RESUMO Nº 2905/2024/Ferramental

Quarta-feira, 29 de Maio de 2024

Aula 22 – Laravel - PHP (Danki)

- **INTRODUÇÃO**
  - I. **Estrutura de Pastas e Arquivos**

A estrutura de organização de código do Laravel é projetada para ser intuitiva e flexível, facilitando o desenvolvimento de aplicações web de todos os tamanhos. Aqui está um resumo dos principais diretórios e arquivos que você encontrará em um projeto Laravel padrão:

/app: Este diretório contém o núcleo do código da sua aplicação. Ele é dividido em várias subpastas, como Models para os modelos Eloquent, e pode ser personalizado para incluir outras classes como Services, Repositories, etc.

/bootstrap: Contém os arquivos que inicializam o framework e configuram o autoloading. O arquivo app.php neste diretório inicia o script do Laravel.

/config: Como o nome sugere, este diretório contém todos os arquivos de configuração da sua aplicação Laravel.

/database: Este diretório contém as migrações, os seeders (dados de teste) e as factories (fábricas de modelos para testes) da sua aplicação.

/public: É o diretório raiz da web. Contém o arquivo index.php, que é o ponto de entrada para todas as solicitações que entram na sua aplicação e configura o autoloading.

/resources: Contém as views (arquivos blade.php), arquivos de linguagem e outros arquivos que são necessários para a renderização da interface do usuário.

/routes: Este diretório contém todos os arquivos de rotas da aplicação. Laravel separa as rotas em arquivos como web.php para rotas web e api.php para rotas de API.

/storage: Usado para armazenar logs, sessões, arquivos compilados de Blade, caches e outros arquivos gerados pelo framework ou pela aplicação.

/tests: Contém os testes automatizados da sua aplicação, incluindo testes unitários e de feature.

/vendor: Este diretório contém as dependências do Composer da sua aplicação.

Além desses diretórios, você encontrará alguns arquivos importantes na raiz do projeto:

.env: Um arquivo de configuração de ambiente onde você pode definir variáveis de ambiente específicas do seu ambiente de desenvolvimento.

composer.json e composer.lock: Arquivos que especificam as dependências do PHP da sua aplicação e garantem que todos que trabalham no projeto tenham as mesmas versões dessas dependências.

artisan: Um arquivo PHP que permite interagir com o Laravel através da linha de comando para realizar tarefas como migrações de banco de dados, testes e execução de comandos personalizados.

II. MVC no Laravel

A estrutura do Laravel é projetada para promover uma organização clara do código e facilitar o desenvolvimento seguindo o padrão MVC (Model-View-Controller), além de adotar outras convenções e padrões de design para tornar o desenvolvimento mais eficiente e a manutenção mais fácil.

O padrão MVC (Model-View-Controller) é uma arquitetura de software que separa a aplicação em três componentes principais: Model, View e Controller. Essa separação ajuda a organizar o código, facilita a manutenção e permite o desenvolvimento paralelo. Vamos explorar cada um desses componentes:

Model
Responsabilidade: O Model é responsável pela lógica de negócios e pelo acesso aos dados. Ele representa a estrutura de dados da aplicação, incluindo as regras para modificar e manipular esses dados.
Características: Geralmente, o Model interage com um banco de dados através de ORM (Object-Relational Mapping) e contém funções para buscar, inserir e atualizar informações no banco.
View
Responsabilidade: A View é responsável pela apresentação dos dados ao usuário. Ela apenas exibe os dados fornecidos pelo Controller, sendo completamente separada da lógica de negócios.
Características: As Views são geralmente compostas por HTML, CSS e JavaScript, podendo também incluir templates que são preenchidos dinamicamente com dados.
Controller
Responsabilidade: O Controller atua como um intermediário entre o Model e a View. Ele recebe as requisições do usuário, processa essas requisições (possivelmente modificando dados através do Model) e então passa os dados para a View ser renderizada.
Características: O Controller define a lógica de controle de fluxo da aplicação, decidindo que View será mostrada ao usuário e com quais dados.

Fluxo de Trabalho no MVC
Requisição do Usuário: Tudo começa com uma requisição do usuário, que é recebida pelo Controller.
Processamento pelo Controller: O Controller interpreta a requisição, utilizando os Models necessários para buscar ou modificar os dados solicitados.
Interação com o Model: O Controller solicita ao Model os dados, que interage com o banco de dados ou qualquer outra fonte de dados.

Resposta do Model: O Model retorna os dados solicitados ao Controller.
Renderização da View: O Controller escolhe qual View deve ser usada para apresentar os dados e passa esses dados para a View.
Apresentação ao Usuário: A View gera o HTML (ou outro formato adequado) e o envia de volta ao usuário.
Vantagens do MVC
Separação de Concerns: Cada componente tem responsabilidades claras, facilitando a manutenção e a escalabilidade da aplicação.
Desenvolvimento Paralelo: Como as responsabilidades são bem definidas, diferentes equipes podem trabalhar em Models, Views e Controllers simultaneamente.
Reusabilidade de Código: Os Models podem ser reutilizados em diferentes partes da aplicação sem necessidade de duplicação de código.
Testabilidade: A separação clara facilita a escrita de testes unitários para cada componente.
O padrão MVC é amplamente utilizado em frameworks de desenvolvimento web, como Laravel (PHP), Ruby on Rails (Ruby), Django (Python) e muitos outros, devido à sua eficiência em organizar o código de aplicações complexas

A estrutura dos arquivos do Laravel está intimamente ligada ao padrão MVC (Model-View-Controller), um paradigma de arquitetura de software que divide a aplicação em três componentes principais para organizar o código, facilitar a manutenção e otimizar o desenvolvimento de aplicações web. Vamos explorar como essa correlação se manifesta no Laravel:

Model: Localizados no diretório **/app/Models**, os Models no Laravel representam a camada de dados da aplicação. Eles são responsáveis pela lógica de negócios e pela interação com o banco de dados, utilizando o Eloquent ORM para facilitar a manipulação e recuperação de dados de forma segura e eficiente.

View: As Views estão armazenadas em **/resources/views** e são encarregadas da apresentação dos dados ao usuário. O Laravel utiliza o motor de templates Blade para permitir a criação de layouts elegantes e a inclusão dinâmica de dados fornecidos pelos Controllers. As Views focam exclusivamente na interface do usuário, separando a lógica de apresentação da lógica de negócios.

Controller: Os Controllers, encontrados em **/app/Http/Controllers**, funcionam como intermediários entre os Models e as Views. Eles processam as requisições dos usuários, utilizam os Models para acessar ou modificar os dados necessários e, em seguida, passam esses dados para as Views para renderização. Os Controllers definem a lógica de controle de fluxo da aplicação, decidindo qual View será exibida e com quais dados.

Fluxo MVC no Laravel:
Requisição do Usuário: Inicia com uma requisição do usuário, capturada pelo Controller.
Processamento pelo Controller: O Controller analisa a requisição e, se necessário, interage com um Model para recuperar ou modificar dados.
Interação com o Model: O Model acessa o banco de dados e retorna os dados solicitados ao Controller.
Envio dos Dados para a View: O Controller seleciona a View apropriada e fornece os dados necessários para sua renderização.
Renderização e Apresentação da View: A View gera a interface do usuário, que é apresentada ao usuário.
Essa estrutura e fluxo baseados no MVC permitem uma clara separação de responsabilidades dentro da aplicação Laravel, facilitando o desenvolvimento, a manutenção e permitindo que equipes trabalhem de forma mais eficiente em diferentes aspectos da aplicação.

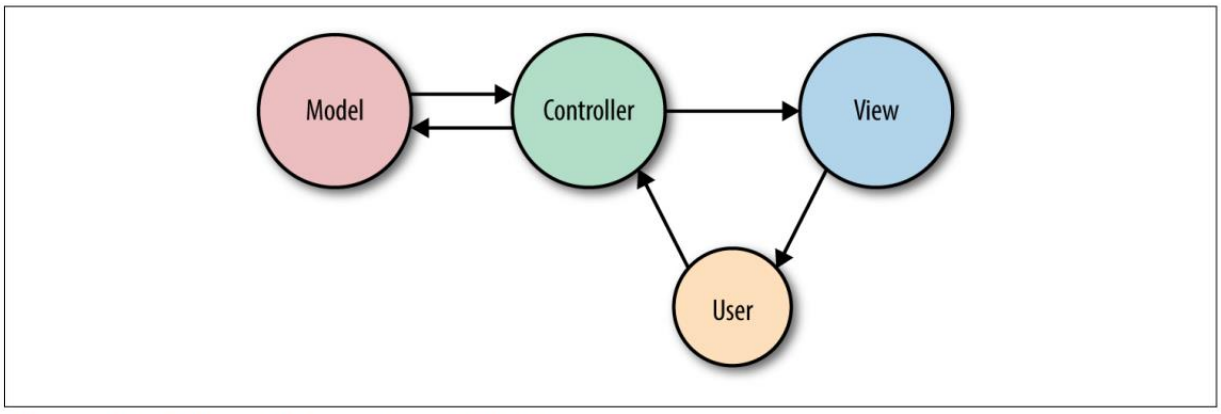

Figure 3-1. A basic illustration of MVC

- **EXEMPLO PRÁTICO**

Passo 1: Instalação do Laravel

Pré-requisitos:
- PHP >= 7.3
- Composer (gerenciador de dependências do PHP)
- MySQL (ou outro banco de dados)

Instalando o Composer
Se você ainda não tem o Composer instalado, pode baixá-lo e instalá-lo seguindo as instruções no site oficial: [Get Composer](https://getcomposer.org/).

Criando um novo projeto Laravel
Abra o terminal e execute o seguinte comando para criar um novo projeto Laravel:

```bash
composer create-project --prefer-dist laravel/laravel blog
```

Passo 2: Estrutura do Projeto Laravel

Navegue até a pasta do projeto:

```bash
cd blog

Abra o arquivo php.ini (geralmente localizado em C:\xampp\php\php.ini).

Habilite as seguintes extensões removendo o ponto e vírgula (;) no início das linhas correspondentes:

ini
Copiar código
extension=fileinfo
extension=openssl
extension=pdo_mysql
extension=mbstring

composer install
```

Estrutura de diretórios principais:

- `app/`: Contém o código da aplicação (models, controllers, middlewares).
- `config/`: Configurações da aplicação.
- `database/`: Migrações e seeds do banco de dados.
- `resources/`: Views, arquivos de idioma e assets não compilados.
- `routes/`: Arquivos de rotas.
- `public/`: Arquivos públicos (index.php, assets compilados).
- `storage/`: Logs, cache e outros arquivos gerados pela aplicação.

Passo 3: Configuração do Banco de Dados

Abra o arquivo `.env` e configure a conexão com o banco de dados:

```env
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=nome_do_banco_de_dados
DB_USERNAME=seu_usuario
DB_PASSWORD=sua_senha
```

Passo 4: Criando uma aplicação CRUD

Vamos criar uma aplicação CRUD (Create, Read, Update, Delete) para gerenciar posts de blog.

Criando a Migração

No terminal, execute o comando para criar uma migração:

```bash
php artisan make:migration create_posts_table --create=posts
```

```

Abra o arquivo da migração em `database/migrations/xxxx_xx_xx_xxxxxx_create_posts_table.php` e defina a estrutura da tabela `posts`:

```php
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('content');
        $table->timestamps();
    });
}
```

Execute a migração para criar a tabela no banco de dados:

'*Laravel e Migrations*
Laravel não cria automaticamente as tabelas quando você cria um modelo. Em vez disso, você cria arquivos de migração que definem as tabelas e suas colunas. Você então executa esses arquivos de migração para criar as tabelas no banco de dados.
Django, por outro lado, gera migrações automaticamente quando você cria ou modifica modelos, e então você pode aplicar essas migrações para atualizar o banco de dados.'

```bash
php artisan migrate
```

Criando o Model

Crie o model para o Post:

```bash
php artisan make:model Post
```

Criando o Controller

Crie um controller para gerenciar os posts:

```bash
php artisan make:controller PostController --resource
```

Passo 5: Definindo Rotas

Abra o arquivo `routes/web.php` e adicione as rotas para os posts:

```php
use App\Http\Controllers\PostController;

Route::resource('posts', PostController::class);
```

Passo 6: Implementando o Controller

Abra o arquivo `app/Http/Controllers/PostController.php` e implemente os métodos do CRUD:

```php
namespace App\Http\Controllers;

use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }

    public function create()
    {
        return view('posts.create');
```
```

```
    }

    public function store(Request $request)
    {
        $post = new Post();
        $post->title = $request->title;
        $post->content = $request->content;
        $post->save();

        return redirect()->route('posts.index');
    }

    public function show($id)
    {
        $post = Post::findOrFail($id);
        return view('posts.show', compact('post'));
    }

    public function edit($id)
    {
        $post = Post::findOrFail($id);
        return view('posts.edit', compact('post'));
    }

    public function update(Request $request, $id)
    {
        $post = Post::findOrFail($id);
        $post->title = $request->title;
        $post->content = $request->content;
        $post->save();

        return redirect()->route('posts.index');
    }

    public function destroy($id)
    {
        $post = Post::findOrFail($id);
        $post->delete();

        return redirect()->route('posts.index');
    }
}
```

** Eloquent ORM (Object-Relational Mapping) é uma ferramenta de mapeamento objeto-relacional incluída no Laravel, um dos frameworks de aplicação web mais populares do PHP. O Eloquent fornece uma abstração elegante e simples para trabalhar com bancos de dados, permitindo que os desenvolvedores interajam com os dados do banco de dados usando modelos PHP. Isso significa que, em vez de escrever código SQL bruto, os desenvolvedores podem utilizar métodos PHP para realizar operações CRUD (Create, Read, Update, Delete) nos registros do banco de dados de forma mais intuitiva e segura.*

Controllers - Laravel 11.x - The PHP Framework For Web Artisans

Passo 7: Criando as Views

Crie a pasta `resources/views/posts` e adicione os arquivos de view para listar, criar, editar e mostrar posts.

resources/views/posts/index.blade.php

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Posts</title>
</head>
<body>
    <h1>Posts</h1>
    <a href="{{ route('posts.create') }}">Create Post</a>
    <ul>
        @foreach ($posts as $post)
            <li>
                <a href="{{ route('posts.show', $post->id) }}">{{ $post->title }}</a>
                <a href="{{ route('posts.edit', $post->id) }}">Edit</a>
                <form action="{{ route('posts.destroy', $post->id) }}" method="POST" style="display: inline;">
                    @csrf
                    @method('DELETE')
```

```
                <button type="submit">Delete</button>
            </form>
        </li>
    @endforeach
  </ul>
</body>
</html>
```

resources/views/posts/create.blade.php

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Create Post</title>
</head>
<body>
  <h1>Create Post</h1>
  <form action="{{ route('posts.store') }}" method="POST">
    @csrf
    <div>
      <label for="title">Title</label>
      <input type="text" id="title" name="title">
    </div>
    <div>
      <label for="content">Content</label>
      <textarea id="content" name="content"></textarea>
    </div>
    <button type="submit">Create</button>
  </form>
</body>
</html>
```

resources/views/posts/edit.blade.php

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Edit Post</title>
</head>
<body>
  <h1>Edit Post</h1>
  <form action="{{ route('posts.update', $post->id) }}" method="POST">
    @csrf
    @method('PUT')
    <div>
      <label for="title">Title</label>
      <input type="text" id="title" name="title" value="{{ $post->title }}">
    </div>
    <div>
      <label for="content">Content</label>
      <textarea id="content" name="content">{{ $post->content }}</textarea>
    </div>
    <button type="submit">Update</button>
  </form>
</body>
</html>
```

resources/views/posts/show.blade.php

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{{ $post->title }}</title>
</head>
<body>
  <h1>{{ $post->title }}</h1>
```

```
    <p>{{ $post->content }}</p>
    <a href="{{ route('posts.index') }}">Back to Posts</a>
</body>
</html>
```

** Blade é o motor de templates do Laravel. Ele fornece uma maneira elegante e poderosa de trabalhar com HTML e dados PHP juntos, permitindo que os desenvolvedores criem layouts de aplicativos dinâmicos e reutilizáveis de forma eficiente.*

Blade Templates - Laravel 11.x - The PHP Framework For Web Artisans

Execute o comando para gerar uma chave php artisan key:generate

EXECUÇÃO

Execute no terminal dentro da pasta do app criado: php artisan serve

Acesse o app -> http://127.0.0.1:8000/posts

posts foi a rota definida no web.php

Playlist

https://www.youtube.com/watch?v=AEC7o2T0k3Y&list=PLnDvRpP8BnewYKI1n2chQrrR4EYiJKbUG

- Fazendo Deploy na Hostinger
    - Execute o seguinte comando no terminal do vs code:

```
npm run build
```

    - Zip os arquivos do projeto, exceto a pasta node_modules
    - Suba os arquivos e extraia na pasta desejada. Para fazer upload mais rápido, faça antes do nível da pasta public_html e depois mova o conteúdo para a pasta dentro de public_html.
    - Configure as permissões das pastas bootstrap e storage

    - Crie uma base de dados e importe o sql do banco de dados. Você pode exportar o sql no phpmyadmin do xampp local
    - Edite o arquivo .env :

```
APP_NAME=Laravel
APP_ENV=production
APP_KEY=base64:hSQL+2Ldxa8ve8ti23eVxtCzJxPh+2qh2a43jBMbmhg=
APP_DEBUG=false
APP_TIMEZONE=UTC
APP_URL=https://samsites.com.br/arqu

APP_LOCALE=en
APP_FALLBACK_LOCALE=en
APP_FAKER_LOCALE=en_US

APP_MAINTENANCE_DRIVER=file
APP_MAINTENANCE_STORE=database

BCRYPT_ROUNDS=12

LOG_CHANNEL=stack
LOG_STACK=single
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=u892138677_blog_laravel
```

```
DB_USERNAME=u892138677_blog_laravel
DB_PASSWORD=Sa747400

....
```

Só acessar https://samsites.com.br/arqu/public

Normalmente, você não deve incluir a pasta vendor no controle de versão (como Git) ou fazer upload dela diretamente para o servidor de produção. Em vez disso, você deve gerar a pasta vendor no servidor de produção usando o Composer. Aqui está o motivo e o processo:

Motivo para não subir a pasta vendor:
1. *Tamanho*: A pasta vendor pode ser bastante grande e inclui muitas dependências de terceiros.
2. *Consistência*: Executar composer install no servidor de produção garante que todas as dependências sejam instaladas na versão correta, conforme especificado no composer.json e composer.lock.

Processo correto para lidar com a pasta vendor:

1. *Suba o código-fonte, exceto a pasta vendor*:
   - Utilize .gitignore para ignorar a pasta vendor se estiver usando Git.

2. *Conecte-se ao servidor de produção*.

3. *Instale as dependências usando o Composer*:
   - No diretório do seu projeto Laravel no servidor de produção, execute:
     sh
     composer install --no-dev --optimize-autoloader

   - A opção --no-dev garante que apenas as dependências de produção sejam instaladas.
   - A opção --optimize-autoloader melhora o desempenho do autoloader do Composer.

Passos adicionais:

1. *Configuração do ambiente*:
   - Certifique-se de que o arquivo .env esteja configurado corretamente no servidor de produção.

2. *Permissões*:
   - Defina as permissões corretas para as pastas storage e bootstrap/cache:
     sh
     chown -R www-data:www-data storage bootstrap/cache
     chmod -R 775 storage bootstrap/cache

3. *Otimizações*:
   - Execute os seguintes comandos para otimizar a aplicação:
     sh
     php artisan config:cache
     php artisan route:cache
     php artisan view:cache

4. *Migrações e seeders*:
   - Execute as migrações de banco de dados:
     sh
     php artisan migrate --force

   - Opcionalmente, execute os seeders:
     sh
     php artisan db:seed --force

Resumo:

- *Não* suba a pasta vendor diretamente.
- *Sim* suba o restante do código-fonte.
- *Instale* as dependências no servidor de produção usando composer install.

Seguindo essas práticas, você manterá um ambiente de produção limpo, eficiente e alinhado com as melhores práticas de deploy para aplicações Laravel.

Na Hostinger, você pode executar o Composer diretamente no servidor através do acesso SSH. Aqui está um passo a passo para você fazer isso:

Passo a Passo para Executar o Composer na Hostinger

1. *Acesse o Painel de Controle*:

- Faça login no seu painel de controle da Hostinger.

2. *Habilitar SSH*:
   - Vá para a seção de *Acesso SSH* e habilite-o se ainda não estiver habilitado.

3. *Obtenha as Credenciais SSH*:
   - Na seção de *Acesso SSH*, você encontrará as credenciais necessárias: Host, Usuário, Senha e Porta.

4. *Conecte-se via SSH*:
   - Use um cliente SSH (como o Terminal no macOS/Linux ou o PuTTY no Windows) para se conectar ao servidor.
   - No terminal, execute:
     sh
     ssh user@host

     Substitua user e host pelas credenciais fornecidas pela Hostinger. Você será solicitado a inserir a senha.

5. *Navegue até o Diretório do Seu Projeto*:
   - Após se conectar ao servidor via SSH, navegue até o diretório onde está o seu projeto Laravel. Isso pode ser algo como:
     sh
     cd /home/username/public_html/nome_do_projeto

6. *Executar o Composer*:
   - Certifique-se de que o Composer está instalado no servidor. Para verificar, execute:
     sh
     composer --version

   - Se o Composer não estiver instalado, siga as instruções de instalação fornecidas pela Hostinger ou no site oficial do Composer.
   - Uma vez no diretório do seu projeto, execute:
     sh
     composer install --no-dev --optimize-autoloader

   - Isso instalará todas as dependências necessárias de acordo com o seu composer.json.

Etapas Adicionais

1. *Configuração do .env*:
   - Certifique-se de que o arquivo .env está configurado corretamente. Você pode copiar o arquivo .env.example e ajustar as configurações:
     sh
     cp .env.example .env
     nano .env

2. *Chave da Aplicação*:
   - Gere a chave da aplicação:
     sh
     php artisan key:generate

3. *Permissões*:
   - Ajuste as permissões das pastas storage e bootstrap/cache:
     sh
     chown -R username:groupname storage bootstrap/cache
     chmod -R 775 storage bootstrap/cache

   - Substitua username e groupname pelos seus detalhes específicos.

4. *Migrações e Seeders*:
   - Execute as migrações:
     sh
     php artisan migrate --force

   - Execute os seeders, se necessário:
     sh
     php artisan db:seed --force

5. *Otimizações*:
   - Otimize as configurações e rotas:
     sh
     php artisan config:cache
     php artisan route:cache
     php artisan view:cache

Resumo:

- Conecte-se ao seu servidor via SSH usando as credenciais fornecidas pela Hostinger.
- Navegue até o diretório do seu projeto Laravel.
- Execute composer install para instalar as dependências.
- Configure o arquivo .env e gere a chave da aplicação.
- Ajuste as permissões necessárias.
- Execute migrações, seeders e otimizações.

Seguindo esses passos, seu projeto Laravel deve estar pronto para rodar na Hostinger.

Na pasta public crie o arquivo .htaccess com o seguinte conteúdo:

```
<IfModule mod_rewrite.c>
    <IfModule mod_negotiation.c>
        Options -MultiViews -Indexes
    </IfModule>

    RewriteEngine On

    # Handle Authorization Header
    RewriteCond %{HTTP:Authorization} .
    RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

    # Redirect Trailing Slashes If Not A Folder...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_URI} (.+)/$
    RewriteRule ^ %1 [L,R=301]

    # Send Requests To Front Controller...
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^ index.php [L]
</IfModule>
```

Na pasta raiz do projeto crie o arquivo .htaccess com o seguinte conteúdo:

```
Options +SymLinksIfOwnerMatch
RewriteEngine On

# Allow Installatron requests
RewriteCond %{REQUEST_FILENAME} deleteme\.\w+\.php
RewriteRule (.*) - [L]

RewriteCond %{REQUEST_URI} !^/public/
RewriteRule ^(.*)$ public/$1 [L]
```

Na pasta raiz crie também o arquivo index.php com o seguinte conteúdo:

```
<?php

use Illuminate\Http\Request;

define('LARAVEL_START', microtime(true));

// Determine if the application is in maintenance mode...
if (file_exists($maintenance = __DIR__.'/storage/framework/maintenance.php')) {
    require $maintenance;
}

// Register the Composer autoloader...
require __DIR__.'/vendor/autoload.php';

// Bootstrap Laravel and handle the request...
(require_once __DIR__.'/bootstrap/app.php')
    ->handleRequest(Request::capture());
```

Observe o nome das pastas nas views, elas são case sensitive. Coloque tudo minúsculo;

*

O `Route::resource` do Laravel cria um conjunto completo de rotas RESTful para o seu controlador, incluindo as rotas para `index`, `create`, `store`, `show`, `edit`, `update`, e `destroy`. Cada rota é associada a um método específico no seu controlador.

No caso do método `update`, a rota gerada pelo `Route::resource` seria algo como:

```
bash
Copiar código
PUT /pessoas/{id}
```

Quando você faz um fetch com o método `PUT` para a URL `{{ url("pessoas") }}/' + id`, o Laravel mapeia automaticamente essa solicitação para o método `update` no `PessoaController`.

Como o Route::resource funciona:

Aqui está a lista das rotas que são registradas pelo `Route::resource`:

```
bash
Copiar código
Verb      | URI                     | Action       | Route Name
----------|-------------------------|--------------|------------------
GET       | /pessoas                | index        | pessoas.index
GET       | /pessoas/create         | create       | pessoas.create
POST      | /pessoas                | store        | pessoas.store
GET       | /pessoas/{id}           | show         | pessoas.show
GET       | /pessoas/{id}/edit      | edit         | pessoas.edit
PUT/PATCH | /pessoas/{id}           | update       | pessoas.update
DELETE    | /pessoas/{id}           | destroy      | pessoas.destroy
```