Laravel 8 Básico

Primeiros passos com Laravel 8

Índice

Advertência

Esta apostila é fruto de pesquisa por vários sites e autores e com alguma contribuição pessoal. Geralmente o devido crédito é concedido, exceto quando eu não mais encontro o site de origem. Isso indica que esta apostila foi criada, na prática, "a várias mãos" e não somente por mim. Caso identifique algo que seja de sua autoria e não tenha o devido crédito, poderá entrar em contato comigo para dar o crédito ou para remover: ribafs @ gmail.com (remova os dois espaços).

É importante ressaltar que esta apostila é distribuída gratuitamente no repositório:

https://github.com/ribafs/apostilas

Sugestões

Sugestões serão bem vindas, assim como aviso de erros no português ou no Laravel. Crie um issue no repositório ou me envie um e-mail ribafs @ gmail.com.

Autor

Ribamar FS

ribafs @ gmail.com

https://ribamar.net.br

https://github.com/ribafs

Fortaleza, 23 de dezembro de 2021

Recomendações

O conhecimento teórico é importante para que entendamos como as coisas funcionam e sua origem, mas para consolidar um conhecimento e nos dar segurança precisamos de prática e muita prática.

Não vale muito a penas ser apenas mais um programador. É importante e útil aprender muito, muito mesmo, ao ponto de sentir forte segurança do que sabe e começar a transmitir isso para os demais.

Tenha seu ambiente de programação em seu desktop para testar com praticidade todo o código desejado.

Caso esteja iniciando com Laravel recomendo que leia por inteiro e em sequência. Mas se já entende algo dê uma olhada no índice e vá aos assuntos que talvez ainda não domine.

Não esqueça de ver e testar também o conteúdo da pasta Anexos do repositório.

Caso tenha alguma dúvida, me parece que a forma mais prática de receber resposta é através de grupos. Depois temos o Google.

Dica: quando tiver uma dúvida não corra para pedir ajuda. Antes analise o problema, empenhe-se em entender o contexto e procure você mesmo resolver. Assim você está passando a responsabilidade para si mesmo, para seu cérebro, que passará a ser mais confiante e poderá te ajudar ainda mais. É muito importante que você confie em si mesmo, de que é capaz de solucionar os problemas. Isso vai te ajudar. Somente depois de tentar bastante e não conseguir, então procure ajuda.

Veja que este material não tem vídeos nem mesmo imagens. Isso em si nos nossos dias é algo que atrai pouca gente, pois vídeos e fotos são mais confortáveis de ler e acompanhar. Ler um texto como este requer mais motivação e empenho. Lembrando que para ser um bom programador precisamos ser daqueles capaz de se empenhar e suportar a leitura e escrita também.

Autodidata

Não tive a pretensão de que esta apostila fosse completa, contendo tudo sobre Laravel, que seria praticamente impossível. Aquele programador que quando não sabe algo seja capaz de pesquisar, estudar, testar e resolver praticamente sozinho. Este é um profissional importante para as empresas e procurado.

1 – Introdução

O laravel é atualmente o framework mais popular entre os de PHP

Laravel é um Framework PHP utilizado para o desenvolvimento web, que utiliza a arquitetura MVC e tem como principal característica ajudar a desenvolver aplicações seguras e performáticas de forma rápida, com código limpo e simples, já que ele incentiva o uso de boas práticas de programação e utiliza o padrão PSR-2 como guia para estilo de escrita do código.

Para a criação de interface gráfica, o Laravel utiliza uma Engine de template chamada Blade, que traz uma gama de ferramentas que ajudam a criar interfaces bonitas e funcionais de forma rápida e evitar a duplicação de código.

Listas impressionantes

Vale a penas gastar algum tempo lendo as listas de links deste repositório para se inspirar, estudar e absorver conhecimento.

https://github.com/ribafs/all-awesomes

1.1 – Alerta

Esta apostila pretende ser o máximo prática, cujo objetivo é o de ensinar a criar aplicativos básicos e com pouca teoria. Procurarei transmitir de forma simples com a intenção de que quando saiba básico possa ir de encontro a conhecimentos mais complexos.

Esta apostila é bem pequena e muito básica e tem bem poucos dos recursos do Laravel 8, que são muitos. Aqui quero despertar o interesse pelo framework que dá os primeiros passos.

2 – Instalação e Configurações

2.1 – Requisitos para a instalação

- PHP >= 7.3
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP Extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Composer

Um servidor web, Apache, Nginx, etc.
SGBD suportados:

- MySQL 5.7+ (Version Policy)
- PostgreSQL 9.6+ (Version Policy)
- SQLite 3.8.8+
- SQL Server 2017+ (Version Policy)

https://laravel.com/docs/8.x

2.2 - Instalação

O Laravel tem basicamente duas formas de instalação:

- Através do composer
composer create-project laravel/laravel products

- Definindo qual versão do Laravel usar:
composer create-project laravel/laravel="VersDaSuaEscolha" nomeApp

- Através do installer

Instalar
composer global require "laravel/installer=~1.1"

No Linux Mint ou Ubuntu executar:
export PATH="~/.config/composer/vendor/bin:$PATH"

Criando aplicativo
laravel new nomeApp

Vamos agora instalar nosso aplicativo

cd /var/www/html ou c:\xampp\htdocs
composer create-project laravel/laravel products
cd products

2.3 – Configurações

O arquivo principal e mais simples de configuração é o

.env

Que se encontra no raiz do app

Se estiver usando um linux ou similar, num gerenciador de arquivos gráficos, tecle Ctrl+H para exibir o .env, visto que é um arquivo oculto.

Nano .env

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=products
DB_USERNAME=root
DB_PASSWORD=root

A tabela products tem apenas os campos id, name e detail.

Na pasta config existem muitos arquivos de configuração.

2.4 – Estrutura de diretórios do Laravel 8

app/ - Diretório dos arquivos da sua aplicação

bootstrap/ - Arquivos de inicialização, são chamados em cada requisição ao servidor

config/ - Arquivos de configuração como banco de dados, serviços, etc.

database/ - Aqui existem basicamente duas pastas, "migrations" que guardam os arquivos que fazem as alterações na estrutura do banco, como nome das tabelas, colunas, etc, e "seeds" que seriam arquivos e classes que geram registros no banco de dados, seja para testes ou para possuir informações iniciais de utilização.

public/ - São os arquivos públicos do sistema, normalmente é aqui que são colocados os arquivos de imagens, CSS e JS.

resources/ - Arquivos de recursos como bibliotecas JS e CSS, arquivos com arrays de traduções de mensagens, normalmente usados numa aplicação multilinguagem e as próprias views do sistema.

routes/ - Rotas

storage/ - Arquivos de cache, sessões (quando usado armazenamento em arquivo), views compiladas e logs.

tests/ - Arquivos de testes do sistema.

vendor/ - pasta criada pelo composer para controle e versionamento de bibliotecas.

3 - Bancos de dados

Para trabalhar com bancos de dados o Laravel traz nativamente vários recursos. Irei analisar alguns.

3.1 – Migrations

Migrations são arquivos com classes que descrevem estrutura de tabela.

Ele cria a tabela no banco de dados e a gerencia, adiciionando campos, alterando, por isso o banco já deve estar previamente configurado no .env antes de executar o migrate

As migrations ficam no diretório /database/migrations

Uma migration pode conter várias tabelas, mas não é uma boa prática

Criar model Product, migrations e também controller e com --resource

php artisan make:model Product -mcr

Editar a migration criada e mudar o método up() assim:

```
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->string('name', 50);
        $table->text('detail');
        $table->timestamps();
    });
}
```

Editar o model
app/Models/Product.php

E deixar assim:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'detail'];
}
```

Quanto ao controller ajustaremos mais a frente.

3.2 – Factories

Mas o que é Factory?
O Factory é usado no momento de desenvolvimento/teste de software. Sabe quando você está programando sua aplicação e precisa ver como ele se comporta com múltiplos registros? Geralmente nesse momento nós inserimos os registros “na mão”. Trabalha com o seeder.

Um factory depende de um model.

Ok, desse modo é chato, mas a maior parte das vezes resolve o problema, só vamos copiar e colar a mesma linha de insert no MySQL, o que complica é quando temos campos de tipo “unique” como e-mails, slugs, nicknames, nesse caso o copiar e colar não vai funcionar, pois as informações não podem se repetir, e pior ainda, quando queremos testar uma paginação de 50, 100, 200 resultados, vai ficar complicado de inserir tudo na mão.

O Factory tem especificamente essa função, ele vai permitir o programador inserir múltiplos registros “aleatórios” dentro do Banco de Dados, se você quiser 50 registros, 100 ou 1000, será só especificar a quantidade.

https://g4br.medium.com/populando-sua-base-de-dados-com-factory-8e46294a891

Criar a factory Product

php artisan make:factory ProductFactory --model=Product

Criar o seeder product

O **Laravel** por padrão inclui um método simples para criarmos e executarmos **seeders** para popular rapidamente nosso banco de desenvolvimento através de classes que chamamos de **seeders**. Um **seeder** contém um **array de dados** que serão inseridos no banco de dados quando chamados.
O uso de **seeders** é muito comum, especialmente em ambientes de testes e homologação, para que você possa ter dados no banco para começar a trabalhar.
https://www.organicadigital.com/seeds/inverse-seeder-no-laravel-o-que-e-e-como-usar/

php artisan make:seeder ProductsTableSeeder

Executar

composer dumpautoload

Editar o factory product e deixar assim:

```
<?php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class ProductFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
```

```
    {
        return [
            'name' => $this->faker->name(),
            'detail' => $this->faker->text()
        ];
    }
}
```

Mudar o método run() em seeders/DatabaseSeeder.php para

```
    public function run()
    {
        \App\Models\Product::factory(50)->create();
    }
```

Executar as migrations com seeders

php artisan migrate:refresh --seed

Referências

https://laravel.com/docs/8.x/migrations

4 – MVC

Vamos supor que nunca criou um sistema back-end antes ou que está simplesmente à procura de novas formas de fazer algumas coisas.

Então o que é MVC?

MVC (Model-view-controller) é um padrão de design comumente utilizado para desenvolver interfaces de utilizador que dividem a lógica do programa relacionado em três elementos interligados. Isto é feito para separar as representações internas da forma como a informação é apresentada ao utilizador e aceite por este.

A visão em si é fácil de compreender, é simplesmente isso, a visão com a qual o utilizador interage.

O modelo é o componente central do padrão. É a estrutura de dados dinâmica da aplicação, independente da interface do utilizador. Gere diretamente os dados, a lógica, e as regras da aplicação.

O controlador aceita comandos do modelo e converte-os para que os dados possam ser utilizados.

Traduzido com a versão gratuita do tradutor - www.DeepL.com/Translator

O Laravel tem comandos que criam os esqueletos do Model e do Controller. A view precisará criar inteiramente manual.

https://dev.to/grahammorby/mvc-and-creating-it-in-laravel-8-2a6b

Pastas do MVC do Laravel 8

- Rotas - /routes/web
- Model - /app/Models (Na versão 8, nas anteriores era na /app)
- Controller - /app/Http/Controllers
- Views - /resources/views

Um router envia a requisição do usuário para o devido action de um controller. Um controller é responsável por mapear as ações do usuário final para a resposta do aplicativo, enquanto as ações de um modelo incluem ativar processos de negócios ou alterar o estado do modelo. As interações do usuário e a resposta do modelo decidem como um controlador responderá selecionando uma view apropriada.

4.1 – Model

Models no Laravel 8

No laravel 8 os models agora tem seu próprio diretório e namespace:

app/Models
App\Models

Models

Cada tabela tem um model correspondente, que é usado para interagir com a tabela. Um model permite gerenciar uma tabela. Cada model tem um controller correspondente. E cada controller tem algumas views ligadas a ele.

Exemplo de paths:

- Tabela - produtos (plural)
- Model - app/Produto.php (singular, também o nome da classe: Produto)
- Controller - app/Http/Controllers/ProdutoController.php (singular com o sufixo Controller)
- Views - resources/views/produtos: index, create, edit e show

Criar uma rota para produtos

Route::resource('produtos', App\Http\Controllers\ProdutoController');

4.2 – Controller

Controllers no Laravel 8

https://laravel.com/docs/8.x/controllers

Os controllers representam o C do MVC e são a camada de mediação entre model e view.

Em vez de definir toda a lógica de tratamento de sua solicitação como Closures em arquivos de rota, você pode desejar organizar esse comportamento usando classes de controller. Os controllers podem agrupar a lógica de tratamento de solicitações relacionadas em uma única classe. Os controllers são armazenados:
app/Http/Controllers

Editar o controller criado e deixar assim

app/Http/Controllers/ProductsController.php

```
<?php

namespace App\Http\Controllers;

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

class ProductController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $products = Product::latest()->paginate(5);

        return view('products.index',compact('products'))
            ->with('i', (request()->input('page', 1) - 1) * 5);
    }

    /**
```

```
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        return view('products.create');
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $request->validate([
            'name' => 'required',
            'detail' => 'required',
        ]);

        Product::create($request->all());

        return redirect()->route('products.index')
                        ->with('success','Product created successfully.');
    }

    /**
     * Display the specified resource.
     *
     * @param  \App\Product  $product
     * @return \Illuminate\Http\Response
     */
    public function show(Product $product)
    {
        return view('products.show',compact('product'));
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  \App\Product  $product
     * @return \Illuminate\Http\Response
     */
    public function edit(Product $product)
    {
        return view('products.edit',compact('product'));
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\Product  $product
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, Product $product)
    {
        $request->validate([
            'name' => 'required',
            'detail' => 'required',
        ]);
```

```
        $product->update($request->all());

        return redirect()->route('products.index')
                        ->with('success','Product updated successfully');
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Product  $product
     * @return \Illuminate\Http\Response
     */
    public function destroy(Product $product)
    {
        $product->delete();

        return redirect()->route('products.index')
                        ->with('success','Product deleted successfully');
    }
}
```

Adicionar a rota tipo resources para products em routes/web.php

```
<?php

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

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

Route::resource('products', ProductController::class);
```

Adicionar as seguintes views

Criar a pasta

resources/views/products

E dentro dela os arquivos:

- layout.blade.php
- index.blade.php
- create.blade.php
- edit.blade.php
- show.blade.php

layout.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Laravel 8 CRUD Products</title>
```

```
    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-alpha/css/bootstrap.css" rel="stylesheet">
</head>
<body>

<div class="container">
    @yield('content')
</div>

</body>
</html>
```

index.blade.php

```
@extends('products.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Laravel 8 CRUD Products</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-success" href="{{ route('products.create') }}"> Create New Product</a>
            </div>
        </div>
    </div>

    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif

    <table class="table table-bordered">
        <tr>
            <th>No</th>
            <th>Name</th>
            <th>Details</th>
            <th width="280px">Action</th>
        </tr>
        @foreach ($products as $product)
        <tr>
            <td>{{ ++$i }}</td>
            <td>{{ $product->name }}</td>
            <td>{{ $product->detail }}</td>
            <td>
                <form action="{{ route('products.destroy',$product->id) }}" method="POST">

                    <a class="btn btn-info" href="{{ route('products.show',$product->id) }}">Show</a>

                    <a class="btn btn-primary" href="{{ route('products.edit',$product->id) }}">Edit</a>

                    @csrf
                    @method('DELETE')

                    <button type="submit" class="btn btn-danger">Delete</button>
                </form>
            </td>
        </tr>
        @endforeach
    </table>
```

```
    {!! $products->links() !!}

@endsection
```

create.blade.php

```
@extends('products.layout')

@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Add New Product</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('products.index') }}"> Back</a>
        </div>
    </div>
</div>

@if ($errors->any())
    <div class="alert alert-danger">
        <strong>Whoops!</strong> There were some problems with your input.<br><br>
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

<form action="{{ route('products.store') }}" method="POST">
    @csrf

     <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Name:</strong>
                <input type="text" name="name" class="form-control" placeholder="Name">
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Detail:</strong>
                <textarea class="form-control" style="height:150px" name="detail" placeholder="Detail"></textarea>
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
                <button type="submit" class="btn btn-primary">Submit</button>
        </div>
    </div>

</form>
@endsection
```

edit.blade.php

```
@extends('products.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
```

```
            <div class="pull-left">
                <h2>Edit Product</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('products.index') }}"> Back</a>
            </div>
        </div>
    </div>

    @if ($errors->any())
        <div class="alert alert-danger">
            <strong>Whoops!</strong> There were some problems with your input.<br><br>
            <ul>
                @foreach ($errors->all() as $error)
                    <li>{{ $error }}</li>
                @endforeach
            </ul>
        </div>
    @endif

    <form action="{{ route('products.update',$product->id) }}" method="POST">
        @csrf
        @method('PUT')

         <div class="row">
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Name:</strong>
                    <input type="text" name="name" value="{{ $product->name }}" class="form-control"
placeholder="Name">
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12">
                <div class="form-group">
                    <strong>Detail:</strong>
                    <textarea class="form-control" style="height:150px" name="detail" placeholder="Detail">{{ $product-
>detail }}</textarea>
                </div>
            </div>
            <div class="col-xs-12 col-sm-12 col-md-12 text-center">
              <button type="submit" class="btn btn-primary">Submit</button>
            </div>
        </div>

    </form>
@endsection
```

show.blade.php

```
@extends('products.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2> Show Product</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('products.index') }}"> Back</a>
            </div>
        </div>
```

```
    </div>

    <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Name:</strong>
                {{ $product->name }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Details:</strong>
                {{ $product->detail }}
            </div>
        </div>
    </div>
@endsection
```

Testando

php artisan serve

http://localhost:8000/products

5 – Criar aplicativo clientes

Agora vamos criar um aplicativo semelhante ao products, mas para a tabela clientes, com os campos nome e e-mail, seguindo passos semelhantes aos anteriores.

Instalar

composer create-project laravel/laravel clientes

cd clientes

Criar o banco clientes e configurar no .env

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=clientes
DB_USERNAME=root
DB_PASSWORD=root
```

Criar model Cliente, migrations e também controller e com --resource

php artisan make:model Cliente -mcr

Editar a migration criada e mudar o método up() assim:

```
    public function up()
    {
        Schema::create('clientes', function (Blueprint $table) {
            $table->id();
            $table->string('nome', 50);
            $table->text('email', 100);
            $table->timestamps();
        });
    }
```

Editar o model
app/Models/Cliente.php

E deixar assim:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Cliente extends Model
{
    use HasFactory;

    protected $fillable = ['nome', 'email'];
}
```

Criar a Factory Cliente

php artisan make:factory ClienteFactory --model=Cliente

Criar o seeder Clientes

php artisan make:seeder ClientesTableSeeder

Executar

composer dumpautoload

Editar o factory criado e mudar somente o return assim:

```
    return [
        'nome' => $this->faker->name(),
        'email' => $this->faker->unique()->safeEmail()
    ];
```

Mudar o método run() em database/seeders/DatabaseSeeder.php para

```
    public function run()
    {
        \App\Models\Cliente::factory(50)->create();
    }
```

Executar as migrations com seeders

php artisan migrate:refresh --seed

Referências

https://laravel.com/docs/8.x/migrations

Editar o controller
app/Http/Controllers/ClienteController.php

E deixar assim:

```
<?php

namespace App\Http\Controllers;

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

class ClienteController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
```

```
    $clientes = Cliente::latest()->paginate(5);

    return view('clientes.index',compact('clientes'))
        ->with('i', (request()->input('page', 1) - 1) * 5);
}

/**
 * Show the form for creating a new resource.
 *
 * @return \Illuminate\Http\Response
 */
public function create()
{
    return view('clientes.create');
}

/**
 * Store a newly created resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $request->validate([
        'nome' => 'required',
        'email' => 'required',
    ]);

    Cliente::create($request->all());

    return redirect()->route('clientes.index')
                    ->with('success','Cliente created successfully.');
}

/**
 * Display the specified resource.
 *
 * @param  \App\Models\Cliente  $cliente
 * @return \Illuminate\Http\Response
 */
public function show(Cliente $cliente)
{
    return view('clientes.show',compact('cliente'));
}

/**
 * Show the form for editing the specified resource.
 *
 * @param  \App\Models\Cliente  $cliente
 * @return \Illuminate\Http\Response
 */
public function edit(Cliente $cliente)
{
    return view('clientes.edit',compact('cliente'));
}

/**
 * Update the specified resource in storage.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \App\Models\Cliente  $cliente
 * @return \Illuminate\Http\Response
```

```
     */
    public function update(Request $request, Cliente $cliente)
    {
        $request->validate([
            'nome' => 'required',
            'email' => 'required',
        ]);

        $cliente->update($request->all());

        return redirect()->route('clientes.index')
                        ->with('success','Cliente updated successfully');
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Models\Cliente  $cliente
     * @return \Illuminate\Http\Response
     */
    public function destroy(Cliente $cliente)
    {
        $cliente->delete();

        return redirect()->route('clientes.index')
                        ->with('success','Cliente deleted successfully');
    }
}
```

Criar a pasta para as views

resources/views/clientes

E d entro os arquivos

- layout.blade.php
- index.blade.php
- edit.blade.php
- create.blade.php
- show.blade.php

layout

```
<!DOCTYPE html>
<html>
<head>
    <title>Laravel 8 CRUD Clientes</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.0.0-alpha/css/bootstrap.css" rel="stylesheet">
</head>
<body>

<div class="container">
    @yield('content')
</div>

</body>
</html>
```

index

```
@extends('clientes.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Laravel 8 CRUD Clientes</h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-success" href="{{ route('clientes.create') }}"> Create New Cliente</a>
            </div>
        </div>
    </div>

    @if ($message = Session::get('success'))
        <div class="alert alert-success">
            <p>{{ $message }}</p>
        </div>
    @endif

    <table class="table table-bordered">
        <tr>
            <th>ID</th>
            <th>Nome</th>
            <th>E-mail</th>
            <th width="280px">Ação</th>
        </tr>
        @foreach ($clientes as $cliente)
        <tr>
            <td>{{ ++$i }}</td>
            <td>{{ $cliente->nome }}</td>
            <td>{{ $cliente->email }}</td>
            <td>
                <form action="{{ route('clientes.destroy',$cliente->id) }}" method="POST">

                    <a class="btn btn-info" href="{{ route('clientes.show',$cliente->id) }}">Show</a>

                    <a class="btn btn-primary" href="{{ route('clientes.edit',$cliente->id) }}">Edit</a>

                    @csrf
                    @method('DELETE')

                    <button type="submit" class="btn btn-danger">Delete</button>
                </form>
            </td>
        </tr>
        @endforeach
    </table>

    {!! $clientes->links() !!}

@endsection
```

create

```
@extends('clientes.layout')

@section('content')
<div class="row">
    <div class="col-lg-12 margin-tb">
```

```
        <div class="pull-left">
            <h2>Add New Cliente</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('clientes.index') }}"> Back</a>
        </div>
    </div>
</div>

@if ($errors->any())
    <div class="alert alert-danger">
        <strong>Whoops!</strong> There were some problems with your input.<br><br>
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif

<form action="{{ route('clientes.store') }}" method="POST">
    @csrf

     <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Nome:</strong>
                <input type="text" name="nome" class="form-control" placeholder="Nome">
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>E-mail:</strong>
                <textarea class="form-control" style="height:150px" name="email" placeholder="E-mail"></textarea>
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
                <button type="submit" class="btn btn-primary">Submit</button>
        </div>
    </div>

</form>
@endsection
```

edit

```
@extends('clientes.layout')

@section('content')
    <div class="row">
        <div class="col-lg-12 margin-tb">
            <div class="pull-left">
                <h2>Edit Cliente </h2>
            </div>
            <div class="pull-right">
                <a class="btn btn-primary" href="{{ route('clientes.index') }}"> Back</a>
            </div>
        </div>
    </div>

    @if ($errors->any())
        <div class="alert alert-danger">
```

```
        <strong>Whoops!</strong> There were some problems with your input.<br><br>
        <ul>
           @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
           @endforeach
        </ul>
     </div>
  @endif

  <form action="{{ route('clientes.update',$cliente->id) }}" method="POST">
     @csrf
     @method('PUT')

      <div class="row">
        <div class="col-xs-12 col-sm-12 col-md-12">
           <div class="form-group">
              <strong>Nme:</strong>
              <input type="text" name="nome" value="{{ $cliente->nome }}" class="form-control"
placeholder="Nome">
           </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
           <div class="form-group">
              <strong>E-mail:</strong>
              <textarea class="form-control" style="height:150px" name="email" placeholder="E-mail">{{ $cliente-
>email }}</textarea>
           </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12 text-center">
          <button type="submit" class="btn btn-primary">Submit</button>
        </div>
     </div>

  </form>
@endsection
```

show

```
@extends('clientes.layout')

@section('content')
  <div class="row">
     <div class="col-lg-12 margin-tb">
        <div class="pull-left">
           <h2> Show Cliente</h2>
        </div>
        <div class="pull-right">
           <a class="btn btn-primary" href="{{ route('clientes.index') }}"> Back</a>
        </div>
     </div>
  </div>

  <div class="row">
     <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
           <strong>Nome:</strong>
           {{ $cliente->nome }}
        </div>
     </div>
     <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
           <strong>E-mail:</strong>
```

```
            {{ $cliente->email }}
          </div>
        </div>
    </div>
@endsection
```

Adicionar uma rota do tipo resources para clientes

routes/web.php

use App\Http\Controllers\ClienteController;

Route::resource('clientes', ClienteController::class);

Testando

php artisan serve

http://localhost:8000/clientes

5.1 - Corrigindo o uso do Bootstrap no Laravel 8

composer require laravel/ui
php artisan ui bootstrap –auth
npm install && npm run dev

Paginação

Para continuar usando bootstrap adicione para o método boot do AppServiceProvider

app/Providers/AppServiceProvider.php

use Illuminate\Pagination\Paginator;

Adicionar ao método boot

Paginator::useBootstrap();

Para que fique assim:

```
<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Pagination\Paginator;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
```

```
     * @return void
     */
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Paginator::useBootstrap();
    }
}
```

https://laravel.com/docs/8.x/pagination#using-bootstrap

5.2 – Alguns Pacotes Úteis

5.2.1 – crud-generator

https://github.com/appzcoder/crud-generator

Este pacote facilita a criação de CRUDs, criando para nós model, migration, controller, views e rotas.

Instalação

```
composer require appzcoder/crud-generator --dev
```

Exemplo de uso

php artisan crud:generate Posts --fields='title#string; content#text; category#select#options={"technology": "Technology", "tips": "Tips", "health": "Health"}' --view-path=admin --controller-namespace=App\\Http\\Controllers\\Admin --route-group=admin --form-helper=html

Veja:

Controller created successfully.
Model created successfully.
Migration created successfully.
View created successfully.
Crud/Resource route added to /backup/www/laravel/clientes/routes/web.php

Rota
adim/posts

Executar
php artisan migrate

php artisan serve

http://localhost:8000/admin/posts

Problema – Acusou a falta do admin.sidebar e o substitui pelo layout.blade.php na pasta admin e mudei as referências nos arquivos.
Entrei no repositório e criei um Issue avisando ao autor e citando minha solução.

5.2.2 – laravel-acl

Implementando ACL em aplicativos laravel 8, para controlar o acesso para as diversas seções por alguns usuários.

https://github.com/ribafs/laravel-acl

Com detalhes sobre a instalação e uso.

6 – Convenções

O Laravel tem uma grande quantidade de convenções e quando as seguimos temos a nossa vida bem facilidade. Caso não as sigamos precisaremos trabalhar/codificar mais.

Vejamos algumas das mais usadas aqui:

- Nomes em inglês
Tabelas, campos, classes, arquivos devem vir em inglês. Podemos traduzir a interface do usuário, as strings, mas tabelas, campos, classes, etc, devemos deixar em inglês

- Nomes de tabelas
Devem ser com tudo em minúsculas e no plural

- Nomes de arquivos e classes
Model – CamelCase e no singular. Ex.: Customer.php, MyCustomer.php

Controller – CamelCase, no singular e com sufixo Controller. Ex.: CustomerController.php

View – pasta no plural e com tudo em minúsculas. Ex.: customers
Todas as views em arquivos em minúsculas assim: index.blade.php

Mais detalhes aqui
https://webdevetc.com/blog/laravel-naming-conventions/

7 – Referências

Como o laravel é um framework muito popular facilita encontrar conteúdo sobre ele na internet.

Estarei mostrando abaixo as duas melhores referências que conheço sobre Laravel 8:

https://laravel.com/docs/8.x

https://www.itsolutionstuff.com/tag/laravel-8.html

Download desta apostila e outras:

- PHP
- MySQL
- PDO
- PHPOO
- MVC

https://github.com/ribafs/apostilas

Juntamente com o código dos dois aplicativos criados, products e clientes na pasta Anexos.