Frameworks de Desenvolvimento PHP

Pós-Graduação em Desenvolvimento Web,
Cloud e Dispositivos Móveis
Prof. Er **Galvão** Abbott

Table of Contents

Introdução

Frameworks

Frameworks são softwares que auxiliam no desenvolvimento de uma aplicação. Tipicamente frameworks implementam um paradigma de desenvolvimento que normalmente seria complexo de implementar "manualmente".

Entre os padrões está o MVC – Model, View, Controller, que utlizaremos neste curso.

Podemos citar como principais vantanges de se utilizar um Framework:

- Organização – Frameworks normalmente possuem uma estrutura organizacional – pelo menos – sugerida.
- Simplicidade – Frameworks tipicamente possuem componentes prontos para tarefas que seriam ou de dificuldade considerável para serem desenvolvidas manualmente ou extremamente trabalhosas de serem mantidas separadamente da aplicação.
- Padronização – Por possuirem uma estrutura e filosofia de desenvolvimento definidas, os frameworks são extremamente úteis para se adicionar novos profissionais a um projeto.
- Qualidade – Como o framework é desenvolvido de forma completamente separada da aplicação, é possível se ter um considerável nível de confiança no software que provê o embasamento da aplicação.

Entre os principais exemplos de frameworks da linguagem PHP podemos citar:

- Code Igniter
- Laravel
- Symfony
- Zend Framework

Neste curso utilizaremos o Zend Framework para o desenvolvimento de nossa aplicação. O ZF, como é popularmente chamado, é considerado um excelente framework e provê uma série de conceitos que uma vez compreendidos podem ser aplicados a qualquer outro framework que você desejar trabalhar.

MVC – Model, View, Controller

MVC é um paradigma de desenvolvimento em camadas, onde procura-se separar as partes (camadas) que compõem uma aplicação.

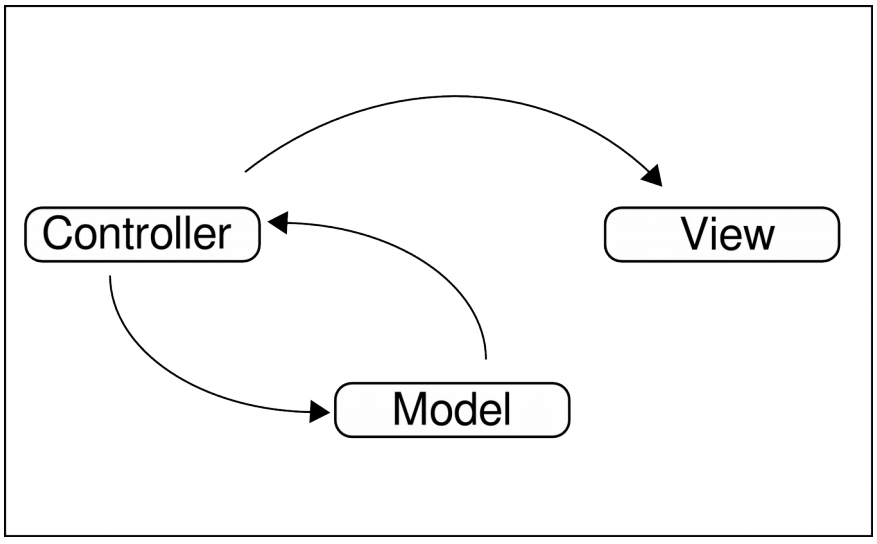
A camada de Model (Modelo) representa os dados, ou seja, frequentemente esta camada representa um banco de dados.

A View (Visão) é a camada de apresentação ou interface e é nela que se encontra a parte visual da aplicação.

A Controller (Controlador), por sua vez, é a camada de processamento, onde se encontra a parte mais pesada de lógica.

A grande vantagem deste paradigma é a separação de código de processamento, dados e código de apresentação, fazendo com que a aplicação se torne mais clara e mais simples de se compreender e manter.

O que um framework MVC faz é implementar esse paradigma, criando a comunicação entre as camadas. No exemplo mais clássico, a Controller se comunica com a Model para, por exemplo, obter dados. A Model devolve estes dados para a Controller que, por sua vez, envia-os para a View para exibição, como mostra a figura na página a seguir.

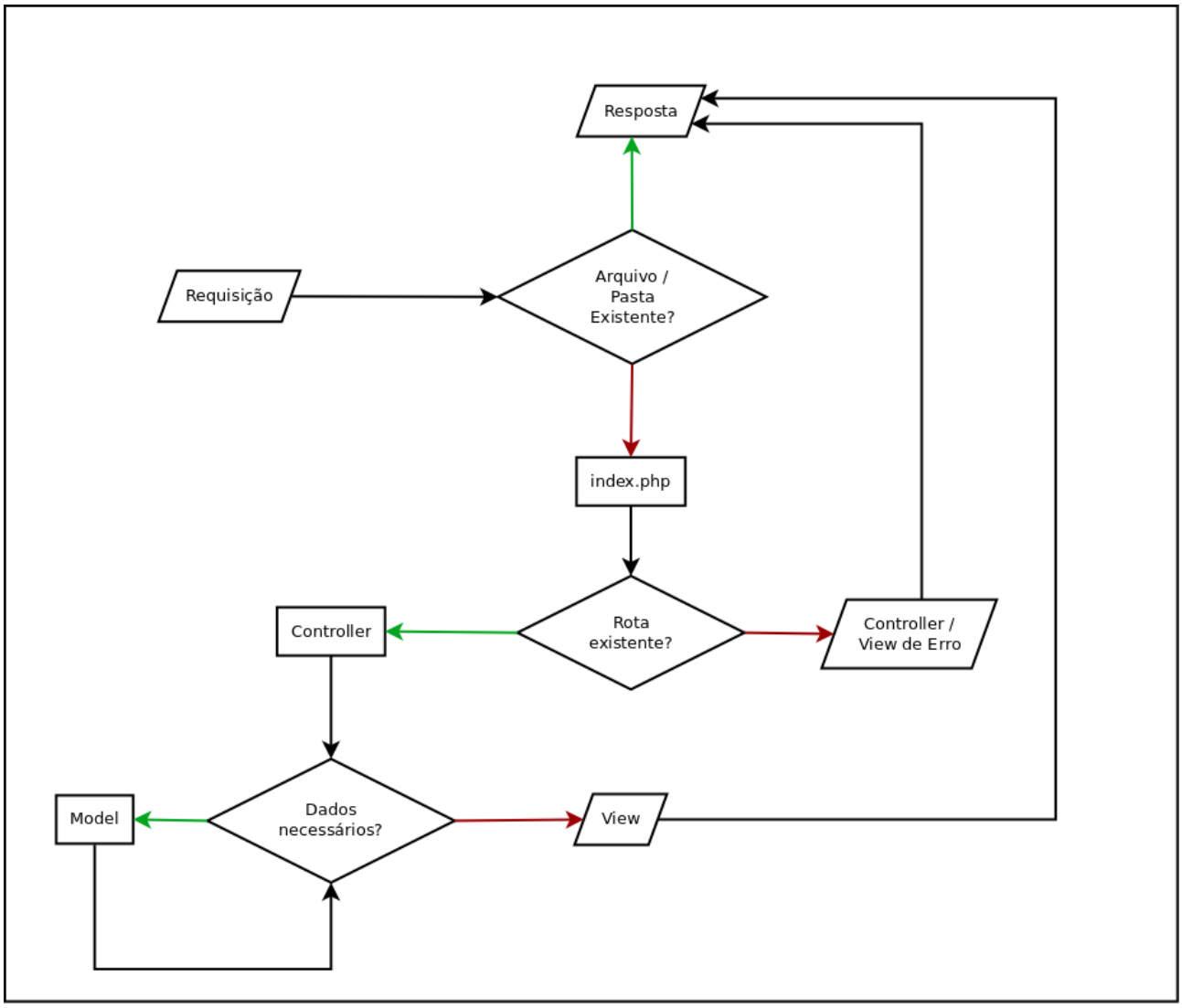


O padrão Front Controller

Assim como ocorre com a maioria dos frameworks disponíveis no mercado de desenvolvimento web, o ZF utiliza-se de um padrão chamado Front Controller para prover o funcionamento da aplicação.

O funcionamento básico deste padrão faz com que todas* as requisições sejam propositalmente recebidas pelo mesmo arquivo PHP, tipicamente o único arquivo PHP disponível na raiz web do servidor. A partir da execução deste arquivo é determinado se existe uma rota condizente com o endereço requisitado. Em caso positivo, uma Controller é executada e ao final da execução o conteúdo de uma View processada é retornado como resposta a requisição.

A figura a seguir ilustra o padrão Front Controller:



Instalando o ZF

Pré-Requisitos

- Servidor Web Apache
  - Módulo mod_rewrite
  - Com suporte a arquivos .htaccess
- PHP >= 5.6
  - PDO com suporte ao SGBD desejado (no caso deste curso, MySQL/MariaDB)
- Composer

Instalando a Aplicação-Esqueleto

Ao se iniciar uma aplicação com ZF tipicamente instalamos primeiramente o que chamamos de Skeleton (Esqueleto), uma estrutura inicial que é considerada comum para qualquer aplicação web. Para criarmos o projeto através da Skeleton rodamos o composer:

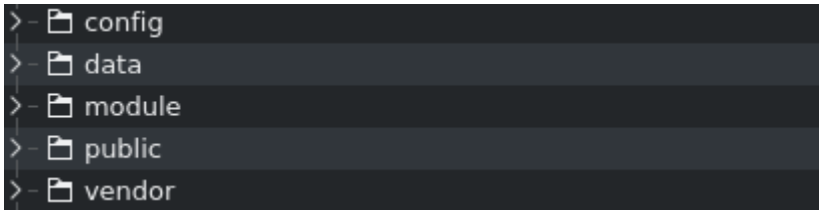
```
1 composer create-project -s dev zendframework/skeleton-application /caminho
```

O ZF é um framework com baixo nível de acoplamento, portanto é possível fazer uma instalação mínima e ir adicionando novos componentes posteriormente. O nível de acoplamento é tão baixo que é possível até mesmo instalarmos componentes pertencentes a outros frameworks ou mesmo independentes.

Ainda assim, por razões óbvias de comodidade é preferível realizar uma instalação o mais completa possível. Veremos em sala de aula como realizar a instalação completa.

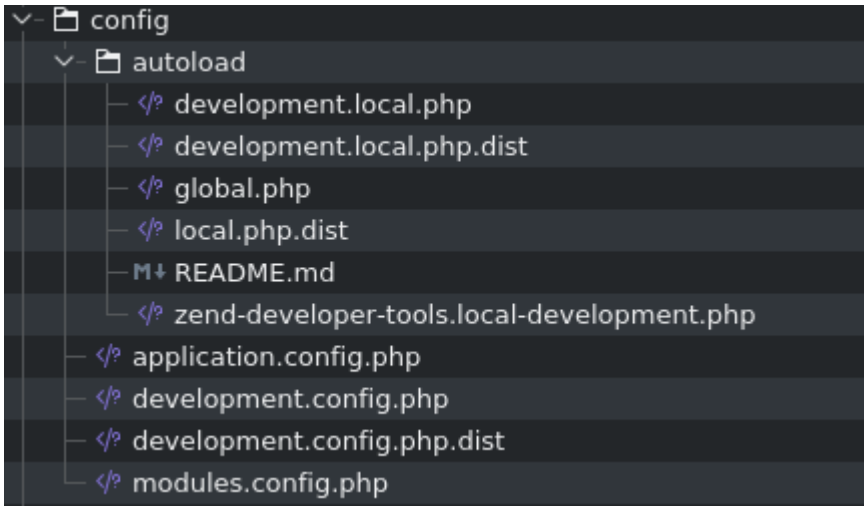
Estrutura Básica

Ao realizarmos uma instalação completa da Skeleton obtemos a seguinte estrutura básica de aplicação:



Config

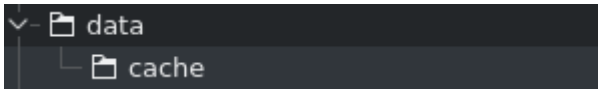
Como o próprio nome sugere, esta é a pasta de configuração. Nela encontramos os arquivos que gerem o funcionamento da aplicação baseada em ZF, bem como configurações específicas de cada ambiente.



A sub-pasta autoload contém os arquivos que serão carregados por ambiente, seja a máquina do desenvolvedor – [*.]local.php – ou globais independentemente de ambiente – [*.]global.php.

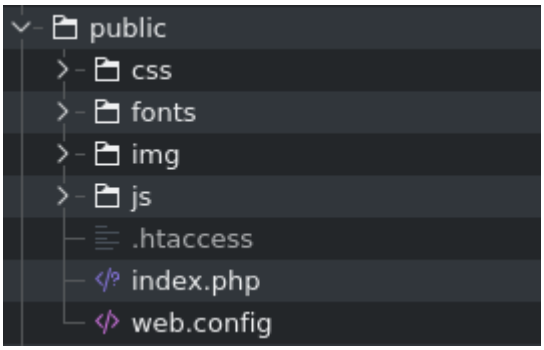
Data

A pasta data é onde guardaremos dados gerados pela aplicação, como arquivos de cache e log, por exemplo.



Public

A pasta public será a raiz web da aplicação. A Skeleton vem, por padrão, com pastas para estilos e scripts client-side, incluindo componentes consolidados, como bootstrap, por exemplo.

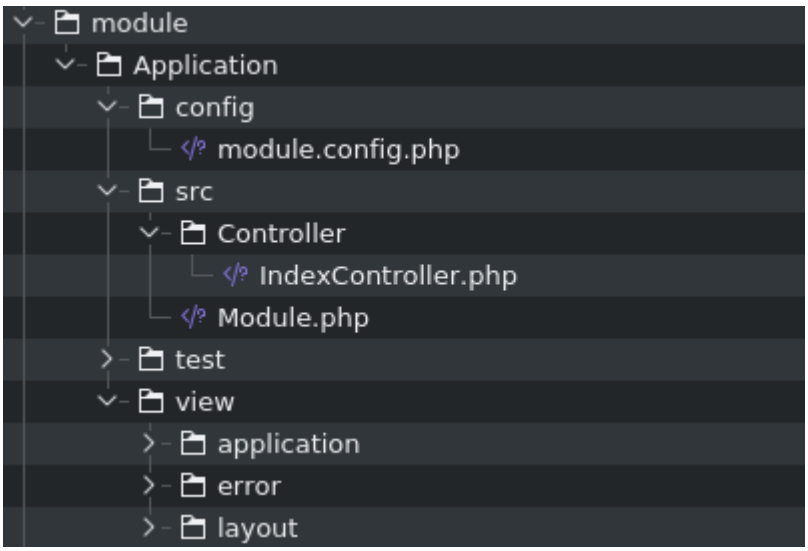


Vendor

A pasta vendor (traduz-se como “fornecedor”) é onde são “instalados” quaisquer componentes que não tenham sido desenvolvidos pelo(a) desenvolvedor(a) / equipe / empresa, o que inclui o próprio ZF.

Module – Onde reside o núcleo da aplicação

A pasta module é onde nossa aplicação de fato reside, mais precisamente na sub-pasta Application.



Detalharemos extensamente esta pasta mais adiante, já que é onde passaremos a maior parte do tempo desenvolvendo.

Passos Finais

Antes de finalmente rodarmos nossa aplicação pela primeira vez executaremos a criação de um VHost (Servidor Virtual) para nossa aplicação. Embora não seja, tecnicamente, um passo obrigatório é prática comum e necessária no mercado.

Detalhes de como criar e ativar o VHost serão fornecidos em sala de aula, mas aqui está, em linhas gerais, o código do arquivo de configuração de nosso VHost:

```
<VirtualHost pos_unoesc:80>
    ServerName pos_unoesc
    DocumentRoot /var/www/html/pos_unoesc_2018/public
    <Directory /var/www/html/pos_unoesc_2018/public>
        DirectoryIndex index.php
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

A partir disso podemos testar nossa aplicação. Abra o navegador e digite: http://pos_unoesc/

Para se certificar de que a aplicação está de fato executando corretamente podemos ainda induzir um erro proposital para verificarmos se a controller e view de erros está sendo de fato executada: http://pos_unoesc/foo

Depois de nos certificarmos de que tudo está funcionando conforme o esperado, faremos uma “limpeza” em alguns arquivos de nossa Skeleton. Isso não apenas nos ajudará a compreender de forma mais simples o funcionamento da mesma, mas também nos permitirá aplicar nossas próprias mudanças.

Rotas

Assim como a maioria dos frameworks de desenvolvimento web, o ZF implementa URLs customizáveis (“Amigáveis”) que tipicamente possuem um método de uma controller (e, posteriormente, claro, uma view) como responsáveis por “atender” a rota.

Rotas são definidas dentro do arquivo module/application/config/module.config.php e têm como sua forma mais simples as rotas literais.

Em sala de aula: Veremos como criar uma rota literal e como criar a forma mais simples de uma Comtroller e uma View para “responderem” por esta rota.

Controllers

Controllers são, tipicamente, classes que extendem uma classe ZF (como, por exemplo a AbstractActionController) e que possuem pelo menos uma “Action”. Parece complicado? Não é: Uma action nada mais é do que um método da classe controller.

Veja o código mais simples possível para uma controller:

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

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

class IndexController extends AbstractActionController
{
    public function indexAction()
    {
        return new ViewModel();
    }
}
```

Views

A View é a camada mais simples de se trabalhar, pois no caso de uma view estática (onde não há dados dinâmicos), a view termina por ser nada mais nada menos do que código Client-Side (HTML/CSS/JavaScript/etc...).

Interação entre camadas

Controller -> View

Para que haja comunicação entre a camada de controller e a view, começamos fazendo que a controller “envie” dados. Observe a diferença:

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

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

class IndexController extends AbstractActionController
{
    public function indexAction()
    {
        return new ViewModel([
          'nomeFramework' => 'ZF',
        ]);
    }
}
```

Na View, por sua vez, esses dados se tornam automaticamente disponíveis através do objeto ViewModel, que na View é tratado como “$this”:

```
1 <?php echo $this->nomeFramework;?>
```

Model

Configurando o acesso a Base de Dados

As configurações de acesso à base de dados, bem como outras configurações específicas de ambiente, ficam salvas em uma configuração “auto-carregável” (arquivos presentes em raiz_da_aplicação/config/autoload). Todo o acesso à Base de Dados é realizado automaticamente pelo framework quando necessário.

Vejamos um exemplo, trabalhando com um arquivo “development.local.php”:

```php
<?php
return [
    'view_manager' => [
        'display_exceptions' => true,
    ],
    'db' => [
        'driver'   => 'Pdo_Mysql',
        'database' => 'pos_unoesc',
        'username' => 'usuario_bd',
        'password' => 'senha_bd',
        'hostname' => 'localhost',
    ],
];
```

Tudo o que precisamos fazer é prover as configurações do acesso à Base de Dados através de um índice “db” no array de configuração.