DIO
CEZAR
GO

Rotas e Templates HTML

ROTEAMENTO

O QUE É ROTEAMENTO?

- quando trabalhamos com MVC sabemos que o controller é o *cara* responsável por decidir o que o model faz com os dados;
- mas ai te pergunto: e quem decide qual *controller* deve ser chamado?
- não é possível (recomendável) criar um único *controller* para comandar tudo;
- em muitos casos o roteamento pode ser feito ainda, diretamente nas configurações do servidor *web* (no caso o apache)
    - estudo complementar `.htaccess`

O QUE É ROTEAMENTO?

- então, devemos pensar em módulos!
- por exemplo, um módulo `ControllerUsuario.php`:
  - `index()` ← mostra a página principal
  - `salvar()` ← salva/atualiza um usuário
  - `excluir()` ← deleta um usuário
  - `consultar()` ← consulta um usuário

O QUE É ROTEAMENTO?

- então, cada módulo deve ser carregado por uma *URL*;
- e esse é o processo de roteamento;
- com o roteamento é possível ainda, acessar um caminho específico que irá simbolizar qual método será acessado de uma *controller*;
- por exemplo: `http://localhost/projeto/usuario/salvar`
  - essa URL irá acessar o método salvar do controller usuário;
  - deve-se enviar para essa URL um POST com os dados a serem salvos;
- ou até mesmo: `http://localhost/projeto/usuario/1`
  - essa URL irá devolver o usuário com id 1;

COMO FAZER UM ROTEAMENTO?

- essencialmente você poderia escrever toda a sua classe de roteamento;
- mas estaria com um trabalho considerável:
  - recortar as strings que são recebidas da url;
  - validar parâmetros;
  - entre outras complicações;
- por isso recomenda-se a utilização de alguma biblioteca, no nosso caso, utilizaremos como exemplo: *Slim*
  - https://www.slimframework.com/

MICRO FRAMEWORK SLIM

- o Slim é um *micro framework* que ajuda você a escrever simples e poderosas aplicações web e APIs;
- para usar o Slim, basta adicioná-lo em seu `composer.json`;

COMPOSER.JSON

```
{
    "require": {
        "slim/slim": "^2.6"
    }
}
```

O QUE ELE FAZ?

- com o Slim temos facilmente o controle de todas as rotas e o que elas devem chamar;
- utiliza-se para isso o conceito de *Clousure* que é a passagem de uma função como parâmetro de outra função;
- é possível ainda capturar parâmetros e utilizá-los nas chamadas de suas *controllers*;

EXEMPLO SLIM

EXEMPLO SLIM

```
<?php
      require_once("./api/autoload.php");
      Slim\Slim::registerAutoloader();
      $app = new \Slim\Slim();
      $app->get('/item', function(){
            // chama a controller responsável
      });
      $app->get('/item/:id', function($id){
            // chama a controller responsável e pode passar $id
      });
      $app->run();
?>
```

TEMPLATES HTML

O QUE SÃO TEMPLATES HTML

- vimos que o PHP pode ser colocado dentro de um “arquivo” HTML;
- isso resolve boa parte das situações na qual precisamos escrever dados de um banco de dados, por exemplo;
- mas nem de longe é a maneira padronizada e mais elegante para trabalhar com a disposição de dados;
- **DISCLAIMER** → ao trabalhar com o PHP imprimindo DIRETAMENTE em arquivos HTML estamos aplicando uma técnica alternativa as APIS Restfull que são a base para a aplicação de *frameworks* como Vue.js e React.js;

O QUE SÃO TEMPLATES HTML

- uma maneira alternativa para isso seria separar o HTML e o PHP;
- imagine um arquivo HTML puro;
- em determinado momento você quer que um dado venha do seu PHP (por exemplo de um banco de dados)
- então, em seu HTML escreva alguma identificação única para uma variável, por exemplo: `{{nome_do_usuario}}`

O QUE SÃO TEMPLATES HTML

- agora em seu PHP, recupere essa variável e armazene em $nome_usuario;
- abra o seu arquivo HTML (dentro do PHP);
- substitua `{{nome_do_usuario}}` por `$nome_usuario;`
- imprima então o resultado da substituição;
- pronto, essa é toda a base para o entendimento de um sistema simples de templates!

SISTEMA DE TEMPLATES SIMPLIFICADO

SISTEMA DE TEMPLATES SIMPLIFICADO

```
<?php
      $nome_usuario = "Diogo Cezar";
      if(!file_exists("template.html")) {
            echo "Error loading template file.";
      }
      $output = file_get_contents("template.html");
      $output = str_replace("{{nome_do_usuario}}", $nome_usuario, $output);
      echo $output;
?>
```

TWIG

- é um motor de *templates* para o PHP;
- possui a substituição de *tags* e muito mais;
- sua documentação está aqui → https://twig.symfony.com/

TWIG

- o twig trabalha com a definição de blocos e variáveis específicas nos arquivos HTML e com uma classe específica para configurar todo o sistema de substituição no PHP;
- em um arquivo principal, pode-se definir blocos específicos que deverão ser substituídos por outros templates que herdam o arquivo principal;

ARQUIVO FONTE LAYOUT.HTML

LAYOUT.HTML

```
<!DOCTYPE html>
<html>
    <head>
    {% block head %}
        <title>{% block title %}{% endblock %} - My Webpage</title>
        <link rel="stylesheet" href="style.css" />
    {% endblock %}
    </head>
    <body>
        <div id="content">{% block content %}{% endblock %}</div>
    </body>
</html>
```

ARQUIVO FILHO INDEX.HTML

INDEX.HTML

```
{% extends "layout.html" %} ← qual arquivo é o fonte

{% block title %}Principal{% endblock %} ← qual o conteúdo que o bloco título terá

{% block head %}
    {{ parent() }} ← importa o que tem definido no arquivo fonte
    <style type="text/css">
        .important { color: #336699; }
    </style>
{% endblock %}

{% block content %}
    <h1>My Webpage</h1> ← imprime este conteúdo em content
{% endblock %}
```

TWIG - FUNCIONALIDADES

- com o *twig* podemos inserir uma série de funcionalidades específicas, como por exemplo
  - blocos de repetição → https://twig.symfony.com/doc/2.x/tags/for.html
  - blocos de condição → https://twig.symfony.com/doc/2.x/tags/if.html
  - entre outros...

TWIG EXEMPLO FOR

USER_TEMPLATE.HTML

```
<!DOCTYPE html>
<html>
<head>
      <title>Usuários</title>
      <link rel="stylesheet" href="">
</head>
<body>
      <h1>{{title}}</h1>
      <ul>
          {% for user in users %}
              <li>
                  <h2>{{ user.name }}</h2>
                  <p>{{ user.email }}</p>
              </li>
          {% endfor %}
      </ul>
</body>
</html>
```

TWIG - E COMO PASSAR OS VALORES

- na parte do PHP, basta seguir os passos:
  - fazer o *include* do `autoload.php`
  - criar um objeto *loader* com o diretório dos *templates*;
  - criar um objeto *twig* com as configurações e passando o *loader* como parâmetro;
    - pode-se opcionalmente habilitar um *cache*, para que o PHP não precise ficar lendo arquivos a todo momento;
  - criar uma variável de *template* com um *template* em questão;
  - e imprimir o *template* passando como parâmetro um *array* com os valores a serem substituídos;

TWIG EXEMPLO FOR

USER_TEMPLATE.HTML

```
<?php
      $dir      = './templates';
      $html     = 'user_template.html';
      $loader   = new \Twig_Loader_Filesystem($dir);
      $twig     = new \Twig_Environment($loader, array('cache' => false));
      $template = $twig->loadTemplate($html);
      $values = array(
            'title' => 'Usuários',
            'users' => array('name' => 'Diogo', 'email' => 'diogo@diogocezar.com')
      );
      echo $template->render($values);
?>
```

MATERIAIS COMPLEMENTARES

- https://goo.gl/qM3zXF
- https://goo.gl/TgEXGG
- https://goo.gl/yaA3L6