

Zend Framework

Documentação

<https://docs.zendframework.com>

Versão 3.0 (lançada 28/06/2016)

```
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*

Modelos

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

Criar o module/Application/src/Model/Beer.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\Factory\Db\Adapter\Adapter::class =>
            Application\Factory\Db\Adapter\Adapter::class,
            Application\Model\Beer\TableGateway::class
            =>
            Application\Factory\Model\Beer\TableGateway::class,
        ],
    ],
    'db' => [

```

```
        'driver' => 'Pdo_Sqlite',  
        'database' => 'beers.db',  
    ],  
];
```

Criar o module/Application/src/Factory/Db/Adapter/Adapter.php com:

```
<?php  
  
namespace Application\Factory\Db\Adapter;  
  
use Interop\Container\ContainerInterface;  
use Zend\Db\Adapter\Adapter as ZendAdapter;  
  
class Adapter  
{  
    public function __invoke(ContainerInterface  
$container)  
    {  
        $config = $container->get('config');  
        return new ZendAdapter($config['db']);  
    }  
}
```

Criar o
module/Application/src/Factory/Model/Beer/TableGateway.php:

```
<?php
```

```
namespace Application\Factory\Model\Beer;

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

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

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

Testando

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

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 no 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\Beer\TableGateway' );
            $controller = new
Controller\BeerController($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>
</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' );  
    }  
}
```

Adicionando uma cerveja

Vamos começar criando um formulário para a inclusão da cerveja. No module/Application/src/Form/Beer.php:

```
<?php  
  
namespace Application\Form;  
  
use Zend\Form\Element;  
use Zend\Form\Form;  
  
class Beer extends Form  
{  
    public function __construct()  
    {  
        parent::__construct();  
  
        $this->add([  
            'name' => 'name',  
            'options' => [  
                'label' => 'Beer name',  
            ],  
            'type' => 'Text',  
        ]);  
        $this->add([  
            'name' => 'style',  
            'options' => [  
                'label' => 'Beer style',  
            ],  
            'type' => 'Text',  
        ]);  
    }  
}
```

```

        ],
        'type' => 'Text',
    ]);

    $this->add([
        'name' => 'img',
        'options' => [
            'label' => 'Beer image',
        ],
        'type' => 'Text',
    ]);

    $this->add([
        'name' => 'send',
        'type' => 'Submit',
        'attributes' => [
            'value' => 'Submit',
        ],
    ]);

    $this->setAttribute('action', '/beer/save');
    $this->setAttribute('method', 'post');
}
}

```

Vamos alterar o BeerController.php para incluir o createAction e mostrar o form:

```

public function createAction()
{
    $form = new \Application\Form\Beer;

```

```

$view = new ViewModel([ 'form' => $form]);
$view->setTemplate( 'application/beer/save.phtml' );

return $view;
}

```

E criar a view em view/application/beer/save.phtml:

```

<?php echo $this->form()->openTag($form); ?>
<div class="form_element">
    <?php echo $this->formRow($form->get('name')); ?>
</div>

<div class="form_element">
    <?php echo $this->formRow($form->get('style')); ?>
</div>

<div class="form_element">
    <?php echo $this->formRow($form->get('img')); ?>
</div>

<?php if ($form->has('id')): ?>
    <?php echo $this->formElement($form->get('id')) ?>
<?php endif;?>

<?php echo $this->formElement($form->get('send')) ?>

<?php echo $this->form()->closeTag() ?>

```

Outra forma de mostrar o form, mas mais simples poderia ser:

```
<?php
echo $this->form()->openTag($form);
echo $this->formCollection($form);
echo $this->form()->closeTag();
?>
```

Vamos alterar novamente o createAction para realizar a lógica da adição da nova cerveja, incluindo a validação dos campos:

```
public function createAction()
{
    $form = new \Application\Form\Beer;
    $form->setAttribute('action', '/beer/create');
    $request = $this->getRequest();
    /* se a requisição é post os dados foram enviados
    via formulário*/
    if ($request->isPost()) {
        $beer = new \Application\Model\Beer;
        /* configura a validação do formulário com os
        filtros e validators da entidade*/
        $form->setInputFilter($beer-
>getInputFilter());
        /* preenche o formulário com os dados que o
        usuário digitou na tela*/
        $form->setData($request->getPost());
        /* faz a validação do formulário*/
        if ($form->isValid()) {
            /* pega os dados validados e filtrados */
            $data = $form->getData();
            unset($data['send']);
            /* salva a cerveja*/
            $this->tableGateway->insert($data);
        }
    }
}
```



```

        /* redireciona para a página inicial que
        mostra todas as cervejas*/
        return $this->redirect()->toUrl('/beer');
    }
}
$view = new ViewModel(['form' => $form]);
$view->setTemplate('application/beer/save.phtml');

return $view;
}

```

Alterando uma cerveja

Vamos incluir o editAction para que ele possa suportar a atualização da cerveja e reaproveitarmos o form:

```

public function editAction()
{
    /* configura o form */
    $form = new \Application\Form\Beer;
    $form->get('send')->setAttribute('value', 'Edit');
    $form->setAttribute('action', '/beer/edit');
    /* adiciona o ID ao form */
    $form->add([
        'name' => 'id',
        'type'  => 'hidden',
    ]);
    $view = new ViewModel(['form' => $form]);
    $view->setTemplate('application/beer/save.phtml');

    $request = $this->getRequest();
    /* se a requisição é post os dados foram enviados
    via formulário*/

```

```
if ($request->isPost()) {
    $beer = new \Application\Model\Beer;
    /* configura a validação do formulário com os
    filtros e validators da entidade*/
    $form->setInputFilter($beer-
>getInputFilter());
    /* preenche o formulário com os dados que o
    usuário digitou na tela*/
    $form->setData($request->getPost());
    /* faz a validação do formulário*/
    if (!$form->isValid()) {
        return $view;
    }
    /* pega os dados validados e filtrados */
    $data = $form->getData();
    unset($data['send']);
    /* salva a cerveja*/
    $this->tableGateway->update($data, 'id =
' . $data['id']);
    /* redireciona para a página inicial que
    mostra todas as cervejas*/
    return $this->redirect()->toUrl('/beer');
}

/* Se não é post deve mostrar os dados */
$id = (int) $this->params()->fromRoute('id', 0);
$beer = $this->tableGateway->select(['id' =>
$id])->toArray();
if (count($beer) == 0) {
    throw new \Exception("Beer not found", 404);
}

/* preenche o formulário com os dados do banco
de dados */
```

```
$form->get( 'id' )->setValue( $beer[0][ 'id' ] );
$form->get( 'name' )->setValue( $beer[0][ 'name' ] );
$form->get( 'style' )->setValue( $beer[0][ 'style' ] );
$form->get( 'img' )->setValue( $beer[0][ 'img' ] );

return $view;
}
```

Autenticação

Vamos usar o conceito de EventManager para criar um evento e proteger a ação de excluir uma cerveja. No arquivo module/Application/src/Module.php vamos incluir:

```
public function onBootstrap($e)
{
    $eventManager = $e->getApplication()-
>getServiceManager()->get( 'EventManager' );
    $eventManager->getSharedManager()-
>attach( 'Zend\Mvc\Controller\AbstractActionController'
, \Zend\Mvc\MvcEvent::EVENT_DISPATCH, [ $this,
'mvcPreDispatch' ], 100 );
}

public function mvcPreDispatch($event)
{
    $routeMatch = $event->getRouteMatch();
    $moduleName = $routeMatch->getParam( 'module' );
    $controllerName = $routeMatch-
>getParam( 'controller' );
}
```

```

        $actionName = $routeMatch->getParam('action');
        if ($controllerName ==
'Application\Controller\BeerController' && $actionName
== 'delete') {
            $authService = $event->getApplication()-
>getServiceManager()->get('Application\Service\Auth');
            if (!$authService->isAuthorized()) {
                $redirect = $event->getTarget()-
>redirect();
                $redirect->toUrl('/');
            }
        }
    }
}

```

Precisamos incluir o novo serviço no config/global.php:

```

return [
    'service_manager' => [
        'factories' => [

Application\Factory\Db\Adapter\Adapter::class =>
Application\Factory\Db\Adapter\Adapter::class,
                Application\Model\Beer\TableGateway::class
=>
Application\Factory\Model\Beer\TableGateway::class,
                Application\Service\Auth::class =>
Application\Factory\Service\Auth::class,
            ],
        ],
    'db' => [
        'driver' => 'Pdo_Sqlite',
        'database' => 'beers.db',
    ],
];

```

```
    ],  
];
```

E criar a Factory e o Service. A factory em
module/Application/src/Factory/Service/Auth.php:

```
<?php  
  
namespace Application\Factory\Service;  
  
use Interop\Container\ContainerInterface;  
  
class Auth  
{  
    public function __invoke(ContainerInterface  
$container)  
    {  
        $adapter = $container-  
>get( 'Application\Factory\Db\Adapter\Adapter' );  
        $request = $container->get( 'Request' );  
  
        return new \Application\Service\Auth($request,  
$adapter);  
    }  
}
```

E o serviço em module/Application/src/Service/Auth.php:

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

```
class Auth
{
    private $request;
    private $adapter;

    public function __construct($request, $adapter)
    {
        $this->request = $request;
        $this->adapter = $adapter;
    }

    public function isAuthenticated()
    {
        if(! $this->request->getHeader('authorization')){
            throw new \Exception("Not authorized",
401);
        }

        if (!$this->isValid()) {
            throw new \Exception("Not authorized",
403);
        }

        return true;
    }

    private function isValid()
    {
        $token = $this->request->getHeader('authorization');
        //validar o token de alguma forma...
        return true;
    }
}
```

```
}  
  
}
```

Cache

Neste exemplo vamos usar Cache para armazenar os dados da nossa base de dados e economizar consultas ao banco.

Vamos incluir a configuração do cache no config/autoload/global.php:

```
'cache' => [  
    'adapter' => [  
        'name' => 'apc',  
        'options' => ['ttl' => 3600],  
    ],  
    'plugins' => [  
        'exception_handler' => ['throw_exceptions'  
=> false],  
        'serializer',  
    ],  
],
```

Mais informações sobre os tipos de adapter estão na documentação oficial: <https://zendframework.github.io/zend-cache/storage/adapter/>

Para facilitar os testes vamos criar uma configuração local diferente, em config/autoload/local.php:

```

return [
    'cache' => [
        'adapter' => [
            'name' => 'filesystem',
            'options' => ['ttl' => 3600, 'namespace'
=> 'dbtable', 'cache_dir' => 'data/cache'],
        ],
        'plugins' => [
            // Don't throw exceptions on cache errors
            'exception_handler' => [
                'throw_exceptions' => true
            ],
            // We store database rows on filesystem so
we need to serialize them
            'serializer',
        ],
    ],
];

```

Ainda no config/autoload/global.php precisamos incluir a factory do cache:

```

'service_manager' => [
    'factories' => [

Application\Factory\Db\Adapter\Adapter::class =>
Application\Factory\Db\Adapter\Adapter::class,
        Application\Model\Beer\TableGateway::class
=>
Application\Factory\Model\Beer\TableGateway::class,
        Application\Service\Auth::class =>
Application\Factory\Service\Auth::class,

```



```
        'Application\Service\Cache' =>
Application\Factory\Service\Cache::class,
    ],
],
```

E criar o arquivo em
module/Application/src/Factory/Service/Cache.php:

```
<?php

namespace Application\Factory\Service;

use Interop\Container\ContainerInterface;
use Zend\Cache\StorageFactory;

class Cache
{
    public function __invoke(ContainerInterface
$container)
    {
        $config = $container->get('Config');
        return
StorageFactory::factory($config['cache']);
    }
}
```

Vamos injetar o serviço de cache no nosso controller. No
Application/config/module.config.php vamos alterar a factory:

```
'controllers' => [
    'factories' => [
```

```

        Controller\IndexController::class =>
    InvokableFactory::class,
        Controller\BeerController::class =>
function(\Interop\Container\ContainerInterface
$container, $requestedName) {
            $tableGateway = $container->
>get( 'Application\Model\Beer\TableGateway' );
            $cache = $container->
>get( 'Application\Service\Cache' );
            $controller = new
Controller\BeerController($tableGateway, $cache);

            return $controller;
        },
    ],
],

```

E nosso controller agora precisa receber o cache para poder usá-lo. O novo BeerController.php ficou desta forma:

```

<?php
namespace Application\Controller;

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

class BeerController extends AbstractActionController
{
    public $tableGateway;
    public $cache;

    public function __construct($tableGateway, $cache)

```

```
{
    $this->tableGateway = $tableGateway;
    $this->cache = $cache;
}

public function indexAction()
{
    $key      = 'beers';
    $beers    = $this->cache->getItem($key,
$success);
    if (! $success) {
        $beers = $this->tableGateway->select()-
>toArray();
        $this->cache->setItem($key, $beers);
    }

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

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]);
    $this->cache->removeItem('beers');
    return $this->redirect()->toUrl('/beer');
```

```

}

public function createAction()
{
    $form = new \Application\Form\Beer;
    $form->setAttribute('action', '/beer/create');
    $request = $this->getRequest();
    /* se a requisição é post os dados foram
    enviados via formulário*/
    if ($request->isPost()) {
        $beer = new \Application\Model\Beer;
        /* configura a validação do formulário com
        os filtros e validators da entidade*/
        $form->setInputFilter($beer-
>getInputFilter());
        /* preenche o formulário com os dados que
        o usuário digitou na tela*/
        $form->setData($request->getPost());
        /* faz a validação do formulário*/
        if ($form->isValid()) {
            /* pega os dados validados e filtrados
            */

            $data = $form->getData();
            unset($data['send']);
            /* salva a cerveja*/
            $this->tableGateway->insert($data);
            /* redireciona para a página inicial
            que mostra todas as cervejas*/
            return $this->redirect()-
>toUrl('/beer');
        }
    }
    $view = new ViewModel(['form' => $form]);
    $view-

```

```

>setTemplate('application/beer/save.phtml');

    return $view;
}

public function editAction()
{
    /* configura o form */
    $form = new \Application\Form\Beer;
    $form->get('send')->setAttribute('value',
'Edit');
    $form->setAttribute('action', '/beer/edit');
    /* adiciona o ID ao form */
    $form->add([
        'name' => 'id',
        'type'  => 'hidden',
    ]);
    $view = new ViewModel(['form' => $form]);
    $view->

>setTemplate('application/beer/save.phtml');

    $request = $this->getRequest();
    /* se a requisição é post os dados foram
enviados via formulário*/
    if ($request->isPost()) {
        $beer = new \Application\Model\Beer;
        /* configura a validação do formulário com
os filtros e validators da entidade*/
        $form->setInputFilter($beer->
>getInputFilter());
        /* preenche o formulário com os dados que
o usuário digitou na tela*/
        $form->setData($request->getPost());
        /* faz a validação do formulário*/
    }
}

```

```
        if (!$form->isValid()) {
            return $view;
        }
        /* pega os dados validados e filtrados */
        $data = $form->getData();
        unset($data['send']);
        /* salva a cerveja */
        $this->tableGateway->update($data, 'id =
'. $data['id']);
        /* redireciona para a página inicial que
mostra todas as cervejas */
        return $this->redirect()->toUrl('/beer');
    }

    /* Se não é post deve mostrar os dados */
    $id = (int) $this->params()-
>fromRoute('id', 0);
    $beer = $this->tableGateway->select(['id' =>
$id])->toArray();
    if (count($beer) == 0) {
        throw new \Exception("Beer not found",
404);
    }

    /* preenche o formulário com os dados do
banco de dados */
    $form->get('id')->setValue($beer[0]['id']);
    $form->get('name')->setValue($beer[0]
['name']);
    $form->get('style')->setValue($beer[0]
['style']);
    $form->get('img')->setValue($beer[0]['img']);

    return $view;
```

```
}  
  
}
```

Alteramos o construtor da classe e o método `indexAction` para usar o cache e armazenar o conteúdo da tabela de acordo com a configuração do adapter de cache.

Trabalho

- Fazer Controller para login, recebendo email e senha e usando um Form
- Validar em um banco de dados.
- Se login válido redirecionar para `/beers`, se login inválido retornar mensagem de erro.
- Sugestões: pesquisar sobre o componente `zend-session`