PHP Frameworks (21h)

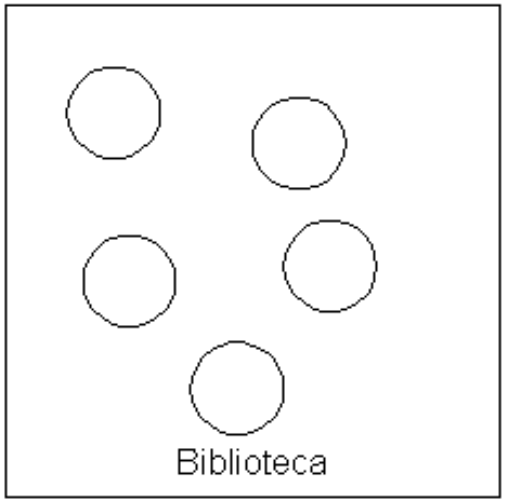
Márcio Pereira – marcio.pereira00@gmail.com

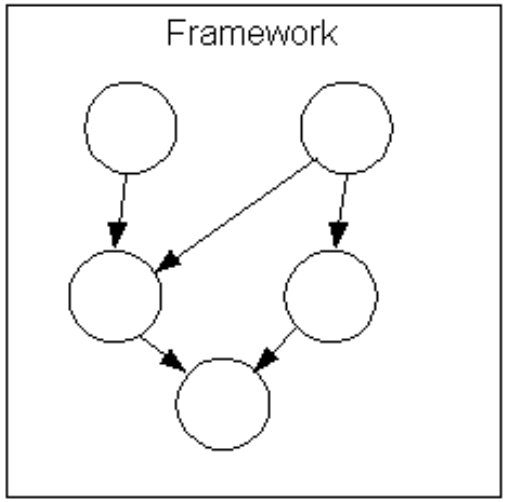
FLAG

- O que é uma framework e uma micro-framework
- Vantagens de usar uma Framework
- Diferenças entre frameworks
- Utilização do composer
- Padrão de desenho MVC
- O que é o controlador
- O que é o modelo
- O que é uma vista
- Criação de uma aplicação PHP usando uma Framework

O que é uma Framework?

É uma estrutura base, que contém um conjunto extensível de classes orientadas a objetos que são integradas para executar conjuntos bem definidos de comportamento computacional.
Esta base pode ser vista como um conjunto de ferramentas, funcionalidades e comportamentos que podem ser usadas pelo programador no desenvolvimento do seu software. Tornando o desenvolvimento de software mais rápido e seguro.





Características Básicas de Frameworks

- **Um framework deve ser reusável**
  - É o principal propósito de uma framework.
  - Para ser reusável, deve primeiro ser *usável*
    - Bem documentado
    - Fácil de usar

- **Deve ser extensível**
  - A framework contém funcionalidade abstrata (sem implementação) que deve ser completada

- **Deve ser de uso seguro**
  - A framework deve ter em atenção todos os aspetos de segurança.

- **Deve ser eficiente**

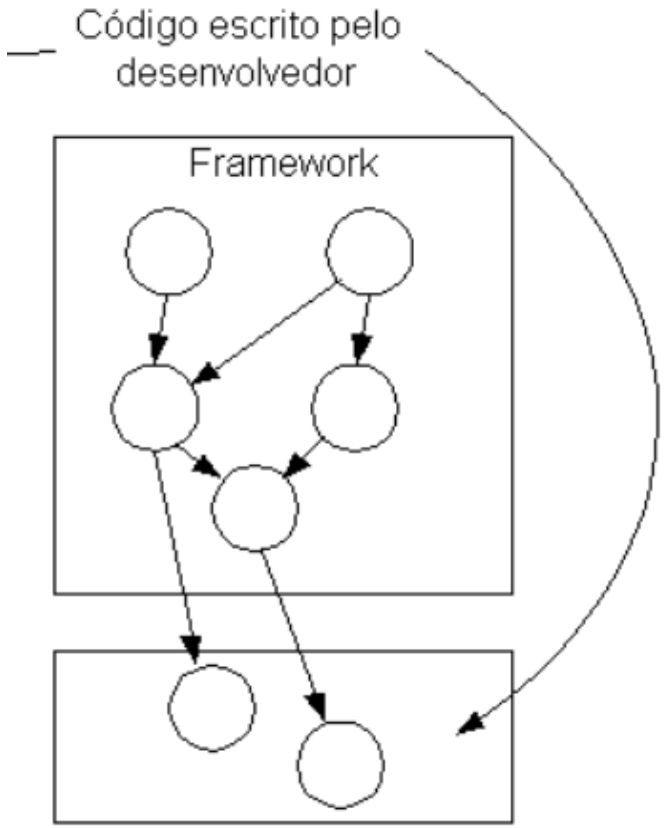
- **Deve ser completo**
  - Quantos mais abstrações e ferramentas a framework disponibilizar mais completa é.

Como funciona uma Framework?

É a framework que chama o código da aplicação

A framework tem um conjunto de comportamentos já codificados, sempre que possível de forma abstrata, deixando para o programador a implementação concreta dessas abstrações.

Uma framework é uma aplicação "quase" completa, cabe ao programador conhecer a framework, para implementar os "pedaços" que faltam da maneira correta.



O framework é usado de acordo com o Hollywood Principle ("*Don't call us, we'll call you*")

Vantagens de usar uma framework

- Maior facilidade para deteção de erros;
- Eficiência na resolução de problemas;
- Otimização de recursos;
- Concentração na abstração de solução de problemas;
- Modeladas com vários padrões de projeto;
- Grande parte do trabalho já está feito;
- Na maior parte existe uma comunidade que trabalha continuamente na melhoria e manutenção;
- Implementam as melhoras práticas de desenvolvimento de software.

Tipo de Frameworks

- **Micro Frameworks**

Os micro-frameworks são essencialmente invólucros para rotear uma requisição HTTP para um callback, ou um controller, ou um método etc., da forma mais rápida possível, e algumas vezes possuem bibliotecas para auxiliar no desenvolvimento, como por exemplo pacotes básicos para bases de dados. Eles são proeminentemente usados para construir serviços HTTP remotos.

- **Full-Stack Frameworks**

Muitas frameworks adicionam um número considerável de funcionalidades ao que está disponível em um micro-framework e são conhecidos como frameworks completas ou full-stack.Frequentemente possuem ORMs, autenticação, etc.

- **Component Frameworks**

Frameworks baseados em componentes são coleções de bibliotecas especializadas ou de propósito-único. Diferentes frameworks baseados em componentes podem ser utilizados conjuntamente para criar uma micro-framework ou uma framework full-stack.

Full-Stack Frameworks PHP

Zend

Criado em 2005, desenvolvida por alguns dos programadores core do PHP, alcançou fama rapidamente, principalmente pelo apoio dado por empresas como Google, Microsoft e Cisco. É um dos frameworks mais atualizados e consistentes do mercado. Não é um modelo simples, de forma que tende a ser mais indicado para os projetos mais robustos.

Symfony

Symfony é uma escolha confiável para projetos de grande escala. Desenvolvido pela Sensiolabs (2005), usa um sistema de componentes reutilizáveis e segue o padrão MVC. Foca essencialmente em regras de negócio da aplicação. Normalmente é indicado em trabalhos de grande escala e mais robustos. É a framework usada por muitos projetos de sucesso em PHP, como o Drupal, Magento, Prestashop... etc.

Full-Stack Frameworks PHP

Phalcon

Foi criado em 2012 por Andres Gutierrez e tem crescido exponencialmente na área de desenvolvimento.
O Phalcon é diferente dos outros frameworks por ser uma extensão escrita em C. Esta abordagem permite aumentar a velocidade de execução e diminuir o uso de recursos. É muito provavelmente o framework mais rápido do mercado.

CakePHP

Foi lançado em 2005.
Trabalha em cima da arquitetura MVC, o que também possibilita a criação de projetos dos mais diversos portes. A documentação é bastante completa, tem perdido alguma popularidade mas continua a ser um dos frameworks mais usados.

Full-Stack Frameworks PHP

Laravel

É um dos frameworks PHP mais utilizados no mercado atualmente.
Lançado em 2011 e desenvolvido por Taylor Orwell com o intuito de corrigir algumas fraquezas que viu na estrutura de aplicativo da web CodeIgniter.
Tem uma comunidade muito ativa de apoio e um extenso ecossistema de tutoriais (ex: Laracasts) e recursos. Possui um interface de linha de comandos (artisan) poderoso, e conta com várias opções para aceder a bases de dados (ORM's) e alguns utilitários úteis para implantar e manter aplicações.

Micro Frameworks PHP

Com o tempo, as frameworks full-stack ficaram cada vez maiores para lidar com os sites cada vez maiores e complexos que apareciam no mundo online. A desvantagem desse crescimento é que ficou mais difícil desenvolver projetos simples e rápidos, para além das muitas funcionalidades não utilizadas.

Em resposta a esses desafios, a comunidade PHP desenvolveu microestruturas. Para projetos menores e casos de uso específicos, essas estruturas simplificadas são mais fáceis de implementar e mais rápidos, pois usam menos recursos.

Com a crescente adoção do paradigma de desenvolvimento em micro-serviços (popularizado pela Netfilx) em que as grandes aplicações são divididas em aplicações mais pequenas, a utilização destas micro frameworks tem aumentado, visto serem ideais para aplicações que se querem pequenas e rápidas.

Vejamos algumas das micro-frameworks mais conhecidas entre a comunidade PHP...

Micro Frameworks PHP

Lumen

Projetada por Taylor Otwell em 2014 (criador do laravel), permite o uso de muitos componentes do Laravel como “middleware”, validação, cache e o contêiner de serviço do Laravel.

Slim

Slim é uma das micro frameworks mais populares de PHP. Projectado por Josh Lockhart autor dos livros "Modern PHP” e “PHP the Right Way.” Conta ja com mais de 1 Milhão de instalações. Oferece um poderoso router, abstrações simples de "request – response" e muitas "ferramentas" auxiliares para facilitar cache e suporte a sessões.

Micro Frameworks PHP

Silex

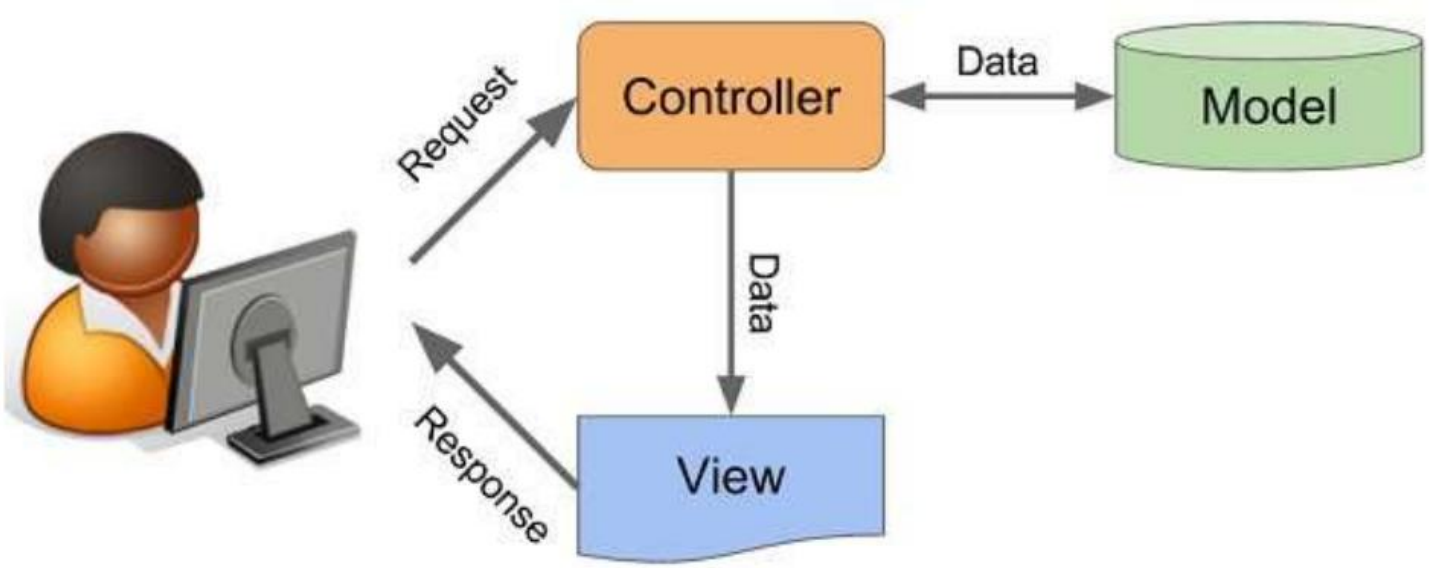
Fabien Potencier e Igor Wiedler criaram o Silex em 2010. Baseado em Symfony e no mecanismo de template Twig. Silex é extremamente leve e permite adicionar recursos conforme necessário. Construído originalmente para lidar com pequenos projetos, pode ser estendido para uma estrutura completa "MVC".

Phalcon

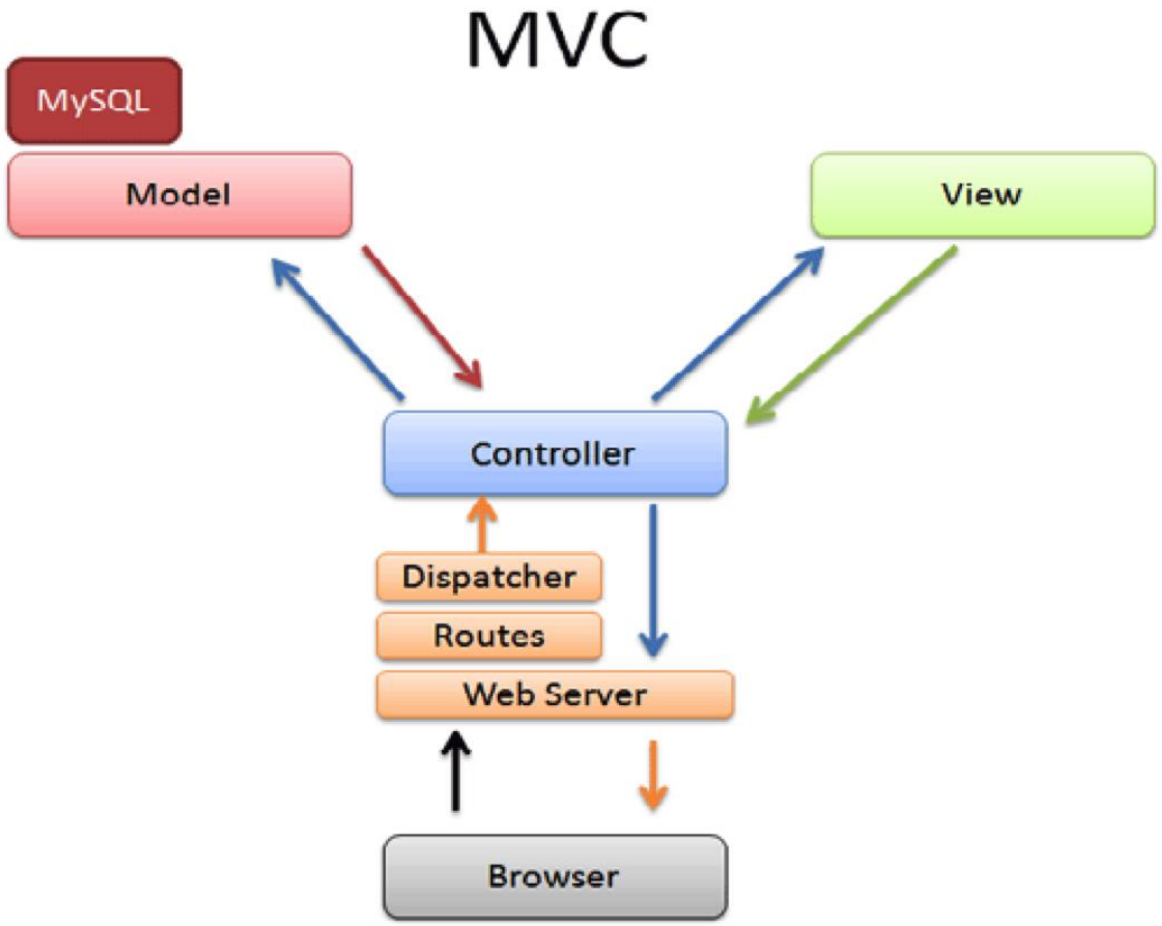
A versão micro-stack da estrutura full-stack Phalcon. Micro-stack Phalcon é a micro-framework mais rápida disponível devido à sua implementação como uma extensão em C (As mesmas razões do seu irmão mais velho). No entanto, Phalcon não tem muito suporte da comunidade e a sua documentação não é tão completa como a das restantes frameworks.

MVC (Model - View – Controller)

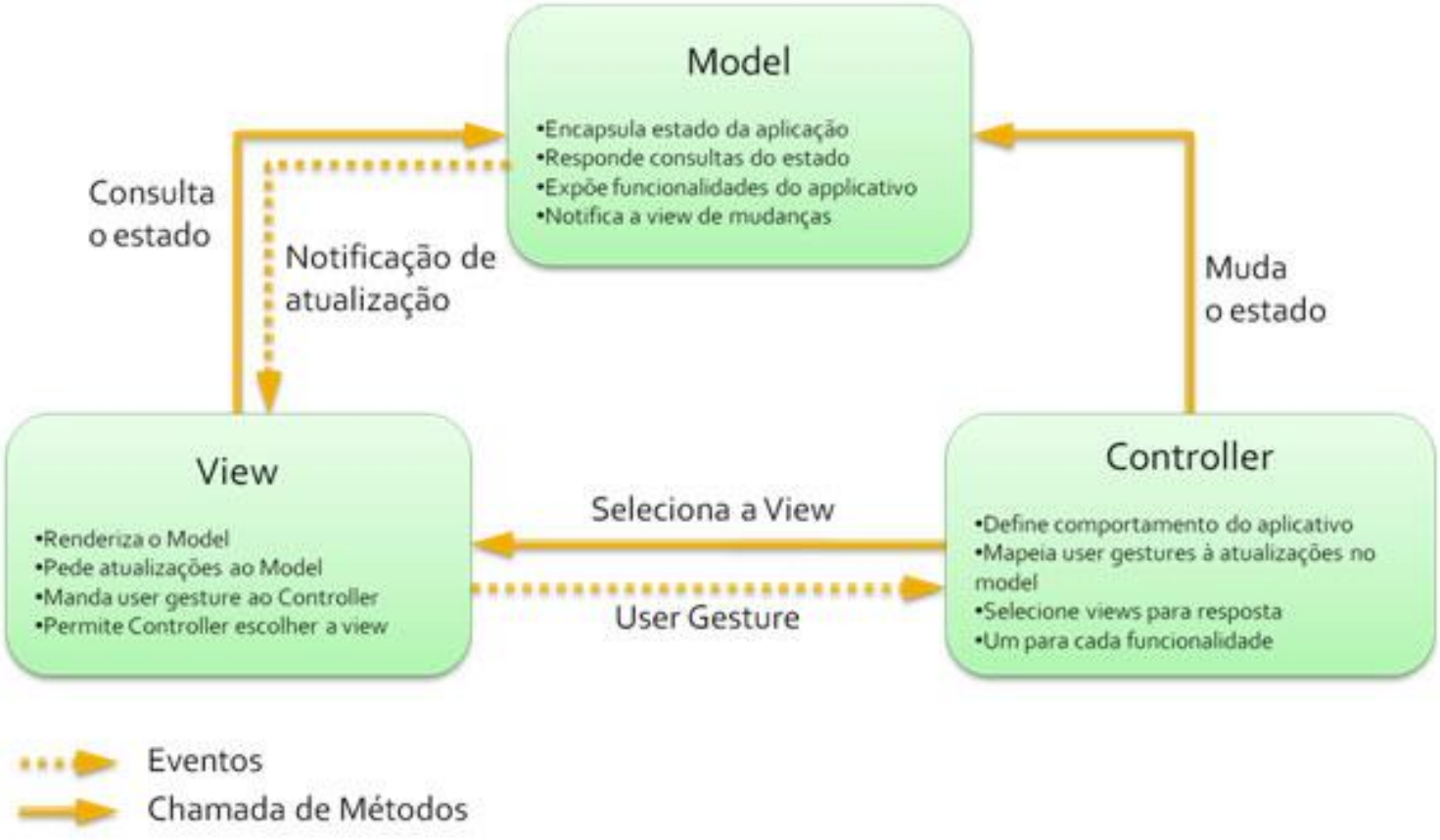
Padrão de desenho MVC



Padrão de desenho MVC



Padrão de desenho MVC



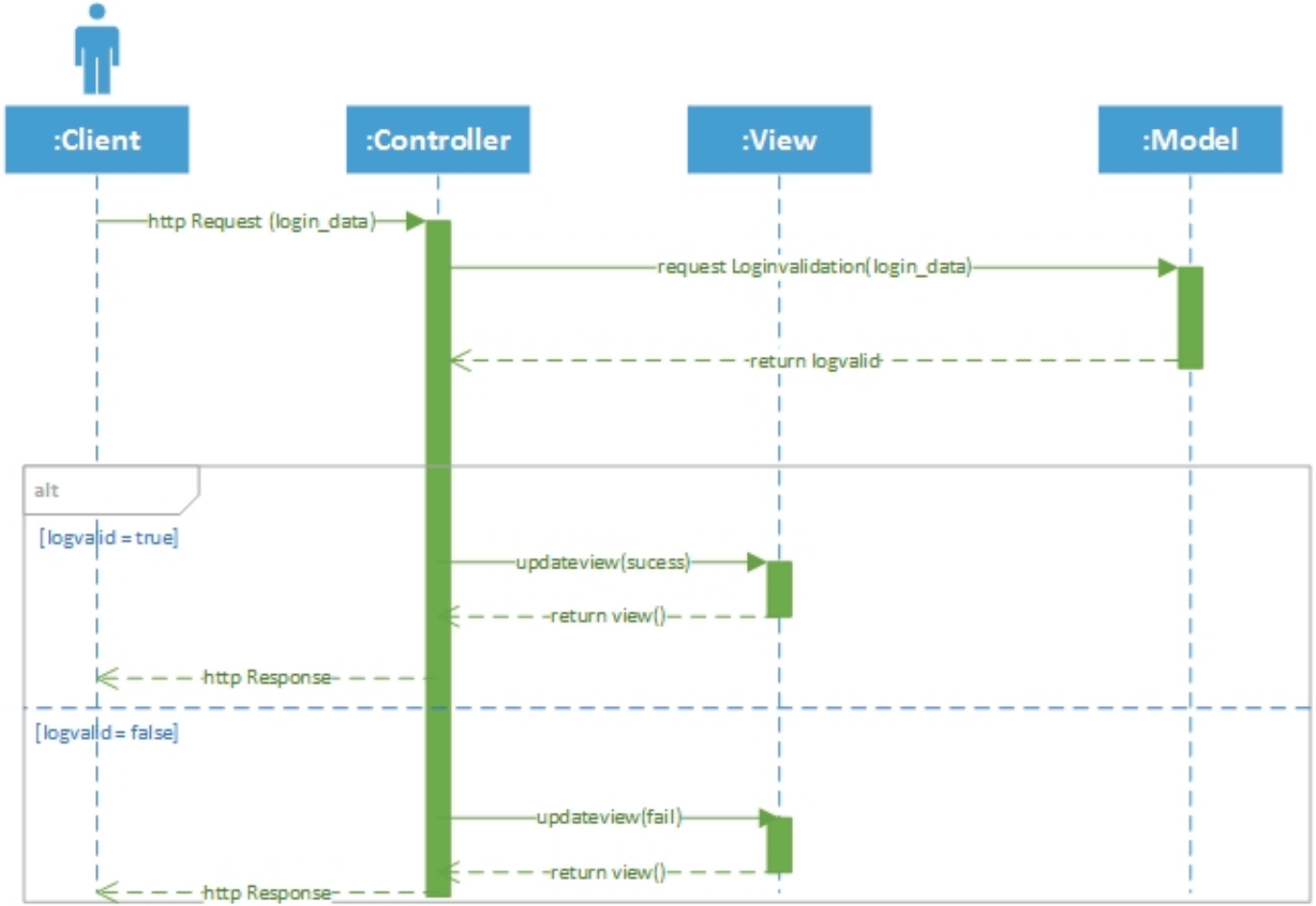
Padrão de desenho MVC

- O **Controlador** (**controller**) envia comandos para o modelo para atualizar o estado (por exemplo, editando um recurso). O controlador também pode enviar comandos para a vista associada para alterar a apresentação da informação do modelo (por exemplo, json ou html).

- Um **modelo** (**model**) Responsável por armazenar a estrutura de dados da aplicação (modelo de dados) e por implementar alguma logica de negócio.

- A **vista** (**view**) gera uma representação dos dados presentes no modelo solicitado, fazendo a exibição dos dados, sendo ela por meio de um html ou xml ou json.

Padrão de desenho MVC



PHP Frameworks

Composer

Composer

O composer é um gestor de dependências para o PHP.
Permite-nos gerir as dependências de ("librarias" / "packages") externos, fazendo o trabalho de download e atualização dessas mesmas dependências por nós.
O composer funciona para o PHP da mesma forma que o NMP para o node ou o maven para o java.

Para o nosso workshop vamos precisar de instalar o composer, veja aqui como instalar.

Composer

Para instalar o composer devemos ter o PHP instalado pois o ficheiro de instalação é em PHP.
Uma das formas de instalar o composer localmente é fazer o download do installer e executa-lo localmente.

```
13:59:41-marcio:~/Downloads$ ls
documento.pdf  installer  phptherightway.pdf
13:59:43-marcio:~/Downloads$ php installer
All settings correct for using Composer
Downloading...

Composer (version 2.0.4) successfully installed to: /home/marcio/Downloads/composer.phar
Use it: php composer.phar
```

A partir deste momento podemos usar o nosso ficheiro composer.phar para executar o composer.

```
14:08:59-marcio:~/Downloads$ ls
composer.phar  documento.pdf  installer  phptherightway.pdf
14:09:00-marcio:~/Downloads$ php composer.phar -V
Composer version 2.0.4 2020-10-30 22:39:11
```

Composer

Como este ficheiro composer.phar tem tudo o que é necessário para executar todas as funcionalidades podemos executa-lo diretamente, mas também podemos usá-lo globalmente, para isso basta que copiemos o ficheiro para dentro de um diretório que a nossa variável global PATH conheça.
Em sistemas Unix, podemos até torná-lo executável e invocá-lo sem explicitar diretamente o interpretador php.

```
14:13:09-marcio:~/Downloads$ ls
composer.phar  documento.pdf  installer  phptherightway.pdf
14:13:10-marcio:~/Downloads$ echo $PATH
/home/marcio/.nvm/versions/node/v8.17.0/bin:/home/marcio/.local/bin:/usr/local/sbin:/usr/local/bin:
in:/usr/games:/usr/local/games:/snap/bin
14:13:13-marcio:~/Downloads$ sudo mv composer.phar /usr/local/bin/composer
[sudo] password for marcio:
14:13:34-marcio:~/Downloads$ composer -V
Composer version 2.0.4 2020-10-30 22:39:11
14:13:40-marcio:~/Downloads$
```

"instalar" o Laravel

Laravel workshop – criar projeto

A partir da linha de comandos, aceda a pasta C:\xampp\htdocs e execute o seguinte comando:

```
> php composer.phar create-project laravel/laravel demo-laravel
```

Pode acontecer de o nosso PHP não ter todos os módulos necessários instalados, caso isso aconteça aparecerá uma mensagem de erro. Para prosseguir basta instalar o modulo em falta e recomeçar o processo.

```
Your requirements could not be resolved to an installable set of packages.

  Problem 1
    - phpunit/phpunit[dev-master, 9.3.3, ..., 9.4.x-dev] require ext-dom * -> it is missing from your system. Install or enabl
e PHP's dom extension.
    - phpunit/phpunit 9.5.x-dev is an alias of phpunit/phpunit dev-master and thus requires it to be installed too.
    - Root composer.json requires phpunit/phpunit ^9.3.3 -> satisfiable by phpunit/phpunit[9.3.3, ..., 9.5.x-dev (alias of dev
-master)].
```

Instalar dependência php em falta (linux)

```
sudo apt-get install php7.4-dom
```

Laravel workshop – criar projeto

Depois de termos descarregado todas as dependências necessárias para o nosso projeto lavarel, devemos ter uma mensagem deste género:

```
> @php artisan key:generate --ansi
Application key set successfully.
15:00:26-marcio:~/Desktop/flag/projectos$
```

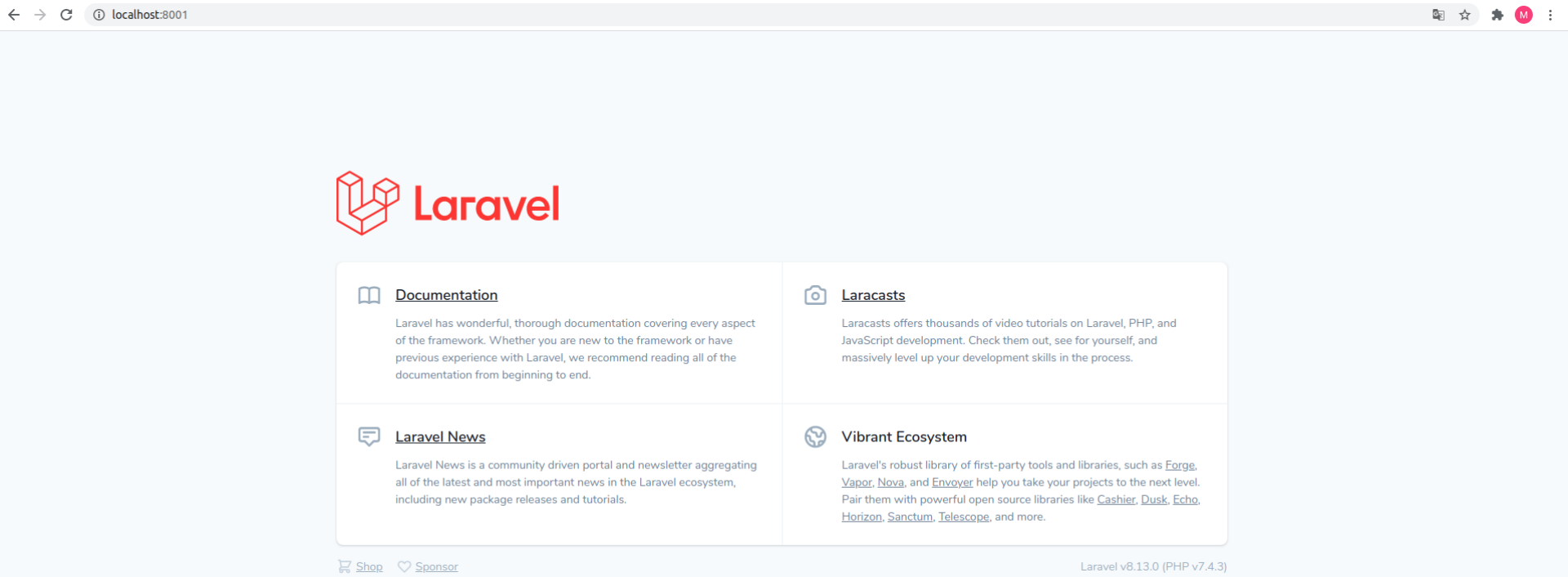
O comando anterior deverá criar uma pasta com o nome de demo-laravel na pasta htdocs do xampp, já com os arquivos de projeto Laravel. Para fazer com que o projeto fique disponível no browser, na linha de comandos digite:

```
> cd demo-laravel
> php artisan serve
```

```
5:19:22-marcio:~/Desktop/flag/projectos/demo-laravel$ php artisan serve
tarting Laravel development server: http://127.0.0.1:8000
Tue Nov  3 15:19:47 2020] Failed to listen on 127.0.0.1:8000 (reason: Address already in use)
tarting Laravel development server: http://127.0.0.1:8001
Tue Nov  3 15:19:48 2020] PHP 7.4.3 Development Server (http://127.0.0.1:8001) started
```

Laravel workshop – criar projeto

A partir deste momento se tudo correu bem... ao aceder ao nosso localhost pela porta especificada pelo ultimo comando (ex: http://localhost:8001/) devemos ter uma pagina HTML fornecida pela nossa nova aplicação laravel.



Laravel workshop – artisan

Artisan é uma interface de linha de comandos incluída no Laravel.
É uma ferramenta que fornece uma série de comandos úteis que nos ajudam enquanto construímos a nossa aplicação. Para ver uma lista de todos os comandos Artisan disponíveis, podemos usar o comando list:

```
> php artisan list
```

Podemos também alterar o formato da resposta

```
> php artisan list --format=xml
```

Para termos uma explicação do que faz um comando do artisan podemos usar o parâmetro –help depois do comando

```
> php artisan route:list --help
```

Laravel - Rotas

Laravel workshop – rotas

O Laravel fornece um sistema completo de para podermos lidar com o routing da nossa aplicação.
Todas as rotas do Laravel são definidas nos ficheiros de rotas, que estão localizados no diretório de rotas. Esses arquivos são carregados automaticamente pelo framework.

- O ficheiro routes / web.php define as rotas que são para sua interface web. Essas rotas são atribuídas ao grupo de middleware da web, que fornece recursos como estado da sessão e proteção contra CSRF (cross-site request forgery) .

- As rotas em routes / api.php não têm estado e são atribuídas ao grupo de middleware api.

Laravel workshop – rotas

Graças a este componente não precisamos de indicar na URL o ficheiro a queremos aceder.

- Ao invés do link http://localhost:8001/pagina.php, poderíamos configurar uma rota para que o recurso fique acessível a partir do link http://localhost:8000/pagina

- Assim, cada recurso da aplicação pode ser representado por um verbo HTTP (GET,POST,PUT,DELETE....etc) e o recurso a queremos aceder;

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Laravel workshop – rotas

Para o desenvolvimento da nossa aplicação vamos usar as rotas web. Como vimos anteriormente é possível criar/editar as nossas rotas através do ficheiro routes/web.php

Atribuições de operações HTTP são realizadas a partir da classe Route e de seus métodos estáticos, que representam cada uma das operações HTTP existentes;

Sintaxe básica para a definição de rota:

```
Route::<operação_HTTP>('/recurso', function() {
        // O que fazer quando esta rota for acedida!
});

// Ex:
Route::get('/', function () {
        return view('welcome');
});
```

Laravel workshop – rotas

Exemplo 1: Abra o arquivo routes/web.php e crie a seguinte rota:

```
Route::get('/php-info', function () {
    phpinfo();
});
```

Experimente devolver uma string

```
Route::get('/hello', function () {
    return 'Hello World';
});
```

Experimente passar parâmetros no url

```
Route::get('user/{id}', function ($id) {
    return 'User '.$id;
});
```

Laravel workshop – rotas

Já que nossa rota espera um id, vamos alterá-la para aceitar apenas números (atualmente ela também aceita strings!);

Para isso, basta invocar o método where após a definição da rota, passando como parâmetro o nome do parâmetro e a expressão regular que impõe o uso de números;

```
Route::get('user/{id}', function ($id) {
    return 'User '.$id;
})->where('id','[0-9]+');
```

Com o uso do where caso o parâmetro não cumpra os requisitos da expressão regular será lançada uma exceção NOT FOUND que apresentará no browser o respetivo erro http 404.

Laravel workshop – rotas

Exemplo 2: Abra o arquivo routes/web.php e crie uma nova rota:

```
Route::get('/formulario', function() {
    return '
        <form method="post" action="/contato">
            Nome: <input type="text" name="nome">
            Email: <input type="text" name="email">
            Mensagem: <textarea name="mensagem"></textarea>
            <input type="submit" value="Enviar">
        </form>';
});
```

Laravel workshop – rotas

Agora crie uma rota para receber o post enviado pelo formulario

```
Route::post('/contato',  function() {
      print_r(Request::all());
});
```

Abra o navegador, http://localhost:8000/formulario e verifique se o formulário criado no arquivo de rotas será apresentado. Digite suas informações de contato e os submeta, afim de verificar se a rota post será executada;

Algum Problema?

Laravel workshop – rotas

Agora crie uma rota para receber o post enviado pelo formulario

```
Route::post('/contato',  function() {
      print_r(Request::all());
});
```

Abra o navegador, http://localhost:8000/formulario e verifique se o formulário criado no arquivo de rotas será apresentado. Digite suas informações de contato e os submeta, afim de verificar se a rota post será executada;

Algum Problema? SIM

Todo formulário Laravel precisa submeter um Token, chamado CSRF, para que possa enviar operações HTTP em aplicações Laravel (por questões de segurança!);

Laravel workshop – rotas

Como só podemos inserir tais tokens em formulários implementados em Views, por enquanto, vamos desligar esse recurso, comentando a linha de código que o implementa no arquivo app/Http/Kernel.php;

```
/**
 * The application's route middleware groups.
 *
 * @var array
 */
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        // \Illuminate\Session\Middleware\AuthenticateSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        //\App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
];
```

Laravel - Controllers

Laravel workshop – controllers

Temos de definir uma rota para cada tipo de *request* HTTP?
Em vez de definir toda a lógica de tratamento das *requests* feitas pelo web browser no ficheiro de rotas web.php, podemos passar esse comportamento para os nossos controllers.

Os controladores podem agrupar a lógica de tratamento de pedidos HTTP. Para isso apenas de invocar o metodo *resource* em vez do verbo http na nossa rota.
O *resource* do Laravel atribui as rotas "CRUD" típicas a um controlador com uma única linha de código

```
Route::resource('rotas', Rotas::class);
```

Para criar o respetivo controlador podemos usar o artisan

```
php artisan make:controller Rotas
```

Laravel workshop – controllers

No Laravel os controladores são armazenados no dirétorio: app/Http/Controllers/
Se abrir-mos o nosso controller Rotas.php podemos verificar que extend a class Controller é essa extensão que lhe dá "os poderes" para poder receber e entender as requests que são encaminhadas pelas rotas.

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class Rotas extends Controller
{
    //
}
```

Laravel workshop – controllers

Neste momento já adicionamos a nossa rota para usar o controlador e já criamos o controlador, que neste momento está "vazio", pelo que se tentarmos aceder á nossa nova rota (http://localhost:8000/rotas) ainda vamos ter o erro 404 (Not Found), como se ela não existesse.

Isto acontece porque não temos qualquer implementação no nosso controller.
Os controllers de laravel devem ter um método por defeito que é executado quando acedemos á nossa rota sem expecifar "o que queremos fazer". Que representa a nossa rota padrão para este controller.

Adicione o método padrão á nossa class app/Http/Controllers/Rotas.php

```
public function index()
{
        return 'Olá, sou a rota padrão do controller!';
}
```

Uma vez que adicionamos o método padrão ao controller já teremos a resposta padrão.

Laravel workshop – controllers

Ok neste momento temos o nosso controller com o metodo padrão implementado, mas então como fazemos para as restantes rotas "CRUD" (Create, Read, Update, Delete), no nosso controller, visto que foi com esse objectivo que definimos o rota com *resource?* O laravel requer que o programador implemente metodos com uma assinatura especifica para esse efeito.

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Laravel workshop – controllers

No controlador de rotas implemtente o seguinte método.
Depois acedendo a http://localhost:8000/rotas/2 pode verificar que o controller "percebeu" qual o método que deveria executar

```
public function show($id)
{
    return 'Olá, '.$id;
}
```

Como estes médodos representam um padrão para um controller do tipo resource, podemos indicar ao artisan para criar estes métodos por nós.

```
php artisan make:controller RotasController --resource
```

NOTA: Outro padrão que devemos adoptar é ao dar nome às nossas classes que representam controller terminar com a palavra Controller. EX: NomeDoRecusoController,php

Laravel workshop – controllers

Acabamos de ver os resource controllers (que representam um recurso na nossa aplicação como por exemplo utilizadores), mas como fazemos para os controllers que representam outros casos de uso não relacionados com recursos ou fora do CRUD.

Funciona exatamente da mesma forma, no entanto nesse caso a nossa rota tem de saber para onde encaminhar o pedido HTTP.

Adicionar rota

```
Route::get(test', [TestController::class, 'test']);
```

Criar o controller

```
php artisan make:controller TestController
```

Implementar o método

```
public function test(){ return "isto é apenas um test"; }
```

Laravel – Views

Laravel workshop – Views

As Views são classe do Modelo MVC responsáveis pela exibição de dados para um cliente HTML após um determinado processamento;

- O Blade é o motor de renderização que o Laravel adota.
- O Blade permite uma sintaxe mais compacta na criação de layouts, além de disponibilizar comandos para gerar conteúdo dinâmico (geralmente começam por @)
- As views são elementos independentes e podem ser alimentadas com dados.
- "Os mesmos dados podem ter várias apresentações distintas"
- As views são armazenadas no diretório: /resource/views

Laravel workshop – Views

O Blade permite ao programador reutilizar outras views na criação de novas views.

Exemplo: Crie na pasta /resource/views a view layout.blade.php:

```
<html>
    <body>
        @section('sidebar')
            Este é o sidebar da página
        @show
        <div>Olá, @yield('content')!</div>
    </body>
</html>
```

Adicione uma nova rota

```
Route::get('/layout', function() {
        return view('layout');
});
```

Laravel workshop – Views

O Blade permite ao programador "passar" dados dinâmicos.

Exemplo: Crie na pasta /resource/views a view test.blade.php:

```
<html>
    <body>
        <h1>Hello, {{ $name }}</h1>
    </body>
</html>
```

Adicione uma nova rota

```
Route::get('/test', function() {
        return view('test', ['name' => 'James']);
});
```

Laravel workshop – Views

O Blade permite ao programador definir seções de página de forma muito simples.

Exemplo: Crie na pasta /resource/views a view pagina1.blade.php:

```
@extends('layout')
```

Adicione uma nova rota

```
Route::get('/pagina1, function() {
        return view('pagina1');
});
```

Laravel workshop – Views

O Blade permite ao programador executar loops

```
@for ($i=0;$i<10;$i++)
    <p>O valor de i é {{ $i }}</p>
@endfor
```

```
@while(true)
    <p>Vou encravar o seu browser!</p>
@endwhile
```

Laravel – Base de dados (Migrações)

Laravel workshop – Configurar BD

No laravel as configurações estão guardadas no diretorio config.
As configs da base de dados estão no ficheiro config/database.php

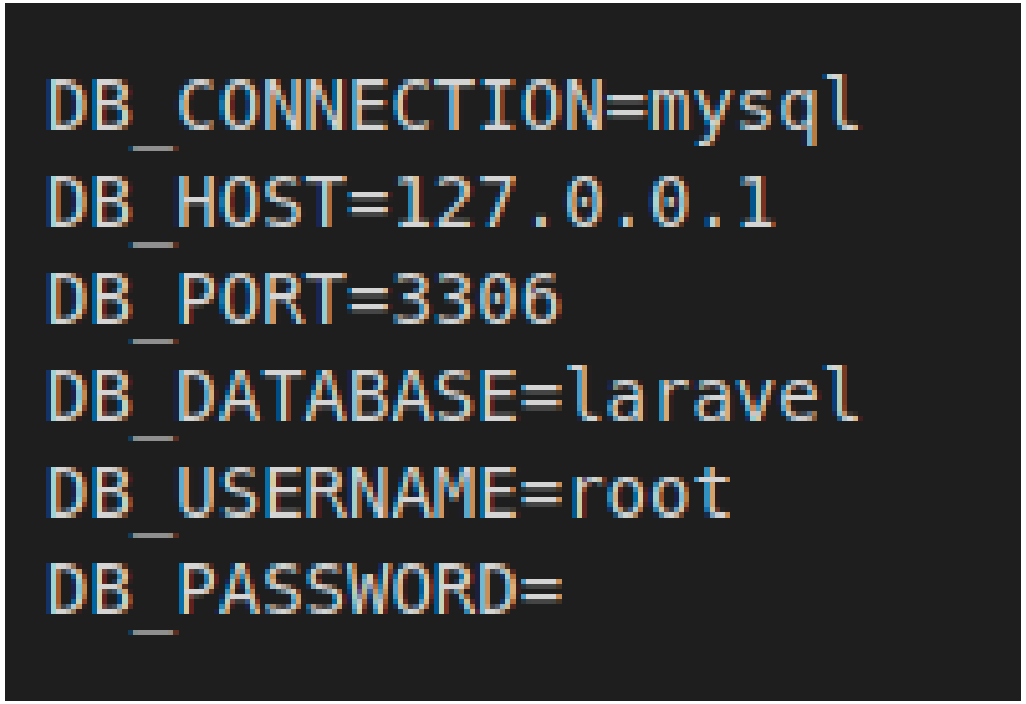
```
'mysql' => [
    'driver' => 'mysql',
    'url' => env('DATABASE_URL'),
    'host' => env('DB_HOST', '127.0.0.1'),
    'port' => env('DB_PORT', '3306'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'unix_socket' => env('DB_SOCKET', ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => null,
    'options' => extension_loaded('pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env('MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```

Onde vemos a expressão env('VARIAVEL_AMBIENTE', 'valor_default').

Deve ler-se que se a variavél de ambiente existir então assina o seu valor, se não usa o valor do segundo parametro como default.

Laravel workshop – Configurar BD

Existe um ficheiro .env na raiz do projecto laravel, que atribui o valor das variáveis de ambiente.



Para configurar a conexão á nossa base de dados, preencher os valores correctos para as variáveis apresentadas aqui ao lado no ficheiro .env na raiz do projecto laravel

Laravel workshop – Migrações

O laravel disponibiliza o conceito de migrações para podermos criar ou alterar os nossos modelos de dados.

Todas as migrações ficam registadas e devem contemplar o up (que executa um conjunto de instruções sql na base de dados como o CREATE TABLE por exemplo) e o down (que visa reverter a instrução dada pelo up DROP TABLE neste caso).

Os ficheiros de migrações encontram-se no diretorio database/migrations e cada um deles representa uma migração executada na base de dados.

Laravel workshop – Migrações

- Crie uma base de dados chamada laravel-demo e configure o ficheiro .env com os dados de acesso á base de dados.
- Na base de dados execute o comando
- Aceda ao diretorio database/migrations.
- Verifique que já existem alguns ficheiros de migrações que visam criar os nossos utilizadores com login.
- Apague as migrações existentes.
- Na linha de comandos execute:

```
php artisan make:migration  nova_tabela
```

- Verifique que foi criado um ficheiro xxxx_nova_tabela na pasta database/migrations, esse ficheiro representa uma nova migração e conta com 2 metodos.
- Up() que será o método chamado quando executarmos a migração.
- Down() que será o método chamado quando fizermos reset à migração

Laravel workshop – Migrações

Implemente os metodos up e down da nova migração.

```php
public function up()
{
    Schema::create('nova_tabela', function(Blueprint $tabela) {
        $tabela->integer('id', true);
        $tabela->text('descricao');
    });
}
```

```php
public function down()
{
    Schema::drop('nova_tabela');
}
```

Laravel workshop – Migrações

Para executar a migração que acabamos de criar podemos usar o comando

```
php artisan migrate
```

Depois de executar o comando verifique que agora existem 2 novas tabelas na base de dados:

- A tabela nova_tabela que foi gerada através do metodo up da nossa migração
- A tabela migrations que regista todas as migrações que são executadas, que contém agora um registo.

Execute agora o seguinte comando:

```
php artisan migrate:reset
```

Depois de executar o comando verifique que:

- A nova_tabela foi apagada da base de dados.
- A tabela migrations mantem-se, mas o registo que existia foi apagado.

Laravel – Models (ORM)

Laravel workshop – Eloquent (ORM)

- O Laravel trabalha com um ORM (Object Relacional Mapping), que se encarrega de abstrair a conversão das estruturas de tabelas de BD e instruções SQL para o mundo da orientação a objetos.
- O Eloquent implementa o padrão Active Record.
- Cada tabela de banco de dados possui um "Modelo" correspondente que é usado para interagir com essa tabela. Os modelos permitem consultar dados em suas tabelas, bem como inserir novos registros na tabela.

Laravel workshop – Eloquent (ORM)

Adicione os atributos de marca e modelo ao Model carro

```
php artisan make:model Carro --migration
```

Depois de executar o comando verifique que na pasta app/models existe uma classe chamada Carro.php que estende a class do Model da framework.
O parâmetro migration indica que também queremos criar uma migration para este modelo.

Atualize a migração para conter duas colunas para alem do id (marca/modelo)

```
Schema::create('carros', function (Blueprint $table) {
        $table->id();
        $table->text('marca');
        $table->text('modelo');
        $table->timestamps();
});
```

Laravel workshop – Eloquent (ORM)

Execute a migração

```
php artisan migrate
```

Verifique que criou a tabela carros na nossa base de dados e insira alguns dados

Crie um controller e uma rota:

```
php artisan make:controller CarrosController --resource
```

```
Route::resource('carros', CarrosController::class);
```

No controller devemos inserir o use do modelo e escrever o seguinte código

```
public function index() {
    return Carro::all();
}
```

Laravel workshop – Eloquent (ORM)

Repare que não precisamos de indicar no nosso modelo qual a tabela a usar, quais as propriedades do modelo. Por defeito o eloquent vai usar para nome da tabela o nome da nossa classe no plural usando a notação snake_case (ex: NomeClass = nome_classes).
Para além disso o eloquent usa como nome das nossas propriedades o nome das respetivas colunas na base de dados.
No entanto é possível alterar isso. Se consultarmos a class Illuminate\Database\Eloquent\Model podemos ver um conjunto de propriedades protected que representam características das nossas tabelas e que podemos reescrever.

```
Class LivroModel extends Model {
        protected $table = 'livro';
        protected $primaryKey = 'id';
        protected $fillable = ['titulo', 'descricao', 'data_publicacao'];
        public $timestamps = false;
}
```

Veja aqui como o laravel faz esta "magia" acontecer

Laravel – Exercício (MVC)

Laravel workshop – Create MVC (Exercício)

Crie num modelo MVC uma representação do recurso médico já abordado

```
php artisan make:model Medico --migration --controller --resource
```

Adicione uma rota do tipo resouce

```
Route::resource(medicos, MedicoController::class);
```

Implemente os metodos de resouce disponiveis no controller.
Use a documentação do laravel para saber como retornar os objectos da BD.
https://laravel.com/docs/8.x/eloquent

Laravel – Autenticação

Laravel workshop – Autenticação

- O Laravel já tem uma implementação de autenticação pré-definida.
- O ficheiro de configuração de autenticação está localizado em config/auth.php, que contém várias opções bem documentadas para ajustar o comportamento dos serviços de autenticação.
- Na pasta database / migrações são carregadas por defeito as migrações referentes às tabelas de utilizadores e de autenticação.
- Existem várias "ferramentas" desenvolvidas pelo laravel que nos premitem o scaffolding da autenticação (no entanto também é possivel implementar todos os passos "á mão").
- Na documentação do laravel podemos ver todas estas opções.

Laravel workshop – Autenticação

Instale o laravel/ui para poder ter um leque mais alargado de opções para as views.
(nota se preferir algo mais completo / robusto pode optar pelo laravel/jetstream)

```
composer require laravel/ui
```

Podemos agora usar o scaffolding do componente ui para montar as nossas vistas e rotas de autenticação. (no lugar de bootstrap tambem é possivel optar por view)

```
php artisan ui:auth bootstrap
```

Instalar as dependências da laravel/ui

```
npm install
```

Compilar ui

```
npm run dev
```

Laravel workshop – Autenticação

Verifique que foram adicionadas as rotas de autenticação e uma nova rota que requer autenticação.

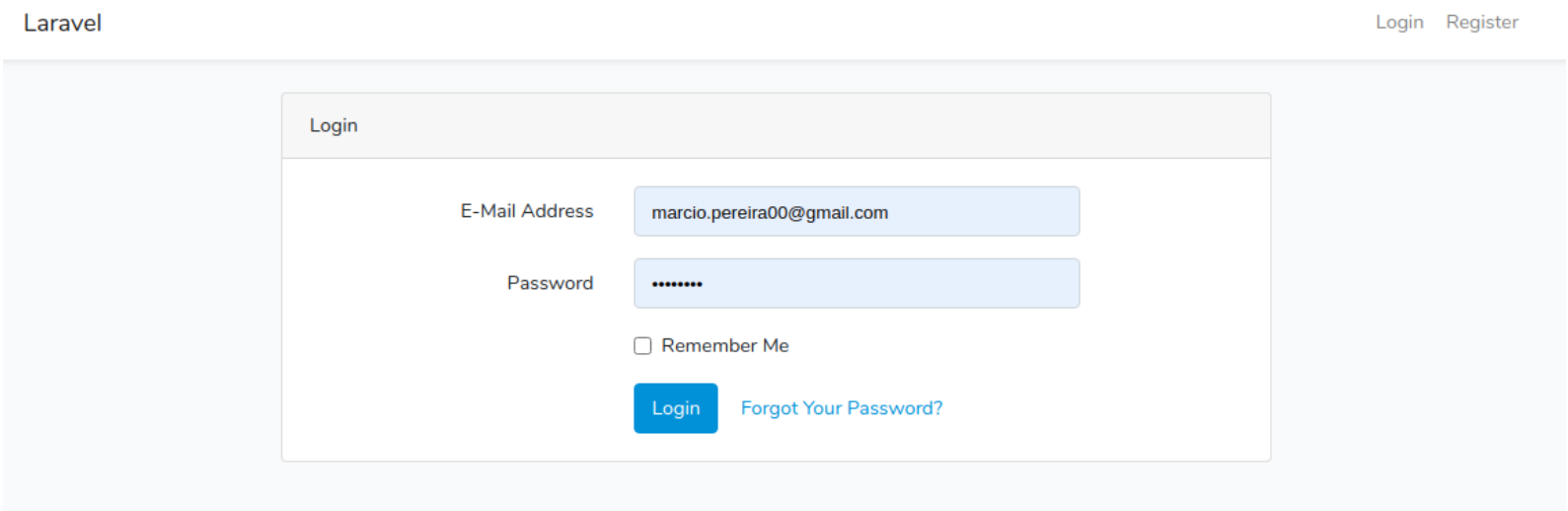
```php
Auth::routes();

Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
```

```php
/**
 * Create a new controller instance.
 *
 * @return void
 */
public function __construct()
{
    $this->middleware('auth');
}
```

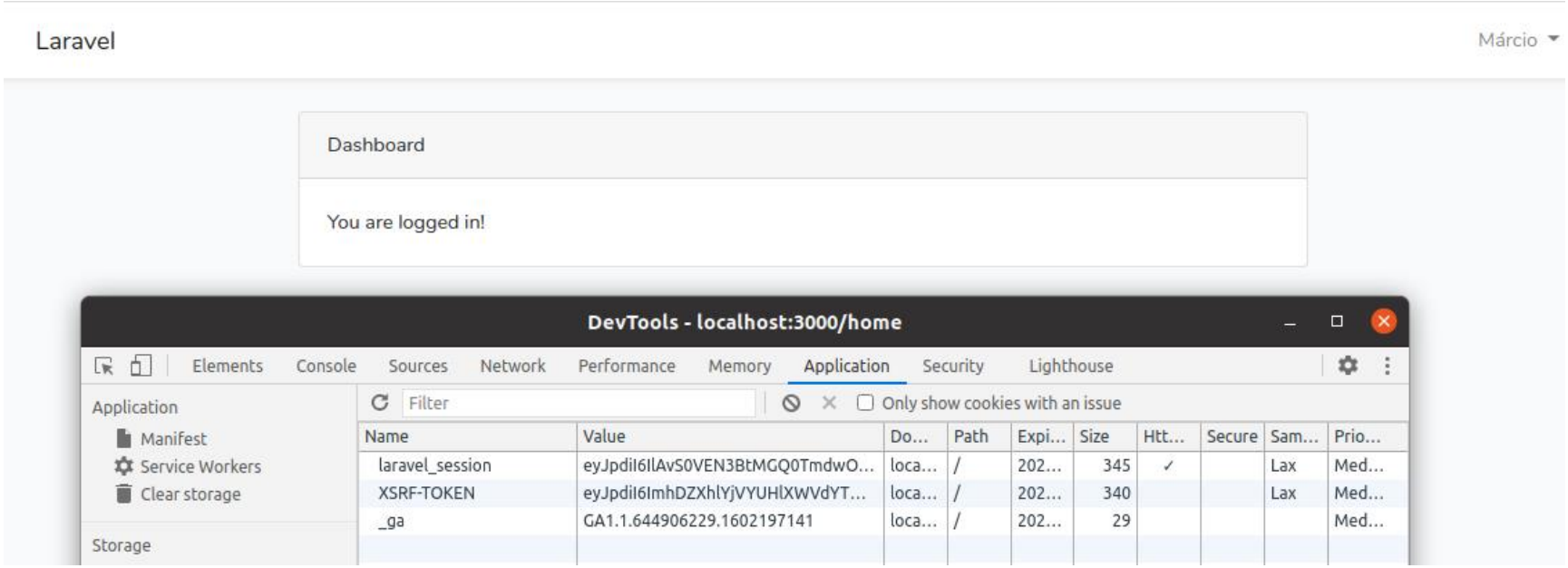
Laravel workshop – Autenticação

Verifique que tem agora novas rotas para de registo e login em localhost:8000/login



Laravel workshop – Autenticação

Creie uma conta e faça login. Verifique que agora já tem sessão.



Laravel – Models (Relationship)

Laravel workshop – Eloquent Relationships

Como podemos representar as relações existentes nos nossos modelos?

As relações entre os diferentes modelos (entidades) da nossa aplicação são definidas na base de dados, quer isto dizer que se estamos a criar o nosso modelo de dados apartir de migrações então temos que criar a relação também nessa migração.

```
public function up()
{
   Schema::create('especialidades', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });

   Schema::table('medicos', function (Blueprint $table) {
       $table->foreignId('especialidade_id')->references('id')->on('especialidades');
   });
}
```

Laravel workshop – Eloquent Relationships

Como podemos representar as relações existentes nos nossos modelos?

Visto ser o eloquent que "trata" do mapeamento dos nossos objectos, faz sentido que também seja ele a definir/implementar essas relações. Por isso o eloquent disponibiliza um conjunto de métodos que nos permitem establecer relações entre as nossas classes modelo. Este tipo de abordagem permite-nos um código limpo e fornece-nos recursos de encadeamento de métodos de consulta. Vejamos o seguinte exemplo:

```
$user->posts()->where('active', 1)->get();
```

Existindo uma relação entre user e post, podemos encadear a query.
Neste caso estamos a retornar todos os posts activos de um utilizador.
Em SQL ficaria algo do género:

```
SELECT * FROM users
INNER JOIN posts ON users.id = posts.user_id AND posts.active = 1;
```

Laravel workshop – Eloquent (Tipos de relação)

Um para um

Como vimos no módulo de bases de dados, temos uma relação "um para um" quando existe um elemento (e apenas um) de uma entidade que "pertence" a outra entidade e vice-versa. Imaginemos como exemplo a relação de uma entidade utilizador e de outra entidade telefene em que um utilizador tem apenas um telefone.

```
class User extends Model {
   public function phone() {
       return $this->hasOne('App\Models\Phone');
   }
}
```

Com esta relação, agora, sempre que selecionamos um utilizador da base de dados, temos também o modelo de telefone correspondente e não apenas o ID

```
$phone = User::find(1)->phone;
```

Laravel workshop – Eloquent (Tipos de relação)

Um para um (inverso)

Já vimos como podemos aceder ao modelo de telefone apartir do nosso utilizador. Agora, vamos ver como fazer a relação inversa, ou seja, como o modelo de telefone que nos permitirá aceder ao respectivo utilizador que possui o telefone.
Podemos definir o inverso de um relacionamento hasOne usando o método belongsTo:

```
class Phone extends Model {
    public function user() {
        return $this->belongsTo('App\Models\User');
    }
}
```

Com esta relação agora sempre que selecionamos um telefone da base de dados, temos também o modelo de utilizador correspondente e não apenas o ID

```
$user = Phone::find(1)->user;
```

Laravel workshop – Eloquent (Tipos de relação)

Um para muitos

Um relacionamento um para muitos é usado para definir relacionamentos em que um único modelo possui varios elementos de outro modelo. Por exemplo, um post de blog pode ter um número infinito de comentários. Vejamos um exemplo de uma possivel relação um para muitos

```
class Post extends Model {
    public function comments() {
        return $this->hasMany('App\Models\Comment');
    }
}
```

Neste caso o resultado será uma lista de instâncias do modelo Comment

```
$comments = App\Models\Post::find(1)->comments;
foreach ($comments as $comment) { // }
```

Laravel workshop – Eloquent (Tipos de relação)

Muitos para um

Já vimos como podemos aceder a todos os comentários de um post.
Vamos definir um relacionamento para permitir que um comentário aceda ao seu post pai. Para definir o inverso de um relacionamento hasMany, podemos usar uma função de relacionamento no modelo filho que chama o método belongsTo:

```
class Comment extends Model {
    public function post() {
        return $this->belongsTo('App\Models\Post');
    }
}
```

Podemos agora aceder ao post apartir do comment

```
$post = App\Models\Comment::find(1)->post;
```

Laravel workshop – Eloquent (Tipos de relação)

Muitos para Muitos

As relações muitos para muitos são ligeiramente mais complicadas do que as relações hasOne e hasMany, isto porque como sabemos, para definir uma relação muitos para muitos precisamos de 3 tabelas na base de dados. Uma para cada entidade e uma intermediária que deverá conter como chave estrangeiras as chaves primárias das entidades a relacionar.

Imaginemos o seguinte exemplo: *Um utilizador tem muitas funções, em que as funções também são compartilhadas por outros utilizadores*.
Teriamos então como possivel solução de base de dados as seguintes tabelas: user, role e role_user. A tabela role_user é derivada dos nomes de modelos relacionados e contém as colunas user_id e role_id.

Laravel workshop – Eloquent (Tipos de relação)

Muitos para Muitos

Estrutura dos dados

```
users
    id - integer
    name - string

roles
    id - integer
    name - string

role_user
    user_id - integer
    role_id - integer
```

Laravel workshop – Eloquent (Tipos de relação)

Muitos para Muitos

No caso de uma relação muitos para muitos os resultados podem ser definidos escrevendo um método que retorna o resultado do método belongsToMany. Por exemplo, vamos definir o método que retorna as funções desempenhadas pelo nosso utilizador:

```
class User extends Model {
    public function roles() {
        return $this->belongsToMany('App\Models\Role');
    }
}
```

Este método irá retornar todas as "roles" do nosso utilizador.

```
$user = App\Models\User::find(1);
foreach ($user->roles as $role) { // }
```

Laravel workshop – Eloquent (Tipos de relação)

Muitos para Muitos (inverso!)

Uma relação muitos para muitos não tem propriamente um inverso, visto ser a mesma relação de "ambos os lados" por isso podemos usar o mesmo método belongsToMany usado no exemplo anterior. Continuando com o mesmo exemplo vamos agora retornar quais os utilizadores que desempenham determinada função.

```
class Role extends Model {
    public function users() {
        return $this->belongsToMany('App\Models\User');
    }
}
```

Este método irá retornar todos os utilizadores que desempenham a função 1

```
$role = App\Models\Role::find(1);
foreach ($role->users as $user) { // }
```

Laravel workshop – Eloquent (Tipos de relação)

Parametros de uma relação

O Eloquent determina a chave estrangeira do relacionamento com base no nome dos modelos. No caso anterior, assume-se automaticamente que o modelo do telefone possui uma chave estrangeira phone_id na tabela user.
Ou seja, por padrão, o Eloquent assumirá que este valor é other_model_name_id.
Para substituir essa convenção, podemos passar um segundo argumento para o método hasOne:

```
class User extends Model {
    public function phone() {
        return $this->hasOne('App\Models\Phone', 'foreign_key');
    }
}
```

Neste caso a chave estrangeira esperada na base de dados na tabela users passa a ser foreign_key em vez do padrão user_id.

Laravel workshop – Eloquent (Tipos de relação)

Parametros de uma relação

Da mesma forma que o eloquent determina o foreign_key por convenção também determina a local_key do modelo que estamos a relacionar.
Por padrão, o Eloquent assumirá que este valor é id (chave primária do modelo atual), mas podemos explicitar no terceiro parâmetro se não for o caso.

```
class Phone extends Model {
    public function user() {
      return $this->belongsTo('App\Models\User', 'foreign_key', 'local_key');
  }
}
```

Neste caso a chave primária esperada na base de dados na tabela phone passa a ser local_key em vez do padrão id.

Laravel workshop – Eloquent (Tipos de relação)

Parametros de uma relação

No caso da relação belongsToMany podemos também especificar qual o nome da tabela de relação isto porque o nosso eloquent por padrão vai escolher o nome da tabela de relação contactando em snake_case o nome das tabelas do nosso modelo.

```
class User extends Model {
  public function roles() {
     return $this->belongsToMany('App\Models\Role',  'role_user', 'user_id', 'role_id');
  }
}
```

Se quisermos alterar a convenção adotada pelo eloquent podemos reescrever o nome da tabela de relação, bem como as chaves estrageiras que representam cada uma das entidades.

Laravel – File Storage

Laravel workshop – Ficheiros

Laravel Filesystem

O Laravel oferece uma poderosa abstração de sistema de ficheiros. A integração do Laravel Flysystem fornece drivers simples de usar para trabalhar com sistemas de ficheiros locais bem como integração com sistemas externos como o Amazon S3. Melhor ainda, é incrivelmente simples alternar entre essas opções de armazenament o, pois a API permanece a mesma para cada sistema. Isto tudo graças á sua abstração e ao pacote Flysystem desenvolvido por Frank de Jonge

Laravel workshop – Ficheiros

Configuração

- O ficheiro de configuração do sistema de ficheiros está localizado em config / filesystems.php.
- Dentro deste arquivo é possível configurar todos os seus "discos".
- Cada disco representa um driver de armazenamento e local de armazenamento específicos.
- Configurações de exemplo para cada driver suportado estão incluídas no ficheiro de configuração. Para o programador as poder alterar conforme as suas necessidades.
- É possível configurar quantos discos quisermos e podemos até ter vários discos que usam o mesmo driver.

Laravel workshop – Eloquent (Tipos de relação)

Disco público

O disco público destina-se a arquivos que serão acessíveis publicamente.
Por padrão, o disco público usa o driver local e armazena esses ficheiros em storage/app/public.
Para torná-los acessíveis na web, devemos criar um link simbólico de public/storage para storage/app/public. Esta convenção manterá os ficheiros publicamente acessíveis num diretório que pode ser facilmente compartilhado diferentes implementações sem a necessidade de ter exatamente a mesma estrutura de diretórios.

Para criar o link simbólico, você pode usar o comando storage: link Artisan:

```
php artisan storage:link
```

Laravel workshop – Eloquent (Tipos de relação)

Disco público

Podemos configurar links simbólicos adicionais no ficheiro de configuração. Cada um dos links configurados será criado quando o comando php artisan storage:link for executado.

```
'links' => [
    public_path('storage') => storage_path('app/public'),
    public_path('images') => storage_path('app/images'),
],
```

Depois que um arquivo foi armazenado e o link simbólico foi criado, podemos criar um URL para os ficheiros usando o asset helper:

```
echo asset('storage/file.txt');
```

Laravel workshop – Eloquent (Tipos de relação)

Disco local

Ao usar o driver local, todas as operações de ficheiros são relativas ao diretório raiz definido no arquivo de configuração do sistema de arquivos.
Por padrão, esse valor é definido para o diretório storage/app. Portanto, o método a seguir armazenaria um arquivo em storage/app/file.txt:

```
Storage::disk('local')->put('file.txt',  'Contents');
```

Pode ler mais acerca do laravel filesystem aqui.

Laravel workshop – Eloquent (Tipos de relação)

Permissões

As permissões por defeito que os ficheiros vão assumir ao serem armazenados nos nossos "discos" estão definidas no ficheiro de configuração.

```
'local' => [
    'driver' => 'local',
    'root' => storage_path('app'),
    'permissions' => [
        'file' => [
            'public' => 0664,
            'private' => 0600,
        ],
        'dir' => [
            'public' => 0775,
            'private' => 0700,
        ],
    ],
],
```

Pode ler mais acerca do laravel filesystem aqui.

Laravel – Links úteis

Laravel workshop – Links úteis

- Documentação
- Laracasts – Centenas de tutorias desenvolvidos pela comunidade laravel que abordam todas as fazes de desenvolvimento, teste e deploy de uma aplicação laravel
- Laravel ecosystem – Um conjunto de packages e ferramentas desenvolvidos para laravel
- Php the right way - Um conjunto de design patterns e boas práticas no desenvolvimento de aplicações PHP.
- Craftable - Aplicação construída em Laravel focada para facilitar o desenvolvimento de backoffice, disponibiliza uma UI "pré-feita" bem como um Crud-Generator, um módulo de traduções, etc.
- Algumas ferramentas laravel

Bibliografia

1. Micro vs Full Stack Frameworks - appdynamics
2. Laravel Framework doc - laravel