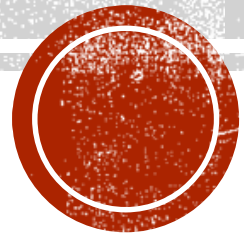
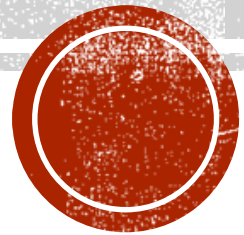
DESENVOLVIMENTO DE SISTEMAS WEB COM O FRAMEWORK LARAVEL

Thyago Maia Tavares de Farias



O QUE É UM FRAMEWORK?



FRAMEWORK

- Possui várias definições:
  - “Um conjunto de classe que encapsula uma abstração de projeto para a solução de uma família de problemas relacionados”;
  - “Um conjunto de objetos que colaboram para realizar um conjunto de responsabilidades para um domínio de subsistema de aplicativos”;
  - “Define um conjunto de classes abstratas e a forma como os objetos dessas classes colaboram”;
  - “Um conjunto extensível de classes orientadas a objetos que são integradas para executar conjuntos bem definidos de comportamento computacional”;
  - “Uma coleção abstraída de classes, interfaces e padrões dedicados a resolver uma classe de problemas através de uma arquitetura flexível e extensível”;

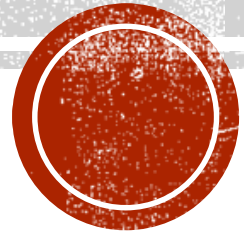
FRAMEWORK

- Observe que um framework é uma aplicação "quase" completa, mas com "pedaços" faltando:
  - Ao receber um framework, seu trabalho consiste em prover os "pedaços" que são específicos para sua aplicação;
  - Um framework provê funcionalidades genéricas, mas pode atingir funcionalidades específicas, por configuração, durante a programação de uma aplicação;

FRAMEWORK

- Vantagens:
  - Maior facilidade para detecção de erros;
  - Eficiência na resolução de problemas;
  - Otimização de recursos;
  - Concentração na abstração de solução de problemas;
  - Modelados com vários padrões de projeto;

O FRAMEWORK LARAVEL

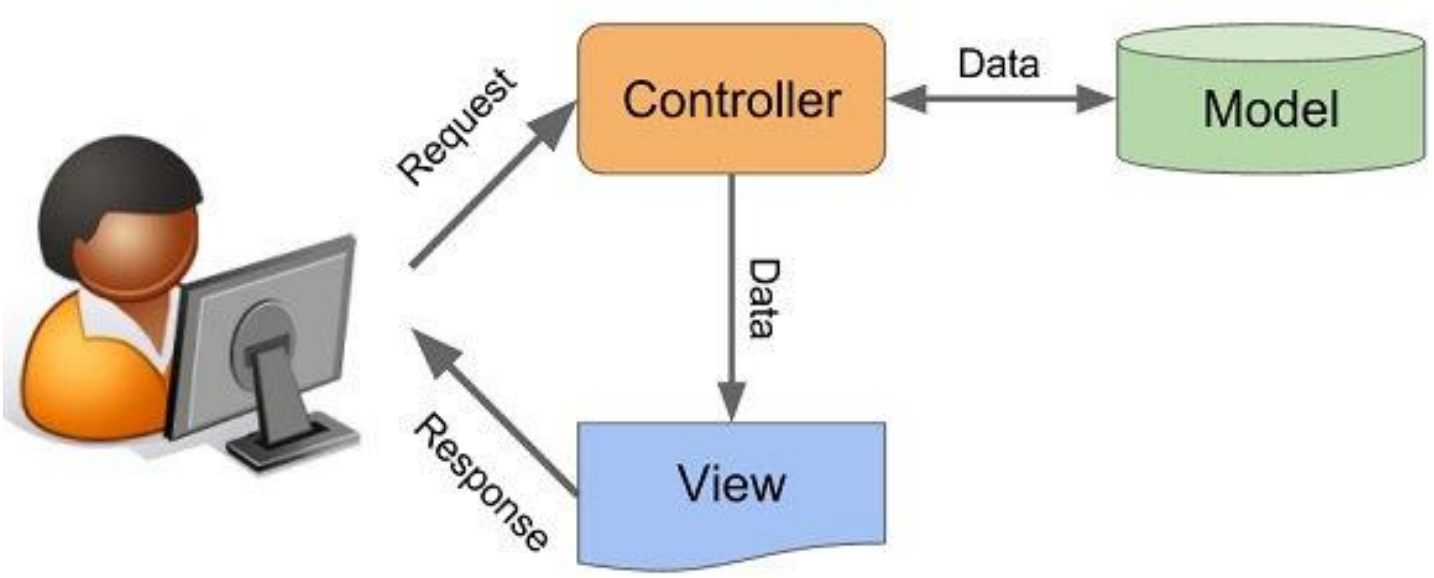


LARAVEL

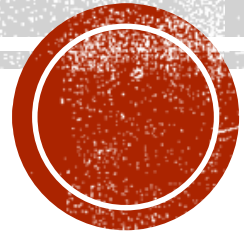
- Framework PHP open-source criado por Taylor Otwell;
- Objetiva o auxílio no desenvolvimento de aplicações Web baseados no padrão de projeto arquitetural MVC;
- Se tornou recentemente um dos frameworks PHP mais populares, ao lado do Symfony, Zend, CodeIgniter, entre outros;
- Hospedado no GitHub e licenciado nos termos da licença MIT;

LARAVEL

- O modelo MVC:



O AMBIENTE DE DESENVOLVIMENTO LARAVEL



AMBIENTE DE DESENVOLVIMENTO LARAVEL

- É possível configurar um ambiente de desenvolvimento Laravel a partir do Xampp. Para isso, será necessário a instalação do Composer;

AMBIENTE DE DESENVOLVIMENTO LARAVEL

- A partir de um prompt de comando, acesse a pasta C:\xampp\htdocs e execute o seguinte comando:

```
composer create-project laravel/laravel laravel "5.1.33"
```

- Será criada a pasta laravel na pasta htdocs do xampp, já com os arquivos de projeto Laravel. Para fazer com que o projeto fique disponível, no prompt de comando, digite:

cd laravel

php artisan serve

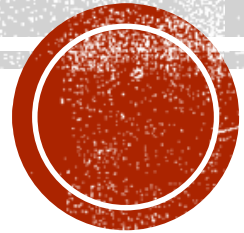
AMBIENTE DE DESENVOLVIMENTO LARAVEL

- Inicie o PHP e o MySQL no control panel do Xampp, clicando nos botões Start;

- Acesse a URL localhost:8000 e verifique se a página de boas vindas do Laravel será apresentada!

Laravel 5

REST



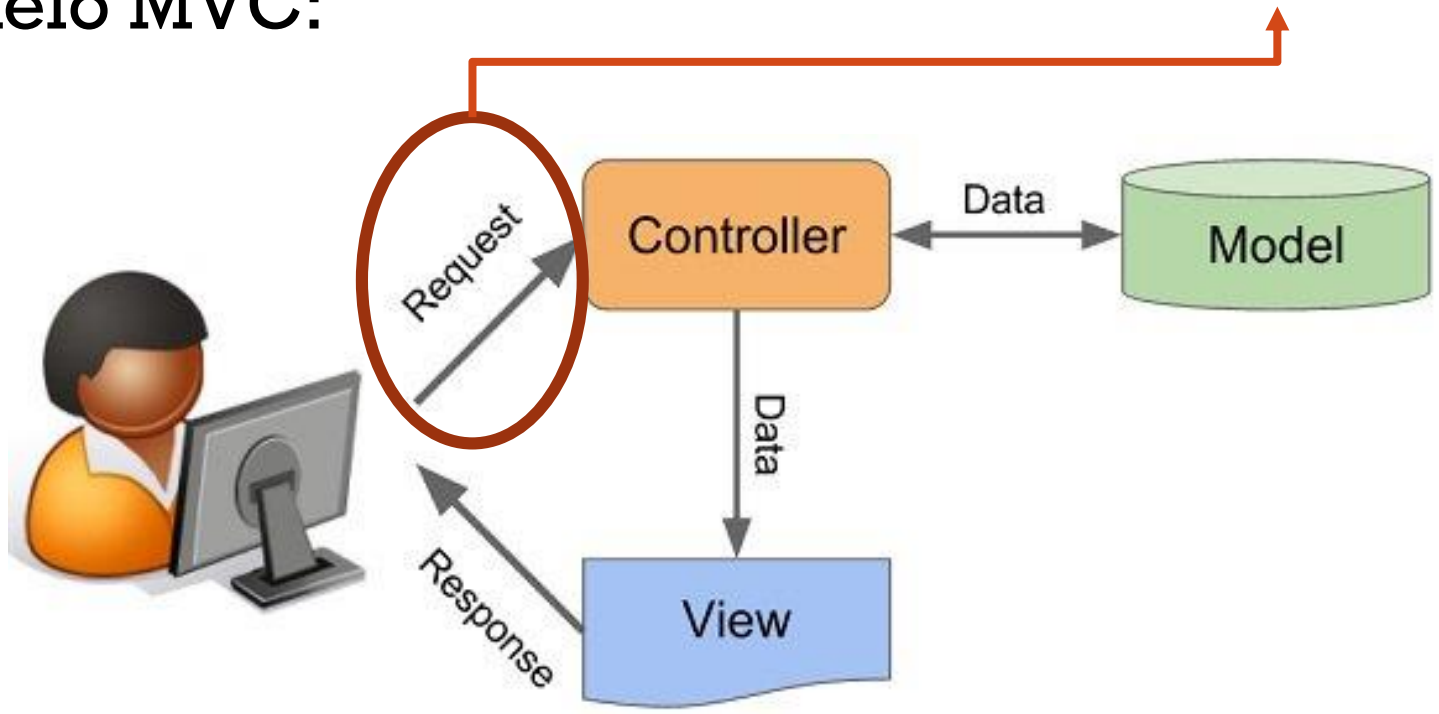
LARAVEL

- Seleciona controllers e/ou métodos de controller a partir do modelo REST:
  - Cada mensagem HTTP contém toda a informação necessária para compreensão de pedidos;
  - Utiliza as operações HTTP para a seleção de controllers e recursos de controller: POST, GET, PUT e DELETE;
  - Classifica operações de CRUD para a persistência de dados.
    - Ex.: Quando uma requisição HTTP do tipo DELETE é lançada para um controller, um método de exclusão de dados em banco poderá ser automaticamente executada;

LARAVEL

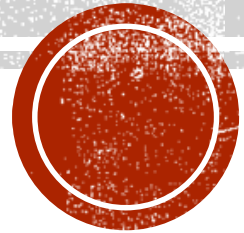
Utiliza um arquivo php de rotas para checar a operação HTTP requisitada e associá-la a um controller e/ou recurso de controller.

- O modelo MVC:



ROTAS LARAVEL

O arquivo Http/web.php



ROTAS LARAVEL

- A partir do arquivo web.php, é possível atribuir uma operação HTTP a um controller e/ou recurso de controller;
  - Ex.: Quando uma requisição HTTP do tipo get for lançada, execute o método da classe controller PrincipalController que apresenta a página de boas-vindas de uma dada aplicação web;

- Graças ao arquivo de rotas, não é necessário indicar na URL o recurso a ser acessado;
  - Ex.: Ao invés do link http://localhost:8000/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;

ROTAS LARAVEL

- 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>('/verbo',  function() {
      // O que será feito quando essa rota for acessada
});
```

ROTAS LARAVEL

- Exemplo: Abra o arquivo Http/web.php e crie a seguinte rota:

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

- Abra o navegador, acesse http://localhost:8000/php-info e verifique se a página de informações sobre o servidor PHP será apresentada;

ROTAS LARAVEL

- Exemplo 2: Abra o arquivo Http/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">
      <input type="submit" value="Enviar">
      </form>';
});
```

ROTAS LARAVEL

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

```
Route::post('/contato',  function() {
      echo Request::input('nome');
      echo "<br/>";
      echo Request::input('email');
});
```

ROTAS LARAVEL

- Abra o navegador, acesse 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?

ROTAS LARAVEL

- Abra o navegador, acesse 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!);

ROTAS LARAVEL

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

```
/**
 * The application's global HTTP middleware stack.
 *
 * @var array
 */
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
    \App\Http\Middleware\EncryptCookies::class,
    \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
    \Illuminate\Session\Middleware\StartSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,
    //\App\Http\Middleware\VerifyCsrfToken::class,
];
```

ROTAS LARAVEL

- Abra o navegador, acesse 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;

ROTAS LARAVEL

- Métodos da classe Route disponíveis para rotas:

- Route::get($uri, $callback); - Utilizado de forma geral para navegar entre páginas;
- Route::post($uri, $callback); - Utilizado de forma geral para alterações no lado do servidor;
- Route::put($uri, $callback); - Utilizado de forma geral para atualizações de um recurso existente;
- Route::patch($uri, $callback); - Utilizado de forma geral para atualizações de um recurso existente;
- Route::delete($uri, $callback); - Utilizado de forma geral para remover um recurso existente;

ROTAS LARAVEL

- Também é possível definir vários verbos para uma única rota com o método match:

```
Route::match(['POST','GET'], '/formulario',  function() {
        if(Request::isMethod('post'))
                print_r(Request::all());
        else {
        return '
        <form method="post" action="/formulario">
        Nome: <input type="text" name="nome">
        Email: <input type="text" name="email">
        Mensagem: <textarea name="mensagem"></textarea>
        <input type="submit" value="Enviar">
        </form>';
        }
});
```

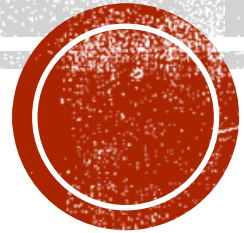
ROTAS LARAVEL

- Também é possível definir todos os verbos para uma única rota com o método any:

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

ROTAS COM PARÂMETROS

O arquivo Http/web.php



ROTAS LARAVEL

- Até o momento, não passamos nenhum parâmetro nas nossas requisições (rotas). Mas em muitas aplicações, essa necessidade pode surgir;

- Para isso, a sintaxe na definição do verbo muda um pouco, onde será necessário explicitar os parâmetros na sua definição;

ROTAS LARAVEL

- Exemplo: Abra o arquivo Http/web.php e crie a seguinte rota:

```
Route::get('/contato/{id}',  function($id) {
      echo "Olá, o seu ID é {$id}";
});
```

- Abra o navegador, acesse http://localhost:8000/contato/25 e verifique se o ID 25 é apresentado no navegador. Se nenhum parâmetro for enviado, será gerada uma exceção;

ROTAS LARAVEL

- 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;

ROTAS LARAVEL

- Exemplo: Abra o arquivo Http/web.php e crie a seguinte rota:

```
Route::get('/contato/{id}',  function($id) {
        echo "Olá, o seu ID é {$id}";
})->where('id','[0-9]+');
```

- Abra o navegador, acesse http://localhost:8000/contato/sport e verifique se o ID 'Sport' é apresentado no navegador. Se nenhum parâmetro for enviado, será gerada uma exceção;

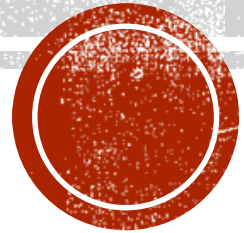
ROTAS LARAVEL

- Exemplo: Podemos passar mais de um parâmetro. Abra o arquivo Http/web.php e edite a seguinte rota:

```
Route::get('/contato/{id}/{nome}', function($id,$nome) {
        echo "Olá {$nome}, o seu ID é {$id}";
});
```

- Abra o navegador, acesse http://localhost:80000/contato/25/Thyago e verifique a saída apresentada no navegador. Se nenhum parâmetro for enviado, será gerada uma exceção;

ROTAS E CONTROLADORES



ROTAS LARAVEL

- E se minha aplicação tiver mil rotas????
  - Você não precisa criar todas as rotas no arquivo web.php!
  - Podemos mapear um controller no arquivo de rotas e, a partir dele, aplicar as rotas;
  - Para isso, em cada parâmetro passado na rota, devemos criar o parâmetro correspondente no método controller;

ROTAS LARAVEL

- O melhor? O Laravel cria o controller para você automaticamente!!!

- Para isso, vá no seu terminal, acesse a partir dele a pasta C:\xampp\htdocs\nomeProjeto e digite o comando:

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

- Um novo controller com o nome Rotas será criado em app/Http/Controllers;

ROTAS LARAVEL

- Agora, só precisaremos de uma linha no arquivo de rotas:

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

- Abra o navegador, acesse http://localhost:8000/rotas. O método index do Controller Rotas será mapeado automaticamente na requisição;

ROTAS LARAVEL

- No controller app/Http/Controllers/Rotas.php, atualize o seguinte método:

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

- Abra o navegador, acesse http://localhost:8000/rotas

ROTAS LARAVEL

- Também podemos criar o nome que desejarmos para nossas rotas. Exemplo: Logo abaixo do método index, crie o método:

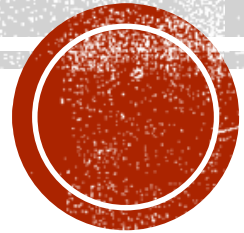
```
public function getRequisicao()     {
        return 'Olá, sou executado a partir de um GET';
}
```

- No arquivo de rotas, adicione a seguinte rota:

```
Route::get('/teste', 'Rotas@getRequisicao');
```

- Abra o navegador, acesse http://localhost:8000/teste

MIDDLEWARE HTTP



MIDDLEWARE HTTP

- Classes Laravel utilizadas para filtrar os dados de entrada dos nossos sistemas;

- Funciona como um filtro de requisição, que executa ações antes ou depois de uma requisição HTTP;

MIDDLEWARE HTTP

- Exemplo: Podemos inserir um token CSRF em formulários implementados em Views para que apenas nossos forms realizem requisições. Precisamos religar esse recurso, retirando o comentário da linha de código que implementa esse recurso no arquivo app/Http/Kernel.php;

```
/**
 * The application's global HTTP middleware stack.
 *
 * @var array
 */
protected $middleware = [
    \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
    \App\Http\Middleware\EncryptCookies::class,
    \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
    \Illuminate\Session\Middleware\StartSession::class,
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,
    //\App\Http\Middleware\VerifyCsrfToken::class,
];
```

MIDDLEWARE HTTP

- Exemplo: Agora, vamos criar nossa primeira View! Vá para a pasta resources/views e crie o arquivo contato.blade.php. Edite-o com o seguinte código HTML:

```
<html>
        <body>
                <form method="post" action="enviar-contato">
                {{ csrf_field() }}
                <p>Nome: <input type="text" name="nome"></p>
                <p>Telefone: <input type="text" name="telefone"></p>
                <p><input type="submit" value="Enviar"></p>
                </form>
        </body>
</html>
```

MIDDLEWARE HTTP

- Exemplo: Agora, vamos criar nossa primeira View! Vá para a pasta resources/views e crie o arquivo contato.blade.php. Edite-o com o seguinte código HTML:

```
<html>
        <body>
                <form method="post" action="enviar-contato">
                {{ csrf_field() }}
                <p>Nome: <input type="text" name="nome"></p>
                <p>Telefone: <input type="text" name="telefone"></p>
                <p><input type="submit" value="enviar"></p>
                </form>
        </body>
</html>
```

Olha o middleware aqui gente!!!

MIDDLEWARE HTTP

- Exemplo: Agora, vamos criar uma rota para chamar nosso formulário de contato no arquivo de rotas:

```
Route::get('/contato', 'ContatoController@contato');

Route::post('/enviar-contato',
'ContatoController@enviarContato');
```

MIDDLEWARE HTTP

- Em seguida, criaremos o controller responsável por exibir e receber dados do formulário;

- Para isso, vá no seu terminal, acesse a partir dele a pasta C:\xampp\htdocs\nomeProjeto e digite o comando:

```
php artisan make:controller ContatoController
```

- Um novo controller com o nome ContatoController será criado em app/Http/Controllers;

MIDDLEWARE HTTP

- Precisaremos criar os métodos contato() e enviarContato(Request $request) no controller em questão:

```
public function contato()     {
        return view('contato');
}
public function enviarContato(Request $request)      {
        return $request->all();
}
```

- Abra o navegador, acesse http://localhost:8000/contato