

Laravel

- Preparando - se para o desenvolvimento

- Pré requisitos
 - PHP instalado
 - Composer instalado

- Iniciando um projeto

- Na pasta onde ficará o projeto : *Name do projeto*

Composer create-project laravel/laravel *Name App*

- Artisan Console

- Interface de linha de comando da laravel que automatiza várias processos na laravel, será muito utilizado durante o desenvolvimento

* *php artisan serve: abre um servidor localhost*

* Rutas (introdução)

- O que são?

• rutas na laravel são meio que o terminal central de diretorios da aplicação, lugar que define onde e o que acontece quando cada rota é chamada

↳ ruta (url)

↳ permite funções / ex: function() {

ex: Route::get('/contato', [\APP\HTTP\Controllers\Contatos::class, 'função']); echo 'AAA';

↓

↳ método

↳ Classe

↓

}

Classe Router

funções a chamar

* Controllers (introdução)

- Criando uma controller:

→ tipo de classe

`php artisan make:controller NomeController`

Nome da classe

↓
diz que é um
comando php

↳ método que
cria classes

- a partir daqui, a criação da regra de negócios é escrita normalmente na controller

* Views (introdução)

- devem obrigatoriamente ter a extensão:

`view.blade.php`

↳ framework de front-end
do laravel

- a partir daqui, a criação da view é escrita normalmente no arquivo.
- mas pra frente, eu mostrarei como integrar o laravel com o angular

* Rotas (intermediário)

- Envando parâmetros

→ declaração de par.

Ex.: Route::get('/contato/{nome}', function(\$nome) {

↳ parâmetro semelhante
(sempre em vermelho)

2º

Ex.: Route::get('/contato/{nome}/{msg}', function(\$nome, \$msg) {

3º

- Parâmetros opcionais e valores padrões

Ex.: Route::get('/contato/{nome}/{msg?}', function(\$nome, \$msg = 'default') {

↳ diz que é
opcional
define um
valor padrão

* Parâmetros opcionais

Devem ser informações da
direita pra esquerda

- Validando parâmetros com regex

→ parâmetro

Route::get('...') → where("id", "[0-9]+"); → regex.

↳ método da rota

- Agrupando rotas

→ prefixo do grupo

Route::prefix('App') → group(function() {

Rotas aqui

↳ método que agrupa

}); → rotas do grupo

18/9/21

D S T Q Q S S

- Nomeando retas

Rute:: get ('...') → name ('reta.tal');

↳ método que define nome

- Chamando reta pelo nome

- Redirecionamento de retas

- bruto

↳ reta de destino

Rute:: redirect ('retta2', /retas/);

↳ reta de origem

- Controlador

function Nome() {

↳ nome da reta

return redirect () → rute ('');

↳ método de

redirect

- Reta de fallback

↳ método de fallback

Rute:: fallback (function () {

meclida a ser tomada

});

D / S / T / Q / Q / S / S

* Controllers (intermediários)

- Parâmetros da regra para controlador

Rute: get ('/contato/p3/p23', 'ctrl @ func');

↓ ↓ ↗ função que se recebe
parâmetros a passar

public function nome (\$p1, \$p2) {

 echo \$p1 + \$p2;

}

- Parâmetros da controlador pra view.

- Ma Ctrl (array associativo)

public function teste (\$p1, \$p2) {

 return view ('nemeView', ['neme' => \$p1, 'p2' => \$p2]);

}

↗ valores da
var
↳ nome que a var tira na view

- Ma View (para todos)

<h1> {{ \$neme }} </h1>

- Ma Ctrl (compact)

public function nome (\$p1, \$p2) {

 return view ('neme', compact ('p1', 'p2'));

}

↳ define o nome e valo-
res da variável de
uma só vez.

20/9/21

D S T Q Q S S

* View (intermediário) (Laravel / App / Resources / Views)

- Blade

Metodo de renderização de views

- Abre bloco de código php

@php

// código aqui.

@endphp

- if e else

@if(\$var)

<h1>AAA</h1>

@else

* pode-se utilizar

<h2>BBB</h2> else, if, else, if, tads.

@endif

- @unless

@if(! \$var) = @unless(\$var)

↳ precisa ser fechado

- @isset

@isset(\$var)

// \$var

@endisset

- @empty

@empty(\$var)

// \$var

@endempty

- @ switch / case

@ switch (\$tal)

@ case (tal)

// tal

@ break

@ default

// tal.

@ endswitch

- @ valor default (ternário)

Ex: {{ \$preco[0] ?? 'Valor não preenchido' }}

↳ se value, então.

- @ for

@ for (\$i=0; \$i < 10; \$i++)

{}{ \$i }

@ endfor

- @ while

@ while (verd)

// tal

@ endwhile

- @ foreach

@ foreach (\$fornecedores as \$clave => \$valor)

// tal

@ endforeach

23/9/21

D S T Q Q S S

- @foreach

@foreach (\$ fornecedores as \$key => \$value)
 //tal coisa.

@empty

Não existem fornecedores

@endforeach

- Escopos da tag de impressão

@{{ \$fornecedores }}

↳ escopo

- \$loop

Variável que só fica disponível dentro de foreach e foreaches, e possui ~~outros~~ atributos como:

\$loop → iteration

\$loop → first

\$loop → last

\$loop → count

Para ver mais sobre essa var, basta dar um dd(\$loop)

- Anets Helper

facilita a inclusão de anets no html

Substituir:

Por: ↗ aneta por padrão o diretório public

- Templates com @extends, @section e @yield.

Método utilizado para encutar e reutilizar melhor os códigos das views, usando esses três @

→ tem por objetivo o diretório views.

@extends ('views.home')

↳ arquivo que servirá de template, tendo cenas como as tags html, head, menu, footer, cenas que não mudam

nome da section ↗

@section ('name')

// códigos como cards, formulários, tables etc.

* tem como parâmetros através de section, ex:

• @endsection

@section ('title', 'name')

↳ nome ↳ value

No arquivo template:

↳ section

@yield ('name')

↳ inclui aquela cena de

cada no meio do meu template

- @include

@include ('- topo')

24 / 9 / 21

D S T Q Q S S

- Envio de dados de um formulário (get)

< form action = {{ route('name') }} method = "get" >

↳ envia os dados do form para a controller dessa rota.

- Envio de dados de um formulário (post)

< form action = {{ route('name') }} method = "post" >

@csrf

↳ nome da rota que pode ser o mesmo que o do get, para manter o verbo post ao enviar os dados

↳ token de auth

↳ nome da rota que pode ser o mesmo que o do get, para manter o verbo post ao enviar os dados

* Models, migrations, seeds, eloquent, orm, factories.

4 / 9 / 21

- Models (Laravel / App)

Classe que se tornam objetos quando instanciadas

php artisan make: model NameModel -m

↳ Cria autom. uma migration ①

- Migrations (Laravel / App / database / Migrations)

São instruções que tornam a construção e manutenção do banco responsabilidade do php, sem necessidade do sql, assim, o banco sempre andará junto do projeto.

Up e Down: Sobem ou desce o banco da aplicação



Cream



Descreve

D S T Q Q S S

4/10/21

- Cuando a tabla de una clase (superior)

No migration:

public function up()
{}
 → name tabela

Schema :: create ('tb-name', function(Blueprint \$table) {

\$table->id();

stable → timestamps();

```
$table->string('name', 50);
```

\$table → integer('type');

↓
tip

↳ neue Coluna

T

* para que isso tudo funcione, é necessário que o mysql esteja instalado e configurado na env

No terminal :

~~# plp artisan migrate~~

orgânicos onde
ficam as variáveis
de ambiente.

21/10/23

D S T Q Q S S

- Validando formulários com Form Request

php artisan make::request StoreUserRequest

↳ Cria um request em App/Http\Requests.

Na Request

```
public function authorize();  
{  
    return true;  
}
```

```
public function rules()  
{  
    return [  
        'name' => ['required', 'string', 'min:3'], ...  
    ];  
}
```

↳ name do input

Na Controller

```
public function store(StoreUserRequest $request)
```

↳ apenas a parametrização da request como parâm. já faz a valid., não restando nada à controller.

php artisan make:model -mca Name
--all

D / S / T / Q / Q / S / S

12/12/21

- Criação de uma migração para alterar tabela existente

php artisan make:migration nome - suggestiv

↳ ex: alter - cental - never -
columns

Na migration:

public function up() {

Schema::table('name', function(Blueprint \$table) {

\$table->string('uf', 2);

\$table->string('email', 80);

});

}

- Métodos up e down

• Up: método responsável por aplicar todas as alterações no db.

• Down: método responsável por reverter tudo o que foi feito no método up.

- Método down da migração acima

public function down() {

Schema::table('name', function(Blueprint \$table) {

\$table->dropColumn(['uf', 'email']);

});

}

php artisan migrate:rollback --step=2
→ quant de migr. a dar
down.

se omitido,
volta apenas 1

13/12/21

D S T Q Q S S

- Criação de colunas mutable e default.

Schema:: create ('produtos', ... {

\$table → string ('nome', 50); ↗ diz que não ser null

\$table → integer ('preco') → nullable ();

\$table → float ('valor', 8, 2) → default (0,01);

});

}.

↳ atribui valor default

28/12/21

- Foreign Keys (1:1) (1:n) ~~(n:n)~~

Schema:: create ('produto_detalhes') function (Blueprint \$table) {

\$table → id();

\$table → unsignedBigInteger ('produto_id');

\$table → float ('largura', 8,2)

\$table → foreign ('produto_id') → references ('id') → on ('produto');

\$table → unique ('produto_id');

↳ () que tem o relacionamento 1:1

}

Coluna

* mitude devo basta dar drop foreign antes de dropar a tabela

- Foreign Keys (n:n)

- Para fazerem um relacionamento, será necessário que creamos uma tabela pivô que vai armazenar os ids das outras tabelas e mais algumas coisas

Schema:: create ('filhos', function (Blueprint \$table) {

\$table->id();

\$table->string ('filial', 30);

→ Cria a tb filhos

Schema:: create ('produto_filhos', function (Blueprint \$table) {

\$table->id();

\$table->unsignedBigInteger ('filial_id'); → Cria a tabela

\$table->unsignedBigInteger ('produto_id'); pivô

\$table->foreign ('filial_id')->references ('id')->on ('filhas');

\$table->foreign ('produto_id')->references ('id')->on ('produtos');

porcima da da

Schema:: table ('produto_detalhes', function ...){ ↑ col. id

\$table->unsignedBigInteger ('unidade_id'); → after ('id');

\$table->foreign ('unidade_id')->references ('id')->on ('unid');

↳ define que age a tabela tal preciso clara foreign

28 / 12 / 21

D S T Q Q S S

- Comandos status, reset, refresh e push;

php artisan migrate:status

- Mostra quais migrações já foram aplicadas ou não.

php artisan migrate:reset

- lava o banco de volta ao estado inicial, apagando tudo

php artisan migrate:refresh

- limpa todas as alterações do banco.

- ele faz isso apagando tudo e depois up em tudo.

php artisan migrate fresh

- apaga tudo e da up de novo

* Eloquent ORM (Object Relational Mapping)

- transforma a classe model em um objeto com todos os propriedades, métodos e relacionamentos daquela entidade, e copia de manipular o banco através da chamada de métodos estáticos.
- Nessa classe, utilizaremos seus atributos, o nome da tabela correspondente no banco, as regras e mensagens de validação e também os seus relacionamentos

* Tinker

- ferramenta nativa do laravel

- console interativo que permite acionar as classes através de el

php artisan tinker

» \$contato = new App\SiteCont(); * podemos rodar comandos assim

» \$contato → nome = 'Pedro'; para testar o ORM

» \$contato → save();

* método estático: não depende da instância
do obj para sua exec.

D / S / T / Q / Q / S / S

3 / 1 / 22

- Iguando o nome da classe à tabela

- Por padrão, o ^{laravel} cria tabelas com o nome da classe em:
 - Snake_case
 - underscore
 - P com um s no final

↓
model

- Quando não chegar o nome da tbl de automatico na classe:

protected \$table = 'tb-tal';

- Inserindo registros com create e fillable na classe:

protected \$fillable = ['neme', 'rte', 'uf'];

No tñem:

\App\Fornecedor::create(['neme' => 'dal', 'rte' => 'tal']);

↳ atribs retados em \$fillable

model

- Selecionando registros ↑

\$fornecedores = Fornecedor::all(); ↳ aceita array

\$fornecedores = Fornecedor::find(id);

↳ \$fornecedores = Fornecedor::where('neme_cel', operador, 'valor') →
→ get(); ↳ first, last entre outros

\$forn = Forn::whereIn('celula', [valor1, valor2]);

=, >, !=, <, *]

↳ where, whereNotIn

quando for

• whereColumn, whereBetween

• whereDate, whereNotBetween

• orwhere

pode ser omitido

- Ordenando registros * aceita concatenações.

- basta passar os metodos:
:: orderBy()

- Collections

• Aceita todos os metodos das collections

→ first();

→ last();

→ reverse();

→ toArray();

→ toJson();

→ pluck(); → , retorna todos os valores de uma
col. na tabela.

- Atualizando registros

→ save();

→ fill(); → preenche por array assoc. (devem ser fillable)

→ where();) se unem.

→ update();

- Deletando registros

→ delete();

⇒ destroy(id);

* Procedores: ~~only Tharcked()~~; with Tharcked();
↳ Trás registros deletados.

9/11/22

D	S	T	Q	Q	S	S
---	---	---	---	---	---	---

- Soft Delete

- Adiciona a coluna deleted_at ao envio de apagar de fato

Na model:

use Illuminate\Database\Eloquent\SoftDeletes;

↳ Recomendo adicionar os comentários.

Class Evernecceler extends Model {

use softDeletes;

}

Na migration:

Up → Schema::create('products', function () {

\$table->id();

\$table->timestamps();

\$table->softDeletes(); → Adiciona as colunas na tb

};

Down → Schema::table('...', function () {

\$table->dropSoftDeletes();

};

Na Ctrl:

\$product->delete(); → liga o softDelete

\$product->forceDelete(); → deleta de fato

\$product = Product::withTharcked()->get(); → pega a lista

\$product[3]->restore(); → restaura

com os erros

9 / 1 / 22

D S T Q Q S S

- Seeders

→ php artisan make:Seeder nome Seeder

↳ Cria uma seed

Na classe seed:

```
public function run() {  
    $fornecedor = new Fornecedor  
    $forn → nome = 'Fern 100';  
    $forn → save();
```

metodo create

```
    }  
    $fornecedor::create([  
        'nome' => 'fern 200'  
    ]);
```

De

* tem que ser
fillable.

metodo insert

```
    }  
    DB::table('forn') → insert([  
        'nome' => 'fern 300'  
    ]);
```

De

Na DB Seeder:

```
public function run() {  
    $this → Class Seeder::class); → add todos os seeds  
}
```

↳ Comentário depois de usar
que quero remar

→ php artisan db:seed → roda o seeder

- Factories

- Semelhante à banco em massa

padrão

php artisan make:factory NomeFactory --model=NomeModel
 ↳ cria uma factory.

No factory:

return [

↳ metodos da lib factory, consultar todos na doc

'name' => \$faker->name,

github.com/brenoott/factory

'email' => \$faker->email,

]

No Seeder:

public function run {

quant de registros

factory('NomeFactory::class', 100)->create();

}

php artisan db:seed --class: Nome seed

↳ opcional

* Middlewares

php artisan make: middleware NomeMiddleware

↳ Cria um middleware

- Middlewares são agentes que interceptam determinadas requisições e respostas antes mesmo de mandar pra processar na controller.

11/1/21

D S T Q Q S S

Na rota desejada:

Rota:: middleware (NameMiddleware :: class) → get ('/...');
↳ Coloca o middleware entre a requisição e a aplicação

Em um controlador:

```
public function __construct () {  
    $this->middleware (NameMiddleware :: class);  
}
```

- Implementando um middleware p/ tratar as rotas.

No Kernel: → Em app/Http

\$protected \$middlewareGroups = [* não ver de modo
 'web' => [sequential

\App\HTTP\Middleware\Name :: class,

],

→ Aplica o middleware à todas as rotas "web"

- Aplicando um middleware

No Kernel:

\$protected \$routeMiddleware = [

'aplicativo' => \App\HTTP\Middleware\Name :: class

Ir: \$this->middleware ('aplicativo') → Chama o middleware

- Encadeamento de microlivros

Route: mcdalllware ('mcd 1','mcd 2') ...

return \$next(\$request);

↳ necessárias para regular o fluxo.

- Parando parámetros para el modelamiento

No middleware:

public function handle(\$request, Closure \$next, \$param) {

The chamado:

Rente:: get('...') → multilayerne ('apliche: param')

declare no
fine.

Otros ferma

\downarrow acetate
Quantities are
given

- Manipulando a resposta de uma requisição.

Na handle che middleware:

Expense = Rent (Request)

↳ var que guarda o resposta

processada pela aplicação em

```
response.setStatusCode(201, 'mensagem');
```

L> manipula e responde.

return \$response

* Trabalhando com formulários

- Request

- Requisição feita pelo cliente para o back-end da aplic.
- Pode ser recuperada em forma de obj no back.

```
public function store name(Request $request) {
    ↳ Obj da requisição
```

\$contato = new Contato();

\$contato->email = \$request->email

\$contato->Save();

↳ nome do input

* A validação do form já foi explicada mais pra trás

- Repetindo o form. (Request obj. input) ↳ nome do input

<input name="nome" value="{{ \$contato['nome'] }}>

↳ Pega o request

anterior..

- Apresentando os erros de validação

Q. if (\$errors->has('nome-input'))

* <label> {{ \$errors->first('nome-input') }} </label>

↳ msg de erro padrão

- Customizando msgs de feedback. ↳ array de valid.

\$request->validate(['nome' => 'required|min:3'],

['nome.required' => 'msg'],

'nome.min' => 'msg'

↳ msg. → validação

* Um termo que serve

↳ nome campo

* Trabalhando com planilhas

- Lib Xaravel Excel

- Biblioteca para tratamento e manipulação de planilhas com Xaravel disponível via composer.

composer require maatwebsite/excel

* APIs, Webservices e Rest.

Tudo duplicado na matéria de base

- API

Um acrônimo de Application Programming Interface, uma API nada mais é que um conjunto de rotinas e pacotes estabelecidos por um software para a utilização de suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software.

• Exemplos:

- O clássico exemplo do restaurante, onde o garçom (api) apenas pega os pedidos dos clientes (front-end) e leva pra cozinha (back-end), sem que o cliente precise se preocupar em como o pedido foi preparado na cozinha.

Apis são usadas também para conectar back-end de front, pois como vimos a seguir, foi conveniencioso que apis sempre devem retornar json, isso possibilita que o mesmo back-end sirva aplicações em diversas plataformas (web, mobile etc), ficando a cargo do seu front exibir esses dados como ele quiser.

- Rest

Um acrônimo de Representational state transfer, rest nada mais é que uma série de padronizações que uma API deve ter para trabalhar com o protocolo HTTP.

Quando uma API atende todos os requisitos de rest, ela se torna uma API Restful.

- Rutas Web e API

Dois grupos de rotas diferentes, sendo o primeiro usado geralmente para fazer uma requisição normal e retornar uma view.

Já a segunda, deve ser utilizada para o endpoint de endpoints da API, retornando um JSON (`/api/{endp}`). ↳ indica

- Content Type

- `Text/html`: Content type de uma resposta que a API retorna uma view.
- `Application/json`: Content type de uma resposta desenvolvida por uma API.

- Rutas API

`Route::apiResource('Insumi', 'ControlInsumi');`

↳ abstrai os recursos `create` e `edit`

↳ Cria rotas invisíveis para cada função

* `php artisan route:list` → Consulta as rotas.

* `App/Providers/RouteServiceProvider.php`

↳ Descomentar a linha com `$namespace`

Para podermos passar apenas o nome da Ctrl.

- Cadaç via api

• Post

• Verbo Http responsável por persistir dados na aplic.

• Os parâmetros inseridos no body da requisição podem ser acessados através da request.

* /api/carros/

↳ endpoint

• Get

• Verbo HTTP responsável por recuperar dados da aplic.

• Existem 2 endpoints padrão para esse fim:

- /api/carros/ → index

* se o param. for tipado.

- /api/carros/4 → show

o laravel recupera o

↳ id do objeto registro com aquele id

• Put / Patch

• Verbo http responsável por alterar dados da aplic.

• Os param a ser alterados devem ser inseridos no body da requisição e tratados na Ctrl.

• Patch = atualiza apenas alguns dados do registro

• Put = altera o registro inteiro.

* /api/carros/2 → id

• Delete

• Verbo Http responsável por apagar registros de banco.

• Pode ser tratado para softDelete

* /api/carros/2 → id

* Url → wwwzap.com

Urn → /api/cadastrar

Uri → wwwzap.com/api/cadastrar

2 / 6 / 22

D S T Q Q S S

- Formas de manipular a model na ctrl.

• Estática

Empresa::all();

• Type Hint

```
public function show(Empresa $empresa) {  
    return $empresa; }  
} → Quando a tipagem é uma  
model, o laravel autem.  
resgata a model da id  
passada.
```

• Injeção de model

```
public function __construct(Marca $marca) {  
    $this->marca = $marca;  
}  
} → A partir disso, referencias  
incluem a classe.
```

```
public function index() {  
    $marcas = $this->marca->all();  
    return $marcas;  
}  
}
```

de → remove a

```
public function show($id) {  
    $marca = $this->marca->find($id);  
    return $marca;  
}  
}
```

tipagem

— API Responses

* o client precisa passar
o header accept =
application/json

```
public function show (Empresa $emp) {
```

```
if ($emp == null) {
```

```
return response () -> json (
```

['erro' => 'msg'] → array com os erros

, 404 → http status code

); Sempre consultar a
tabela.

Validação de dados com Validator

Validator :: make (\$request, \$rules, \$msg) → msgs definidas na rule
↳ regras definidas
no model

Variáveis em sessão com Session

Session :: put ('nome', \$valor); → seta o valor

Session :: get ('nome'); → Regata

Session :: forget ('nome'); → esquece.

Criar diretiva blade

na class App Service Provider: → provider heretado em toda a aplic.

```
public function boot () {
```

```
Blade :: directive ('name', callbackfunction());
```

↳ módulo

↳ função

Gravação de logs.

`Log::info('mensagem', [$clados])`

↓ ↳ nível de log, existe: debug
info

facade

notice

warning

error

critical

alert

emergency.

Facade

Interfaces estáticas fornecidas pelo Laravel como:

- Log
- Service
- Validator
- Db

• Str. → Para serem chamadas por esses apelidos, elas devem ser definidas no array de aliases referenciando a facade em config/app.php

Providers etc.

App Service Provider → Local onde podemos aplicar algumas config para a nossa aplic. assim que ela boot

Mithstatus e Mitherrors

Na controller:

return redirect() → back() → mitherrors('msg'); array de
→ mithstatus('msg'); mensagens.

Na view:

@if(\$errors → any(1))

 <div class="alert alert-danger"> → Classe Bootstrap

 @foreach (\$errors → all() as \$error)

 {{ \$error }} → Array de msgs

 @endforeach → msg

</div>

@endif.

@if (session('status'))

 <div class="alert alert-success">

 {{ session('status') }}

</div>

@endif.

Ajax + Laravel

→ tipo Jquery → id do elemento

Ex: `$("#botao").click(function(e){`

`e.preventDefault();` → não envia o form

Jquery → `$.ajax({`

`url: "{ route('name') }",`

`method: "post",`

`data: {`

`_token: "{{ csrf_token() }}",` → request
`name: $("#name")`

`},`

`success: function(result){`

`console.log(result);`

`},`

↳ prints resultado

`error: function(error){`

`console.log(error);`

`});`

`return false;`

`});`