Zend Framework

Documentação

https://docs.zendframework.com

Versão 3.0

```
curl -s http://getcomposer.org/installer | php
php composer.phar create-project -s dev
zendframework/skeleton-application zf3
```

Responder *n* para a pergunta:

```
Do you want a minimal install (no optional packages)?
Y/n
```

E *y* para as demais. Desta forma instalamos todos os componentes do framework para podermos testá-los.

Para a pergunta:

```
Please select which config file you wish to inject
'ZendDeveloperTools' into:
```

Vamos escolher a opção referente ao *config/modules.config.php* e responder *y* para a próxima pergunta:

```
Remember this option for other packages of the same
type? (y/N)
```

Vamos também instalar uma dependência extra necessária para o cache e o ZendDeveloperTools usando o comando:

```
cd zf3
php ../composer.phar require zendframework/zend-
serializer
```

E também vamos incluir a configuração no *config/modules.config.php*

Configuração do ambiente de desenvolvimento

Renomear o config/development.config.php.dist para config/development.config.php

Testando

```
php -S localhost:8000 -t public/
```

Modelos

Copiar o beers.db de http://cl.ly/2e473b2M2k1Z e salvar no diretório *data* do projeto

Criar o module/Application/src/Model/Beer.php

```php
<?php
namespace Application\Model;

use Zend\InputFilter\InputFilter;

class Beer
{
    public $id;
    public $name;
    public $style;
    public $img;

    /**
     * Configura os filtros dos campos da classe
     *
     * @return Zend\InputFilter\InputFilter
     */
    public function getInputFilter()
    {
        $inputFilter = new InputFilter();

        $inputFilter->add(array(
            'name'     => 'id',
            'required' => false,
            'filters'  => array(
```

```
            array('name' => 'Int'),
        ),
    ));

    $inputFilter->add(array(
        'name'     => 'name',
        'required' => true,
        'filters'  => array(
            array('name' => 'StripTags'),
            array('name' => 'StringTrim'),
        ),
        'validators' => array(
            array(
                'name'    => 'StringLength',
                'options' => array(
                    'encoding' => 'UTF-8',
                    'min'      => 1,
                    'max'      => 100,
                ),
            ),
        ),
    ));

    $inputFilter->add(array(
        'name'     => 'style',
        'required' => true,
        'filters'  => array(
            array('name' => 'StripTags'),
            array('name' => 'StringTrim'),
        ),
        'validators' => array(
            array(
                'name'    => 'StringLength',
                'options' => array(
```

```
                        'encoding' => 'UTF-8',
                        'min'      => 1,
                        'max'      => 100,
                    ),
                ),
            ),
        ));

        $inputFilter->add(array(
            'name'     => 'img',
            'required' => false,
            'filters'  => array(
                array('name' => 'StripTags'),
                array('name' => 'StringTrim'),
            ),
        ));

        return $inputFilter;
    }
}
```

Configurando

Alterar o config/autoload/global.php:

```
return [
    'service_manager' => [
        'factories' => [
            Application\Model\BeerTableGateway::class
=>  Application\Factory\BeerTableGateway::class,
```

```
            Application\Factory\DbAdapter::class =>
Application\Factory\DbAdapter::class,
        ],
    ],
    'db' => [
        'driver' => 'Pdo_Sqlite',
        'database' => 'data/beers.db',
    ],
];
```

Criar o module/Application/src/Factory/DbAdapter.php com:

```
<?php

namespace Application\Factory;

use Interop\Container\ContainerInterface;
use Zend\Db\Adapter\Adapter as ZendAdapter;

class DbAdapter
{
    public function __invoke(ContainerInterface
$container)
    {
        $config = $container->get('config');
        return new ZendAdapter($config['db']);
    }
}
```

Criar o module/Application/src/Factory/BeerTableGateway.php:

```php
<?php

namespace Application\Factory;

use Interop\Container\ContainerInterface;
use Zend\Db\Adapter\Adapter as ZendAdapter;

class BeerTableGateway
{
    public function __invoke(ContainerInterface
$container)
    {
        $adapter = $container-
>get('Application\Factory\DbAdapter');

        return new
\Zend\Db\TableGateway\TableGateway('beer', $adapter);
    }
}
```

Crud de cervejas

Configurar as rotas

Vamos abrir um pequeno parênteses aqui, e comentar sobre como os controladores e *actions* funcionam. Geralmente os controladores são classes com o nome terminando em *Controller* como o *PostController* (apesar disso não ser mais obrigatório a partir do Zend Framework 2 ainda continua-se usando esse padrão). Cada controlador possui uma ou mais *actions* que são métodos públicos cujo nome termina com

Action como o *indexAction*. As *actions* são as ações que os usuários podem acessar via URL, links ou botões na tela. Por exemplo, caso o usuário acesse a url:

```
http://beer.dev/admin/index/index/id/1
```

Isto é traduzido pelo framework usando o padrão:

```
http://servidor/modulo/controller/action/parametro/valor
```

Então:

- Servidor = *beer.dev*
- Módulo = *Admin*
- Controller = *IndexController.php*
- Action = *indexAction* (dentro do arquivo *IndexController.php*)
- Parâmetro = *id*
- Valor = 1

Este é o comportamento padrão esperado pelo framework mas nós podemos criar as nossas rotas da melhor forma que nosso projeto necessitar. Vamos criar as seguintes rotas para nosso pequeno projeto:

- /beer: vai ser a lista de posts. A lógica vai estar no *indexAction*
- /beer/create: vai ser a inclusão de cervejas. A lógica vai estar no *createAction*
- /beer/edit/NUMERO: vai ser a edição de cerveja. A lógica vai estar no *editAction*
- /beer/delete/NUMERO: vai ser a exclusão de beer. A lógica vai

estar no *deleteAction*

Para isso vamos criar uma nova chave no array *router* do arquivo *module/Application/config/module.config.php*, logo após a chave *application*:

```
'beer' => [
                'type'    => Segment::class,
                'options' => [
                    'route'    => '/beer[/][:action]
[/:id]',
                    'constraints' => [
                        'action' => '[a-zA-Z][a-zA-Z0-
9_-]*',
                        'id'     => '[0-9]+',
                    ],
                    'defaults' => [
                        'controller' =>
Controller\BeerController::class,
                        'action'     => 'index',
                    ],
                ],
            ],
```

Precisamos incluir o novo Controller na lista de Factories e passar para ele o TableGateway como uma dependência. Ainda no *module.config* alterar:

```
'controllers' => [
        'factories' => [
            Controller\IndexController::class =>
```

```
InvokableFactory::class,
            Controller\BeerController::class =>
function(\Interop\Container\ContainerInterface
$container, $requestedName) {
                $tableGateway = $container-
>get('Application\Model\BeerTableGateway');
                $controller = new
Controller\BeerController($tableGateway);

                return $controller;
            },
        ],
    ],
```

Listando as cervejas

E vamos criar o module/Application/src/Controller/BeerController.php:

```
<?php
namespace Application\Controller;

use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;

class BeerController extends AbstractActionController
{
    public $tableGateway;

    public function __construct($tableGateway)
    {
        $this->tableGateway = $tableGateway;
    }
```

```
    public function indexAction()
    {
        $beers = $this->tableGateway->select()->toArray();

        return new ViewModel(['beers' => $beers]);
    }
}
```

Vamos também criar a primeira view, para mostrar as cervejas em view/application/beer/index.phtml :

```
<a href="/beer/create">New Beer</a>

<table class="table table-striped">
  <thead>
    <tr>
      <th>Id</th>
      <th>Name</th>
      <th>Style</th>
      <th>Img</th>
      <th>Edit</th>
      <th>Delete</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($this->beers as $beer): ?>
      <tr>
        <th scope="row"><?php echo $beer['id'];?></th>
        <td><?php echo $beer['name'];?></td>
        <td><?php echo $beer['style'];?></td>
        <td><img src="<?php echo $beer['img'];?>">
```

```
</td>
        <td><a href="/beer/edit/<?php echo
$beer['id'];?>">Edit</a></td>
        <td><a href="/beer/delete/<?php echo
$beer['id'];?>">Delete</a></td>
      </tr>
    <?php endforeach; ?>
  </tbody>
</table>
```

Removendo uma cerveja

Adicionar um novo método ao BeerController.php:

```
public function deleteAction()
    {
        $id = (int) $this->params()->fromRoute('id');

        $beer = $this->tableGateway->select(['id' =>
$id]);
        if (count($beer) == 0) {
            throw new \Exception("Beer not found",
404);

        }

        $this->tableGateway->delete(['id' => $id]);

        return $this->redirect()->toUrl('/beer');

    }
```