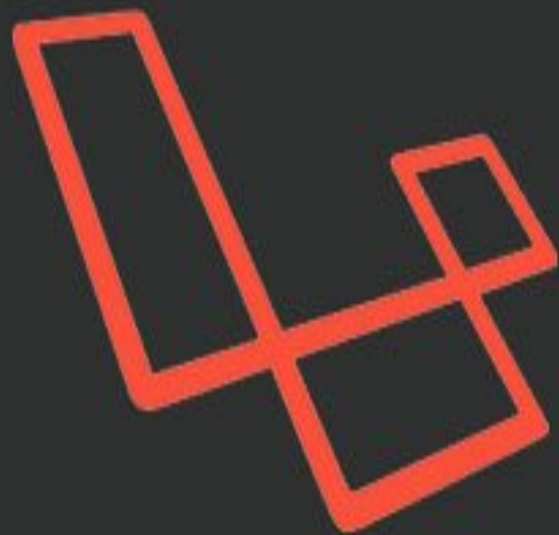


# Desenvolvimento de Aplicações Web

**Prof. Dr. Guilherme A. Madalozzo**

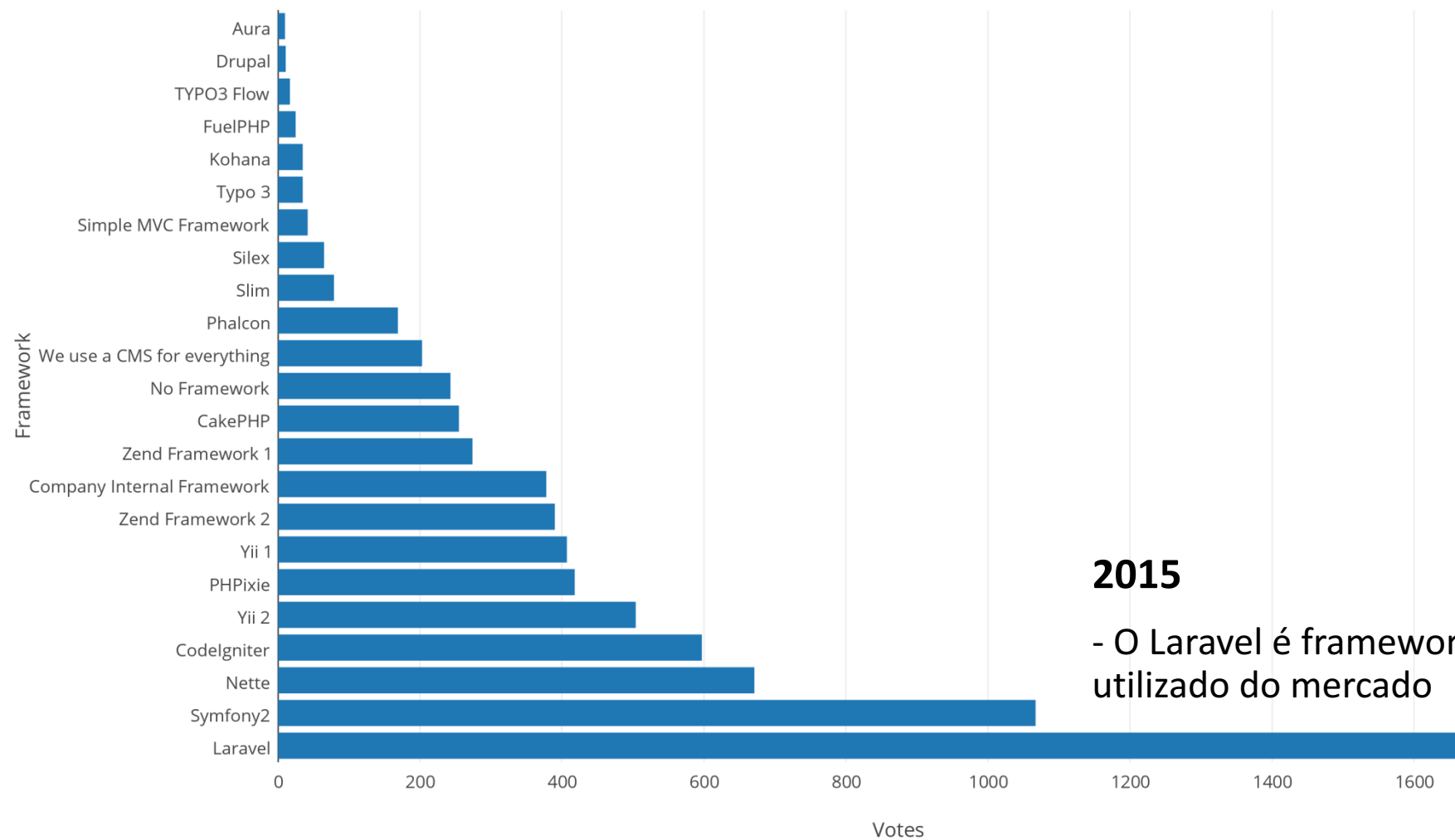


laravel

# Laravel

- Programadores não querem se preocupar com infraestrutura
- Então, surgiram os frameworks
  - Nos ajudam a agilizar o processo de desenvolvimento
  - Deixa o projeto bastante organizado
  - Evita repetições de código
- **Laravel** é um framework PHP extremamente produtivo com arquitetura **MVC**
- Está cada vez mais popular e sendo utilizado no mercado

PHP Framework Popularity at Work - SitePoint, 2015



**2015**

- O Laravel é framework mais utilizado do mercado

# Laravel

- O Laravel possui uma interface de linha de comando
  - **Artisan** (artesão)
- Através do **Artisan** podemos realizar diversas ações em nossas aplicações
  - Configuração de ambiente
  - Verificar rotas (veremos a seguir o que são) existentes
  - Interagir com a aplicação
  - Criar diversos tipos de arquivos
    - Migrations (banco de dados)
    - Controllers
    - Models

# Laravel

- Para a criação de interface gráfica o Laravel utiliza uma ferramenta de template
  - **Blade**
- Blade traz um quantidade grande de ferramentas que ajudam na criação de bonitas interfaces de forma rápida

# Laravel

- Para a comunicação com bancos de dados o Laravel usa uma implementação chamada Eloquent ORM (Mapeador Relacional de Objetos)
- O Eloquent ORM é uma ferramenta com funcionalidades que facilitam a inserção, atualização, busca e exclusão de registros diretamente no banco de dados

# Laravel

## Pré-requisitos do Laravel

- PHP na versão 5.6 ou superior
- Habilitar extensões no php.ini
  - OpenSSL PHP
  - PDO PHP
  - Mbstring PHP
  - Tokenizer
- Composer
- Node.js e NPM



# Fazer o tutorial de instalação e configuração dos requisitos do **Laravel**

*Tutorial está na intranet*

# Laravel na prática!

# Laravel na Prática

- Vamos iniciar o desenvolvimento de uma aplicação web (futuramente, se der tempo, essa aplicação será migrada para mobile)
- A aplicação em questão é um sistema de controle de hábitos
  - Neste sistema controlaremos os hábitos do nosso dia-a-dia
  - Com ele, vamos saber o que estamos deixando de fazer
  - Com ele, vamos saber quais novos hábitos estamos construindo
  - Toda vez que o usuário fizer algo, marcará no sistema
  - Com isso, teremos um histórico dos hábitos

# Laravel na Prática

- O sistema conterá, basicamente, três cadastros
  - Cadastro de hábitos
  - Cadastro de histórico
  - Cadastro de usuários

# Laravel na Prática

- O sistema conterà, basicamente, três cadastros
  - Cadastro de hábitos
  - Cadastro de histórico
  - Cadastro de usuários
- Cadastro de hábitos
  - Nome
  - Descrição
  - Tipo de hábitos
  - Dia de início de controle

# Laravel na Prática

- O sistema conterá, basicamente, três cadastros
  - Cadastro de hábitos
  - Cadastro de histórico
  - Cadastro de usuários
- Cadastro de hábitos
  - Objetivo (quantidade que deseja fazer)
    - Quantidade mensal
    - Quantidade semanal
    - Quantidade diária

# Laravel na Prática

- O sistema conterá, basicamente, três cadastros
  - Cadastro de hábitos
  - Cadastro de histórico
  - Cadastro de usuários
- Cadastro de histórico
  - Data
  - Hora
  - Local

# Laravel na Prática

- O sistema conterá, basicamente, três cadastros
  - Cadastro de hábitos
  - Cadastro de histórico
  - Cadastro de usuários
- Cadastro de usuários
  - O Laravel possui um mecanismo de autenticação automatizado
  - Veremos na sequência do desenvolvimento da aplicação



# ***Etapas 1***

Iniciando o projeto MeuHabitto.Com

Entendendo a estrutura de pastas

# MeuHabitto.Com

- Como vimos no tutorial, criar um projeto Laravel é simples
  - Utilizaremos o `cmd` para isso

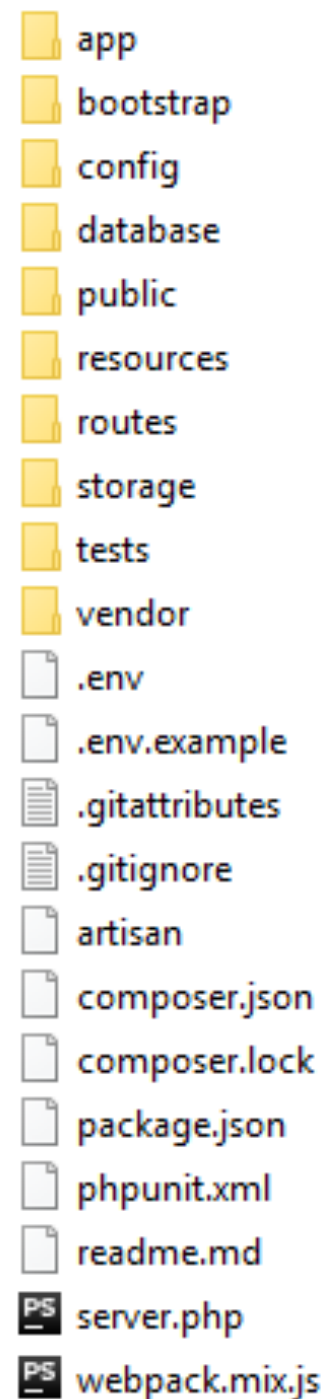
```
$ composer create-project --prefer-dist laravel/laravel meu_habito
```

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~  
$ composer create-project --prefer-dist laravel/laravel meu_habito  
Installing laravel/laravel (v5.4.19)  
- Installing laravel/laravel (v5.4.19): Loading from cache  
Created project in meu_habito  
> php -r "file_exists('env') || copy('env.example', 'env');"  
> php artisan optimize  
Generating optimized class loader  
The compiled services file has been removed.  
> php artisan key:generate  
Application key [base64:B8R0YgeA97lity/H1J75GecWCAKwCvEq2W8mPHJ1Ako=] set successfully.
```

[https://github.com/guimadalozzo/meu\\_habito](https://github.com/guimadalozzo/meu_habito)

# MeuHabitto.Com

- Ao criarmos um projeto, o Laravel cria toda a arquitetura
- Podemos notar que o diretório meu\_habito está estruturado

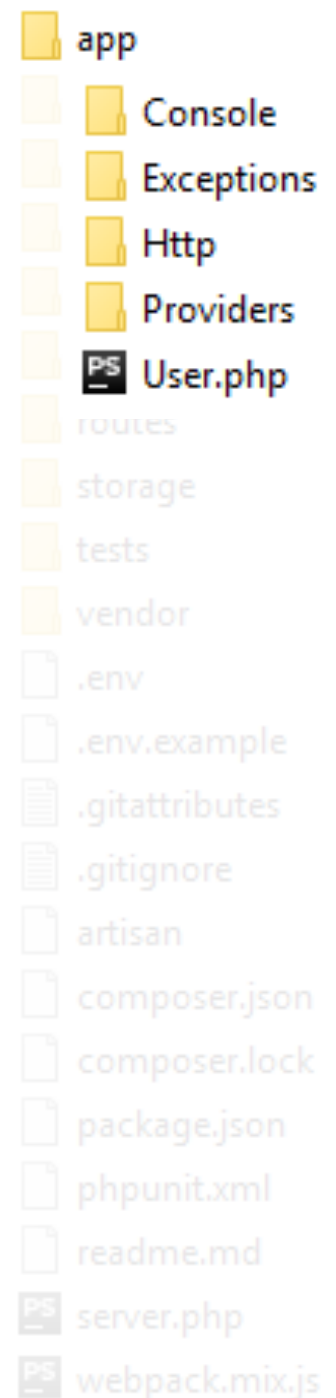


The image shows a file explorer window displaying the directory structure of a Laravel project. The files and folders are listed in a vertical column on the right side of the screen. The folders are represented by yellow icons, and the files are represented by white icons. The files include configuration files like .env, .env.example, .gitattributes, and .gitignore, as well as Laravel-specific files like artisan, composer.json, composer.lock, package.json, phpunit.xml, and readme.md. There are also two PHP files, server.php and webpack.mix.js, which are represented by black icons with 'PS' in the top left corner.

- app
- bootstrap
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js

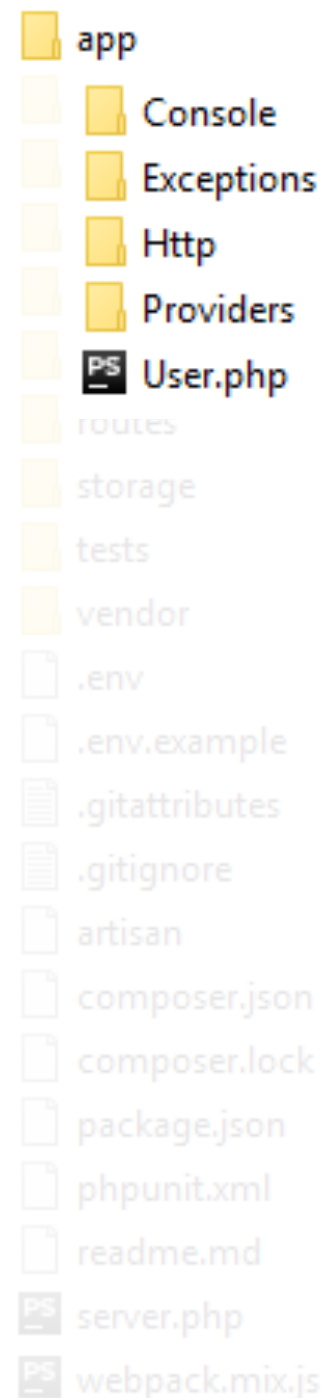
# MeuHabitto.Com

- A pasta *app* é a principal pasta do projeto
  - É onde vamos desenvolver a aplicação
  - A subpasta *Console* para criar comando (se necessário)
    - Artisan
  - A subpasta *Exceptions* responsável pelo tratamento de erros do Laravel



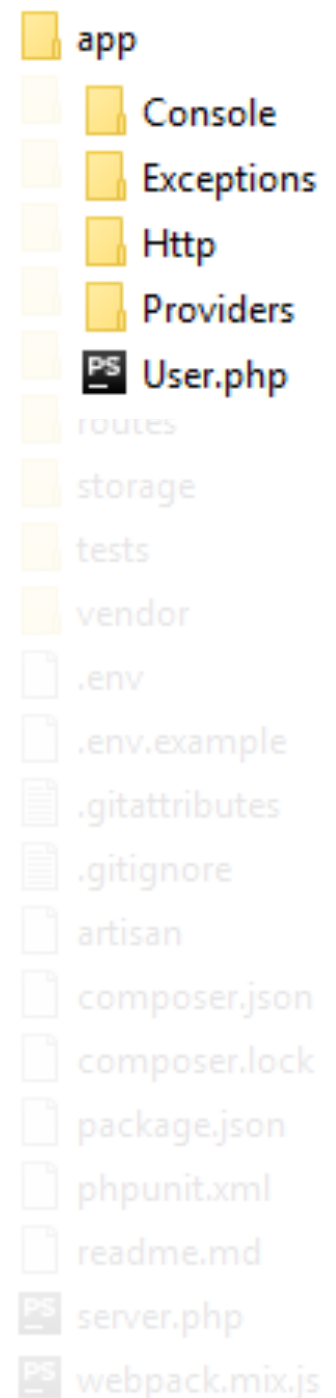
# MeuHabitto.Com

- A pasta *app* é a principal pasta do projeto
  - É onde vamos desenvolver a aplicação
  - A subpasta *HTTP* conterà
    - Controllers → gerenciamento das ações
    - Middleware → recurso importante para a web, onde podemos interceptar a requisição do usuário e analisa-la antes de devolver uma resposta



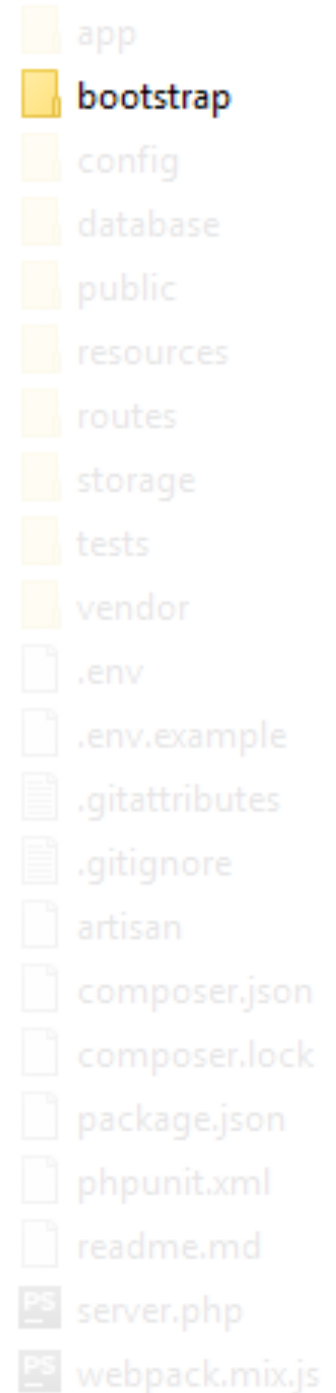
# MeuHabitto.Com

- A pasta *app* é a principal pasta do projeto
  - É onde vamos desenvolver a aplicação
  - A subpasta *Providers* conterá os provedores de serviços da aplicação → gerenciadores de rotas, autenticação, notificações real-time, etc
  - O arquivo *user.php* é o Model de usuário



# MeuHabitto.Com

- A pasta *bootstrap* é responsável por gerenciar os pacotes de autoload

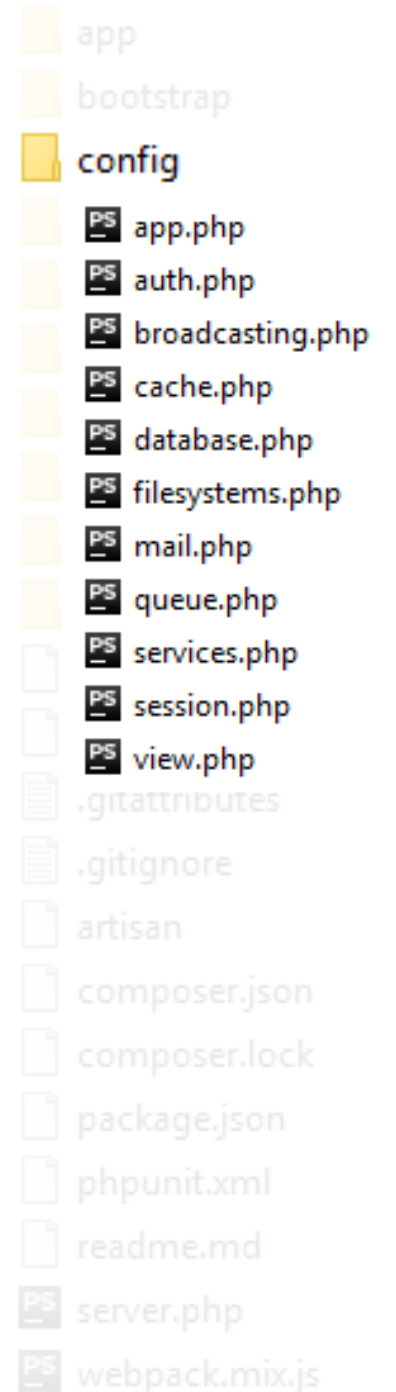


A vertical list of files and folders, each preceded by a small icon. The 'bootstrap' folder is highlighted with a yellow background. The icons are: yellow folder for directories, white document for text files, and blue document with 'PS' for PHP files.

- app
- bootstrap**
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js

# MeuHabitto.Com

- A pasta *config* é a pasta para a configuração da nossa aplicação Laravel
  - Autenticação
  - Banco de dados
  - BroadCasting (notificações RT)
  - Emails
  - Sessões
  - Upload de arquivos
  - Serviços de terceiros



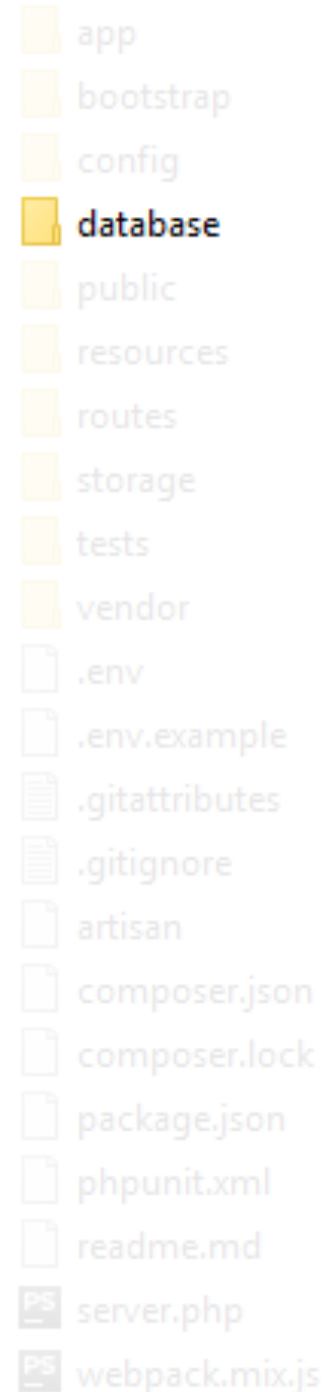
A vertical list of files and folders from a file explorer, representing a Laravel project structure. The 'config' folder is highlighted in yellow. Files with a 'PS' icon (PHP) are also highlighted in yellow, while others have a white document icon. The list includes:

- app
- bootstrap
- config
- app.php
- auth.php
- broadcasting.php
- cache.php
- database.php
- filesystems.php
- mail.php
- queue.php
- services.php
- session.php
- view.php
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js



# MeuHabitto.Com

- A pasta *Database* é a pasta onde criaremos as *migrations* para versionamento da base de dados
- Migrations são arquivos de código que representam operações na estrutura do banco de dados

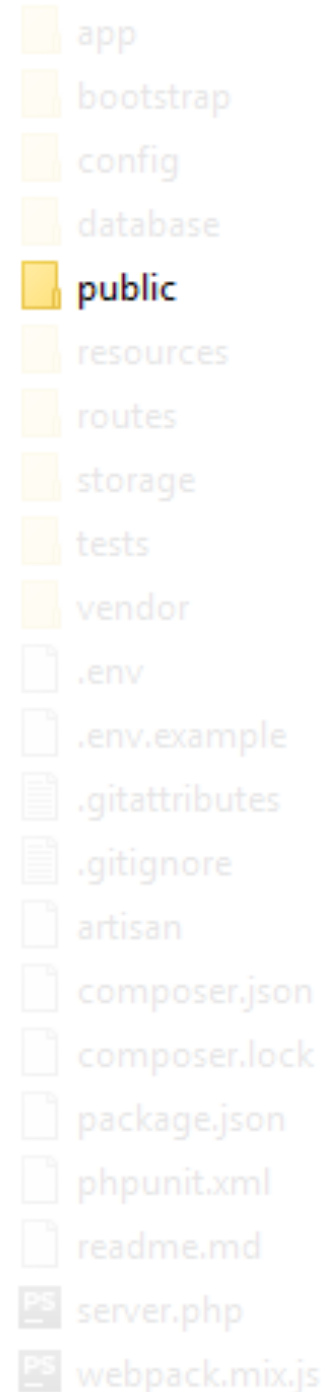


A vertical list of files and folders in a file explorer. The 'database' folder is highlighted in yellow. Other folders include 'app', 'bootstrap', 'config', 'public', 'resources', 'routes', 'storage', 'tests', and 'vendor'. Files include '.env', '.env.example', '.gitattributes', '.gitignore', 'artisan', 'composer.json', 'composer.lock', 'package.json', 'phpunit.xml', 'readme.md', 'server.php', and 'webpack.mix.js'. The 'server.php' and 'webpack.mix.js' files have a 'PS' icon next to them.

- app
- bootstrap
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js

# MeuHabitto.Com

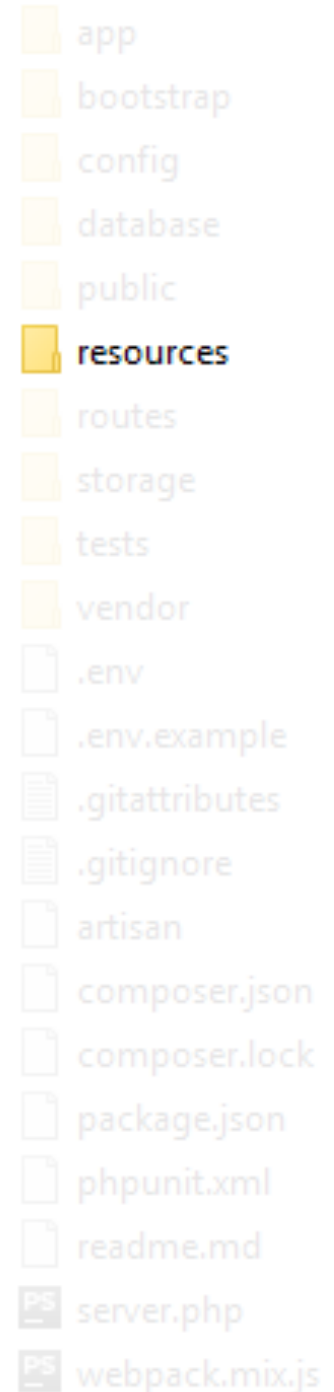
- A pasta *Public* é a pasta onde contém o index.php
  - Para iniciar a aplicação
- Também contém o .htaccess para passar configurações ao servidor apache



app  
bootstrap  
config  
database  
public  
resources  
routes  
storage  
tests  
vendor  
.env  
.env.example  
.gitattributes  
.gitignore  
artisan  
composer.json  
composer.lock  
package.json  
phpunit.xml  
readme.md  
server.php  
webpack.mix.js

# MeuHabitto.Com

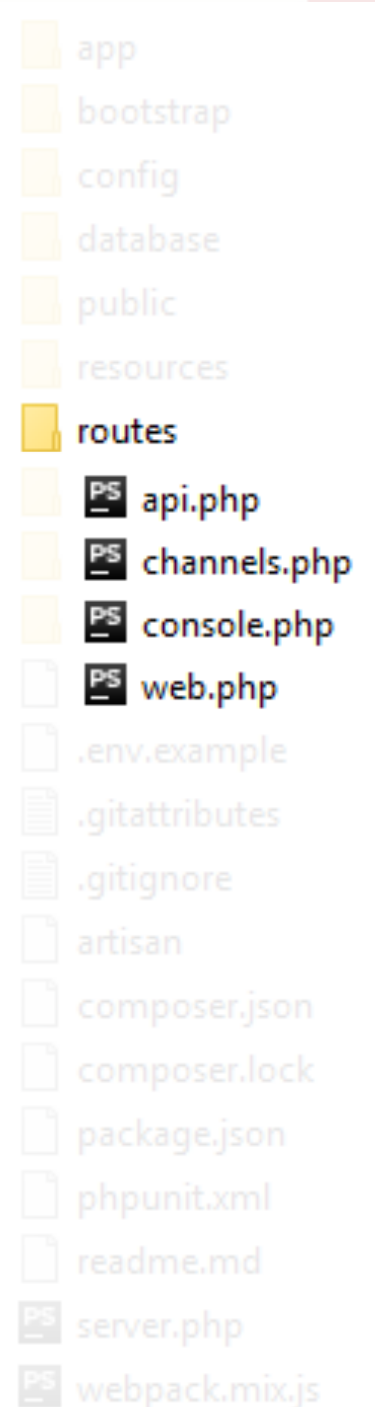
- A pasta *Resources* é a pasta contém algumas configurações front-end
  - Podemos trabalhar nossa aplicação em diferentes linguagens (inglês é o padrão)
  - Contém a pasta *Views* onde terão nossos templates da visão da aplicação



app  
bootstrap  
config  
database  
public  
resources  
routes  
storage  
tests  
vendor  
.env  
.env.example  
.gitattributes  
.gitignore  
artisan  
composer.json  
composer.lock  
package.json  
phpunit.xml  
readme.md  
server.php  
webpack.mix.js

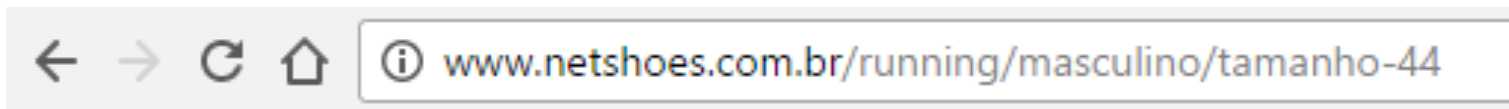
# MeuHabitto.Com

- A pasta *Routes* é a pasta onde nos preocuparemos com as rotas da aplicação
  - Rotas são definições de qual Controller será acessado a partir da informação da URL
  - Podemos usar comando simples nas rotas, conforme requisição do http informado
    - `Route::get();`
    - `Route::post();`
    - `Route::put();`
    - `Route::delete();`
    - `Route::any();`

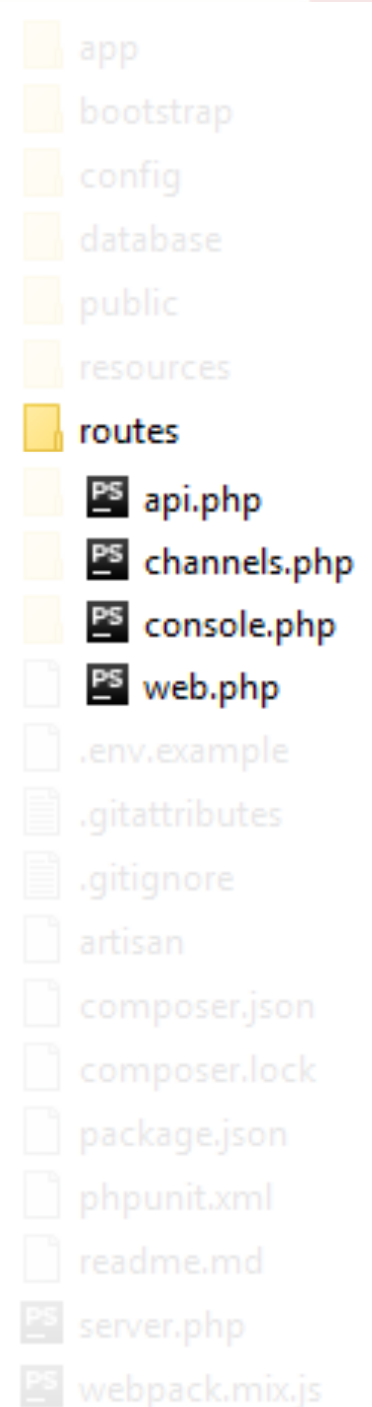


# MeuHabito.Com

- A pasta *Routes* é a pasta onde nos preocuparemos com as rotas da aplicação
  - Através das rotas podemos parametrizar a URL, por exemplo
    - No site da netshoes eu quero ver os tênis de corrida, masculino e tamanho 44

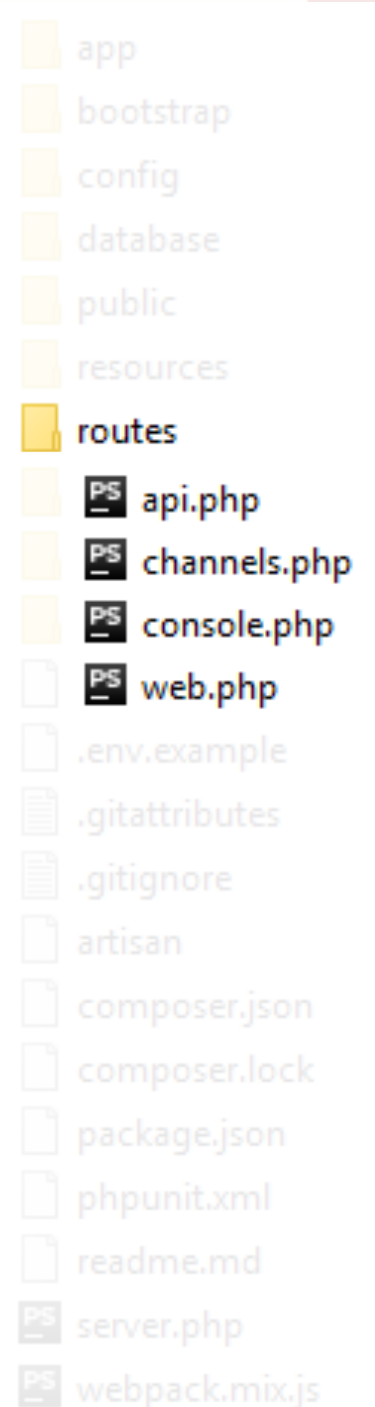


- Notamos 3 parâmetros
  - Categoria (**running**), Gênero (**masculino**), Tamanho (**tamanho-44**)



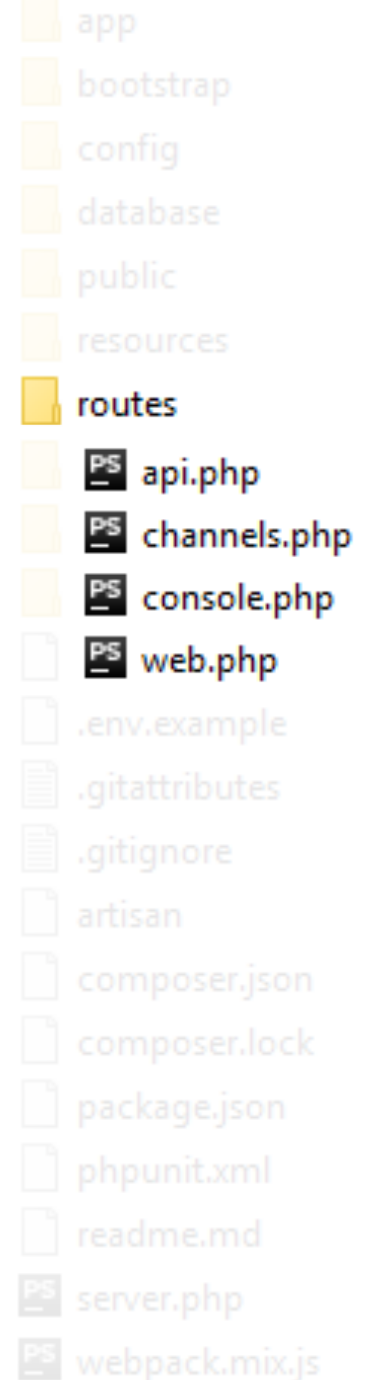
# MeuHabitto.Com

- A pasta *Routes* é a pasta onde nos preocuparemos com as rotas da aplicação
  - Então, pode-se dizer que as rotas são as URLs que o usuário utiliza para acessar as páginas dentro do sistema
  - O programador pode controlar e personalizar estas rotas para que fique mais amigável aos usuários



# MeuHabitto.Com

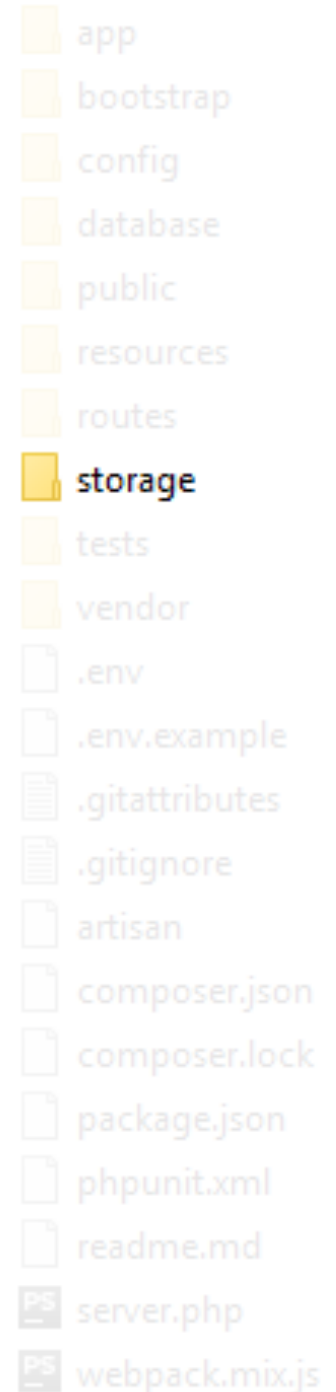
- A pasta *Routes* é a pasta onde nos preocuparemos com as rotas da aplicação
  - No Laravel configuramos as rotas no arquivo web.php
  - O console.php é o arquivo para configurar rotas do artisan
  - Para notificações RT usamos o channels.php
  - O arquivo api.php faz rotas dos middlewares



app  
bootstrap  
config  
database  
public  
resources  
routes  
api.php  
channels.php  
console.php  
web.php  
.env.example  
.gitattributes  
.gitignore  
artisan  
composer.json  
composer.lock  
package.json  
phpunit.xml  
readme.md  
server.php  
webpack.mix.js

# MeuHabito.Com

- Na pasta *Storage* o Laravel mantém um histórico de logs da aplicação
- Também, vai armazenar as caches e sessões do sistema



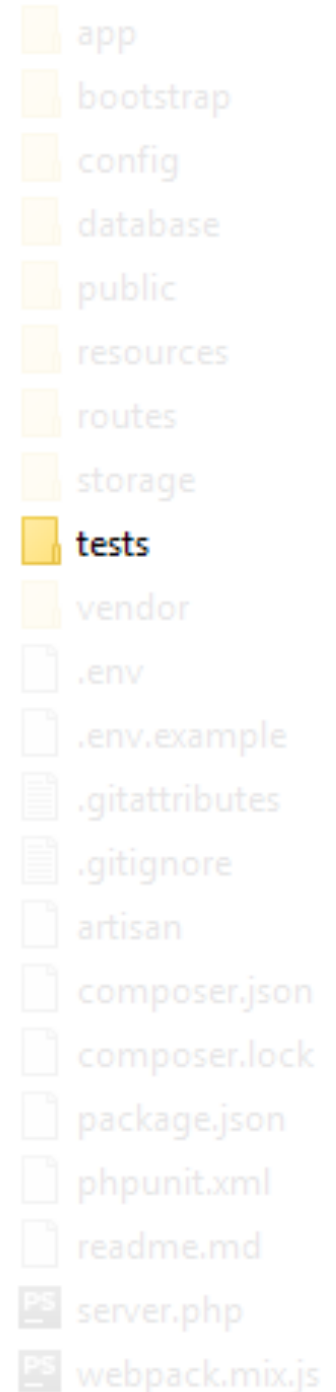
A vertical sidebar on the right side of the slide, styled like a file explorer, lists the files and folders of a Laravel project. The items are: app, bootstrap, config, database, public, resources, routes, storage (highlighted with a yellow folder icon), tests, vendor, .env, .env.example, .gitattributes, .gitignore, artisan, composer.json, composer.lock, package.json, phpunit.xml, readme.md, server.php, and webpack.mix.js. The last three items have a small 'PS' icon next to them.

- app
- bootstrap
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js



# MeuHabitto.Com

- Na pasta *Test* o Laravel possibilita que trabalhemos com o PHPUnit para realização de testes unitários de nossas classes

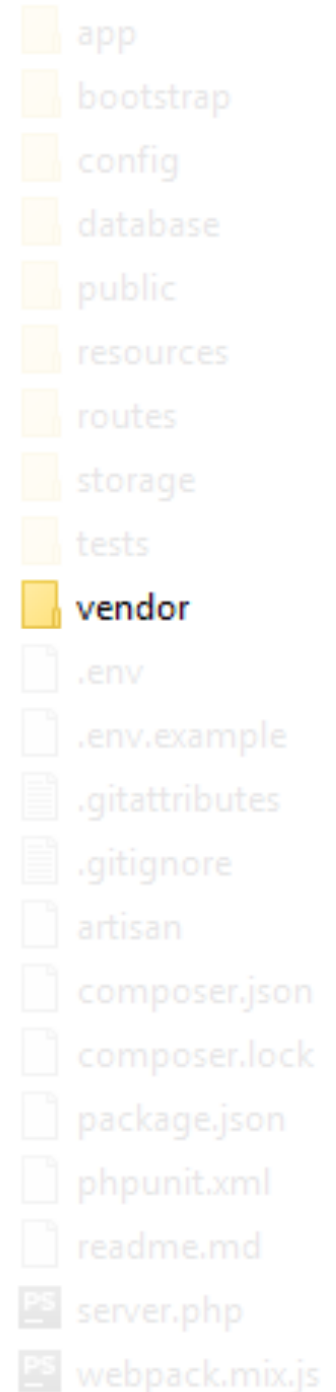


A vertical list of files and directories from a Laravel project. The 'tests' directory is highlighted with a yellow folder icon. Other items include 'app', 'bootstrap', 'config', 'database', 'public', 'resources', 'routes', 'storage', 'vendor', '.env', '.env.example', '.gitattributes', '.gitignore', 'artisan', 'composer.json', 'composer.lock', 'package.json', 'phpunit.xml', 'readme.md', 'server.php', and 'webpack.mix.js'.

- app
- bootstrap
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js

# MeuHabitto.Com

- Na pasta *Vendor* contém todos os pacotes de terceiros que usamos na nossa aplicação
- Nunca deve ser alterada



A vertical list of files and folders, each preceded by a small icon. The 'vendor' folder is highlighted with a yellow icon and bold text. Other folders like 'app', 'bootstrap', 'config', 'database', 'public', 'resources', 'routes', 'storage', and 'tests' have yellow folder icons. Configuration files like '.env', '.env.example', '.gitattributes', and '.gitignore' have document icons. Build files like 'artisan', 'composer.json', 'composer.lock', 'package.json', 'phpunit.xml', 'readme.md', 'server.php', and 'webpack.mix.js' have document icons, with the last two having a 'PS' icon in the bottom left corner of their respective document icons.

- app
- bootstrap
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor**
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json
- phpunit.xml
- readme.md
- server.php
- webpack.mix.js

## ***Etapa 2***

Configurando a base de dados do MeuHabito.Com

Para executar projeto:  
cd meu\_habito  
php artisan serve

# Configuração da Base de Dados

- Por padrão o Laravel possibilita o uso dos seguintes gerenciadores de banco de dados
  - Sqlite
  - MySQL
  - PostgreSQL

# Configuração da Base de Dados

- Por padrão o Laravel possibilita o uso dos seguintes gerenciadores de banco de dados
  - Abrindo o arquivo *config/database.php*
    - Veremos que os três gerenciadores de banco de dados estão configurados no array de conexões
  - Podemos notar, também, que existe uma conexão 'default'

```
'default' => env( key: 'DB_CONNECTION', default: 'mysql' ),
```

# Configuração da Base de Dados

- Por padrão o Laravel possibilita o uso dos seguintes gerenciadores de banco de dados
- Para o sistema MeuHabitto.com usaremos o MySQL

```
42 'mysql' => [  
43     'driver' => 'mysql',  
44     'host' => env( key: 'DB_HOST', default: '127.0.0.1'),  
45     'port' => env( key: 'DB_PORT', default: '3306'),  
46     'database' => env( key: 'DB_DATABASE', default: 'forge'),  
47     'username' => env( key: 'DB_USERNAME', default: 'forge'),  
48     'password' => env( key: 'DB_PASSWORD', default: ''),  
49     'unix_socket' => env( key: 'DB_SOCKET', default: ''),  
50     'charset' => 'utf8mb4',  
51     'collation' => 'utf8mb4_unicode_ci',  
52     'prefix' => '',  
53     'strict' => true,  
54     'engine' => null,  
55 ],
```

# Configuração da Base de Dados

- Podemos notar que alguns atributos contêm a definição através da função env(...)

```
42 'mysql' => [  
43   'driver' => 'mysql',  
44   'host' => env key: 'DB_HOST', default: '127.0.0.1'),  
45   'port' => env key: 'DB_PORT', default: '3306'),  
46   'database' => env key: 'DB_DATABASE', default: 'forge'),  
47   'username' => env key: 'DB_USERNAME', default: 'forge'),  
48   'password' => env key: 'DB_PASSWORD', default: ''),  
49   'unix_socket' => env( key: 'DB_SOCKET', default: ''),  
50   'charset' => 'utf8mb4',  
51   'collation' => 'utf8mb4_unicode_ci',  
52   'prefix' => '',  
53   'strict' => true,  
54   'engine' => null,  
55 ],
```

# Configuração da Base de Dados

- Esta função busca dados vindos do arquivo `.env` (raíz do projeto)

```
42 'mysql' => [  
43   'driver' => 'mysql',  
44   'host' => env( key: 'DB_HOST', default: '127.0.0.1'),  
45   'port' => env( key: 'DB_PORT', default: '3306'),  
46   'database' => env( key: 'DB_DATABASE', default: 'forge'),  
47   'username' => env( key: 'DB_USERNAME', default: 'forge'),  
48   'password' => env( key: 'DB_PASSWORD', default: ''),  
49   'unix_socket' => env( key: 'DB_UNIX_SOCKET', default: ''),  
50   'charset' => 'utf8mb4',  
51   'collation' => 'utf8mb4_unicode_ci',  
52   'prefix' => '',  
53   'strict' => true,  
54   'engine' => null,  
55 ],
```

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret



# Configuração da Base de Dados

- Mudaremos o `.env` adicionando as configurações do banco de dados de nossa aplicação

```
42 'mysql' => [  
43   'driver' => 'mysql',  
44   'host' => env( key: 'DB_HOST', default: '127.0.0.1'),  
45   'port' => env( key: 'DB_PORT', default: '3306'),  
46   'database' => env( key: 'DB_DATABASE', default: 'forge'),  
47   'username' => env( key: 'DB_USERNAME', default: 'forge'),  
48   'password' => env( key: 'DB_PASSWORD', default: ''),  
49   'unix_socket' => env(  
50   'charset' => 'utf8mb4',  
51   'collation' => 'utf8mb4',  
52   'prefix' => '',  
53   'strict' => true,  
54   'engine' => null,  
55 ],
```

```
DB_CONNECTION=mysql  
DB_HOST=127.0.0.1  
DB_PORT=3306  
DB_DATABASE=meuhabito  
DB_USERNAME=root  
DB_PASSWORD=root1706
```

# ***Etapa 3***

## Configurando rotas

# Configuração de Rota

- O Kernel do Laravel tem a função de inicializar seus principais componentes para podermos tratar os requests de usuários
- Uma forma para o tratamento de requests são as rotas
  - Temos diferentes maneiras de manipularmos as rotas

# Configuração de Rota

- Quando a rota está definida e ocorre um *match* com a requisição do usuário, a função da rota é executada
- O arquivo de configuração das rotas é o `routes/web.php`

```
1  <?php
2
3  /* ... */
13
14  Route::get('/', function () {
15      return view('welcome');
16  });
```

# Configuração de Rota

- A rota padrão do projeto é a rota do welcome

Laravel

DOCUMENTATION

LARACASTS

NEWS

FORGE

GITHUB

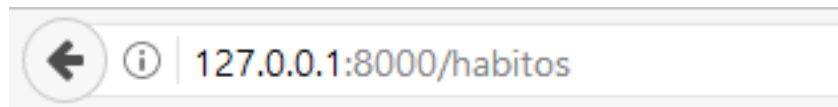
- Vamos manter ela!!

# Configuração de Rota

- Vamos criar uma rota do tipo GET → chamada “habitos”

```
18 Route::get('habitos', function() {  
19     return "Olá!" .  
20         "<br>Esta é a rota para cadastro/listagem de hábitos!";  
21 });
```

- Inicialmente esta rota irá apresentar uma mensagem em tela



Olá!

Esta é a rota para cadastro/listagem de hábitos!

# Configuração de Rota

- Quando alguém acessar o `/habitos`, via método GET, o Laravel retornará a `mensagem` configurada na rota



- Porém, muitas vezes queremos e devemos fazer com que uma rota execute uma função de um “terceiro”
  - Neste ponto entra o **Controller**!

# Configuração de Rota

- Já tivemos contado com controller em outros momentos
  - Por ora, é importante lembrar que o controller é uma classe que possui métodos
- Sendo assim, uma rota pode encaminhar a requisição (no caso, get) para uma determinada classe (controller) e para um determinado método

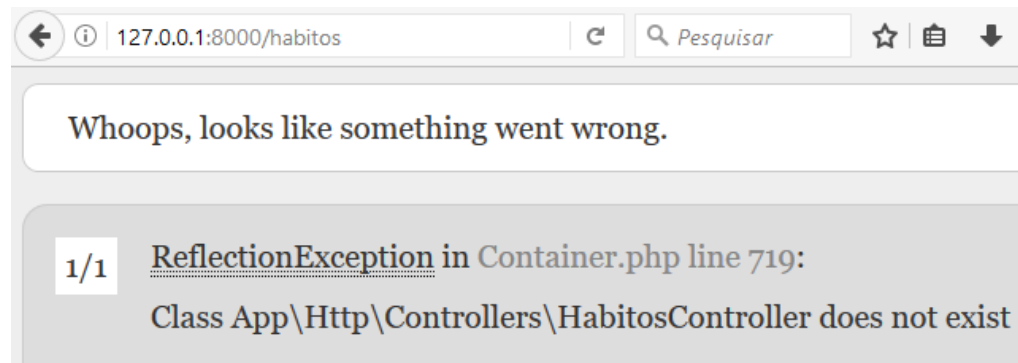


# Configuração de Rota

- Vamos alterar nossa rota de *habitos* para encaminhar a requisição ao método *index* do *HabitosController*

```
18 Route::get('habitos', 'HabitosController@index');
```

- Se fizermos uma requisição para hábitos, dará erro pois o controlador não está criado



# Configuração de Rota

- Vamos alterar nossa rota de *habitos* para encaminhar a requisição ao método *index* do *HabitosController*

```
18 Route::get('habitos', 'HabitosController@index');
```

- Dessa maneira, estamos querendo dizer
  - Quando um request para URL /habitos, através do método HTTP GET, for solicitado
  - Executaremos o método index da classe HabitosController

# ***Etapa 4***

## Criando o Controller

# Controlador

- O **Controller** é a camada conhecida como **intermediador**
  - Ele recebe uma requisição
  - Analisa as necessidades da requisição
  - Pode chamar os *models* para consultar ou persistir dados no BD
  - Depois, deve definir como a resposta será retornada
- Inicialmente, vamos criar o **controller** e no método index vamos **retornar um HTML** como **resposta**

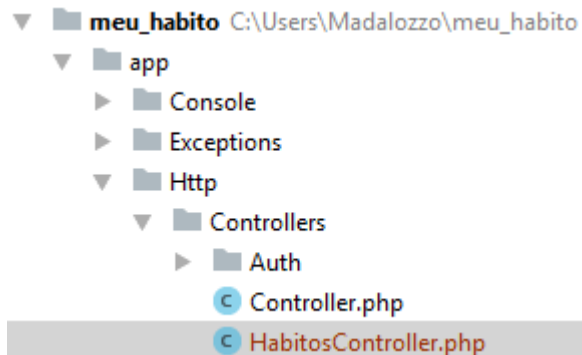
# Controlador

- Muitas coisas no Laravel são feitas automáticas
  - A criação do controller é uma delas
- Para criarmos o controller vamos utilizar o Artisan
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de criação do controlador
    - `php artisan make:controller HabitrosController`

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan make:controller HabitrosController
Controller created successfully.
```

# Controlador

- Analisando a árvore de diretórios, podemos notar que os controladores são criados dentro de app/Http/Controllers
- Caso o comando não funcione ou demore muito, pode-se criar “a mão” o arquivo nesta pasta



```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Http\Controllers\Controller;
7
8 class HabitosController extends Controller
9 {
10     //
11 }
```

# Controlador

HabitosController.php

- Vamos programar o método index, retornando um HTML

```
8  class HabitosController extends Controller
9  {
10     public function index() {
11         return "Olá!".
12             "<br>" .
13             "Sou o Index do Controlador de Hábitos";
14     }
15 }
```

← → ⓘ | 127.0.0.1:8000/habitos

Olá!

Sou o Index do Controlador de Hábitos!

## ***Etapas 5***

Criando a camada *View*



# Camada View

- Para que não misturemos o HTML ao controller, criamos a camada View
  - Desta maneira, o controlador irá solicitar a renderização de uma visão, e não o retorno de um HTML
- Primeiro, vamos alterar o controlador para solicitar a renderização de uma view

# Camada View

HabitosController.php

- Controller irá retornar a renderização de uma view

```
8  class HabitosController extends Controller
9  {
10     public function index() {
11         $nome = "Guilherme Madalozzo";
12         return view( view: 'habitos', ['nome'=>$nome] );
13     }
14 }
```

- Aqui estamos solicitando a renderização da view “habitos” passando o argumento ‘nome’ como parâmetro

# Camada View

- Para manipulação e renderização da camada de visão, o Laravel contém um recurso chamado **Blade**
- O Blade é um sistema de template simples e poderoso
  - Nele você pode utilizar todos os recursos que ele fornece, ou até mesmo programar normalmente em PHP
  - É um facilitador de desenvolvimento front-end

# Camada View

habitos.blade.php

- Vamos criar, dentro de *resources/views* o `habitos.blade.php`

```
1 <html>
2 <head>
3     <title>Hábitos</title>
4 </head>
5 <body>
6     <h1>Olá {{ $nome }}</h1>
7     Esta view é para apresentar os hábitos!
8 </body>
9 </html>
```

- Note que usamos o parâmetro **\$nome**, vindo do controlador
- O uso da tag `{{ ... }}` é específico do Blade, facilitando

## ***Etapas 6***

Entendendo a camada *Model*

# Camada Model

- Quando falamos na arquitetura MVC, falamos em separação de responsabilidades em nossa aplicação
- Já vimos o “C”ontroller e a “V”iew
- Agora, temos que responsabilizar o “M”odel

# Camada Model

- A camada Model é a camada responsável por fazer qualquer consumo e transação de dados na aplicação
  - Logo, quando falamos em Model, falamos em DADOS!
- Quando falamos em Model, estamos falando de diversas fontes de dados
  - Arquivos Json
  - Banco de dados
  - WebServices
  - Arquivos de texto
  - Etc

# Camada Model

- Assim como o Controller, criar um Model é simples
- Para criarmos o model vamos utilizar o Artisan
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de criação do model
    - `php artisan make:model Habito -m`

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan make:model Habito -m
Model created successfully.
Created Migration: 2017_05_04_183744_create_habitos_table
```



# Camada Model

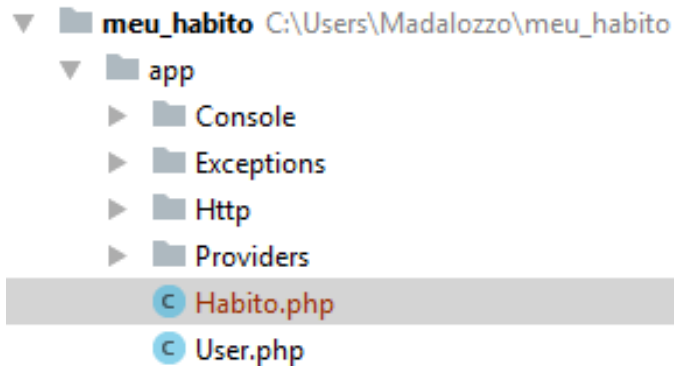
- Analisando as mensagens retornadas pelo comando executado pelo Artisan notamos que dois arquivos foram criados

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan make:model Habito -m
Model created successfully.
Created Migration: 2017_05_04_183744_create_habitos_table
```

- O model
- E a *Migration* (parâmetro *-m*)
  - São arquivos para configuração de banco de dados
  - Facilita a criação e alteração de campos em tabelas

# Camada Model

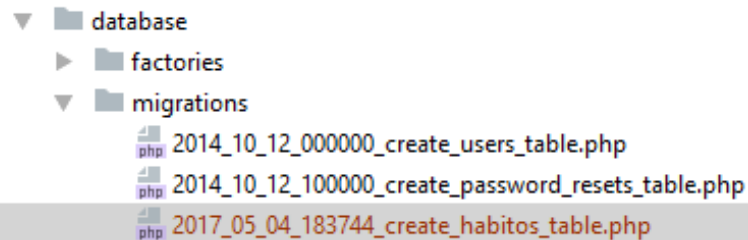
- Analisando a árvore de diretórios, podemos notar que os models são criados dentro de app/
- Caso o comando não funcione ou demore muito, pode-se criar “a mão” o arquivo nesta pasta



```
1  <?php
2
3  namespace App;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Habito extends Model
8  {
9      //
10 }
```

# Camada Model

- Analisando a árvore de diretórios, podemos notar que as migrations são criados dentro de database/migrations
- As migrations, em particular, devem ser geradas pelo comando
- O comando configura o autoload para carregamento



- A *migration* contém dois métodos principais (**up** e **down**)

# Camada Model

- Migration é um recurso do Laravel
  - Objetivo é gerenciar as mudanças estruturais do banco de dados da aplicação
  - Para cada tabela, coluna ou índice criados em nosso banco de dados podemos usar a migration para fazer essa operação automática
- Podemos notar que a migration contém dois métodos
  - **Up** → executado sempre que executamos a migration
  - **Down** → executado sempre que executamos um rollback da migration

# Camada Model

- Note que o método Up está sendo responsável por uma tabela chamada “habitos”
- Por padrão, o Laravel cria a migration com duas colunas
  - ID → usando tipo auto\_incremento
  - Atributo timestamps → cria duas tabelas de data/hora no banco
    - Created\_at
    - Updated\_at

# Camada Model

- Seguindo o desenvolvimento de nossa aplicação, vamos adicionar os campos necessário para os hábitos
  - Nome → texto com tamanho 20
  - Descrição → texto com tamanho 50
  - Tipo de hábitos → texto tamanho 10
  - Dia de início de controle → campo de data
  - Objetivo → numérico
- Para conhecer os tipos de dados de migrations no Laravel
  - Acessar “Available Column Types” em <https://laravel.com/docs/5.4/migrations>

# Camada Model

2017\_05\_04\_183744\_create\_habitos\_table.php

- Vamos adicionar os novos campos ao método Up()

```
14 public function up()  
15 {  
16     Schema::create('habitos', function (Blueprint $table) {  
17         $table->increments( column: 'id');  
18         $table->string( column: 'nome', length: 20);  
19         $table->string( column: 'descricao', length: 50);  
20         $table->string( column: 'tp_habito', length: 10);  
21         $table->date( column: 'dt_inicio_ctrl');  
22         $table->integer( column: 'objetivo');  
23         $table->timestamps();  
24     });  
25 }
```

# Camada Model

- Para podermos executar a migration criada, primeiro devemos criar o banco de dados “meuhabito”
- Para isso, acesse o gerenciador de BD (MySQLWorkBench ou phpMyAdmin) e crie o banco



# Camada Model

- Com o banco de dados criado
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de execução de migrations
    - `php artisan migrate`

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrating: 2017_05_04_183744_create_habitos_table
Migrated: 2017_05_04_183744_create_habitos_table
```

# Camada Model

- Podemos notar que as 3 migrations foram executadas
  - Users
  - Password resets
  - Habitos

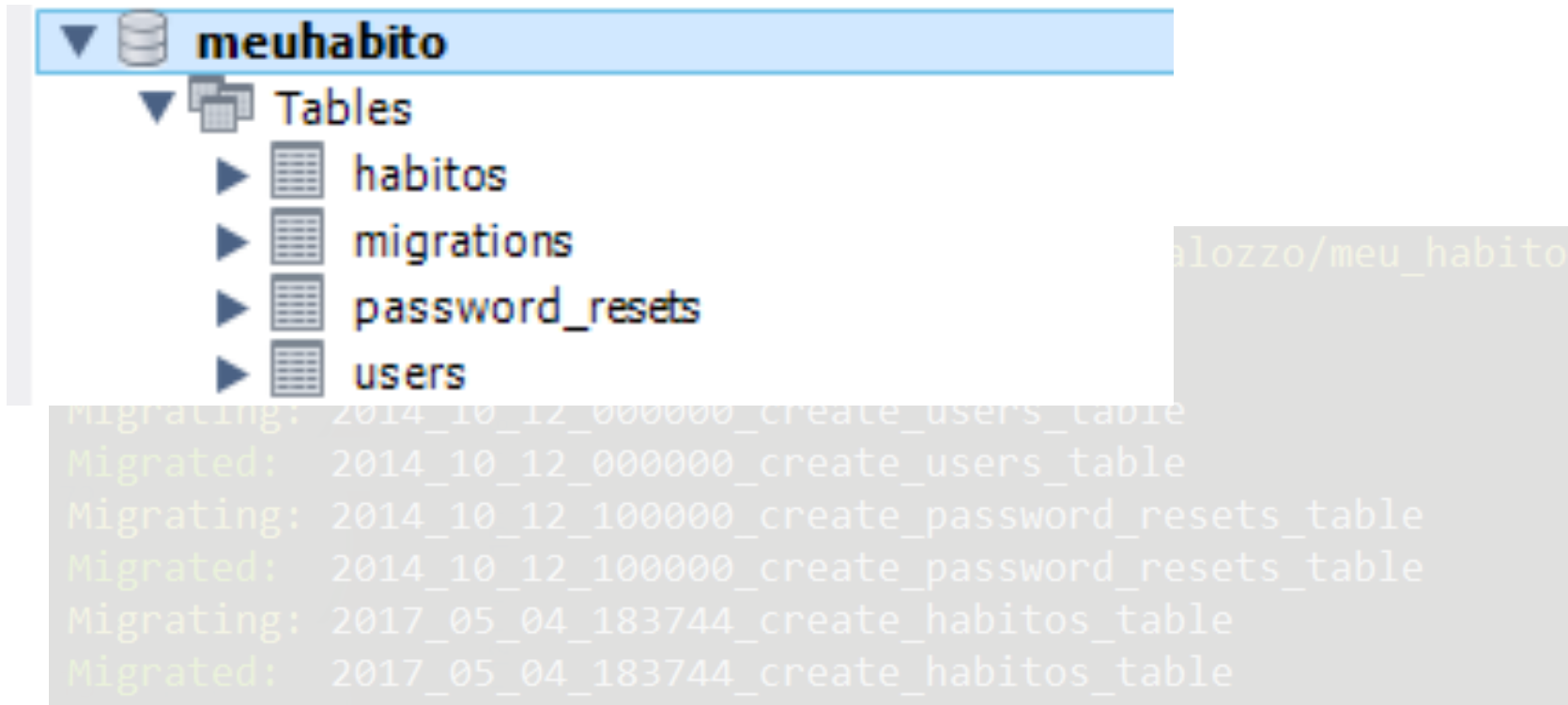
```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrating: 2017_05_04_183744_create_habitos_table
Migrated: 2017_05_04_183744_create_habitos_table
```

# Camada Model

- PARAMOS AQUI!!!!!!

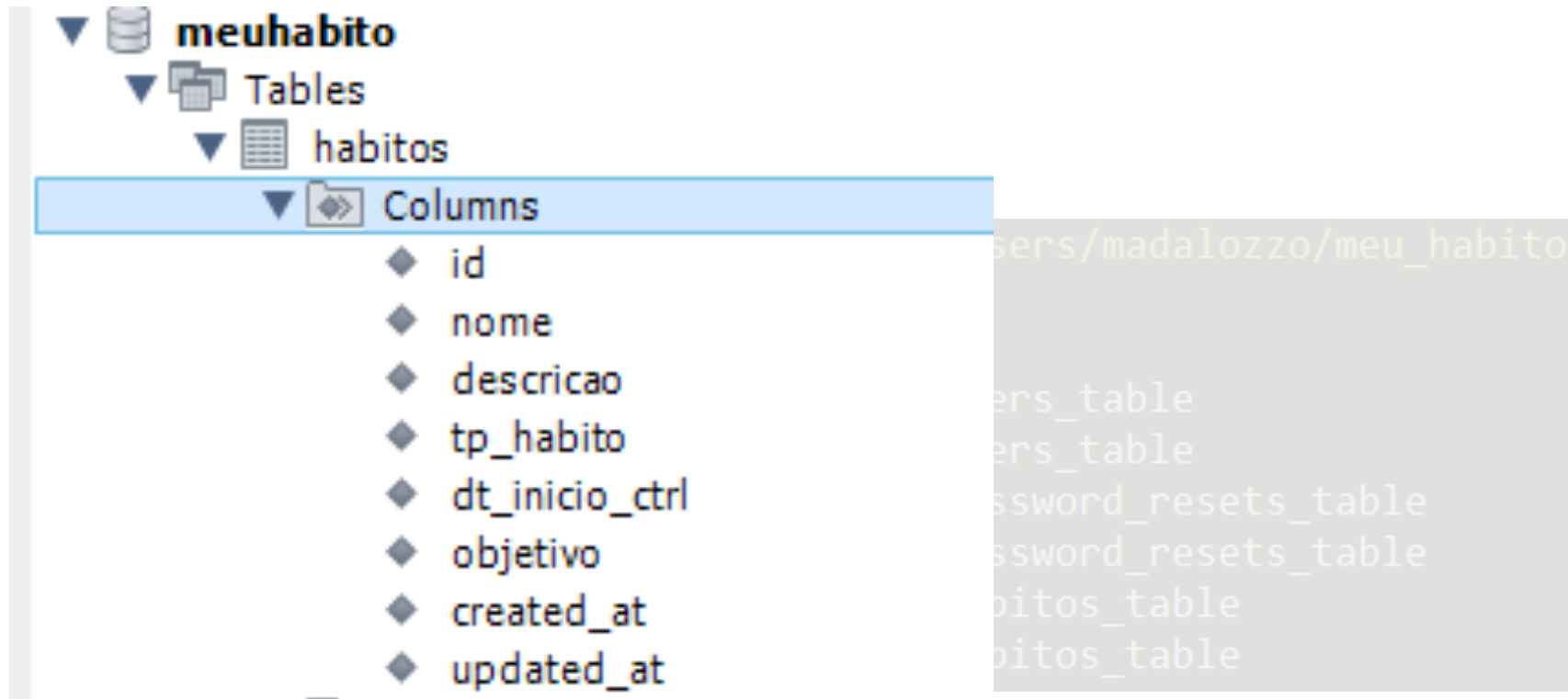
# Camada Model

- Podemos notar que as tabelas foram criadas



# Camada Model

- Podemos notar as colunas da tabela “habitos” criadas



## ***Etapa 7***

Brincando com os dados através do recurso *Tinker*

# Camada Model

- A classe model, criada, não contém métodos programados
- Porém, ele possui uma infinidade de recursos para podermos manipular os dados da tabela
  - Devido ao uso da biblioteca Eloquent
- O Laravel contém um recurso, console iterativo, chamado **Tinker**

# Camada Model

- Com o Tinker podemos “brincar” com nossos dados
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de execução do Tinker
    - `php artisan tinker`

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> |
```

- Agora, podemos criar, procurar, editar e excluir registros de dentro das tabelas do banco de nossa aplicação



# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Abrir o tiker

# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Informamos ao Tinker que vamos usar o model “Habito”

# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Criamos o objeto `$hab` que é uma instância do `Habito`

# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Atribuimos uma valor ao atributo  
nome

# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Atribuímos uma valor ao atributo descricao

# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Atribuímos uma valor ao atributo  
tp\_habito

# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> $hab = new Habito();
=> App\Habito {#663}
>>> $hab->nome = "Correr";
=> "Correr"
>>> $hab->descricao = "Correr 45min ao dia";
=> "Correr 45min ao dia"
>>> $hab->tp_habito = "Bom";
=> "Bom"
>>> $hab->objetivo = 1;
=> 1
```

Atribuimos uma valor ao atributo objetivo



# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MI >>> $hab->save();  
$ php artisan tinker => true  
Psy Shell v0.8.3 (PHP 7.1.1  
Unable to check for updates  
>>> use App\Habito;  
=> null  
>>> $hab = new Habito();  
=> App\Habito {#663}  
>>> $hab->nome = "Correr";  
=> "Correr"  
>>> $hab->descricao = "Correr 45min ao dia";  
=> "Correr 45min ao dia"  
>>> $hab->tp_habito = "Bom";  
=> "Bom"  
>>> $hab->objetivo = 1;  
=> 1
```

Para salvar os dados informados no objeto \$hab, invocamos o método save()

Note que o método save() não foi programado

Porém, o Laravel utiliza a biblioteca Eloquent, que contém os principais métodos para CRUD já programados



# Camada Model

- Usaremos o Tinker para inserir alguns registros na tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MI >>> $hab->save();  
=> true  
$ php artisan tinker  
Psy Shell v0.8.3 (PHP 7.1.1  
Unable to check for updates  
>>> use App\Habito;  
=> null  
>>> $hab = new Habito();  
=> null  
>>> $hab->dt_inicio_ctrl = "2017-05-05 00:00:00"  
=> null  
>>> $hab->objetivo = 1;  
=> 1
```

Para salvar os dados informados no objeto \$hab, invocamos o método save()

Note que o método save() não foi programado  
Porém, o Laravel utiliza a biblioteca Eloquent, que contém os principais métodos para CRUD já programados

# Camada Model

- Usaremos o Tinker consultar os registros da tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 /c/users/madalozzo/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
Unable to check for updates
>>> use App\Habito;
=> null
>>> Habito::all();
=> Illuminate\Database\Eloquent\Collection {#672
  all: [
    App\Habito {#673
      id: 1,
      nome: "Correr",
      descricao: "Correr 45min ao dia",
      tp_habito: "Bom",
      dt_inicio_ctrl: null,
      objetivo: 1,
      created_at: "2017-05-04 21:19:47",
      updated_at: "2017-05-04 21:19:47",
    },
  ],
}
```

Usamos o método `all()`  
→ `Habito::all()` ;

# Camada Model

- Adicione três novos hábitos via Tinker

Obs: para inserir um valor a um campo datetime temos usar a seguinte string

“AAAA-MM-DD HH:MM:SS”

“2017-05-04 21:00:00” → Dia 05/04/2017 às 21hr

# Camada Model

- Usaremos o Tinker alterar um registro da tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
New version is available (current: v0.8.3, latest: v0.8.5)
>>> use App\Habito;
=> null
>>> $hab = Habito::find(1);
=> App\Habito {#676
    id: 1,
    nome: "Correr",
    descricao: "Correr 45min ao dia",
    tp_habito: "Bom",
    dt_inicio_ctrl: null,
    objetivo: 1,
    created_at: "2017-05-04 21:19:47",
    updated_at: "2017-05-04 21:19:47",
}
>>> $hab->nome = "Corrida";
=> "Corrida"
>>> $hab->save();
=> true
```

Abrimos o Tinker e informamos o uso do model “Habito”

# Camada Model

- Usaremos o Tinker alterar um registro da tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
New version is available (current: v0.8.3, latest: v0.8.5)
>>> use App\Habito;
=> null
>>> $hab = Habito::find(1);
=> App\Habito {#676
    id: 1,
    nome: "Correr",
    descricao: "Correr 45min ao dia",
    tp_habito: "Bom",
    dt_inicio_ctrl: null,
    objetivo: 1,
    created_at: "2017-05-04 21:19:47",
    updated_at: "2017-05-04 21:19:47",
}
>>> $hab->nome = "Corrida";
=> "Corrida"
>>> $hab->save();
=> true
```

Usamos o método “find(*id*)” para localizar um registro com o ID informado no parâmetro

# Camada Model

- Usaremos o Tinker alterar um registro da tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
New version is available (current: v0.8.3, latest: v0.8.5)
>>> use App\Habito;
=> null
>>> $hab = Habito::find(1);
=> App\Habito {#676
    id: 1,
    nome: "Correr",
    descricao: "Correr 45min ao dia",
    tp_habito: "Bom",
    dt_inicio_ctrl: null,
    objetivo: 1,
    created_at: "2017-05-04 21:19:47",
    updated_at: "2017-05-04 21:19:47",
}
>>> $hab->nome = "Corrida";
=> "Corrida"
>>> $hab->save();
=> true
```

Com o uso do find o Tinker irá apresentar os dados do registro buscado

# Camada Model

- Usaremos o Tinker alterar um registro da tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
New version is available (current: v0.8.3, latest: v0.8.5)
>>> use App\Habito;
=> null
>>> $hab = Habito::find(1);
=> App\Habito {#676
    id: 1,
    nome: "Correr",
    descricao: "Correr 45min ao dia",
    tp_habito: "Bom",
    dt_inicio_ctrl: null,
    objetivo: 1,
    created_at: "2017-05-04 21:19:47",
    updated_at: "2017-05-04 21:19:47",
}
>>> $hab->nome = "Corrida";
=> "Corrida"
>>> $hab->save();
=> true
```

Alteramos o valor do atributo nome de “Correr” para “Corrida”

# Camada Model

- Usaremos o Tinker alterar um registro da tabela “Habitos” usando o *model* programado

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ php artisan tinker
Psy Shell v0.8.3 (PHP 7.1.1 - cli) by Justin Hileman
New version is available (current: v0.8.3, latest: v0.8.5)
>>> use App\Habito;
=> null
>>> $hab = Habito::find(1);
=> App\Habito {#676
    id: 1,
    nome: "Correr",
    descricao: "Correr 45min ao dia",
    tp_habito: "Bom",
    dt_inicio_ctrl: null,
    objetivo: 1,
    created_at: "2017-05-04 21:19:47",
    updated_at: "2017-05-04 21:19:47",
}
>>> $hab->nome = "Corrida";
=> "Corrida"
>>> $hab->save();
=> true
```

Depois salvamos o objeto com o uso do método `save()`



# Camada Model

- Usaremos o Tinker alterar um registro da tabela “Habitos” usando o *model* programado

```
>>> $hab = Habito::find(1);  
=> App\Habito {#677  
  id: 1,  
  nome: "Corrida",  
  descricao: "Correr 45min ao dia",  
  tp_habito: "Bom",  
  dt_inicio_ctrl: null,  
  objetivo: 1,  
  created_at: "2017-05-04 21:19:47",  
  updated_at: "2017-05-18 00:19:56",  
}
```

Para mostrar a atualização do objeto no banco de dados invocamos novamente o método `find(id)`

## ***Etapa 8***

Listando os Hábitos cadastrados

# Camada Model

- A classe model, criada, não contém métodos programados
- Porém, ele possui uma infinidade de recursos para podermos manipular os dados da tabela
  - Devido ao uso da biblioteca Eloquent
- NÃO PROGRAMAMOS NENHUM MÉTODO NA CLASSE
- ISSO É ELOQUENT

# Controller

- Agora, com dados cadastrados no BD, vamos listar em tela para o usuário visualizar
- Para isso, vamos alterar o HabitotsController.php passando os dados do banco como parâmetro no método index

# Controller

- Vamos alterar o HabitosController.php passando os dados do banco como parâmetro no método index

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Habitto;
6  use Illuminate\Http\Request;
7  use App\Http\Controllers\Controller;
8
9  class HabitosController extends Controller
10 {
11     public function index() {
12         $shabitos = Habitto::all();
13         return view('view: 'habitos', ['habitos'=>$shabitos]);
14     }
15 }
```

# Controller

- Vamos alterar o HabitosController.php passando os dados do banco como parâmetro no método index

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use App\Habito;
6  use Illuminate\Http\Request;
7  use App\Http\Controllers\Controller;
8
9  class HabitosController extends Controller
10 {
11     public function index() {
12         $shabitos = Habito::all();
13         return view('habitos', ['habitos'=>$shabitos]);
14     }
15 }
```

1- Adicionamos o uso do model “Habito”

# Controller

- Vamos alterar o HabitosController.php passando os dados do banco como parâmetro no método index

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Habito;
6 use Illuminate\Http\Request;
7 use App\Http\Controllers\Controller;
8
9 class HabitosController extends Controller
10 {
11     public function index() {
12         $shabitos = Habito::all();
13         return view('habitos', ['habitos'=>$shabitos]);
14     }
15 }
```

2- Usamos a função all() do Habitos (EloquentORM) para armazenarmos os registros na variável \$shabitos

3- Passamos através de parâmetro os registros para a rota “/hábitos”

# View

- Agora, na View (habitos.blade.php) vamos apresentar os registros passados por parâmetro

```
1 <html>
2 <head>
3     <title>Hábitos</title>
4 </head>
5 <body>
6     <h1>Hábitos</h1>
7     <ul>
8         @foreach($habitos as $hab)
9             <li>{{ $hab->nome }}<br>
10                {{ $hab->descricao }}
11            </li>
12            <br>
13        @endforeach
14    </ul>
15 </body>
16 </html>
```



## ***Etapa 9***

Desenvolvendo um template padrão

# View

- Nossa tela de listagem está funcional
- Como já visto, o blade é uma ferramenta poderosa
- Com o Blade podemos montar uma visão padronizada e apenas importa-la em nossas visões de dados
- Desta forma, baixe o arquivo `app.blade.php` da intranet e coloque na pasta *resource/views*

# View

- Podemos notar que no final do arquivo temos

```
59     </nav>
60
61     @yield('content')
62
63     </body>
```

- A TAG @yield reserva uma seção das páginas herdadas para ser substituída pela nova seção... Vamos ver!

# View

- Vamos herdar o template app no habits.blade.php

```
1 @extends ( 'app' )
```

- Agora podemos usar a seção 'content' reservada no template através do @yield

# View

- Agora podemos usar a seção 'content' reservada no template através do @yield

```
1 @extends('app')
2
3 @section('content')
4     <div class="container">
5         <h1>Hábitos</h1>
6         <ul>
7             @foreach($habitots as $hab)
8                 <li>{{ $hab->nome }}<br>
9                     {{ $hab->descricao }}
10                </li>
11                <br>
12            @endforeach
13        </ul>
14    </div>
15 @endsection
```

# View

- Agora podemos usar a seção 'content' reservada no template através do @yield

```
1  @extends('app')
2
3  @section('content')
4      <div class="container">
5          <h1>Hábitos</h1>
6          <ul>
7              @foreach($habitots as $hab)
8                  <li>{{ $hab->nome }}<br>
9                      {{ $hab->descricao }}
10                 </li>
11                 <br>
12             @endforeach
13         </ul>
14     </div>
15 @endsection
```

Com o extends do template "app" podemos abrir a seção 'content'

Na linha 3 abrimos a seção e na linha 15 fechamos a mesma

Com isso, tudo que estiver dentro da seção 'content' será aplicado ao espaço reservado pela TAG @yield

## ***Etapa 10***

Listando os Hábitos cadastrados em um tabela

# View

- Vamos aprimorar nossa listagem mostrando nossos registros em uma tabela

## Hábitos

Nome	Descrição	Tipo	Ação
Corrida	Correr 45min ao dia	Bom	
Tomar refrigerante	Refrigerante não é bom	Ruim	



# View

- Vamos aprimorar nossa listagem mostrando nossos registros em uma tabela → criamos primeiro o cabeçalho

```
1  @extends('app')
2
3  @section('content')
4      <div class="container">
5          <h1>Hábitos</h1>
6
7          <table class="table table-striped table-bordered table-hover">
8              <thead>
9                  <tr>
10                     <th>Nome</th>
11                     <th>Descrição</th>
12                     <th>Tipo</th>
13                     <th>Ação</th>
14                 </tr>
15             </thead>
```

# View

- Vamos aprimorar nossa listagem mostrando nossos registros em uma tabela → criamos a listagem de dados

```
16
17     <tbody>
18     @foreach($habitos as $hab)
19     <tr>
20         <td>{{ $hab->nome }}</td>
21         <td>{{ $hab->descricao }}</td>
22         <td>{{ $hab->tp_habito }}</td>
23     </tr>
24     @endforeach
25 </tbody>
26 </table>
27 </div>
28 @endsection
```

## ***Etapa 11***

Tela de cadastro de novos hábitos

# Cadastro

- Nossa tela de listagem está funcional e padronizada
- Agora devemos criar uma tela de cadastro de novos hábitos
- Para isso, temos que criar um método no controller para renderizar a tela de criação

# Cadastro

- No HabitotsController.php vamos criar o método “create”

```
9  class HabitotsController extends Controller
10 {
11     public function index() {
12         $habitots = Habito::all();
13         return view( view: 'habitots.index', ['habitots'=>$habitots]);
14     }
15
16     public function create() {
17         return view( view: 'habitots.create');
18     }
19 }
```

# Cadastro

- No HabitotsController.php vamos criar o método “create”

```
9  class HabitotsController extends Controller
10 {
11     public function index() {
12         $habitots = Habito::all();
13         return view( view: 'habitots.index', ['habitots'=>$habitots]);
14     }
15
16     public function create() {
17         return view( view: 'habitots.create' );
18     }
19 }
```

Criamos o método create para renderizar a view habitots.create

# Cadastro

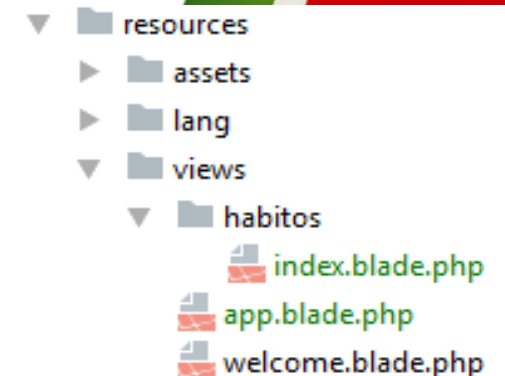
- No HabitotsController.php vamos criar o método “create”

```
9  class HabitotsController extends Controller
10 {
11     public function index() {
12         $habitots = Habito::all();
13         return view( view: 'habitots.index', ['habitots'=>$habitots] );
14     }
15
16     public function create() {
17         return view( view: 'habitots.create' );
18     }
19 }
```

Também, alteramos o retorno da view do método index para renderizar a view index do habitots

# Cadastro

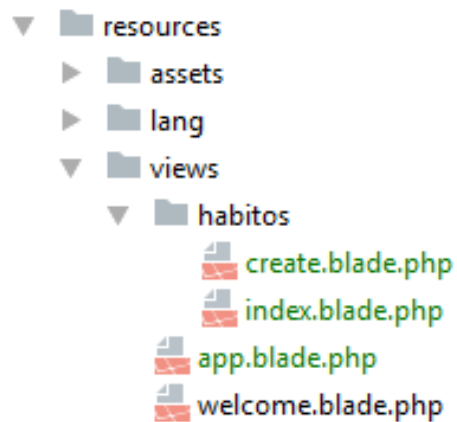
- Para facilitar a organização de nosso código, criaremos uma pasta chamada `habitos` dentro da pasta `resources/views`
- Assim, todas as views referentes ao nosso CRUD de hábitos estará nela
- Primeiro, renomeie o arquivo `habitos.blade.php` para `index.blade.php` e coloque dentro de `resources/views/habitos`





# Cadastro

- Vamos criar a tela de cadastro de novo hábito
  - Crie o arquivo `create.blade.php`



```
1  @extends('app') @extends('adminlte::default')
2
3  @section('content')
4      <div class="container">
5          <h1>Novo Hábito</h1>
6
7      </div>
8  @endsection
```

# Cadastro

- Criamos o método create no controller e a tela de inserção de dados
- Agora temos que criar a rota para que isso tudo seja acessado pelo usuário
  - Adicione a rota no `web.php`
  - Temos a rota `get /habitos` para a listagem
  - E agora temos a rota `/habitos/create` para adicionar registro

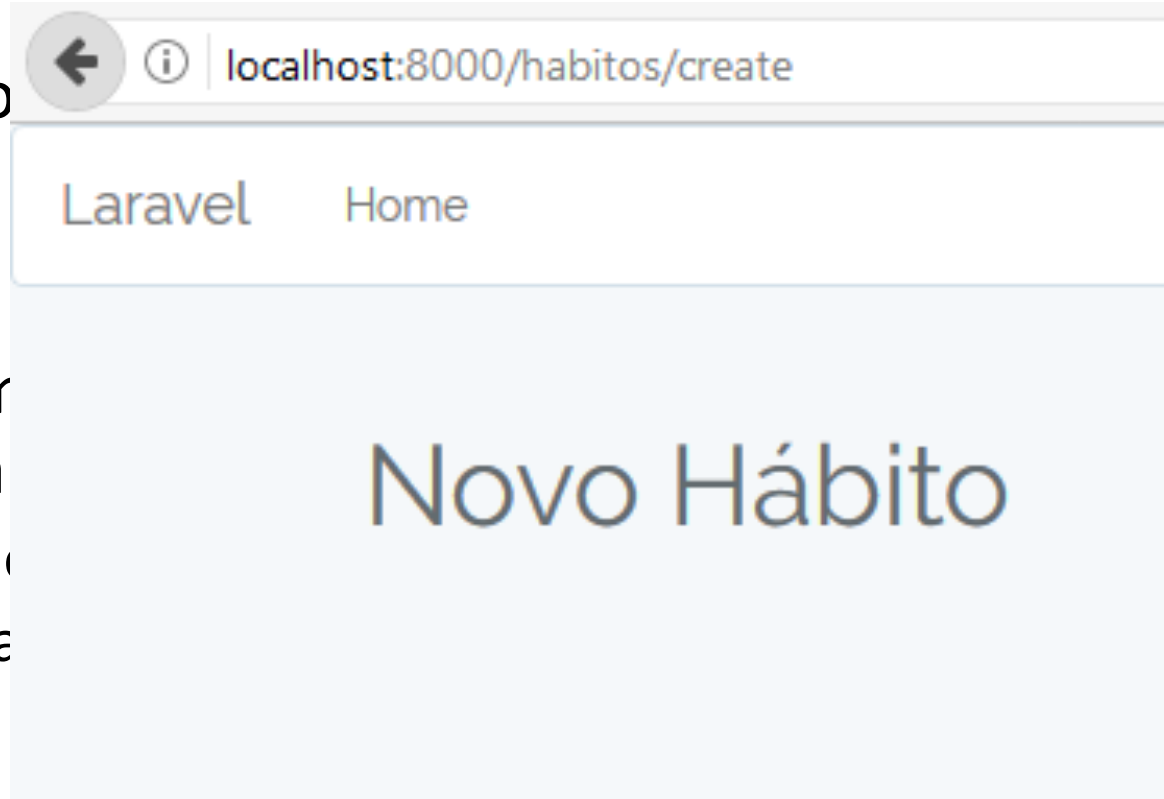
```
18 Route::get('habitos', 'HabitosController@index');  
19 Route::get('habitos/create', 'HabitosController@create');
```

# Cadastro

- Criamos o formulário e inserção de dados

- Agora tem o formulário pelo usuário

- Adicionamos o formulário
- Temos a rota para o formulário
- E agora



e inserção de

seja acessado

registro

18

```
Route::get('habitos/create', function() {
```

```
return view('habitos.create');
```

19

```
Route::get('habitos/create', 'HabitosController@create');
```

## ***Etapa 12***

Vamos trabalhar com formulários no Blade

# Form::

- Para trabalharmos com formulários no laravel vamos instalar o `laravelcollective/html`
- É um pacote que contém componentes projetados para o Laravel

# Form::

- Para trabalharmos com o laravelcollective/html execute

```
$ composer require laravelcollective/html
```

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ composer require laravelcollective/html
Using version ^5.4 for laravelcollective/html
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing laravelcollective/html (v5.4.4): Downloading (100%)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postUpdate
> php artisan optimize
Generating optimized class loader
The compiled services file has been removed.
```

# Form::

- Sempre que instalarmos pacotes devemos registrá-lo como servidor de pacotes para que o laravel possa encontrar os componentes
- Para isso, vamos adicionar ao seguinte linha no final do array providers do arquivo config/app.php

```
138 'providers' => [  
139  
140     /*  
141     * Laravel Framework Service Providers...  
142     */  
143     Illuminate\View\ViewServiceProvider::class,  
164     Collective\Html\HtmlServiceProvider::class,  
165  
166
```

# Form::

- Sempre que instalarmos pacotes devemos registrá-lo como servidor de pacotes para que o laravel possa encontrar os componentes
- Para isso, vamos adicionar ao seguinte linha no final do array aliases do arquivo `config/app.php`

```
194 'aliases' => [  
195  
196  
228 'Form' => Collective\Html\FormFacade::class,  
229 'Form' => Collective\Html\FormFacade::class,  
230 'Html' => Collective\Html\HtmlFacade::class,
```



# Form::

- Com essa instalação e configuração podemos começar a utilizar os pacotes nos formulários

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

```
1  @extends('app')
2
3  @section('content')
4      <div class="container">
5          <h1>Novo Hábito</h1>
6
7          {!! Form::open() !!}
8
9          {!! Form::close() !!}
10
11      </div>
12  @endsection
```

Primeiro informamos ao Laravel que estamos “abrindo” um formulário nesta view

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Vamos adicionar o componente “Nome”  
no formulário

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

```
7      {!! Form::open() !!}
8
9      <div class="form-group">
10         {!! Form::label('nome', 'Nome:') !!}
11         {!! Form::text('nome', null, ['class'=>'form-control']) !!}
12      </div>
13
14      {!! Form::close() !!}
```

Vamos adicionar o componente “Nome”  
no formulário

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

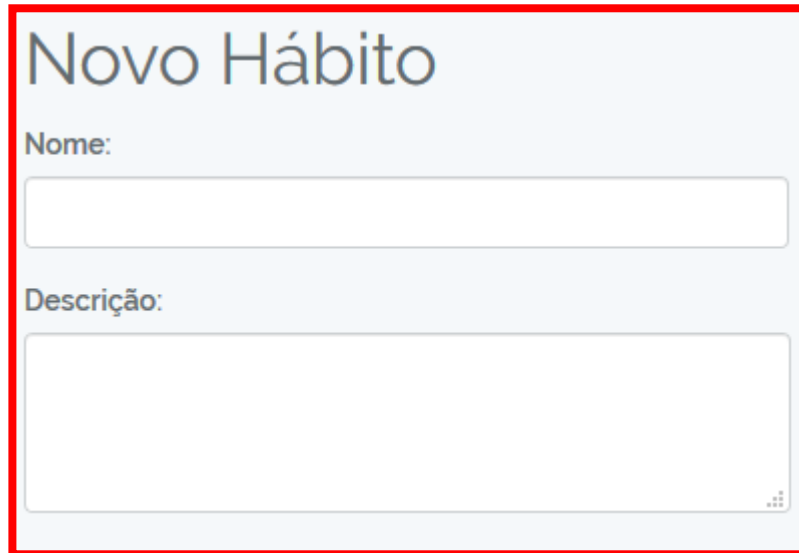
```
7      {!! Form::open() !!}
8
9      <div class="form-group">
10         {!! Form::label('nome', 'Nome:') !!}
11         {!! Form::text('nome', null, ['class'=>'form-control']) !!}
12      </div>
13
14      {!! Form::close() !!}
```

Podemos notar que criamos um “label” e um “text” através do uso do componente instalado

Form::

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`



The image shows a web form titled "Novo Hábito" (New Habit). It has a light blue header with the title. Below the header, there are two input fields. The first field is labeled "Nome:" (Name) and is a single-line text input. The second field is labeled "Descrição:" (Description) and is a multi-line text area. The form is enclosed in a red border.

Vamos adicionar o componente  
“Descrição” no formulário

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

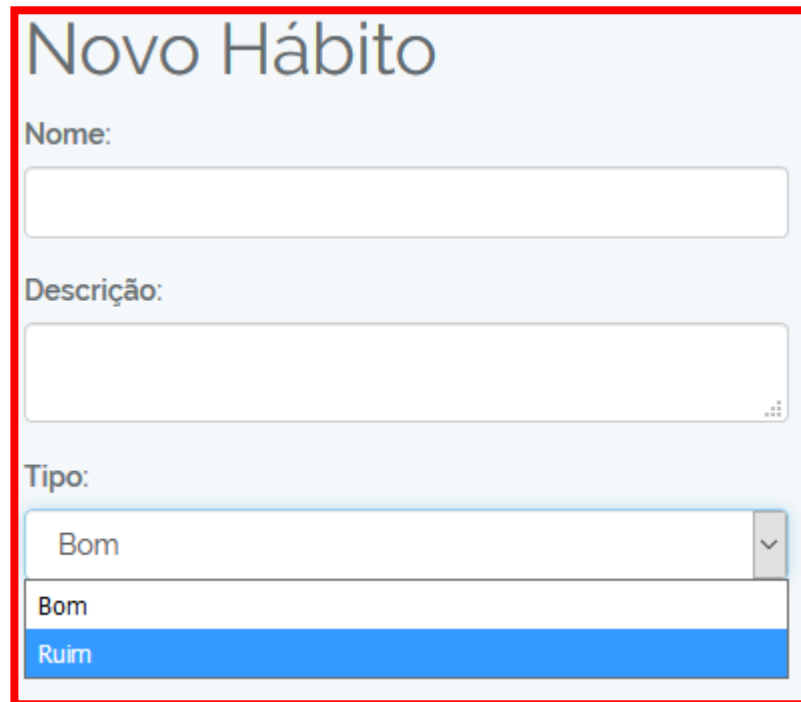
Descrição:

Vamos adicionar o componente  
“Descrição” no formulário

```
12 </div>
13
14 <div class="form-group">
15     {!! Form::label('descricao', 'Descrição:') !!}
16     {!! Form::textarea('descricao', null, ['class'=>'form-control']) !!}
17 </div>
```

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`



The screenshot shows a web form titled "Novo Hábito" (New Habit). It contains three input fields: "Nome:" (Name), "Descrição:" (Description), and "Tipo:" (Type). The "Tipo:" field is a dropdown menu with a blue border and a downward arrow icon. The dropdown is open, showing two options: "Bom" (Good) and "Ruim" (Bad). The "Ruim" option is highlighted with a blue background. The form is enclosed in a red rectangular border.

Vamos adicionar o componente “Tipo”  
no formulário, como um DropDown



# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Vamos adicionar o componente “Tipo”  
no formulário, como um DropDown

```
19 <div class="form-group">
20     {!! Form::label('tp_habito', 'Tipo:') !!}
21     {!! Form::select('tp_habito',
22         array('B' => 'Bom', 'R' => 'Ruim'),
23         'M',
24         ['class'=>'form-control']) !!}
25 </div>
```

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Tipo:

Bom

Objetivo:

2

Vamos adicionar o componente  
“Objetivo” no formulário, como um  
campo numérico

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Vamos adicionar o componente “Objetivo” no formulário, como um campo numérico

```
27 <div class="form-group">
28     {!! Form::label('objetivo', 'Objetivo:') !!}
29     {!! Form::number('objetivo', null, ['class'=>'form-control']) !!}
30 </div>
```

2

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Tipo:

Bom

Objetivo:

2

Data:

2017-05-18 00:00:00

Vamos adicionar o componente “Data”  
no formulário, como um campo *Date*

OBS  
*Vamos usar um componente específico  
para data, então, por ora, vamos deixar  
esta data fixa*

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Vamos adicionar o componente “Data”  
no formulário, como um campo *Date*

```
32 <div class="form-group">
33     {!! Form::label('dt_inicio_ctrl', 'Data:') !!}
34     {!! Form::date('dt_inicio_ctrl',
35         '2017-05-18 00:00:00',
36         ['class'=>'form-control']) !!}
37 </div>
```

2017-05-18 00:00:00

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Tipo:

Bom

Objetivo:

1

Data:

2017-05-18 00:00:00

Criar hábito

Vamos adicionar um botão para poder  
submeter os dados para serem salvos no  
BD

# Form::

- Vamos alterar o `resource/views/habitos/create.blade.php`

## Novo Hábito

Nome:

Descrição:

Tipo:

Objetivo:

2017-03-10 00:00:00

Criar hábito

Vamos adicionar um botão para poder submeter os dados para serem salvos no BD

```
39 <div class="form-group">  
40     {!! Form::submit('Criar hábito', ['class'=>'btn btn-primary']) !!}  
41 </div>
```

# Form::

- Se tentarmos criar um novo hábito veremos um erro
  - Todo formulário deve ter uma ação, e não configuramos a ação ao formulário ainda
- Para isso devemos criar uma nova *action* que será responsável por receber a requisição (*request*) do *form*
  - Essa *action* deve persistir os dados no banco usando o model *Habito.php*



# Form::

- Antes de criarmos a nova action, precisamos criar uma nova rota no `routes/web.php`

```
18 Route::get('habitos', 'HabitosController@index');  
19 Route::get('habitos/create', 'HabitosController@create');  
20 Route::post('habitos/store', 'HabitosController@store');
```

- Como podemos ver, estamos invocando um método “store” do controlador

# Form::

- Agora, no HabitosController.php vamos criar este novo método

```
20 public function store(Request $request) {  
21     $novo_habito = $request->all();  
22     Habito::create($novo_habito);  
23  
24     return redirect(to: 'habitos');  
25 }
```

# Form::

- Agora, no HabitosController.php vamos criar este novo método

```
20 public function store(Request $request) {  
21     $novo_habito = $request->all();  
22     Habito::create($novo_habito);  
23  
24     return redirect(to: 'habitos');  
25 }
```

O parâmetro Request contém todas as informações recebidas via form

# Form::

- Agora, no HabitosController.php vamos criar este novo método

```
20 public function store(Request $request) {  
21     $novo_habito = $request->all();  
22     Habito::create($novo_habito);  
23  
24     return redirect(to: 'habitos');  
25 }
```

Aqui atribuídos todas as informações do request em uma variável que será passada como parâmetro para a inserção do registro no BD

# Form::

- Agora, no HabitosController.php vamos criar este novo método

```
20 public function store(Request $request) {  
21     $novo_habito = $request->all();  
22     Habito::create($novo_habito);  
23  
24     return redirect(to: 'habitos');  
25 }
```

Por fim, redirecionamos o usuário para a tela de listagem de hábitos

# Form::

- Temos a rota criada e o novo método implementado
  - Agora devemos definir a ação do formulário
  - Para isso, vamos alterar o `habitos/create.blade.php`

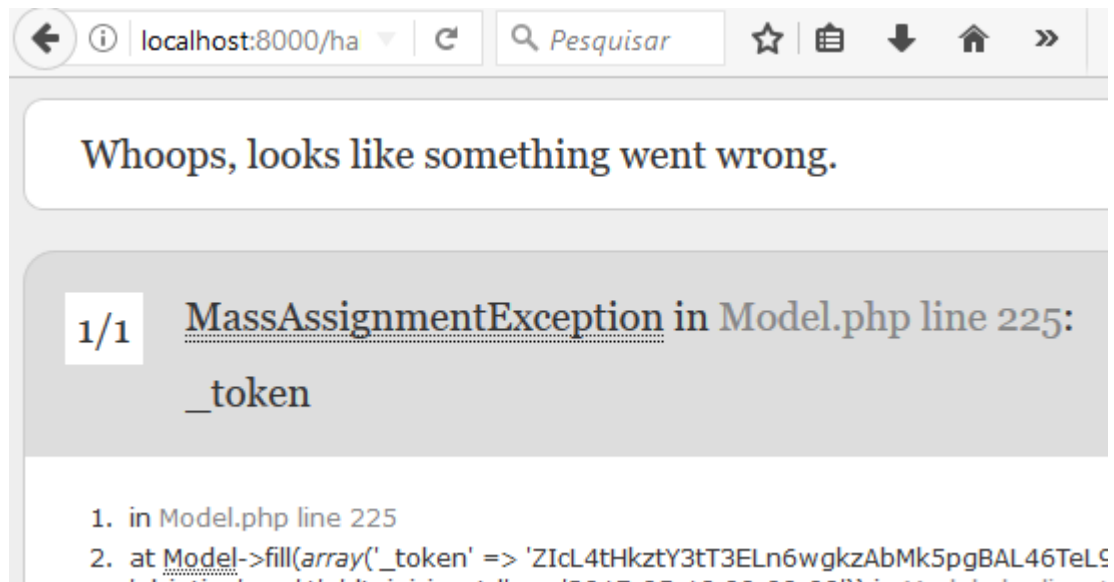
7

```
{!! Form::open(['url' => 'habitos/store']) !!}
```

- Nota-se que na abertura do form definimos que a ação do formulário será redirecionada para a rota `habitos/store`

# Form::

- Vamos cadastrar um novo hábito!!
- OPS, deu o seguinte erro!!!!!!



O Laravel trabalha com uma proteção nos Models contra inserção em massa de dados (segurança contra ataques)

# Form::

- Para resolver este problema, temos que informar ao nosso Model que ele pode aceitar a inserção de dados em massa para os campos apresentados em tela
  - Vamos trabalhar com o `app\Habito.php`
  - Adicionaremos um atributo chamado `fillable` (preenchível)

```
7  class Habito extends Model
8  {
9      protected $fillable = ['nome', 'descricao',
10                             'tp_habito', 'dt_inicio_ctrl', 'objetivo'];
11 }
```



# Form::

- Pronto!!
  - Agora podemos inserir novo hábito
  - Após a inserção, seremos redirecionados para a tela de listagem

## ***Etapa 13***

Validação dos dados a serem cadastrados

# Validação

- O Laravel nos possibilita validar, facilmente, as informações que estão sendo enviadas através do request
  - Utilizaremos o artisan para gerar automaticamente a classe de Request
    - Então, usando o terminal (cmd) dentro da pasta meu\_habito
    - Execute o comando para criação do request
- `php artisan make:request HabitoRequest`

```
Madalozzo@DESKTOP-7Q69GCM MINGW32 ~/meu_habito
$ php artisan make:request HabitoRequest
Request created successfully.
```

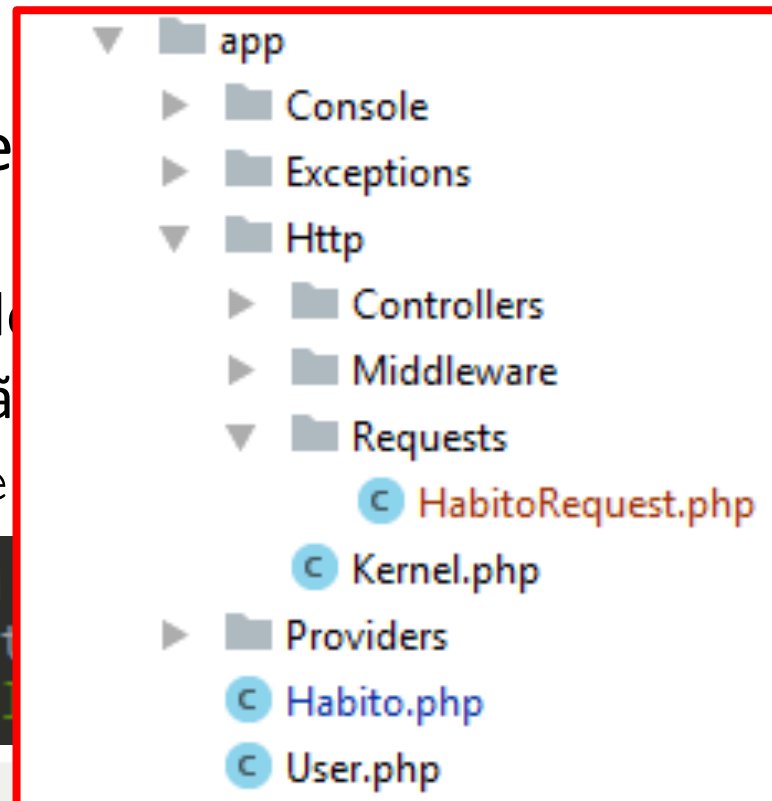
# Validação

- O Laravel nos possibilita validar, facilmente, as informações que estão sendo enviadas através do request

- Utilizaremos o artisan para gerar o Request

- Então, usando o terminal (cmd)
- Execute o comando para criação  
→ `php artisan make:request`

```
Madalozzo@DESKTOP-7Q69GCM  
$ php artisan make:request  
Request created successfully
```



# Validação

- Ao abrirmos a classe criada (**HabitoRequest.php**) podemos perceber que dois métodos foram criados

## **authorize()**

- Determina se temos autorização de realizar essa requisição
- Se retornar *true* quer dizer que a requisição pode ser executada
- Se retornar *false* a requisição será bloqueada
- Podemos definir a regra que quisermos nela
- Para o nosso caso, retornaremos *true* por padrão

```
14     public function authorize()  
15     {  
16         return true;  
17     }
```

# Validação

- Ao abrirmos a classe criada (**HabitoRequest.php**) podemos perceber que dois métodos foram criados

## **rules ()**

- Método responsável pelas regras de validação de campos
- O Laravel possui uma série e padrões possíveis para validação

<https://laravel.com/docs/5.0/validation#available-validation-rules>

# Validação

- Ao abrirmos a classe criada ([HabitoRequest.php](#)) podemos perceber que dois métodos foram criados

## **rules()**

- Método responsável pelas regras de validação de campos

```
24 public function rules()  
25 {  
26     return [  
27         'nome' => 'required|min:5',  
28         'descricao' => 'required',  
29         'tp_habito' => 'required',  
30         'dt_inicio_ctrl' => 'required',  
31         'objetivo' => 'required|numeric'  
32     ];  
33 }
```

# Validação

- Com as regras de request definidas, devemos injetar esta classe no método store do **HabitosController.php**

```
7 use App\Http\Controllers\Controller;  
8 use App\Http\Requests\Habitorequest;
```

```
21 public function store(Habitorequest $request) {  
22     $novo_habito = $request->all();  
23     Habito::create($novo_habito);  
24  
25     return redirect(to: 'habitos');  
26 }
```



# Validação

- Dessa forma, toda requisição de inserção de hábito irá passar pelas regras de validação
- Agora, se tentarmos cadastrar um novo hábito sem informar os dados nos campos, nada acontecerá
  - Nem irá cadastrar novo hábito
  - Nem seremos direcionados à listagem de hábito
- Isso ocorre pois o método store nem chegou a ser executado, já que o HabitoRequest bloqueou as regras

# Validação

- Precisamos informar ao usuário que erros ocorreram
  - Trabalharemos com a variável `$errors`, que armazena todos os erros de validação encontrados para listarmos ao usuário
  - Vamos alterar o `habitos/create.blade.php`

```
5      <h1>Novo Hábito</h1>
6
7      @if ($errors->any())
8          <ul class="alert alert-danger">
9              @foreach($errors->all() as $error)
10                 <li>{{ $error }}</li>
11             @endforeach
12          </ul>
13      @endif
14
15      {!! Form::open(['url' => 'habitos/store'])
```

## ***Etapa 14***

Removendo hábito cadastrado

# Remoção

- Para fazer a remoção de um registro, devemos criar uma **action** no controlador (**HabitosController.php**)
  - Criar o método `destroy()` para deletar o registro

```
28 public function destroy($id) {  
29     Habito::find($id)->delete();  
30     return redirect(to: 'habitos');  
31 }
```

# Remoção

- Para fazer a remoção de um registro, devemos criar uma **rota** (**routes/web.php**)

```
18 Route::get('habitos', 'HabitosController@index');
19 Route::get('habitos/create', 'HabitosController@create');
20 Route::post('habitos/store', 'HabitosController@store');
21 Route::get('habitos/{id}/destroy', 'HabitosController@destroy');
```

- Nesse caso, quando acessarmos a rota <http://localhost:8000/habitos/1/destroy> automaticamente o registro de ID 1 será excluído

## ***Etapa 15***

Tela de edição de hábitos cadastrados

# Edição

- A tela de edição de um registro deve ser idêntica a tela de inserção
  - Começamos criando duas novas rotas ([routes/web.php](#))

```
18 Route::get('habitos', 'HabitosController@index');
19 Route::get('habitos/create', 'HabitosController@create');
20 Route::post('habitos/store', 'HabitosController@store');
21 Route::get('habitos/{id}/destroy', 'HabitosController@destroy');
22 Route::get('habitos/{id}/edit', 'HabitosController@edit');
23 Route::put('habitos/{id}/update', 'HabitosController@update');
```

# Edição

- A tela de edição de um registro deve ser idêntica a tela de inserção
  - Note, que a rota update utiliza o HTTP PUT
  - Este método é único do Laravel para tratar com segurança os dados

```
18 Route::get('habitos', 'HabitosController@index');
19 Route::get('habitos/create', 'HabitosController@create');
20 Route::post('habitos/store', 'HabitosController@store');
21 Route::get('habitos/{id}/destroy', 'HabitosController@destroy');
22 Route::get('habitos/{id}/edit', 'HabitosController@edit');
23 Route::put('habitos/{id}/update', 'HabitosController@update');
```



# Edição

- A tela de edição de um registro deve ser idêntica a tela de inserção
  - Note, que devemos criar dois novos métodos no controlador

```
18 Route::get('habitos', 'HabitosController@index');
19 Route::get('habitos/create', 'HabitosController@create');
20 Route::post('habitos/store', 'HabitosController@store');
21 Route::get('habitos/{id}/destroy', 'HabitosController@destroy');
22 Route::get('habitos/{id}/edit', 'HabitosController@edit');
23 Route::put('habitos/{id}/update', 'HabitosController@update');
```

# Edição

- Alterando o **controlador** para buscar os **dados** do **BD** e **enviar** para a **rota** de **edição** (tela de apresentação ao usuário)

```
33 public function edit($id) {  
34     $habito = Habito::find($id);  
35     return view('habitots.edit', compact('habito'));  
36 }
```

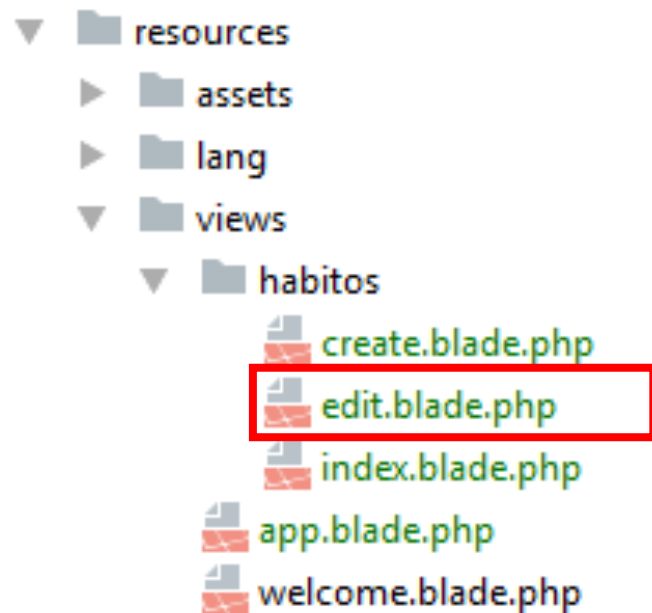
# Edição

- Alterando o **controlador** para **atualizar** os **dados** no **BD**

```
38 public function update(HabitoRequest $request, $id) {  
39     $habito = Habito::find($id)->update($request->all());  
40     return redirect(to: 'habitos');  
41 }
```

# Edição

- Agora temos que criar a tela de edição
  - Crie o arquivo php `edit.blade.php`



# Edição

- Copie o conteúdo do arquivo `create.blade.php` para o `edit`

# Edição

- Vamos começar a adaptar o código para a edição
  - Altere a tag `<h1>` para informarmos o habito em edição

```
1  @extends('app')
2
3  @section('content')
4      <div class="container">
5          <h1>Editando hábito: {{ $habito->nome }}</h1>
6
7          @if ($errors->any())
8              <ul class="alert alert-danger">
9                  @foreach($errors->all() as $error)
10                     <li>{{ $error }}</li>
11                 @endforeach
12             </ul>
13         @endif
```

# Edição

- Vamos começar a adaptar o código para a edição
  - Altere a ação e o método de requisição do form

```
14  
15 {!! Form::open(['url' => "habitots/$habito->id/update"  
16 'method'=>'put']) !!}  
17
```

# Edição

- Vamos começar a adaptar o código para a edição
  - Altere a ação e o método de requisição do form

```
14  
15 {!! Form::open(['url' => "habitos/$habito->id/update",  
16   'method'=>'put']) !!}  
17
```



# Edição

- Vamos começar a adaptar o código para a edição
  - Todos os campos “null” serão substituídos pelos seus devidos valores

```
18 <div class="form-group">
19     {!! Form::label('nome', 'Nome:') !!}
20     {!! Form::text('nome', null, ['class'=>'form-control']) !!}
21 </div>
```

```
18 <div class="form-group">
19     {!! Form::label('nome', 'Nome:') !!}
20     {!! Form::text('nome', $habito->nome, ['class'=>'form-control']) !!}
21 </div>
```

# Edição

- Vamos começar a adaptar o código para a edição
  - Todos os campos “null” serão substituídos pelos seus devidos valores
  - Fazer esta alteração em todos os campos do formulário

```
<div class="form-group">
  {!! Form::label('descricao', 'Descrição:') !!}
  {!! Form::textarea('descricao', $habito->descricao, ['class'=>'form-control']) !!}
</div>
```

```
<div class="form-group">
  {!! Form::label('tp_habito', 'Tipo:') !!}
  {!! Form::select('tp_habito',
    array('B' => 'Bom', 'R' => 'Ruim',
    $habito->tp_habito,
    ['class'=>'form-control']) !!}
</div>
```

```
<div class="form-group">
  {!! Form::label('dt_inicio_ctrl', 'Data:') !!}
  {!! Form::date('dt_inicio_ctrl',
    $habito->dt_inicio_ctrl,
    ['class'=>'form-control']) !!}
</div>
```

```
<div class="form-group">
  {!! Form::label('objetivo', 'Objetivo:') !!}
  {!! Form::number('objetivo', $habito->objetivo, ['class'=>'form-control']) !!}
</div>
```

# Edição

- Vamos começar a adaptar o código para a edição
  - Por fim, alterar a descrição do botão

```
48 <div class="form-group">  
49   {!! Form::submit('Editar hábito', ['class'=>'btn btn-primary']) !!}  
50 </div>
```

# Edição

- Resultado ao acessar </habitos/{id}/edit>

Acesse a tela de edição e altere o dado de algum registro

## Editando hábito: teste

Nome:

teste

Descrição:

teste

Tipo:

Bom

Objetivo:

1

Data:

2017-05-18

Editar hábito

## ***Etapas 16***

Ajustando as rotas

# Organizando as rotas

- Vamos organizar as rotas
  - Routes/web.php

```
18 Route::get('habitos', 'HabitosController@index');
19 Route::get('habitos/create', 'HabitosController@create');
20 Route::post('habitos/store', 'HabitosController@store');
21 Route::get('habitos/{id}/destroy', 'HabitosController@destroy');
22 Route::get('habitos/{id}/edit', 'HabitosController@edit');
23 Route::put('habitos/{id}/update', 'HabitosController@update');
```

# Organizando as rotas

- Vamos organizar as rotas
  - Vamos agrupar as rotas, adicionando prefixo para facilitar a manutenção

```
18 Route::group(['prefix'=>'habitots', 'where'=>['id'=>'[0-9]+']], function() {
19     Route::get('', ['as'=>'habitots', 'uses'=>'HabitotsController@index']);
20     Route::get('create', ['as'=>'habitots.create', 'uses'=>'HabitotsController@create']);
21     Route::get('{id}/destroy', ['as'=>'habitots.destroy', 'uses'=>'HabitotsController@destroy']);
22     Route::get('{id}/edit', ['as'=>'habitots.edit', 'uses'=>'HabitotsController@edit']);
23     Route::put('{id}/update', ['as'=>'habitots.update', 'uses'=>'HabitotsController@update']);
24     Route::post('store', ['as'=>'habitots.store', 'uses'=>'HabitotsController@store']);
25     });
```

# Organizando as rotas

- Vamos organizar as rotas
  - Se mudarmos o prefixo, todas as rotas mudam

```
18 Route::group(['prefix'=>'habitots', 'where'=>['id'=>'[0-9]+']], function() {  
19     Route::get('', ['as'=>'habitots', 'uses'=>'HabitotsController@index']);  
20     Route::get('create', ['as'=>'habitots.create', 'uses'=>'HabitotsController@create']);  
21     Route::get('{id}/destroy', ['as'=>'habitots.destroy', 'uses'=>'HabitotsController@destroy']);  
22     Route::get('{id}/edit', ['as'=>'habitots.edit', 'uses'=>'HabitotsController@edit']);  
23     Route::put('{id}/update', ['as'=>'habitots.update', 'uses'=>'HabitotsController@update']);  
24     Route::post('store', ['as'=>'habitots.store', 'uses'=>'HabitotsController@store']);  
25     });
```



# Organizando as rotas

- Vamos organizar as rotas
  - Quando utilizamos rotas nomeadas, podemos acessar as rotas pelos nomes ao invés de chamar uma URL fixa

```
18 Route::group(['prefix'=>'habitots', 'where'=>['id'=>'[0-9]+']], function() {
19     Route::get('', ['as'=>'habitots', 'uses'=>'HabitotsController@index']);
20     Route::get('create', ['as'=>'habitots.create', 'uses'=>'HabitotsController@create']);
21     Route::get('{id}/destroy', ['as'=>'habitots.destroy', 'uses'=>'HabitotsController@destroy']);
22     Route::get('{id}/edit', ['as'=>'habitots.edit', 'uses'=>'HabitotsController@edit']);
23     Route::put('{id}/update', ['as'=>'habitots.update', 'uses'=>'HabitotsController@update']);
24     Route::post('store', ['as'=>'habitots.store', 'uses'=>'HabitotsController@store']);
25     });
```

# Organizando as rotas

- Vamos organizar as rotas
  - Usamos uma *key* 'as' para nomearmos a rota e uma *key* 'uses' para definirmos qual a ação da rota

```
18 Route::group(['prefix'=>'habitots', 'where'=>['id'=>'[0-9]+']], function() {
19     Route::get('', ['as'=>'habitots', 'uses'=>'HabitotsController@index']);
20     Route::get('create', ['as'=>'habitots.create', 'uses'=>'HabitotsController@create']);
21     Route::get('{id}/destroy', ['as'=>'habitots.destroy', 'uses'=>'HabitotsController@destroy']);
22     Route::get('{id}/edit', ['as'=>'habitots.edit', 'uses'=>'HabitotsController@edit']);
23     Route::put('{id}/update', ['as'=>'habitots.update', 'uses'=>'HabitotsController@update']);
24     Route::post('store', ['as'=>'habitots.store', 'uses'=>'HabitotsController@store']);
25     });
```

# Organizando as rotas

- Vamos organizar as rotas
  - Usamos a clausula where para informar que o parâmetro ID deve ser numérico

```
18 Route::group(['prefix'=>'habitros', 'where'=>['id'=>'[0-9]+']] function() {  
19     Route::get('', ['as'=>'habitros', 'uses'=>'HabitrosController@index']);  
20     Route::get('create', ['as'=>'habitros.create', 'uses'=>'HabitrosController@create']);  
21     Route::get('{id}/destroy', ['as'=>'habitros.destroy', 'uses'=>'HabitrosController@destroy']);  
22     Route::get('{id}/edit', ['as'=>'habitros.edit', 'uses'=>'HabitrosController@edit']);  
23     Route::put('{id}/update', ['as'=>'habitros.update', 'uses'=>'HabitrosController@update']);  
24     Route::post('store', ['as'=>'habitros.store', 'uses'=>'HabitrosController@store']);  
25     });
```

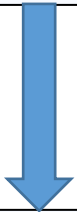
# Organizando as rotas

- Vamos organizar as rotas
  - Agora devemos alterar todos os lugares que temos rotas fixas pelas rotas nomeadas

# Organizando as rotas

- Vamos organizar as rotas
  - No controlador (`app/Http/Controllers/HabitosController.php`)
  - Alterar todos os retornos para rotas `habitos`

```
return redirect( to: 'habitos' );
```

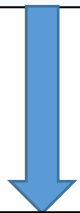


```
return redirect()->route( route: 'habitos' );
```

# Organizando as rotas

- Vamos organizar as rotas
  - Nas visões (**resources/views/habitos**) → **CREATE.BLADE.PHP**
  - **Alterar todos as rotas do form open**

```
{!! Form::open(['url' => 'habitos/store']) !!}
```

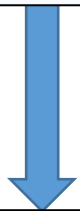


```
{!! Form::open(['route' => 'habitos.store']) !!}
```

# Organizando as rotas

- Vamos organizar as rotas
  - Nas visões (**resources/views/habitos**) → **EDIT.BLADE.PHP**
  - **Alterar todos as rotas do form open**

```
{!! Form::open(['url' => "habitos/$habito->id/update", 'method'=>'put']) !!}
```



```
{!! Form::open(['route' => ["habitos.update", $habito->id], 'method'=>'put']) !!}
```

## ***Etapa 17***

Ajustando links na tela de listagem

*Adicionando botões de edição e exclusão*



# Otimizando a listagem

- Vamos melhorar nossa tela de listagem adicionando dois botões



# Otimizando a listagem

- Vamos melhorar nossa tela de listagem adicionando dois botões

```
<td>
  <a href="{{ route('habitos.edit', ['id'=>$hab->id]) }}"
    class="btn-sm btn-success">Editar</a>
  <a href="{{ route('habitos.destroy', ['id'=>$hab->id]) }}"
    class="btn-sm btn-danger">Remover</a>
</td>
</tr>
@endforeach
</tbody>
```

# Otimizando a listagem

- Vamos melhorar nossa tela de listagem alterando a forma de impressão do tipo de hábito (bom ou ruim)

```
@if ($hab->tp_habito == 'B')  
    <td>Bom</td>  
@elseif ($hab->tp_habito == 'R')  
    <td>Ruim</td>  
@endif
```

## ***Etapa 18***

Criando o cadastro de Histórico

# Camada Model

- Para criarmos o model e a migration dos historicos vamos utilizar o Artisan
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de criação do model
    - `php artisan make:model Historico -m`

```
C:\Users\Madalozzo\meu_habito (aula_1)
λ php artisan make:model Historico -m
Model created successfully.
Created Migration: 2017_06_08_152907_create_historicos_table
```

## ***Etapa 19***

Criando campo FK

# Migration

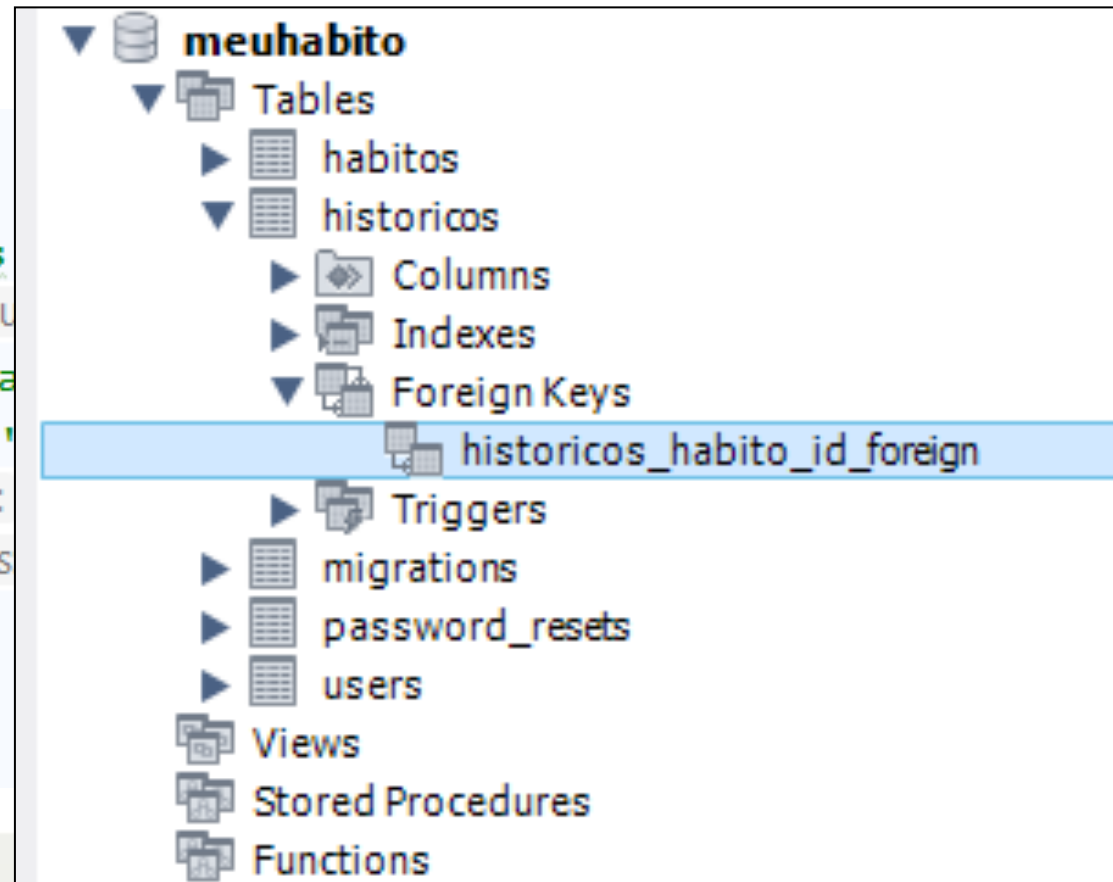
- Vamos alterar o arquivo de migração para criarmos a tabela no banco

```
public function up()
{
    Schema::create('historicos', function (Blueprint $table) {
        $table->increments( column: 'id');
        $table->date( column: 'data');
        $table->string( column: 'hora')->nullable();
        $table->integer( column: 'habito_id')->unsigned();
        $table->foreign( columns: 'habito_id')->references('id')->on('habitots');
        $table->timestamps();
    });
}
```

# Migration

- Vamos alterar o arquivo de migração para criarmos a tabela no banco

```
public function up()
{
    Schema::create('historicos', function($table) {
        $table->increments('id');
        $table->date('data');
        $table->string('descricao', 255);
        $table->integer('habitoid');
        $table->foreign('habitoid', 'habitoid');
        $table->timestamps();
    });
}
```




itos');



# Migration



- Vamos alterar o arquivo de migração para criarmos a tabela no banco

▼  meuhabito

Column Name	Datatype	PK	NN
id	INT(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
data	DATE	<input type="checkbox"/>	<input checked="" type="checkbox"/>
hora	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>
habito_id	INT(10)	<input type="checkbox"/>	<input checked="" type="checkbox"/>

```
reprint $table) {
```

```
$table->string( column: 'hora')->nullable();  
$table->integer( column: 'habito_id')->unsigned();  
$table->foreign( columns: 'habito_id')->references('id')->on('habitos');  
$table->timestamps();  
});  
}
```

 Stored Procedures  
 Functions

## ***Etapa 20***

Criando relacionamento da FK

# Model

- Vamos alterar o model Historico ([app/Historico.php](#))
- Primeiro vamos resolver o problema de MassAssignment adicionando o campo **fillable**

```
7  class Historico extends Model
8  {
9      protected $fillable = [
10         'data',
11         'hora',
12         'habito_id'
13     ];
14 }
```

# Model

- Vamos alterar o model Historico ([app/Historico.php](#))
  - Agora, dentro do histórico, vamos criar uma função para retornar o Habito cadastrado no Histórico

```
15 public function habito() {  
16     return $this->belongsTo( related: 'App\Habito' );  
17 }
```

# Model

- Vamos alterar o model Historico ([app/Historico.php](#))
  - A função **belongsTo** informa ao Laravel que o Historico pertence ao Habito, pois simplesmente contém um habito

```
15 public function habito() {  
16     return $this->belongsTo( related: 'App\Habito' );  
17 }
```

- Um hábito pode existir sem um histórico?
- E um histórico, pode existir sem um hábito?

Por isso um histórico pertence a um hábito

# Model

- Vamos alterar o model Habito ([app/Habito.php](#))
  - Agora vamos fazer o inverso
  - Vamos informar ao Habito que ele possui diversos Históricos

```
12     public function historicos() {  
13         return $this->hasMany( related: 'App\Historico' );  
14     }
```

# Model

- Vamos alterar o model Habito ([app/Habito.php](#))
  - A função `hasMany` informa ao Laravel que um Habito pode estar em diversos históricos

```
12 public function historicos() {  
13     return $this->hasMany(related: 'App\Historico');  
14 }
```

## ***Etapa 21***

Criando as rotas



# Model

- Vamos adicionar as rotas dos historicos ([routes/web.php](#))
  - Adicionaremos as mesmas rotas dos habitos

```
27 Route::group(['prefix'=>'historicos', 'where'=>['id'=>'[0-9]+' ]], function() {
28     Route::get('', ['as'=>'historicos', 'uses'=>'HistoricosController@index']);
29     Route::get('create', ['as'=>'historicos.create', 'uses'=>'HistoricosController@create']);
30     Route::get('{id}/destroy', ['as'=>'historicos.destroy', 'uses'=>'HistoricosController@destroy']);
31     Route::get('{id}/edit', ['as'=>'historicos.edit', 'uses'=>'HistoricosController@edit']);
32     Route::put('{id}/update', ['as'=>'historicos.update', 'uses'=>'HistoricosController@update']);
33     Route::post('store', ['as'=>'historicos.store', 'uses'=>'HistoricosController@store']);
34     });
```

## ***Etapa 22***

Criando o controlador

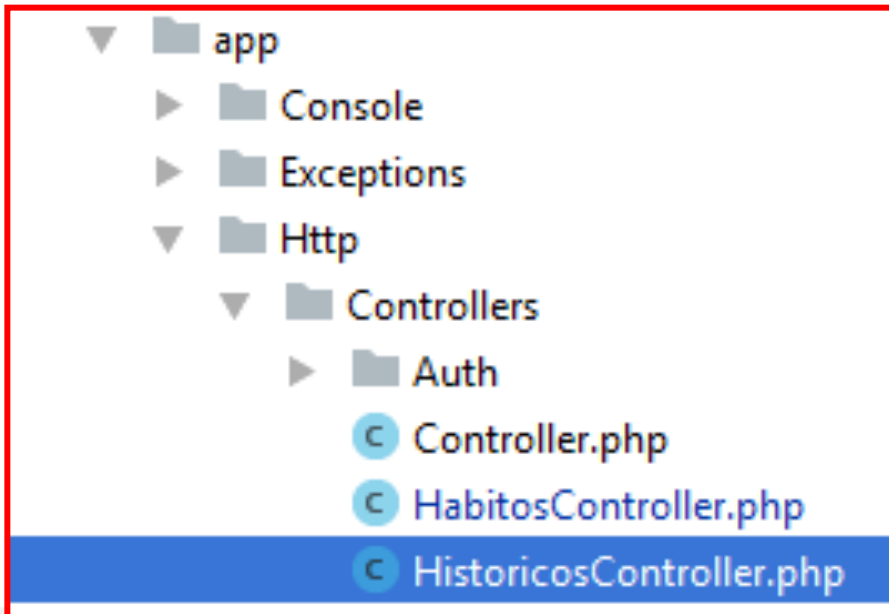
# Controlador

- Para criarmos o controller vamos utilizar o Artisan
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de criação do controlador
    - `php artisan make:controller HistoricosController`

```
C:\Users\Madalozzo\meu_habito (aula_1)  
λ php artisan make:controller HistoricosController  
Controller created successfully.
```

# Controlador

- Para criarmos o controller vamos utilizar o Artisan
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando de criação do controlador  
→ `php artisan make:controller HistoricosController`



```
meu_habito (aula_1)  
php artisan make:controller HistoricosController  
Successfully.
```

# Controlador

- Agora, com o controlador criado, vamos criar os métodos necessários para o CRUD
  - Adicionaremos o **index()**

```
12 public function index() {  
13     $historicos = Historico::all();  
14     return view( view: 'historicos.index',  
15                 ['historicos'=>$historicos]);  
16 }
```

# Controlador

- Agora, com o controlador criado, vamos criar os métodos necessários para o CRUD
  - Adicionaremos o `create()`

```
16 public function create() {  
17     return view( view: 'historicos.create' );  
18 }
```

# Controlador

- Agora, com o controlador criado, vamos criar os métodos necessários para o CRUD
  - Adicionaremos o **store()**

```
20 public function store(HistoricoRequest $request) {  
21     $novo_historico = $request->all();  
22     Historico::create($novo_historico);  
23  
24     return redirect()->route('historicos');  
25 }
```

# Controlador

- Agora, com o controlador criado, vamos criar os métodos necessários para o CRUD
  - Adicionaremos o **destroy()**

```
27 public function destroy($id) {  
28     Historico::find($id)->delete();  
29     return redirect()->route('historicos');  
30 }
```



# Controlador

- Agora, com o controlador criado, vamos criar os métodos necessários para o CRUD
  - Adicionaremos o **edit()**

```
32 public function edit($id) {  
33     $historico = Historico::find($id);  
34     return view('historico.edit', compact('historico'));  
35 }
```

# Controlador

- Agora, com o controlador criado, vamos criar os métodos necessários para o CRUD
  - Adicionaremos o **update()**

```
37 public function update(HistoricoRequest $request, $id) {  
38     $historico = Historico::find($id)->update($request->all());  
39     return redirect()->route('historicos');  
40 }
```

## ***Etapa 23***

Criando o request para validação dos dados

# Validação

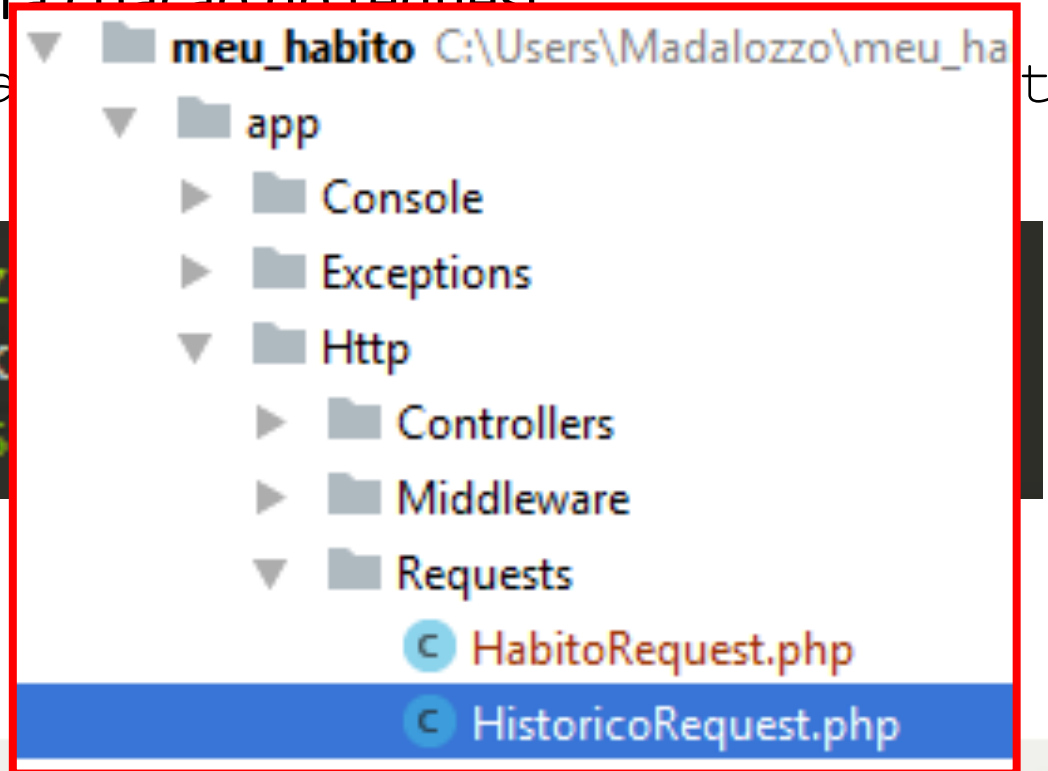
- Utilizaremos o artisan para gerar a classe de Request
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando para criação do request
    - `php artisan make:request HistoricoRequest`

```
C:\Users\Madalozzo\meu_habito (aula_1)
λ php artisan make:request HistoricoRequest
Request created successfully.
```

# Validação

- Utilizaremos o artisan para gerar a classe de Request
  - Então, usando o terminal (cmd) dentro da pasta meu\_habito
  - Execute o comando para criação de request  
→ `php artisan make:request`

```
C:\Users\Madalozzo> php artisan make:request  
Request created successfully
```



# Validação

- Na classe **HistoricoRequest.php** vamos alterar o return do método authorize para TRUE

```
14      public function authorize()  
15      {  
16          return true;  
17      }
```

# Validação

- Na classe **HistoricoRequest.php** vamos adicionar as regras no método `rules()`

```
24     public function rules()  
25     {  
26         return [  
27             'date' => 'required',  
28             'habito_id' => 'required',  
29         ];  
30     }
```

# Validação

- Na classe **HistoricoController.php** devemos adicionar o uso de dois arquivos (Model e Request)

```
5  use Illuminate\Http\Request;  
6  use App\Http\Controllers\Controller;  
7  use App\Historico;  
8  use App\Http\Requests\HistoricoRequest;
```

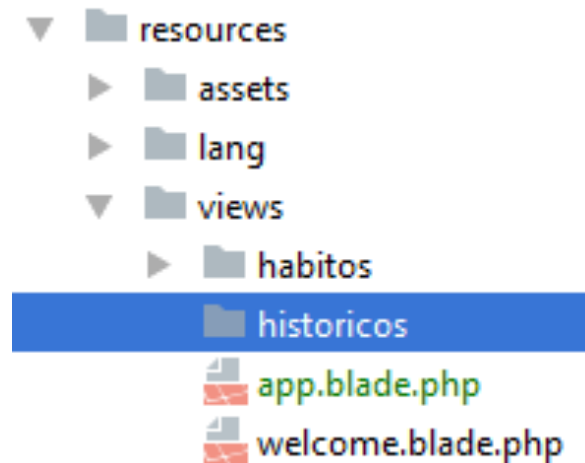


## ***Etapa 24***

Criando a visão

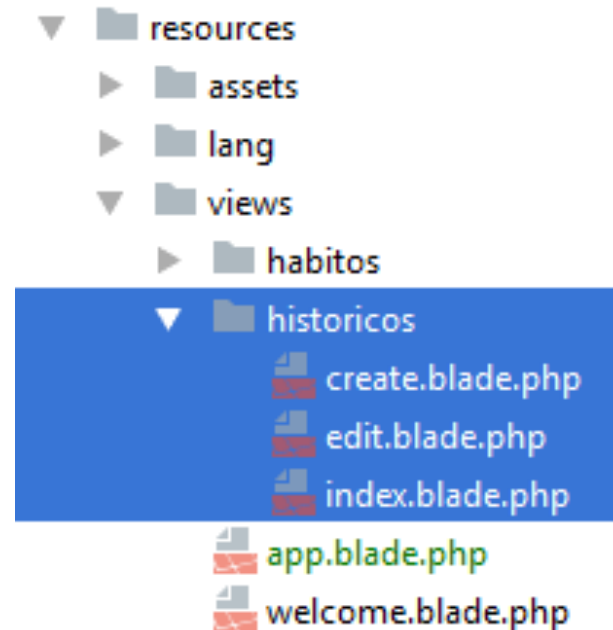
# Visão

- Devemos criar agora as telas do histórico
  - Então, vamos criar uma pasta chamada historicos dentro de resources/views



# Visão

- Devemos criar agora as telas do histórico
  - `create.blade.php`
  - `edit.blade.php`
  - `index.blade.php`



# Visão

- Vamos criar a tela de listagem ([index.blade.php](#)) → Parte 1

```
1  @extends('app') @extends('adminlte::default')
2
3  @section('content')
4      <div class="container">
5          <h1>Históricos</h1>
6
7          <table class="table table-striped table-bordered table-hover">
8              <thead>
9                  <tr>
10                     <th>Hábito</th>
11                     <th>Data</th>
12                     <th>Ação</th>
13                 </tr>
14             </thead>
```

# Visão

- Vamos criar a tela de listagem ([index.blade.php](#)) → Parte 2

```
16      <tbody>
17      @foreach($historicos as $hist)
18      <tr>
19          <td>{{ $hist->habito->nome }}</td>
20          <td>{{ $hist->data }}</td>
21
22          <td>
23              <a href="{{ route('historicos.edit', ['id'=>$hist->id]) }}"
24                  class="btn-sm btn-success">Editar</a>
25              <a href="{{ route('historicos.destroy', ['id'=>$hist->id]) }}"
26                  class="btn-sm btn-danger">Remover</a>
27          </td>
28      </tr>
29      @endforeach
30      </tbody>
31  </table>
32 </div>
33 @endsection
```

# Visão

- Vamos criar a tela de criação (**create.blade.php**)
  - Primeiro adicionamos o layout e apresentamos os erros

```
1  @extends('app') @extends('adminlte::default')
2
3  @section('content')
4      <div class="container">
5          <h1>Novo Histórico</h1>
6
7          @if ($errors->any())
8              <ul class="alert alert-danger">
9                  @foreach($errors->all() as $error)
10                     <li>{{ $error }}</li>
11                 @endforeach
12             </ul>
13         @endif
```

# Visão

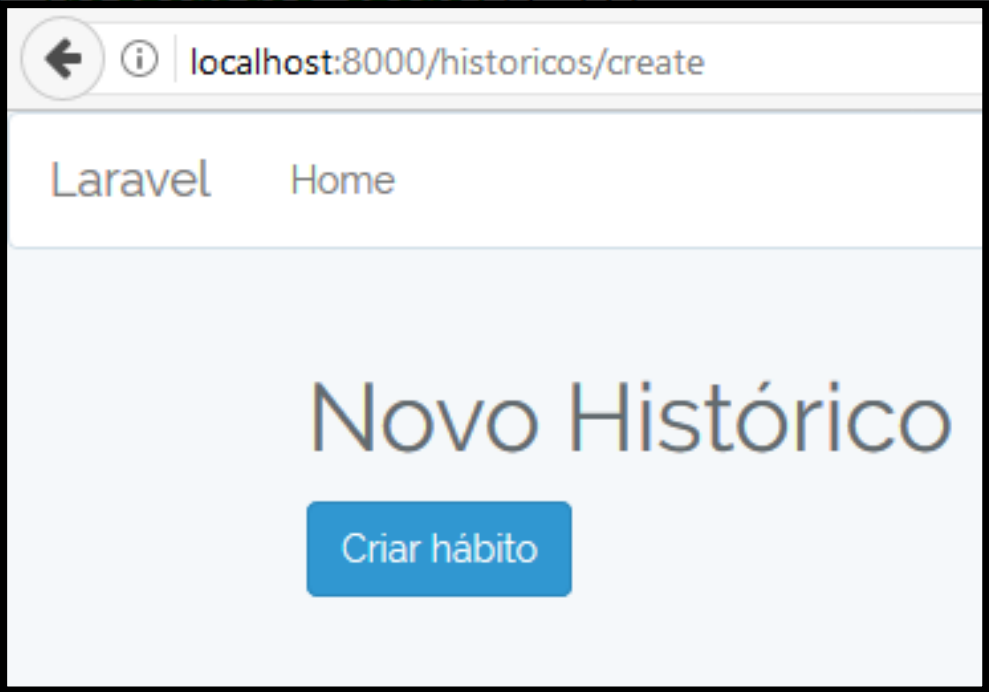
- Vamos criar a tela de criação (**create.blade.php**)
  - Depois abrimos o formulário, adicionamos o botão de criação (realizada ação do form) e depois fechamos o formulário

```
15      {!! Form::open(['route' => 'historicos.store']) !!}
16
17
18
19
20      <div class="form-group">
21          {!! Form::submit('Criar histórico', ['class'=>'btn btn-primary']) !!}
22      </div>
23
24      {!! Form::close() !!}
25
26      </div>
27  @endsection
```

# Visão

- Vamos criar a tela de criação (**create.blade.php**)
  - Depois abrimos o formulário, adicionamos o botão de criação (realizada ação do form) e depois fechamos o formulário

```
15 {!! Form::open(['route' => 'historicos.create']) !!}  
16  
17  
18  
19  
20 <div class="form-group">  
21 {!! Form::submit('Criar histórico', ['class' => 'btn btn-primary']) !!}  
22 </div>  
23  
24 {!! Form::close() !!}  
25  
26 </div>  
27 @endsection
```



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/historicos/create'. The browser's navigation bar includes 'Laravel' and 'Home' links. The main content area of the page features the title 'Novo Histórico' in a large, dark font. Below the title is a blue button with the text 'Criar hábito'.



# Visão

- Vamos criar a tela de criação (**create.blade.php**)
  - Vamos adicionar o primeiro campo ao formulário
  - Adicionando o campo Hábitos
  - Usamos o campo do tipo “select” e depois usamos os métodos orderBy e pluck do EloquentORM presente no Model Habito

```
15      {!! Form::open(['route' => 'historicos.store']) !!}
16
17      <!-- Campo Habito_Id -->
18      <div class="form-group">
19          {!! Form::label('habito_id', 'Hábito:') !!}
20          {!! Form::select('habito_id',
21                          \App\Habito::orderBy('nome')->pluck('nome', 'id')->toArray(), null,
22                          ['class'=>'form-control']) !!}
23      </div>
```

# Visão

- Vamos criar a tela de criação (`create.blade.php`)
  - Vamos adicionar o primeiro campo ao formulário
  - Adicionando o campo Hábitos
  - Usamos o campo do tipo “select” e depois usamos os métodos `orderBy` e `pluck` do EloquentORM presente no Model Habito

```
15      {!! Form::open(['route' => 'historicos.store']) !!}
16
17      <!-- Campo Habito_Id -->
18      <div class="form-group">
19          {!! Form::label('habito_id', 'Hábito:') !!}
20          {{ Form::select('habito_id',
21              \App\Habito::orderBy('nome')->pluck('nome', 'id')->toArray(), null,
22              ['class'=>'form-control']) }}
23      </div>
```

Apresenta na listagem o  
Nome e grava o Id no campo  
habito\_id

# Visão

- Vamos criar a tela de criação (**create.blade.php**)
  - Vamos adicionar o campo de data e hora

```
25      <!-- Campo Data -->
26      <div class="form-group">
27          {!! Form::label('data', 'Data:') !!}
28          {!! Form::text('data', null, ['class'=>'form-control']) !!}
29      </div>
30
31      <!-- Campo Hora -->
32      <div class="form-group">
33          {!! Form::label('hora', 'Hora:') !!}
34          {!! Form::text('hora', null, ['class'=>'form-control']) !!}
35      </div>
```

# Visão

- Vamos criar a tela de criação
  - Vamos adicionar o campo de

```
25 <!-- Campo Data -->
26 <div class="form-group">
27     {!! Form::label('Data', 'Data:') !!}
28     {!! Form::text('data') !!}
29 </div>
30
31 <!-- Campo Hora -->
32 <div class="form-group">
33     {!! Form::label('Hora', 'Hora:') !!}
34     {!! Form::text('hora') !!}
35 </div>
```

localhost:8000/historicos/create

Laravel Home

## Novo Histórico

Hábito:

Correr

Correr

NovoHaub

Hora:

Criar histórico

# Visão

- Vamos criar a tela de edição (**edit.blade.php**)
  - Copie o código do create.blade.php do histórico
  - Cole no arquivo edit.blade.php do histórico
  - Altere o título do formulário de “Novo Historico” para “Editando Historico”

```
1 @extends ( 'app' )
2
3 @section ( 'content' )
4     <div class="container">
5         <h1>Editando Histórico</h1>
```

# Visão

- Vamos criar a tela de edição ([edit.blade.php](#))
  - Altere o Form::Open

```
15 {!! Form::open(['route' => ["historicos.update",  
16                               $historico->id],  
17                               'method'=>'put']) !!}  
18 }
```

# Visão

- Vamos criar a tela de edição ([edit.blade.php](#))
  - Por fim, substitua todos os parâmetros “NULL” pelos campos corretos de edição

```
<!-- Campo Habito_Id -->
<div class="form-group">
    {!! Form::label('habito_id', 'Hábito:') !!}
    {!! Form::select('habito_id',
        \App\Habito::orderBy('nome')->pluck('nome', 'id')->toArray(),
        $historico->habito_id,
        ['class'=>'form-control']) !!}
</div>
```

```
<!-- Campo Hora -->
<div class="form-group">
    {!! Form::label('hora', 'Hora:') !!}
    {!! Form::text('hora', $historico->hora, ['class'=>'form-control']) !!}
</div>
```

```
<!-- Campo Hora -->
<div class="form-group">
    {!! Form::label('hora', 'Hora:') !!}
    {!! Form::text('hora', $historico->hora, ['class'=>'form-control']) !!}
</div>
```