Desenvolvimento Web III

LARAVEL – AULA 2

PROFª MARIA CAROLINA

O que vimos na aula passada?

- Como abrir um novo projeto do Laravel
- Como definir rotas
- Como inserir uma view
- Aonde colocar arquivos estáticos (img, js, css)
- Primeiro projeto funcional no Laravel

Layouts com Blade

Section Directives

Layouts com Blade

- O seu topo e o seu rodapé são exatamente iguais para todas as páginas do seu projeto?

Não me diga! XD

- Com a definição de layouts, podemos reunir os elementos que se repetem para serem apenas referenciados

- Benefício: diminui as linhas de código das páginas, organização, código “mais limpo”

Como criar:

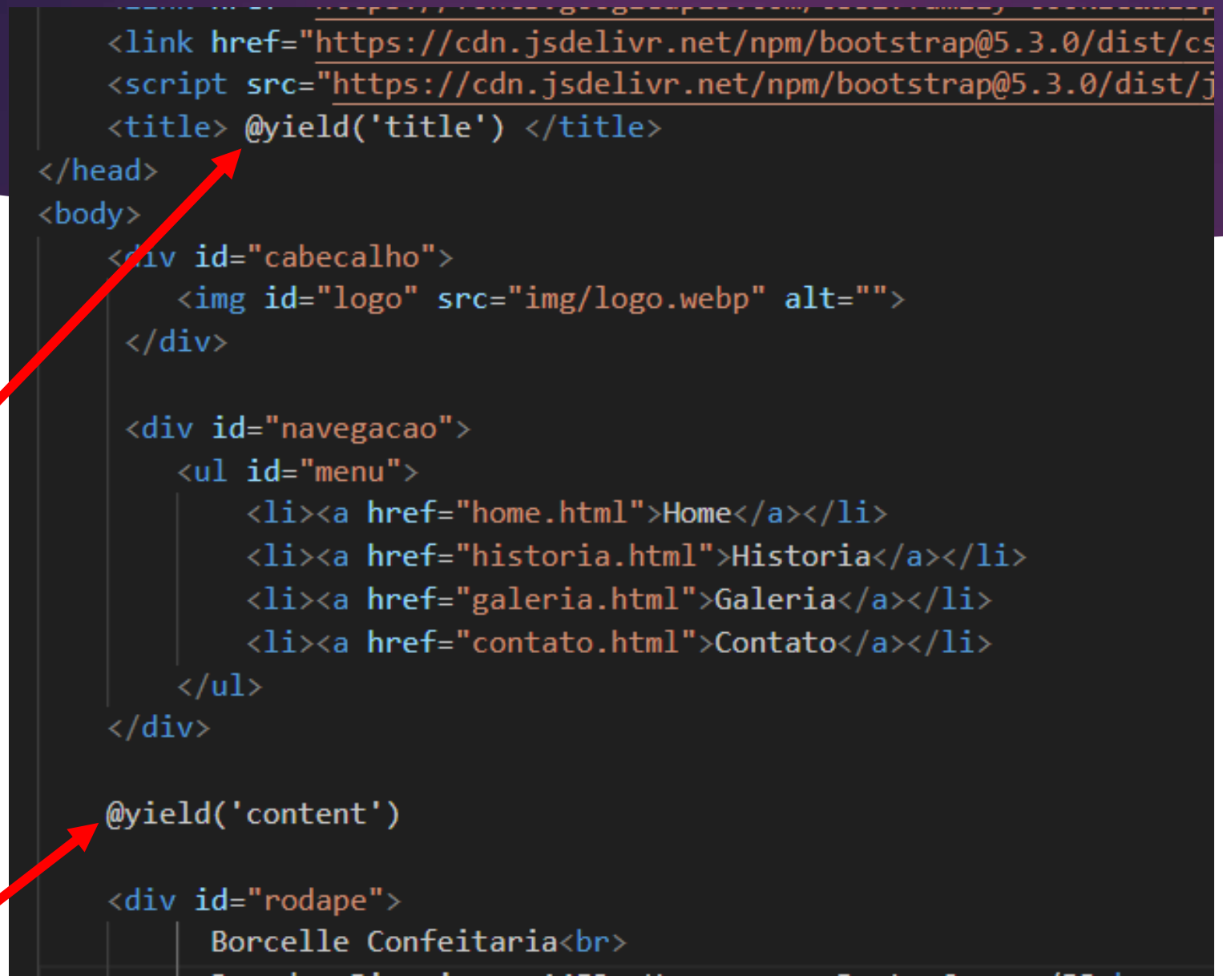
- Crie uma pasta "**layouts**" dentro da pasta **views**
- Crie um arquivo **main.blade.php** dentro da pasta layout
- Copie e cole o conteúdo de um html do seu projeto, e deixe apenas o que é padrão para todo o projeto (topo e rodapé)
- É nesse arquivo que vamos definir o que será mostrado para todas as páginas.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/estilo.css">
    <link href="https://fonts.googleapis.com/css2?family=Cookie&display=swap" rel
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.m
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.b
    <title>     </title>
</head>
<body>
    <div id="cabecalho">
        <img id="logo" src="img/logo.webp" alt="">
    </div>

    <div id="navegacao">
        <ul id="menu">
            <li><a href="home.html">Home</a></li>
            <li><a href="historia.html">Historia</a></li>
            <li><a href="galeria.html">Galeria</a></li>
            <li><a href="contato.html">Contato</a></li>
        </ul>
    </div>


    <div id="rodape">
        Borcelle Confeitaria<br>
        Rua dos Pioneiros, 1453, Uvaranas - Ponta Grossa/PR<br>
```

* no print consta um recorte do código.

- **Note que deixamos um espaço entre o título e as divs “navegacao“ e “rodape“**

Arquivo main.blade.php



- É aqui que vamos definir o que será alterado conforme a navegação do usuário.
- **Para isso, usamos a diretiva @yield seguida do parâmetro que vamos enviar**

Arquivo main.blade.php

Nas páginas de cada conteúdo:

- Importamos o layout através da diretiva **extends**

@extends('layouts.main')

- Usamos a diretiva **section** para enviar as informações como parâmetro:

@section('title', 'Fale Conosco')

@section('content')

...CONTEÚDO

@endsection

Exemplo – faleconosco.blade.php

```
@extends('layouts.main')

@section('title','Fale conosco')

@section('content')
<div id="contato">
        <form action="#">
            <label>Nome:</label>
            <input class="form-control" type="text" name="nome">
            <label>Telefone:</label>
            <input class="form-control" type="tel" name="telefone">
            <label>E-mail</label>
            <input class="form-control" type="email" name="email">
            <label>Digite a sua mensagem:</label>
            <textarea class="form-control" name="mensagem" cols="30" rows="10"></textarea>
            <input class="btn btn-primary" type="submit"  value="Enviar">
        </form>

        <iframe src="https://www.google.com/maps/embed?pb=!1m14!1m8!1m3!1d14452.999928855523!2
    </div>
@endsection
```

Atividade 1:

- Altere seu projeto criando um layout padrão para utilizar nas suas páginas

View/Blade

Views / Blades

- Podemos incrementar códigos php nas nossas views!
- Desta forma, também poderemos trabalhar com informações que posteriormente virão do banco de dados e também com tags HTML
- Isso fará com que o conteúdo fique dinâmico! ☺
- Vamos criar uma rota e uma view/blade para um teste?

Adicione uma nova Rota:

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

Adicione uma nova View com html inicial:

teste.blade.php

Diretivas blade

- São instruções especiais utilizadas para incorporar lógicas e funcionalidades do php diretamente página.

- Essas diretivas são reconhecidas pelo uso do símbolo **@** seguido de uma palavra-chave que indica o que precisa ser feito

- Exemplos mais comuns: **comentários, impressão de variáveis**, **estrutura condicional** e **laços de repetição**

Diretivas blade - Comentários

Comentário Blade: {{-- Este é um comentário no Blade **--}}**

É totalmente removido no HTML final

Comentário HTML: <!-- Este é um comentário HTML **-->**

Aparece no código-fonte final, que pode ser visto ao inspecionar a página no navegador.

Diretivas blade – Impressão de Variáveis

- Para imprimir variáveis, pode ser usado chaves duplas:

```
<body>
    @php
        $nome = “Josué”;
    @endphp

        {{ $nome }}
</body>
```

Diretivas blade - Estrutura condicional

```
@php
        $clientes = array("João", "Pedro", "Lucas");
@endphp

@if (count($clientes) == 0)
    <p>Não há clientes cadastrados</p>
@else
    <p>Há clientes cadastrados</p>
@endif
```

A função count conta quantos elementos existem no array. Nesse exemplo, ela vai retornar o número 3

Diretivas blade - Estrutura condicional II

```
@php
        $clientes = array("João", "Pedro", "Lucas");
@endphp

@if (count($clientes) == 0)
    <p> Não existe cliente cadastrado</p>
@elseif (count($clientes) == 1)
    <p> Há 1 cliente cadastrado </p>
@else
  <p>Foram encontrados {{ count($clientes) }} cliente cadastrados </p>
@endif
```

Diretivas blade – Laço For

```
@for ( $i = 0 ; $i < 5 ; $i ++)

    {{ $i }}

@endfor
```

Diretivas blade – Laço Foreach

```
@php
        $listaclientes = array("João", "Pedro", "Lucas");
@endphp

@ foreach ( $listaclientes  as  $cliente )

        {{ $cliente }}

@ endforeach
```

O objeto loop nos laços

```
@php
        $listaclientes = array("João", "Pedro", "Lucas");
@endphp

@ foreach ( $listaclientes  as  $cliente )
        {{ $loop->index }}
        {{ $cliente }}

@ endforeach
```

É uma propriedade que faz parte da classe de loop utilizada em diretivas Blade, e representa o índice atual **da iteração dentro do loop**, começando em 0(zero)

O objeto loop nos laços

$loop->iteration: Retorna o número da iteração atual, começando em 1.

$loop->remaining: Retorna o número de iterações restantes.

$loop->count: Retorna o número total de itens no loop.

$loop->first: Indica se a iteração atual é a primeira.

$loop->last: Indica se a iteração atual é a última.

Diretivas blade – Laço Foreach II

```
@php
  $listaclientes = array(
      0 => array('nome' => 'João', 'telefone' => '9999-9999'),
      1 => array('nome' => 'Maria', 'telefone' => '8888-8888'),
      2 => array('nome' => 'Pedro', 'telefone' => '7777-7777'),
      3 => array('nome' => 'Ana', 'telefone' => '6666-6666')
  );
  @endphp

  @foreach ($listaclientes as $posicao => $valor)
      <p>{{ $posicao }}: {{ $valor['nome'] }} - Telefone: {{ $valor['telefone'] }}</p>
  @endforeach
```

Diretivas blade – Laço Forelse

O **forelse** lida automaticamente com o caso em que o array está vazio. Muito útil na leitura do banco de dados para listagens.

```
@php
    $listaclientes = array();
 @endphp


    @forelse ($listaclientes as $posicao => $valor)
            <p>{{ $posicao }}: {{ $valor['nome'] }} - Telefone: {{ $valor['telefone'] }}</p>
    @empty
            <p> Não existe usuário cadastrado </p>
    @endforelse
```

Diretivas blade – Laço While

```
@php
        $listaclientes = array("João", "Pedro", "Lucas");
@endphp

@while ($cliente= array_pop($listaclientes))
   {{ $cliente }}
@endwhile
```

A função vai iterar sobre o array $listaclientes removendo o último nome ao exibí-lo na view, até que o array $listaclientes esteja vazio.

Diretivas blade – Laço While – Exemplo II

```
@php
        $listaclientes = array("João", "Pedro", "Lucas");
      $i = 0;
@endphp

@while($i < count($listaclientes))
      {{ $listaclientes[$i] }}
        @php
            $i++;
      @endphp
@endwhile
```

Mas profe, não é feio declarar as variáveis na view?

Mas profe, não é feio declarar as variáveis na view?

SIM

CONTRADITÓRIO

Então...COMO ENVIAR os dados para a view?

Com enviar dados para a view: Modo 1 - Através das rotas

Routes/web.php

- Dentro da função view() podemos enviar dados para uma blade.
- A função view() aceita um array de dados:

```
Route::get('/teste', function () {
    $produto = “maçã”;
    return view(‘teste‘ ,[‘item’=>$produto]);
});
```

Routes/web.php

- Dentro da função view() podemos enviar dados para uma blade.
- A função view() **aceita um array de dados**:

```
Route::get('/teste', function () {
    $produto = “maçã”;
    return view(‘teste‘ ,[‘item’=>$produto]);
});
```

É esse o nome da variável que vamos usar na blade para obter o valor “maçã”.

Resources/views/teste.blade.php

```
...
<body>

    <h1> {{ $item }} </h1>

</body>

...
```

Routes/web.php – **mais de um parâmetro**

- Dentro da função view() podemos enviar dados para uma blade.
- A função view() aceita um array de dados:

```
Route::get('/teste', function () {
    $produto = “maçã”;
    $valor = 12.30;
    return view(‘teste‘ ,[‘item’=>$produto, ‘preco’=>$valor]);
});
```

Resources/views/teste.blade.php

```
...
<body>

    <h1> {{ $item }} </h1>
    <h3>  Está custando : R$ {{ $preco }} </h3>

</body>

...
```

Routes/web.php – **função compact**

```
Route::get('/teste', function () {
        $produto = “cebola”;
        $valor = 1.50;
       return view(‘teste‘ , compact(‘produto’,valor’));
});
```

- Com a função compact() não precisamos declarar um array manualmente. Basta colocar o nome das variáveis definidas anteriormente, sem o $.

Resources/views/teste.blade.php

```
...
<body>

    <h1> {{ $produto }} </h1>
    <h3>  Está custando : R$ {{ $valor }} </h3>

</body>

...
```

Routes/web.php – **enviando um array**

```
Route::get('/teste', function () {
    $clientes = array("João", "Pedro", "Lucas");
    return view(‘teste‘ , compact(‘clientes‘);
});
```

ou...

```
Route::get('/teste', function () {
    $clientes = array("João", "Pedro", "Lucas");
    return view('teste' , [‘clientes’=>$clientes]);
});
```

Resources/views/teste.blade.php

```
...
<body>

    @forelse ($clientes as $cliente)
            <p>{{ $cliente }}</p>
    @empty
            <p> Não existe usuário cadastrado </p>
    @endforelse

</body>

...
```

Desafio – Tabuada temática

- Crie uma blade chamada tabuadatematica.blade.php
- Crie uma rota para esta blade, enviando como informação qual a tabuada desejada
- Na pasta public/img salve uma imagem para cada número, de 1 a 10
- Escolha um personagem, um animal, uma comida... como tema para sua tabuada, salve a imagem do tema na pasta public/img
- Na blade, mostre a imagem do número correspondente à tabuada escolhida e enviada pela rota

Desafio – Tabuada temática

- Ao lado de cada linha da tabuada, mostre tema sendo repetido o número de vezes do resultado:

2x0 = 0

2x1 = 2 

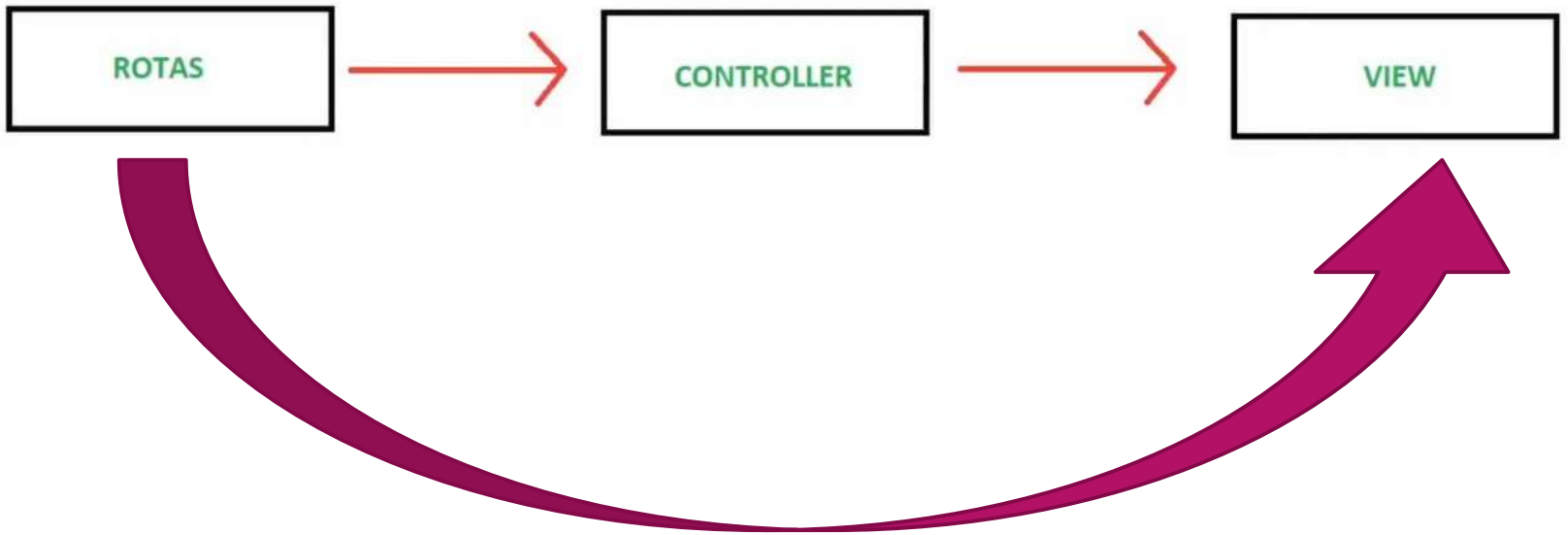
2x2 = 4 

2x3 = 6 

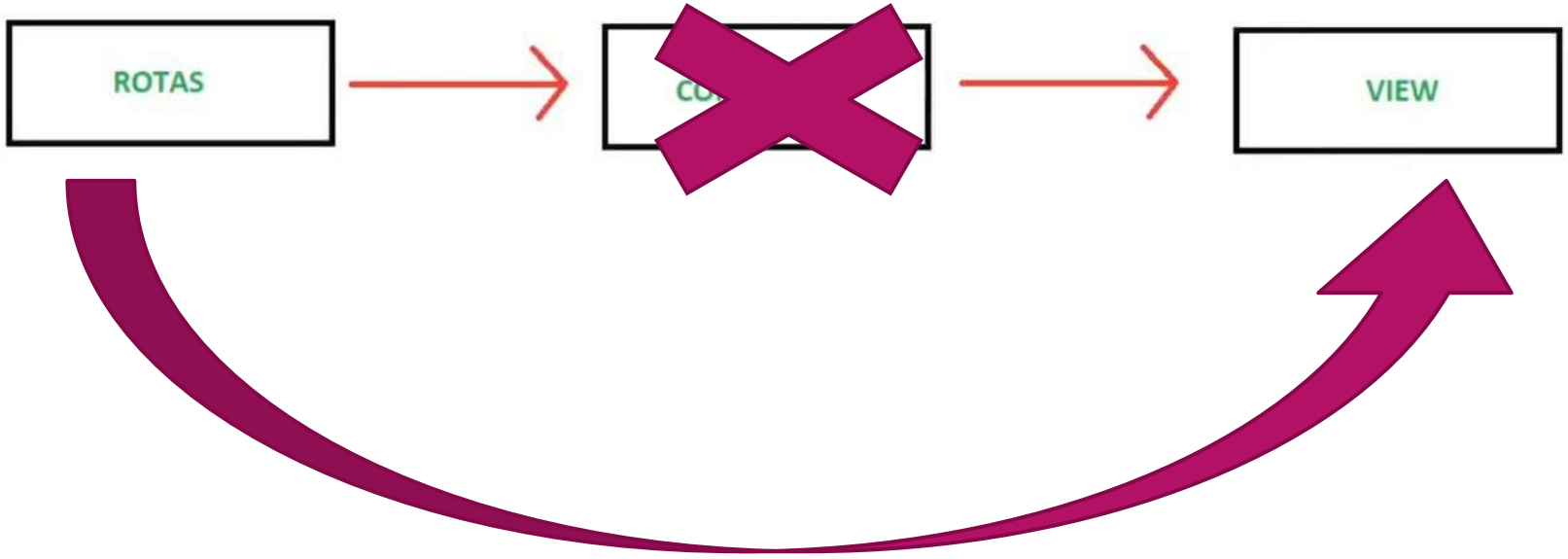
2x4 = 8 

E assim por diante...

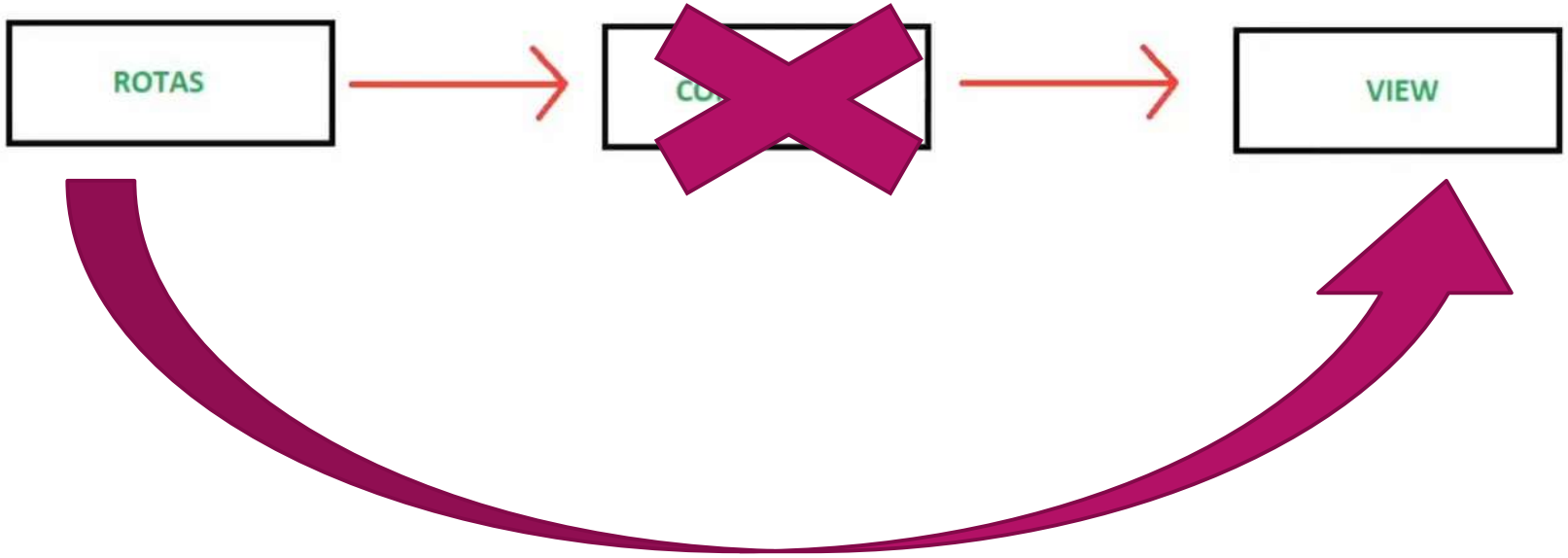
Modo 1
Enviando dados através das rotas



Modo 1
Enviando dados através das rotas



Modo 1
Enviando dados através das rotas



Agora vamos aprender como usar os controladores, e como enviar os dados para as views através deles

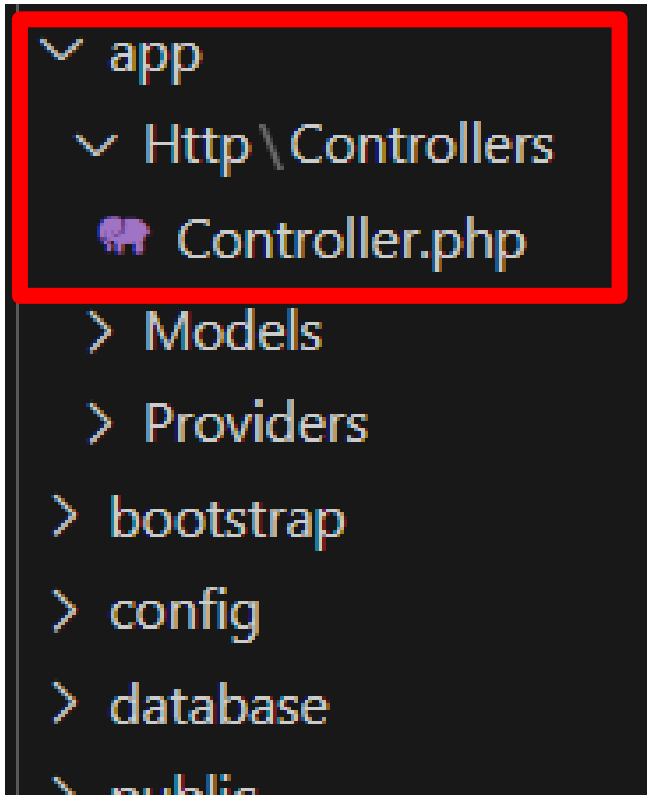
Com enviar dados para a view: Modo 2 - Através dos Controllers

Analisando...

- Perceberam que as **rotas chamam um controller**?

- Desta forma, vamos criar um controller para nossa rota.

Onde ficam os controllers no Laravel?



Comando artisan para criar um controller

- Pare de rodar sua aplicação Laravel através do comando CTRL+C no terminal

- Digite o comando

php artisan make:controller TesteController

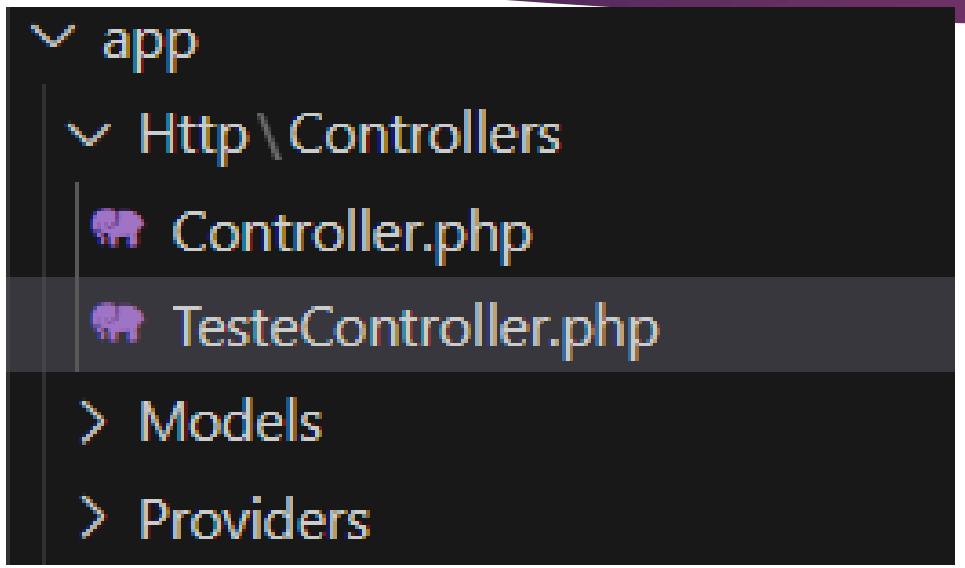
Comando artisan para criar um controller

- Pare de rodar sua aplicação Laravel através do comando CTRL+C no terminal

- Digite o comando

É esse o nome do controller criado

php artisan make:controller TesteController




```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TesteController extends Controller
{
    //
}
```

Actions dos Controllers

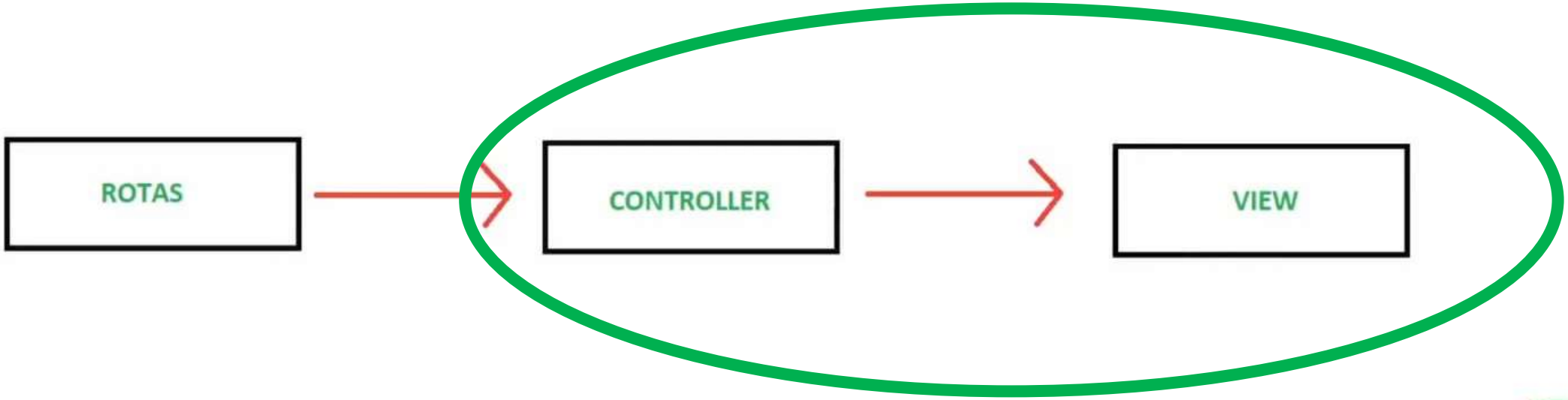
- Actions são funções que compõem os controllers
- São definidas dentro da classe, e realizam as regras lógicas da aplicação
- Nesse primeiro momento, vamos criar uma action para ser invocada por uma rota
- Dentro da action, definimos qual view ela vai "mandar aparecer" para o usuário

TesteController.php

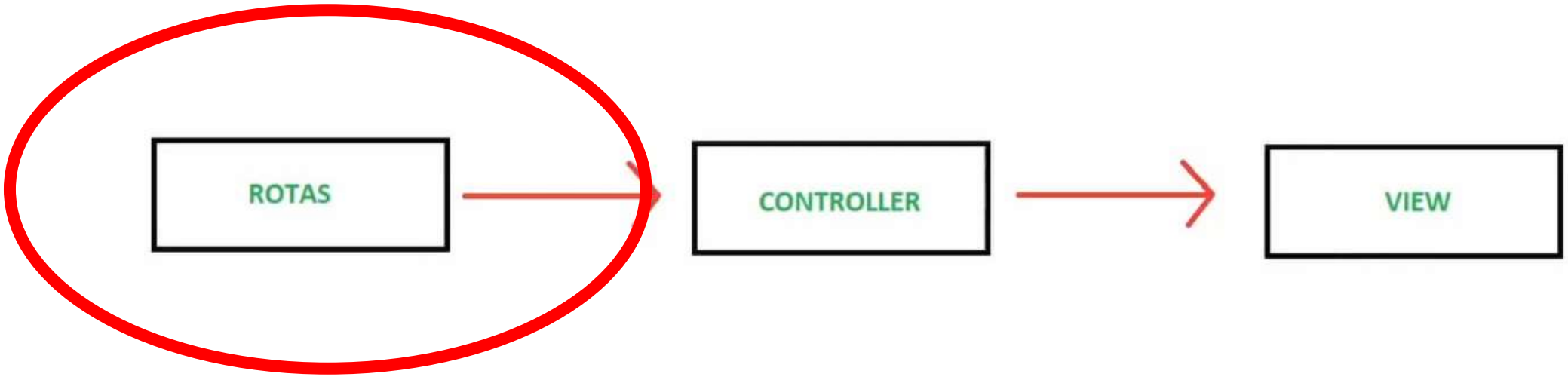
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TesteController extends Controller
{
    public function testar(){
        return view('teste');
    }
}
```

Action testar que vai retornar a view teste

Chegamos até aqui



Vamos para essa etapa



Rotas
web.php

- A primeira mudança no nosso código da rota é que não vamos invocar a function view

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

Vamos fazer a nossa rota chamar a action testar

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TesteController;

Route::get('teste', [TesteController::class, 'testar']);
```

Precisamos colocar a classe que vamos usar

Invocando a action testar da classe TesteController

Vamos fazer a nossa rota chamar a action testar

```php
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\TesteController;

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

Route::get('teste', function () {
    return view('teste');
});

Route::get('/teste', [TesteController::class, 'testar']);
```

Beleza, mas e os dados que eu passava pra view?

Exatamente como já aprendemos antes, mas agora enviamos via action do controller

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class TesteController extends Controller
{
    public function testar(){
        $nome = "Josué";
        return view('teste', compact('nome'));
    }
}
```

Atividade:

- Passe os exercícios da lista 1 de php (primeria aula) para o Laravel seguindo as instruções:

1. Crie um novo projeto Laravel para essa atividade
2. Crie uma blade, uma rota e uma action num controller para cada exercício da lista
3. Defina os valores das variáveis utilizadas no exercício via controller
4. Use as diretivas blade para implementação dos exercícios
5. Crie um menu como página principal para acessar os exercícios