada Controller

A camada Controller manipula requisições dos usuários. É responsável por renderizar uma resposta com o auxílio de ambas as camadas, Model e View respectivamente.

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, checka 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



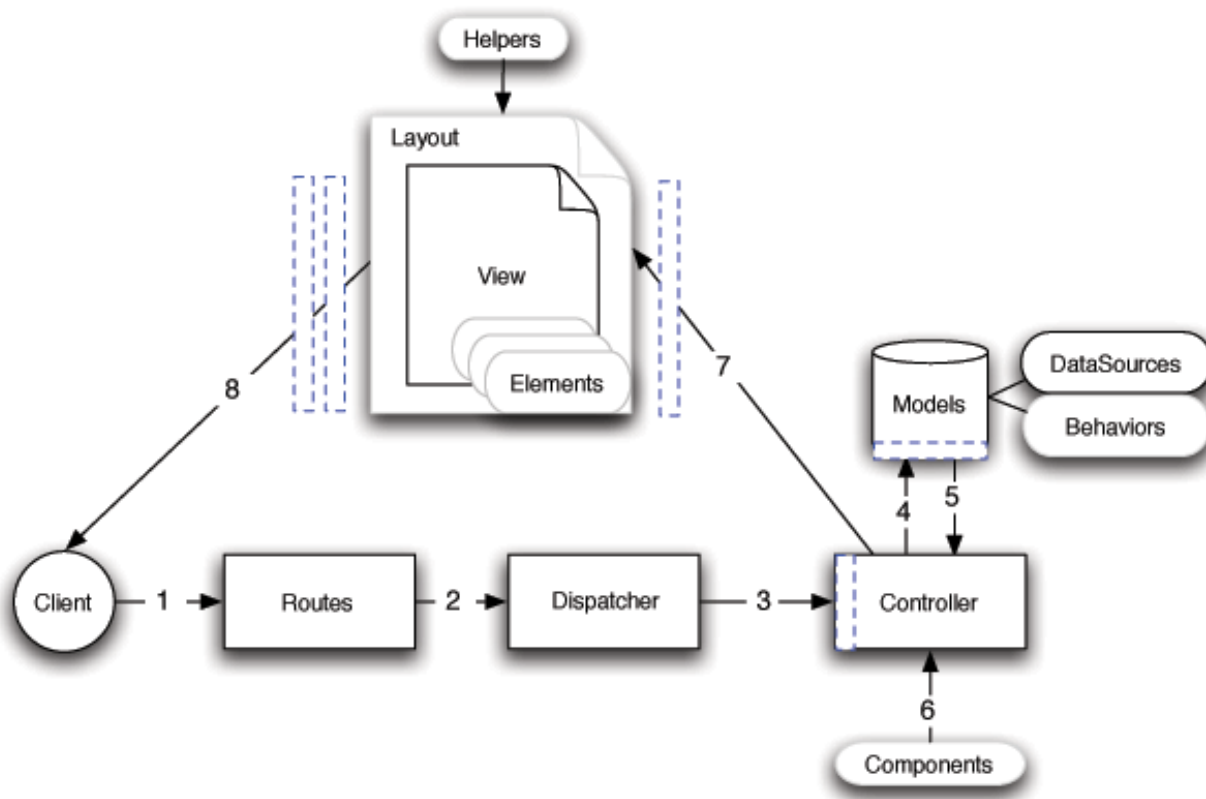
finalmente delega o processo de renderização para a camada View. Um exemplo de controller para registro de usuário seria:

```
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, ['validate' => 'registration'])) {
            $this->Flash->success(__('Você está registrado.'));
        } else {
            $this->Flash->error(__('Houve algum problema.'));
        }
    }
    $this->set('user', $user);
}
```

Você pode perceber que nós nunca renderizamos uma view explicitamente. As convenções do CakePHP tomarão cuidado de selecionar a view correta e renderizá-la como os dados definidos com `set()`.

## Ciclo de Requisições do CakePHP

Agora que você é familiar com as diferentes camadas no CakePHP, vamos revisar como um ciclo de requisição funciona no CakePHP:



O ciclo de requisição típico do CakePHP começa com um usuário solicitando uma página ou recurso na sua aplicação. Em alto nível cada requisição vai através dos seguintes passos:

1. A requisição é primeiramente processada pela suas rotas.
2. Depois da requisição ter sido roteada, o despachante irá selecionar o objeto de controller correto para manipulá-la.
3. A action do controller é chamada e o controller interage com os models e components requisitados.
4. O controller delega a criação de resposta à view para gerar os dados de saída resultantes dos dados do model.

## Apenas o Começo

Esperamos que essa rápida visão geral tenha despertado seu interesse. Alguns outros grandes recursos no CakePHP são:

- *Framework de cache* que integra com Memcached, Redis e outros backends.
- Poderosas *ferramentas de geração de código* para você sair em disparada.
- *Framework de teste integrado* para você assegurar-se que seu código funciona perfeitamente.

Os próximos passos óbvios são *baixar o CakePHP*, ler o *tutorial e construir algo fantástico*.

## Leitura adicional

### Onde Conseguir Ajuda

#### O website oficial do CakePHP

<http://www.cakephp.org>

O website oficial do CakePHP é sempre um ótimo lugar para visitar. Ele provê links para ferramentas comunmente utilizadas por desenvolvedores, screencasts, oportunidades de doação e downloads.

#### O Cookbook

<http://book.cakephp.org>

Esse manual deveria ser o primeiro lugar para onde você iria afim de conseguir respostas. Assim como muitos outros projetos de código aberto, nós conseguimos novos colaboradores regularmente. Tente o seu melhor para responder suas questões por si só. Respostas vão vir lentamente, e provavelmente continuarão longas. Você pode suavizar nossa carga de suporte. Tanto o manual quanto a API possuem um componente online.

## A Bakery

<http://bakery.cakephp.org>

A “padaria” do CakePHP é um local para todas as coisas relacionadas ao CakePHP. Visite-a para tutoriais, estudos de caso e exemplos de código. Uma vez que você tenha se familiarizado com o CakePHP, autentique-se e compartilhe seu conhecimento com a comunidade, ganhe instantaneamente fama e fortuna.

## A API

<http://api.cakephp.org/>

Diretamente ao ponto, dos desenvolvedores do núcleo do CakePHP, a API (Application Programming Interface) do CakePHP é a mais compreensiva documentação sobre os detalhes técnicos e minuciosos sobre o funcionamento interno do framework.

## Os Testes de Caso

Se você sente que a informação provida pela API não é suficiente, verifique os códigos de testes de caso do CakePHP. Eles podem servir como exemplos práticos para funções e utilização de dados referentes a uma classe.:

```
tests/TestCase/
```

## O canal de IRC

### Canal de IRC na irc.freenode.net:

- #cakephp – Discussão geral
- #cakephp-docs – Documentação
- #cakephp-bakery – Bakery
- #cakephp-fr – Canal francês.

Se você está travado, nos faça uma visita no canal de IRC do CakePHP. Alguém do [time de desenvolvimento](#)<sup>1</sup> normalmente está conectado, especialmente nos horários diurnos da América do Sul e América do Norte. Nós apreciaríamos ouvi-lo se você precisar de ajuda, se quiser encontrar usuários da sua área ou ainda se quiser doar seu novo carro esporte.

## Grupo oficial de discussão do CakePHP

Grupo de discussão do Google<sup>2</sup>

---

<sup>1</sup><https://github.com/cakephp?tab=members>

<sup>2</sup><http://groups.google.com/group/cake-php>

O CakePHP também possui seu grupo de discussão oficial no Google Grupos. Existem milhares de pessoas discutindo projetos CakePHP, ajudando uns aos outros, resolvendo problemas, construindo projetos e compartilhando idéias. Pode ser uma grande fonte para encontrar respostas arquivadas, perguntas frequentes e conseguir respostas para problemas imediatos. Junte-se a outros usuários do CakePHP e comece a conversar.

### Stackoverflow

<http://stackoverflow.com/><sup>3</sup>

Marque suas questões com a tag `cakephp` e especifique a versão que você está utilizando para permitir que usuários do stackoverflow achem suas questões.

### Onde conseguir ajuda em sua língua

#### Francês

- [Comunidade CakePHP francesa](#)<sup>4</sup>

#### Português brasileiro

- [Comunidade CakePHP brasileira](#)<sup>5</sup>

### Convenções do CakePHP

Nós somos grandes fãs de convenção sobre configuração. Apesar de levar um pouco de tempo para aprender as convenções do CakePHP, você economiza tempo a longo prazo. Ao seguir as convenções, você ganha funcionalidades instantaneamente e liberta-se do pesadelo de manutenção e rastreamento de arquivos de configuração. Convenções também prezam por uma experiência de desenvolvimento uniforme, permitindo que outros desenvolvedores ajudem mais facilmente.

### Convenções para Controllers

Os nomes das classes de Controllers são pluralizados, CamelCased, e terminam em `Controller`. `PeopleController` e `LatestArticlesController` são exemplos de nomes convencionais para controllers.

Métodos públicos nos Controllers são frequentemente referenciados como 'actions' acessíveis através de um navegador web. Por exemplo, o `/articles/view` mapeia para o método `view()` do `ArticlesController` sem nenhum esforço. Métodos privados ou protegidos não podem ser acessados pelo roteamento.

---

<sup>3</sup><http://stackoverflow.com/questions/tagged/cakephp/>

<sup>4</sup><http://cakephp-fr.org>

<sup>5</sup><http://cakephp-br.org>

## Considerações de URL para nomes de Controller

Como você acabou de ver, controllers singulares mapeiam facilmente um caminho simples, todo em minúsculo. Por exemplo, `ApplesController` (o qual deveria ser definido no arquivo de nome `'ApplesController.php'`) é acessado por `http://example.com/apples`.

Controllers com múltiplas palavras *podem* estar em qualquer forma 'flexionada' igual ao nome do controller, então:

- `/redApples`
- `/RedApples`
- `/Red_apples`
- `/red_apples`

Todos resolverão para o index do controller `RedApples`. Porém, a forma correta é que suas URLs sejam minúsculas e separadas por sublinhado, portanto `/red_apples/go_pick` é a forma correta de acessar a action `RedApplesController::go_pick`.

When you create links using `this->Html->link()`, you can use the following conventions for the url array:

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

Para mais informações sobre o manuseio de URLs e parâmetros do CakePHP, veja *routes-configuration*.

## Convenções para nomes de Classes e seus nomes de arquivos

No geral, nomes de arquivos correspondem aos nomes das classes, e seguem os padrões PSR-0 ou PSR-4 para auto-carregamento. A seguir seguem exemplos de nomes de classes e de seus arquivos:

- A classe de Controller **KissesAndHugsController** deveria ser encontrada em um arquivo nomeado **KissesAndHugsController.php**
- A classe de Component **MyHandyComponent** deveria ser encontrada em um arquivo nomeado **MyHandyComponent.php**
- A classe de Table **OptionValuesTable** deveria ser encontrada em um arquivo nomeado **OptionValuesTable.php**.
- A classe de Entity **OptionValue** deveria ser encontrada em um arquivo nomeado **OptionValue.php**.
- A classe de Behavior **EspecialyFunkableBehavior** deveria ser encontrada em um arquivo nomeado **EspecialyFunkableBehavior.php**
- A classe de View **SuperSimpleView** deveria ser encontrada em um arquivo nomeado **SuperSimpleView.php**

- A classe de Helper **BestEverHelper** deveria ser encontrada em um arquivo nomeado **BestEverHelper.php**

Cada arquivo deveria estar localizado no diretório/namespace apropriado de sua aplicação.

### Convenções para Models e Databases

Os nomes de classe de Tables são pluralizadas e CamelCased. People, BigPeople, and ReallyBigPeople são todos exemplos convencionais de models.

Os nomes de Tables correspondentes aos models do CakePHP são pluralizadas e separadas por sublinhado. As tables sublinhadas para os models mencionados acima seriam `people`, `big_people`, e `really_big_people`, respectively.

Você pode utilizar a biblioteca utility `Cake\Utility\Inflector` para checar o singular/plural de palavras. Veja o *Inflector* para mais informações. Recomenda-se que as tables sejam criadas e mantidas na língua inglesa.

Campos com duas ou mais palavras são separados por sublinhado: `first_name`.

Chaves estrangeiras nos relacionamentos `hasMany`, `belongsTo` ou `hasOne` são reconhecidas por padrão como o nome (singular) da table relacionada seguida por `_id`. Então se `Bakers` `hasMany` `Cakes`, a table `cakes` irá referenciar-se para a table `bakers` através da chave estrangeira `baker_id`. Para uma tabela como `category_types` a qual o nome contém mais palavras, a chave estrangeira seria a `category_type_id`.

tables de união, usadas no relacionamento `BelongsToMany` entre models, devem ser nomeadas depois das tables que ela está unindo, ordenadas em ordem alfabética (`apples_zebras` ao invés de `zebras_apples`).

### Convenções para Views

Arquivos de template views são nomeadas seguindo as funções que a exibem do controller, separadas por sublinhado. A função `getReady()` da classe `PeopleController` buscará por um template view em `src/Template/People/get_ready.ctp`. O padrão é `src/Template/Controller/underscored_function_name.ctp`.

Por nomear as partes de sua aplicação utilizando as convenções do CakePHP, você ganha funcionalidades sem luta e sem amarras de configuração. Aqui está um exemplo final que enlaça as convenções juntas:

- Table: “people”
- Classe Table: “PeopleTable”, encontrada em `src/Model/Table/PeopleTable.php`
- Classe Entity: “Person”, encontrada em `src/Model/Entity/Person.php`
- Classe Controller: “PeopleController”, encontrada em `src/Controller/PeopleController.php`
- View template, encontrado em `src/Template/People/index.ctp`

Utilizando estas convenções, o CakePHP sabe que uma requisição para `http://example.com/people/` mapeia para uma chamada da função `index()` do `PeopleController`, onde o model `Person` é automaticamente disponibilizado (e automaticamente amarrado à table ‘people’ no banco de dados), e então renderiza-se um arquivo view template. Nenhuma destes relacionamentos foi configurado de qualquer forma se não por criar classes e arquivos que você precisaria criar de qualquer forma.

Agora que você foi introduzido aos fundamentos do CakePHP, você pode tentar seguir através do [Tutorial - Criando um Blog - Parte 1](#) para ver como as coisas se encaixam juntas.

## Estrutura de pastas do CakePHP

Depois de você ter baixado e extraído o CakePHP, aí estão os arquivos e pastas que você deve ver:

- bin
- config
- logs
- plugins
- src
- tests
- tmp
- vendor
- webroot
- .htaccess
- composer.json
- index.php
- README.md

Você notará alguns diretórios principais:

- The *bin* folder holds the Cake console executables.
- O diretório *config* contem os (poucos) *Configuração* arquivos de configuração que o CakePHP utiliza. Detalhes de conexão com banco de dados, inicialização, arquivos de configuração do núcleo da aplicação, e relacionados devem ser postos aqui.
- O diretório *logs* será normalmente onde seus arquivos de log ficarão, dependendo das suas configurações.
- O diretório *plugins* será onde *Plugins* que sua aplicação utiliza serão armazenados.
- O diretório *src* será onde você fará sua magia: é onde os arquivos da sua aplicação serão colocados.
- O diretório *tests* será onde você colocará os testes de caso para sua aplicação.
- O diretório *tmp* será onde o CakePHP armazenará dados temporários. O modo como os dados serão armazenados depende da configuração do CakePHP, mas esse diretório é comunmente usado para armazenar descrições de modelos e algumas vezes informação de sessão.
- O diretório *vendor* será onde o CakePHP e outras dependências da aplicação serão instalados. Faça uma nota pessoal para **não** editar arquivos deste diretório. Nós não podemos ajudar se você tivé-lo feito.

- O diretório *webroot* será a raiz pública de documentos da sua aplicação. Ele contém todos os arquivos que você gostaria que fossem públicos.

Certifique-se que os diretórios *tmp* e *logs* existem e são passíveis de escrita, senão a performance de sua aplicação será severamente impactada. Em modo de debug, o CakePHP irá alertá-lo se este for o caso.

### O diretório *src*

O diretório *src* do CakePHP é onde você fará a maior parte do desenvolvimento de sua aplicação. Vamos ver mais de perto a estrutura de pastas dentro de *src*.

**Console** Contém os comandos e tarefas de console para sua aplicação. Para mais informações veja [Console e Shells](#).

**Controller** Contém os controllers de sua aplicação e seus componentes.

**Locale** Armazena arquivos textuais para internacionalização.

**Model** Contém as tables, entities e behaviors de sua aplicação.

**View** Classes de apresentação são alocadas aqui: cells, helpers, e arquivos view.

**Template** Arquivos de apresentação são alocados aqui: elements, páginas de erro, layouts, e templates view.



---

# Guia de Início Rápido

---

A melhor forma de viver experiências e aprender sobre CakePHP é sentar e construir algo. Para começar nós iremos construir uma aplicação simples de blog.

## Tutorial - Criando um Bookmarker - Parte 1

Esse tutorial vai guiar você através da criação de uma simples aplicação de marcação (bookmarker). Para começar, nós vamos instalar o CakePHP, criar nosso banco de dados, e usar as ferramentas que o CakePHP fornece para obter nossa aplicação de pé rápido.

Aqui está o que você vai precisar:

1. Um servidor de banco de dados. Nós vamos usar o servidor MySQL neste tutorial. Você precisa saber o suficiente sobre SQL para criar um banco de dados: O CakePHP vai tomar as rédeas a partir daí. Por nós estarmos usando o MySQL, também certifique-se que você tem a extensão `pdo_mysql` habilitada no PHP.
2. Conhecimento básico sobre PHP.

Vamos começar!

### Instalação do CakePHP

A maneira mais fácil de instalar o CakePHP é usando Composer, um gerenciador de dependências para o PHP. É uma forma simples de instalar o CakePHP a partir de seu terminal ou prompt de comando. Primeiro, você precisa baixar e instalar o Composer. Se você tiver instalada a extensão `cURL` do PHP, execute o seguinte comando:

```
curl -s https://getcomposer.org/installer | php
```

Ao invés disso, você também pode baixar o arquivo `composer.phar` do [site](https://getcomposer.org/)<sup>1</sup> oficial.

---

<sup>1</sup><https://getcomposer.org/download/>

Em seguida, basta digitar a seguinte linha no seu terminal a partir do diretório onde se localiza o arquivo `composer.phar` para instalar o esqueleto de aplicações do CakePHP no diretório `bookmark`.

```
php composer.phar create-project --prefer-dist cakephp/app bookmarker
```

A vantagem de usar Composer é que ele irá completar automaticamente um conjunto importante de tarefas, como configurar as permissões de arquivo e criar a sua **config/app.php**.

Há outras maneiras de instalar o CakePHP. Se você não puder ou não quiser usar Composer, veja a seção *Instalação*.

Independentemente de como você baixou o CakePHP, uma vez que sua instalação for concluída, a estrutura dos diretórios deve ficar parecida com o seguinte:

```
/bookmarker
  /bin
  /config
  /logs
  /plugins
  /src
  /tests
  /tmp
  /vendor
  /webroot
  .editorconfig
  .gitignore
  .htaccess
  .travis.yml
  composer.json
  index.php
  phpunit.xml.dist
  README.md
```

Agora pode ser um bom momento para aprender sobre como a estrutura de diretórios do CakePHP funciona: Confira a seção *Estrutura de pastas do CakePHP*.

### Verificando nossa instalação

Podemos checar rapidamente que a nossa instalação está correta, verificando a página inicial padrão. Antes que você possa fazer isso, você vai precisar iniciar o servidor de desenvolvimento:

```
bin/cake server
```

Isto irá iniciar o servidor embutido do PHP na porta 8765. Abra `http://localhost:8765` em seu navegador para ver a página de boas-vindas. Todas as verificações devem estar checadadas corretamente, a não ser a conexão com banco de dados do CakePHP. Se não, você pode precisar instalar extensões do PHP adicionais, ou definir permissões de diretório.

### Criando o banco de dados

Em seguida, vamos criar o banco de dados para a nossa aplicação. Se você ainda não tiver feito isso, crie um banco de dados vazio para uso nesse tutorial, com um nome de sua escolha, por exemplo,

cake\_bookmarks. Você pode executar o seguinte SQL para criar as tabelas necessárias:

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    created DATETIME,
    modified DATETIME
);

CREATE TABLE bookmarks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    title VARCHAR(50),
    description TEXT,
    url TEXT,
    created DATETIME,
    modified DATETIME,
    FOREIGN KEY user_key (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    created DATETIME,
    modified DATETIME,
    UNIQUE KEY (title)
);

CREATE TABLE bookmarks_tags (
    bookmark_id INT NOT NULL,
    tag_id INT NOT NULL,
    PRIMARY KEY (bookmark_id, tag_id),
    INDEX tag_idx (tag_id, bookmark_id),
    FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
    FOREIGN KEY bookmark_key(bookmark_id) REFERENCES bookmarks(id)
);
```

Você deve ter notado que a tabela `bookmarks_tags` utilizada uma chave primária composta. O CakePHP suporta chaves primárias compostas quase todos os lugares, tornando mais fácil construir aplicações multi-arrendados.

Os nomes de tabelas e colunas que usamos não foram arbitrárias. Usando *convenções de nomenclatura* do CakePHP, podemos alavancar o desenvolvimento e evitar ter de configurar o framework. O CakePHP é flexível o suficiente para acomodar até mesmo esquemas de banco de dados legados inconsistentes, mas aderir às convenções vai lhe poupar tempo.

## Configurando o banco de dados

Em seguida, vamos dizer ao CakePHP onde o nosso banco de dados está como se conectar a ele. Para muitos, esta será a primeira e última vez que você vai precisar configurar qualquer coisa.

A configuração é bem simples: basta alterar os valores do array `Datasources.default` no arquivo

`config/app.php` pelos que se aplicam à sua configuração. A amostra completa da gama de configurações pode ser algo como o seguinte:

```
return [
    // Mais configuração acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_bookmarks',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Mais configuração abaixo.
];
```

Depois de salvar o seu arquivo `config/app.php`, você deve notar que a mensagem ‘CakePHP is able to connect to the database’ tem uma marca de verificação.

---

**Nota:** Uma cópia do arquivo de configuração padrão do CakePHP é encontrado em `config/app.default.php`.

---

## Gerando o código base

Devido a nosso banco de dados seguir as convenções do CakePHP, podemos usar o *bake console* para gerar rapidamente uma aplicação básica . Em sua linha de comando execute:

```
bin/cake bake all users
bin/cake bake all bookmarks
bin/cake bake all tags
```

Isso irá gerar os controllers, models, views, seus casos de teste correspondentes, e fixtures para os nossos users, bookmarks e tags. Se você parou seu servidor, reinicie-o e vá para `http://localhost:8765/bookmarks`.

Você deverá ver uma aplicação que dá acesso básico, mas funcional a tabelas de banco de dados. Adicione alguns users, bookmarks e tags.

## Adicionando criptografia de senha

Quando você criou seus users, você deve ter notado que as senhas foram armazenadas como texto simples. Isso é muito ruim do ponto de vista da segurança, por isso vamos consertar isso.

Este também é um bom momento para falar sobre a camada de modelo. No CakePHP, separamos os métodos que operam em uma coleção de objetos, e um único objeto em diferentes classes. Métodos que operam na recolha de entidades são colocadas na classe *Table*, enquanto as características pertencentes a um único registro são colocados na classe *Entity*.

Por exemplo, a criptografia de senha é feita no registro individual, por isso vamos implementar esse comportamento no objeto entidade. Dada a circunstância de nós querermos criptografar a senha cada vez que é definida, vamos usar um método modificador/definidor. O CakePHP vai chamar métodos de definição baseados em convenções a qualquer momento que uma propriedade é definida em uma de suas entidades. Vamos adicionar um definidor para a senha. Em `src/Model/Entity/User.php` adicione o seguinte:

```
namespace App\Model\Entity;

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

class User extends Entity
{
    // Code from bake.

    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($value);
    }
}
```

Agora atualize um dos usuários que você criou anteriormente, se você alterar sua senha, você deve ver um senha criptografada ao invés do valor original nas páginas de lista ou visualização. O CakePHP criptografa senhas com `bcrypt`<sup>2</sup> por padrão. Você também pode usar `sha1` ou `md5` caso venha a trabalhar com um banco de dados existente.

## Recuperando bookmarks com uma tag específica

Agora que estamos armazenando senhas com segurança, podemos construir algumas características mais interessantes em nossa aplicação. Uma vez que você acumulou uma coleção de bookmarks, é útil ser capaz de pesquisar através deles por tag. Em seguida, vamos implementar uma rota, a ação do controller, e um método localizador para pesquisar através de bookmarks por tag.

Idealmente, nós teríamos uma URL que se parece com `http://localhost:8765/bookmarks/tagged/funny/ca`. Isso deveria nos permitir a encontrar todos os bookmarks que têm as tags 'funny', 'cat' e 'gifs'. Antes de podermos implementar isso, vamos adicionar uma nova rota. Em `config/routes.php`, adicione o seguinte na parte superior do arquivo:

```
Router::scope(
    '/bookmarks',
    ['controller' => 'Bookmarks'],
    function ($routes) {
        $routes->connect('/tagged/*', ['action' => 'tags']);
    }
);
```

<sup>2</sup><http://codahale.com/how-to-safely-store-a-password/>

```
}  
);
```

O acima define uma nova “rota” que liga o caminho `/bookmarks/tagged/*`, a `BookmarksController::tags()`. Ao definir rotas, você pode isolar como suas URLs parecerão, de como eles são implementadas. Se fôssemos visitar `http://localhost:8765/bookmarks/tagged`, deveríamos ver uma página de erro informativa do CakePHP. Vamos implementar esse método ausente agora. Em `src/Controller/BookmarksController.php` adicione o seguinte:

```
public function tags()  
{  
    $tags = $this->request->params['pass'];  
    $bookmarks = $this->Bookmarks->find('tagged', [  
        'tags' => $tags  
    ]);  
    $this->set(compact('bookmarks', 'tags'));  
}
```

### Criando o método localizador

No CakePHP nós gostamos de manter as nossas ações do controller enxutas, e colocar a maior parte da lógica de nossa aplicação nos modelos. Se você fosse visitar a URL `/bookmarks/tagged` agora, você veria um erro sobre o método `findTagged` não estar implementado ainda, então vamos fazer isso. Em `src/Model/Table/BookmarksTable.php` adicione o seguinte:

```
public function findTagged(Query $query, array $options)  
{  
    $fields = [  
        'Bookmarks.id',  
        'Bookmarks.title',  
        'Bookmarks.url',  
    ];  
    return $this->find()  
        ->distinct($fields)  
        ->matching('Tags', function ($q) use ($options) {  
            return $q->where(['Tags.title IN' => $options['tags']]);  
        });  
}
```

Nós implementamos um método *localizador customizado*. Este é um conceito muito poderoso no CakePHP que lhe permite construir consultas reutilizáveis. Em nossa pesquisa, nós alavancamos o método `matching()` que nos habilita encontrar bookmarks que têm uma tag ‘correspondente’.

### Criando a view

Agora, se você visitar a URL `/bookmarks/tagged`, o CakePHP irá mostrar um erro e deixá-lo saber que você ainda não fez um arquivo view. Em seguida, vamos construir o arquivo view para a nossa ação `tags`. Em `src/Template/Bookmarks/tags.ctp` coloque o seguinte conteúdo:

```

<h1>
    Bookmarks tagged with
    <?= $this->Text->toList($tags) ?>
</h1>

<section>
<?php foreach ($bookmarks as $bookmark): ?>
    <article>
        <h4><?= $this->Html->link($bookmark->title, $bookmark->url) ?></h4>
        <small><?= h($bookmark->url) ?></small>
        <?= $this->Text->autoParagraph($bookmark->description) ?>
    </article>
<?php endforeach; ?>
</section>

```

O CakePHP espera que os nossos templates sigam a convenção de nomenclatura onde o nome do template é a versão minúscula e grifada do nome da ação do controller.

Você pode perceber que fomos capazes de utilizar as variáveis `$tags` e `bookmarks` em nossa view. Quando usamos o método `set()` em nosso controller, automaticamente definimos variáveis específicas que devem ser enviadas para a view. A view vai tornar todas as variáveis passadas disponíveis nos templates como variáveis locais.

Em nossa view, usamos alguns dos *helpers* nativos do CakePHP. Helpers são usados para criar lógica reutilizável para a formatação de dados, a criação de HTML ou outra saída da view.

Agora você deve ser capaz de visitar a URL `/bookmarks/tagged/funny` e ver todas os bookmarks com a tag 'funny'.

Até agora, nós criamos uma aplicação básica para gerenciar bookmarks, tags e users. No entanto, todos podem ver as tags de toda a gente. No próximo capítulo, vamos implementar a autenticação e restringir os bookmarks visíveis para somente aqueles que pertencem ao usuário atual.

Agora vá a [Tutorial - Criando um Bookmarker - Parte 2](#) para continuar a construir sua aplicação ou mergulhe na documentação para saber mais sobre o que CakePHP pode fazer por você.

## Tutorial - Criando um Bookmarker - Parte 2

Depois de terminar a *primeira parte deste tutorial*, você deve ter uma aplicação muito básica. Neste capítulo iremos adicionar autenticação e restringir as bookmarks para que cada usuário possa ver/modificar somente aquelas que possuam.

### Adicionando login

No CakePHP, a autenticação é feita por *Components (Componentes)*. 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. Para começar, vamos adicionar o `AuthComponent` a nossa aplicação. Nós vamos querer muito que cada método exija autenticação, por isso vamos acrescentar o *AuthComponent* em nosso `AppController`:

```
// Em src/Controller/AppController.php
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authenticate' => [
                'Form' => [
                    'fields' => [
                        'username' => 'email',
                        'password' => 'password'
                    ]
                ]
            ],
            'loginAction' => [
                'controller' => 'Users',
                'action' => 'login'
            ]
        ]);

        // Permite a ação display, assim nosso pages controller
        // continua a funcionar.
        $this->Auth->allow(['display']);
    }
}
```

Acabamos de dizer ao CakePHP que queremos carregar os components `Flash` e `Auth`. Além disso, temos a configuração personalizada do `AuthComponent`, assim a nossa tabela `users` pode usar `email` como `username`. Agora, se você for a qualquer URL, você vai ser chutado para `/users/login`, que irá mostrar uma página de erro já que não escrevemos o código ainda. Então, vamos criar a ação de login:

```
// Em src/Controller/UsersController.php

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.');
```

E em `src/Template/Users/login.ctp` adicione o seguinte:



```
<h1>Login</h1>
<?= $this->Form->create() ?>
<?= $this->Form->input('email') ?>
<?= $this->Form->input('password') ?>
<?= $this->Form->button('Login') ?>
<?= $this->Form->end() ?>
```

Agora que temos um simples formulário de login, devemos ser capazes de efetuar login com um dos users que tenham senha criptografada.

---

**Nota:** Se nenhum de seus users tem senha criptografada, comente a linha `loadComponent('Auth')`. Então vá e edite o user, salvando uma nova senha para ele.

---

Agora você deve ser capaz de entrar. Se não, certifique-se que você está usando um user que tenha senha criptografada.

## Adicionando logout

Agora que as pessoas podem efetuar o login, você provavelmente vai querer fornecer uma maneira de encerrar a sessão também. Mais uma vez, no `UsersController`, adicione o seguinte código:

```
public function logout ()
{
    $this->Flash->success('You are now logged out. ');
    return $this->redirect($this->Auth->logout ());
}
```

Agora você pode visitar `/users/logout` para sair e ser enviado à página de login.

## Ativando inscrições

Se você não estiver logado e tentar visitar `/usuários/adicionar` você vai ser expulso para a página de login. Devemos corrigir isso se quisermos que as pessoas se inscrevam em nossa aplicação. No `UsersController` adicione o seguinte:

```
public function beforeFilter(\Cake\Event\Event $event)
{
    $this->Auth->allow(['add']);
}
```

O texto acima diz ao `AuthComponent` que a ação `add` não requer autenticação ou autorização. Você pode querer dedicar algum tempo para limpar a `/users/add` e remover os links enganosos, ou continuar para a próxima seção. Nós não estaremos construindo a edição do usuário, visualização ou listagem neste tutorial, então eles não vão funcionar, já que o `AuthComponent` vai negar-lhe acesso a essas ações do controller.

## Restringindo acesso

Agora que os usuários podem conectar-se, nós vamos querer limitar os bookmarks que podem ver para aqueles que fizeram. Nós vamos fazer isso usando um adaptador de ‘autorização’. Sendo os nossos requisitos bastante simples, podemos escrever um código em nossa `BookmarksController`. Mas antes de fazer isso, vamos querer dizer ao `AuthComponent` como nossa aplicação vai autorizar ações. Em seu `AppController` adicione o seguinte:

```
public function isAuthorized($user)
{
    return false;
}
```

Além disso, adicione o seguinte à configuração para `Auth` em seu `AppController`:

```
'authorize' => 'Controller',
```

Seu método `initialize` agora deve parecer com:

```
public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize'=> 'Controller', //added this line
        'authenticate' => [
            'Form' => [
                'fields' => [
                    'username' => 'email',
                    'password' => 'password'
                ]
            ]
        ],
        'loginAction' => [
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => $this->referer()
    ]);

    // Permite a ação display, assim nosso pages controller
    // continua a funcionar.
    $this->Auth->allow(['display']);
}
```

Vamos usar como padrão, negação do acesso, e de forma incremental conceder acesso onde faça sentido. Primeiro, vamos adicionar a lógica de autorização para os bookmarks. Em seu `BookmarksController` adicione o seguinte:

```
public function isAuthorized($user)
{
    $action = $this->request->params['action'];

    // As ações add e index são permitidas sempre.
    if (in_array($action, ['index', 'add', 'tags'])) {
```

```

        return true;
    }
    // Todas as outras ações requerem um id.
    if (empty($this->request->params['pass'][0])) {
        return false;
    }

    // Checa se o bookmark pertence ao user atual.
    $id = $this->request->params['pass'][0];
    $bookmark = $this->Bookmarks->get($id);
    if ($bookmark->user_id == $user['id']) {
        return true;
    }
    return parent::isAuthorized($user);
}

```

Agora, se você tentar visualizar, editar ou excluir um bookmark que não pertença a você, você deve ser redirecionado para a página de onde veio. No entanto, não há nenhuma mensagem de erro sendo exibida, então vamos corrigir isso a seguir:

```

// In src/Template/Layout/default.ctp
// Under the existing flash message.
<?=$this->Flash->render('auth') ?>

```

Agora você deve ver as mensagens de erro de autorização.

## Corrigindo a view de listagem e formulários

Enquanto view e delete estão trabalhando, edit, add e index tem alguns problemas:

1. Ao adicionar um bookmark, você pode escolher o user.
2. Ao editar um bookmark, você pode escolher o user.
3. A página de listagem mostra os bookmarks de outros users.

Vamos enfrentar o formulário de adição em primeiro lugar. Para começar remova o input ('user\_id') a partir de `src/Template/Bookmarks/add.ctp`. Com isso removido, nós também vamos atualizar o método add:

```

public function add()
{
    $bookmark = $this->Bookmarks->newEntity();
    if ($this->request->is('post')) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('The bookmark has been saved.');

```

```
$this->set(compact('bookmark', 'tags'));
}
```

Ao definir a propriedade da entidade com os dados da sessão, nós removemos qualquer possibilidade de user modificar de que outro user um bookmark seja. Nós vamos fazer o mesmo para o formulário edit e action edit. Sua ação edit deve ficar assim:

```
public function edit($id = null)
{
    $bookmark = $this->Bookmarks->get($id, [
        'contain' => ['Tags']
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('The bookmark has been saved.');
```

### View de listagem

Agora, nós precisamos apenas exibir bookmarks para o user logado. Nós podemos fazer isso ao atualizar a chamada para `paginate()`. Altere sua ação `index`:

```
public function index()
{
    $this->paginate = [
        'conditions' => [
            'Bookmarks.user_id' => $this->Auth->user('id'),
        ]
    ];
    $this->set('bookmarks', $this->paginate($this->Bookmarks));
}
```

Nós também devemos atualizar a action `tags()` e o método localizador relacionado, mas vamos deixar isso como um exercício para que você conclua por si.

### Melhorando a experiência com as tags

Agora, adicionar novas tags é um processo difícil, pois o `TagsController` proíbe todos os acessos. Em vez de permitir o acesso, podemos melhorar a interface do usuário para selecionar tags usando um campo de texto separado por vírgulas. Isso permitirá dar uma melhor experiência para os nossos usuários, e usar mais alguns grandes recursos no ORM.

## Adicionando um campo computado

Porque nós queremos uma maneira simples de acessar as tags formatados para uma entidade, podemos adicionar um campo virtual/computado para a entidade. Em `src/Model/Entity/Bookmark.php` adicione o seguinte:

```
use Cake\Collection\Collection;

protected function _getTagString()
{
    if (isset($this->_properties['tag_string'])) {
        return $this->_properties['tag_string'];
    }
    if (empty($this->tags)) {
        return '';
    }
    $tags = new Collection($this->tags);
    $str = $tags->reduce(function ($string, $tag) {
        return $string . $tag->title . ', ';
    }, '');
    return trim($str, ', ');
}
```

Isso vai nos deixar acessar a propriedade computada `$bookmark->tag_string`. Vamos usar essa propriedade em inputs mais tarde. Lembre-se de adicionar a propriedade `tag_string` a lista `_accessible` em sua entidade.

Em `src/Model/Entity/Bookmark.php` adicione o `tag_string` ao `_accessible` desta forma:

```
protected $_accessible = [
    'user_id' => true,
    'title' => true,
    'description' => true,
    'url' => true,
    'user' => true,
    'tags' => true,
    'tag_string' => true,
];
```

## Atualizando as views

Com a entidade atualizado, podemos adicionar uma nova entrada para as nossas tags. Nas views `add` e `edit`, substitua `tags._ids` pelo seguinte:

```
<?= $this->Form->input('tag_string', ['type' => 'text']) ?>
```

## Persistindo a string tag

Agora que podemos ver as tags como uma string existente, vamos querer salvar os dados também. Por marcar o `tag_string` como acessível, o ORM irá copiar os dados do pedido em nossa entidade. Podemos

usar um método `beforeSave` para analisar a cadeia tag e encontrar/construir as entidades relacionadas. Adicione o seguinte em `src/Model/Table/BookmarksTable.php`:

```
public function beforeSave($event, $entity, $options)
{
    if ($entity->tag_string) {
        $entity->tags = $this->_buildTags($entity->tag_string);
    }
}

protected function _buildTags($tagString)
{
    $new = array_unique(array_map('trim', explode(',', $tagString)));
    $out = [];
    $query = $this->Tags->find()
        ->where(['Tags.title IN' => $new]);

    // Remove tags existentes da lista de novas tags.
    foreach ($query->extract('title') as $existing) {
        $index = array_search($existing, $new);
        if ($index !== false) {
            unset($new[$index]);
        }
    }
    // Adiciona tags existentes.
    foreach ($query as $tag) {
        $out[] = $tag;
    }
    // Adiciona novas tags.
    foreach ($new as $tag) {
        $out[] = $this->Tags->newEntity(['title' => $tag]);
    }
    return $out;
}
```

Embora esse código seja um pouco mais complicado do que o que temos feito até agora, ele ajuda a mostrar o quão poderosa a ORM do CakePHP é. Você pode facilmente manipular resultados da consulta usando os métodos de *Collections (Coleções)*, e lidar com situações em que você está criando entidades sob demanda com facilidade.

## Terminando

Nós expandimos nossa aplicação bookmarker para lidar com situações de autenticação e controle de autorização/acesso básico. Nós também adicionamos algumas melhorias agradáveis à UX, aproveitando os recursos FormHelper e ORM.

Obrigado por dispor do seu tempo para explorar o CakePHP. Em seguida, você pode saber mais sobre o *Models (Modelos)*, ou você pode ler os `/topics`.

---

## 3.0 Guia de migração

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

## Tutoriais & Exemplos

---

Nesta seção, você poderá caminhar através de típicas aplicações CakePHP para ver como todas as peças se encaixam.

Como alternativa, você pode preferir visitar o repositório não oficial de plugins para o CakePHP [CakePackages](http://plugins.cakephp.org/)<sup>1</sup> e a [Bakery \(Padaria\)](http://bakery.cakephp.org/)<sup>2</sup> para conhecer aplicações e componentes existentes.

### Tutorial - Criando um Bookmarker - Parte 1

Esse tutorial vai guiar você através da criação de uma simples aplicação de marcação (bookmarker). Para começar, nós vamos instalar o CakePHP, criar nosso banco de dados, e usar as ferramentas que o CakePHP fornece para obter nossa aplicação de pé rápido.

Aqui está o que você vai precisar:

1. Um servidor de banco de dados. Nós vamos usar o servidor MySQL neste tutorial. Você precisa saber o suficiente sobre SQL para criar um banco de dados: O CakePHP vai tomar as rédeas a partir daí. Por nós estarmos usando o MySQL, também certifique-se que você tem a extensão `pdo_mysql` habilitada no PHP.
2. Conhecimento básico sobre PHP.

Vamos começar!

### Instalação do CakePHP

A maneira mais fácil de instalar o CakePHP é usando Composer, um gerenciador de dependências para o PHP. É uma forma simples de instalar o CakePHP a partir de seu terminal ou prompt de comando. Primeiro, você precisa baixar e instalar o Composer. Se você tiver instalada a extensão `cURL` do PHP, execute o seguinte comando:

---

<sup>1</sup><http://plugins.cakephp.org/>

<sup>2</sup><http://bakery.cakephp.org/>

```
curl -s https://getcomposer.org/installer | php
```

Ao invés disso, você também pode baixar o arquivo `composer.phar` do [site<sup>3</sup>](#) oficial.

Em seguida, basta digitar a seguinte linha no seu terminal a partir do diretório onde se localiza o arquivo `composer.phar` para instalar o esqueleto de aplicações do CakePHP no diretório `bookmarker`.

```
php composer.phar create-project --prefer-dist cakephp/app bookmarker
```

A vantagem de usar Composer é que ele irá completar automaticamente um conjunto importante de tarefas, como configurar as permissões de arquivo e criar a sua **config/app.php**.

Há outras maneiras de instalar o CakePHP. Se você não puder ou não quiser usar Composer, veja a seção *Instalação*.

Independentemente de como você baixou o CakePHP, uma vez que sua instalação for concluída, a estrutura dos diretórios deve ficar parecida com o seguinte:

```
/bookmarker
  /bin
  /config
  /logs
  /plugins
  /src
  /tests
  /tmp
  /vendor
  /webroot
  .editorconfig
  .gitignore
  .htaccess
  .travis.yml
  composer.json
  index.php
  phpunit.xml.dist
  README.md
```

Agora pode ser um bom momento para aprender sobre como a estrutura de diretórios do CakePHP funciona: Confira a seção *Estrutura de pastas do CakePHP*.

### Verificando nossa instalação

Podemos checar rapidamente que a nossa instalação está correta, verificando a página inicial padrão. Antes que você possa fazer isso, você vai precisar iniciar o servidor de desenvolvimento:

```
bin/cake server
```

Isto irá iniciar o servidor embutido do PHP na porta 8765. Abra `http://localhost:8765` em seu navegador para ver a página de boas-vindas. Todas as verificações devem estar checadadas corretamente, a não ser a conexão com banco de dados do CakePHP. Se não, você pode precisar instalar extensões do PHP adicionais, ou definir permissões de diretório.

---

<sup>3</sup><https://getcomposer.org/download/>

## Criando o banco de dados

Em seguida, vamos criar o banco de dados para a nossa aplicação. Se você ainda não tiver feito isso, crie um banco de dados vazio para uso nesse tutorial, com um nome de sua escolha, por exemplo, `cake_bookmarks`. Você pode executar o seguinte SQL para criar as tabelas necessárias:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  created DATETIME,
  modified DATETIME
);

CREATE TABLE bookmarks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  title VARCHAR(50),
  description TEXT,
  url TEXT,
  created DATETIME,
  modified DATETIME,
  FOREIGN KEY user_key (user_id) REFERENCES users(id)
);

CREATE TABLE tags (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255),
  created DATETIME,
  modified DATETIME,
  UNIQUE KEY (title)
);

CREATE TABLE bookmarks_tags (
  bookmark_id INT NOT NULL,
  tag_id INT NOT NULL,
  PRIMARY KEY (bookmark_id, tag_id),
  INDEX tag_idx (tag_id, bookmark_id),
  FOREIGN KEY tag_key(tag_id) REFERENCES tags(id),
  FOREIGN KEY bookmark_key(bookmark_id) REFERENCES bookmarks(id)
);
```

Você deve ter notado que a tabela `bookmarks_tags` utilizada uma chave primária composta. O CakePHP suporta chaves primárias compostas quase todos os lugares, tornando mais fácil construir aplicações multi-arrendados.

Os nomes de tabelas e colunas que usamos não foram arbitrários. Usando *convenções de nomenclatura* do CakePHP, podemos alavancar o desenvolvimento e evitar ter de configurar o framework. O CakePHP é flexível o suficiente para acomodar até mesmo esquemas de banco de dados legados inconsistentes, mas aderir às convenções vai lhe poupar tempo.

## Configurando o banco de dados

Em seguida, vamos dizer ao CakePHP onde o nosso banco de dados está como se conectar a ele. Para muitos, esta será a primeira e última vez que você vai precisar configurar qualquer coisa.

A configuração é bem simples: basta alterar os valores do array `Datasources.default` no arquivo **config/app.php** pelos que se aplicam à sua configuração. A amostra completa da gama de configurações pode ser algo como o seguinte:

```
return [
    // Mais configuração acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_bookmarks',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
        ],
    ],
    // Mais configuração abaixo.
];
```

Depois de salvar o seu arquivo **config/app.php**, você deve notar que a mensagem ‘CakePHP is able to connect to the database’ tem uma marca de verificação.

---

**Nota:** Uma cópia do arquivo de configuração padrão do CakePHP é encontrado em **config/app.default.php**.

---

## Gerando o código base

Devido a nosso banco de dados seguir as convenções do CakePHP, podemos usar o *bake console* para gerar rapidamente uma aplicação básica. Em sua linha de comando execute:

```
bin/cake bake all users
bin/cake bake all bookmarks
bin/cake bake all tags
```

Isso irá gerar os controllers, models, views, seus casos de teste correspondentes, e fixtures para os nossos users, bookmarks e tags. Se você parou seu servidor, reinicie-o e vá para <http://localhost:8765/bookmarks>.

Você deverá ver uma aplicação que dá acesso básico, mas funcional a tabelas de banco de dados. Adicione alguns users, bookmarks e tags.

## Adicionando criptografia de senha

Quando você criou seus users, você deve ter notado que as senhas foram armazenadas como texto simples. Isso é muito ruim do ponto de vista da segurança, por isso vamos consertar isso.

Este também é um bom momento para falar sobre a camada de modelo. No CakePHP, separamos os métodos que operam em uma coleção de objetos, e um único objeto em diferentes classes. Métodos que operam na recolha de entidades são colocadas na classe *Table*, enquanto as características pertencentes a um único registro são colocados na classe *Entity*.

Por exemplo, a criptografia de senha é feita no registro individual, por isso vamos implementar esse comportamento no objeto entidade. Dada a circunstância de nós querermos criptografar a senha cada vez que é definida, vamos usar um método modificador/definidor. O CakePHP vai chamar métodos de definição baseados em convenções a qualquer momento que uma propriedade é definida em uma de suas entidades. Vamos adicionar um definidor para a senha. Em **src/Model/Entity/User.php** adicione o seguinte:

```
namespace App\Model\Entity;

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

class User extends Entity
{
    // Code from bake.

    protected function _setPassword($value)
    {
        $hasher = new DefaultPasswordHasher();
        return $hasher->hash($value);
    }
}
```

Agora atualize um dos usuários que você criou anteriormente, se você alterar sua senha, você deve ver um senha criptografada ao invés do valor original nas páginas de lista ou visualização. O CakePHP criptografa senhas com `bcrypt`<sup>4</sup> por padrão. Você também pode usar `sha1` ou `md5` caso venha a trabalhar com um banco de dados existente.

## Recuperando bookmarks com uma tag específica

Agora que estamos armazenando senhas com segurança, podemos construir algumas características mais interessantes em nossa aplicação. Uma vez que você acumulou uma coleção de bookmarks, é útil ser capaz de pesquisar através deles por tag. Em seguida, vamos implementar uma rota, a ação do controller, e um método localizador para pesquisar através de bookmarks por tag.

Idealmente, nós teríamos uma URL que se parece com `http://localhost:8765/bookmarks/tagged/funny/cat`. Isso deveria nos permitir a encontrar todos os bookmarks que têm as tags 'funny', 'cat' e 'gifs'. Antes de podermos implementar isso, vamos adicionar uma nova rota. Em **config/routes.php**, adicione o seguinte na parte superior do arquivo:

<sup>4</sup><http://codahale.com/how-to-safely-store-a-password/>

```
Router::scope(
    '/bookmarks',
    ['controller' => 'Bookmarks'],
    function ($routes) {
        $routes->connect('/tagged/*', ['action' => 'tags']);
    }
);
```

O acima define uma nova “rota” que liga o caminho `/bookmarks/tagged/*`, a `BookmarksController::tags()`. Ao definir rotas, você pode isolar como suas URLs parecerão, de como eles são implementadas. Se fôssemos visitar `http://localhost:8765/bookmarks/tagged`, deveríamos ver uma página de erro informativa do CakePHP. Vamos implementar esse método ausente agora. Em `src/Controller/BookmarksController.php` adicione o seguinte:

```
public function tags()
{
    $tags = $this->request->params['pass'];
    $bookmarks = $this->Bookmarks->find('tagged', [
        'tags' => $tags
    ]);
    $this->set(compact('bookmarks', 'tags'));
}
```

## Criando o método localizador

No CakePHP nós gostamos de manter as nossas ações do controller enxutas, e colocar a maior parte da lógica de nossa aplicação nos modelos. Se você fosse visitar a URL `/bookmarks/tagged` agora, você veria um erro sobre o método `findTagged` não estar implementado ainda, então vamos fazer isso. Em `src/Model/Table/BookmarksTable.php` adicione o seguinte:

```
public function findTagged(Query $query, array $options)
{
    $fields = [
        'Bookmarks.id',
        'Bookmarks.title',
        'Bookmarks.url',
    ];
    return $this->find()
        ->distinct($fields)
        ->matching('Tags', function ($q) use ($options) {
            return $q->where(['Tags.title IN' => $options['tags']]);
        });
}
```

Nós implementamos um método *localizador customizado*. Este é um conceito muito poderoso no CakePHP que lhe permite construir consultas reutilizáveis. Em nossa pesquisa, nós alavancamos o método `matching()` que nos habilita encontrar bookmarks que têm uma tag ‘correspondente’.

## Criando a view

Agora, se você visitar a URL `/bookmarks/tagged`, o CakePHP irá mostrar um erro e deixá-lo saber que você ainda não fez um arquivo view. Em seguida, vamos construir o arquivo view para a nossa ação `tags`. Em `src/Template/Bookmarks/tags.ctp` coloque o seguinte conteúdo:

```
<h1>
    Bookmarks tagged with
    <?= $this->Text->toList($tags) ?>
</h1>

<section>
    <?php foreach ($bookmarks as $bookmark): ?>
        <article>
            <h4><?= $this->Html->link($bookmark->title, $bookmark->url) ?></h4>
            <small><?= h($bookmark->url) ?></small>
            <?= $this->Text->autoParagraph($bookmark->description) ?>
        </article>
    <?php endforeach; ?>
</section>
```

O CakePHP espera que os nossos templates sigam a convenção de nomenclatura onde o nome do template é a versão minúscula e grifada do nome da ação do controller.

Você pode perceber que fomos capazes de utilizar as variáveis `$tags` e `bookmarks` em nossa view. Quando usamos o método `set()` em nosso controller, automaticamente definimos variáveis específicas que devem ser enviadas para a view. A view vai tornar todas as variáveis passadas disponíveis nos templates como variáveis locais.

Em nossa view, usamos alguns dos *helpers* nativos do CakePHP. Helpers são usados para criar lógica reutilizável para a formatação de dados, a criação de HTML ou outra saída da view.

Agora você deve ser capaz de visitar a URL `/bookmarks/tagged/funny` e ver todas os bookmarks com a tag 'funny'.

Até agora, nós criamos uma aplicação básica para gerenciar bookmarks, tags e users. No entanto, todos podem ver as tags de toda a gente. No próximo capítulo, vamos implementar a autenticação e restringir os bookmarks visíveis para somente aqueles que pertencem ao usuário atual.

Agora vá a [Tutorial - Criando um Bookmarker - Parte 2](#) para continuar a construir sua aplicação ou mergulhe na documentação para saber mais sobre o que CakePHP pode fazer por você.

## Tutorial - Criando um Bookmarker - Parte 2

Depois de terminar a *primeira parte deste tutorial*, você deve ter uma aplicação muito básica. Neste capítulo iremos adicionar autenticação e restringir as bookmarks para que cada usuário possa ver/modificar somente aquelas que possuam.

## Adicionando login

No CakePHP, a autenticação é feita por *Components (Componentes)*. 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. Para começar, vamos adicionar o AuthComponent a nossa aplicação. Nós vamos querer muito que cada método exija autenticação, por isso vamos acrescentar o *AuthComponent* em nosso ApplicationController:

```
// Em src/Controller/AppController.php
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
    public function initialize()
    {
        $this->loadComponent('Flash');
        $this->loadComponent('Auth', [
            'authenticate' => [
                'Form' => [
                    'fields' => [
                        'username' => 'email',
                        'password' => 'password'
                    ]
                ]
            ],
            'loginAction' => [
                'controller' => 'Users',
                'action' => 'login'
            ]
        ]);

        // Permite a ação display, assim nosso pages controller
        // continua a funcionar.
        $this->Auth->allow(['display']);
    }
}
```

Acabamos de dizer ao CakePHP que queremos carregar os components Flash e Auth. Além disso, temos a configuração personalizada do AuthComponent, assim a nossa tabela users pode usar email como username. Agora, se você for a qualquer URL, você vai ser chutado para /users/login, que irá mostrar uma página de erro já que não escrevemos o código ainda. Então, vamos criar a ação de login:

```
// Em src/Controller/UsersController.php

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.');
```

E em `src/Template/Users/login.ctp` adicione o seguinte:

```

<h1>Login</h1>
<?= $this->Form->create() ?>
<?= $this->Form->input('email') ?>
<?= $this->Form->input('password') ?>
<?= $this->Form->button('Login') ?>
<?= $this->Form->end() ?>
```

Agora que temos um simples formulário de login, devemos ser capazes de efetuar login com um dos users que tenham senha criptografada.

---

**Nota:** Se nenhum de seus users tem senha criptografada, comente a linha `loadComponent('Auth')`. Então vá e edite o user, salvando uma nova senha para ele.

---

Agora você deve ser capaz de entrar. Se não, certifique-se que você está usando um user que tenha senha criptografada.

## Adicionando logout

Agora que as pessoas podem efetuar o login, você provavelmente vai querer fornecer uma maneira de encerrar a sessão também. Mais uma vez, no `UsersController`, adicione o seguinte código:

```

public function logout()
{
    $this->Flash->success('You are now logged out.');
```

Agora você pode visitar `/users/logout` para sair e ser enviado à página de login.

## Ativando inscrições

Se você não estiver logado e tentar visitar `/usuários/adicionar` você vai ser expulso para a página de login. Devemos corrigir isso se quisermos que as pessoas se inscrevam em nossa aplicação. No `UsersController` adicione o seguinte:

```

public function beforeFilter(\Cake\Event\Event $event)
{
    $this->Auth->allow(['add']);
}
```

O texto acima diz ao `AuthComponent` que a ação `add` não requer autenticação ou autorização. Você pode querer dedicar algum tempo para limpar a `/users/add` e remover os links enganosos, ou continuar para a

próxima seção. Nós não estaremos construindo a edição do usuário, visualização ou listagem neste tutorial, então eles não vão funcionar, já que o `AuthComponent` vai negar-lhe acesso a essas ações do controller.

## Restringindo acesso

Agora que os usuários podem conectar-se, nós vamos querer limitar os bookmarks que podem ver para aqueles que fizeram. Nós vamos fazer isso usando um adaptador de ‘autorização’. Sendo os nossos requisitos bastante simples, podemos escrever um código em nossa `BookmarksController`. Mas antes de fazer isso, vamos querer dizer ao `AuthComponent` como nossa aplicação vai autorizar ações. Em seu `AppController` adicione o seguinte:

```
public function isAuthorized($user)
{
    return false;
}
```

Além disso, adicione o seguinte à configuração para Auth em seu `AppController`:

```
'authorize' => 'Controller',
```

Seu método `initialize` agora deve parecer com:

```
public function initialize()
{
    $this->loadComponent('Flash');
    $this->loadComponent('Auth', [
        'authorize' => 'Controller', //added this line
        'authenticate' => [
            'Form' => [
                'fields' => [
                    'username' => 'email',
                    'password' => 'password'
                ]
            ]
        ],
        'loginAction' => [
            'controller' => 'Users',
            'action' => 'login'
        ],
        'unauthorizedRedirect' => $this->referer()
    ]);

    // Permite a ação display, assim nosso pages controller
    // continua a funcionar.
    $this->Auth->allow(['display']);
}
```

Vamos usar como padrão, negação do acesso, e de forma incremental conceder acesso onde faça sentido. Primeiro, vamos adicionar a lógica de autorização para os bookmarks. Em seu `BookmarksController` adicione o seguinte:

```

public function isAuthorized($user)
{
    $action = $this->request->params['action'];

    // As ações add e index são permitidas sempre.
    if (in_array($action, ['index', 'add', 'tags'])) {
        return true;
    }
    // Todas as outras ações requerem um id.
    if (empty($this->request->params['pass'][0])) {
        return false;
    }

    // Checa se o bookmark pertence ao user atual.
    $id = $this->request->params['pass'][0];
    $bookmark = $this->Bookmarks->get($id);
    if ($bookmark->user_id == $user['id']) {
        return true;
    }
    return parent::isAuthorized($user);
}

```

Agora, se você tentar visualizar, editar ou excluir um bookmark que não pertença a você, você deve ser redirecionado para a página de onde veio. No entanto, não há nenhuma mensagem de erro sendo exibida, então vamos corrigir isso a seguir:

```

// In src/Template/Layout/default.ctp
// Under the existing flash message.
<?=$this->Flash->render('auth') ?>

```

Agora você deve ver as mensagens de erro de autorização.

## Corrigindo a view de listagem e formulários

Enquanto view e delete estão trabalhando, edit, add e index tem alguns problemas:

1. Ao adicionar um bookmark, você pode escolher o user.
2. Ao editar um bookmark, você pode escolher o user.
3. A página de listagem mostra os bookmarks de outros users.

Vamos enfrentar o formulário de adição em primeiro lugar. Para começar remova o input ('user\_id') a partir de `src/Template/Bookmarks/add.ctp`. Com isso removido, nós também vamos atualizar o método add:

```

public function add()
{
    $bookmark = $this->Bookmarks->newEntity();
    if ($this->request->is('post')) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
        if ($this->Bookmarks->save($bookmark)) {

```

```

        $this->Flash->success('The bookmark has been saved.');
```

```

        return $this->redirect(['action' => 'index']);
    }
    $this->Flash->error('The bookmark could not be saved. Please, try again.');
```

```

}
$tags = $this->Bookmarks->Tags->find('list');
```

```

$this->set(compact('bookmark', 'tags'));
}

```

Ao definir a propriedade da entidade com os dados da sessão, nós removemos qualquer possibilidade do user modificar de que outro user um bookmark seja. Nós vamos fazer o mesmo para o formulário edit e action edit. Sua ação edit deve ficar assim:

```

public function edit($id = null)
{
    $bookmark = $this->Bookmarks->get($id, [
        'contain' => ['Tags']
    ]);
    if ($this->request->is(['patch', 'post', 'put'])) {
        $bookmark = $this->Bookmarks->patchEntity($bookmark, $this->request->data);
        $bookmark->user_id = $this->Auth->user('id');
```

```

        if ($this->Bookmarks->save($bookmark)) {
            $this->Flash->success('The bookmark has been saved.');
```

```

            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error('The bookmark could not be saved. Please, try again.');
```

```

    }
    $tags = $this->Bookmarks->Tags->find('list');
```

```

    $this->set(compact('bookmark', 'tags'));
}

```

## View de listagem

Agora, nós precisamos apenas exibir bookmarks para o user logado. Nós podemos fazer isso ao atualizar a chamada para `paginate()`. Altere sua ação `index`:

```

public function index()
{
    $this->paginate = [
        'conditions' => [
            'Bookmarks.user_id' => $this->Auth->user('id'),
        ]
    ];
    $this->set('bookmarks', $this->paginate($this->Bookmarks));
}

```

Nós também devemos atualizar a action `tags()` e o método localizador relacionado, mas vamos deixar isso como um exercício para que você conclua por si.

## Melhorando a experiência com as tags

Agora, adicionar novas tags é um processo difícil, pois o `TagsController` proíbe todos os acessos. Em vez de permitir o acesso, podemos melhorar a interface do usuário para selecionar tags usando um campo de texto separado por vírgulas. Isso permitirá dar uma melhor experiência para os nossos usuários, e usar mais alguns grandes recursos no ORM.

### Adicionando um campo computado

Porque nós queremos uma maneira simples de acessar as tags formatados para uma entidade, podemos adicionar um campo virtual/computado para a entidade. Em `src/Model/Entity/Bookmark.php` adicione o seguinte:

```
use Cake\Collection\Collection;

protected function _getTagString()
{
    if (isset($this->_properties['tag_string'])) {
        return $this->_properties['tag_string'];
    }
    if (empty($this->tags)) {
        return '';
    }
    $tags = new Collection($this->tags);
    $str = $tags->reduce(function ($string, $tag) {
        return $string . $tag->title . ', ';
    }, '');
    return trim($str, ', ');
}
```

Isso vai nos deixar acessar a propriedade computada `$bookmark->tag_string`. Vamos usar essa propriedade em inputs mais tarde. Lembre-se de adicionar a propriedade `tag_string` a lista `_accessible` em sua entidade.

Em `src/Model/Entity/Bookmark.php` adicione o `tag_string` ao `_accessible` desta forma:

```
protected $_accessible = [
    'user_id' => true,
    'title' => true,
    'description' => true,
    'url' => true,
    'user' => true,
    'tags' => true,
    'tag_string' => true,
];
```

### Atualizando as views

Com a entidade atualizado, podemos adicionar uma nova entrada para as nossas tags. Nas views `add` e `edit`, substitua `tags._ids` pelo seguinte:

```
<?= $this->Form->input('tag_string', ['type' => 'text']) ?>
```

## Persistindo a string tag

Agora que podemos ver as tags como uma string existente, vamos querer salvar os dados também. Por marcar o `tag_string` como acessível, o ORM irá copiar os dados do pedido em nossa entidade. Podemos usar um método `beforeSave` para analisar a cadeia tag e encontrar/construir as entidades relacionadas. Adicione o seguinte em `src/Model/Table/BookmarksTable.php`:

```
public function beforeSave($event, $entity, $options)
{
    if ($entity->tag_string) {
        $entity->tags = $this->_buildTags($entity->tag_string);
    }
}

protected function _buildTags($tagString)
{
    $new = array_unique(array_map('trim', explode(',', $tagString)));
    $out = [];
    $query = $this->Tags->find()
        ->where(['Tags.title IN' => $new]);

    // Remove tags existentes da lista de novas tags.
    foreach ($query->extract('title') as $existing) {
        $index = array_search($existing, $new);
        if ($index !== false) {
            unset($new[$index]);
        }
    }
    // Adiciona tags existentes.
    foreach ($query as $tag) {
        $out[] = $tag;
    }
    // Adiciona novas tags.
    foreach ($new as $tag) {
        $out[] = $this->Tags->newEntity(['title' => $tag]);
    }
    return $out;
}
```

Embora esse código seja um pouco mais complicado do que o que temos feito até agora, ele ajuda a mostrar o quão poderosa a ORM do CakePHP é. Você pode facilmente manipular resultados da consulta usando os métodos de *Collections (Coleções)*, e lidar com situações em que você está criando entidades sob demanda com facilidade.

## Terminando

Nós expandimos nossa aplicação bookmarker para lidar com situações de autenticação e controle de autorização/acesso básico. Nós também adicionamos algumas melhorias agradáveis à UX, aproveitando os

recursos FormHelper e ORM.

Obrigado por dispor do seu tempo para explorar o CakePHP. Em seguida, você pode saber mais sobre o *Models (Modelos)*, ou você pode ler os `/topics`.

## Tutorial - Criando um Blog - Parte 1

Este tutorial irá orientá-lo através da criação de um simples blog. Faremos a instalação do CakePHP, criaremos um banco de dados e implementaremos a lógica capaz de listar, adicionar, editar e apagar postagens do blog.

Aqui está o que você vai precisar:

1. Um servidor web em funcionamento. Nós iremos assumir que você esteja usando o Apache, embora as instruções para outros servidores sejam bem similares. Talvez seja preciso alterar um pouco a configuração do servidor, mas a maioria das pessoas pode ter o CakePHP instalado e funcionando sem qualquer trabalho extra. Certifique-se de que você tem o PHP 5.4.16 ou superior, e que as extensões *mbstring* e *intl* estejam habilitadas no PHP. Caso não saiba a versão do PHP que está instalada, utilize a função `phpinfo()` ou digite `php -v` no seu terminal de comando.
2. Um servidor de banco de dados. Nós vamos usar o servidor *MySQL* neste tutorial. Você precisa saber o mínimo sobre SQL para então criar um banco de dados, depois disso o CakePHP vai assumir as rédeas. Já que usaremos o *MySQL*, também certifique-se que a extensão `pdo_mysql` está habilitada no PHP.
3. Conhecimento básico sobre PHP.

Vamos começar!

### Instalação do CakePHP

A maneira mais fácil de instalar o CakePHP é usando Composer, um gerenciador de dependências para o PHP. Se trata de uma forma simples de instalar o CakePHP a partir de seu terminal ou prompt de comando. Primeiro, você precisa baixar e instalar o Composer. Se possuir instalada a extensão *cURL* do PHP, execute o seguinte comando:

```
curl -s https://getcomposer.org/installer | php
```

Você também pode baixar o arquivo `composer.phar` do [site](https://getcomposer.org/)<sup>5</sup> oficial do Composer.

Em seguida, basta digitar a seguinte linha de comando no seu terminal a partir do diretório onde se localiza o arquivo `composer.phar` para instalar o esqueleto da aplicação do CakePHP no diretório `[nome_do_app]`.

```
php composer.phar create-project --prefer-dist cakephp/app [nome_do_app]
```

A vantagem de usar o Composer é que ele irá completar automaticamente um conjunto importante de tarefas, como configurar corretamente as permissões de pastas e criar o `config/app.php` para você.

---

<sup>5</sup><https://getcomposer.org/download/>

Há outras maneiras de instalar o CakePHP. Se você não puder ou não quiser usar o Composer, confira a seção [Instalação](#).

Independentemente de como você baixou o CakePHP, uma vez que sua instalação for concluída, a estrutura dos diretórios deve ficar parecida com o seguinte:

```
/nome_do_app
  /bin
  /config
  /logs
  /plugins
  /src
  /tests
  /tmp
  /vendor
  /webroot
  .editorconfig
  .gitignore
  .htaccess
  .travis.yml
  composer.json
  index.php
  phpunit.xml.dist
  README.md
```

Agora pode ser um bom momento para aprender sobre como a estrutura de diretórios do CakePHP funciona: Confira a seção [Estrutura de pastas do CakePHP](#).

### Permissões dos diretórios tmp e logs

Os diretórios **tmp** e **logs** precisam ter permissões adequadas para que possam ser alterados pelo seu servidor web. Se você usou o Composer na instalação, ele deve ter feito isso por você e confirmado com uma mensagem “Permissions set on <folder>”. Se você ao invés disso, recebeu uma mensagem de erro ou se quiser fazê-lo manualmente, a melhor forma seria descobrir por qual usuário o seu servidor web é executado (`<? = 'whoami' ; ?>`) e alterar o proprietário desses dois diretórios para este usuário. Os comandos finais a serem executados (em \*nix) podem ser algo como:

```
chown -R www-data tmp
chown -R www-data logs
```

Se por alguma razão o CakePHP não puder escrever nesses diretórios, você será informado por uma advertência enquanto não estiver em modo de produção.

Embora não seja recomendado, se você é incapaz de redefinir as permissões do seu servidor web, você pode simplesmente alterar as permissões de gravação diretamente nos diretórios, executando os seguintes comandos:

```
chmod 777 -R tmp
chmod 777 -R logs
```



## Criando o banco de dados do Blog

Em seguida, vamos configurar o banco de dados para o nosso blog. Se você ainda não tiver feito isto, crie um banco de dados vazio para usar neste tutorial, com um nome de sua escolha, por exemplo, `cake_blog`. Agora, vamos criar uma tabela para armazenar nossos artigos:

```
/* 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
);
```

Nós vamos também inserir alguns artigos para usarmos em nossos testes. Execute os seguintes comandos SQL em seu banco de dados:

```
/* 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());
```

Os nomes de tabelas e colunas que usamos não foram arbitrários. Usando *convenções de nomenclatura* do CakePHP, podemos alavancar o desenvolvimento e acelerar a configuração do framework. O CakePHP é flexível o suficiente para acomodar até mesmo esquemas de banco de dados legados inconsistentes, mas aderir às convenções vai lhe poupar tempo.

## Configurando o banco de dados do Blog

Em seguida, vamos dizer ao CakePHP onde nosso banco de dados está e como se conectar a ele. Para muitos, esta será a primeira e última vez que será necessário configurar algo.

A configuração é bem simples e objetiva: basta alterar os valores no array `Datasources.default` localizado no arquivo `config/app.php`, pelos valores que se aplicam à sua configuração. Um exemplo completo de configurações deve se parecer como o seguinte:

```
return [
    // Mais configurações acima.
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            'username' => 'cakephp',
            'password' => 'AngelF00dC4k3~',
            'database' => 'cake_blog',
            'encoding' => 'utf8',
```

```
        'timezone' => 'UTC',
        'cacheMetadata' => true,
    ],
],
// Mais configurações abaixo.
];
```

Depois de salvar o arquivo **config/app.php**, você deve notar a mensagem *CakePHP is able to connect to the database* ao acessar o Blog pelo seu navegador.

---

**Nota:** Uma cópia do arquivo de configuração padrão do CakePHP pode ser encontrada em **config/app.default.php**.

---

## Configurações opcionais

Há alguns outros itens que podem ser configurados. Muitos desenvolvedores completam esta lista de itens, mas os mesmos não são obrigatórios para este tutorial. Um deles é definir uma sequência personalizada (ou “salt”) para uso em hashes de segurança.

A sequência personalizada (ou salt) é utilizada para gerar hashes de segurança. Se você utilizou o Composer, ele cuidou disso para você durante a instalação. Apesar disso, você precisa alterar a sequência personalizada padrão editando o arquivo **config/app.php**. Não importa qual será o novo valor, somente deverá ser algo difícil de descobrir:

```
'Security' => [
    'salt' => 'algum valor longo contendo uma mistura aleatória de valores.',
],
```

## Observação sobre o mod\_rewrite

Ocasionalmente, novos usuários irão se atralhar com problemas de mod\_rewrite. Por exemplo, se a página de boas vindas do CakePHP parecer estranha (sem imagens ou estilos CSS). Isto provavelmente significa que o mod\_rewrite não está funcionando em seu servidor. Por favor, verifique a seção *Reescrita de URL* para obter ajuda e resolver qualquer problema relacionado.

Agora continue o tutorial em *Tutorial - Criando um Blog - Parte 2* e inicie a construção do seu Blog com o CakePHP.

## Tutorial - Criando um Blog - Parte 2

### Criando o 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`.

O arquivo que criaremos deverá ficar salvo em `src/Model/Table/ArticlesTable.php`:

```
// 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');
    }
}
```

Convenções de nomenclatura são muito importantes no CakePHP. Ao nomear nosso objeto como `ArticlesTable`, o CakePHP automaticamente deduz que o mesmo utilize o `ArticlesController` e seja relacionado à tabela `articles`.

**Nota:** O CakePHP criará automaticamente um objeto model se não puder encontrar um arquivo correspondente em `src/Model/Table`. Se você nomear incorretamente seu arquivo (isto é, `articlestable.php` ou `ArticleTable.php`), o CakePHP não reconhecerá suas definições e usará o model gerado como alternativa.

Para mais informações sobre models, como callbacks e validação, visite o capítulo *Models (Modelos)* do manual.

**Nota:** Se você completou a *primeira parte* do tutorial e criou a tabela `articles`, você pode tomar proveito da capacidade de geração de código do `bake` através do console do CakePHP para criar o model `ArticlesTable`:

```
bin/cake bake model Articles
```

Para mais informações sobre o `bake` e suas características relacionadas a geração de código, visite o capítulo *Geração de código com o Bake* do manual.

## Criando o controller

A seguir, criaremos um controller (controlador) para nossos artigos. O controller é responsável pela lógica de interação da aplicação. É o lugar onde você utilizará as regras contidas nos models e executará tarefas relacionadas aos artigos. Criaremos um arquivo chamado `ArticlesController.php` no diretório `src/Controller`:

```
// src/Controller/ArticlesController.php

namespace App\Controller;
```

```
class ArticlesController extends AppController
{
}
```

Agora, vamos adicionar uma action (ação) ao nosso controller. Actions frequentemente, representam uma função ou interface em uma aplicação. Por exemplo, quando os usuários requisitarem `www.example.com/articles/index` (sendo o mesmo que `www.example.com/articles/`), eles esperam ver uma lista de artigos:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

class ArticlesController extends AppController
{
    public function index()
    {
        $articles = $this->Articles->find('all');
        $this->set(compact('articles'));
    }
}
```

Ao definir a função `index()` em nosso `ArticlesController`, os usuários podem acessá-la requisitando `www.example.com/articles/index`. Similarmente, se definíssemos uma função chamada `foobar()`, os usuários poderiam acessá-la em `www.example.com/articles/foobar`.

**Aviso:** Vocês podem ser tentados a nomear seus controllers e actions para obter uma certa URL. Resista a essa tentação. Siga as *Convenções do CakePHP* e crie nomes de action legíveis e compreensíveis. Você pode mapear URLs para o seu código utilizando *Roteamento*.

A instrução na action usa `set()` para passar dados do controller para a view. A variável é definida como `'articles'`, sendo igual ao valor retornado do método `find('all')` do objeto `ArticlesTable`.

**Nota:** Se você completou a *primeira parte* do tutorial e criou a tabela `articles`, você pode tomar proveito da capacidade de geração de código do `bake` através do console do CakePHP para criar o controller `ArticlesController`:

```
bin/cake bake controller Articles
```

Para mais informações sobre o `bake` e suas características sobre geração de código, visite o capítulo *Geração de código com o Bake* do manual.

## Criando as views

Agora que nós temos os dados fluindo pelo nosso model, e nossa lógica da aplicação definida em nosso controller, vamos criar uma view (visualização) para a action `index()`.

As views do CakePHP são camadas de apresentação que se encaixam nos layouts da aplicação. Para a maioria das aplicações, elas são uma mescla entre HTML e PHP, mas também podem ser distribuídas como

XML, CSV, ou ainda dados binários.

Um layout é um conjunto de códigos encontrado ao redor das views. Múltiplos layouts podem ser definidos, e você pode alterar entre eles, mas agora, vamos usar o default, localizado em **src/Template/Layout/default.ctp**.

Lembra que na última sessão atribuímos a variável 'articles' à view usando o método `set()`? Isso levará a coleção de objetos gerada pela query a ser invocada numa iteração `foreach`.

Arquivos de template do CakePHP são armazenados em **src/Template** dentro de uma pasta com o nome do controller correspondente (nós teremos que criar a pasta 'Articles' nesse caso). Para distribuir os dados de artigos em uma tabela, precisamos criar uma view assim:

```
<!-- File: src/Template/Articles/index.ctp -->

<h1>Blog articles</h1>
<table>
  <tr>
    <th>Id</th>
    <th>Title</th>
    <th>Created</th>
  </tr>

  <!-- Aqui é onde iremos iterar nosso objeto de solicitação $articles, exibindo informa

  <?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>
  </tr>
  <?php endforeach; ?>
</table>
```

Você deve ter notado o uso de um objeto chamado `$this->Html`, uma instância da classe `Cake\View\Helper\HtmlHelper` do CakePHP. O CakePHP vem com um conjunto de view helpers que simplificam tarefas como gerar links e formulários. Você pode aprender como usá-los em *Helpers (Facilitadores)*, mas aqui é importante notar que o método `link()` irá gerar um link HTML com o referido título (primeiro parâmetro) e URL (segundo parâmetro).

Quando se especifica URLs no CakePHP, é recomendado o uso do formato de array. Isto será melhor explicado posteriormente na seção Rotas. Usando o formato de array para URLs, você toma vantagem das capacidades de roteamento reverso do CakePHP. Você também pode especificar URLs relativas a base da aplicação com o formato `/controller/action/param1/param2` ou usar *named routes*.

Neste ponto, você pode visitar <http://www.example.com/articles/index> no seu navegador. Você deve ver sua view corretamente formatada listando os artigos.

Se você clicar no link do título de um artigo listado, provavelmente será informado pelo CakePHP que a action ainda não foi definida, então vamos criá-la no `ArticlesController` agora:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

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'));
    }
}
```

O uso do `set()` deve parecer familiar. Repare que você está usando `get()` ao invés de `find('all')` porque nós queremos a informação de apenas um artigo.

Repare que nossa action recebe um parâmetro: o ID do artigo que gostaríamos de visualizar. Esse parâmetro é entregue para a action através da URL solicitada. Se o usuário requisitar `/articles/view/3`, então o valor `'3'` é passado como `$id` para a action.

Ao usar a função `get()`, fazemos também algumas verificações para garantir que o usuário realmente está acessando um registro existente, se não ou se o `$id` for indefinido, a função irá lançar uma `NotFoundException`.

Agora vamos criar a view para nossa action em `src/Template/Articles/view.ctp`

```
<!-- File: src/Template/Articles/view.ctp -->

<h1><?= h($article->title) ?></h1>
<p><?= h($article->body) ?></p>
<p><small>Criado: <?= $article->created->format (DATE_RFC850) ?></small></p>
```

Verifique se está tudo funcionando acessando os links em `/articles/index` ou manualmente solicite a visualização um artigo acessando `articles/view/{id}`. Lembre-se de substituir `{id}` por um `'id'` de um artigo.

## Adicionando artigos

Primeiro, comece criando a action `add()` no `ArticlesController`:

```
// src/Controller/ArticlesController.php

namespace App\Controller;

use App\Controller\AppController;
```

```

class ArticlesController extends AppController
{
    public function initialize()
    {
        parent::initialize();

        $this->loadComponent('Flash'); // Inclui o FlashComponent
    }

    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, $this->request->data);
            if ($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.));
        }
        $this->set('article', $article);
    }
}

```

---

**Nota:** Você precisa incluir o *Flash* component em qualquer controller que vá usá-lo. Se necessário, inclua no `AppController` e assim o `FlashComponent` estará disponível para todos os controllers da aplicação.

---

A action `add()` checa se o método HTTP da solicitação foi POST, e então tenta salvar os dados utilizando o model `Articles`. Se por alguma razão ele não salvar, apenas renderiza a view. Isto nos dá a chance de exibir erros de validação ou outros alertas.

Cada requisição do CakePHP instancia um objeto `Request` que é acessível usando `$this->request`. O objeto contém informações úteis sobre a requisição que foi recebida e pode ser usado para controlar o fluxo de sua aplicação. Nesse caso, nós usamos o método `Cake\Network\Request::is()` para checar se a requisição é do tipo HTTP POST.

Quando se usa um formulário para postar dados, essa informação fica disponível em `$this->request->data`. Você pode usar as funções `pr()` ou `debug()` caso queira verificar esses dados.

Usamos os métodos `success()` e `error()` do `FlashComponent` para definir uma mensagem que será armazenada numa variável de sessão. Esses métodos são gerados usando os [recursos de métodos mágicos](#)<sup>6</sup> do PHP. Mensagens flash serão exibidas na página após um redirecionamento. No layout nós temos `<?= $this->Flash->render() ?>` que exibe a mensagem e limpa a variável de sessão. A função do controller `Cake\Controller\Controller::redirect` redireciona para qualquer outra URL. O parâmetro `['action' => 'index']` corresponde a URL `/articles`, isto é, a action `index()` do `ArticlesController`. Você pode consultar a função `Cake\Routing\Router::url()` na [API](#)<sup>7</sup> e checar os formatos a partir dos quais você pode montar uma URL.

Chamar o método `save()` vai checar erros de validação e abortar o processo caso os encontre. Nós vamos abordar como esses erros são tratados nas sessões a seguir.

### Validando artigos

O CakePHP torna mais prática e menos monótona a validação de dados de formulário.

Para tirar proveito dos recursos de validação, você vai precisar usar o *Form* helper em suas views. O `Cake\View\Helper\FormHelper` está disponível por padrão em todas as views pelo uso do `$this->Form`.

Segue a view correspondente a action `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();
?>
```

Nós usamos o `FormHelper` para gerar a tag de abertura HTML de um formulário. Segue o HTML gerado por `$this->Form->create()`:

```
.. code-block:: html
```

```
<form method="post" action="/articles/add">
```

Se `create()` é chamado sem parâmetros fornecidos, assume-se a construção de um formulário que submete dados via POST para a action `add()` (ou `edit()` no caso de um `id` estar incluído nos dados do formulário).

O método `$this->Form->input()` é usado para criar elementos do formulário do mesmo nome. O primeiro parâmetro diz ao CakePHP qual é o campo correspondente, e o segundo parâmetro permite que você especifique um vasto array de opções, nesse, o número de linhas para o textarea. `input()` vai gerar diferentes elementos de formulários baseados no tipo de campo especificado no model.

---

<sup>6</sup><http://php.net/manual/en/language.oop5.overloading.php#object.call>

<sup>7</sup><http://api.cakephp.org>



O `$this->Form->end()` fecha o formulário, entregando também elementos ocultos caso a prevenção contra CSRF/Form Tampering esteja habilitada.

Agora vamos voltar e atualizar nossa view `src/Template/Articles/index.ctp` para incluir um novo link. Antes do `<table>`, adicione a seguinte linha:

```
<?= $this->Html->link('Adicionar artigo', ['action' => 'add']) ?>
```

Você deve estar se perguntando: como eu digo ao CakePHP meus critérios de validação? Regras de validação são definidas no model. Vamos fazer alguns ajustes no nosso model:

```
// src/Model/Table/ArticlesTable.php

namespace App\Model\Table;

use Cake\ORM\Table;
use Cake\Validation\Validator;

class ArticlesTable extends Table
{
    public function initialize(array $config)
    {
        $this->addBehavior('Timestamp');
    }

    public function validationDefault(Validator $validator)
    {
        $validator
            ->notEmpty('title')
            ->notEmpty('body');

        return $validator;
    }
}
```

O método `validationDefault()` diz ao CakePHP como validar seus dados quando o método `save()` for solicitado. Aqui, estamos especificando que tanto o campo `body` quanto `title` não devem estar vazios. O CakePHP possui muitos recursos de validação e um bom número de regras pré-determinadas (número de cartões, endereços de email, etc), além de flexibilidade para adicionar regras de validação customizadas. Para mais informações sobre configuração de validações, visite a documentação em [Validação](#).

Agora que suas regras de validação estão definidas, tente adicionar um artigo sem definir o campo `title` e `body` para ver como a validação funciona. Desde que tenhamos usado o método `Cake\View\Helper\FormHelper::input()` do `FormHelper` para criar nossos elementos, nossas mensagens de alerta da validação serão exibidas automaticamente.

## Editando artigos

Edição, aí vamos nós! Você já é um profissional do CakePHP agora, então possivelmente detectou um padrão... Cria-se a action e então a view. Aqui segue a action `edit()` que deverá ser inserida no `ArticlesController`:

```
// src/Controller/ArticlesController.php

public function edit($id = null)
{
    $article = $this->Articles->get($id);
    if ($this->request->is(['post', 'put'])) {
        $this->Articles->patchEntity($article, $this->request->data);
        if ($this->Articles->save($article)) {
            $this->Flash->success(__('Seu artigo foi atualizado.));
            return $this->redirect(['action' => 'index']);
        }
        $this->Flash->error(__('Seu artigo não pôde ser atualizado.));
    }

    $this->set('article', $article);
}

```

Essa action primeiramente certifica-se que o registro apontado existe. Se o parâmetro `$id` não foi passado ou se o registro é inexistente, uma `NotFoundException` é lançada pelo `ErrorHandler` do CakePHP.

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()`. Então finalmente usamos o `ArticlesTable` para salvar a entidade.

Segue a view correspondente a action edit:

```
<!-- File: src/Template/Articles/edit.ctp -->

<h1>Edit 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();
?>

```

Essa view retorna o formulário de edição com os dados populadas, juntamente com qualquer mensagem de erro proveniente de validações.

O CakePHP irá determinar se o `save()` vai inserir ou atualizar um registro baseado nos dados da entidade.

Você pode atualizar sua view index com os links para editar artigos:

```
<!-- File: src/Template/Articles/index.ctp (edit links added) -->

<h1>Blog articles</h1>
<p><?= $this->Html->link("Adicionar artigo", ['action' => 'add']) ?></p>
<table>
    <tr>
        <th>Id</th>
        <th>Título</th>
        <th>Criado</th>
        <th>Ações</th>
    </tr>

```

```

</tr>

<!-- Aqui é onde iremos iterar nosso objeto de solicitação $articles, exibindo informações

<?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->Html->link('Editar', ['action' => 'edit', $article->id]) ?>
    </td>
  </tr>
<?php endforeach; ?>

</table>

```

## Deletando artigos

A seguir, vamos criar uma forma de deletar artigos. Comece com uma action `delete()` no `ArticlesController`:

```

// src/Controller/ArticlesController.php

public function delete($id)
{
    $this->request->allowMethod(['post', 'delete']);

    $article = $this->Articles->get($id);
    if ($this->Articles->delete($article)) {
        $this->Flash->success(__('O artigo com id: {0} foi deletado.', h($id)));
        return $this->redirect(['action' => 'index']);
    }
}

```

Essa lógica deleta o artigo especificado pelo `$id` e usa `$this->Flash->success()` para exibir uma mensagem de confirmação após o redirecionamento para `/articles`. Tentar excluir um registro usando uma requisição GET, fará com que o `allowMethod()` lance uma exceção. Exceções são capturadas pelo gerenciador de exceções do CakePHP e uma página de erro é exibida. Existem muitos *Exceptions* embutidos que podem indicar variados erros HTTP que sua aplicação possa precisar.

Por estarmos executando apenas lógica e redirecionando, essa action não tem uma view. Vamos atualizar nossa view index com links para excluir artigos:

```

<!-- File: src/Template/Articles/index.ctp (delete links added) -->

<h1>Blog articles</h1>
<p><?= $this->Html->link('Adicionar artigo', ['action' => 'add']) ?></p>

```

```

<table>
  <tr>
    <th>Id</th>
    <th>Título</th>
    <th>Criado</th>
    <th>Ações</th>
  </tr>

  <!-- Aqui é onde iremos iterar nosso objeto de solicitação $articles, exibindo informa

  <?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 (
        'Deletar',
        ['action' => 'delete', $article->id],
        ['confirm' => 'Tem certeza?'])
      ?>
      <?= $this->Html->link('Edit', ['action' => 'edit', $article->id]) ?>
    </td>
  </tr>
  <?php endforeach; ?>

</table>

```

Usar `View\Helper\FormHelper::postLink()` vai criar um link que usa JavaScript para criar uma requisição POST afim de deletar um artigo.

**Aviso:** Permitir que registros sejam deletados usando requisições GET é perigoso, pois rastreadores na web podem acidentalmente deletar todo o seu conteúdo.

**Nota:** Esse código da view também usa o `FormHelper` para confirmar a action através de JavaScript.

## Rotas

Para muitos o roteamento padrão do CakePHP funciona bem o suficiente. Desenvolvedores que consideram facilidade de uso e SEO irão apreciar a forma como o CakePHP mapeia determinadas URLs para actions específicas. Vamos realizar uma pequena mudança nas rotas neste tutorial.

Para mais informações sobre técnicas avançadas de roteamento, visite *routes-configuration*.

Por padrão, o CakePHP responde a uma requisição pela raiz do seu site usando o `PagesController`, ao renderizar uma view chamada **home.ctp**. Alternativamente, nós vamos substituir esse comportamento pelo

ArticlesController ao criar uma regra de roteamento.

A configuração de rotas do CakePHP pode ser encontrada em **config/routes.php**. Você deve comentar ou remover a linha que define o roteamento padrão:

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

Essa linha conecta a URL '/' com a página padrão do CakePHP. Nós queremos que ela conecte-se ao nosso próprio controller, então a substitua por esta:

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

Isso irá conectar requisições por '/' a action `index()` do nosso ArticlesController

---

**Nota:** O CakePHP aproveita-se do uso de roteamento reverso. Se com a rota anterior definida você gerar um link com a seguinte estrutura de array: `['controller' => 'Articles', 'action' => 'index']`, a URL resultante será '/'. Portanto, é uma boa ideia sempre usar arrays para URLs, pois assim suas rotas definem o endereço gerado e certificam-se que os links apontem sempre para o mesmo lugar.

---

## Conclusão

Simple, não é? Tenha em mente que esse foi um tutorial básico. O CakePHP tem  *muito*  mais recursos a oferecer. Não abordamos outros tópicos aqui para manter a simplicidade. Use o restante do manual como um guia para criar aplicações mais ricas.

Agora que você criou uma aplicação básica no CakePHP, você pode continuar no *Tutorial - Criando um Blog - Parte 3*, ou começar seu próprio projeto. Você também pode folhear os `/topics` ou a *API* <<http://api.cakephp.org/3.0>> para aprender mais sobre o CakePHP.

Se você precisar de ajuda, há muitas formas de conseguir, por favor, visite a página *Onde Conseguir Ajuda* e bem-vindo(a) ao CakePHP!

## Leitura complementar

Existem tópicos comuns que as pessoas que estão estudando o CakePHP normalmente visitam a seguir:

1. *view-layouts*: Customizando o layout da aplicação
2. *view-elements*: Inclusão e reutilização de elementos na view
3. *Geração de código com o Bake*: Gerando código CRUD
4. *Tutorial - Criando um Blog - Autenticação e Autorização*: Tutorial de autorização e autenticação

## Tutorial - Criando um Blog - Parte 3

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>8</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Tutorial - Criando um Blog - Autenticação e Autorização

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>9</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>8</sup><https://github.com/cakephp/docs>

<sup>9</sup><https://github.com/cakephp/docs>

---

# Contribuindo

---

Existem várias maneiras de contribuir com o CakePHP. As seções abaixo irão abordar estas formas de contribuição:

## Documentação

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Tickets

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Código

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Padrões de código

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Guia de retrocompatibilidade

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>5</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>3</sup><https://github.com/cakephp/docs>

<sup>4</sup><https://github.com/cakephp/docs>

<sup>5</sup><https://github.com/cakephp/docs>



---

# Instalação

---

O CakePHP é rápido e fácil de instalar. Os requisitos mínimos são um servidor web e uma cópia do CakePHP, só isso! Apesar deste manual focar principalmente na configuração do Apache (porque ele é o mais simples de instalar e configurar), o CakePHP vai ser executado em uma série de servidores web como nginx, LightHTTPD, ou Microsoft IIS.

## Requisitos

- HTTP Server. Por exemplo: Apache. De preferência com mod\_rewrite ativo, mas não é obrigatório.
- PHP 5.4.16 ou superior.
- extensão mbstring
- extensão intl

---

**Nota:** Tanto no XAMPP quanto no WAMP, as extensões mcrypt e mbstring são setadas por padrão.

Se você estiver usando o XAMPP, já tem a extensão intl inclusa, mas é preciso descomentar a linha `extension=php_intl.dll` no arquivo `php.ini` e então, reiniciar o servidor através do painel de controle do XAMPP.

Caso você esteja usando o WAMP, a extensão intl está “ativa” por padrão, mas não está funcional. Para fazê-la funcionar, você deve ir à pasta do php (que por padrão é) `C:\wamp\bin\php\php{version}`, copiar todos os arquivos que se pareçam com `icu***.dll` e colá-los no diretório “bin” do apache `C:\wamp\bin\apache\apache{version}\bin`. Reiniciando todos os serviços a extensão já deve ficar ok.

---

Apesar de um mecanismo de banco de dados não ser exigido, nós imaginamos que a maioria das aplicações irá utilizar um. O CakePHP suporta uma variedade de mecanismos de armazenamento de banco de dados:

- MySQL (5.1.10 ou superior)
- PostgreSQL

- Microsoft SQL Server (2008 ou superior)
- SQLite 3

---

**Nota:** Todos os drivers incluídos internamente requerem PDO. Você deve assegurar-se que possui a extensão PDO correta instalada.

---

## Instalando o CakePHP

O CakePHP utiliza [Composer](#)<sup>1</sup>, uma ferramenta de gerenciamento de dependências para PHP 5.3+, como o método suportado oficialmente para instalação.

Primeiramente, você precisará baixar e instalar o Composer se não o fez anteriormente. Se você tem cURL instalada, é tão fácil quanto executar o seguinte:

```
curl -s https://getcomposer.org/installer | php
```

Ou, você pode baixar `composer.phar` do [Site oficial do Composer](#)<sup>2</sup>.

Para sistemas Windows, você pode baixar o instalador [aqui](#)<sup>3</sup>. Mais instruções para o instalador Windows do Composer podem ser encontradas dentro do LEIA-ME [aqui](#)<sup>4</sup>.

Agora que você baixou e instalou o Composer, você pode receber uma nova aplicação CakePHP executando:

```
php composer.phar create-project --prefer-dist cakephp/app [app_name]
```

Ou se o Composer estiver instalado globalmente:

```
composer create-project --prefer-dist cakephp/app [app_name]
```

Uma vez que o Composer terminar de baixar o esqueleto da aplicação e o núcleo da biblioteca CakePHP, você deve ter uma aplicação funcional instalada via Composer. Esteja certo de manter os arquivos `composer.json` e `composer.lock` com o restante do seu código fonte.

You can now visit the path to where you installed your CakePHP application and see the setup traffic lights.

## Mantendo sincronização com as últimas alterações no CakePHP

Se você quer se manter atualizado com as últimas mudanças no CakePHP, você pode adicionar o seguinte ao `composer.json` de sua aplicação:

```
"require": {  
    "cakephp/cakephp": "dev-master"  
}
```

---

<sup>1</sup><http://getcomposer.org>

<sup>2</sup><https://getcomposer.org/download/>

<sup>3</sup><https://github.com/composer/windows-setup/releases/>

<sup>4</sup><https://github.com/composer/windows-setup>

Onde `<branch>` é o nome do branch que você segue. Toda vez que você executar `php composer.phar update` você receberá as últimas atualizações do branch escolhido.

## Permissões

O CakePHP utiliza o diretório `tmp` para diversas operações. Descrição de models, views armazenadas em cache e informações de sessão são apenas alguns exemplos. O diretório `logs` é utilizado para escrever arquivos de log pelo mecanismo padrão `FileLog`.

Como tal, certifique-se que os diretórios `logs`, `tmp` e todos os seus sub-diretórios em sua instalação CakePHP são graváveis pelo usuário relacionado ao servidor web. O processo de instalação do Composer faz `tmp` e seus sub-diretórios globalmente graváveis para obter as coisas funcionando rapidamente, mas você pode atualizar as permissões para melhor segurança e mantê-los graváveis apenas para o usuário relacionado ao servidor web.

Um problema comum é que os diretórios e sub-diretórios de `logs` e `tmp` devem ser graváveis tanto pelo servidor quanto pelo usuário da linha de comando. Em um sistema UNIX, se seu usuário relacionado ao servidor web é diferente do seu usuário da linha de comando, você pode executar somente uma vez os seguintes comandos a partir do diretório da sua aplicação para assegurar que as permissões serão configuradas corretamente:

```
HTTPDUSER=`ps aux | grep -E '[a]pache|[h]ttpd|[_]www|[w]ww-data|[n]ginx' | grep -v root | l
setfacl -R -m u:${HTTPDUSER}:rwx tmp
setfacl -R -d -m u:${HTTPDUSER}:rwx tmp
setfacl -R -m u:${HTTPDUSER}:rwx logs
setfacl -R -d -m u:${HTTPDUSER}:rwx logs
```

## Servidor de Desenvolvimento

Uma instalação de desenvolvimento é o método mais rápido de configurar o CakePHP. Neste exemplo, nós vamos utilizar o console CakePHP para executar o servidor integrado do PHP que vai tornar sua aplicação disponível em `http://host:port`. A partir do diretório da aplicação, execute:

```
bin/cake server
```

Por padrão, sem nenhum argumento fornecido, isso vai disponibilizar a sua aplicação em `http://localhost:8765/`.

Se você tem algo conflitante com `localhost` ou porta `8765`, você pode dizer ao console CakePHP para executar o servidor web em um host e/ou porta específica utilizando os seguintes argumentos:

```
bin/cake server -H 192.168.13.37 -p 5673
```

Isto irá disponibilizar sua aplicação em `http://192.168.13.37:5673/`.

É isso aí! Sua aplicação CakePHP está instalada e funcionando sem ter que configurar um servidor web.

**Aviso:** O servidor de desenvolvimento *nunca* deve ser usado em um ambiente de produção. Destina-se apenas como um servidor de desenvolvimento básico.

Se você preferir usar um servidor web real, você deve ser capaz de mover a instalação do CakePHP (incluindo os arquivos ocultos) para dentro do diretório raiz do seu servidor web. Você deve, então, ser capaz de apontar seu navegador para o diretório que você moveu os arquivos para dentro e ver a aplicação em ação.

## Produção

Uma instalação de produção é uma forma mais flexível de configurar o CakePHP. Usar este método permite total domínio para agir como uma aplicação CakePHP singular. Este exemplo o ajudará a instalar o CakePHP em qualquer lugar em seu sistema de arquivos e torná-lo disponível em <http://www.example.com>. Note que esta instalação pode exigir os direitos de alterar o `DocumentRoot` em servidores web Apache.

Depois de instalar a aplicação usando um dos métodos acima no diretório de sua escolha - vamos supor que você escolheu `/cake_install` - sua configuração de produção será parecida com esta no sistema de arquivos:

```
/cake_install/  
  bin/  
  config/  
  logs/  
  plugins/  
  src/  
  tests/  
  tmp/  
  vendor/  
  webroot/ (esse diretório é definido como DocumentRoot)  
  .gitignore  
  .htaccess  
  .travis.yml  
  composer.json  
  index.php  
  phpunit.xml.dist  
  README.md
```

Desenvolvedores utilizando Apache devem definir a diretiva `DocumentRoot` pelo domínio para:

```
DocumentRoot /cake_install/webroot
```

Se o seu servidor web está configurado corretamente, agora você deve encontrar sua aplicação CakePHP acessível em <http://www.example.com>.

## Aquecendo

Tudo bem, vamos ver o CakePHP em ação. Dependendo de qual configuração você usou, você deve apontar seu navegador para <http://example.com/> ou <http://localhost:8765/>. Nesse ponto, você será apresentado à página home padrão do CakePHP e uma mensagem que diz a você o estado da sua conexão atual com o banco de dados.

Parabéns! Você está pronto para *create your first CakePHP application*.

## Reescrita de URL

### Apache

Apesar do CakePHP ser construído para trabalhar com `mod_rewrite` fora da caixa, e normalmente o faz, nos atentamos que alguns usuários lutam para conseguir fazer tudo funcionar bem em seus sistemas.

Aqui estão algumas coisas que você poderia tentar para conseguir tudo rodando corretamente. Primeiramente observe seu `httpd.conf`. (Tenha certeza que você está editando o `httpd.conf` do sistema ao invés de um usuário, ou site específico.)

Esses arquivos podem variar entre diferentes distribuições e versões do Apache. Você também pode pesquisar em <http://wiki.apache.org/httpd/DefaultLayout> para maiores informações.

1. Tenha certeza que a sobreescrita do `.htaccess` está permitida e que `AllowOverride` está definido para `All` no correto `DocumentRoot`. Você deve ver algo similar a:

```
# Cada diretório ao qual o Apache tenha acesso pode ser configurado com respeito
# a quais serviços e recursos estão permitidos e/ou desabilitados neste
# diretório (e seus sub-diretórios).
#
# Primeiro, nós configuramos o "default" para ser um conjunto bem restrito de
# recursos.
<Directory />
    Options FollowSymLinks
    AllowOverride All
#     Order deny,allow
#     Deny from all
</Directory>
```

2. Certifique-se que o `mod_rewrite` está sendo carregado corretamente. Você deve ver algo como:

```
LoadModule rewrite_module libexec/apache2/mod_rewrite.so
```

Em muitos sistemas estará comentado por padrão, então você pode apenas remover os símbolos `#`.

Depois de fazer as mudanças, reinicie o Apache para certificar-se que as configurações estão ativas.

Verifique se os seus arquivos `.htaccess` estão realmente nos diretórios corretos. Alguns sistemas operacionais tratam arquivos iniciados com `'.'` como ocultos e portanto, não os copia.

3. Certifique-se de sua cópia do CakePHP vir da seção de downloads do site ou do nosso repositório Git, e que foi descompactado corretamente, verificando os arquivos `.htaccess`.

O diretório `app` do CakePHP (será copiado para o diretório mais alto de sua aplicação através do `bake`):

```
<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteRule ^$ webroot/ [L]
```

```
RewriteRule      (.*) webroot/$1      [L]
</IfModule>
```

O diretório webroot do CakePHP (será copiado para a raiz de sua aplicação através do bake):

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Se o seu site CakePHP ainda possuir problemas com mod\_rewrite, você pode tentar modificar as configurações para Virtual Hosts. No Ubuntu, edita o arquivo /etc/apache2/sites-available/default (a localização depende da distribuição). Nesse arquivo, certifique-se que AllowOverride None seja modificado para AllowOverride All, então você terá:

```
<Directory />
    Options FollowSymLinks
    AllowOverride All
</Directory>
<Directory /var/www>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride All
    Order Allow,Deny
    Allow from all
</Directory>
```

No Mac OSX, outra solução é usar a ferramenta [virtualhostx](#)<sup>5</sup> para fazer um Virtual Host apontar para o seu diretório.

Para muitos serviços de hospedagem (GoDaddy, land1), seu servidor web é na verdade oferecido a partir de um diretório de usuário que já utiliza mod\_rewrite. Se você está instalando o CakePHP em um diretório de usuário (<http://example.com/~username/cakephp/>), ou qualquer outra estrutura URL que já utilize mod\_rewrite, você precisará adicionar declarações RewriteBase para os arquivos .htaccess que o CakePHP utiliza. (.htaccess, webroot/.htaccess).

Isso pode ser adicionado na mesma seção com a diretiva RewriteEngine, por exemplo, seu arquivo webroot/.htaccess ficaria como:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /path/to/app
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Os detalhes dessas mudanças vão depender da sua configuração, e podem incluir coisas adicionais que não estão relacionadas ao CakePHP. Por favor, busque pela documentação online do Apache para mais informações.

4. (Opcional) Para melhorar a configuração de produção, você deve prevenir conteúdos inválidos de serem analisados pelo CakePHP. Modifique seu webroot/.htaccess para algo como:

---

<sup>5</sup><http://clickontyler.com/virtualhostx/>

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /path/to/app/
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_URI} !^(webroot/)?(img|css|js)/(.*)$
  RewriteRule ^ index.php [L]
</IfModule>
```

Isto irá simplesmente prevenir conteúdo incorreto de ser enviado para o `index.php` e então exibir sua página de erro 404 do servidor web.

Adicionalmente você pode criar uma página HTML de erro 404 correspondente, ou utilizar a padrão do CakePHP ao adicionar uma diretiva `ErrorDocument`:

```
ErrorDocument 404 /404-not-found
```

## nginx

nginx não utiliza arquivos `.htaccess` como o Apache, então é necessário criar as reescritas de URL na configuração de sites disponíveis. Dependendo da sua configuração, você precisará modificar isso, mas pelo menos, você vai precisar do PHP rodando como uma instância FastCGI:

```
server {
    listen 80;
    server_name www.example.com;
    rewrite ^(.*) http://example.com$1 permanent;
}

server {
    listen 80;
    server_name example.com;

    # root directive should be global
    root /var/www/example.com/public/webroot/;
    index index.php;

    access_log /var/www/example.com/log/access.log;
    error_log /var/www/example.com/log/error.log;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }

    location ~ \.php$ {
        try_files $uri =404;
        include /etc/nginx/fastcgi_params;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }
}
```

## IIS7 (Windows hosts)

IIS7 não suporta nativamente arquivos .htaccess. Mesmo existindo add-ons que adicionam esse suporte, você também pode importar as regras .htaccess no IIS para utilizar as reescritas nativas do CakePHP. Para isso, siga os seguintes passos:

1. Utilize o [Microsoft's Web Platform Installer](#)<sup>6</sup> para instalar o [Rewrite Module 2.0](#)<sup>7</sup> ou baixe-o diretamente (32-bit<sup>8</sup> / 64-bit<sup>9</sup>).
2. Crie um novo arquivo chamado web.config em seu diretório raiz do CakePHP.
3. Utilize o Notepad ou qualquer editor seguro XML para copiar o seguinte código em seu novo arquivo web.config:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <system.webServer>
    <rewrite>
      <rules>
        <rule name="Exclude direct access to webroot/*"
          stopProcessing="true">
          <match url="^webroot/(.*)$" ignoreCase="false" />
          <action type="None" />
        </rule>
        <rule name="Rewrite routed access to assets(img, css, files, js, favicon)
          stopProcessing="true">
          <match url="^(img|css|files|js|favicon.ico)(.*)$" />
          <action type="Rewrite" url="webroot/{R:1}{R:2}"
            appendQueryString="false" />
        </rule>
        <rule name="Rewrite requested file/folder to index.php"
          stopProcessing="true">
          <match url="^(.*)$" ignoreCase="false" />
          <action type="Rewrite" url="index.php"
            appendQueryString="true" />
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
</configuration>
```

Uma vez que o arquivo web.config é criado com as regras amigáveis de reescrita do IIS, os links, CSS, JavaScript, e roteamento do CakePHP agora devem funcionar corretamente.

## Não posso utilizar Reescrita de URL

Se você não quer ou não pode ter mod\_rewrite (ou algum outro módulo compatível) funcionando no seu servidor, você precisará utilizar as URLs amigáveis nativas do CakePHP. No **config/app.php**, descomente

<sup>6</sup><http://www.microsoft.com/web/downloads/platform.aspx>

<sup>7</sup><http://www.iis.net/downloads/microsoft/url-rewrite>

<sup>8</sup><http://www.microsoft.com/en-us/download/details.aspx?id=5747>

<sup>9</sup><http://www.microsoft.com/en-us/download/details.aspx?id=7435>



a linha que se parece como:

```
'App' => [  
    // ...  
    // 'baseUrl' => env('SCRIPT_NAME'),  
]
```

Também remova esses arquivos .htaccess:

```
/.htaccess  
webroot/.htaccess
```

Isso fará suas URLs parecerem como `www.example.com/index.php/controllername/actionname/param` ao invés de `www.example.com/controllername/actionname/param`.



---

# Configuração

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](https://github.com)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Roteamento

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Filtros do Dispatcher

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>



---

## Objetos de requisição e resposta

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

## Controllers (Controladores)

---

**class** Cake\Controller\Controller

Os controllers (controladores) 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 `AppController`, a qual por sua vez estende a classe do core `Controller`. A classe `AppController` pode ser definida em `src/Controller/AppController.php` e deve conter métodos que são compartilhados entre todos os controllers de sua aplicação.

Os controllers fornecem uma série de métodos que lidam com requisições. Estas são chamados de *actions*. Por padrão, todos os métodos públicos em um controller são uma action e acessíveis por uma URL. Uma action é responsável por interpretar a requisição e criar a resposta. Normalmente as respostas são na forma de uma view renderizada, mas também existem outras formas de criar respostas.

### O App Controller

Como mencionado anteriormente, a classe `AppController` é a mãe de todos os outros controllers da sua aplicação. A própria `AppController` é estendida da classe `Cake\Controller\Controller` incluída no CakePHP. Assim sendo, `AppController` é definida em `src/Controller/AppController.php` como a seguir:

```
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{
}
```

Os atributos e métodos criados em seu `AppController` vão estar disponíveis para todos os controllers que a estendam. Components (sobre os quais você irá aprender mais tarde) são a melhor alternativa para códigos usados por muitos (mas não necessariamente em todos) controllers.

Você pode usar seu `AppController` para carregar components que serão usados em cada controller de sua aplicação. O CakePHP oferece um método `initialize()` que é invocado ao final do construtor do controller para esse tipo de uso:

```
namespace App\Controller;

use Cake\Controller\Controller;

class AppController extends Controller
{

    public function initialize()
    {
        // Sempre habilite o CSRF component.
        $this->loadComponent('Csrf');
    }

}
```

Em adição ao método `initialize()`, a antiga propriedade `$components` também vai permitir você declarar quais components devem ser carregados. Enquanto heranças objeto-orientadas normais são encaixadas, os components e helpers usados por um controller são especialmente tratados. Nestes casos, os valores de propriedade do `AppController` são mesclados com arrays de classes controller filhas. Os valores na classe filha irão sempre sobre-escrever aqueles na `AppController`.

## Fluxo de requisições

Quando uma requisição é feita para uma aplicação CakePHP, a classe `Cake\Routing Router` e a classe `Cake\Routing Dispatcher` usam *routes-configuration* para encontrar e criar a instância correta do controller. Os dados da requisição são encapsulados em um objeto de requisição. O CakePHP coloca todas as informações importantes de uma requisição na propriedade `$this->request`. Veja a seção *cake-request* para mais informações sobre o objeto de requisição do CakePHP.

## Métodos (actions) de controllers

Actions de controllers são responsáveis por converter os parâmetros de requisição em uma resposta para o navegador/usuário que fez a requisição. O CakePHP usa convenções para automatizar este processo e remove alguns códigos clichês que você teria que escrever de qualquer forma.

Por convenção, o CakePHP renderiza uma view com uma versão flexionada do nome da action. Retornando ao nosso exemplo da padaria online, nosso `RecipesController` poderia abrigar as actions `view()`, `share()` e `search()`. O controller seria encontrado em `src/Controller/RecipesController.php` contendo:

```
// src/Controller/RecipesController.php

class RecipesController extends AppController
{
    function view($id)
    {
        // A lógica da action vai aqui.
    }

    function share($customerId, $recipeId)
    {
        // A lógica da action vai aqui.
    }

    function search($query)
    {
        // A lógica da action vai aqui.
    }
}
```

Os arquivos de template para estas actions seriam `src/Template/Recipes/view.ctp`, `src/Template/Recipes/share.ctp` e `src/Template/Recipes/search.ctp`. A nomenclatura convencional para arquivos view é a versão lowercased (minúscula) e underscored (sem sublinhado) do nome da action.

Actions dos controllers geralmente usam `Controller::set()` para criar um contexto que a `View` usa para renderizar a camada view. Devido às convenções que o CakePHP usa, você não precisa criar e renderizar as views manualmente. Ao invés, uma vez que uma action de controller é completada, o CakePHP irá manipular a renderização e devolver a view.

Se por alguma razão você quiser pular o comportamento padrão, você pode retornar um objeto `Cake\Network\Response` a partir da action com a resposta definida.

Quando você usa métodos do controller com `Routing\RequestActionTrait::requestAction()` você irá tipicamente retornar uma instância `Response`. Se você tem métodos de controller que são usados para requisições web normais + `requestAction`, você deve checar o tipo de requisição antes de retornar:

```
// src/Controller/RecipesController.php

class RecipesController extends AppController
{
    public function popular()
    {
```

```
$popular = $this->Recipes->find('popular');
if ($this->request->is('requested')) {
    $this->response->body(json_encode($popular));
    return $this->response;
}
$this->set('popular', $popular);
}
```

A action do controller acima é um exemplo de como um método pode ser usado com `Routing\RequestActionTrait::requestAction()` e requisições normais.

Para que você possa utilizar um controller de forma eficiente em sua própria aplicação, nós iremos cobrir alguns dos atributos e métodos oferecidos pelo controller do core do CakePHP.

## Interagindo com views

Os controllers interagem com as views de diversas maneiras. Primeiro eles são capazes de passar dados para as views usando `Controller::set()`. Você também pode decidir no seu controller qual arquivo view deve ser renderizado através do controller.

## Definindo variáveis para a view

`Cake\Controller\Controller::set (string $var, mixed $value)`

O método `Controller::set()` é a principal maneira de enviar dados do seu controller para a sua view. Após ter usado o método `Controller::set()`, a variável pode ser acessada em sua view:

```
// Primeiro você passa os dados do controller:
$this->set('color', 'pink');

// Então, na view, você pode utilizar os dados:
?>
```

Você selecionou a cobertura `<?php echo $color; ?>` para o bolo.

O método `Controller::set()` também aceita um array associativo como primeiro parâmetro. Isto pode oferecer uma forma rápida para atribuir uma série de informações para a view:

```
$data = [
    'color' => 'pink',
    'type' => 'sugar',
    'base_price' => 23.95
];

// Faça $color, $type, e $base_price
// disponíveis na view:

$this->set($data);
```

## Renderizando uma view

`Cake\Controller\Controller::render()` (*string \$view, string \$layout*)

O método `Controller::render()` é chamado automaticamente no fim de cada ação requisitada de um controller. Este método executa toda a lógica da view (usando os dados que você passou usando o método `Controller::set()`), coloca a view em `View::$layout`, e serve de volta para o usuário final.

O arquivo view usado pelo método `Controller::render()` é determinado por convenção. Se a action `search()` do controller `RecipesController` é requisitada, o arquivo view encontrado em **src/Template/Recipes/search.ctp** será renderizado:

```
namespace App\Controller;

class RecipesController extends AppController
{
    // ...
    public function search()
    {
        // Render the view in src/Template/Recipes/search.ctp
        $this->render();
    }
    // ...
}
```

Embora o CakePHP irá chamar o método `Controller::render()` automaticamente (ao menos que você altere o atributo `$this->autoRender` para `false`) após cada action, você pode usá-lo para especificar um arquivo view alternativo especificando o nome do arquivo view como primeiro parâmetro do método `Controller::render()`.

Se o parâmetro `$view` começar com '/', é assumido ser um arquivo view ou elemento relativo ao diretório `/src/Template`. Isto permite a renderização direta de elementos, muito útil em chamadas AJAX:

```
// Renderiza o elemento em src/Template/Element/ajaxreturn.ctp
$this->render('/Element/ajaxreturn');
```

O segundo parâmetro `$layout` do `Controller::render()` permite que você especifique o layout pelo qual a view é renderizada.

## Renderizando uma view específica

Em seu controller você pode querer renderizar uma view diferente do que a convencional. Você pode fazer isso chamando o método `Controller::render()` diretamente. Uma vez chamado o método `Controller::render()`, o CakePHP não tentará renderizar novamente a view:

```
namespace App\Controller;

class PostsController extends AppController
{
    public function my_action()
    {
        $this->render('custom_file');
    }
}
```

```
}
}
```

Isto renderizaria o arquivo `src/Template/Posts/custom_file.ctp` ao invés de `src/Template/Posts/my_action.ctp`

Você também pode renderizar views de plugins utilizando a seguinte sintaxe: `$this->render('PluginName.PluginController/custom_file')`. Por exemplo:

```
namespace App\Controller;

class PostsController extends AppController
{
    public function my_action()
    {
        $this->render('Users.UserDetails/custom_file');
    }
}
```

Isto renderizaria `plugins/Users/src/Template/UserDetails/custom_file.ctp`

## Redirecionando para outras páginas

`Cake\Controller\Controller::redirect` (*stringlarray \$url, integer \$status*)

O método de controle de fluxo que você vai usar na majoritariamente é `Controller::redirect()`. Este método recebe seu primeiro parâmetro na forma de uma URL relativa do CakePHP. Quando um usuário executar um pedido com êxito, você pode querer redirecioná-lo para uma tela de recepção.

```
public function place_order()
{
    // Logic for finalizing order goes here
    if ($success) {
        return $this->redirect(
            ['controller' => 'Orders', 'action' => 'thanks']
        );
    }
    return $this->redirect(
        ['controller' => 'Orders', 'action' => 'confirm']
    );
}
```

Este método irá retornar a instância da resposta com cabeçalhos apropriados definidos. Você deve retornar a instância da resposta da sua action para prevenir renderização de view e deixar o dispatcher controlar o redirecionamento corrente.

Você também pode usar uma URL relativa ou absoluta como o parâmetro `$url`:

```
return $this->redirect('/orders/thanks');
return $this->redirect('http://www.example.com');
```

Você também pode passar dados para a action:

```
return $this->redirect(['action' => 'edit', $id]);
```

O segundo parâmetro passado no `Controller::redirect()` permite a você definir um código de status HTTP para acompanhar o redirecionamento. Você pode querer usar o código 301 (movido permanentemente) ou 303 (veja outro), dependendo da natureza do redirecionamento.

Se você precisa redirecionar o usuário de volta para a página que fez a requisição, você pode usar:

```
$this->redirect($this->referer());
```

Um exemplo usando seqüências de consulta e hash pareceria com:

```
return $this->redirect([
    'controller' => 'Orders',
    'action' => 'confirm',
    '?' => [
        'product' => 'pizza',
        'quantity' => 5
    ],
    '#' => 'top'
]);
```

A URL gerada seria:

```
http://www.example.com/orders/confirm?product=pizza&quantity=5#top
```

## Redirecionando para outra action no mesmo Controller

`Cake\Controller\Controller::setAction($action, $args...)`

Se você precisar redirecionar a atual action para uma diferente no *mesmo* controller, você pode usar `Controller::setAction()` para atualizar o objeto da requisição, modificar o template da view que será renderizado e redirecionar a execução para a action especificada:

```
// De uma action delete, você pode renderizar uma página
// de índice atualizada.
$this->setAction('index');
```

## Carregando models adicionais

`Cake\Controller\Controller::loadModel(string $modelClass, string $type)`

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.:

```
// Em um método do controller.
$this->loadModel('Articles');
$recentArticles = $this->Articles->find('all', [
    'limit' => 5,
    'order' => 'Articles.created DESC'
]);
```

Se você está usando um provedor de tabelas que não os da ORM nativa você pode ligar este sistema de tabelas aos controllers do CakePHP conectando seus métodos de factory:

```
// Em um método do controller.
$this->modelFactory(
    'ElasticIndex',
    ['ElasticIndexes', 'factory']
);
```

Depois de registrar uma tabela factory, você pode usar o `loadModel` para carregar instâncias:

```
// Em um método do controller
$this->loadModel('Locations', 'ElasticIndex');
```

---

**Nota:** O `TableRegistry` da ORM nativa é conectado por padrão como o provedor de ‘Tabelas’.

---

## Paginando um model

`Cake\Controller\Controller::paginate()`

Este método é usado para fazer a paginação dos resultados retornados por seus models. Você pode especificar o tamanho da página (quantos resultados serão retornados), as condições de busca e outros parâmetros. Veja a seção *pagination* para mais detalhes sobre como usar o método `paginate()`

O atributo `paginate` lhe oferece uma forma fácil de customizar como `paginate()` se comporta:

```
class ArticlesController extends AppController
{
    public $paginate = [
        'Articles' => [
            'conditions' => ['published' => 1]
        ]
    ];
}
```

## Configurando components para carregar

`Cake\Controller\Controller::loadComponent($name, $config = [])`

Em seu método `initialize()` do controller você pode definir qualquer component que quiser carregado, e qualquer configuração de dados para eles:

```
public function initialize()
{
    parent::initialize();
    $this->loadComponent('Csrft');
    $this->loadComponent('Comments', Configure::read('Comments'));
}
```



**property** `Cake\Controller\Controller::$components`

A propriedade `$components` em seus controllers permitem a você configurar components. Components configurados e suas dependências serão criados pelo CakePHP para você. Leia a seção *configuring-components* para mais informações. Como mencionado anteriormente, a propriedade `$components` será mesclada com a propriedade definida em cada classe parente do seu controller.

## Configurando helpers para carregar

**property** `Cake\Controller\Controller::$helpers`

Vamos observar como dizer ao controller do CakePHP que você planeja usar classes MVC adicionais:

```
class RecipesController extends AppController
{
    public $helpers = ['Form'];
}
```

Cada uma dessas variáveis são mescladas com seus valores herdados, portanto não é necessário (por exemplo) redeclarar o `FormHelper`, ou qualquer coisa declarada em seu `AppController`.

## Ciclo de vida de callbacks em uma requisição

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

`Cake\Controller\Controller::beforeFilter` (*Event \$event*)

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.

---

**Nota:** O método `beforeFilter()` será chamado para ações perdidas.

---

`Cake\Controller\Controller::beforeRender` (*Event \$event*)

Chamada após a lógica da action de um controller, mas antes da view ser renderizada. Esse callback não é usado frequentemente, mas pode ser necessário se você estiver chamando `Controller\Controller::render()` manualmente antes do final de uma determinada action.

`Cake\Controller\Controller::afterFilter` ()

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

Em adição ao ciclo de vida dos callbacks do controller, *Components (Componentes)* também oferece um conjunto de callbacks similares.

Lembre de chamar os callbacks do `AppController` em conjunto com os callbacks dos controllers para melhores resultados:

```
public function beforeFilter(Event $event)
{
    parent::beforeFilter($event);
}
```

## Mais sobre controllers

### O Pages Controller

CakePHP é distribuído com o controller **PagesController.php**. Esse controller é simples, seu uso é opcional e normalmente direcionado a prover páginas estáticas. A homepage que você vê logo depois de instalar o CakePHP utiliza esse controller e o arquivo da view fica em **src/Template/Pages/home.ctp**. Se você criar o arquivo **src/Template/Pages/about.ctp**, você poderá acessá-lo em **http://example.com/pages/about**. Fique a vontade para alterar esse controller para atender suas necessidades ou mesmo excluí-lo.

Quando você cria sua aplicação pelo Composer, o `PagesController` vai ser criado na pasta **src/Controller/**.

### Components (Componentes)

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Authentication

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### CookieComponent

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Cross Site Request Forgery

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Flash

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>5</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Security

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>6</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Pagination

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

---

<sup>3</sup><https://github.com/cakephp/docs>

<sup>4</sup><https://github.com/cakephp/docs>

<sup>5</sup><https://github.com/cakephp/docs>

<sup>6</sup><https://github.com/cakephp/docs>

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>7</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Request Handling

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>8</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>7</sup><https://github.com/cakephp/docs>

<sup>8</sup><https://github.com/cakephp/docs>

---

# Views (Visualização)

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## More About Views

### View Cells (Células de Visualização)

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Temas

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>

<sup>3</sup><https://github.com/cakephp/docs>

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Views JSON & XML

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Helpers (Facilitadores)

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>5</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Flash

```
class Cake\View\Helper\FlashHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>6</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Form

```
class Cake\View\Helper\FormHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

---

<sup>4</sup><https://github.com/cakephp/docs>

<sup>5</sup><https://github.com/cakephp/docs>

<sup>6</sup><https://github.com/cakephp/docs>

Por favor, sintaxe a vontade para nos enviar um pull request no [Github](#)<sup>7</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Html

```
class Cake\View\Helper\HtmlHelper (View $view, array $config =[])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintaxe a vontade para nos enviar um pull request no [Github](#)<sup>8</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Number

```
class Cake\View\Helper\NumberHelper (View $view, array $config =[])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintaxe a vontade para nos enviar um pull request no [Github](#)<sup>9</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Paginator

```
class Cake\View\Helper\PaginatorHelper (View $view, array $config =[])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintaxe a vontade para nos enviar um pull request no [Github](#)<sup>10</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>7</sup><https://github.com/cakephp/docs>

<sup>8</sup><https://github.com/cakephp/docs>

<sup>9</sup><https://github.com/cakephp/docs>

<sup>10</sup><https://github.com/cakephp/docs>

### RSS

```
class Cake\View\Helper\RssHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>11</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Session

```
class Cake\View\Helper\SessionHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>12</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Text

```
class Cake\View\Helper\TextHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>13</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Time

```
class Cake\View\Helper\TimeHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>14</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

---

<sup>11</sup><https://github.com/cakephp/docs>

<sup>12</sup><https://github.com/cakephp/docs>

<sup>13</sup><https://github.com/cakephp/docs>

<sup>14</sup><https://github.com/cakephp/docs>



Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Url

```
class Cake\View\Helper\UrlHelper (View $view, array $config = [])
```

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>15</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>15</sup><https://github.com/cakephp/docs>



---

## Models (Modelos)

---

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 estendido 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.

Antes de nós começarmos explorando o ORM, tenha certeza que você *configure your database connections*.

---

**Nota:** Se você é familiarizado com versões anteriores do CakePHP, você deveria ler o *Guia de atualização para o novo ORM* para esclarecer diferenças importantes entre o CakePHP 3.0 e suas versões antigas.

---

### Exemplo rápido

Para começar você não precisa escrever código. Se você seguiu as convenções do CakePHP para suas tabelas de banco de dados, você pode simplesmente começar a usar o ORM. Por exemplo, se quiséssemos carregar alguns dados da nossa tabela `articles` poderíamos fazer:

```
use Cake\ORM\TableRegistry;
$articles = TableRegistry::get('Articles');
$query = $articles->find();
```

```
foreach ($query as $row) {
    echo $row->title;
}
```

Nota-se que nós não temos que criar qualquer código ou definir qualquer configuração. As convenções do CakePHP nos permitem pular alguns códigos clichê, e permitir que o framework insira classes básicas enquanto sua aplicação não criou uma classe concreta. Se quiséssemos customizar nossa classe `ArticlesTable` adicionando algumas associações ou definir alguns métodos adicionais, deveríamos acrescentar o seguinte a `src/Model/Table/ArticlesTable.php` após a tag de abertura `<?php`:

```
namespace App\Model\Table;

use Cake\ORM\Table;

class ArticlesTable extends Table
{
}
```

Classes de tabela usam a versão `CamelCased` do nome da tabela com o sufixo `Table` como o nome da classe. Uma vez que sua classe fora criada, você recebe uma referência para esta utilizando o `ORM\TableRegistry` como antes:

```
use Cake\ORM\TableRegistry;
// Agora $articles é uma instância de nossa classe ArticlesTable.
$articles = TableRegistry::get('Articles');
```

Agora que temos uma classe de tabela concreta, nós provavelmente vamos querer usar uma classe de entidade concreta. As classes de entidade permitem definir métodos de acesso, métodos mutantes, definir lógica personalizada para os registros individuais e muito mais. Vamos começar adicionando o seguinte para `src/Model/Entity/Article.php` após a tag de abertura `<?php`:

```
namespace App\Model\Entity;

use Cake\ORM\Entity;

class Article extends Entity
{
}
```

Entidades usam a versão singular `CamelCase` do nome da tabela como seu nome de classe por padrão. Agora que nós criamos nossa classe de entidade, quando carregarmos entidades do nosso banco de dados, nós iremos receber instâncias da nossa nova classe `Article`:

```
use Cake\ORM\TableRegistry;

// Agora uma instância de ArticlesTable.
$articles = TableRegistry::get('Articles');
$query = $articles->find();

foreach ($query as $row) {
    // Cada linha é agora uma instância de nossa classe Article.
}
```

```
echo $row->title;
}
```

CakePHP utiliza convenções de nomenclatura para ligar as classes de tabela e entidade juntas. Se você precisar customizar qual entidade uma tabela utiliza, você pode usar o método `entityClass()` para definir nomes de classe específicos.

Veja os capítulos em *Objetos de tabela* e *Entities (Entidades)* para mais informações sobre como usar objetos de tabela e entidades em sua aplicação.

## Mais informação

### O básico sobre banco de dados

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Construtor de queries

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Objetos de tabela

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>

<sup>3</sup><https://github.com/cakephp/docs>

## Entities (Entidades)

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Requisitando dados e conjuntos de resultados

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>5</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Validando dados

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>6</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Salvando dados

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>7</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>4</sup><https://github.com/cakephp/docs>

<sup>5</sup><https://github.com/cakephp/docs>

<sup>6</sup><https://github.com/cakephp/docs>

<sup>7</sup><https://github.com/cakephp/docs>

## Deletando dados

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>8</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Associações - Conectando tabelas

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>9</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Behaviors (Comportamentos)

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>10</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## CounterCache

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>11</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>8</sup><https://github.com/cakephp/docs>

<sup>9</sup><https://github.com/cakephp/docs>

<sup>10</sup><https://github.com/cakephp/docs>

<sup>11</sup><https://github.com/cakephp/docs>

## Timestamp

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>12</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Translate

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>13</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Tree

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>14</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Schema

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>15</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>12</sup><https://github.com/cakephp/docs>

<sup>13</sup><https://github.com/cakephp/docs>

<sup>14</sup><https://github.com/cakephp/docs>

<sup>15</sup><https://github.com/cakephp/docs>



## ORM Cache Shell

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>16</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>16</sup><https://github.com/cakephp/docs>



---

# Authentication

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Bake Console

---

O bake console do CakePHP é outro empenho para você ter o CakePHP configurado e funcionando rápido. O bake console pode criar qualquer ingrediente básico do CakePHP: models, behaviors, views, helpers, components, test cases, fixtures e plugins. E nós não estamos apenas falando de classes esqueleto: O Bake pode criar uma aplicação totalmente funcional em questão de minutos. De fato, o Bake é um passo natural a se dar uma vez que a aplicação tem seu alicerce construído.

## Instalação

Antes de tentar usar ou estender o bake, tenha certeza que ele está instalado em sua aplicação. O bake é distribuído como um plugin que você pode instalar com o Composer:

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

Isto irá instalar o bake como uma dependência de desenvolvimento, sendo assim, não instalado quando em um ambiente de produção. As seções a seguir cobrem o uso do bake com mais detalhes:

## Geração de código com o Bake

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>1</sup><https://github.com/cakephp/docs>

## Extendendo o Bake

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>2</sup><https://github.com/cakephp/docs>

---

# Caching

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

## Console e Shells

---

O CakePHP não oferece um framework apenas para desenvolvimento web, mas também um framework para criação de aplicações de console. Estas aplicações são ideais para manipular variadas tarefas em segundo plano como manutenção e complementação de trabalho fora do ciclo requisição-resposta. As aplicações de console do CakePHP permitem a você reutilizar suas classes de aplicação a partir da linha de comando.

O CakePHP traz consigo algumas aplicações de console nativas. Algumas dessas aplicações são utilizadas em conjunto com outros recursos do CakePHP (como `i18n`), e outros de uso geral para aceleração de trabalho.

### O Console do CakePHP

Esta seção provê uma introdução à linha de comando do CakePHP. Ferramentas de console são ideais para uso em cron jobs, ou utilitários baseados em linha de comando que não precisam ser acessíveis por um navegador web.

O PHP provê um cliente CLI que faz interface com o seu sistema de arquivos e aplicações de forma muito mais suave. O console do CakePHP provê um framework para criar scripts shell. O console utiliza uma configuração tipo dispatcher para carregar uma shell ou tarefa, e prover seus parâmetros.

---

**Nota:** Uma linha de comando (CLI) constituída a partir do PHP deve estar disponível no sistema se você planeja utilizar o Console.

---

Antes de entrar em detalhes, vamos ter certeza de que você pode executar o console do CakePHP. Primeiro, você vai precisar executar um sistema shell. Os exemplos apresentados nesta seção serão em bash, mas o Console do CakePHP é compatível com o Windows também. Este exemplo assume que o usuário está conectado em um prompt do bash e está atualmente na raiz de uma aplicação CakePHP.

Aplicações CakePHP possuem um diretório *Console* que contém todas as shells e tarefas para uma aplicação. Ele também vem com um executável:

```
$ cd /path/to/app
$ bin/cake
```

Executar o Console sem argumentos produz esta mensagem de ajuda:

```
Welcome to CakePHP v3.0.0 Console
-----
App : App
Path: /Users/markstory/Sites/cakephp-app/src/
-----
Current Paths:

-app: src
-root: /Users/markstory/Sites/cakephp-app
-core: /Users/markstory/Sites/cakephp-app/vendor/cakephp/cakephp

Changing Paths:

Your working path should be the same as your application path. To change your path use the
Example: -app relative/path/to/myapp or -app /absolute/path/to/myapp

Available Shells:

[Bake] bake

[Migrations] migrations

[CORE] i18n, orm_cache, plugin, server

[app] behavior_time, console, orm

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
```

A primeira informação impressa refere-se a caminhos. Isso é útil se você estiver executando o console a partir de diferentes partes do sistema de arquivos.

## Criando uma Shell

Vamos criar uma shell para utilizar no Console. Para este exemplo, criaremos uma simples Hello World (Olá Mundo) shell. No diretório **src/Shell** de sua aplicação crie **HelloShell.php**. Coloque o seguinte código dentro do arquivo recém criado:

```
namespace App\Shell;

use Cake\Console\Shell;

class HelloShell extends Shell
{
    public function main()
    {
        $this->out('Hello world.');
```

```
}
}
```

As convenções para as classes de shell são de que o nome da classe deve corresponder ao nome do arquivo, com o sufixo de Shell. No nosso shell criamos um método `main()`. Este método é chamado quando um shell é chamado sem comandos adicionais. Vamos adicionar alguns comandos daqui a pouco, mas por agora vamos executar a nossa shell. A partir do diretório da aplicação, execute:

```
bin/cake hello
```

Você deve ver a seguinte saída:

```
Welcome to CakePHP Console
-----
App : app
Path: /Users/markstory/Sites/cake_dev/src/
-----
Hello world.
```

Como mencionado antes, o método `main()` em shells é um método especial chamado sempre que não há outros comandos ou argumentos dados para uma shell. Por nosso método principal não ser muito interessante, vamos adicionar outro comando que faz algo:

```
namespace App\Shell;

use Cake\Console\Shell;

class HelloShell extends Shell
{
    public function main()
    {
        $this->out('Hello world.');
```

```
    public function heyThere($name = 'Anonymous')
    {
        $this->out('Hey there ' . $name);
    }
}
```

Depois de salvar o arquivo, você deve ser capaz de executar o seguinte comando e ver o seu nome impresso:

```
bin/cake hello hey_there your-name
```

Qualquer método público não prefixado por um `_` é permitido para ser chamado a partir da linha de comando. Como você pode ver, os métodos invocados a partir da linha de comando são transformados do argumento prefixado para a forma correta do nome camel-cased (camelizada) na classe.

No nosso método `heyThere()` podemos ver que os argumentos posicionais são providos para nossa função `heyThere()`. Argumentos posicionais também estão disponíveis na propriedade `args`. Você pode acessar switches ou opções em aplicações shell, estando disponíveis em `$this->params`, mas nós iremos cobrir isso daqui a pouco.

Quando utilizando um método `main()` você não estará liberado para utilizar argumentos posicionais. Isso

se deve ao primeiro argumento posicional ou opção ser interpretado(a) como o nome do comando. Se você quer utilizar argumentos, você deve usar métodos diferentes de `main()`.

## Usando Models em suas shells

Você frequentemente precisará acessar a camada lógica de negócios em seus utilitários shell; O CakePHP faz essa tarefa super fácil. Você pode carregar models em shells assim como faz em um controller utilizando `loadModel()`. Os models carregados são definidos como propriedades anexas à sua shell:

```
namespace App\Shell;

use Cake\Console\Shell;

class UserShell extends Shell
{
    public function initialize()
    {
        parent::initialize();
        $this->loadModel('Users');
    }

    public function show()
    {
        if (empty($this->args[0])) {
            return $this->error('Por favor, indique um nome de usuário.');
        }
        $user = $this->Users->findByUsername($this->args[0])->first();
        $this->out(print_r($user, true));
    }
}
```

A shell acima, irá preencher um user pelo seu username e exibir a informação armazenada no banco de dados.

## Tasks de Shell

Haverão momentos construindo aplicações mais avançadas de console que você vai querer compor funcionalidades em classes reutilizáveis que podem ser compartilhadas através de muitas shells. Tasks permitem que você extraia comandos em classes. Por exemplo, o `bake` é feito quase que completamente de tasks. Você define tasks para uma shell usando a propriedade `$tasks`:

```
class UserShell extends Shell
{
    public $tasks = ['Template'];
}
```

Você pode utilizar tasks de plugins utilizando o padrão *plugin syntax*. Tasks são armazenadas sob `Shell/Task/` em arquivos nomeados depois de suas classes. Então se nós estivéssemos criando uma nova task 'FileGenerator', você deveria criar `src/Shell/Task/FileGeneratorTask.php`.

Cada task deve ao menos implementar um método `main()`. O `ShellDispatcher`, vai chamar esse método quando a task é invocada. Uma classe task se parece com:

```
namespace App\Shell\Task;

use Cake\Console\Shell;

class FileGeneratorTask extends Shell
{
    public function main()
    {
    }
}
```

Uma shell também pode prover acesso a suas tasks como propriedades, que fazem tasks serem ótimas para criar punhados de funcionalidade reutilizáveis similares a *Components (Componentes)*:

```
// Localizado em src/Shell/SeaShell.php
class SeaShell extends Shell
{
    // Localizado em src/Shell/Task/SoundTask.php
    public $tasks = ['Sound'];

    public function main()
    {
        $this->Sound->main();
    }
}
```

Você também pode acessar tasks diretamente da linha de comando:

```
$ cake sea sound
```

**Nota:** Para acessar tasks diretamente através da linha de comando, a task **deve** ser incluída na propriedade da classe shell `$tasks`. Portanto, esteja ciente que um método chamado “sound” na classe `SeaShell` deve sobrescrever a habilidade de acessar a funcionalidade na task `Sound`, especificada no array `$tasks`.

## Carregando Tasks em tempo-real com TaskRegistry

Você pode carregar arquivos em tempo-real utilizando o Task registry object. Você pode carregar tasks que não foram declaradas no `$tasks` dessa forma:

```
$project = $this->Tasks->load('Project');
```

Carregará e retornará uma instância `ProjectTask`. Você pode carregar tasks de plugins usando:

```
$progressBar = $this->Tasks->load('ProgressBar.ProgressBar');
```

## Invocando outras Shells a partir da sua Shell

Cake\Console\**dispatchShell** (*\$args*)

Existem ainda muitos casos onde você vai querer invocar uma shell a partir de outra. `Shell::dispatchShell()` lhe dá a habilidade de chamar outras shells ao providenciar o `argv` para a sub shell. Você pode providenciar argumentos e opções tanto como variáveis ou como strings:

```
// Como uma string
$this->dispatchShell('schema create Blog --plugin Blog');

// Como um array
$this->dispatchShell('schema', 'create', 'Blog', '--plugin', 'Blog');
```

O conteúdo acima mostra como você pode chamar a shell schema para criar o schema de um plugin de dentro da shell do próprio.

## Recenendo Input de usuários

Cake\Console\**in** (*\$question*, *\$choices = null*, *\$default = null*)

Quando construir aplicações interativas pelo console você irá precisar receber inputs dos usuários. CakePHP oferece uma forma fácil de fazer isso:

```
// Receber qualquer texto dos usuários.
$color = $this->in('What color do you like?');

// Receber uma escolha dos usuários.
$selection = $this->in('Red or Green?', ['R', 'G'], 'R');
```

A validação de seleção é insensitiva a maiúsculas / minúsculas.

## Criando Arquivos

Cake\Console\**createFile** (*\$path*, *\$contents*)

Muitas aplicações Shell auxiliam tarefas de desenvolvimento e implementação. Criar arquivos é frequentemente importante nestes casos de uso. O CakePHP oferece uma forma fácil de criar um arquivo em um determinado diretório:

```
$this->createFile('bower.json', $stuff);
```

Se a Shell for interativa, um alerta vai ser gerado, e o usuário questionado se ele quer sobrescrever o arquivo caso já exista. Se a propriedade de interação da shell for `false`, nenhuma questão será disparada e o arquivo será simplesmente sobrescrito.

## Saída de dados do Console

A classe `Shell` oferece alguns métodos para direcionar conteúdo:

```
// Escreve para stdout
$this->out('Normal message');

// Escreve para stderr
$this->err('Error message');

// Escreve para stderr e para o processo
$this->error('Fatal error');
```

A `Shell` também inclui métodos para limpar a saída de dados, criando linhas em branco, ou desenhando uma linha de traços:

```
// Exibe 2 linhas novas
$this->out($this->nl(2));

// Limpa a tela do usuário
$this->clear();

// Desenha uma linha horizontal
$this->hr();
```

Por último, você pode atualizar a linha atual de texto na tela usando `_io->overwrite()`:

```
$this->out('Counting down');
$this->out('10', 0);
for ($i = 9; $i > 0; $i--) {
    sleep(1);
    $this->_io->overwrite($i, 0, 2);
}
```

É importante lembrar, que você não pode sobrescrever texto uma vez que uma nova linha tenha sido exibida.

## Console Output Levels

Shells frequentemente precisam de diferentes níveis de verbosidade. Quando executadas como cron jobs, muitas saídas são desnecessárias. E há ocasiões que você não estará interessado em tudo que uma shell tenha a dizer. Você pode usar os níveis de saída para sinalizar saídas apropriadamente. O usuário da shell, pode então decidir qual nível de detalhe ele está interessado ao sinalizar o chamado da shell. `Cake\Console\Shell::out()` suporta 3 tipos de saída por padrão.

- QUIET - Apenas informação absolutamente importante deve ser sinalizada.
- NORMAL - O nível padrão, e uso normal.
- VERBOSE - Sinalize mensagens que podem ser irritantes em demasia para uso diário, mas informativas para depuração como VERBOSE.

Você pode sinalizar a saída da seguinte forma:

```
// Deve aparecer em todos os níveis.
$this->out('Quiet message', 1, Shell::QUIET);
$this->quiet('Quiet message');

// Não deve aparecer quando a saída quiet estiver alternado.
$this->out('normal message', 1, Shell::NORMAL);
$this->out('loud message', 1, Shell::VERBOSE);
$this->verbose('Verbose output');

// Deve aparecer somente quando a saída verbose estiver habilitada.
$this->out('extra message', 1, Shell::VERBOSE);
$this->verbose('Verbose output');
```

Você pode controlar o nível de saída das shells, ao usar as opções `--quiet` e `--verbose`. Estas opções são adicionadas por padrão, e permitem a você controlar consistentemente níveis de saída dentro das suas shells do CakePHP.

### Estilizando a saída de dados

Estilizar a saída de dados é feito ao incluir tags - como no HTML - em sua saída. O `ConsoleOutput` irá substituir estas tags com a seqüência correta de código ansi. Há diversos estilos nativos, e você pode criar mais. Os nativos são:

- `error` Mensagens de erro. Texto sublinhado vermelho.
- `warning` Mensagens de alerta. Texto amarelo.
- `info` Mensagens informativas. Texto ciano.
- `comment` Texto adicional. Texto azul.
- `question` Texto que é uma questão, adicionado automaticamente pela shell.

Você pode criar estilos adicionais usando `$this->stdout->styles()`. Para declarar um novo estilo de saída você pode fazer:

```
$this->_io->styles('flashy', ['text' => 'magenta', 'blink' => true]);
```

Isso deve então permiti-lo usar uma `<flashy>` tag na saída de sua shell, e se as cores ansi estiverem habilitadas, o seguinte pode ser renderizado como texto magenta piscante `$this->out('<flashy>Whooooa</flashy> Something went wrong');`. Quando definir estilos você pode usar as seguintes cores para os atributos `text` e `background`:

- `black`
- `red`
- `green`
- `yellow`
- `blue`
- `magenta`



- cyan
- white

Você também pode usar as seguintes opções através de valores booleanos, defini-los com valor positivo os habilita.

- bold
- underline
- blink
- reverse

Adicionar um estilo o torna disponível para todas as instâncias do `ConsoleOutput`, então você não tem que redeclarar estilos para os objetos `stdout` e `stderr` respectivamente.

## Desabilitando a colorização

Mesmo que a colorização seja incrível, haverá ocasiões que você querará desabilitá-la, ou forçá-la:

```
$this->_io->outputAs(ConsoleOutput::RAW);
```

O citado irá colocar o objeto de saída em modo raw. Em modo raw, nenhum estilo é aplicado. Existem três modos que você pode usar.

- `ConsoleOutput::RAW` - Saída raw, nenhum estilo ou formatação serão aplicados. Este é um modo indicado se você estiver exibindo XML ou, quiser depurar porquê seu estilo não está funcionando.
- `ConsoleOutput::PLAIN` - Saída de texto simples, tags conhecidas de estilo serão removidas da saída.
- `ConsoleOutput::COLOR` - Saída onde a cor é removida.

Por padrão em sistemas \*nix objetos `ConsoleOutput` padronizam-se a a saída de cores. Em sistemas Windows, a saída simples é padrão a não ser que a variável de ambiente `ANSICON` esteja presente.

## Opções de configuração e Geração de ajuda

**class** `Cake\Console\ConsoleOptionParser`

`ConsoleOptionParser` oferece uma opção de CLI e analisador de argumentos.

`OptionParsers` permitem a você completar dois objetivos ao mesmo tempo. Primeiro, eles permitem definir opções e argumentos para os seus comandos. Isso permite separar validação básica de dados e seus comandos do console. Segundo, permite prover documentação, que é usada para gerar arquivos de ajuda bem formatados.

O console framework no CakePHP recebe as opções do seu interpretador shell ao chamar `$this->getOptionParser()`. Sobre-escrever esse método permite configurar o `OptionParser` para definir as entradas aguardadas da sua shell. Você também pode configurar interpretadores de subcomandos,

que permitem ter diferentes interpretadores para subcomandos e tarefas. O `ConsoleOptionParser` implementa uma interface fluida e inclui métodos para facilmente definir múltiplas opções/argumentos de uma vez:

```
public function getOptionParser()
{
    $parser = parent::getOptionParser();
    // Configure parser
    return $parser;
}
```

## Configurando um interpretador de opção com a interface fluida

Todos os métodos que configuram um interpretador de opções podem ser encadeados, permitindo definir um interpretador de opções completo em uma série de chamadas de métodos:

```
public function getOptionParser()
{
    $parser = parent::getOptionParser();
    $parser->addArgument('type', [
        'help' => 'Either a full path or type of class.'
    ]->addArgument('className', [
        'help' => 'A CakePHP core class name (e.g: Component, HtmlHelper).'
    ]->addOption('method', [
        'short' => 'm',
        'help' => __('The specific method you want help on.')
    ]->description(__('Lookup doc block comments for classes in CakePHP.));
    return $parser;
}
```

Os métodos que permitem encadeamento são:

- `description()`
- `epilog()`
- `command()`
- `addArgument()`
- `addArguments()`
- `addOption()`
- `addOptions()`
- `addSubcommand()`
- `addSubcommands()`

`Cake\Console\ConsoleOptionParser::description($text = null)`

Recebe ou define a descrição para o interpretador de opções. A descrição é exibida acima da informação do argumento e da opção. Ao instanciar tanto em array como em string, você pode definir o valor da descrição. Instanciar sem argumentos vai retornar o valor atual:

```
// Define múltiplas linhas de uma vez
$parser->description(['line one', 'line two']);

// Lê o valor atual
$parser->description();
```

`Cake\Console\ConsoleOptionParser::epilog($text = null)`

Recebe ou define o epílogo para o interpretador de opções. O epílogo é exibido depois da informação do argumento e da opção. Ao instanciar tanto em array como em string, você pode definir o valor do epílogo. Instanciar sem argumentos vai retornar o valor atual:

```
// Define múltiplas linhas de uma vez
$parser->epilog(['line one', 'line two']);

// Lê o valor atual
$parser->epilog();
```

## Adicionando argumentos

`Cake\Console\ConsoleOptionParser::addArgument($name, $params = [])`

Argumentos posicionais são frequentemente usados em ferramentas de linha de comando, e `ConsoleOptionParser` permite definir argumentos bem como torná-los requiríveis. Você pode adicionar argumentos um por vez com `$parser->addArgument()`; ou múltiplos de uma vez com `$parser->addArguments()`;

```
$parser->addArgument('model', ['help' => 'The model to bake']);
```

Você pode usar as seguintes opções ao criar um argumento:

- `help` O texto de ajuda a ser exibido para este argumento.
- `required` Se esse parâmetro é requisito.
- `index` O índice do argumento, se deixado indefinido, o argumento será colocado no final dos argumentos. Se você definir o mesmo índice duas vezes, a primeira opção será sobrescrita.
- `choices` Um array de opções válidas para esse argumento. Se deixado vazio, todos os valores são válidos. Uma exceção será lançada quando `parse()` encontrar um valor inválido.

Argumentos que forem definidos como requisito lançarão uma exceção quando interpretarem o comando se eles forem omitidos. Então você não tem que lidar com isso em sua shell.

`Cake\Console\ConsoleOptionParser::addArguments(array $args)`

Se você tem um array com múltiplos argumentos você pode usar `$parser->addArguments()` para adicioná-los de uma vez.:

```
$parser->addArguments([
    'node' => ['help' => 'The node to create', 'required' => true],
    'parent' => ['help' => 'The parent node', 'required' => true]
]);
```

Assim como todos os métodos de construção no `ConsoleOptionParser`, `addArguments` pode ser usado como parte de um fluido método encadeado.

### Validando argumentos

Ao criar argumentos posicionais, você pode usar a marcação `required` para indicar que um argumento deve estar presente quando uma shell é chamada. Adicionalmente você pode usar o `choices` para forçar um argumento a ser de uma lista de escolhas válidas:

```
$parser->addArgument('type', [
    'help' => 'The type of node to interact with.',
    'required' => true,
    'choices' => ['aro', 'aco']
]);
```

O código acima irá criar um argumento que é requisitado e tem validação no input. Se o argumento está tanto indefinido, ou possui um valor incorreto, uma exceção será lançada e a shell parará.

### Adicionando opções

`Cake\Console\ConsoleOptionParser::addOption($name, $options = [])`

Opções são frequentemente usadas em ferramentas CLI. `ConsoleOptionParser` suporta a criação de opções com `verbose` e aliases curtas, suprimindo padrões e criando ativadores booleanos. Opções são criadas tanto com `$parser->addOption()` ou `$parser->addOptions()`:

```
$parser->addOption('connection', [
    'short' => 'c',
    'help' => 'connection',
    'default' => 'default',
]);
```

O código citado permite a você usar tanto `cake myshell --connection=other`, `cake myshell --connection other`, ou `cake myshell -c other` quando invocando a shell. Você também criar ativadores booleanos. Estes ativadores não consomem valores, e suas presenças apenas os habilitam nos parâmetros interpretados.:

```
$parser->addOption('no-commit', ['boolean' => true]);
```

Com essa opção, ao chamar uma shell como `cake myshell --no-commit something` o parâmetro `no-commit` deve ter um valor de `true`, e `something` deve ser tratado como um argumento posicional. As opções nativas `--help`, `--verbose`, e `--quiet` usam essa funcionalidade.

Ao criar opções você pode usar os seguintes argumentos para definir o seu comportamento:

- `short` - A variação de letra única para essa opção, deixe indefinido para `none`.
- `help` - Texto de ajuda para essa opção. Usado ao gerar ajuda para a opção.
- `default` - O valor padrão para essa opção. Se não estiver definido o valor padrão será `true`.
- `boolean` - A opção não usa valor, é apenas um ativador booleano. Por padrão `false`.

- `choices` - Um array de escolhas válidas para essa opção. Se deixado vazio, todos os valores são considerados válidos. Uma exceção será lançada quando `parse()` encontrar um valor inválido.

`Cake\Console\ConsoleOptionParser::addOptions` (*array \$options*)

Se você tem um array com múltiplas opções, você pode usar `$parser->addOptions()` para adicioná-las de uma vez.:

```
$parser->addOptions([
    'node' => ['short' => 'n', 'help' => 'The node to create'],
    'parent' => ['short' => 'p', 'help' => 'The parent node']
]);
```

Assim como com todos os métodos construtores, no `ConsoleOptionParser`, `addOptions` pode ser usado como parte de um método fluente encadeado.

## Validando opções

Opções podem ser fornecidas com um conjunto de escolhas bem como argumentos posicionais podem ser. Quando uma opção define escolhas, essas são as únicas opções válidas para uma opção. Todos os outros valores irão gerar um `InvalidArgumentException`:

```
$parser->addOption('accept', [
    'help' => 'What version to accept.',
    'choices' => ['working', 'theirs', 'mine']
]);
```

## Usando opções booleanas

As opções podem ser definidas como opções booleanas, que são úteis quando você precisa criar algumas opções de marcação. Como opções com padrões, opções booleanas sempre irão incluir `-se` nos parâmetros analisados. Quando as marcações estão presentes elas são definidas para `true`, quando elas estão ausentes, são definidas como `false`:

```
$parser->addOption('verbose', [
    'help' => 'Enable verbose output.',
    'boolean' => true
]);
```

A opção seguinte resultaria em `$this->params['verbose']` sempre estando disponível. Isso permite a você omitir verificações `empty()` ou `isset()` em marcações booleanas:

```
if ($this->params['verbose']) {
    // Do something.
}
```

Desde que as opções booleanas estejam sempre definidas como `true` ou `false`, você pode omitir métodos de verificação adicionais.

## Adicionando subcomandos

```
Cake\Console\ConsoleOptionParser::addSubcommand($name, $options = [])
```

Aplicativos de console são muitas vezes feitas de subcomandos, e esses subcomandos podem exigir a análise de opções especiais e terem a sua própria ajuda. Um perfeito exemplo disso é `bake`. `Bake` é feita de muitas tarefas separadas e todas têm a sua própria ajuda e opções. `ConsoleOptionParser` permite definir subcomandos e fornecer comandos analisadores de opção específica, de modo que a shell sabe como analisar os comandos para as suas funções:

```
$parser->addSubcommand('model', [
    'help' => 'Bake a model',
    'parser' => $this->Model->getOptionParser()
]);
```

A descrição acima é um exemplo de como você poderia fornecer ajuda e um especializado interpretador de opção para a tarefa de uma shell. Ao chamar a tarefa de `getOptionParser()` não temos de duplicar a geração do interpretador de opção, ou misturar preocupações no nosso shell. Adicionar subcomandos desta forma tem duas vantagens. Primeiro, ele permite que o seu shell documente facilmente seus subcomandos na ajuda gerada. Ele também dá fácil acesso ao subcomando `help`. Com o subcomando acima criado você poderia chamar `cake myshell --help` e ver a lista de subcomandos, e também executar o `cake myshell model --help` para exibir a ajuda apenas o modelo de tarefa.

---

**Nota:** Uma vez que seu Shell define subcomandos, todos os subcomandos deve ser explicitamente definidos.

---

Ao definir um subcomando, você pode usar as seguintes opções:

- `help` - Texto de ajuda para o subcomando.
- `parser` - Um `ConsoleOptionParser` para o subcomando. Isso permite que você crie métodos analisadores de opção específicos. Quando a ajuda é gerada por um subcomando, se um analisador está presente ele vai ser usado. Você também pode fornecer o analisador como uma matriz que seja compatível com `Cake\Console\ConsoleOptionParser::buildFromArray()`

Adicionar subcomandos pode ser feito como parte de uma cadeia de métodos fluente.

## Construir uma ConsoleOptionParser de uma matriz

```
Cake\Console\ConsoleOptionParser::buildFromArray($spec)
```

Como mencionado anteriormente, ao criar interpretadores de opção de subcomando, você pode definir a especificação interpretadora como uma matriz para esse método. Isso pode ajudar fazer analisadores mais facilmente, já que tudo é um array:

```
$parser->addSubcommand('check', [
    'help' => __('Check the permissions between an ACO and ARO.'),
    'parser' => [
        'description' => [
            __("Use this command to grant ACL permissions. Once executed, the "),
            __("ARO specified (and its children, if any) will have ALLOW access "),
        ]
    ]
]);
```

```

        __("to the specified ACO action (and the ACO's children, if any).")
    ],
    'arguments' => [
        'aro' => ['help' => __('ARO to check.'), 'required' => true],
        'aco' => ['help' => __('ACO to check.'), 'required' => true],
        'action' => ['help' => __('Action to check')]
    ]
]
]);

```

Dentro da especificação do interpretador, você pode definir as chaves para arguments, options, description e epilog. Você não pode definir subcommands dentro de um construtor estilo array. Os valores para os argumentos e opções, devem seguir o formato que `Cake\Console\ConsoleOptionParser::addArguments()` e `Cake\Console\ConsoleOptionParser::addOptions()` usam. Você também pode usar `buildFromArray` por conta própria, para construir um interpretador de opção:

```

public function getOptionParser()
{
    return ConsoleOptionParser::buildFromArray([
        'description' => [
            __("Use this command to grant ACL permissions. Once executed, the "),
            __("ARO specified (and its children, if any) will have ALLOW access "),
            __("to the specified ACO action (and the ACO's children, if any).")
        ],
        'arguments' => [
            'aro' => ['help' => __('ARO to check.'), 'required' => true],
            'aco' => ['help' => __('ACO to check.'), 'required' => true],
            'action' => ['help' => __('Action to check')]
        ]
    ]);
}

```

## Recebendo ajuda das Shells

Com a adição de `ConsoleOptionParser` receber ajuda de shells é feito de uma forma consistente e uniforme. Ao usar a opção `--help` ou `-h` você pode visualizar a ajuda para qualquer núcleo shell, e qualquer shell que implementa um `ConsoleOptionParser`:

```

cake bake --help
cake bake -h

```

Ambos devem gerar a ajuda para o `bake`. Se o shell suporta subcomandos você pode obter ajuda para estes de uma forma semelhante:

```

cake bake model --help
cake bake model -h

```

Isso deve fornecer a você a ajuda específica para a tarefa `bake` dos `models`.

## Recebendo ajuda como XML

Quando a construção de ferramentas automatizadas ou ferramentas de desenvolvimento que necessitam interagir com shells do CakePHP, é bom ter ajuda disponível em uma máquina capaz interpretar formatos. O ConsoleOptionParser pode fornecer ajuda em xml, definindo um argumento adicional:

```
cake bake --help xml
cake bake -h xml
```

O trecho acima deve retornar um documento XML com a ajuda gerada, opções, argumentos e subcomando para o shell selecionado. Um documento XML de amostra seria algo como:

```
<?xml version="1.0"?>
<shell>
  <command>bake fixture</command>
  <description>Generate fixtures for use with the test suite. You can use
    `bake fixture all` to bake all fixtures.</description>
  <epilog>
    Omitting all arguments and options will enter into an interactive
    mode.
  </epilog>
  <subcommands/>
  <options>
    <option name="--help" short="-h" boolean="1">
      <default/>
      <choices/>
    </option>
    <option name="--verbose" short="-v" boolean="1">
      <default/>
      <choices/>
    </option>
    <option name="--quiet" short="-q" boolean="1">
      <default/>
      <choices/>
    </option>
    <option name="--count" short="-n" boolean="">
      <default>10</default>
      <choices/>
    </option>
    <option name="--connection" short="-c" boolean="">
      <default>default</default>
      <choices/>
    </option>
    <option name="--plugin" short="-p" boolean="">
      <default/>
      <choices/>
    </option>
    <option name="--records" short="-r" boolean="1">
      <default/>
      <choices/>
    </option>
  </options>
  <arguments>
    <argument name="name" help="Name of the fixture to bake.
```



```

        Can use Plugin.name to bake plugin fixtures." required="">
        <choices/>
    </argument>
</arguments>
</shell>

```

## Roteamento em Shells / CLI

Na interface de linha de comando (CLI), especificamente suas shells e tarefas, `env('HTTP_HOST')` e outras variáveis de ambiente `webbrowser` específica, não estão definidas.

Se você gerar relatórios ou enviar e-mails que fazem uso de `Router::url()`, estes conterão a máquina padrão `http://localhost/` e resultando assim em URLs inválidas. Neste caso, você precisa especificar o domínio manualmente. Você pode fazer isso usando o valor de configuração `App.fullBaseUrl` no seu bootstrap ou na sua configuração, por exemplo.

Para enviar e-mails, você deve fornecer a classe `CakeEmail` com o host que você deseja enviar o e-mail:

```

$email = new CakeEmail();
$email->domain('www.example.org');

```

Iste afirma que os IDs de mensagens geradas são válidos e adequados para o domínio a partir do qual os e-mails são enviados.

## Métodos enganchados

`Cake\Console\ConsoleOptionParser::initialize()`

Inicializa a Shell para atua como construtor de subclasses e permite configuração de tarefas antes de desenvolver a execução.

`Cake\Console\ConsoleOptionParser::startup()`

Inicia-se a Shell e exibe a mensagem de boas-vindas. Permite a verificação e configuração antes de comandar ou da execução principal.

Substitua este método se você quiser remover as informações de boas-vindas, ou outra forma modificar o fluxo de pré-comando.

## Mais tópicos

### Shell Helpers

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

<sup>1</sup><https://github.com/cakephp/docs>

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Console Interativo (REPL)

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Executando Shells como Cron Jobs

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### I18N Shell

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Completion Shell

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>5</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

---

<sup>2</sup><https://github.com/cakephp/docs>

<sup>3</sup><https://github.com/cakephp/docs>

<sup>4</sup><https://github.com/cakephp/docs>

<sup>5</sup><https://github.com/cakephp/docs>

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Plugin Shell

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>6</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Routes Shell

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>7</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Upgrade Shell

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>8</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>6</sup><https://github.com/cakephp/docs>

<sup>7</sup><https://github.com/cakephp/docs>

<sup>8</sup><https://github.com/cakephp/docs>



---

# Debugging

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Implantação

---

Uma vez que sua aplicação está completa, ou mesmo antes quando você quiser colocá-la no ar. Existem algumas poucas coisas que você deve fazer quando colocar em produção uma aplicação CakePHP.

## Atualizar config/app.php

Atualizar o arquivo **core.php**, especificamente o valor do `debug` é de extrema importância. Tornar o `debug` igual a `false` desabilita muitos recursos do processo de desenvolvimento que nunca devem ser expostos ao mundo. Desabilitar o `debug`, altera as seguintes coisas:

- 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.

Além dos itens citados acima, muitos plugins e extensões usam o valor do `debug` para modificarem seus comportamentos.

Por exemplo, você pode setar uma variável de ambiente em sua configuração do Apache:

```
SetEnv CAKEPHP_DEBUG 1
```

E então você pode definir o level de debug dinamicamente no **config/app.php**:

```
$debug = (bool) getenv('CAKEPHP_DEBUG');  
  
return [  
    'debug' => $debug,  
    .....  
];
```

## Checar a segurança

Se você está jogando sua aplicação na selva, é uma boa idéia certificar-se que ela não possui vulnerabilidades óbvias:

- 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.

## Definir a raiz do documento

Definir a raiz do documento da sua aplicação corretamente é um passo importante para manter seu código protegido e sua aplicação mais segura. As aplicações desenvolvidas com o CakePHP devem ter a raiz apontando para o diretório `webroot`. Isto torna a aplicação e os arquivos de configurações inacessíveis via URL. Configurar a raiz do documento depende de cada servidor web. Veja a *Reescrita de URL* para informações sobre servidores web específicos.

De qualquer forma você vai querer definir o host/domínio virtual para o `webroot/`. Isso remove a possibilidade de arquivos fora do diretório raiz serem executados.

## Aprimorar a performance de sua aplicação

O carregamento de classes pode alocar facilmente o tempo de processamento de sua aplicação. A fim de evitar esse problema, é recomendado que você execute este comando em seu servidor de produção uma vez que a aplicação esteja implantada:

```
php composer.phar dumpautoload -o
```

Sabendo que manipulação de referências estáticas, como imagens, JavaScript e arquivos CSS, plugins, através do `Dispatcher` é incrivelmente ineficiente, é fortemente recomendado referenciá-los simbolicamente para produção. Por exemplo:

```
ln -s Plugin/YourPlugin/webroot/css/yourplugin.css webroot/css/yourplugin.css
```



---

## Email

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## Erros & Exceções

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Sistema de eventos

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Internacionalização e Localização

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

# Logging

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](https://github.com)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## Formulários sem models

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Pagination

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Plugins

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

# REST

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Segurança

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Segurança

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Cross Site Request Forgery

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>

<sup>3</sup><https://github.com/cakephp/docs>

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Security

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>4</sup><https://github.com/cakephp/docs>

---

# Sessions

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## Testando

---

O CakePHP vem com suporte interno para testes e integração para o [PHPUnit](http://phpunit.de)<sup>1</sup>. Em adição aos recursos oferecidos pelo PHPUnit, o CakePHP oferece alguns recursos adicionais para fazer testes mais facilmente. Esta seção abordará a instalação do PHPUnit, começando com testes unitários e como você pode usar as extensões que o CakePHP oferece.

---

<sup>1</sup><http://phpunit.de>





---

# Validação

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](https://github.com)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## App Class

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## Collections (Coleções)

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## Arquivos & Pastas

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

# Hash

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Http Client

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Inflector

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Número

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

# Objetos de Registro

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

## Texto

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sintá-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Tempo

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](https://github.com)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Xml

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

## Constantes e Funções

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>



---

# Debug Kit

---

DebugKit é um plugin suportado pelo time principal que oferece uma barra de ferramentas para ajudar a fazer a depuração de aplicações do CakePHP mais facilmente.

## Instalação

Por padrão o DebugKit é instalado com o esqueleto padrão da aplicação. Se você o removeu e gostaria de reinstalá-lo, você pode fazê-lo ao executar o seguinte comando a partir do diretório raiz da aplicação (onde o arquivo `composer.json` está localizado):

```
php composer.phar require --dev cakephp/debug_kit "~3.0"
```

## Armazenamento do DebugKit

Por padrão, o DebugKit usa um pequeno banco de dados SQLite no diretório `/tmp` de sua aplicação para armazenar os dados do painel. Se você quizesse que o DebugKit armazenasse seus dados em outro lugar, você deve definir uma conexão `debug_kit`.

## Configuração do banco de dados

Por padrão, o DebugKit armazenará os dados do painel em um banco de dados SQLite no diretório `/tmp` de sua aplicação. Se você não puder instalar a extensão do PHP `pdo_sqlite`, você pode configurar o DebugKit para usar um banco de dados diferente ao definir uma conexão `debug_kit` em seu arquivo **`config/app.php`**.

## Uso da barra de ferramentas

A Barra de Ferramentas DebugKit é composta por vários painéis, que são mostrados ao clicar no ícone do CakePHP no canto inferior direito da janela do seu navegador. Uma vez que a barra de ferramentas é aberta, você deve ver uma série de botões. Cada um desses botões expande-se em um painel de informações relacionadas.

Cada painel permite que você olhar para um aspecto diferente da sua aplicação:

- **Cache** Exibe o uso de cache durante uma solicitação e limpa caches.
- **Environment** Exibe variáveis de ambiente relacionadas com PHP + CakePHP.
- **History** Exibe uma lista de requisições anteriores, e permite que você carregue e veja dados da barra de ferramentas a partir de solicitações anteriores.
- **Include** Exibe os arquivos inclusos divididos por grupo.
- **Log** Exibe todas as entradas feitas nos arquivos de log este pedido.
- **Request** Exibe informações sobre a requisição atual, GET, POST, informações sobre a rota atual do Cake e Cookies.
- **Session** Exibe a informação atual da sessão.
- **Sql Logs** Exibe logs SQL para cada conexão com o banco de dados.
- **Timer** Exibe qualquer temporizador que fora definido durante a requisição com `DebugKit\DebugTimer`, e memória utilizada coletada com `DebugKit\DebugMemory`.
- **Variables** Exibe variáveis de View definidas em um Controller.

Tipicamente, um painel manipula a recolha e apresentação de um único tipo de informações como logs ou informações de requisições. Você pode optar por visualizar painéis padrão da barra de ferramentas ou adicionar seus próprios painéis personalizados.

## Usando o painel History

O painel History é uma das mais frequentemente confundidas características do DebugKit. Ele oferece uma forma de visualizar os dados da barra de ferramentas de requisições anteriores, incluindo erros e redirecionamentos.

Como você pode ver, o painel contém uma lista de requisições. Na esquerda você pode ver um ponto marcando a requisição ativa. Clicar em quaisquer dados de requisição vai carregar os dados do painel para aquela requisição. Quando os dados são carregados, os títulos do painel vão sofrer uma transição para indicar que dados alternativos foram carregados.

## Desenvolvendo seus próprios painéis

Você pode criar seus próprios painéis customizados para o DebugKit para ajudar na depuração de suas aplicações.

## History



10 previous requests available

● Back to current request

1/11/15, 2:02 AM  
GET 404 text/html /bookmarks/derps

1/11/15, 2:01 AM  
GET 200 text/html /bookmarks

1/11/15, 2:01 AM  
GET 200 text/html /users

1/11/15, 2:00 AM  
GET 200 text/html /bookmarks/

1/11/15, 2:00 AM  
GET 200 text/html /bookmarks/

1/11/15, 2:00 AM  
GET 200 text/html /bookmarks/

1/11/15, 2:00 AM  
GET 200 text/html /bookmarks/

Cache Environment **History** Include Log 6 Request Session Sql Log 6 - 3 ms



## Criando uma Panel Class

Panel Classes precisam ser colocadas no diretório **src/Panel**. O nome do arquivo deve combinar com o nome da classe, então a classe `MyCustomPanel` deveria ter o nome de arquivo **src/Panel/MyCustomPanel.php**:

```
namespace App\Panel;

use DebugKit\DebugPanel;

/**
 * My Custom Panel
 */
class MyCustomPanel extends DebugPanel
{
    ...
}
```

Perceba que painéis customizados são necessários para estender a classe `DebugPanel`.

## Callbacks

Por padrão objetos do painel possuem dois callbacks, permitindo-lhes acoplar-se na requisição atual. Painéis inscrevem-se aos eventos `Controller.initialize` e `Controller.shutdown`. Se o seu painel precisa inscrever-se a eventos adicionais, você pode usar o método `implementedEvents` para definir todos os eventos que o seu painel possa estar interessado.

Você deveria estudar os painéis nativos para absorver alguns exemplos de como construir painéis.

## Elementos do painel

Cada painel deve ter um elemento view que renderiza o conteúdo do mesmo. O nome do elemento deve ser sublinhado e flexionado a partir do nome da classe. Por exemplo `SessionPanel` possui um elemento nomeado `session_panel.ctp`, e `SqllogPanel` possui um elemento nomeado `sqllog_panel.ctp`. Estes elementos devem estar localizados na raiz do seu diretório `src/Template/Element`.

## Títulos personalizados e Elementos

Os painéis devem pegar o seu título e nome do elemento por convenção. No entanto, se você precisa escolher um nome de elemento personalizado ou título, você pode definir métodos para customizar o comportamento do seu painel:

- `title()` - Configure o título que é exibido na barra de ferramentas.
- `elementName()` - Configure qual elemento deve ser utilizada para um determinado painel.

## Painéis em outros plugins

Painéis disponibilizados por *Plugins* funcionam quase que totalmente como outros plugins, com uma pequena diferença: Você deve definir `public $plugin` para ser o nome do diretório do plugin, com isso os elementos do painel poderão ser encontrados no momento de renderização:

```
namespace MyPlugin\Panel;

use DebugKit\DebugPanel;

class MyCustomPanel extends DebugPanel
{
    public $plugin = 'MyPlugin';
    ...
}
```

Para usar um plugin ou painel da aplicação, atualize a configuração do DebugKit de sua aplicação para incluir o painel:

```
Configure::write(
    'DebugKit.panels',
    array_merge(Configure::read('DebugKit.panels'), ['MyCustomPanel'])
);
```

O código acima deve carregar todos os painéis padrão tanto como os outros painéis customizados do `MyPlugin`.

---

# Migrations

---

---

**Nota:** A documentação não é atualmente suportada pela lingua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](https://github.com)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>1</sup><https://github.com/cakephp/docs>





---

# Apêndices

---

Os apêndices contêm informações sobre os novos recursos introduzidos em cada versão e a forma de executar a migração entre versões.

## Guia de Migração para a versão 3.0

A versão 3.0 ainda está em desenvolvimento, e qualquer mudança estará disponível apenas no branch 3.0 do git.

### 3.0 Guia de migração

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>1</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Guia de atualização para o novo ORM

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>2</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

<sup>1</sup><https://github.com/cakephp/docs>

<sup>2</sup><https://github.com/cakephp/docs>

## Guia de Migração para a versão 3.1

### 3.1 Guia de migração

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>3</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

## Informações Gerais

### Processo de desenvolvimento no CakePHP

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>4</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

### Glossário

---

**Nota:** A documentação não é atualmente suportada pela língua portuguesa nesta página.

Por favor, sinta-se a vontade para nos enviar um pull request no [Github](#)<sup>5</sup> ou use o botão **Improve This Doc** para propor suas mudanças diretamente.

Você pode referenciar-se à versão inglesa no menu de seleção superior para obter informações sobre o tópico desta página.

---

---

<sup>3</sup><https://github.com/cakephp/docs>

<sup>4</sup><https://github.com/cakephp/docs>

<sup>5</sup><https://github.com/cakephp/docs>

---

# PHP Namespace Index

---

## C

Cake\Console, 107

Cake\Controller, 75

Cake\View\Helper, 91



## Symbols

() (Cake\Console\ method), **112**

## A

addArgument() (Cake\Console\ConsoleOptionParser method), **117**

addArguments() (Cake\Console\ConsoleOptionParser method), **117**

addOption() (Cake\Console\ConsoleOptionParser method), **118**

addOptions() (Cake\Console\ConsoleOptionParser method), **119**

addSubcommand() (Cake\Console\ConsoleOptionParser method), **120**

afterFilter() (Cake\Controller\Controller method), **83**

## B

beforeFilter() (Cake\Controller\Controller method), **83**

beforeRender() (Cake\Controller\Controller method), **83**

buildFromArray() (Cake\Console\ConsoleOptionParser method), **120**

## C

Cake\Console (namespace), **107**

Cake\Controller (namespace), **75**

Cake\View\Helper (namespace), **88–91**

components (Cake\Controller\Controller property), **82**

ConsoleOptionParser (classe em Cake\Console), **115**

Controller (classe em Cake\Controller), **75**

## D

description() (Cake\Console\ConsoleOptionParser method), **116**

## E

epilog() (Cake\Console\ConsoleOptionParser method), **117**

## F

FlashHelper (classe em Cake\View\Helper), **88**

FormHelper (classe em Cake\View\Helper), **88**

## H

helpers (Cake\Controller\Controller property), **83**

HtmlHelper (classe em Cake\View\Helper), **89**

## I

initialize() (Cake\Console\ConsoleOptionParser method), **123**

## L

loadComponent() (Cake\Controller\Controller method), **82**

loadModel() (Cake\Controller\Controller method), **81**

## N

NumberHelper (classe em Cake\View\Helper), **89**

## P

paginate() (Cake\Controller\Controller method), **82**

PaginatorHelper (classe em Cake\View\Helper), **89**

## R

redirect() (Cake\Controller\Controller method), [80](#)

render() (Cake\Controller\Controller method), [79](#)

RssHelper (classe em Cake\View\Helper), [90](#)

## S

SessionHelper (classe em Cake\View\Helper), [90](#)

set() (Cake\Controller\Controller method), [78](#)

setAction() (Cake\Controller\Controller method), [81](#)

startup() (Cake\Console\ConsoleOptionParser method), [123](#)

## T

TextHelper (classe em Cake\View\Helper), [90](#)

TimeHelper (classe em Cake\View\Helper), [90](#)

## U

UrlHelper (classe em Cake\View\Helper), [91](#)