Criando aplicativo no CakePHP from Scratch

Para aprender mais sobre a codificação e como as informações caminham numa aplicação, iremos criar um pequenino aplicativo inteiramente digitando o código, na unha.

Criaremos apenas um banco de dados apenas com uma tabela (users), um Controller (Users) e seus actions e as ViewsTemplates: index, view, add e edit .

Vamos começar criando o Controller.

No Cake 2.x um modelo era composto por apenas uma classe Tables. No Cake 3 temos uma classe Table e uma Entity, cada uma em seu respectivo diretório (Table e Entity). Uma classe Table guarda informações sobre uma tabela, associações, validações e behaviors e uma Entity guarda informações sobre entidades/registros.

Bem, para não dizer que não tivemos ajuda criaremos o Controller com o Bake, mas, sem os actions, somente a classe vazia.

Instalação do Cake

Instalar Cake criando aplicativo scratch

Criando comp.bat para criar aplicativos

Para tornar a coisa mais prática eu criei um arquivo .bat com o comando do composer para criar aplicativo. O mesmo pode ser feito no Windows, Linux e Mac OS.
Apenas criei um arquivo comp.bat e coloquei dentro esta linha:

```
php ..\composer.phar create-project --prefer-dist cakephp/app %1
```

..\composer.phar – indica que o composer está na pasta anterior a atual.

Salvei na pasta c:\xampp\htdocs\cake, que é a pasta onde crio os aplicativos cake.
Como o composer.phar está no htdocs ficou com os ..
Como quero sempre passar o nome do aplicativo que quero criar como parâmetro usei o %1, para criar sem precisar editar o comp.bat.

Para criar um aplicativo, por exemplo, uso:
cd\xampp\htdocs\cake
comp scratch

que irá criar:
c:\xampp\htdocs\cake\scratch

Criar e Configurar banco scratch

Abrir phpmyadmin e criar o banco scratch.

Então clique na aba SQL e cole o script abaixo:

```
CREATE TABLE `users` (
  `id` int(10) UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `username` varchar(50) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  `role` varchar(20) DEFAULT NULL,
  `created` datetime DEFAULT now(),
  `modified` datetime DEFAULT now()
) ENGINE=InnoDB;

INSERT INTO `users` (`id`, `username`, `password`, `role`, `created`, `modified`) VALUES
(1, 'ribafs', 'ribafs', 'admin', now(), now()),
(2, 'fatima', 'fatima', 'user', now(), now()),
(3, 'tiago', 'tiago', 'admin', now(), now()),
(4, 'elias', 'elias', 'user', now(), now());
```

Criar banco e configurar aplicativo para o banco, contendo apenas a tabela users.

Configurar banco em config/app.php

Criar Controller sem actions

```
cd\xampp\htdocs\cake\scratch
bin\cake bake controller Users --no-actions
```

Com o comando acima o bake criou apenas isto:

```
<?php
namespace App\Controller;
use App\Controller\AppController;

class UsersController extends AppController
{
}
```

Se chamarmos no navegador com
http://localhost/cake/scratch/

Veremos apenas a view src\Template\Pages\home.ctp mostrada pelo action Display.

O casamento, já que não segue o padrão de mesmo nome para action e view, o casamento é feito pelo routes padrão do CakePHP. Veja:

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

Se chamarmos assim:
http://localhost/cake/scratch/users/

Ele vai nos avisar que falta o action index() do controller Users, que é o action default.

Criar um método público (action)
Então vamos criar o action index() dentro da classe UsersController.

Nossa classe deverá ficar assim:

```
class UsersController extends AppController
{
	public function index()
	{
		echo "Action index()";
		exit();
	}
}
```

Se chamarmos novamente no navegador
http://localhost/cake/scratch/users/

Agora nos mostrará a frase do echo.

Se chamarmos assim:
http://localhost/cake/scratch/users/index

Será semelhante, ou seja, o index() é o padrão e não precisa indicar.

Criar o action view()

Adicionar abaixo do index():

```
	public function view()
	{
		echo "Action view()";
		exit();
	}
```

Chamar pela URL
http://localhost/cake/scratch/users/view

Como a view() não é o action default somos obrigados e indicar na URL.

Agora vamos criar o Template para mostrar as informações. Ao invés de mostrar mensagens com echo diretamente no controller vamos fazer isso pelas views do Template, que serão chamadas pelos actions. Exemplo: o usuário digitou o endereço de um action de um controller, então este action chama a respectiva view do template.

Veja que mostramos uma mensagem nos actions e encerramos o processamento com a função exit(). Se comentar a funçãi exit() então o CakePHP continua o processamento e automaticamente chama uma view de mesmo nome do action.

Action index() chama a view index.ctp
Action view() chama a view view.ctp
E assim por diante.

O UsersController procura views em src\Template\Users

Criar a view index.ctp (ctp - Cake Template Page)

Criar a pasta Users na pasta Template (deve ser o mesmo nome do controller)
Criar dentro da pasta Users o arquivo index.ctp e apenas digitar nele:
<h1>Index</h1>

Vá até o action index() do controller e comente suas duas linhas, o echo e o exit();

Agora acesse pelo navegador:
http://localhost/cake/scratch/users/
ou
http://localhost/cake/scratch/users/index

Nos mostra a index e a página já vem com um cabeçalho e rodapé.
Veja que automaticamente o action index chama a views index.ctp.

Agora façamos algo mais interessante:

No action index() remova as linhas existentes e adicione estas duas:

```
// Recuperar todos os usuários cadastrados
$users = $this->Users->find('all');
$this->set('users',$users);
```

Assim:

```
public function index()
{
    // Recuperar todos os usuários cadastrados, do método Users do model
    $users = $this->Users->find('all');
    $this->set('users',$users);  // O método set deixa o objeto $users disponível na view index.ctp
}
```

Ou assim, com apenas uma linha, mas prefiro com as duas, mais organizado:
`$this->set('users',$this->Users->find('all'));`

O método set é destinado a armazenar o objeto $users na variável que será recebida pela respectiva view, no caso a index.ctp.

No CakePHP 2 trabalhávamos com arrays e no Cake 3 trabalhamos com objetos.

O Cake 2 enviava a variável contendo um array para a view, já o Cake 3 envia a variável contendo um objeto.

Uma alternativa o index() em apenas uma linha, mas o mesmo código:
`public function index()`

```
{
	$this->set('users', $this->Users->find('all'));
}
```

Paginação de Resultados

Para paginar os resultados no Cake apenas altere a primeira linha do action index() para:

```
	$users = $this->paginate($this->Users);
```

Para configurar a paginação na view usaremos o bake para gerar o código para nós.
Antes elimine o diretório Template\Users ou sobresceva.
bin\cake bake all Users

Assim ele gerou um aplicativo básico mas completo.
Poderemos ver no Template\Users\index.ctp ao ser visualizado no navegador que ele implementou a paginação.
Mesmo que tenha poucos registros, ainda assim poderá ver abaixo os links <previous e next>.

Agora nós podemos ver um código completo do aplicativo.
Veja que ele implementou uma grande quantidade de bons recursos: paginação, ordenação de colunas, template com CSS, menu lateral, etc.

Criação de view index.ctp

Crie o diretório src/Template/Users e dentro dele a index.ctp
Agora vamos melhorar a view src/Template/Users/index.ctp para receber as informações do index(), encluindo a linha digitada e adicionando estas:

```
<ul>
	<?php foreach($users as $user){?>
	<li>
		<?php print $user->username; ?>
	<?php }?>
	</li>
</ul>
```

Outra

```
<table>
	<?php foreach($users as $user){?>
	<tr>
		<td><?php print $user->username; ?></td><td><?php print $user->role; ?
></td>
	</tr>
	<?php }?>
</table>
```

Versão melhorada:

```
<table>
    <?php foreach ($users as $user): ?>
    <tr>
        <td><?= $this->Number->format($user->id) ?></td>
        <td><?= h($user->username) ?></td>
        <td><?= $user->has('group') ? $this->Html->link($user->group->name, ['controller' =>
'Groups', 'action' => 'view', $user->group->id]) : '' ?></td>
                    <td><?= h($user->role) ?></td>
        <td><?= h($user->created) ?></td>
        <td><?= h($user->modified) ?></td>
        <td>
            <?= $this->Html->link(__('View'), ['action' => 'view', $user->id]) ?>
            <?= $this->Html->link(__('Edit'), ['action' => 'edit', $user->id]) ?>
            <?= $this->Form->postLink(__('Delete'), ['action' => 'delete', $user->id], ['confirm'
=> __('Tem certeza de que deseja excluir # '.$user->username.'?', $user->id)]) ?>
        </td>
    </tr>
    <?php endforeach; ?>
</table>
```

Na versão criada pelo bake (remova password, created e modified).

Criar o action add()

Adicionar abaixo do view() no controller UsersController:

```
        public function add()
        {
                $user = $this->Users->newEntity();
                $this->set(compact('user'));
        }
```

Adicionar a view add.ctp ao src\Template\Users contendo:

Usaremos o Helper Form e alguns métodos: create, input, button e end

```
<h2>Novo Usuário</h2>

<?php
        echo $this->Form->create($user);
        echo $this->Form->input('username');
        echo $this->Form->input('password');
        echo $this->Form->input('role',
['options'=>['admin'=>'Administrador','user'=>'Usuário']]);
        echo $this->Form->input('created');
        echo $this->Form->input('modified');
        echo $this->Form->button('Criar Usuário');
        echo $this->Form->end();
```

Outra alternativa (do tutorial Blog):

```
<div class="users form">
<?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Adicionar Usuário') ?></legend>
        <?= $this->Form->input('username') ?>
        <?= $this->Form->input('password') ?>
        <?= $this->Form->input('role', ['options' => ['admin' => 'Admin', 'author' => 'Author']]) ?
>
        <?= $this->Form->input('created') ?>
        <?= $this->Form->input('modified') ?>
    </fieldset>
<?= $this->Form->button(__('Adicionar')); ?>
<?= $this->Form->end() ?>
</div>
```

Melhorando add()
Voltando ao action add(), vamos adicionar uma linha entre as duas existentes:

```
debug($this->request->data);
```

Ficará assim:

```
        public function add()
        {
                $user = $this->Users->newEntity();
                debug($this->request->data);
                $this->set(conpact('user'));
        }
```

Se executar agora o método add, preencher o formulário e clicar no submit verá um array acima com todos os campos e seus valores, resultante
da função debug.

Comentemos a linha do debug e criemos outra logo abaixo, deixando assim:

```
        public function add()
        {
        $user = $this->Users->newEntity();
        if ($this->request->is('post')) {
            $user = $this->Users->patchEntity($user, $this->request->data);
            if ($this->Users->save($user) && ($user->username != null)) {
                $this->Flash->success(__('O usuário foi criado corretamente.'));
                return $this->redirect(['action' => 'index']);
            }
            else
            {
                    $this->Flash->error(__('O usuário não pode ser criado. Username, senha e
role são requeridos. Tente novamente!'));
            }
        }
        $this->set('user', $user);
    }
```

O método patchEntity valida os dados do registro
Assim nosso novo usuário pode ser cadastrado corretamente, ou quase, visto que falta criptografar sua senha.

bin\cake bake template Users add

Para ver como é o código padrão.

Adicionando Criptografia nas Senhas no Model Entity

Isso será feito no Model, especificamente na classe em
src\Model\Entity\User.php

Vamos usar o bake para criar:
bin\cake bake model Users

Edite o arquivo acima e...
logo abaixo da linha:
use Cake\ORM\Entity;

Digite
use Cake\Auth\DefaultPasswordHasher;

A versão 3 do Cake usa o algoritmo bCrypt para a criptografia de senhas.
Adicionar ao final da classe User o método abaixo:

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

Assim nossos usuários serão corretamente salvos com senha criptografada.

As mensagens mostradas nas views do Cake são criadas pela classe flash e seus métodos. Seu código fica no diretório:
src/Template/Element/Flash

Podemos criar mensagens personalizadas, para tanto criar um arquivo no diretório Flash acima com as nossas preferências.
Podemos copiar um dos arquivos error.ctp ou success.ctp e renomear para o nosso e mudar seu conteúdo e passar a usar na chamada da classe Flash.

Action view() simples:

```
public function view($id)
{
    $user = $this->Users->get($id);
    $this->set(compact('user'));
}
```

View view.ctp

Simples

```
<h1>Usuário</h1>
    <table class="vertical-table">
                    <tr><td><?= $this->Number->format($user->id) ?></td></tr>
        <tr><td><?= h($user->username) ?></td></tr>
        <tr><td><?= h($user->created) ?></td></tr>
        <tr><td><?= h($user->modified) ?></td></tr>
    </table>
```

Agora crie uma view pelo bake e sobrescreva a existente para comparar os códigos.
bin\cake bake template Users view

Gera o código do menu lateral dentro da tag <nav> e abaixo mostrando os campos dentro de uma tabela).

Action edit()

Código simples:

```
    public function edit($id = null)
    {
        $user = $this->Users->get($id);
        $user = $this->Users->patchEntity($user, $this->request->data);
        $this->set(compact('user'));
    }
```

Código gerando com o bake:
bin\cake bake controller Users

```
    public function edit($id = null)
    {
        $user = $this->Users->get($id, [
            'contain' => []
        ]);
        if ($this->request->is(['patch', 'post', 'put'])) {
            $user = $this->Users->patchEntity($user, $this->request->data);
            if ($this->Users->save($user)) {
                $this->Flash->success(__('The user has been saved.'));
                return $this->redirect(['action' => 'index']);
            } else {
                $this->Flash->error(__('The user could not be saved. Please, try again.'));
            }
        }
        $this->set(compact('user'));
        $this->set('_serialize', ['user']);
    }
```

Criação da view edit.ctp

Código simples

```
    <?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Edit User') ?></legend>
        <?php
            echo $this->Form->input('username');
            echo $this->Form->input('password');
            echo $this->Form->input('role');
        ?>
    </fieldset>
    <?= $this->Form->button(__('Submit')) ?>
    <?= $this->Form->end() ?>
```

Código gerado pelo bake:

```
<nav class="large-3 medium-4 columns" id="actions-sidebar">
    <ul class="side-nav">
        <li class="heading"><?= __('Actions') ?></li>
        <li><?= $this->Form->postLink(
                __('Delete'),
                ['action' => 'delete', $user->id],
                ['confirm' => __('Are you sure you want to delete # {0}?', $user->id)]
            )
        ?></li>
        <li><?= $this->Html->link(__('List Users'), ['action' => 'index']) ?></li>
    </ul>
</nav>
<div class="users form large-9 medium-8 columns content">
    <?= $this->Form->create($user) ?>
    <fieldset>
        <legend><?= __('Edit User') ?></legend>
        <?php
            echo $this->Form->input('username');
            echo $this->Form->input('password');
            echo $this->Form->input('role');
        ?>
    </fieldset>
    <?= $this->Form->button(__('Submit')) ?>
    <?= $this->Form->end() ?>
</div>
```

Criando o action delete():

```
public function delete($id = null)
{
    $this->request->allowMethod(['post', 'delete']);
    $user = $this->Users->get($id);
    return $this->redirect(['action' => 'index']);
}
```

View delete
Para delete não temos view, será usado apenas um link na view index.ctp para cada registro.

Crédito
Adaptado do vídeo do Edson (http://www.edsonmm.com/), do tutorial do Blog http://book.cakephp.org/3.0/en/tutorials-and-examples/blog-auth-example/auth.html e de código gerado pelo bake.