

Criando aplicativos **Usando Laravel**

Com bons recursos

Versão 9

Julho de 2022

por Ribamar FS

Índice

Público Alvo.....	5
Agradecimentos.....	5
Sobre Ribamar FS.....	5
Prefácio.....	6
Introdução.....	11
Primeira Parte.....	13
0 - Introdução ao Laravel.....	13
1.1 – Ambiente de Desenvolvimento.....	19
Principais ferramentas para a criação do nosso aplicativo.....	19
Requisitos para a instalação do Laravel 9.....	20
1.2 - Ambiente em Desktop Linux.....	21
1.3 – Lagaron no Windows.....	22
1.4 – Algumas Convenções do Laravel.....	22
1.5 - Criação do Banco de Dados.....	24
1.6 - Configurações diversas.....	25
1 – Criar repositório e instalar laravel.....	29
2 – Implementar permissions.....	33
3 – Ajustes na ACL.....	35
Implementação do AdminLTE.....	43
Script que marca todos os checkboxes de roles.....	61
4 – Criar os Seeders.....	63
5 – Criação dos CRUDs Posts e Comments.....	68
6 – Ajustes na paginação.....	71
Criação de commands.....	76
Vamos agora configurar a Localization para pt-BR.....	76
Instalar e configurar o pacote DebugBar, para melhorar o debug.....	77
7 – Tratamento de Erros e Sugestões.....	78
Tratamento de erros.....	78
Ajustar as rotas.....	79
Ajuste fino no controle de acesso.....	79
Ferramenta para gerar seeders de tabelas existentes.....	83
Sugestões para a adição de um novo CRUD.....	83
Criação de Script.....	84
Um pouco sobre padrões.....	87
Suporte aos 3 SGBFs free e open.....	88
MySQL/MariaDb.....	88
PostgreSQL.....	88
SQLite.....	89
8 – Ajustes Finais no Aplicativo.....	91
Adicionar o CDN abaixo ao final do app.blade.php logo abaixo de app.js.....	92
Criar o CRUD para permissions.....	92
Ajustar o alert do Bootstrap.....	93
Ocultando o link Register.....	96
Então ficará assim.....	97
Uma boa opção para gerar dados de teste é o faker-restaurant.....	97
9 – Produtividade do Programador.....	98

Segunda Parte.....	102
1 - Novidades da Versão 9.....	102
2 - Planejamento.....	106
3 – Maneiras de Instalar o laravel 9.....	107
Instalação com docker/SAIL.....	107
Instalar com o instalador global no Ubuntu.....	108
Bootstrap.....	108
Vue.....	108
React.....	108
Instalar o Docker no Windows.....	109
4 - Convenções no Laravel.....	110
5 – Bancos de Dados.....	114
Migrations.....	114
Tipos de campos suportados nas migrations.....	116
Modificadores de campos.....	117
Alteração de tabelas através de migrations.....	118
Factory no Laravel.....	120
Criando dados faker com Factory.....	121
Seeder no Laravel 9.....	124
Usando o FakerRestaurant.....	131
Exemplo de DatabaseSeeder.php.....	133
Bancos de dados no Laravel.....	134
Criar aplicativo de testes.....	139
Eloquent ORM.....	140
Query Builder.....	143
Como obter registros aleatórios em Laravel.....	145
Popular select com registros de outra tabela.....	147
Tinker.....	148
Trabalhando com datas.....	150
6 – MVC.....	152
Model.....	152
Relacionamentos.....	152
Views.....	165
Paginação.....	165
Validação de formulários.....	166
Customizar as mensagens de erro.....	169
Controller.....	171
Rotas.....	171
7 – Segurança.....	174
Autenticação.....	174
Autorização.....	176
Roles and Permissions.....	177
8 - Recursos usando Laravel.....	186
E-commerce usando Laravel.....	186
9 – Dicas sobre Laravel.....	187
Tornando a instalação mais amigável.....	187
Limpando o cache de views.....	188
Algumas funções.....	188

10 – Ferramentas.....	190
Artisan.....	190
Backup.....	193
Aplicativo para Testes.....	193
11 – Deploy.....	194
12 – Boas Práticas.....	198
Recursos do PHP Moderno.....	203
Clean Code.....	205
Porque usar um CMS.....	205
Porque usar um bom Framework.....	206
13 – Laravel Skeleton.....	207
Na prática, podemos pegar todo o aplicativo criado ou qualquer outro e criar um esqueleto.	213
Como instalar.....	214
14 – Criando um demo para o aplicativo.....	215
15 – Customização do Aplicativo.....	217
16 – Referências.....	218
17 – Bônus.....	222
Criando aplicativo com ACL usando o CakePHP 3.....	222

Público Alvo

Este livro destina-se a programadores com conhecimento do PHP, especialmente com conhecimento de orientação a objetos e MVC no PHP e também a programadores iniciantes no framework Laravel.

Programadores de outras linguagens, desde que bem motivados, podem acompanhar o conteúdo do livro, pois o assunto abordado não é avançado e é abordado de forma propositalmente simplificada.

Agradecimentos

Gostaria de deixar registrado meu agradecimento a todos que colaboraram direta ou indiretamente para que este livro fosse publicado:

- Equipe do Laravel por criar este excelente framework em PHP
- Equipe do Google por criar um excelente instrumento de busca, que facilita encontrar conteúdo
- A equipe do Github, pela criação de um excelente serviço de rede social de TI
- Ao pessoal do StackOverflow, pela criação da estrutura quase onipresente na solução de problemas
- Aos colegas em grupos do Facebook, que me ajudaram na solução de dúvidas
- A diversos criadores de conteúdo que compartilham tutoriais em seus sites
- A a outros que porventura esqueci de citar

Sobre Ribamar FS

Programador PHP com mais de 20 anos de experiência. Trabalho atualmente no DNOCS como programador web, onde já trabalhei como administrador dos servidores, como administrador do SGBD PostgreSQL, como professor e mais.

Experiência com o CMS Joomla (desde a versão 1.0), com o framework Laravel (desde a versão 5), com alguns livros publicados (<https://ribafs.github.io/phpecia/sobre/meus-livros.html>), diversos projetos publicados e vários forks no Github (<https://github.com/ribafs>).

Participo ativamente de alguns grupos no Facebook, como PHP Brasil, Laravel Brasil, Clube dos programadores e outros.

Mais algumas informações sobre mim (<https://ribamar.net.br/portal/sobre/curriculo>) em meu novo site, que está apenas iniciando.

Finalizando, sou apaixonado por programação, por servidores, por escrever e ensinar.

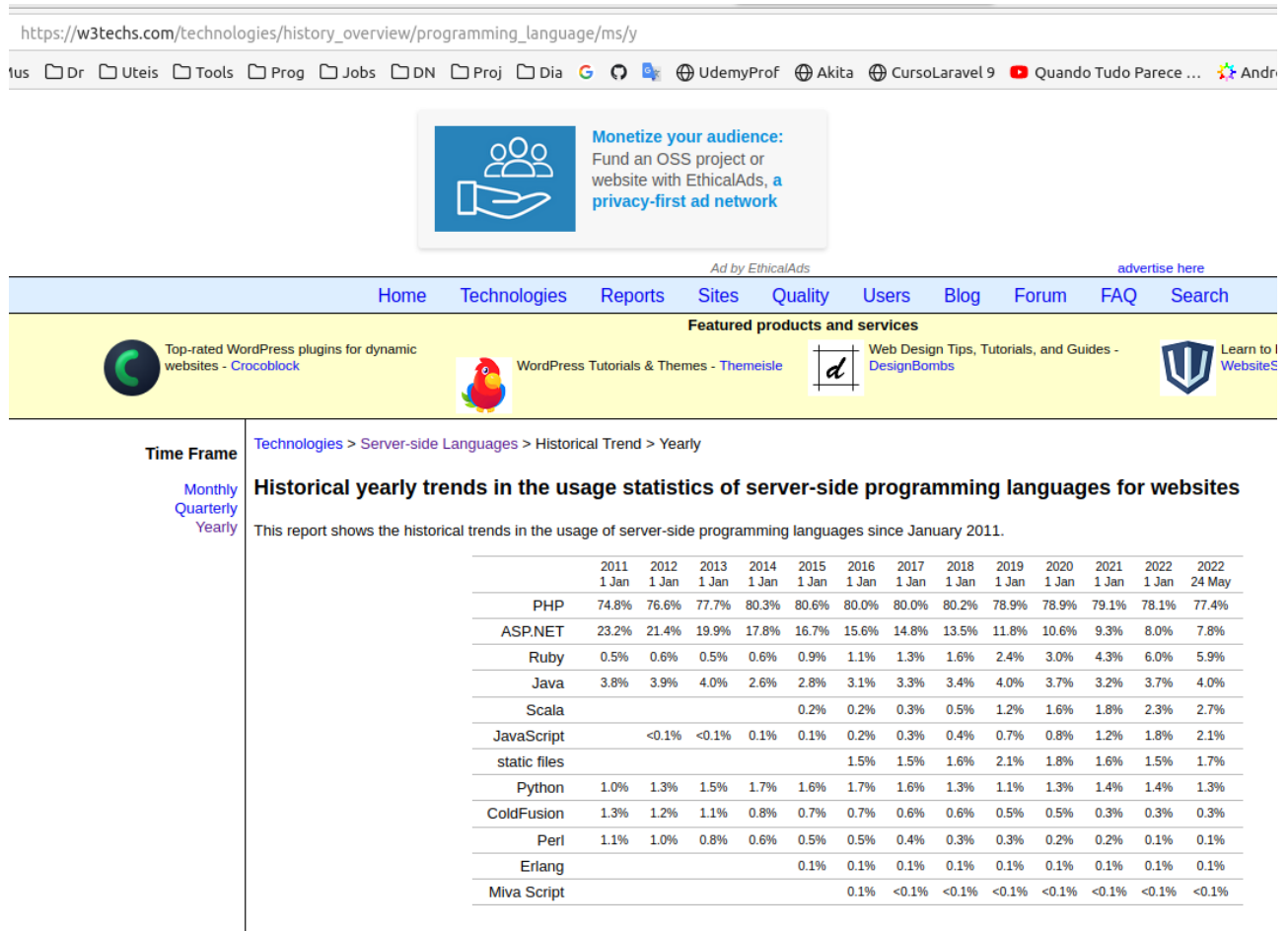
Selecionei alguns dos meus repositórios, visto que a maioria é fork:

<https://github.com/ribafs/alguns-repos>

Prefácio

PHP

O PHP é atualmente a linguagem web mais popular e isso já se repete por 11 anos. Veja a estatística



É importante observar que durante estes 11 anos o PHP ainda melhorou no ranking. Em 2022 ele está com um percentual ainda maior que em 2011.

https://w3techs.com/technologies/overview/programming_language

Mus Dr Uteis Tools Prog Jobs DN Proj Dia UdemProf Akita CursoLaravel 9 Quando Tu

Ad by EthicalAds

Home Technologies Reports Sites Quality Users Blog Forum F



Web Design Tips, Tutorials, and Guides -
DesignBombs



Top-rated WordPress plugins for dynamic
websites - Crocoblock



Learn to build a website, step-by-step -
WebsiteSetup

Technologies

Content Management
Server-side Languages
Client-side Languages
JavaScript Libraries
CSS Frameworks
Web Servers
Web Panels
Operating Systems
Web Hosting
Data Centers
Reverse Proxies
DNS Servers
Email Servers
SSL Certificate Authorities
Content Delivery
Traffic Analysis Tools
Advertising Networks
Tag Managers
Social Widgets
Site Elements
Structured Data
Markup Languages
Character Encodings
Image File Formats
Top Level Domains
Server Locations
Content Languages

Trends

History

Market

Technologies > Server-side Languages

Usage statistics of server-side programming languages for websites

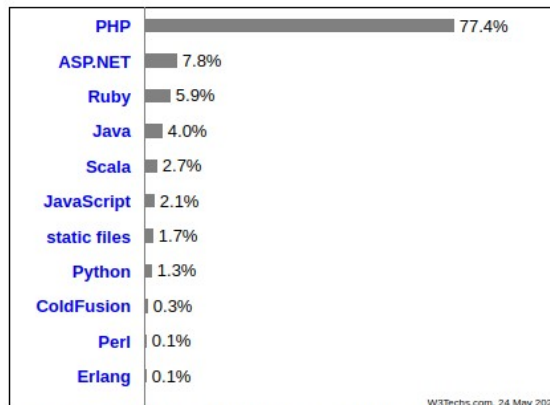
Request an extensive server-side programming languages market report.

[Learn more](#)

This diagram shows the percentages of websites using various server-side programming languages. See [technologies overview](#) for explanations on the methodologies used in the surveys. Our reports are updated daily.

How to read the diagram:

PHP is used by 77.4% of all the websites whose server-side programming language we know.

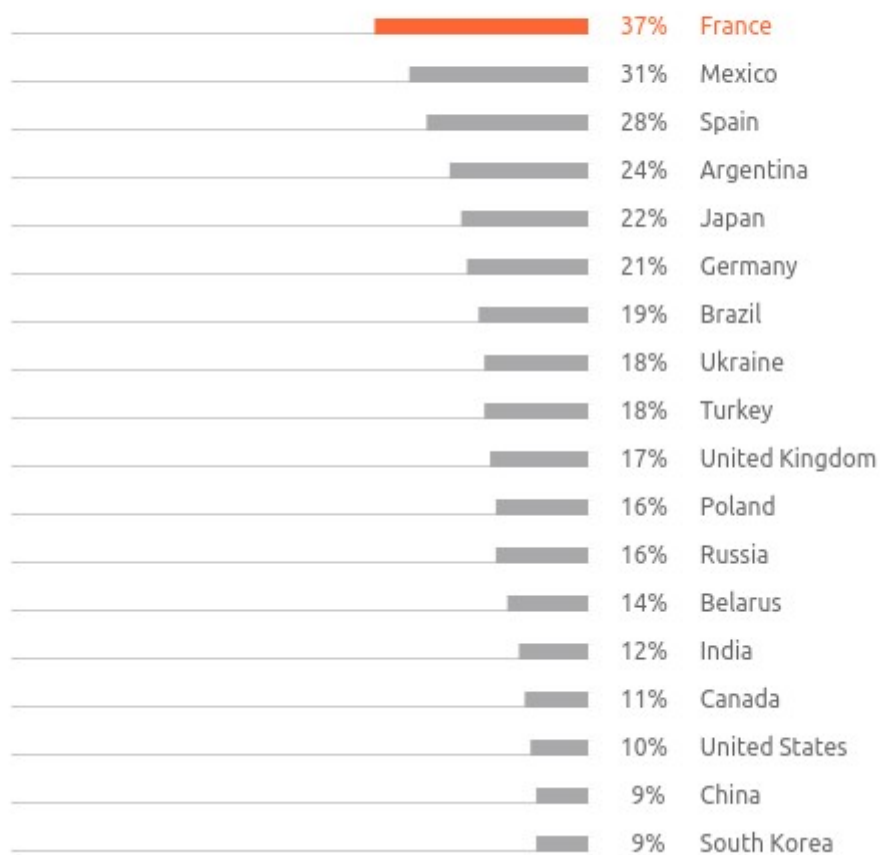


W3Techs.com, 24 May 2022

Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

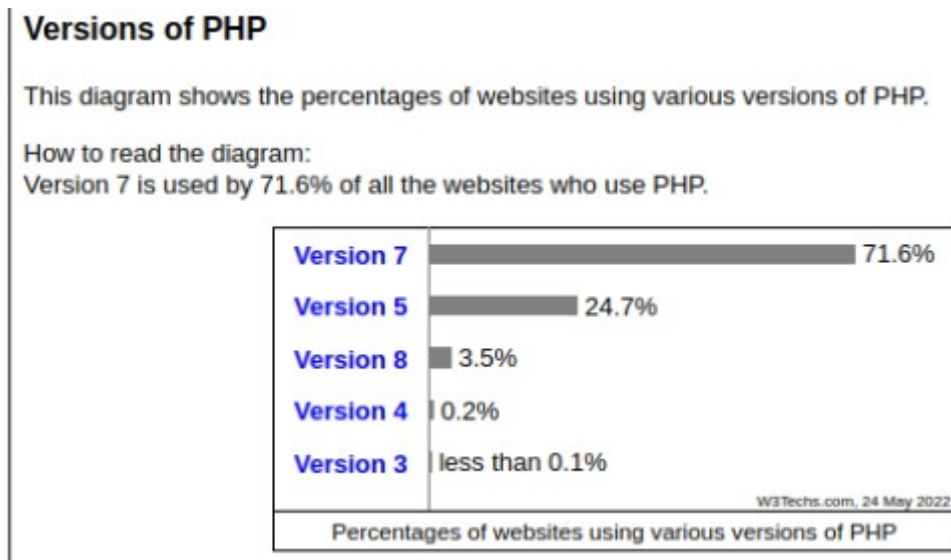
Mais uma estatística

Share of PHP as a primary programming language by country



Países que mais usam o PHP

Uso do PHP por versão



O PHP tem a curva de aprendizado mais simples se comparado com as demais linguagens web. Além disso é uma linguagem que tem bons recursos e melhora a cada versão.

Laravel

Já o Laravel é atualmente o framework PHP mais popular que existe.

Site oficial

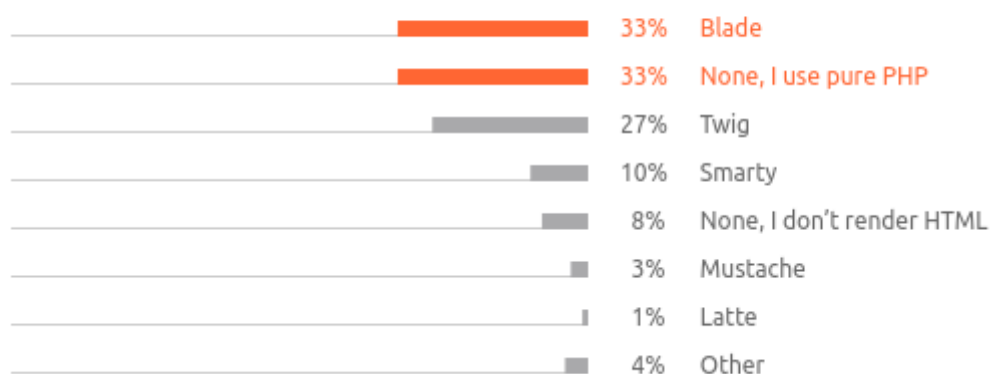
<https://laravel.com>

Algumas informações aqui:

<https://socialcompare.com/es/comparison/popular-php-frameworks>

Template engines mais usados

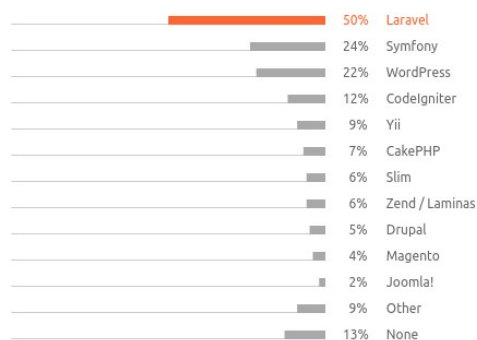
What template engines do you use?



<https://www.jetbrains.com/idea/devecosystem-2020/php/>

Mus Desp Uteis Tools Insp Prog Jobs DN Proj Dia Launch Meeting - Zoom LaraAdm Laravel test Laravel | TDD PHF

Which PHP frameworks and platforms do you regularly use, if any?



Roughly half of the developers we surveyed use Laravel, which makes it one of the 3 most popular platforms, along with Symfony and WordPress.



Uma estatística da JetBrains

<https://trends.google.com.br/trends/explore?date=today-5-y&q=laravel,cakephp,codeigniter,zend,Symfony>

Mus Desp Uteis Tools Insp Prog Jobs DN Proj Dia Launch Meeting - Zoom LaraAdm Laravel test Laravel | TDD PHPUnitLarav

Comparar

laravel

Termo de pesquisa

cakephp

Termo de pesquisa

codeigniter

Termo de pesquisa

zend

Termo de pesquisa

Symfony

Termo de pesquisa

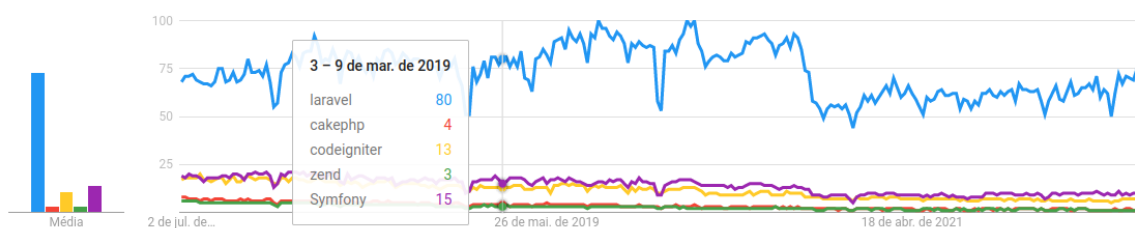
Todo o mundo

Nos últimos 5 anos

Todas as categorias

Pesquisa na Web

Interesse ao longo do tempo



No Google Trends

<https://trends.google.com.br/trends/?geo=BR>

Na grande maioria dos sites onde encontramos comparações de frameworks PHP, o Laravel encontra-se em primeiro lugar.

O laravel tem uma baixa curva de aprendizagem e traz bons padrões de projeto e boas práticas, sem contar que usa a arquitetura MVC. Além dos bons recursos que ele traz, existe uma enorme quantidade de pacotes que estendem suas funcionalidades.

Por ser muito popular encontra-se facilmente muita material sobre o mesmo na internet, o que facilita seu aprendizado além das facilidades oferecidas pelo framework.

Atualmente temos mais de 31 mil projetos com laravel no Github: <https://github.com/topics/laravel>

Introdução

O livro será estruturado em duas partes, uma abordando diretamente a criação do aplicativo, de forma bem simplificada e a outra parte que explicará a primeira, como uma referência.

Os principais recursos usados no aplicativo foram:

- Migrations
- Seeders com faker para geração de dados de teste
- Uso do framework Bootstrap nas views
- Paginação customizada usando Bootstrap
- Busca nas views index
- Gerador de CRUDs
- ACL com Spatie
- Ajuste fino do controle de acesso nas vies
- Laravel Collective
- Localization
- DebugBar
- Customização de mensagens de erro
- Layout customizado
- AdminLTE como dashboard
- Ajustes na view welcome.blade.php e também na home.blade.php
- Criação de alguns commands para o gerenciamento do aplicativo
- Sugestões para adição de novos CRUDs
- Uso do git desde o início e em todas as fases, sincronizando um repositório local com um remoto no GitHub
- Suporte aos SGBDs free e open:
 - MySQL/MariaDb
 - PostgreSQL
 - SQLite

Testado em:

- Linux Mint 20.3
- Windows 10

Caso você seja iniciante no framework é recomendado que pelo menos conheça a documentação oficial ou pelo menos parte dela, especificamente sobre MVC e sobre a criação de um CRUD.

Criaremos um repositório no GH para o livro, chamado livro-laravel9, sem README.md, com licença MIT e private.

Iniciaremos clonando localmente o projeto do Github. Adicionaremos os recursos e testaremos localmente.

Depois, a cada capítulo, efetuaremos commit, sincronizando ambos.

Inicialmente criei o aplicativo em 12 fases. Criei a primeira apenas clonando o repositório:

<https://github.com/savanihd/Laravel-9-roles-and-permissions>

E seguindo o tutorial:

<https://www.itsolutionstuff.com/post/laravel-9-user-roles-and-permissions-tutorialexample.html>

Depois fui adicionando funcionalidades e testando. A cada funcionalidade que gostava criava uma fase, uma pasta para cada uma.

Então resolvi fazer algo mais profissional e usar o próprio git para guardar as versões e poder recuperar com mais praticidade. Além do versionamento também criei branches novas.

Um tutorial resumido com o passo a passo sobre o Git está na seção Bônus.

É importante destacar a importância do terminal atualmente:

- Terminal
- Shell Script, Batchs, PowerShell, etc
- Artisan
- Tinker
- Commands
- Git
- Composer
- etc
- Até o Windows atualmente conta com um terminal: Windows Terminal (baixar pela loja)

Primeira Parte

0 - Introdução ao Laravel

Este livro tem a pretensão de ser prático para a criação de aplicativos com Laravel 9 e com bons recursos.

Por conta de o PHP ser já conhecido por sua inconsistência com nomes alguns programadores mais exigentes migraram para linguagens como Ruby e Python. O framework Laravel também veio suprir esta deficiência do PHP, lidando de forma mais coerente com os nomes de suas propriedades, funções/métodos e classes.

O que é um framework?

Um framework é um facilitador no desenvolvimento de diversas aplicações e, sem dúvida, sua utilização poupa tempo e custos para quem o utiliza, pois de forma mais básica, é um conjunto de bibliotecas utilizadas para criar uma base onde as aplicações são construídas, um otimizador de recursos. Tem como principal objetivo resolver problemas recorrentes com uma abordagem mais genérica. Ele permite ao desenvolvedor focar nos “problemas” da aplicação, não na arquitetura e configurações.

Mais detalhes em:

<https://www.treinaweb.com.br/blog/o-que-e-laravel>

Os frameworks são a base para a criação de sites, unindo ferramentas, funções, recursos e outros detalhes para encurtar o trabalho do profissional na hora de criar uma página online em determinada linguagem de código.

A dúvida que fica para quem está começando a desenvolver sites, aplicativos e APIs e está à procura de um framework para otimizar seu tempo é: qual o melhor framework de PHP?

No mercado, existem diversas opções, como: Symfony, CodeIgniter, Zend, Phalcon, CakePHP e Yii cada uma com suas vantagens e desvantagens. Mas o framework mais popular é, sem dúvidas, o Laravel.

Mais detalhes aqui:

<https://www.tecmundo.com.br/software/223718-laravel-conheca-o-framework-php-utilizado.htm>

Laravel é um framework PHP open source, robusto e fácil de entender. Ele segue um padrão de design MVC. O Laravel reutiliza os componentes existentes de diferentes frameworks, o que ajuda na criação de uma aplicação web. A aplicação web assim concebida é mais estruturada e pragmática.

O Laravel oferece um rico conjunto de funcionalidades que incorporam os recursos básicos de frameworks PHP como CodeIgniter, Yii e outras linguagens de programação como Ruby on Rails assim aumentarão a velocidade do desenvolvimento web.

Se você estiver familiarizado com o core do PHP e com o PHP avançado, o Laravel facilitará suas tarefas. Economiza muito tempo se você planeja desenvolver um site do zero. Além disso, um site construído em Laravel é seguro e evita vários ataques na web.

Detalhes:

https://www.tutorialspoint.com/laravel/laravel_overview.htm

Laravel incorpora muitas das features de alguns frameworks como CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra, Symfony entre outros. Ao invés de criar seus componentes do zero, o Laravel aproveitou alguns componentes já testados de outros softwares, sendo assim, componentes maduros.

Uma das características do Laravel é que ele está constantemente de olho nas novidades do PHP. Um exemplo é que existem integrantes da equipe estudando as novidades das novas versões do PHP para integrar nas novas versões do Laravel. Podemos ver abaixo que isso é perseguido:

<https://github.com/laravel/framework/pulls>

Aqui os releases do framework

<https://github.com/laravel/framework/releases>

Algumas das características do Laravel

- Modulatidade
- Gerenciamento de e-mails com SwiftMailer (<https://laravel.com/docs/8.x/mail>)
- MVC
- Data e hora com Carbon
- Flexível sistema de rotas
- Doctrine para abstração de bancos de dados
- Configurações no .env e no config/. As mensagens de erro em produção são diferentes das mensagens locais
- Schema builder, factories, migrations e seeders
- Template engine com Blender
- Autenticação
- Autorização
- Código simples e bem expressivo
- Convenções sobre configurações

O framework Laravel nos oferece uma estrutura com muitos e bons recursos para a criação de aplicativos, APIs, sites, etc.

Os pacotes criados com Laravel abrem ainda mais o leque do que se pode criar com o mesmo. Ainda esta semana encontrei um pacote que cria e-books com conteúdo markdown no laravel. Realmente a nossa criatividade e conhecimento é o limite.

O Laravel é, basicamente, sobre equipar e capacitar os desenvolvedores. Seu objetivo é fornecer informações claras, código e recursos simples e bonitos que ajudam os desenvolvedores a aprender, iniciar e desenvolver rapidamente, e escreva um código simples, claro e que dure.

“Desenvolvedores felizes fazem o melhor código” está escrito na documentação. “A felicidade do desenvolvedor, do download ao deploy” foi o slogan não oficial por um tempo.

Onde outros frameworks podem ter como objetivo principal a pureza arquitetônica, ou compatibilidade com os objetivos e valores das equipes de desenvolvimento corporativo, o principal recurso do Laravel, seu foco está em servir o desenvolvedor individual.

O Laravel se concentra em "convenção sobre configuração" - o que significa que, se você estiver disposto para usar os padrões do Laravel, você precisará fazer muito menos trabalho do que com outros frameworks que exigem que você declare todas as suas configurações, mesmo se você estiver usando a recomendada configuração. Os projetos criados no Laravel levam menos tempo do que os criados na maioria dos outros frameworks PHP.

Nos bastidores, o Laravel usa muitos padrões de design e à medida que avança. Desde o início, o Laravel o ajuda a codificar ambiente acoplado. A estrutura se concentra no design de classes para ter uma única responsabilidade, evitando a codificação embutida nos módulos de nível superior. Módulos de nível superior não dependem dos módulos de nível inferior, tornando a codificação uma experiência agradável.

Devido a essa flexibilidade, o Laravel se tornou um dos frameworks mais populares da atualidade.

O Laravel pode lidar com seu ciclo de vida de request/response com bastante facilidade. A complexidade dos requests não existe aqui.

Você pode pensar na primeira camada como a autenticação, enquanto o middleware testa se o usuário está autenticado. Se o usuário não estiver autenticado, o usuário é enviado de volta à página de login. Se o usuário efetuar login com sucesso, o usuário deve enfrentar a segunda camada, que pode consistir na validação do token CSRF. o processo continua assim e os casos de uso mais comuns do middleware Laravel que protege seu aplicativo: autenticação, validação de token CSRF, modo de manutenção, e assim por diante. Quando seu aplicativo está no modo de manutenção, uma visualização personalizada é exibida para todos os pedidos.

É importante se acostumar a consultar a documentação oficial do Laravel:

<https://laravel.com/docs/9.x>

O autor do Laravel, Taylor Otwell, resumiu os recursos flexíveis do Laravel da seguinte forma (em uma conferência do PHPWorld em 2014):

- Procure simplicidade
- Configuração mínima
- Sintaxe concisa, memorável e expressiva
- Infraestrutura poderosa para o PHP moderno
- Ótimo para iniciantes e desenvolvedores avançados
- Abraça da comunidade PHP

Versões do Laravel

1.0	June 2011		
2.0	September 2011		
3.0	February 22, 2012		
3.1	March 27, 2012		
3.2	May 22, 2012		
4.0	May 28, 2013	≥ 5.3.0	
4.1	December 12, 2013	≥ 5.3.0	
4.2	June 1, 2014	≥ 5.4.0	
5.0	February 4, 2015	≥ 5.4.0	
5.1 LTS	June 9, 2015	≥ 5.5.9	
5.2	December 21, 2015	≥ 5.5.9	
5.3	August 23, 2016	≥ 5.6.4	
5.4	January 24, 2017	≥ 5.6.4	
5.5 LTS	August 30, 2017	≥ 7.0.0	
5.6	February 7, 2018	≥ 7.1.3	
5.7	September 4, 2018	≥ 7.1.3	
5.8	February 26, 2019	≥ 7.1.3	
6 LTS	September 3, 2019	≥ 7.2.0	
7	March 3, 2020 ^[17]	≥ 7.2.5 ^[18]	
8	September 3, 2020 (TBC)	≥ 7.3.0	
9	February 8, 2022	≥ 8.0.2	
10	February 7, 2023	≥ 8.1.0	

<https://laravel.com/docs/9.x/releases#laravel-9>

Version	Release	Bug Fixes Until	Security Fixes Until
6 (LTS)	September 3rd, 2019	October 5th, 2021	September 3rd, 2022
7	March 3rd, 2020	October 6th, 2020	March 3rd, 2021
8	September 8th, 2020	April 6th, 2021	September 8th, 2021
9	February 8th, 2022	February 8th, 2023	February 8th, 2024
10	February 7th, 2023	August 7th, 2024	February 7th, 2025

Tempo de Suporte das Versões

Para versões LTS, como o 6, são corrigidos bugs por 2 anos e correções de segurança por 3 anos.

Versões que não são LTS, como a 7, 8 e 9, recebem correção de bugs por 7 meses e correções de segurança por um ano.

Início dos frameworks, seguindo o Rails

CakePHP foi o primeiro em 2005, foi seguido pelo Symfony, CodeIgniter, Zend Framework e Kohana (um fork do CodeIgniter). Yii em 2008 e Aura e Slim em 2010 . Em 2011 vieram o FuelPHP e Laravel

Alguns dos principais recursos atuais:

- Instalação facilitada
- Configurações simples no .env
- namespace com PSR-4
- Convenções sobre configurações
- Router
- MVC
- ORM Eloquente
- Validação de dados
- Templates com blade para as views
- Biblioteca para a criação de formulários
- Artisan
- Tinker
- Migrations
- Seeds
- Tratamento de erros
- Autenticação
- Entre outros

Exemplo usando as convenções: produtos

- Router - routes/web.php
- Model - app/Produto.php
- Controller - app/Http/Controllers/ProdutoController.php
- View - resources/views/produtos

Métodos/actions padrões de CRUD em controller:

- index - listagem dos registros
- create - form para adicionar novo registro. Trabalha em conjunto com store para adicionar ao banco
- store - armazena no banco novos registros
- show - mostra um registro na tela
- edit - form para edição de registro. Trabalha em conjunto com update, que armazena no banco
- update - armazena no banco alteração de registro

Fluxo resumido de código de uma aplicação com Laravel

- Um usuário através do Navegador acessa uma rota
- A rota (routes/web.php) geralmente chama um action de um controller
- O controller/action carrega o model respectivo e executa o código do action
- O action processa o código vindo do model
- Então o model efetua a consulta ao banco de dados
- O banco de dados devolve o resultado para o model
- O model devolve para o controller/action
- O controller chama a view respectiva, passando para ela as informações
- Um form ou link de uma view pode chamar outro action do controller através de uma rota
- Na view é recebido pelo template/layout que se integra à view, que mostra para o usuário as informações solicitadas

Dependency Injection – DI

É o processo de instanciar de forma simples uma classe dentro de outra

Exemplo, onde \$request é uma instância de Request

```
public function store(Request $request)
{
    $requestData = $request->all();
    Cliente::create($requestData);
    return redirect('clientes')->with('flash_message', 'Cliente added!');
}
```

É semelhante a

```
public function store()
{
    $request = new Request;
    $requestData = $request->all();
    Cliente::create($requestData);
    return redirect('clientes')->with('flash_message', 'Cliente added!');
}
```

No caso a classe Request precisa ser importada no início: use Illuminate\Http\Request; // Por padrão já vem nos controllers criados com artisan

O jeito nativo com que o laravel manipula os models

Doc em pt-br

<https://laravel-docs-pt-br.readthedocs.io/en/5.0/artisan/>

Dicas

<http://artesaos.github.io/laravel-br-awesome/>

1.1 – Ambiente de Desenvolvimento

O ambiente em que programamos em nosso desktop é de suma importância para que o resultado do nosso trabalho seja mais eficiente e que tenhamos a menor quantidade de conflitos entre nosso desktop e o servidor.

Sabe-se também que a maioria dos programadores PHP usa Windows. Ao meu ver o Laragon (<https://laragon.org>) é o pacote mais completo para ambiente de programação no Windows. Ele já vem com composer, git, notepad++, além dos esperados, apache, php (várias versões), mysql, etc. E bem mais. Quando o instalo no Windows apenas baixo a versão atual do PHP, descompacto e copio a pasta completa para c:\laragon\bin\php.

Mas é claro que somos livres para usar o pacote do qual mais gostamos. O mais importante é que tenhamos um ambiente com todos os requisitos para o laravel 9.

Principais ferramentas para a criação do nosso aplicativo

- Navegador Firefox, Chrome ou outro
- GitHub Desktop
- Editor VSCode (<https://code.visualstudio.com>)

Instalar o Github Desktop

Pois o usaremos durante a criação do aplicativo para clonar o repositório do Github para usar localmente.

E durante a criação do aplicativo estaremos sincronizando o local com o remoto.

Download

<https://github.com/shiftkey/desktop/releases>

For Debian

<https://github.com/shiftkey/desktop/releases/download/release-3.0.1-linux1/GitHubDesktop-linux-3.0.1-linux1.deb>

For Windows

<https://desktop.github.com/>

For Mac - <https://central.github.com/deployments/desktop/desktop/latest/darwin>

Requisitos para a instalação do Laravel 9

É importante que nos acostumemos a consultar a documentação oficial do laravel e de qualquer outro software que estejamos usando. Então apenas indicarei aqui o link para o site oficial:

<https://laravel.com/docs/9.x/deployment#server-requirements>

Lembrando que atualmente é exigida a versão 8.0.2 do PHP e não somente. Veja o link acima.

- PHP >= 8

Extensões:

bcmath
gd
mysql
pgsql
sqlite3
curl
pear
dom
xml (traz o dom)
xsl
xdebug
intl
zip
mbstring
gettext
ctype
fileinfo
json
tokenizer

No Linux os 4 acima fazem parte do php8.1-common, que já é instalado por default

Habilitando openssl no php.ini

No caso do Linux Mint 20.3 com PHP 8.1 (adapte para seu ambiente)

```
sudo nano /etc/php/8.1/apache2/php.ini
```

Localizar
;extension=openssl

E remover o ; do início
Salvar com Ctrl+O e Ctrl+X

Reiniciar o Apache
sudo service apache2 restart

Para visualizar a lista de extensões instaladas:

php -m
ou
php -m | more
Tecle espaço para ver a próxima tela e 'q' para sair
<https://laravel.com/docs/9.x/deployment#server-requirements>

1.2 - Ambiente em Desktop Linux

Meu ambiente usando o Linux Mint 20 é criado praticamente por um script que se encontra na pasta de scripts (cujo link está ao final). O script instala:

- Apache2 com mod_rewrite
- PHP 8.1 com extensões
- MariaDB 15
- composer
- git
- node 16
- e bem mais

Então complemento com Xed, VSCode, adminer, etc.

Como já relatei, instalar o ambiente com pacotes da distribuição tem a vantagem extra de estar sempre com os mesmos atualizados, pois quando aparecem atualizações recebo uma notificação do sistema operacional então autorizo a atualização. É importante estar com os softwares que usamos atualizados, tanto para corrigir bugs quando para ajustes na segurança. Por isso faça o possível, em usando Linux, para usar os repositórios oficiais para a instalação. Isso torna seu sistema mais seguro e com mais recursos.

1.3 – Lagaron no Windows

Alguns dos seus recursos

- notepad++,
- composer,
- cmdr, (mais amigável e produtivo que o prompt do windows)
- git,
- nodejs,
- putty,
- winscp,
- telnet,
- yarn,
- adminer
- Podemos instalar outras versões do php
- Podemos instalar outra versão do mariadb

Mais

- Cria automaticamente virtualhost para cada aplicação - pasta.dev
- Na opção criar rapidamente um website, pode criar: Joomla, Wordpress, Drupal, laravel, cakephp, symfony, lumen, etc

O Laragon tem o NGROK.io integrado! O NGROK te dá uma URL pública para seu amigo ou cliente acessar o aplicativo web em que você está trabalhando. Vantagem de ver em tempo real as alterações e não ter todo aquele trabalho de colocar na hospedagem antecipadamente!

Com o NGROK você consegue liberar o localhost para acesso externo. É bem simples de configurar e mais fácil ainda de usar. Usei recentemente num projeto. Recebi de um colega num grupo do Facebook.

Configurar o phpmyadmin no Linux para ser usado com mysql sem senha

<https://github.com/ribafs/laravel/blob/main/1Introducao/Ambiente/phpmyadmin.md>

1.4 – Algumas Convenções do Laravel

Ao seguirmos as convenções definidas por um framework estamos garantindo boa performance, facilidade de manutenção, baixa curva de aprendizagem ao contratar programadores, além de segurança e escalabilidade.

Um banco de dados que não atende a convenção pode gerar confusão e dificuldade de manutenção do código da aplicação.

Ao não utilizar a convenção de nomenclatura de banco de dados do framework é possível que o programador tenha trabalho para definir em cada model qual é a sua respectiva tabela. Imagine um cenário de 100 models ou mais...

Uma aplicação que segue as convenções do laravel garante o principal objetivo da utilização do framework: produtividade. Com isso também garante uma pequena curva de aprendizagem ao trocar ou expandir a equipe além, claro, dos fatores de segurança, performance e organização que são essenciais para uma aplicação robusta e escalável.

<https://medium.com/@carloscarneiropmw/overview-das-conven%C3%A7%C3%B5es-do-laravel-d2e76b3db38a>

Ajuda muito seguir algumas convenções do Laravel

- Nomes de tabelas e campos - no plural e tudo em minúsculas (snake case) : clientes, name, name_user
- Model - singular e inicial maiúscula (CamelCase) : User, MyClient
- Controller - singular e inicial maiúscula e todas as actions em minúsculas (CamelCase com sufixo Controller) : ClientController, index(), edit(), destroy()
- Views - mesmos nomes das actions do controller, pastas no plural e minúsculas : clients, index.blade.php, edit.blade.php
- Rotas - mesmos nomes das views : posts, users, etc
Route::get('/users', 'UserController@index'); Route::resource('photos', 'PhotoController');
- Usa por padrão os timestamps: created_at e updated_at
- PrimaryKey - id
- Foreign Key - table_id. Ex: client_id
- Preferir usar nomes de variáveis e de arquivos, como também de tabelas e campos em inglês, visto que o framework usa tudo em inglês
- Nomes padrões dos actions dos controllers: index, create, show, store, edit, update, destroy
- <https://laravel.com/docs/9.x/eloquent#eloquent-model-conventions>

Quando não seguimos algumas das convenções podemos dizer ao Laravel para usar o nome que escolhemos. Isso se faz no Model, mas somente é aconselhado fazer em caso de necessidade.

Exemplo:

```
protected $table = "my_table"; protected $primaryKey = "myKey";
```

```
//Tipo da chave primária, sendo string
```

```
protected $keyType = 'string';
```

```
// Para usar nomes diferentes nos timestamps
```

```
const CREATED_AT = 'creation_date';
```

```
const UPDATED_AT = 'last_update';
```

Alerta

É muito comum os iniciantes acharem prático setar validações dentro dos models mas isso significa que o framework somente irá validar os dados no ato de salvar no banco e se por acaso não forem dados válidos teria ocorrido um processamento desnecessário e a aplicação lançará uma exceção — crash — possivelmente deixando o usuário frustrado.

```
<?php

namespace App;
use Illuminate\Database\Eloquent\Model;

class Cliente extends Model
{
    protected $table = 'clientes';
    protected $primaryKey = 'id';
    protected $fillable = ['nome', 'email'];
    public $timestamp = true;

    // Caso use uma conexão diferente da default
    protected $connection = 'connection-name';
}
```

O Laravel é usado por milhares de desenvolvedores todos os dias para criar todos os tipos de sites e aplicativos. Mas, felizmente, existem algumas convenções de nomenclatura muito usadas que são seguidas por muitos desenvolvedores ao nomear as variáveis, métodos e funções de seus projetos Laravel. Aqui está minha visão geral das melhores convenções de nomenclatura para Laravel.

<https://veesworld.com/laravel/laravel-naming-conventions>

1.5 - Criação do Banco de Dados

Geralmente nossos aplicativos usam bancos de dados. E geralmente usamos o MySQL, que atualmente vem sendo substituído pelo seu recente fork MariaDB. Mas o Laravel trabalha também com outros SGBDs:

- MySQL 5.7+
- MariaDb 10.2+
- PostgreSQL 10.0+
- SQLite 3.8.8+
- SQL Server 2017+

Detalhes:

<https://laravel.com/docs/9.x/database>

Podemos criar o banco em qualquer gerenciador de banco que usemos, inclusive na console.

Exemplo:

```
mysql -uroot
```

```
create database laraveldb CHARACTER SET utf8 COLLATE utf8_general_ci;
\q
```

1.6 - Configurações diversas

.env

Este arquivo de configurações existe no raiz do aplicativo.

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

Na pasta **config** existem muitos arquivos de configuração.

Alguns parâmetros que devemos ajustar no .env:

```
APP_NAME="Meu Aplicativo"
APP_ENV=local
APP_DEBUG=true
APP_URL=http://localhost
```

Para checar o tipo de ambiente que você está usando, execute:

```
php artisan env
```

Alterar dados do banco

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

Algumas configurações no config/app.php

```
Locales
    'locale' => 'en',
```

Talém podemos adicionar o nosso (requer outras configurações)

```
'locale' => 'pt_BR',
```

Timezone

```
'timezone' => 'America/Fortaleza',
```

```
APP_ENV=local
```

Para jogar em produção APP_ENV=production

Configurar o Timezone

```
config/app.php
```

```
'timezone' => 'America/Fortaleza',
```

Configuração do ambiente

Geralmente é útil ter valores de configuração diferentes com base no ambiente em que o aplicativo está sendo executado. Por exemplo, você pode querer usar um driver de cache diferente localmente do que você usa em seu servidor de produção.

Para tornar isso fácil, o Laravel utiliza a biblioteca PHP DotEnv. Em uma nova instalação do Laravel, o diretório raiz do seu aplicativo conterá um arquivo .env.example que define muitas variáveis de ambiente comuns. Durante o processo de instalação do Laravel, este arquivo será copiado automaticamente para .env.

O arquivo .env padrão do Laravel contém alguns valores de configuração comuns que podem diferir dependendo se seu aplicativo está sendo executado localmente ou em um servidor web de produção. Esses valores são então recuperados de vários arquivos de configuração do Laravel dentro do diretório de configuração usando a função env do Laravel.

Se você estiver desenvolvendo com uma equipe, talvez queira continuar incluindo um arquivo .env.example em seu aplicativo. Ao colocar valores de espaço reservado no arquivo de configuração de exemplo, outros desenvolvedores em sua equipe podem ver claramente quais variáveis de ambiente são necessárias para executar seu aplicativo.

Um .env básico:

```
APP_NAME=Laravel
```

```
APP_ENV=local
```

```
APP_KEY=base64:xnNCqSFbGLS82wUYO+1GvV5GmHfa0S7us3YCYD5IycU=
```

```
APP_DEBUG=true
```

```
APP_URL=http://crud.test
```

```
LOG_CHANNEL=stack
```

LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug

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

BROADCAST_DRIVER=log
CACHE_DRIVER=file
FILESYSTEM_DISK=local
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

MEMCACHED_HOST=127.0.0.1

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="\${APP_NAME}"

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="\${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="\${PUSHER_APP_CLUSTER}"

Configuração do ambiente

Geralmente é útil ter valores de configuração diferentes com base no ambiente em que o aplicativo está sendo executado. Por exemplo, você pode querer usar um driver de cache diferente localmente do que você usa em seu servidor de produção.

Para tornar isso fácil, o Laravel utiliza a biblioteca PHP DotEnv. Em uma nova instalação do Laravel, o diretório raiz do seu aplicativo conterá um arquivo `.env.example` que define muitas variáveis de ambiente comuns. Durante o processo de instalação do Laravel, este arquivo será copiado automaticamente para `.env`.

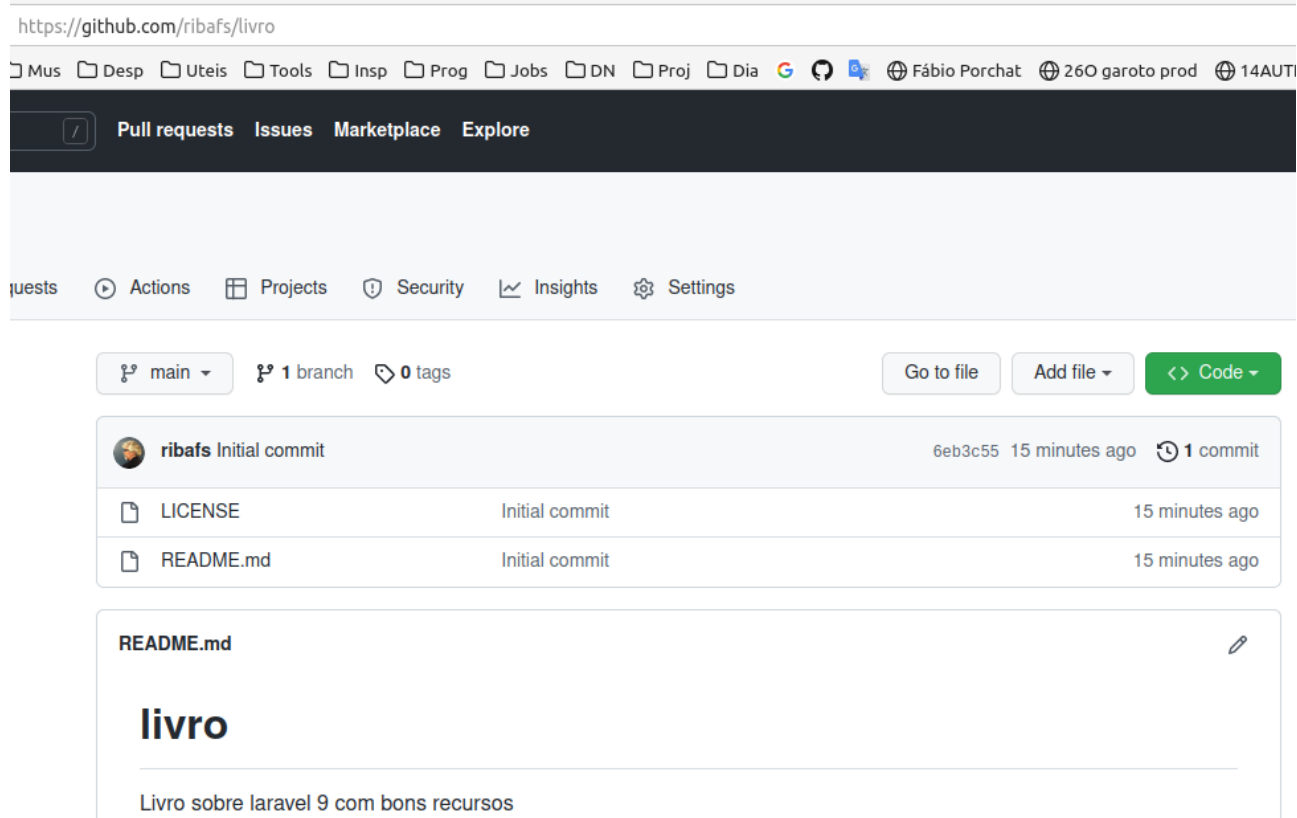
O arquivo `.env` padrão do Laravel contém alguns valores de configuração comuns que podem diferir dependendo se seu aplicativo está sendo executado localmente ou em um servidor web de produção. Esses valores são então recuperados de vários arquivos de configuração do Laravel dentro do diretório de configuração usando a função `env` do Laravel.

Se você estiver desenvolvendo com uma equipe, talvez queira continuar incluindo um arquivo `.env.example` em seu aplicativo. Ao colocar valores de espaço reservado no arquivo de configuração de exemplo, outros desenvolvedores em sua equipe podem ver claramente quais variáveis de ambiente são necessárias para executar seu aplicativo.

1 – Criar repositório e instalar laravel

Criar repositório no Github

Tipo private, sem README.md, com apenas a Licença do tipo MIT e com nome 'livro-laravel9'



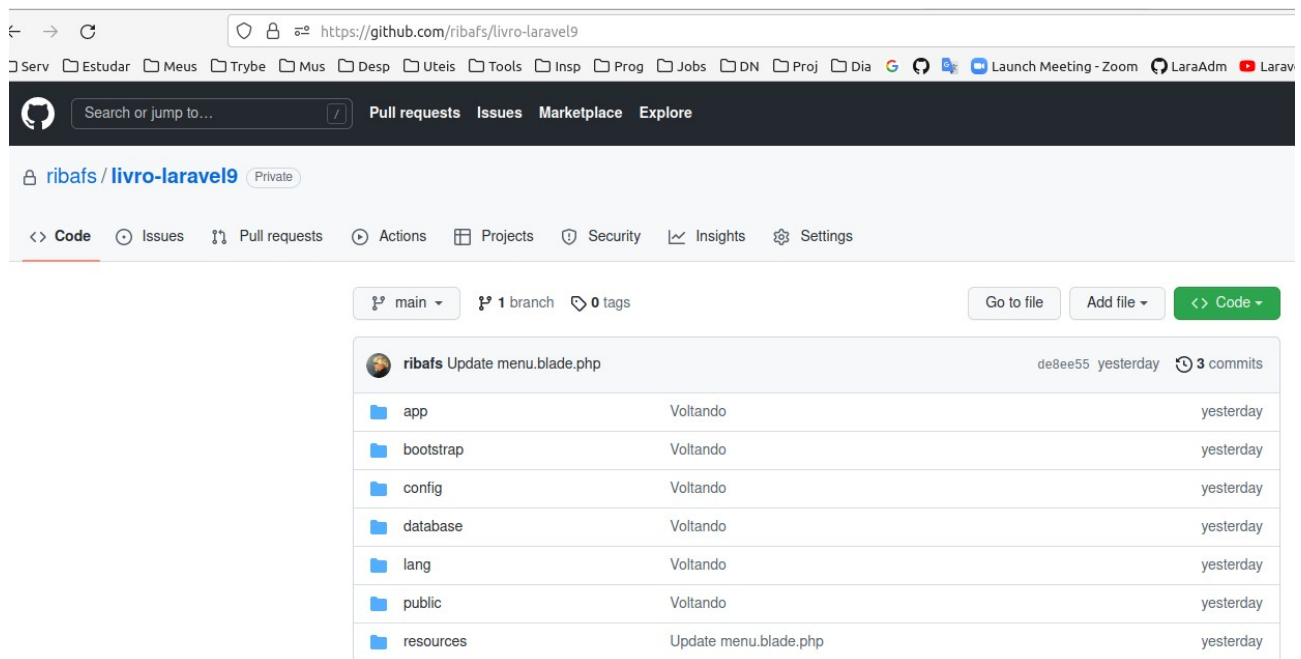
Precisa de Ajuda?

Caso precise de ajuda para criar uma conta no Github, para criar repositório, para usar o Git ou o GitHub Desktop, veja estes pequenos tutoriais:

<https://github.com/ribafs/livrolaravel9>

Alguns tutoriais que criei durante a elaboração deste livro.

No caso inicialmente criei com o nome livro, depois mudei para livro-laravel9. Veja abaixo:



Instalar o laravel 9 localmente na pasta ~/livro

```
cd ~/
laravel new livro
```

Abrir o Github Desktop

Clonar o repositório livro-laravel9

Teclar Ctrl+Shift+F para ver o repositório no gerenciador de arquivos. Ativar visualização em painel duplo.

Copiar tudo da pasta ~/livro, inclusive ocultos para a pasta ~/GitHub/liveo-laravel9

Voltar ao GH Desktop e sincronizar com o Github usando uma mensagem no Summary:

Instalação limpa do laravel 9

Abrir o VSCode na pasta ~/GitHub/liveo-laravel9

Exibir o terminal para instalar pacotes e efetuar alterações:

Criando e usando scaffold da autenticação

```
composer require laravel/ui
php artisan ui bootstrap --auth
```

Criar banco chamado livro

```
mysqladmin -uroot -proot create livro
```

Configurar no .env

`nano .env`

`npm install`

`npm run dev`

`npm install resolve-url-loader@^5.0.0 --save-dev --legacy-peer-deps`

`npm run dev`

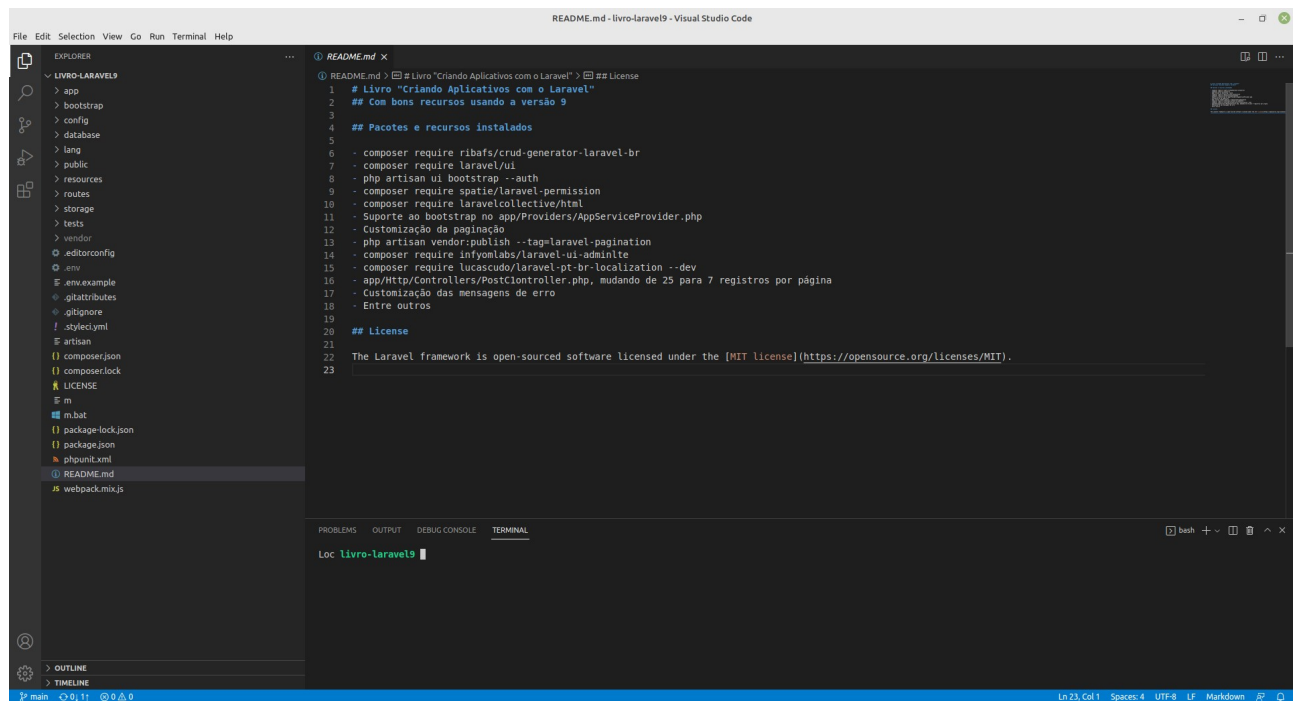
`php artisan serve`

`http://127.0.0.1:8000`

Abrir o Github Desktop e então abrir o VSCode

Teclar Ctrl+Shift+A

Estando com o nosso aplicativo aberto no VSCode, já estamos na pasta ~/Github/livro-laravel9



Customizar o README.md

Clique no arquivo README.md à esquerda

Remova todo o seu conteúdo

E cole apenas as 4 linhas abaixo, sem -- Início -- nem -- Final --:

-- Início --

`# Livro "Criando Aplicativos com o Laravel"`

`## Com bons recursos usando a versão 9`

```

## Pacotes e recursos instalados
- composer require ribafs/crud-generator-laravel-br
- composer require laravel/ui
- php artisan ui bootstrap --auth
- composer require spatie/laravel-permission
- composer require laravelcollective/html
- Suporte ao bootstrap no app/Providers/AppServiceProvider.php
- Customização da paginação
- php artisan vendor:publish --tag=laravel-pagination
- composer require infyomlabs/laravel-ui-adminlte
- composer require lucascudo/laravel-pt-br-localization --dev
- app/Http/Controllers/PostController.php, mudando de 25 para 7 registros por página
- Customização das mensagens de erro
- Entre outros

```

License

```

The Laravel framework is open-sourced software licensed under the [MIT
license](https://opensource.org/licenses/MIT).
-- Final --

```

Ainda no VSCode já podemos observar na listagem de arquivos, que README.md mudou de cor e tem agora um M à sua direita, como também o ícone do Source Control, acima e à esquerda acusa que 1 arquivo foi alterado.

Voltar ao Github Desktop e clicar em Summary e digitar a mensagem

Capítulo 1

Então clique em Commit to main
Depois em Push origin

Se voltarmos ao repositório no GH veremos que as alterações já foram publicadas.

Vamos ao Capítulo 2

2 – Implementar permissions

Veja que temos alterações em nosso projeto git.

- Implementando o pacote de permissões Spatie via terminal do VSCode

Instalar

composer require spatie/laravel-permission

Publicar

php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"

Obs.: quando instalamos um pacote, como este acima e o publicamos, ele cria uma pasta na pasta vendor do aplicativo. No caso criará a pasta spatie e dentro dela uma pasta com o nome do pacote, no nosso caso "laravel-permission". Aí ficam todos os arquivos do pacote.

- Instalando o pacote para lidar com HTML e Forms

composer require laravelcollective/html

- Criar banco e configurar o .env via terminal

Lembrando que meu mysql usa o **root** com senha '**root**'

mysqladmin -uroot -proot create livro

Abrir .env e configurar

nano .env

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

Executar para criar as tabelas

php artisan migrate

Com o comando acima as tabelas básicas serão criadas e também as do pacote permissions do Spatie, entre elas:

roles,
permissions,
model_has_permissions,
model_has_roles,
roles_has_permissions e já relacionadas

Testar

```
php artisan serve  
http://127.0.0.1:8000
```

Praticamente nada que tenha alterado a interface do nosso aplicativo.
Então bola pra frente :)

Voltar ao Github Desktop para sincronizar com o Github

Digitar uma mensagem na caixa Summary, vamos usar o nome do capítulo nas mensagens:

"Capítulo 2"

Clicar em Commit to main

E em Push origin

Então podemos checar as alterações no repositório livro-laravel9 do Github. Podemos verificar a mensagem deste último commit nos arquivos alterados.

Obs.: Idealmente cada alteração deve ser commitada com uma respectiva mensagem para que possamos restaurar cada ponto.

Vamos ao Capítulo 3

3 – Ajustes na ACL

- Editar o model User para adaptar a ACL do Spatie. Remova todo o conteúdo existente e adicione o conteúdo abaixo

app/Models/User.php

```
<?php
namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable, HasRoles;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array
     */
}
```

```
protected $casts = [
    'email_verified_at' => 'datetime',
];
}
```

Adicionar os middlewares do Spatie ao Kernel

```
app/Http/Kernel.php
....
protected $routeMiddleware = [
    ....
    'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,
    'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,
    'role_or_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class,
]
....
```

- Atualizar as rotas

```
routes/web.php

<?php
use Illuminate\Support\Facades\Route;

use App\Http\Controllers\HomeController;
use App\Http\Controllers\RoleController;
use App\Http\Controllers\UserController;

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

Auth::routes();

Route::get('/home', [HomeController::class, 'index'])->name('home');

Route::group(['middleware' => ['auth']], function() {
    Route::resource('roles', RoleController::class);
    Route::resource('users', UserController::class);
});
```

Estas rotas são para os CRUDs que o Spatie já traz. Depois adicionaremos as rotas para os CRUDs que criaremos, que são posts e comments. Ainda está faltando alguns componentes do Spatie, complementemos...

Vamos criar o controller User

app/Http/Controllers/UserController.php

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Models\User;
use Spatie\Permission\Models\Role;
use DB;
use Hash;
use Illuminate\Support\Arr;

class UserController extends Controller
{

    function __construct()
    {
        $this->middleware('permission:user-list|user-create|user-edit|user-delete', ['only' =>
['index','store']]);
        $this->middleware('permission:user-create', ['only' => ['create','store']]);
        $this->middleware('permission:user-edit', ['only' => ['edit','update']]);
        $this->middleware('permission:user-delete', ['only' => ['destroy']]);
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request)
    {
        $data = User::orderBy('id','DESC')->paginate(5);
        return view('users.index',compact('data'))
            ->with('i', ($request->input('page', 1) - 1) * 5);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        $roles = Role::pluck('name','name')->all();
        return view('users.create',compact('roles'));
```

```

}

/**
 * Store a newly created resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @return \Illuminate\Http\Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'name' => 'required',
        'email' => 'required|email|unique:users,email',
        'password' => 'required|same:confirm-password',
        'roles' => 'required'
    ]);

    $input = $request->all();
    $input['password'] = Hash::make($input['password']);

    $user = User::create($input);
    $user->assignRole($request->input('roles'));

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

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $user = User::find($id);
    return view('users.show',compact('user'));
}

/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $user = User::find($id);

```

```

    $roles = Role::pluck('name','name')->all();
    $userRole = $user->roles->pluck('name','name')->all();

    return view('users.edit',compact('user','roles','userRole'));
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function update(Request $request, $id)
{
    $this->validate($request, [
        'name' => 'required',
        'email' => 'required|email|unique:users,email,.'. $id,
        'password' => 'same:confirm-password',
        'roles' => 'required'
    ]);

    $input = $request->all();
    if(!empty($input['password'])){
        $input['password'] = Hash::make($input['password']);
    }else{
        $input = Arr::except($input,array('password'));
    }

    $user = User::find($id);
    $user->update($input);
    DB::table('model_has_roles')->where('model_id',$id)->delete();

    $user->assignRole($request->input('roles'));

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

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    User::find($id)->delete();
}

```

```

        return redirect()->route('users.index')
            ->with('success','User deleted successfully');
    }
}

```

Agora o Role controller

app/Http/Controllers/RoleController.php

```

<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use DB;

class RoleController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    function __construct()
    {
        $this->middleware('permission:role-list|role-create|role-edit|role-delete', ['only' =>
['index','store']]);
        $this->middleware('permission:role-create', ['only' => ['create','store']]);
        $this->middleware('permission:role-edit', ['only' => ['edit','update']]);
        $this->middleware('permission:role-delete', ['only' => ['destroy']]);
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request)
    {
        $roles = Role::orderBy('id','DESC')->paginate(5);
        return view('roles.index',compact('roles'))
            ->with('i', ($request->input('page', 1) - 1) * 5);
    }

    /**
     * Show the form for creating a new resource.

```



```

*
* @return \Illuminate\Http\Response
*/
public function create()
{
    $permission = Permission::get();
    return view('roles.create',compact('permission'));
}

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

    $role = Role::create(['name' => $request->input('name')]);
    $role->syncPermissions($request->input('permission'));

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

/**
 * Display the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function show($id)
{
    $role = Role::find($id);
    $rolePermissions =
Permission::join("role_has_permissions","role_has_permissions.permission_id","=", "permissions.
id")
    ->where("role_has_permissions.role_id",$id)
    ->get();

    return view('roles.show',compact('role','rolePermissions'));
}

/**
 * Show the form for editing the specified resource.

```

```

*
* @param int $id
* @return \Illuminate\Http\Response
*/
public function edit($id)
{
    $role = Role::find($id);
    $permission = Permission::get();
    $rolePermissions = DB::table("role_has_permissions")-
>where("role_has_permissions.role_id",$id)
    ->pluck('role_has_permissions.permission_id','role_has_permissions.permission_id')
    ->all();

    return view('roles.edit',compact('role','permission','rolePermissions'));
}

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

    $role = Role::find($id);
    $role->name = $request->input('name');
    $role->save();

    $role->syncPermissions($request->input('permission'));

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

/**
 * Remove the specified resource from storage.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function destroy($id)
{
    DB::table("roles")->where('id',$id)->delete();
}

```

```

        return redirect()->route('roles.index')
        ->with('success','Role deleted successfully');
    }
}

```

Agora vamos adicionar as views/blade correspondentes a estes controllers que acabamos de criar.

Antes disso vamos instalar o pacote que implementa o dashboard do AdminLTE

Implementação do AdminLTE

Ctrl+C para interromper a execução do aplicativo e executar:

```

composer require infyomlabs/laravel-ui-adminlte
php artisan ui adminlte --auth
npm install && npm run dev

```

Ícones encontrados aqui:

<https://fontawesome.com/v5/search>

Ele substituiu o resources/views/layouts/sidebar.blade.php por um dele, que chama o menu.blade.php, criado por ele.

Editei o menu.blade.php e adaptei para que ficasse parecido com o antigo sidebar.blade.php.

Remover o conteúdo existente em
resources/views/layouts/menu.blade.php

E inserir este abaixo:

```

<li class="nav-item">
    <a href="{{ route('home') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
        <i class="nav-icon fas fa-home"></i>
        <p>Home</p>
    </a>
    @role('User')
    <a href="{{ url('posts') }}" class="nav-link {{ Request::is('posts') ? 'active' : '' }}">
        <i class="nav-icon fas fa-people-arrows"></i>
        <p>Posts</p>
    </a>
    @endrole
    @hasanyrole('Super|Manager')
    <a href="{{ url('posts') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
        <i class="nav-icon fas fa-people-arrows"></i>
        <p>Posts</p>
    </a>

```

```

<a href="{{ url('comments') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
  <i class="nav-icon fas fa-apple-alt"></i>
  <p>Comments</p>
</a>
@endhasanyrole
  @hasanyrole('Super|Admin')
  <a href="{{ url('roles') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
    <i class="nav-icon fas fa-balance-scale-left"></i>
    <p>Roles</p>
  </a>
  <a href="{{ url('users') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
    <i class="nav-icon far fa-grin-beam"></i>
    <p>Users</p>
  </a>
@endhasanyrole
</li>

```

Crie o arquivo public/css/style.css contendo:

```

* {
  text-align: center;
}

```

```

a:link {
  color: red;
}

```

```

a:visited {
  color: #b2ac6c;
}

```

```

a:hover {
  color: #0d8405;
}

```

```

a:active {
  color: blue;
}

```

```

.bloco{
  display: flex;
}

```

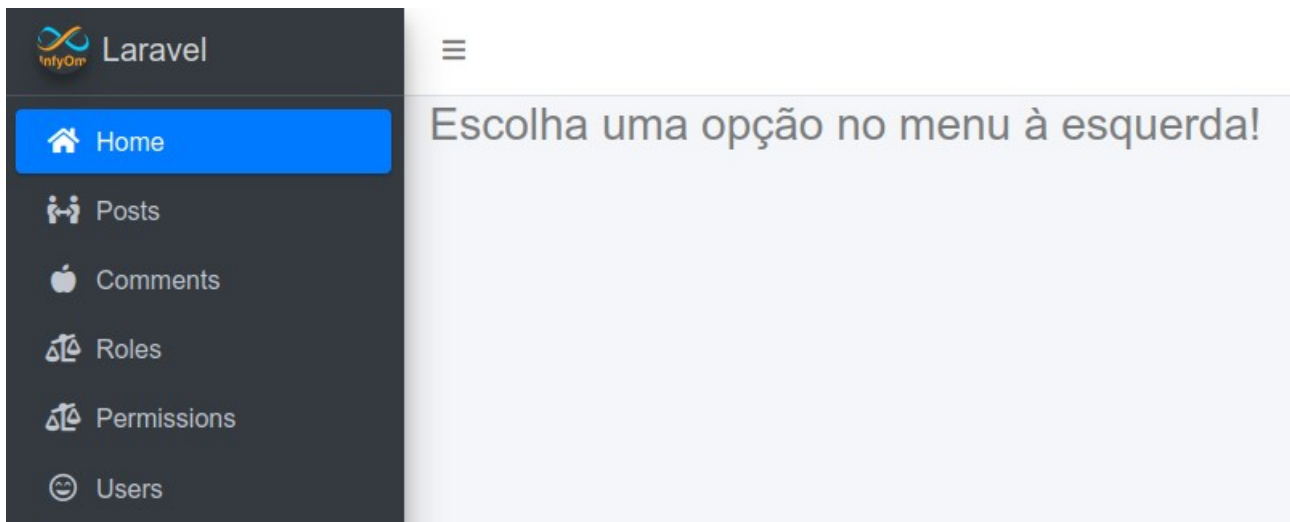
```

.login{
  padding: 50px;
}

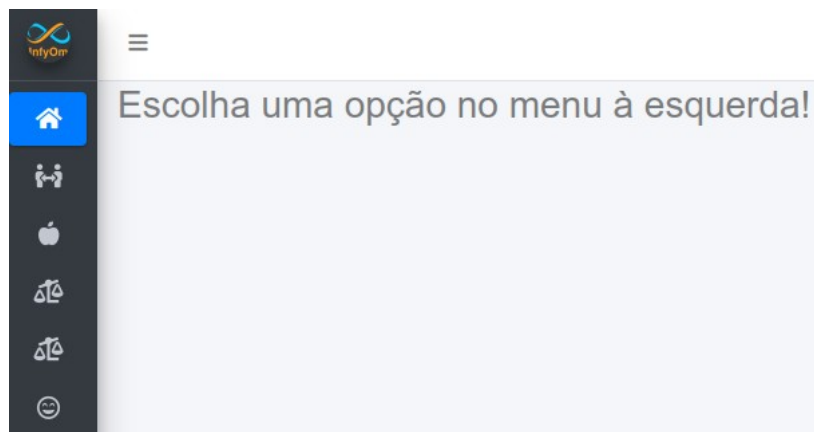
```

```
h1 {
    padding: 100px;
    margin: 0 auto;
    text-align: center;
    color: #8e6348;
}
```

Veja o menu do AdminLTE com seu dashboard



Quando clicamos no ícone do hamburger para mobile então o menu é recolhido assim:



- **Atualize a view welcome.blade.php para:**

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

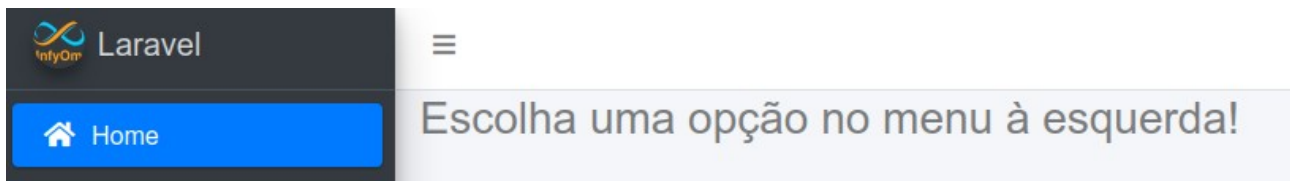
    <title>Aplicativo Completo com Laravel 9</title>
    <link href="{{ mix('css/app.css') }}" rel="stylesheet">
```


E também a `home.blade.php`:

```
@extends('layouts.app')

@section('content')
    <div class="container-fluid">
        <h3 class="text-black-50">Escolha uma opção no menu à esquerda!</h3>
    </div>
@endsection
```

Veja a home



Agora atualizar o layout

resources/views/layouts/app.blade.php

Removeremos o conteúdo da `app.blade.php` existente e adicionaremos o conteúdo abaixo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ config('app.name') }}</title>
    <meta content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no"
name='viewport'>

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.14.0/css/
all.min.css"
        integrity="sha512-
1PKOgIY59xJ8Co8+NE6FZ+LOAZKjy+KY8iq0G4B3CyeY6wYHN3yt9PW0XpSriVlkMXe40PTKn
XrLnZ9+fkDaog=="
        crossorigin="anonymous"/>

    <link href="{{ mix('css/app.css') }}" rel="stylesheet">
    <script
        src="https://code.jquery.com/jquery-3.4.1.js"
        integrity="sha256-WpOohJOqMqqyKL9FccASB900KwACQJpFTUBLTYOVvVU="
        crossorigin="anonymous"></script>

    @yield('third_party_stylesheets')
```

```

    @stack('page_css')
</head>

<body class="hold-transition sidebar-mini layout-fixed">
<div class="wrapper">
    <!-- Main Header -->
    <nav class="main-header navbar navbar-expand navbar-white navbar-light">
        <!-- Left navbar links -->
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" data-widget="pushmenu" href="#" role="button"><i class="fas fa-
bars"></i></a>
            </li>
        </ul>

        <ul class="navbar-nav ml-auto">
            <li class="nav-item dropdown user-menu">
                <a href="#" class="nav-link dropdown-toggle" data-toggle="dropdown">
                    
                    <span class="d-none d-md-inline">{{ Auth::user()->name }}</span>
                </a>
                <ul class="dropdown-menu dropdown-menu-lg dropdown-menu-right">
                    <!-- User image -->
                    <li class="user-header bg-primary">
                        
                        <p>
                            {{ Auth::user()->name }}
                            <small>Member since {{ Auth::user()->created_at->format('M. Y') }}</small>
                        </p>
                    </li>
                    <!-- Menu Footer-->
                    <li class="user-footer">
                        <a href="#" class="btn btn-default btn-flat">Profile</a>
                        <a href="#" class="btn btn-default btn-flat float-right"
                            onclick="event.preventDefault(); document.getElementById('logout-
form').submit();">
                            Sign out
                        </a>
                        <form id="logout-form" action="{{ route('logout') }}" method="POST" class="d-
none">
                            @csrf
                        </form>
                    </li>
                </ul>
            </li>
        </ul>
    </div>

```



```
</ul>
</nav>
```

```
<!-- Left side column. contains the logo and sidebar -->
@include('layouts.sidebar')
```

```
<!-- Content Wrapper. Contains page content -->
<div class="content-wrapper">
  <section class="content">
    @yield('content')
  </section>
</div>
```

```
<!-- Main Footer -->
<footer class="main-footer">
  <div class="float-right d-none d-sm-block">
    <b>Version</b> 3.0.5
  </div>
  <strong>Copyright &copy; 2014-2022 <a
href="https://adminlte.io">AdminLTE.io</a>.</strong> All rights
  reserved.
</footer>
</div>
```

```
<script src="{{ mix('js/app.js') }}" defer></script>
```

```
@yield('third_party_scripts')
```

```
@stack('page_scripts')
</body>
</html>
```

Agora adicionaremos as views para Users:

Criaremos a pasta resources/views/users

E dentro dela criaremos as views

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

- create.blade.php

```
@extends('layouts.app')
@section('content')
<div class="container">
```

```

<div class="card">
<div class="row">
  @include('layouts.sidebar')
  <div class="col-lg-12 margin-tb">
    <div class="pull-left">
      <h2>Create New User</h2>
    </div>
    <div class="pull-right">
      <a class="btn btn-primary" href="{{ route('users.index') }}"> Voltlar</a>
    </div>
  </div>
</div>
@if (count($errors) > 0)
  <div class="alert alert-danger">
    <strong>Whoops!</strong> There were some problems with your input.<br><br>
    <ul>
      @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
      @endforeach
    </ul>
  </div>
@endif
{!! Form::open(array('route' => 'users.store','method'=>'POST')) !!}
<div class="row">
  <div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
      <strong>Name:</strong>
      {!! Form::text('name', null, array('placeholder' => 'Name','class' => 'form-control')) !!}
    </div>
  </div>
  <div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
      <strong>Email:</strong>
      {!! Form::text('email', null, array('placeholder' => 'Email','class' => 'form-control')) !!}
    </div>
  </div>
  <div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
      <strong>Password:</strong>
      {!! Form::password('password', array('placeholder' => 'Password','class' => 'form-
control')) !!}
    </div>
  </div>
  <div class="col-xs-12 col-sm-12 col-md-12">
    <div class="form-group">
      <strong>Confirm Password:</strong>
      {!! Form::password('confirm-password', array('placeholder' => 'Confirm Password','class'
=> 'form-control')) !!}
    </div>
  </div>

```

```

</div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
  <div class="form-group">
    <strong>Role:</strong>
    {!! Form::select('roles[]', $roles,[], array('class' => 'form-control','multiple')) !!}
  </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12 text-center">
  <button type="submit" class="btn btn-primary">Submit</button>
</div>
</div>
</div>
{!! Form::close() !!}
@endsection

```

Ctrl+S para salvar

No momento não estamos preocupados com a ACL, com o controle de acesso, pois estaremos ajustando isso mais tarde. Embora o comportamento default dos controllers com o método `__construct()` do Spatie já nos garante controle de acesso.

Create New User

Name:

Email:

Password:

Confirm Password:

Role:

- Admin
- Manager
- Super**
- User

- edit.blade.php

```

@extends('layouts.app')
@section('content')
<div class="container">
<div class="card">
<div class="row">
    @include('layouts.sidebar')
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Edit New User</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('users.index') }}"> Voltlar</a>
        </div>
    </div>
</div>
@if (count($errors) > 0)
<div class="alert alert-danger">
    <strong>Whoops!</strong> There were some problems with your input.<br><br>
    <ul>
        @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
        @endforeach
    </ul>
</div>
@endif
{!! Form::model($user, ['method' => 'PATCH','route' => ['users.update', $user->id]]) !!}
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Name:</strong>
            {!! Form::text('name', null, array('placeholder' => 'Name','class' => 'form-control')) !!}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Email:</strong>
            {!! Form::text('email', null, array('placeholder' => 'Email','class' => 'form-control')) !!}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Password:</strong>
            {!! Form::password('password', array('placeholder' => 'Password','class' => 'form-control')) !!}
        </div>
    </div>
</div>

```

```

<div class="col-xs-12 col-sm-12 col-md-12">
  <div class="form-group">
    <strong>Confirm Password:</strong>
    {!! Form::password('confirm-password', array('placeholder' => 'Confirm Password','class'
=> 'form-control')) !!}
  </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12">
  <div class="form-group">
    <strong>Role:</strong>
    {!! Form::select('roles[]', $roles,$userRole, array('class' => 'form-control','multiple')) !!}
  </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12 text-center">
  <button type="submit" class="btn btn-primary">Submit</button>
</div>
</div>
</div>
{!! Form::close() !!}
@endsection

```

Edit New User

Voltar

Name:

Email:

Password:

Confirm Password:

Role:

Admin
Manager
Super
User

- index.blade.php

```

@extends('layouts.app')
@section('content')
<div class="container">
<div class="card">
<div class="row">
  @include('layouts.sidebar')

```

```

<div class="col-lg-12 margin-tb">
  <div class="pull-left">
    <h2>Users Management</h2>
  </div>
  <div class="pull-right">
    <a class="btn btn-success" href="{{ route('users.create') }}"> Create New User</a>
  </div>
</div>
</div>
@if ($message = Session::get('success'))
<div class="alert alert-success">
  <p>{{ $message }}</p>
</div>
@endif
<table class="table table-bordered">
<tr>
  <th>No</th>
  <th>Name</th>
  <th>Email</th>
  <th>Roles</th>
  <th width="280px">Action</th>
</tr>
@foreach ($data as $key => $user)
<tr>
  <td>{{ ++$i }}</td>
  <td>{{ $user->name }}</td>
  <td>{{ $user->email }}</td>
  <td>
    @if(!empty($user->getRoleNames()))
      @foreach($user->getRoleNames() as $v)
        <label class="badge badge-success">{{ $v }}</label>
      @endforeach
    @endif
  </td>
  <td>
    <a class="btn btn-info" href="{{ route('users.show',$user->id) }}">Show</a>
    <a class="btn btn-primary" href="{{ route('users.edit',$user->id) }}">Edit</a>
    {!! Form::open(['method' => 'DELETE','route' => ['users.destroy', $user->id], 'style' => 'display:inline']) !!}
    {!! Form::submit('Delete', ['class' => 'btn btn-danger']) !!}
    {!! Form::close() !!}
  </td>
</tr>
@endforeach
</table>
</div>
{!! $data->render() !!}
@endsection

```

Users Management				Create New User
No	Name	Email	Roles	Action
1	User User	user@mail.org	User	Show Edit Delete
2	Manager User	manager@mail.org	Manager	Show Edit Delete
3	Admin User	admin@mail.org	Admin	Show Edit Delete
4	Super User	super@mail.org	Super	Show Edit Delete

- show.blade.php

```

@extends('layouts.app')
@section('content')
<div class="container">
<div class="card">
<div class="container">
<div class="row">
    @include('layouts.sidebar')
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2> Show User</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('users.index') }}"> Voltlar</a>
        </div>
    </div>
</div>
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Name:</strong>
            {{ $user->name }}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Email:</strong>
            {{ $user->email }}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Roles:</strong>
            @if(!empty($user->getRoleNames()))
                @foreach($user->getRoleNames() as $v)

```

```
<label class="badge badge-success">{{ $v }}</label>
@endforeach
@endif
</div>
</div>
</div>
</div>
</div>
@endsection
```

Show User

[Voltar](#)

Name: Super User

Email: super@mail.org

Roles: Super

A tela de confirmação de exclusão

É importante que ao remover um registro solicitemos a confirmação

Posts

+ Adicionar novo

Busca...

ID	Name	Ações
1	Dr. James Valentin de Souza	Ver Editar Excluir
2	Sr. Márcio Ferreira Madeira Filho	Ver Editar Excluir
3	Thalissa Souza	Ver Editar Excluir
4	Sra. Gisela R	Ver Editar Excluir
5	Daniel Reis F	Ver Editar Excluir
6	Betina Ortiz Filho	Ver Editar Excluir
7	Giovana Sheila Lira	Ver Editar Excluir

Mostrando 1 até 7 de 100 resultados

Primeira 1 2 3 4 5 6 7 8 9 10 ... 14 15 Próxima

127.0.0.1:8001

Confirma exclusão?

Cancelar OK

Agora criaremos a pasta resources/views/roles

E dentro dela as views:

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

- create.blade.php

```
@extends('layouts.app')
@section('content')
<div class="container">
<div class="card">
<div class="container">
<div class="row">
    @include('layouts.sidebar')

    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Create New Role</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('roles.index') }}"> Voltar</a>
        </div>
    </div>
</div>
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <strong>Whoops!</strong> There were some problems with your input.<br><br>
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
{!! Form::open(array('route' => 'roles.store','method'=>'POST')) !!}
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Name:</strong>
            {!! Form::text('name', null, array('placeholder' => 'Name','class' => 'form-control')) !!}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <div>Marcar todas <input type="checkbox" id="selectAll"/></div>
        </div>
    </div>
</div>
```

```

        <strong>Permission:</strong>
        <br/>
        @foreach($permission as $value)
            <label>{{ Form::checkbox('permission[]', $value->id, false, array('class' => 'name
all')) }}
            {{ $value->name }}</label>
        <br/>
        @endforeach
    </div>
</div>
<div class="col-xs-12 col-sm-12 col-md-12 text-center">
    <button type="submit" class="btn btn-primary">Submit</button>
</div>
</div>
</div>
{!! Form::close() !!}

<script>
$("#selectAll").click(function () {
    $(".all").prop('checked', $(this).prop('checked'));
});
</script>
@endsection

```

- edit.blade.php

```

@extends('layouts.app')
@section('content')
<div class="container">
<div class="card">
<div class="container">
<div class="row">
    @include('layouts.sidebar')
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Edit Role</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('roles.index') }}"> Voltlar</a>
        </div>
    </div>
</div>
</div>
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <strong>Whoops!</strong> There were some problems with your input. <br> <br>
        <ul>

```

```

    @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
    @endforeach
</ul>
</div>
@endif
{!! Form::model($role, ['method' => 'PATCH','route' => ['roles.update', $role->id]]) !!}
<div class="row">
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <strong>Name:</strong>
            {!! Form::text('name', null, array('placeholder' => 'Name','class' => 'form-control')) !!}
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12">
        <div class="form-group">
            <div>Marcar todas <input type="checkbox" id="selectAll"/></div>
            <strong>Permission:</strong>
            <br/>
            @foreach($permission as $value)
                <label>{{ Form::checkbox('permission[]', $value->id, in_array($value->id,
$rolePermissions) ? true : false, array('class' => 'name all')) }}
                {{ $value->name }}</label>
            <br/>
            @endforeach
        </div>
    </div>
    <div class="col-xs-12 col-sm-12 col-md-12 text-center">
        <button type="submit" class="btn btn-primary">Submit</button>
    </div>
</div>
</div>
{!! Form::close() !!}

<script>
$("#selectAll").click(function () {
    $(".all").prop('checked', $(this).prop('checked'));
});
</script>
@endsection

```

- index.blade.php

```

@extends('layouts.app')
@section('content')
<div class="container">
<div class="card">
<div class="row">
    @include('layouts.sidebar')
    <div class="col-lg-12 margin-tb">
        <div class="pull-left">
            <h2>Role Management</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-success" href="{{ route('roles.create') }}"> Create New Role</a>
        </div>
    </div>
</div>
</div>
@if ($message = Session::get('success'))
    <div class="alert alert-success">
        <p>{{ $message }}</p>
    </div>
@endif

<table class="table table-bordered">
    <tr>
        <th>No</th>
        <th>Name</th>
        <th width="280px">Action</th>
    </tr>
    @foreach ($roles as $key => $role)
        <tr>
            <td>{{ ++$i }}</td>
            <td>{{ $role->name }}</td>
            <td>
                <a class="btn btn-info" href="{{ route('roles.show',$role->id) }}">Show</a>
                <a class="btn btn-primary" href="{{ route('roles.edit',$role->id) }}">Edit</a>
                {!! Form::open(['method' => 'DELETE','route' => ['roles.destroy', $role->id], 'style' => 'display:inline']) !!}
                {!! Form::submit('Delete', ['class' => 'btn btn-danger']) !!}
                {!! Form::close() !!}
            </td>
        </tr>
    @endforeach
</table>
</div>
{!! $roles->render() !!}
@endsection

```

- show.blade.php

```

@extends('layouts.app')
@section('content')

<div class="container">
<div class="card">
<div class="row">
    @include('layouts.sidebar')
    <div class="col-md-12">
        <div class="pull-left">
            <h2> Show Role</h2>
        </div>
        <div class="pull-right">
            <a class="btn btn-primary" href="{{ route('roles.index') }}"> Voltar</a>
        </div>

        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Name:</strong>
                {{ $role->name }}
            </div>
        </div>
        <div class="col-xs-12 col-sm-12 col-md-12">
            <div class="form-group">
                <strong>Permissions:</strong>
                @if(!empty($rolePermissions))
                    @foreach($rolePermissions as $v)
                        <label class="label label-success">{{ $v->name }},</label>
                    @endforeach
                @endif
            </div>
        </div>
    </div>
</div>
</div>
</div>

@endsection

```

Script que marca todos os checkboxes de roles

- Fiz uma pesquisa e criei um script que marca todos os checkboxes de roles. Veja que estão em:

resources/views/roles/create.blade.php
resources/views/roles/edit.blade.php

Edit Role

Voltar

Name:

Marcar todas ☐

Permission:

- ☒ role-list
- ☒ role-create
- ☒ role-edit
- ☒ role-delete
- ☒ user-list
- ☒ user-create
- ☒ user-edit
- ☒ user-delete
- ☒ permission-list
- ☒ permission-create
- ☒ permission-edit
- ☒ permission-delete
- ☒ post-list
- ☒ post-create
- ☒ post-edit
- ☒ post-delete
- ☒ comment-list
- ☒ comment-create
- ☒ comment-edit
- ☒ comment-delete

Submit

Ao clicar pela primeira vez em Marcar todos ele marca todos. Clicando novamente ele desmarca todos.

A jQuery foi adicionada ao app.blade.php

```
npm install
npm run dev
```

Testar

```
php artisan serve
http://127.0.0.1:8000/login
```

Ainda não temos os usuários com seus respectivos privilégios de acesso, mas já temos uma welcome customizada.

Bola pra frente

Voltemos ao Github Desktop e sincronizemos com o repositório remoto.

Inserir uma mensagem em Summary "Capítulo 3" e clicar em Commit e em Push origin

Vamos ao Capítulo 4

4 – Criar os Seeders

- Agora criaremos alguns seeders, que popularão as tabelas para que possamos testar melhor nosso aplicativo

Voltar ao terminal do VSCode

Executar

```
php artisan make:seeder PermissionsSeeder
```

Este define as permissões iniciais. Veja que já adicionamos para post e comment, mas estes serão criadas em seguida

Abrir o arquivo gerado database/seeder/PermissionsSeeder.php

E substituir o código existente por este abaixo:

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Spatie\Permission\Models\Permission;

class PermissionsSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $permissions = [
            'role-list',
            'role-create',
            'role-edit',
            'role-delete',
            'user-list',
            'user-create',
            'user-edit',
            'user-delete',
            'post-list',
            'post-create',
            'post-edit',
            'post-delete',
            'comment-list',
```

```

        'comment-create',
        'comment-edit',
        'comment-delete'
    ];

    foreach ($permissions as $permission) {
        Permission::create(['name' => $permission]);
    }
}
}

```

Jogar na tabela:

```
php artisan db:seed --class=PermissionsSeeder
```

Importante

Aqui criamos um padrão. Ao meu ver padrões não são leis que devem rigorosamente ser seguidos, mas algo que nos ajuda a trabalhar. Vejamos: o padrão nas permissões é assim

nome_tabela_singular-nome_view. Exemplo:

role-list (neste caso diz respeito ao acesso às views index e show da tabela roles). Podemos mudar isso, mas é bom seguir um padrão, pois assim evita erros e fica mais fácil de memorizar.

```
php artisan make:seeder RolesSeeder
```

- **database/seeder/RolesSeeder.php** - definição das roles

```

<?php
namespace Database\Seeders;

use Spatie\Permission\Models\Role;
use Illuminate\Database\Seeder;

class RolesSeeder extends Seeder
{
    public function run()
    {
        $roles = [ 'Super', 'Admin', 'Manager', 'User' ];

        foreach ($roles as $role) {
            Role::create([
                'name' => $role,
                'guard_name' => 'web'
            ]);
        }
    }
}

```


Jogar na tabela:

```
php artisan db:seed --class=RolesSeeder
```

Criar o UsersSeeder

```
php artisan make:seeder UsersSeeder
```

- database/seeder/UsersSeeder.php - Definição dos usuários

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\User;
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;
use Illuminate\Support\Facades\DB;

class UsersSeeder extends Seeder
{
    public function run()
    {
        // Super
        $super = User::create([
            'name' => 'Super User',
            'email' => 'super@mail.org',
            'password' => bcrypt('123456')
        ]);

        $role = Role::findByName('Super');
        $permissions = Permission::pluck('name','name')->all();
        $role->syncPermissions($permissions);
        $super->assignRole([$role->name]);

        // Admin
        $adminu = User::create([
            'name' => 'Admin User',
            'email' => 'admin@mail.org',
            'password' => bcrypt('123456')
        ]);

        $admins = ['role-list','role-create','role-edit','role-delete','user-list','user-create','user-
edit','user-delete'];
        $radmin = Role::findByName('Admin');
        $radmin->syncPermissions($admins);
        $adminu->assignRole([$radmin->name]);

        // Manager
```

```

$manageru = User::create([
    'name' => 'Manager User',
    'email' => 'manager@mail.org',
    'password' => bcrypt('123456')
]);

$managers = ['post-list', 'post-create', 'post-edit', 'post-delete', 'comment-list', 'comment-
create', 'comment-edit', 'comment-delete'];
$rmanager = Role::findByName('Manager');
$rmanager->syncPermissions($managers);
$manageru->assignRole([$rmanager->name]);

// User
$useru = User::create([
    'name' => 'User User',
    'email' => 'user@mail.org',
    'password' => bcrypt('123456')
]);

$users = ['post-list'];
$ruser = Role::findByName('User');
$ruser->syncPermissions($users);
$useru->assignRole([$ruser->name]);
}
}

```

Jogar na tabela:

php artisan db:seed --class=UsersSeeder

Testar

php artisan serve

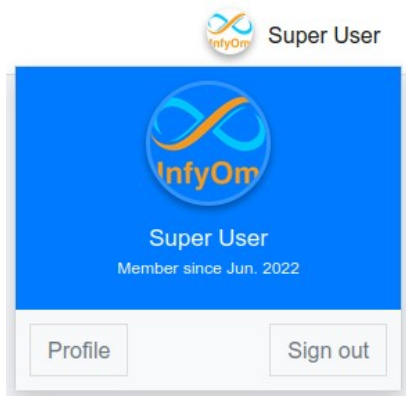
http://127.0.0.1:8000/login

Acessar com

super@mail.org e 123456

Veja que já temos algo substancial e podemos explorar bem, pois estamos logado com o manda chuva, que é o user Super. Mas somente roles e users estão disponíveis, falta adicionar posts e comments, que serão adicionados no próximo capítulo.

A tela de log out



Também podemos registrar um novo usuário

A screenshot of a web form titled 'Laravel' for registering a new membership. The form has a title 'Register a new membership'. It contains four input fields: 'Full name' with a person icon, 'Email' with an envelope icon, 'Password' with a lock icon, and 'Retype password' with a lock icon. Below the fields is a checkbox labeled 'I agree to the terms' and a blue 'Register' button. At the bottom, there is a link that says 'I already have a membership'.

Voltemos ao GH Desktop para sincronizar com a mensagem Capítulo 4.

No próximo capítulo estaremos adicionando os CRUDs posts e comments.

Vamos ao Capítulo 5

5 – Criação dos CRUDs Posts e Comments

- Criar mais dois crud's

Vamos agora criar a estrutura de mais dois CRUDs para deixar nosso aplicativo mais interessante. Para isso usarei o fork de um ótimo gerador de CRUDs que traduzi e adaptei para nosso caso:

<https://github.com/ribafs/crud-generator-laravel-br>

Voltar ao terminal do VSCode

Teclar Ctrl+C para interromper

Instalar

```
composer require ribafs/crud-generator-laravel-br
```

Publicar

```
php artisan vendor:publish --provider="Ribafs\CrudGenerator\CrudGeneratorServiceProvider"
```

Criar os CRUDs

posts (somente o campo name) e comments (campos post_id e comment), relacionados pelo post_id em comments

```
php artisan crud:generate Posts --fields='name#string;' --view-path= --controller-namespace=App\Http\Controllers --route-group= --form-helper=html
```

```
php artisan crud:generate Comments --fields='post_id#integer; comment#text;' --view-path= --controller-namespace=App\Http\Controllers --route-group= --form-helper=html
```

Editar as migrations criadas e ajustar os tamanhos de campos e o relacionamento

posts

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->string('name', 45)->nullable();
    $table->timestamps();
});
```

commants

```
Schema::create('comments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('post_id')->constrained('posts')->unique()->notNullable();
    $table->text('comment')->nullable();
    $table->timestamps();
});
```

Vamos editar o PostController e o CommentController e adicionar o método `__construct()` aos mesmos com o controle de acesso do Spatie permission

CommentController

```
function __construct()
{
    $this->middleware('permission:comment-list|comment-create|comment-edit|comment-delete',
['only' => ['index','store']]);
    $this->middleware('permission:comment-create', ['only' => ['create','store']]);
    $this->middleware('permission:comment-edit', ['only' => ['edit','update']]);
    $this->middleware('permission:comment-delete', ['only' => ['destroy']]);
}
```

PostController

```
function __construct()
{
    $this->middleware('permission:post-list|post-create|post-edit|post-delete', ['only' =>
['index','store']]);
    $this->middleware('permission:post-create', ['only' => ['create','store']]);
    $this->middleware('permission:post-edit', ['only' => ['edit','update']]);
    $this->middleware('permission:post-delete', ['only' => ['destroy']]);
}
```

Criar os seeders para os novos CRUDs criados no terminal do VSCode:

`php artisan make:seeder PostsSeeder`

- databases/seeders/PostsSeeder.php

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class PostsSeeder extends Seeder
{
    public function run()
    {
        $faker = \Faker\Factory::create('pt_BR');
        for ($i=0; $i < 100; $i++) {
            DB::table('posts')->insert([
                'name' => $faker->name,
            ]);
        }
    }
}
```

```

    }
}
}

```

Criar as tabelas posts e comments

`php artisan migrate`

Popular a tabela posts com massa de testes

`php artisan db:seed --class=PostsSeeder`

Criar o CommentsSeeder

`php artisan make:seeder CommentsSeeder`

- databases/seeder/CommentsSeeder.php

```

<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class CommentsSeeder extends Seeder
{
    public function run()
    {
        $faker = \Faker\Factory::create('pt_BR');

        for ($i=0; $i < 100; $i++) {
            DB::table('comments')->insert([
                'comment' => $faker->sentence($nbWords = 6, $variableNbWords = true),
                'post_id' => $faker->numberBetween($min = 1, $max = 100)
            ]);
        }
    }
}

```

Jogar na tabela:

`php artisan db:seed --class=CommentsSeeder`

Melhor é testar no próximo capítulo, pois falta algo que irá gerar erros.

Voltar ao GH Desktop e sincronizar com a mensagem Capítulo 5

Vejamos o Capítulo 6

6 – Ajustes na paginação

Agora vamos adicionar o suporte ao Bootstrap nas views

app/Providers/AppServiceProvider.php

```
...
use Illuminate\Pagination\Paginator;
...
public function boot()
{
    Paginator::useBootstrap();
}
...
```

- Instalar para customização da paginação

php artisan vendor:publish --tag=laravel-pagination

Será Criada a pasta

resources/views/vendor/pagination

Contendo os arquivos:

```
bootstrap-4.blade.php
bootstrap-5.blade.php
default.blade.php
semantic-ui.blade.php
simple-bootstrap-4.blade.php
simple-bootstrap-5.blade.php
simple-default.blade.php
simple-tailwind.blade.php
tailwind.blade.php
```

Copiei a

resources/views/vendor/pagination/bootstrap-5.blade.php

Para

resources/views/vendor/pagination/bootstrap-5custom.blade.php

E customizei para isso:

```
@if ($paginator->hasPages())
    <nav class="d-flex justify-items-center justify-content-between">
        <div class="d-flex justify-content-between flex-fill d-sm-none">
            <ul class="pagination">
                {{-- Previous Page Link --}}
```

```

    @if ($paginator->onFirstPage())
        <li class="page-item disabled" aria-disabled="true">
            <span class="page-link">@lang('pagination.previous')</span>
        </li>
    @else
        <li class="page-item">
            <a class="page-link" href="{{ $paginator->previousPageUrl() }}"
rel="prev">@lang('pagination.previous')</a>
        </li>
    @endif

```

```

    {{-- Next Page Link --}}
    @if ($paginator->hasMorePages())
        <li class="page-item">
            <a class="page-link" href="{{ $paginator->nextPageUrl() }}"
rel="next">@lang('pagination.next')</a>
        </li>
    @else
        <li class="page-item disabled" aria-disabled="true">
            <span class="page-link">@lang('pagination.next')</span>
        </li>
    @endif
</ul>
</div>

```

```

<div class="d-none flex-sm-fill d-sm-flex align-items-sm-center justify-content-sm-between">
    <div>
        <p class="small text-muted">
            {!! __('Showing') !!}
            <span class="font-medium">{{ $paginator->firstItem() }}</span>
            {!! __('to') !!}
            <span class="font-medium">{{ $paginator->lastItem() }}</span>
            {!! __('of') !!}
            <span class="font-medium">{{ $paginator->total() }}</span>
            {!! __('results') !!}
        </p>
    </div>
</div>

```

```

<div>
    <ul class="pagination">
        {{-- Previous Page Link --}}
        @if ($paginator->onFirstPage())
            <li class="page-item disabled" aria-disabled="true" aria-
label="@lang('pagination.previous')">
                <span class="page-link" aria-hidden="true">Primeira</span>
            </li>
        @else
            <li class="page-item">

```



```

        <a class="page-link" href="{{ $paginator->previousPageUrl() }}" rel="prev"
aria-label="@lang('pagination.previous')">Anterior</a>
    </li>
@endif

```

```

    {{-- Pagination Elements --}}
    @foreach ($elements as $element)
        {{-- "Three Dots" Separator --}}
        @if (is_string($element))
            <li class="page-item disabled" aria-disabled="true"><span class="page-
link">{{ $element }}</span></li>
        @endif
    @endforeach

```

```

    {{-- Array Of Links --}}
    @if (is_array($element))
        @foreach ($element as $page => $url)
            @if ($page == $paginator->currentPage())
                <li class="page-item active" aria-current="page"><span class="page-
link">{{ $page }}</span></li>
            @else
                <li class="page-item"><a class="page-link"
href="{{ $url }}">{{ $page }}</a></li>
            @endif
        @endforeach
    @endif
@endif

```

```

    {{-- Next Page Link --}}
    @if ($paginator->hasMorePages())
        <li class="page-item">
            <a class="page-link" href="{{ $paginator->nextPageUrl() }}" rel="next" aria-
label="@lang('pagination.next')">Próxima</a>
        </li>
    @else
        <li class="page-item disabled" aria-disabled="true" aria-
label="@lang('pagination.next')">
            <span class="page-link" aria-hidden="true">Última</span>
        </li>
    @endif
</ul>
</div>
</div>
</nav>
@endif

```

Customizar a paginação

- Customizar o controller PostsController para que mostre somente 7 registros por página.

Editar index() e trocar 25 por 7:

```
public function index(Request $request)
{
    $keyword = $request->get('search');
    $perPage = 7;
```

Observação


Veja que estou fazendo certas customizações apenas no controller Post e nas views posts. Deixando que você faça o mesmo em comments.






















Veja detalhes sobre a paginação

Veja que quando chegamos na primeira página o botão Primeira fica desabilitado

Posts

+ Adicionar novo

Busca... 

ID	Name	Ações
100	Diego Victor Dominato Sobrinho	 Ver  Editar  Excluir
2	Sra. Allison Faria Valentin Jr.	 Ver  Editar  Excluir
3	Vitória Lara Beltrão	 Ver  Editar  Excluir
4	Isabel Marília Ramos	 Ver  Editar  Excluir
5	William Batista	 Ver  Editar  Excluir
6	Olga Casanova Neto	 Ver  Editar  Excluir
7	Sra. Naomi Larissa Verdara Filho	 Ver  Editar  Excluir

Mostrando 1 até 7 de 100 resultados

Primeira

1

2

3

4

5

6

7

8

9

10


...

14


15

Próxima

Quando chegamos na última o botão Última fica desabilitado

Posts		
+ Adicionar novo		Busca... 
ID	Name	Ações
36	Sra. Silvana Ramires Valência Neto	Ver Editar Excluir
50	Dr. Paloma Michelle Gusmão Sobrinho	Ver Editar Excluir
Mostrando 99 até 100 de 100 resultados		Anterior 1 2 ... 6 7 8 9 10 11 12 13 14 15 Última

Veja detalhes sobre a busca. Digita-se uma string e teclamos enter ou clicamos no botão então ele mostra somente os registros que contém a string

Posts		
+ Adicionar novo		br 
ID	Name	Ações
100	Diego Victor Dominato Sobrinho	Ver Editar Excluir
17	Breno Ramos Quintana Neto	Ver Editar Excluir
20	Srta. Karina Quintana Salas Sobrinho	Ver Editar Excluir
22	Mel Velasques Godói Sobrinho	Ver Editar Excluir
25	Sr. Miguel Vale Sobrinho	Ver Editar Excluir
50	Dr. Paloma Michelle Gusmão Sobrinho	Ver Editar Excluir
64	Sra. Daniele Barreto Balestero Sobrinho	Ver Editar Excluir
Mostrando 1 até 7 de 15 resultados		Primeira 1 2 3 Próxima

Criação de commands

Vamos criar alguns commands para nos ajudar a administrar o aplicativo.

Eles ficam na pasta `app/Console/Commands`

```
BackupDbCommand.php
BackupFilesCommand.php
DbCreateCommand.php
DbSetupCommand.php
LogsClearCommand.php
PermsAddCommand.php
PermsDelCommand.php
RolesAddCommand.php
```

Para criar um command no laravel fazemos assim:

```
php artisan make:command NomeCommand
```

Ele então criará o arquivo:

```
app/Console/Commands/NomeCommand.php
```

Aproveitar e instalar (somente no linux)

```
composer require nunomaduro/laravel-console-menu
```

Que é usado por alguns dos commands. **Este componente não deve ser instalado no windows, pois tem uma dependência que não é compatível com o mesmo e impedirá a execução do composer install.**

Vamos agora configurar a Localization para pt-BR

```
composer require lucascudo/laravel-pt-br-localization --dev
php artisan vendor:publish --tag=laravel-pt-br-localization
```

Configurações para utilizar 'pt-BR' como linguagem padrão

```
// Altere Linha 85 do arquivo config/app.php para:
'locale' => 'pt-BR',
```

Caso deseje, configure o Framework para utilizar 'America/Fortaleza' como o timezone padrão

```
// Altere Linha 70 do arquivo config/app.php para:
'timezone' => 'America/Fortaleza',
```

Instalar e configurar o pacote DebugBar, para melhorar o debug

Lembrando que devemos instalar a debugBar somente em ambiente de desenvolvimento para ajudar a debugar erros

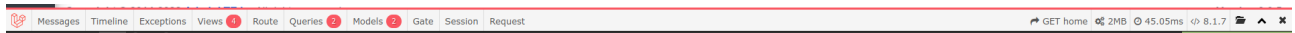
```
composer require barryvdh/laravel-debugbar --dev  
php artisan vendor:publish --provider="Barryvdh\Debugbar\ServiceProvider"
```

config/app.php

```
// ServiceProviders  
Barryvdh\Debugbar\ServiceProvider::class,  
  
// Facades  
'Debugbar' => Barryvdh\Debugbar\Facade::class,
```

Testar

```
php artisan serve  
http://127.0.0.1:8000/login
```



Agora sim, já podemos experimentar com os 4 usuários criados, mas ainda faltam alguns recursos.

Voltar ao GH Desktop e sincronizar com Capítulo 6

Vejamos o Capítulo 7

7 – Tratamento de Erros e Sugestões

Tratamento de erros

Para interceptar a mensagem do erro 403.

Atualmente se algum user tentar acessar algum recurso ao qual não tem permissão, ele receberá a mensagem:

403

User does not have the right permissions. Ex.: Acessar com admin e tentar acessar posts:

http://127.0.0.1:8000/posts

Vamos interceptar esta mensagem e melhorar.

Criar a pasta:

resources/views/errors

Dentro dela criar o arquivo

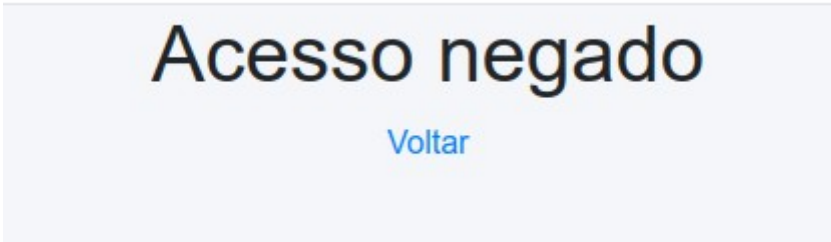
403.blade.php

Contendo:

```
@extends('layouts.app')

@section('content')
<div class="text-center">
    <h1>{{ 'Acesso negado' }}</h1>
    <a href="#" onClick="history.back()">Voltar</a>
</div>
@endsection
```

Agora quando um user tentar acessar um recurso que ele não ter permissão de acesso, receberá a mensagem



Acesso negado

[Voltar](#)

Ajustar as rotas

routes/web.php

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\RoleController;
use App\Http\Controllers\UserController;

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

Auth::routes();

Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

Route::group(['middleware' => ['auth']], function() {
    Route::resource('roles', RoleController::class);
    Route::resource('users', UserController::class);
    Route::resource('comments', 'App\Http\Controllers\CommentsController');
    Route::resource('posts', 'App\Http\Controllers\PostsController');
});
```

Ajuste fino no controle de acesso

Como está já existe um ótimo controle de acesso, pois cada user somente acessa aquilo que demos permissão a ele nos controllers, no método `__construct()`.

Mas podemos controlar cada uma das regiões de cada view, dando acesso somente para quem o tem. E isso é importante para CRUDs cujos users não tenham acesso a todas as views, como é o caso do usuário User, que tem acesso somente para a view index e show.

Neste caso ele acessa a view index de posts e verá também os botões Adicionar Novo, Editar e Excluir. Quando ele clicar em qualquer um destes botões receberá a mensagem de acesso negado, mas idealmente devemos ocultar estes botões de posts/index sempre que for acessado pelo user User.

Para isso usando a diretiva `@can()` nas views/ blades e existem diversos outros recursos.

Veja então como ficará a view posts/index:

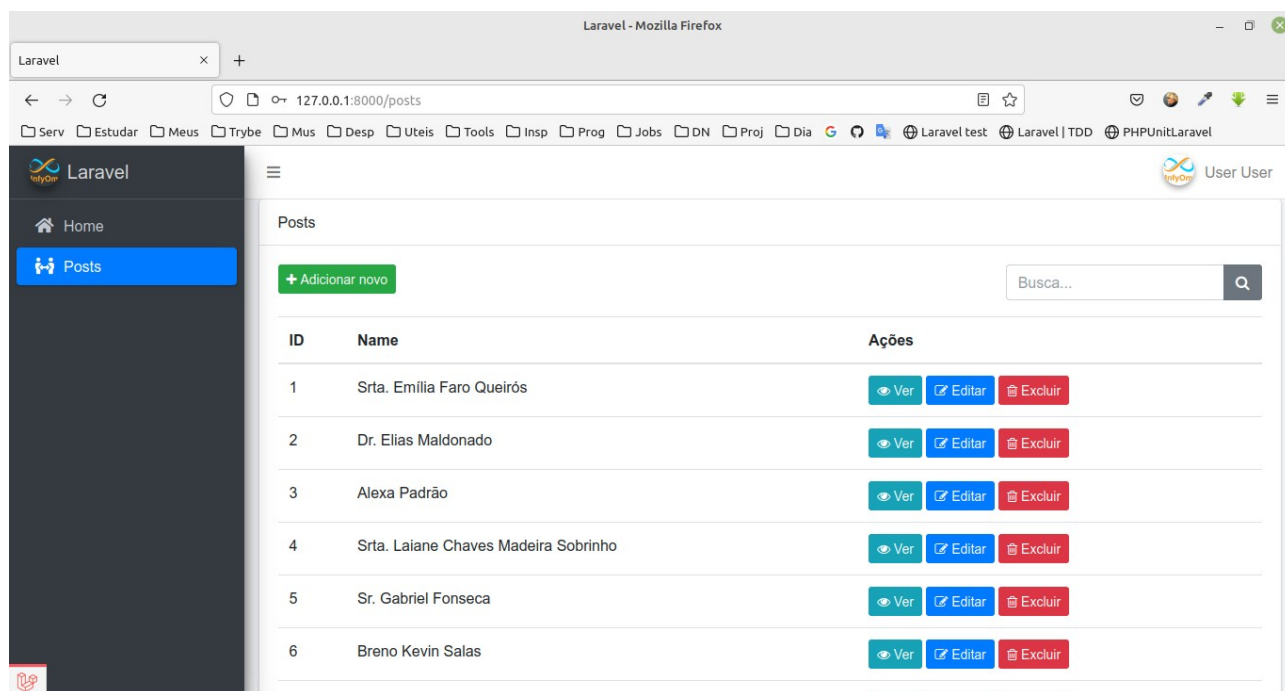
```
@extends('layouts.app')

@section('content')
    <div class="container">
        <div class="row">
            @include('layouts.sidebar')

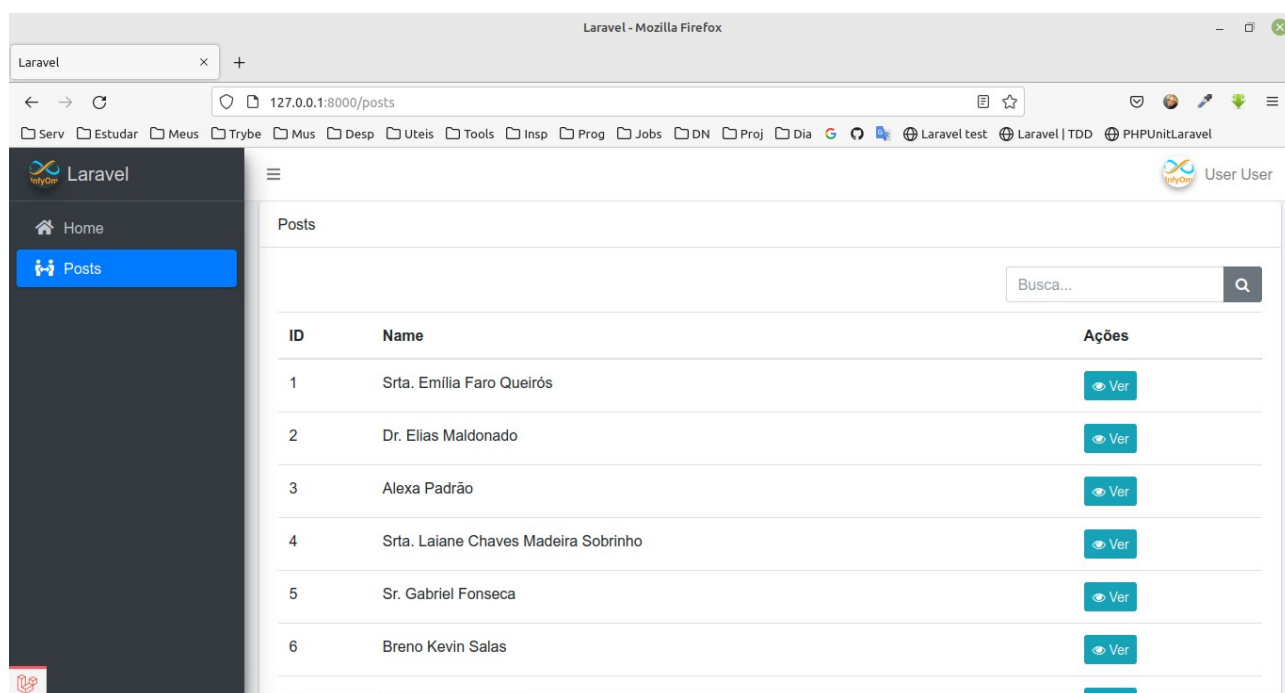
            <div class="col-md-12">
                <div class="card">
                    <div class="card-header">Posts</div>
                    <div class="card-body">
                        @can('post-create')
                            <a href="{{ url('/posts/create') }}" class="btn btn-success btn-sm" title="Adicionar
Post">
                                <i class="fa fa-plus" aria-hidden="true"></i> Adicionar novo
                            </a>
                        @endcan
                    <form method="GET" action="{{ url('/posts') }}" accept-charset="UTF-8"
class="form-inline my-2 my-lg-0 float-right" role="search">
                        <div class="input-group">
                            <input type="text" class="form-control" name="search"
placeholder="Busca..." value="{{ request('search') }}">
                            <span class="input-group-append">
                                <button class="btn btn-secondary" type="submit">
                                    <i class="fa fa-search"></i>
                                </button>
                            </span>
                        </div>
                    </form>

                    <br/>
                    <br/>
                    <div class="table-responsive">
                        <table class="table">
                            <thead>
                                <tr>
                                    <th>ID</th> <th>Name</th> <th>Ações</th>
                                </tr>
                            </thead>
                            <tbody>
                                @foreach($posts as $item)
                                    <tr>
                                        <td>{{ $item->id }}</td>
                                        <td>{{ $item->name }}</td>
                                        <td>
```


Veja como era a tela indes/posts antes do ajuste fino



Agora após o ajuste



Ferramenta para gerar seeders de tabelas existentes

composer require orangehill/iseed

php artisan iseed nomeTabela

<https://github.com/orangehill/iseed>

Sugestões para a adição de um novo CRUD

- Usar o crud-generator-laravel-br
- Ajustar o tamanho dos campos na migration

- Alterar as views. Onde tem:

```
@include('admin.sidebar')
```

Mudar para

```
@include('layouts.sidebar')
```

- Editar o controller e adicionar o método __construct()

```
function __construct()
{
    $this->middleware('permission:post-list|post-create|post-edit|post-delete', ['only' =>
['index','store']]);
    $this->middleware('permission:post-create', ['only' => ['create','store']]);
    $this->middleware('permission:post-edit', ['only' => ['edit','update']]);
    $this->middleware('permission:post-delete', ['only' => ['destroy']]);
}
```

Supondo CRUD vendas, mudar de post- para venda-

- Adicionar as 4 permissões no PermissionTableSeeder.php. Isso deve ser feito antes de executar os seeders

```
'venda-list',
'venda-create',
'venda-edit',
'venda-delete'
```

Para isso podemos usar o command perms:add

php artisan perms:add

Supondo que queremos adicionar o CRUD Vendas, então entramos

venda

E ele cria as 4 citadas acima.

Também podemos usar o command `perms:del`, caso desejemos excluir alguma permissão.

- Editar o `resources/views/layouts/sidebar.blade.php` para adicionar o link do CRUD Vendas.
- Editar o `resources/views/layouts/app.blade.php` para adicionar o link do CRUD Vendas.

Criação de Script

Criei um script que facilitam muito a vida do programador, executam os comandos acima, fazem backup, limpam os logs, acessar o aplicativo via navegador padrão, etc

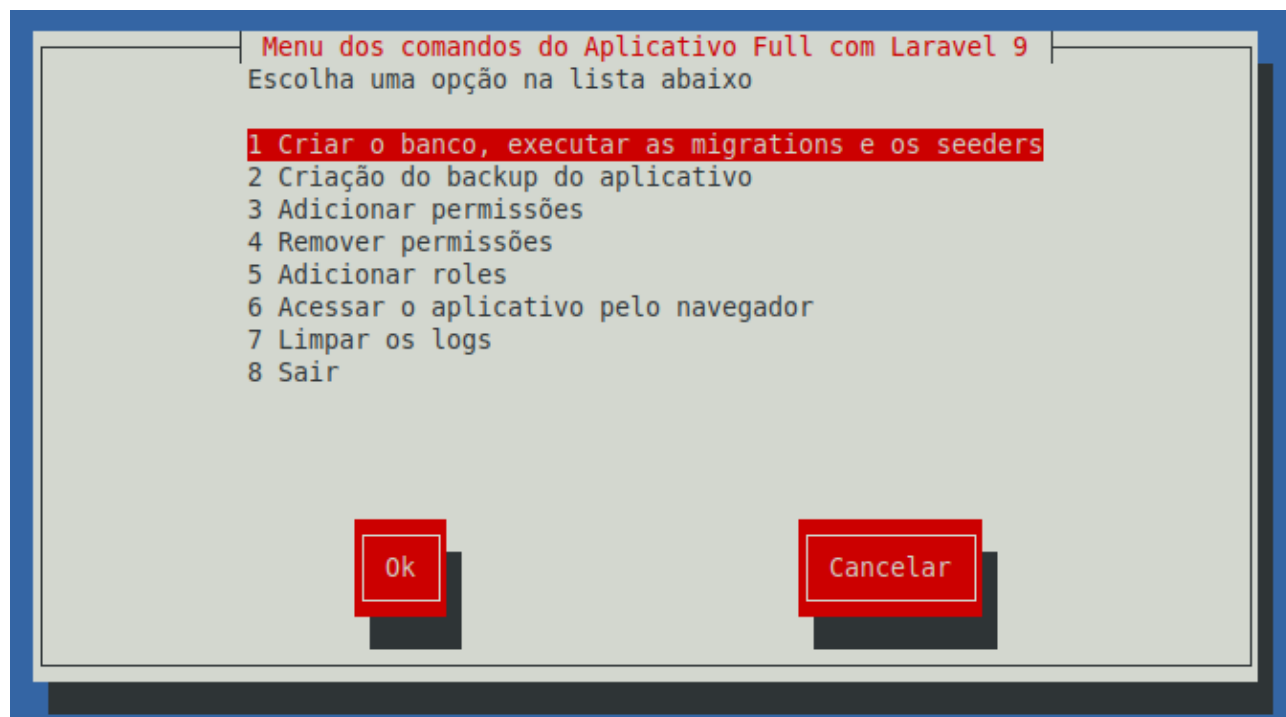
m para linux

Não consegui criar algo decente para windows

Estão no raiz do aplicativo e podem ser executados via terminal com

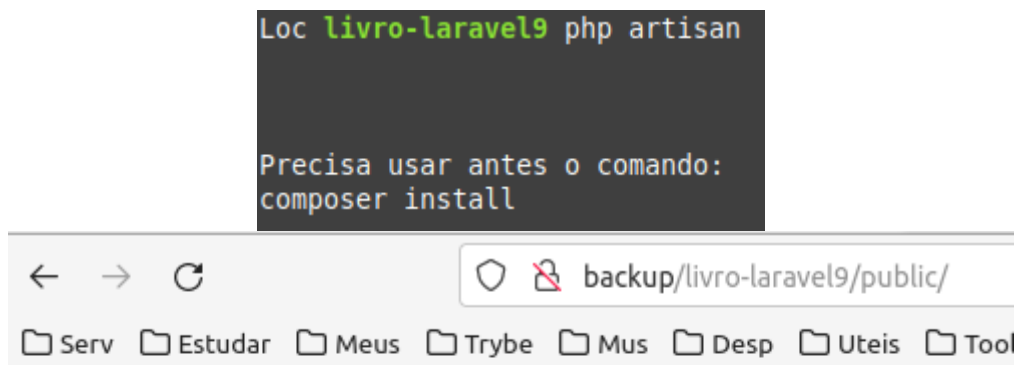
```
./m
```

A versão para linux tem mais recursos:



Também customizei

artisan do raiz,
o *public/index.php*



**Precisa usar antes o comando:
composer install**

Quando clonamos o aplicativo do github ele não vem com a pasta vendor. Então se o user tentar executar o comando *artisan* ele dará uma mensagem em português dizendo que precisa executar antes "*composer install*". Assim fica melhor que receber aqueles vários erros e nem saber porque. Da mesma forma se chamar o *public* via navegador.


Acesse com os user e senhas


email senha permissões


super@mail.org 123456 tudo
admin@mail.org 123456 users, roles e permissions
manager@mail.org 123456 posts e comments
user@mail.org 123456 index em posts


Veja o menu que cada user acessa:


Super


 Laravel





 Home

 Posts

 Comments


 Roles


 Permissions


 Users


Escolha uma opção no menu à esquerda!


Admin


 Laravel



 Home


 Roles


 Permissions


 Users


Escolha uma opção no menu à esquerda!


Manager

 Laravel




 Home


 Posts


 Comments


Escolha uma opção no menu à esquerda!

User

 Laravel



 Home

 Posts

Escolha uma opção no menu à esquerda!

Um pouco sobre padrões

Estudando a documentação do pacote de permissões da Spatie, vemos que usam nomes de permissões e roles com padrões diferentes dos usados aqui. Particularmente gosto do padrão usado no tutorial em que me baseei para criar este aplicativo.

Permissões - client-edit

Roles - Admin

As permissões com tudo em minúsculas, iniciando com o nome da tabela no singular, seguido de hífen e finalizando com o nome da view: role-create.

As roles com nomes representativos, no singular e com inicial maiúscula.

Não existe o certo, do jeito que criamos devemos depois chamar. Então criar um padrão irá nos ajudar a lembrar dele e a evitar assim erros.

Sugestões

Mudar em UsersSeeder de

```
$role = Role::find(1);
$permissions = Permission::pluck('id','id')->all();
$role->syncPermissions($permissions);
$super->assignRole([$role->id]);
```

```
$admins = [1, 2, 3, 4, 5, 6, 7, 8]; // IDs das permissões da role Admin sobre users e roles
$radmin = Role::find(2);
$radmin->syncPermissions($admins);
$adminu->assignRole([$radmin->id]);
```

Para

```
$role = Role::findByName('Super');
$permissions = Permission::pluck('name','name')->all();
$role->syncPermissions($permissions);
$super->assignRole([$role->name]);
```

```
$admins = ['role-list','role-create','role-edit','role-delete','user-list','user-create','user-edit','user-delete'];
$radmin = Role::findByName('Admin');
$radmin->syncPermissions($admins);
$adminu->assignRole([$radmin->name]);
```

E para os demais users também.

Troquei id por name das permissions.

Suporte aos 3 SGBDs free e open

MySQL/MariaDb

No caso estou usando:

user - root

senha - root

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

Adicionar suporte ao PostgreSQL e ao SQLite

Como o Laravel usa o PDO e este tem suporte a vários SGBDs, então mostrarei como uso o PostgreSQL e o SQLite. Apenas precisamos ajustar o script de conexão, o .env.

PostgreSQL

Caso precise de ajuda para configurar o PostgreSQL veja o respectivo item.

Vamos criar o banco livro

Meu super user é postgres e sua senha é postgres

```
sudo -u postgres -p postgres createdb livro
```

nano .env

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=livro
DB_USERNAME=postgres
DB_PASSWORD=postgres
```


Executar pelo terminal:

O script de menu m não está ajustado para postgres, apenas para mysql, então faremos manualmente:

```
composer install
php artisan migrate
php artisan db:seed --class=PermissionsSeeder
php artisan db:seed --class=RolesSeeder
php artisan db:seed --class=UsersSeeder
php artisan db:seed --class=PostsSeeder
php artisan db:seed --class=CommentsSeeder
```

```
php artisan serve
http://127.0.0.1:8000/login
```

Pode testar com os 4 users

Veja o Capítulo 8 para suporte na busca ao Postgres e ao SQLite

SQLite

```
touch database/livro.sqlite
```

```
nano .env
```

```
DB_CONNECTION=sqlite
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=/home/ribafs/GitHub/livro-laravel9/database/livro.sqlite
DB_USERNAME=postgres
DB_PASSWORD=postgres
```

```
php artisan migrate
php artisan db:seed --class=PermissionsSeeder
php artisan db:seed --class=RolesSeeder
php artisan db:seed --class=UsersSeeder
php artisan db:seed --class=PostsSeeder
php artisan db:seed --class=CommentsSeeder
```

```
php artisan serve
http://127.0.0.1:8000/login
```

Podemos testar com os 4 usuários e veja que não existe diferença entre os 3 SGBDs. Mas se fizer uma busca no PostgreSQL verá que o case insensitive não está ativado. Iremos corrigir isso no próximo capítulo.

Voltar ao GH Desktop e sincronizar com a mensagem Capítulo 7

Vejamos o Capítulo 8

8 – Ajustes Finais no Aplicativo

Ajustes finais

Ajustes nos controllers Post e Comment para que a busca ofereça suporte ao PostgreSQL e ao SQLite

Mudar o método index() de PostController para:

```
public function index(Request $request)
{
    $keyword = $request->get('search');
    $perPage = 7;
    $env = env('DB_CONNECTION'); // https://stackoverflow.com/questions/44861859/laravel-env-in-database-config

    if (!empty($keyword)) {
        if ($env == 'mysql' || $env == 'sqlite') {
            $posts = Post::where('name', 'LIKE', "%$keyword%")->latest()->paginate($perPage);
        } else if ($env == 'pgsql') {
            $posts = Post::where('name', 'ILIKE', "%$keyword%")->latest()->paginate($perPage);
        }
    } else {
        $posts = Post::latest()->paginate($perPage);
    }

    return view('posts.index', compact('posts'));
}
```

Para Comment

```
public function index(Request $request)
{
    $keyword = $request->get('search');
    $perPage = 25;
    $env = env('DB_CONNECTION'); // https://stackoverflow.com/questions/44861859/laravel-env-in-database-config

    if (!empty($keyword)) {
        if ($env == 'mysql' || $env == 'sqlite') {
            $comments = Comment::where('post_id', 'LIKE', "%$keyword%")->orWhere('comment', 'LIKE', "%$keyword%")->latest()->paginate($perPage);
        } else if ($env == 'pgsql') {
            $comments = Comment::where('post_id', 'ILIKE', "%$keyword%")->orWhere('comment', 'ILIKE', "%$keyword%")->latest()->paginate($perPage);
        }
    } else {

```

```
$comments = Comment::latest()->paginate($perPage);
}
```

```
return view('comments.index', compact('comments'));
}
```

Assim poderemos efetuar buscas sem nos preocupar com o case.

Adicionar o CDN abaixo ao final do app.blade.php logo abaixo de app.js

```
<script src="https://use.fontawesome.com/6c213b8c33.js"></script>
```

Para que mostre os ícones nas views corretamente.

Seguindo em frente

Adicionar mais uma role, a role User ao User Admin e capturar a tela do index de users

Criar o CRUD para permissions

```
php artisan crud:generate Permissions --fields='name#string; guard_name#string' --view-path= --
controller-namespace=App\Http\Controllers --route-group= --form-helper=html
```

Remover a migration xxx_permissions_xxx.php criada pelo gerador de cruds, pois a tabela já existe.

Adicionar o método __construct() ao controller Permissions:

```
function __construct()
{
    $this->middleware('permission:permission-list|permission-create|permission-edit|permission-
delete', ['only' => ['index','store']]);
    $this->middleware('permission:permission-create', ['only' => ['create','store']]);
    $this->middleware('permission:permission-edit', ['only' => ['edit','update']]);
    $this->middleware('permission:permission-delete', ['only' => ['destroy']]);
}
```

Habilitar o suporte para o Postgres e o SQLite no método index()

```
public function index(Request $request)
{
    $keyword = $request->get('search');
    $perPage = 7;
    $env = env('DB_CONNECTION'); // https://stackoverflow.com/questions/44861859/laravel-
env-in-database-config
```

```

        if (!empty($keyword)) {
            if($env == 'mysql' || $env == 'sqlite'){
                $permissions = Permission::where('name', 'LIKE', "%$keyword%")->latest()-
>paginate($perPage);
            }else if($env == 'pgsql'){
                $permissions = Permission::where('name', 'ILIKE', "%$keyword%")->latest()-
>paginate($perPage);
            }
        } else {
            $permissions = Permission::latest()->paginate($perPage);
        }

        return view('permissions.index', compact('permissions'));
    }

```

Adicionar permission ao PermissionsSeeder.php

Adicionei logo abaixo de user-list:

```

...
    'permission-list',
    'permission-create',
    'permission-edit',
    'permission-delete',
...

```

Após fazer isso precisaremos remover os registros da tabela permissions e popular novamente executando o PermissionsSeeder.php. A forma mais simples de fazer isso é executando o script ./m e sua primeira opção.

Ajustar o alert do BootStrap

Pois nosso pacote do gerador ainda não foi atualizado. Quando você estiver criando este aplicativo não precisará fazer este ajuste, pois já terei atualizado o gerador.

Logo abaixo de Add new permission no resources/views/permissions/index.blade.php

```

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

```

Após efetuar uma alteração de registro aparece a mensagem abaixo do botão Adicionar novo

Posts

[+ Adicionar novo](#)

Post updated!

Busca...

ID	Name	Ações
100	Diego Victor Dominato Sobrinho2	Ver Editar Excluir

E assim aparecerá ao adicionar um novo e ao excluir.

Adicionar Permissions ao resources/layouts/menu.blade.php

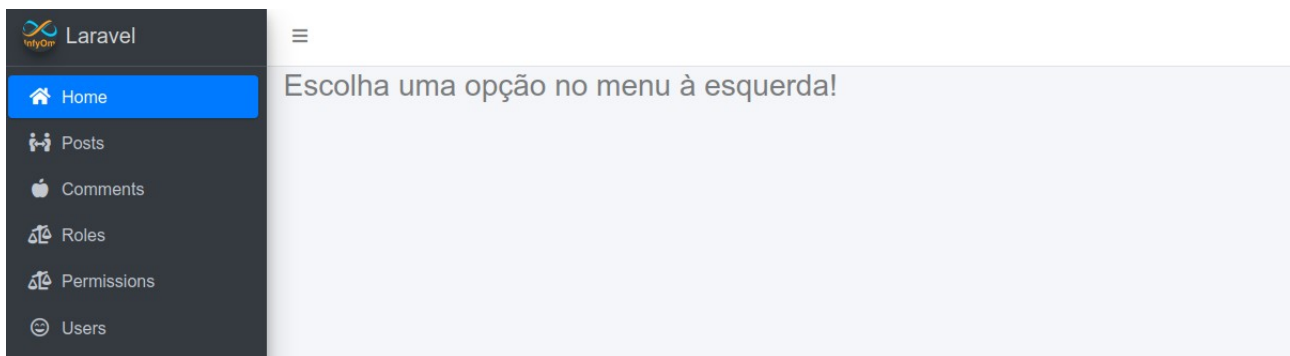
Deve ficar assim:

```
<li class="nav-item">
  <a href="{{ route('home') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
    <i class="nav-icon fas fa-home"></i>
    <p>Home</p>
  </a>
  @role('User')
  <a href="{{ url('posts') }}" class="nav-link {{ Request::is('posts') ? 'active' : '' }}">
    <i class="nav-icon fas fa-people-arrows"></i>
    <p>Posts</p>
  </a>
  @endrole
  @hasanyrole('Super|Manager')
  <a href="{{ url('posts') }}" class="nav-link {{ Request::is('posts') ? 'active' : '' }}">
    <i class="nav-icon fas fa-people-arrows"></i>
    <p>Posts</p>
  </a>
  <a href="{{ url('comments') }}" class="nav-link {{ Request::is('comments') ? 'active' : '' }}">
    <i class="nav-icon fas fa-apple-alt"></i>
    <p>Comments</p>
  </a>
  @endhasanyrole
  @hasanyrole('Super|Admin')
  <a href="{{ url('roles') }}" class="nav-link {{ Request::is('roles') ? 'active' : '' }}">
    <i class="nav-icon fas fa-balance-scale-left"></i>
    <p>Roles</p>
  </a>
  <a href="{{ url('permissions') }}" class="nav-link {{ Request::is('permissions') ? 'active' : '' }}">
    <i class="nav-icon fas fa-balance-scale-left"></i>
    <p>Permissions</p>
  </a>
</li>
```

```

</a>
<a href="{{ url('users') }}" class="nav-link {{ Request::is('users') ? 'active' : '' }}">
  <i class="nav-icon far fa-grin-beam"></i>
  <p>Users</p>
</a>
@endhasanyrole
</li>

```



Quando tentar acessar mesmo que com o Super receberá um acesso negado. Isto acontece porque permissions ainda não está na lista de permissões do Super.

Primeiro precisamos adicionar as permissões abaixo ao permissions:

```

permission-list
permission-create
permission-edit
permission-delete

```

A forma mais prática de fazer isso agora é usando o command
perms:add

```
php artisan perms:add
```

Entrar

permission

Acessar com o user Super ou Admin

Editar Roles Super
 Adicionar as permissões

```

permission-list
permission-create
permission-edit
permission-delete

```

Agora já podemos acessar Permissions com o user Super ou com Admin

O CRUD permissions pode ser melhorado para que use os recursos existentes no CRUD users, especificamente no create e no edit, onde podemos seleccionar as 4 views.

Tanto podemos usar o form.blade.php, quanto podemos substituí-lo por código HTML, com um select multiple.

Nosso aplicativo tem suporte também a:

- Usuário pertencendo a duas ou mais roles

Veja abaixo o user Admin que pertence as roles Admin e User

Users Management

[Create New User](#)

User updated successfully

No	Name	Email	Roles	Action
1	User User	user@mail.org	User	Show Edit Delete
2	Manager User	manager@mail.org	Manager	Show Edit Delete
3	Admin User	admin@mail.org	Admin User	Show Edit Delete
4	Super User	super@mail.org	Super	Show Edit Delete

Acesse o aplicativo como Admin ou Super

Acesse Users

Edit o Admin

Veja que ele está ligado a role Admin

Vamos adicionar a role User, para que ele também tenha acesso à view index de posts

Para isso prendemos o Ctrl e clicamos em User em Role e Submit

Fazemos logout e login com Admin para testar

Ocultando o link Register

Vamos ocultar o link para registro de novo usuário caso isso seja importante

Para isso basta comentar o link de registro em

resources/views/welcome.blade.php

```
...  
<!--  
    @if (Route::has('register'))  
        <a href="{{ route('register') }}">Register</a>  
    @endif  
-->  
...
```

Então ficará assim



Uma boa opção para gerar dados de teste é o faker-restaurant

Usar nomes de frutas, alimentos, etc

composer require jzonta/faker-restaurant

Sincronizemos via GH Desktop como Capítulo 8

Vejamos o Capítulo 9

9 – Produtividade do Programador

Neste capítulo quero abordar alguns tópicos para estimular você a assumir uma atitude produtiva e se tornar alguém bem focado em produtividade.

Ter a eficiência, a simplificação e a performance como atitude é importante para o programador e também para o resultado do seu trabalho.

Para isso é bom cuidar de vários aspectos para conseguir criar ao redor de si um ambiente que o estimule a ser eficiente e produtivo.

- Tornar o seu hardware o mais produtivo possível, otimizando o que for possível/viável
- Selecionar e usar o melhor sistema operacional possível para você e otimizar este sistema (isso é relativo, mas existe o mais adequado para você)

Algo que ajuda aqui é criar teclas de atalho customizadas para os aplicativos mais usados: navegador, processador de texto, VSCode, etc

Também usar as teclas de atalho default do sistema operacional e dos aplicativos: Ctrl+S, Ctrl+O, Ctrl+N, Ctrl+W, etc.

- Usar a melhor linguagem (ou linguagens), o melhor SGBD, frameworks, editores/IDEs, etc
- Usar o melhor ambiente de desenvolvimento, o mais produtivo
- Organização é parceira importante, pois evita perda de tempo procurando algo
- Procurar adotar boas práticas de programação e bons padrões de projeto (bons frameworks já fazem isso)
- Adoção de padrões e convenções também ajuda a evitar erros e a memorizar mais facilmente nomes
- Manter um forte foco e evitando distrações
- Mantenha-se atento para cultivar a atitude voltada para a produtividade

O mais importante é a atitude que assumimos em nossa vida, de ser mais eficiente e produtivo. É importante que procuremos ser assim em tudo que fazemos, para que isso se torne algo que o subconsciente se encarregue e então começaremos a agir assim naturalmente. Será nossa forma de agir.

Cada um de nós tem exatamente 24 horas a cada dia, mas algumas pessoas conseguem realizar uma maior quantidade de atividades que outras nas mesmas 24 horas e, ainda conseguem ser mais eficientes. Bem, não dá para saber o que se passa com todas as pessoas mais produtivas mas dá para procurarmos ser mais produtivo em nossas atividades, nos organizando mais, evitando distrações, usando as ferramentas e as práticas mais adequadas. Ficar de olho em qualquer coisa que possa tornar mais produtivo nosso dia.

Algumas das atividades do meu dia que acredito tornam ele mais produtivo:

Primeiro, eu acredito que "Em casa de ferreiro que se preza, o espeto é de inox". Com isso quero dizer que se eu estou aqui defendendo a produtividade é absurdo eu ser hipócrita e não procurar ser produtivo.

- Desktop Linux (Linux Mint 20) em meu dia a dia. Com isso tenho relativamente um sistema mais robusto, estável e produtivo para mim.
- Apache, PHP, MariaDB e cia instalados via pacotes e com um script de uma só vez. Além de ser prático eu não preciso ficar me preocupando com atualizar os mesmos. Sempre que sai uma atualização de qualquer um o Mint me avisa e apenas autorizo a atualização. Isso é muito importante para ter os novos recursos e corrigir vulnerabilidades.
- Uso no dia a dia para codificar, o editor de código mais leve que conheço, o Xed, que acompanha o Mint. Quando preciso de uma ajuda extra para debugar algo eu uso o VSCode.
- Para editar imagens, algumas vezes, uso o Kolourpaint. Algo como um Paint melhorado.
- Para elaborar meus conteúdos geralmente uso o LibreOffice Writer
- Meu navegador padrão é o Firefox, mas não por conta de desempenho, porque me acostumei com ele e me atende
- Capturo telas com agilidade com o capturador de telas do Gnome criar conteúdo
- Configuro tudo para se adaptar melhor para meu uso: teclas de atalho, atalhos no gerenciador de arquivos nemo, configuro para usar com apenas um único clique, divido a tela do Nemo em dois painéis, etc
- Quando uso Windows procuro também ser produtivo. Meu ambiente preferido era o Xampp, mas agora é o Laragon, que já traz praticamente tudo que usamos atualmente.
- Uso o adminer ao invés do phpmyadmin, por dois motivos, por ser mais ágil (é apenas um arquivo PHP) e por suportar diversos SGBD, inclusive o PostgreSQL, que uso também.
- Crio scripts e aliases que agilizam as minhas tarefas e economizam tempo.

Economizando tempo em cada atividade, ao final do dia terei ganhado algum tempo.

Enumerei estes procedimentos e ferramentas não para dar uma receita de como ser produtivo, pois a produtividade também é relativa. O que é produtivo para uns pode não ser para outros. Então a intenção é estimular para que descubra como melhorar a sua própria produtividade.

A seguir eu compartilho diversos scripts que criei para mim. Altere estes, adapte para você, crie os seus.

Scripts e Aliases

<https://github.com/ribafs/laravel/tree/main/1Introducao/Ambiente/Scripts>

<https://github.com/ribafs/laravel/blob/main/1Introducao/Ambiente/aliases.md>

Se usando Windows pode suar os exemplos abaixo para adaptar os scripts e aliases para arquivos batchs do Windows/DOS.

<https://github.com/ribafs/laravel/blob/main/1Introducao/Ambiente/Batchs.md>

Criação de alguns script para agilizar tarefas

Gosto de criar scripts que agilizem minhas tarefas diárias e no `/usr/local/bin`, para que fique no path e eu possa executar de qualquer pasta.

Trago apenas versões para Linux. Até tentei criar correspondentes para windows mas não consegui.

Gosto tanto de criar scripts e os colocar no path, que acabei criando dois scripts somente para agilizar a criação de outros scripts.

ubin - cria scripts

umod - seta a permissão de execução nos scripts

```
sudo nano /usr/local/bin/ubin
```

Contendo:

```
sudo nano /usr/local/bin/$1
```

```
sudo nano /usr/local/bin/umod
```

Contendo:

```
sudo chmod +x /usr/local/bin/$1
```

Executando

Se quero criar um script chamado teste, ao invés de digitar: `sudo nano /usr/local/bin/teste`, digito apenas `ubin teste`

```
ubin teste
```

Para torná-lo executável

```
umod teste
```

O laravel tem muitos comandos para o terminal que começam com: "php artisan" e daí recebem um ou dois parâmetros

```
ubin pa
```

```
pa $1 $2
```

```
umod pa
```

Agora veja:

pa migrate

pa key:generate

pa serve

Ao final do dia terei ganhado alguns toques e no início exercitei minha solução de problemas em busca da produtividade, na criação dos scripts.

Caso sinta necessidade consulte a segunda parte deste livro, que contém informações teóricas sobre vários tópicos deste livro.

Segunda Parte

1 - Novidades da Versão 9

O laravel lança novas versões maiores a cada ano, em fevereiro. Na versão 8 era de 6 em 6 meses. Enquanto lançamentos menores e de patch podem ser lançados a cada semana. As versões secundárias e de patch nunca devem conter alterações importantes, que possam quebrar as maiores.

Laravel 8 - PHP 7.3 - 8.1, lançado em September 8th, 2020, corrigirá bugs até July 26th, 2022 e bugs de segurança até January 24th, 2023

Laravel 9 - requer PHP 8.0 - 8.1 (corrigirá bugs até 08/08/2023 e bugs de segurança até 08/02/2024)

Laravel 10 - PHP 8.1 (lançamento em 02/2023, corrigirá bugs até 07/08/2024 e bugs de segurança até 07/02/2025)

As versões anteriores estão em descoberto, portanto sem correção de bugs

Controller Route Groups

Melhorias no grupo de rotas foram contribuídas por Luke Downing.

Agora você pode usar o método do controlador para definir o controlador comum para todas as rotas dentro do grupo. Então, ao definir as rotas, você só precisa fornecer o método do controlador que eles invocam:

```
use App\Http\Controllers\OrderController;
```

```
Route::controller(OrderController::class)->group(function () {
    Route::get('/orders/{id}', 'show');
    Route::post('/orders', 'store');
});
```

Bootstrap 5 Pagination Views

As views de paginação do Bootstrap 5 foram contribuídas por Jared Lewis.

O Laravel agora inclui views de paginação criadas usando Bootstrap 5. Para usar essas views em vez das views padrão do Tailwind, você pode chamar o método useBootstrapFive do paginador dentro do método de boot de sua classe App\Providers\AppServiceProvider, como a seguir:

```
use Illuminate\Pagination\Paginator;
```

```
/**
 * Bootstrap any application services.
 *
 * @return void
 */
```

```
public function boot()
{
    Paginator::useBootstrapFive();
}
```

Rota melhorada:list CLI Output

Rota melhorada:list CLI output foi contribuída pelo Nuno Maduro.

A saída da CLI route:list foi significativamente melhorada para a versão Laravel 9.x, oferecendo uma bela e nova experiência ao explorar suas definições de rota.

Novos Helpers

O Laravel 9.x apresenta duas novas e convenientes funções auxiliares que você pode usar em seu próprio aplicativo.

str

A função str retorna uma nova instância de Illuminate\Support\Stringable para a sequência dada. Esta função é equivalente ao método Str::of:

```
$string = str('Taylor')->append(' Otwell');
```

```
// 'Taylor Otwell'
```

Se nenhum argumento for fornecido para a função str, a função retornará uma instância de Illuminate\Support\Str:

```
$snake = str()->snake('LaravelFramework');
```

```
// 'laravel_framework'
```

to_route

A função to_route gera uma resposta HTTP de redirecionamento para uma determinada rota nomeada, fornecendo uma maneira expressiva de redirecionar para rotas nomeadas de suas rotas e controladores:

```
return to_route('users.show', ['user' => 1]);
```

Se necessário, você pode passar o código de status HTTP que deve ser atribuído ao redirecionamento e quaisquer cabeçalhos de resposta adicionais como terceiro e quarto argumentos para o método to_route:

```
return to_route('users.show', ['user' => 1], 302, ['X-Framework' => 'Laravel']);
```

<https://laravel.com/docs/9.x/releases>

O que esperar do Laravel 9

Programado para ser lançado até setembro de 2021, o Laravel 9 foi transferido para janeiro de 2022, tornando-o o primeiro lançamento de suporte de longo prazo (LTS) a ser introduzido após o ciclo de

lançamento de 12 meses. Este atraso resulta de muitas razões, que incluem mas não se limitam às seguintes:

1. Laravel usa variedades de projetos comunitários e cerca de nove bibliotecas Symfony. No entanto, a Symfony está planejando o lançamento da versão 6.0 até novembro de 2021. O atraso permitirá à equipe da Laravel incorporar esta nova versão da Symfony como parte do Laravel 9.
2. O atraso dará tempo à equipe para monitorar como Laravel interage com a nova versão da Symfony por dois meses. Também lhes dá espaço para corrigir quaisquer alterações ou bugs de quebra.
3. Finalmente, adiar o lançamento do Laravel 9 posiciona melhor a equipe Laravel para futuros lançamentos anuais. Isso dará à equipe dois meses de tempo de rampa adicional após os lançamentos da Symfony.

Por estas razões, você pode ver que o atraso no seu lançamento vale a pena esperar.

Migração anônima de Stub

Laravel define para fazer da migração anônima de stub o comportamento padrão quando você executa o comando de migração popular:

```
php artisan make:migration
```

O recurso de migração anônima de stub foi lançado pela primeira vez no Laravel 8.37 para resolver este problema do Github. O problema é que múltiplas migrações com o mesmo nome de classe podem causar problemas ao tentar recriar o banco de dados a partir do zero. O novo recurso de migração de stub elimina colisões de nomes de classes de migração.

Do Laravel 8.37, o framework agora suporta arquivos de migração de classe anônima, e no Laravel 9, será o comportamento padrão.

Nova interface do Query Builder

Com o novo Laravel 9, o tipo hinting é altamente confiável para refatoração, análise estática e autocompletar código em seus IDEs. Isso se deve à falta de uma interface compartilhada ou herança entre Query\Builder, Eloquent\Builder e Eloquent\Relation. Ainda assim, com o Laravel 9, os desenvolvedores agora podem aproveitar a nova interface do construtor de consultas para a sugestão de tipo, refatoração e análise estática.

Breeze API with Next.js

O Laravel 9 inclui uma implementação de front-end complementar do Next.js em seu kit inicial do Breeze. Ao usar este scaffolding do kit inicial, você pode criar aplicativos Laravel que servem como back-end e front-end JavaScript usando a autenticação Laravel Sanctum.

Checked and selected Blade directives

Não sei quantas vezes pesquisei no Google "Como marcar uma caixa de seleção no Laravel". O Laravel v9 tornou isso mais fácil. Agora você pode usar a diretiva @checked para definir uma caixa de seleção como marcada. Se for avaliado como verdadeiro, será verificado o eco.


```
<input type="checkbox"
      name="optin"
      value="optin"
      @checked(old('optin', $user->optin)) />
```

Há também uma diretiva @selected semelhante para definir a opção selecionada em um select.

```
<select name="notification">
  @foreach ($notifications as $notification)
    <option value="{{ $notification }}" @selected(old('notification') == $notification)>
      {{ $notification }}
    </option>
  @endforeach
</select>
```

2 - Planejamento

Algo que colabora com a nossa produtividade é elaborar um planejamento sempre que iniciar um site ou aplicativo. Este planejamento deve ser mais bem elaborado quando o aplicativo for mais importante. Isso evitará retrabalho para corrigir algo que não previmos no início.

Ganhamos em produtividade e em eficiência se nossa primeira providência ao criar um aplicativo/site for a elaboração de um roteiro/planejamento do mesmo, com os passos que precisaremos seguir e com bastante informações e detalhes sobre o mesmo.

Antes de iniciar, colete do usuário/cliente tudo que ele precisa para o aplicativo. Anotar este roteiro e usar como um guia para ir construindo o aplicativo e ticando o que já fez:

- planejamento detalhado do banco de dados:
 - Criação das tabelas, campos, constraints, tamanhos, etc
 - Relacionamentos
 - Ao final gerar um DER irá ajudar
 - etc
- planejamento detalhado do aplicativo:
 - Rotas
 - Migrations
 - Seeders
 - Controllers
 - Models
 - Views, com layouts e includes
 - etc

Somente após a conclusão do projeto do banco e do aplicativo, devemos iniciar a execução de cada etapa planejada. É bom também criar telas pretendidas para o aplicativo, de acordo com o que o cliente deseja

Lembre que isto tanto organiza nosso trabalho quanto o torna mais eficiente. Se temos um roteiro basta ir seguindo as etapas.

3 – Maneiras de Instalar o laravel 9

Instalação com docker/SAIL

Laravel Sail

Laravel Sail é uma interface de linha de comando simples para interagir com o ambiente de desenvolvimento Docker padrão do Laravel. O Sail fornece um ótimo ponto de partida para criar um aplicativo Laravel usando PHP, MySQL e Redis sem exigir experiência prévia com Docker.

Em sua essência, Sail é o arquivo docker-compose.yml e o script sail que é armazenado na raiz do seu projeto. O script sail fornece uma CLI com métodos convenientes para interagir com os contêineres do Docker definidos pelo arquivo docker-compose.yml.

O Laravel Sail é compatível com macOS, Linux e Windows (via WSL2).

Para instalar o sail num aplicativo de testes existente:
composer require laravel/sail --dev

```
php artisan sail:install
```

```
./vendor/bin/sail up
```

Start e stop

```
sail up
```

```
sail up -d
```

```
sail stop
```

Ou

```
curl -s "https://laravel.build/example-app" | bash  
cd example-app
```

```
./vendor/bin/sail up
```

Detalhes

<https://laravel.com/docs/9.x/sail>

<https://laravel.com/docs/9.x>

Instalar com o instalador global no Ubuntu

```
composer global require laravel/installer
```

```
export PATH="~/config/composer/vendor/bin:$PATH"
```

```
php artisan serve
```

```
php artisan serve --port=8080
```

Criar um aplicativo

```
laravel new crud
```

```
cd crud
```

Criando e usando scaffold da autenticação

```
composer require laravel/ui
```

Alternativas de suporte no Front-end (scaffold, login e registro):

Bootstrap

```
php artisan ui bootstrap
```

```
php artisan ui bootstrap --auth
```

Vue

```
php artisan ui vue
```

```
php artisan ui vue --auth
```

React

```
php artisan ui react
```

```
php artisan ui react --auth
```

```
npm install
```

```
npm run dev
```

```
php artisan migrate
```

```
php artisan serve
```

Instalar o Docker no Windows

1) Se tiver o Docker instalado, remova!

2) Habilite o WSL no Windows 10

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart  
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

3) Habilitar o WSL para a versão 2

```
wsl --set-default-version 2
```

4) Instalar o Ubuntu na Microsoft Store

5) Instalar o Windows Terminal

6) Desabilitar o Hyper-v

7) Criar o arquivo .wslconfig em "C:\Users\<seu_usuario>"

```
[wsl2]
```

```
memory=8GB
```

```
processors=4
```

```
swap=2GB
```

8) Instalar o Docker

<https://gist.github.com/carlosfgti/12d9f174b5727afd1d563fb4e278854a>

4 - Convenções no Laravel

Ao seguirmos as convenções definidas por um framework estamos garantindo boa performance, facilidade de manutenção, baixa curva de aprendizagem ao contratar programadores, segurança e escalabilidade.

Um banco de dados que não atende a convenção pode gerar confusão e dificuldade de manutenção do código da aplicação.

Ao não utilizar a convenção de nomenclatura de banco de dados do framework é possível que o programador tenha trabalho para definir em cada model qual é a sua respectiva tabela. Imagine um cenário de 100 models ou mais...

Uma aplicação que segue as convenções de seu framework garante o principal objetivo da utilização da ferramenta: produtividade. Com isso também garante uma pequena curva de aprendizagem ao trocar ou expandir a equipe além, claro, dos fatores de segurança, performance e organização que são essenciais para uma aplicação robusta e escalável.

<https://medium.com/@carloscarneiropmw/overview-das-conven%C3%A7%C3%B5es-do-laravel-d2e76b3db38a>

Ajuda muito seguir algumas convenções do Laravel

- Nomes de tabelas e campos - no plural e tudo em minúsculas (snake case)
- Model - singular e inicial maiúscula (CamelCase)
- Controller - singular e inicial maiúscula e todas as ações em minúsculas (CamelCase com sufixo Controller)
- Views - mesmos nomes das actions do controller
- Rotas - mesmos nomes dos actions do controller
- Usa por padrão os timestamps: created_at e updated_at
- PrimaryKey - id
- Foreign Key - tabela_id. Ex: cliente_id
- Preferir usar nomes de variáveis e de arquivos, como também de tabelas e campos em inglês
- Nomes padrões dos actions dos controllers: index, create, show, store, edit, update, destroy

Quando não seguirmos algumas das convenções podemos dizer ao Laravel para usar o nome que escolhemos. Isso se faz no Model, mas somente é aconselhado fazer em caso de necessidade.

Exemplo:

```
protected $table = "minha_tabela";
protected $primaryKey = "minhaChave";
etc
```

```
//Tipo da chave primária, sendo string
protected $keyType = 'string';
```

```
// Para usar nomes diferentes nos timestamps
const CREATED_AT = 'creation_date';
const UPDATED_AT = 'last_update';
```

Validação

É muito comum os iniciantes acharem prático setar validações dentro dos models mas isso significa que o framework somente irá validar os dados no ato de salvar no banco e se por acaso não forem dados válidos teria ocorrido um processamento desnecessário e a aplicação lançará uma exceção — crash — possivelmente deixando o usuário frustrado.

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class Cliente extends Model
{
    protected $table = 'clientes';
    protected $primaryKey = 'id';
    protected $fillable = ['nome', 'email'];
    public $timestamp = true;
    protected $connection = 'connection-name'; //Somente caso use uma conexão diferente da
default
}
```

Routes

resources no plural e minúsculas

```
Route::get('/users', 'UserController@index');
Route::resource('photos', 'PhotoController');
```

<https://laravel.com/docs/9.x/eloquent#eloquent-model-conventions>

Ao seguirmos as convenções definidas por um framework estamos garantindo boa performance, facilidade de manutenção, baixa curva de aprendizagem ao contratar programadores, segurança e escalabilidade.

Ao não utilizar a convenção de nomenclatura de banco de dados do framework o programador terá mais trabalho, pois precisará definir em cada model o nome da respectiva tabela. Imagine um aplicativo com muitas tabelas.

Quando um programador usa as convenções do Laravel ele conseguirá criar o aplicativo mais rapidamente e com mais facilidade.

Outras convenções do Laravel

- Nomes de tabelas e campos - no plural e tudo em minúsculas (snake case). Ex: my_clients
- Model - singular e inicial maiúscula (CamelCase) . Ex.: Client
- Controller - singular e inicial maiúscula e todas as ações em minúsculas (CamelCase com sufixo Controller) . Ex: ClientController
- Migrations: create_clients_table
- Tabela pivô: role_user (singular e em ordem alfabética). Não user_role, nem users_roles
- Seeders Ex: CreateUsersSeeder
- Views - mesmos nomes das actions do controller : Ex: clients/index.blade.php
- Rotas - mesmos nomes dos actions do controller Ex: clients, edit
- Usa por padrão os timestamps: created_at e updated_at
- PrimaryKey - id
- Foreign Key - tabela_id. Ex: client_id
- Preferir usar nomes todos em inglês
- Nomes padrões dos actions dos controllers: index, create, show, store, edit, update, destroy
- Nomes compostos de variáveis do blade: \$minha_var. Não use \$minha-var.

Quando não seguirmos algumas das convenções podemos dizer ao Laravel para usar o nome da tabela que escolhemos. Isso se faz no Model, mas somente é aconselhado fazer em caso de necessidade. Exemplo:

```
protected $table = "minha_tabela";
protected $primaryKey = "minhaPK";
etc
```

```
//Tipo da chave primária, sendo string
protected $keyType = 'string';
// Para usar nomes diferentes nos timestamps
const CREATED_AT = 'creation_date';
const UPDATED_AT = 'last_update';
```

Variáveis e Métodos/funções

Variáveis e métodos em camelCase

Exemplos: \$users = ..., \$bannedUsers =

Bad examples: \$all_banned_users = ..., \$Users=....

Propriedades de Models

Minúsculas, snake_case.

Exemplos: \$this->updated_at, \$this->title.

Relacionamentos

hasOne ou belongsTo

camelCase a primeira letra em minúsculas

Exemplos:

public function postAuthor(),

public function phone().

hasMany, belongsToMany, hasManyThrough

Traits

Traits devem ser adjetivos.

Exemplos: Notifiable, Dispatchable, etc.

Arquivos das views Blade
Minúsculas e snake_case

Exemplos: all.blade.php, all_posts.blade.php, etc

Referências

<https://veesworld.com/laravel/laravel-naming-conventions>

5 – Bancos de Dados

Migrations

As migrações são como controle de versão para seu banco de dados, permitindo que sua equipe defina e compartilhe a definição do esquema de banco de dados do aplicativo. Se você já teve que dizer a um colega de equipe para adicionar manualmente uma coluna ao esquema de banco de dados local depois de obter suas alterações do controle de origem, você enfrentou o problema que as migrações de banco de dados resolvem.

A Facade Laravel Schema fornece suporte agnóstico de banco de dados para criar e manipular tabelas em todos os SGBDs suportados pelo Laravel. Normalmente, as migrações usarão essa facade para criar e modificar tabelas e colunas do banco de dados.

Recriar todas as migrations removendo inclusive os registros das tabelas e executar todos os seeders

```
php artisan migrate:fresh --seed
```

Gerando um arquivo de migrations

```
php artisan make:migration create_posts_table
```

Também podemos fazer assim

```
php artisan make:migration posts --table
```

E assim

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

A primeira forma (create_posts_table) gera a classe já com o esqueleto do método up():

```
public function up()
{
    Schema::create('clients', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

Enquanto a primeira forma, usando --table, gera um método up() vazio, então é melhor usar a primeira forma.

A terceira forma é similar a primeira

Estrutura básica depois ajustada com os campos title e description

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```

return new class extends Migration
{
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->text('description');
            $table->tinyInteger('status')->default('1');
            $table->timestamps();
        });
    }
    public function down()
    {
        Schema::dropIfExists('posts');
    }
};

```

Executar a migration criada, criando a respectiva tabela no banco

```
php artisan migrate
```

Com o comando acima criamos todas as migrations existentes em database/migrations.

Para executar apenas uma única migrations:

```
php artisan migrate --path=/database/migrations/2022_04_01_064006_create_posts_table.php
```

Verificar o estado de todas as migrations criadas

```
php artisan migrate:status
```

Rollback

```
php artisan migrate:rollback
```

<https://www.itsolutionstuff.com/post/how-to-create-migration-in-laravel-9example.html>

Criar um model juntamente com uma migration

```
php artisan make:model Post -m
```

Similar a usar

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

```
php artisan make:migration create_posts_table
```

Criará

app/Models/Post.php
database/migrations/xxx-posts-

Excluir todas as migrations da tabela e seus registros e criar novamente

php artisan migrate:fresh

Tipos de campos suportados nas migrations

bigIncrements
bigInteger
binary
boolean
char
dateTimeTz
dateTime
date
decimal
double
enum
float
foreignId
foreignIdFor
foreignUuid
geometryCollection
geometry
id
increments
integer
ipAddress
json
jsonb
lineString
longText
macAddress
mediumIncrements
mediumInteger
mediumText
morphs
multiLineString
multiPoint
multiPolygon
nullableMorphs
nullableTimestamps
nullableUuidMorphs
point

polygon
rememberToken
set
smallIncrements
smallInteger
softDeletesTz
softDeletes
string
text
timeTz
time
timestampTz
timestamp
timestampsTz
timestamps
tinyIncrements
tinyInteger
tinyText
unsignedBigInteger
unsignedDecimal
unsignedInteger
unsignedMediumInteger
unsignedSmallInteger
unsignedTinyInteger
uuidMorphs
uuid
year

Modificadores de campos

Permitir que um campo de email seja nulo

```
$table->string('email')->nullable();
```

Não permitir que um campo de email seja nulo

```
$table->string('email')->notNullable();
```

Column Order

Quando usando o MySQL, o método `after` deve ser usado para adicionar campos após um existente campo num schema:

```
$table->after('password', function ($table) {
    $table->string('address_line1');
    $table->string('address_line2');
    $table->string('city');
});
```

Alterando o tamanho de campos

```
Schema::table('users', function (Blueprint $table) {
    $table->string('name', 50)->change();
});
```

```
Schema::table('users', function (Blueprint $table) {
    $table->string('name', 50)->nullable()->change();
});
```

<https://laravel.com/docs/9.x/migrations>

Alteração de tabelas através de migrations

São classes que agem nas tabelas como faz o comando alter table do SQL.

Adicionar o campo latitude para a tabela existente cities

Para adicionar uma nova coluna à tabela existente usando a migração laravel. Vamos abrir seu terminal e criar um arquivo de migração usando o comando abaixo:

```
php artisan make:migration add_latitude_to_cities
```

```
...
public function up()
{
    Schema::table('cities', function (Blueprint $table) {
        $table->string('latitude', 20)->nullable();
    });
}

public function down()
{
    Schema::table('cities', function (Blueprint $table) {
        $table->dropColumn(['latitude']);
    });
}
}
```

Alterar a tabela

php artisan migrate

Adicionar várias colunas na tabela existente

Se você precisar adicionar várias colunas, precisará adicionar o seguinte comando no terminal.

php artisan make:migration add_multiple_column_to_notes

O comando acima criará um arquivo de migração, então vá para a pasta database/migration e abra seu arquivo e adicione a coluna de acordo com você

```
...
public function up()
{
    Schema::table('notes', function (Blueprint $table) {
        $table->string('short_description', 50)->notNullable();
        $table->string('type')->notNullable();
        $table->text('long_description')->nullable();
    });
}

public function down()
{
    Schema::table('notes', function (Blueprint $table) {
        $table->dropColumn(['short_description', 'type', 'long_description']);
    });
}
}
```

php artisan migrate

<https://www.tutsmake.com/laravel-migration-add-a-column-to-an-existing-table/>

Jogar para o banco todas as migrations e os seeders

php artisan migrate:fresh --seed

Factory no Laravel

Definindo Factory do Model

Primeiro, vamos falar sobre as factories do model Eloquent. Ao testar, em desenvolvimento, pode ser necessário inserir alguns registros em seu banco de dados antes de executar seu teste. Em vez de especificar manualmente o valor de cada campo ao criar esses dados de teste, o Laravel permite definir um conjunto de atributos padrão para cada um de seus models Eloquent usando factories de models.

Exemplo

```
<?php
namespace Database\Factories;

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

class UserFactory extends Factory
{
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
            'email_verified_at' => now(),
            'password' => '$2y$10$92IXUNpkjO0rQQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', //
            'remember_token' => Str::random(10),
        ];
    }
}
```

Como você pode ver, em sua forma mais básica, as factories são classes que estendem a classe Factory base do Laravel e definem um método definition(). O método definition() retorna o conjunto padrão de valores de atributo que devem ser aplicados ao criar um model usando a factory.

Por meio da propriedade faker, as factories têm acesso à biblioteca Faker do PHP (fakerphp/faker), que permite gerar convenientemente vários tipos de dados aleatórios para teste.

Você pode definir a localidade do Faker do seu aplicativo adicionando uma opção faker_locale ao seu arquivo de configuração config/app.php.

Convenções de descoberta do model e factory

Depois de definir suas factories, você pode usar o método estático da factory fornecido aos seus models pelo atributo Illuminate\Database\Eloquent\Factories\HasFactory para instanciar uma instância de factory para esse model.

O método HasFactory do trait usará as convenções para determinar a factory apropriada para o model ao qual o trait é atribuído. Especificamente, o método procurará uma factory no namespace Database\Factories que tenha um nome de classe que corresponda ao nome do model e tenha o sufixo Factory.

<https://laravel.com/docs/9.x/database-testing#factory-and-model-discovery-conventions>

Criando dados faker com Factory

Laravel 9 gera dados fictícios usando factory, migrates e seeders.

Sempre que você quiser testar seu aplicativo Laravel, você precisa de registros falsos no banco de dados.

Criar o aplicativo

laravel new factory

Criar o banco e configurar no .env

Criar a migration com o model

cd factory

php artisan make:model Post -m

databases/migrations/xxx_create_posts_table.php

Editar a migration criada e adicionar os campos

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

Edite o model gerado:

app/Models/Post.php

E deixe assim:

```
<?php
namespace App\Models;

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

class Post extends Model
{
    use HasFactory;

    protected $fillable = [
        'title', 'description'
    ];
}
```

Criar a classe factory

php artisan make:factory PostFactory --model=Post

Edite database/factories/PostFactory.php

```
<?php
namespace Database\Factories;

use App\Models\Post;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class PostFactory extends Factory
{
    protected $model = Post::class;

    public function definition()
    {
        return [
            'title' => $this->faker->title,
            'description' => $this->faker->text,
        ];
    }
}
```

Criar um seeder

php artisan make:seeder PostSeeder

Atualizar DatabaseSeeder:

databases/seeder/DatabaseSeeder.php

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        \App\Models\Post::factory(100)->create();
    }
}
```

Criar banco e configurar .env

composer dump-autoload

php artisan migrate

php artisan db:seed

Adicionar 20 registros para a tabela usando o tinker

composer dump-autoload

php artisan tinker

Post::factory()->count(100)->create()

<https://www.tutsmake.com/laravel-9-factory-generate-dummy-data-tutorial/>

<https://onlinewebtutorblog.com/laravel-9-seed-database-using-faker-library-tutorial/>

Seeder no Laravel 9

Até a versão 7 a pasta que guardava os seeders era chamada de seeds. A partir da versão 8 ela foi renomeada para seeders.

O Laravel inclui a capacidade de popular seu banco de dados com dados usando classes de seeders/sementes. Todas as classes de seeders são armazenadas no diretório database/seeders. Por padrão, uma classe DatabaseSeeder é definida para você. A partir dessa classe, você pode usar o método call para executar outras classes de propagação, permitindo controlar a ordem de propagação dos dados.

Uma classe seeder contém apenas um método por padrão: o run(). Este método é chamado quando o comando db:seed Artisan é executado. Dentro do método run, você pode inserir dados em seu banco de dados como desejar. Você pode usar o construtor de consultas/query-builder para inserir dados manualmente associado ou não com a biblioteca faker ou pode usar factories de models do Eloquent.

Chamando Seedes Adicionais

Dentro da classe DatabaseSeeder, você pode usar o método call() para executar classes seeders adicionais. O uso do método call permite que você divida a propagação do banco de dados em vários arquivos para que nenhuma classe de seeder seja muito grande. O método call() aceita um array de classes seeder que devem ser executadas:

```
public function run()
{
    $this->call([
        UserSeeder::class,
        PostSeeder::class,
        CommentSeeder::class,
    ]);
}
```

Criar migrate para a tabela articles

```
php artisan make:migration create_articles_table
```

Editar e adicionar os campos

```
$table->id();
$table->string('title', 100);
$table->string('email', 254)->nullable();
$table->string('body', 255);
$table->timestamps();
```

Criar arquivo de seeder

php artisan make:seeder ArticleSeeder

databases/seeder/ArticleSeeder

Criará algo como

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;

class ArticleSeeder extends Seeder
{
    public function run()
    {
        //
    }
}
```

Mudar para algo como

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Str;

class ArticleSeeder extends Seeder
{
    public function run()
    {
        for($count=0;$count<10;$count++){
            DB::table('articles')->insert([
                'title' => Str::random(10),
                'email' => Str::random(10).'@gmail.com',
                'body' => Str::random(100),
                'updated_at' => date("Y-m-d H:i:s"),
                'created_at' => date("Y-m-d H:i:s"),
            ]);
        }
    }
}
```

Execute

```
php artisan db:seed --class=ArticleSeeder
```

Criará os 10 registros

Ou rodar a migration juntamente com o seeder:

```
php artisan migrate:fresh --seed
```

<https://readerstacks.com/how-to-create-seeders-in-laravel-9-with-example/>

Criando um único registro

```
public function run()
{
    Employee::create([
        'name' => 'John Doe',
        'email' => 'doejohn@gmail.com',
        'phone' => '1234567890',
        'department' => 'hr',
        'dob' => '2010-01-03',
    ]);
}
```

Editar

database/seeds/DatabaseSeeder.php

```
<?php
use Illuminate\Database\Seeder;
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(EmployeeSeeder::class);
    }
}
```

Executar todos os seeders

```
php artisan db:seed
```

Executar apenas um seeder:

```
php artisan db:seed --class=ClientsSeeder
```

Executar todas as migrations e todos os seeders:

```
php artisan migrate:fresh --seed
```

<https://www.positronx.io/laravel-database-seeder-tutorial-example/>

Criar projeto

```
composer create-project laravel/laravel laravel-seeder --prefer-dist
```

Criar banco e conf .env

Criar model e migration

```
php artisan make:model Student -m
```

Editar a migration e add

```
$table->string('name');  
$table->string('phone');  
$table->string('email')->unique();  
$table->string('dob');  
$table->string('stream');
```

Editar o model e

```
class Student extends Model  
{  
    use HasFactory;  
    public $fillable = [  
        'name',  
        'phone',  
        'email',  
        'dob',  
        'stream',  
    ];  
}
```

Executar

```
php artisan migrate
```

Criar

```
php artisan make:seeder StudentSeeder
```

Editar o seeder criado e atualizar

```
public function run()
{
    Student::create([
        'name' => 'John Addison',
        'phone' => '88992299112',
        'email' => 'john@yahoo.com',
        'dob' => '2011-02-04',
        'stream' => 'Physics',
    ]);
}
```

Rodar

```
php artisan db:seed --class=StudentSeeder
```

Editar

database/seeds/DatabaseSeeder.php

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(StudentSeeder::class);
    }
}
```

Rodar

```
php artisan db:seed
```

<https://remotestack.io/how-to-create-database-seeder-in-laravel-application/>

composer create-project laravel/laravel myblog

Criar banco e conf .env

Criar

```
php artisan make:migration create_students_table
```

Editar


```

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('students', function (Blueprint $table) {
            $table->id();
            $table->string("name", 120);
            $table->string("email", 50)->nullable();
            $table->string("mobile", 50)->nullable();
            $table->integer("age");
            $table->enum("gender", ["male", "female", "others"]);
            $table->text("address_info");
            $table->timestamp("created_at")->useCurrent();
            $table->timestamp("updated_at")->useCurrent();
        });
    }

    public function down()
    {
        Schema::dropIfExists('students');
    }
};

```

Rodar

php artisan migrate

Criar model

php artisan make:model Student

Criar factory

php artisan make:factory StudentFactory

Editar

```

<?php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class StudentFactory extends Factory
{
    public function definition()
    {
        return [
            "name" => $this->faker->name(),
            "email" => $this->faker->safeEmail,
            "mobile" => $this->faker->phoneNumber,
            "age" => $this->faker->numberBetween(25, 45),
            "gender" => $this->faker->randomElement([
                "male",
                "female",
                "others"
            ]),
            "address_info" => $this->faker->address
        ];
    }
}

```

Editar o DatabasSeeder.php

```

<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        \App\Models\Student::factory(100)->create();
    }
}

```

Rodar

php artisan db:seed

<https://onlinewebtutorblog.com/laravel-9-seed-database-using-faker-library-tutorial/>

Usando o FakerRestaurant

<https://github.com/jzonta/FakerRestaurant>

Instalar

`composer require jzonta/faker-restaurant`

```
$faker = \Faker\Factory::create();
$faker->addProvider(new \FakerRestaurant\Provider\pt_BR\Restaurant($faker));
```

```
// Generator
$faker->foodName();    // A random Food Name
$faker->beverageName(); // A random Beverage Name
$faker->dairyName();   // A random Dairy Name
$faker->vegetableName(); // A random Vegetable Name
$faker->fruitName();   // A random Fruit Name
$faker->meatName();    // A random Meat Name
$faker->sauceName();   // A random Sauce Name
```

`php artisan make:seeder ProdutosSeeder`

```
<?php
namespace Database\Seeders;
```

```
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
```

```
class ProdutosSeeder extends Seeder
{
    public function run()
    {
        $faker = \Faker\Factory::create();
        $faker->addProvider(new \FakerRestaurant\Provider\pt_BR\Restaurant($faker));
```

```
        for ($i=0; $i < 100; $i++) {
            DB::table('produtos')->insert([
                'descricao' => $faker->fruitName(),
                'estoque_minimo' => $faker->numberBetween($min = 5, $max = 20),
                'estoque_maximo' => $faker->numberBetween($min = 20, $max = 100),
            ]);
        }
    }
}
```

Editar o DatabaseSeeder

```
public function run()
{
    // \App\Models\User::factory(10)->create();
    $this->call(ProdutosSeeder::class);
}
```

`php artisan db:seed`

`php artisan make:model OrdemDePagamento -a`

```
public function up()
{
    Schema::create('ordem_de_pagamentos', function (Blueprint $table) {
        $table->increments('id');
        $table->string('nome');
        $table->string('CPF', 14);
        $table->string('CNPJ', 14);
        $table->string('telefone', 15);
        $table->string('endereco');
        $table->string('cidade');
        $table->string('UF', 2);
        $table->decimal('valor', 9, 2);
        $table->timestamp('created_at')->useCurrent();
        $table->timestamp('updated_at')->useCurrent();
    });
}
```

`php artisan migrate`

`php artisan make:seeder OrdemSeeder`

```
use Illuminate\Support\Facades\DB;
```

```
public function run()
{
    $faker = \Faker\Factory::create('pt_BR');
    for ($i=0; $i<50; $i++) {
        DB::table('ordem_de_pagamentos')->insert([
            'nome' => $faker->name,
            'CPF' => $faker->cpf,
            'CNPJ' => $faker->cnpi(false),
            'telefone' => sprintf('(0%s) %s', $faker->areaCode, $faker->landline),
            'endereco' => $faker->address,
```

```

        'cidade' => $faker->city,
        'UF' => $faker->stateAbbr,
        'valor' => $faker->randomFloat(2, 12, 150000),
    ]);
}
}

```

database/seeds/DatabaseSeeder.php

```

public function run()
{
    $this->call(OrdemDePagamentoSeeder::class);
}

```

php artisan db:seed --class=OrdemSeeder

Exemplo de DatabaseSeeder.php

```

<?php
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(PostsSeeder::class);
    }
}

```

Bancos de dados no Laravel

Quase todos os aplicativos modernos da Web interagem com um banco de dados. O Laravel torna a interação com bancos de dados extremamente simples em uma variedade de bancos de dados suportados usando SQL bruto, um construtor de consultas/query-builder fluente e o Eloquent ORM. Atualmente, o Laravel fornece suporte primário para cinco SGBDs:

- MariaDB 10.2+ (Version Policy)
- MySQL 5.7+ (Version Policy)
- PostgreSQL 10.0+ (Version Policy)
- SQLite 3.8.8+
- SQL Server 2017+ (Version Policy)

Consulta

Para executar uma consulta SELECT básica, você pode usar o método select na facade do banco de dados:

```
<?php
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function index()
    {
        $users = DB::select('select * from users where active = ?', [1]);
        return view('users.index', ['users' => $users]);
    }
}

use Illuminate\Support\Facades\DB;

$users = DB::select('select * from users');

foreach ($users as $user) {
    echo $user->name;
}
```

Insert

```
use Illuminate\Support\Facades\DB;
```

```
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Marc']);
```

Update

```
$affected = DB::update(
    'update users set votes = 100 where name = ?',
    ['Anita']
);
```

Delete

```
use Illuminate\Support\Facades\DB;
```

```
$deleted = DB::delete('delete from users');
```

Transações

```
use Illuminate\Support\Facades\DB;
```

```
DB::beginTransaction();
```

```
DB::rollBack();
```

```
DB::commit();
```

Conectar via artisan

```
php artisan db mysql
```

Query Builder

O construtor de consultas/query builder de banco de dados do Laravel fornece uma interface conveniente e fluente para criar e executar consultas de banco de dados. Ele pode ser usado para executar a maioria das operações de banco de dados em sua aplicação e funciona perfeitamente com todos os sistemas de banco de dados suportados pelo Laravel.

O construtor de consultas Laravel usa o PDO para proteger seu aplicativo contra ataques de injeção de SQL. Não há necessidade de limpar ou sanitizar as strings passadas ao construtor de consultas como ligações de consulta.

Executando consultas

Recebendo todos os registros de uma tabela

```
<?php
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function index()
    {
        $users = DB::table('users')->get();
        return view('users.index', ['users' => $users]);
    }
}
```

Podemos acessar cada campo

```
use Illuminate\Support\Facades\DB;

$users = DB::table('users')->get();

foreach ($users as $user) {
    echo $user->name;
}
```

Recebendo o valor de um campo

```
$user = DB::table('users')->where('name', 'John')->first();
return $user->email;
$email = DB::table('users')->where('name', 'John')->value('email');
$user = DB::table('users')->find(3);

use Illuminate\Support\Facades\DB;

$titles = DB::table('users')->pluck('title');

foreach ($titles as $title) {
    echo $title;
}

$titles = DB::table('users')->pluck('title', 'name');
```



```
foreach ($titles as $name => $title) {
    echo $title;
}
```

Funções de Agregação

```
use Illuminate\Support\Facades\DB;

$users = DB::table('users')->count();

$price = DB::table('orders')->max('price');

$users = DB::table('users')
    ->select(DB::raw('count(*) as user_count, status'))
    ->where('status', '<>', 1)
    ->groupBy('status')
    ->get();
```

Join

```
use Illuminate\Support\Facades\DB;

$users = DB::table('users')
    ->join('contacts', 'users.id', '=', 'contacts.user_id')
    ->join('orders', 'users.id', '=', 'orders.user_id')
    ->select('users.*', 'contacts.phone', 'orders.price')
    ->get();
```

```
$users = DB::table('users')
    ->leftJoin('posts', 'users.id', '=', 'posts.user_id')
    ->get();
```

```
$users = DB::table('users')
    ->rightJoin('posts', 'users.id', '=', 'posts.user_id')
    ->get();
```

```
use Illuminate\Support\Facades\DB;

$first = DB::table('users')
    ->whereNull('first_name');

$users = DB::table('users')
    ->whereNull('last_name')
    ->union($first)
    ->get();
```

```
$users = DB::table('users')  
    ->where('votes', '=', 100)  
    ->where('age', '>', 35)  
    ->get();
```

```
$users = DB::table('users')  
    ->where('votes', '>', 100)  
    ->orWhere(function($query) {  
        $query->where('name', 'Abigail')  
            ->where('votes', '>', 50);  
    })  
    ->get();
```

Criar aplicativo de testes

Criar um aplicativo para testes de bancos de dados

```
laravel new testes
```

```
cd testes
```

Usar Model User

Rodar

```
php artisan migrate
```

```
php artisan tinker
```

```
User::factory()->count(100)->create()
```

Controller

```
php artisan make:controller UserController -r
```

```
use App\Models\User;
```

```
class UserController extends Controller
{
    public function index()
    {
        $users = User::get();
        foreach ($users as $user) {
            echo $user->name.'<br>';
        }
    }
}
```

```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\UserController;
```

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

```
Route::resource('users', UserController::class);
```

```
php artisan serve
```

```
http://127.0.0.1:8000/users
```

Eloquent ORM

O mapeador objeto-relacional (ORM) para Laravel é chamado Eloquent, e é uma das melhores características da Laravel, pois permite uma interação perfeita com o modelo de dados e banco de dados de escolha.

Com Eloquent, Laravel abstrai cada obstáculo envolvendo interação e escrita de consultas SQL complexas para acessar dados do seu banco de dados.

O Laravel inclui o Eloquent, um mapeador objeto-relacional (ORM) que torna agradável interagir com seu banco de dados. Ao usar o Eloquent, cada tabela do banco de dados possui um "Model" correspondente que é usado para interagir com essa tabela. Além de recuperar registros da tabela do banco de dados, os models do Eloquent também permitem inserir, atualizar e excluir registros da tabela.

Coleções

Como vimos, os métodos Eloquent `get` e `all` recuperam vários registros do banco de dados. Então esses métodos não retornam um array PHP simples. Em vez disso, uma instância de `Illuminate\Database\Eloquent\Collection` é retornada.

A classe Eloquent Collection estende a classe base `Illuminate\Support\Collection` do Laravel, que fornece uma variedade de métodos úteis para interagir com coleções de dados. Por exemplo, o método de rejeição pode ser usado para remover modelos de uma coleção com base nos resultados de um encerramento invocado:

```
$clients = Client::where('name', 'Laury Davis')->get();
$clients = $clients->reject(function ($client) {
    return $client->cancelled;
});
```

```
<?php
namespace App\Http\Controllers;
```

```
use App\Models\Client;
use Illuminate\Http\Request;
```

```
class ClientController extends Controller
{
    public function index()
    {
        $clients = Client::where('id', 1)->get();

        foreach ($clients as $client) {
            echo $client->name.'<br>';
        }
    }
}
```

A maioria dos métodos retorna instâncias Illuminate\Database\Eloquent\Collection; no entanto, alguns métodos, como modelKeys, retornam uma instância Illuminate\Support\Collection.

contains
diff
except
find
fresh
intersect
load
loadMissing
modelKeys
makeVisible
makeHidden
only
toQuery
unique

```
contains($key, $operator = null, $value = null)
```

```
Flight::where('departed', true)
->chunkById(200, function ($flights) {
    $flights->each->update(['departed' => false]);
}, $column = 'id');
```

Subqueries select

```
use App\Models\Destination;
use App\Models\Flight;
```

```
return Destination::addSelect(['last_flight' => Flight::select('name')
->whereColumn('destination_id', 'destinations.id')
->orderByDesc('arrived_at')
->limit(1)
])->get();
```

```
return Destination::orderByDesc(
    Flight::select('arrived_at')
    ->whereColumn('destination_id', 'destinations.id')
    ->orderByDesc('arrived_at')
    ->limit(1)
)->get();
```

```
use App\Models\Flight;
```

```
// Retrieve a model by its primary key...
$flight = Flight::find(1);
```

```
// Retrieve the first model matching the query constraints...
$flight = Flight::where('active', 1)->first();
```

```
// Alternative to retrieving the first model matching the query constraints...
$flight = Flight::firstWhere('active', 1);
```

```
$count = Flight::where('active', 1)->count();
```

```
$max = Flight::where('active', 1)->max('price');
```

Insert

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use App\Http\Controllers\Controller;
```

```
use App\Models\Flight;
```

```
use Illuminate\Http\Request;
```

```
class FlightController extends Controller
```

```
{
```

```
    /**
```

```
     * Store a new flight in the database.
```

```
     *
```

```
     * @param \Illuminate\Http\Request $request
```

```
     * @return \Illuminate\Http\Response
```

```
     */
```

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

```
    {
```

```
        // Validate the request...
```

```
        $flight = new Flight;
```

```
        $flight->name = $request->name;
```

```
        $flight->save();
```

```
    }
```

```
}
```

Update

```
use App\Models\Flight;
```

```
$flight = Flight::find(1);
```

```
$flight->name = 'Paris to London';
```

```
$flight->save();
```

Delete

```
use App\Models\Flight;  
  
$flight = Flight::find(1);  
$flight->delete();  
  
Flight::truncate();
```

Incremento e Decremento

```
DB::table('users')->increment('votes');  
  
DB::table('users')->increment('votes', 5);  
  
DB::table('users')->decrement('votes');  
  
DB::table('users')->decrement('votes', 5);
```

You may also specify additional columns to update during the operation:

```
DB::table('users')->increment('votes', 1, ['name' => 'John']);
```

Query Builder

O construtor de consultas/query-builder de banco de dados do Laravel fornece uma interface conveniente e fluente para criar e executar consultas a banco de dados. Ele pode ser usado para executar a maioria das operações de banco de dados em sua aplicação e funciona perfeitamente com todos os SGBDs suportados pelo Laravel.

O construtor de consultas Laravel usa a ligação de parâmetro PDO para proteger seu aplicativo contra ataques de injeção de SQL. Não há necessidade de limpar ou sanitizar as strings passadas ao construtor de consultas como ligações de consulta.

O PDO não oferece suporte a nomes de colunas de associação. Portanto, você nunca deve permitir que a entrada do usuário dite os nomes das colunas referenciados por suas consultas, incluindo colunas "ordenar por".

Recebendo todos os registros de uma tabela num controller

```
<?php
namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;

class UserController extends Controller
{
    public function index()
    {
        $users = DB::table('users')->get();
        return view('user.index', ['users' => $users]);
    }
}
```

```
use Illuminate\Support\Facades\DB;

$users = DB::table('users')->get();

foreach ($users as $user) {
    echo $user->name;
}
```

Recebendo um único registro

```
$user = DB::table('users')->where('name', 'John')->first();
return $user->email;
```

Recebendo um registro pelo seu id

```
$user = DB::table('users')->find(3);
```

Recebendo uma lista de valores de um único campo:

```
use Illuminate\Support\Facades\DB;

$titles = DB::table('users')->pluck('title');

foreach ($titles as $title) {
    echo $title;
}
```

Você pode especificar o campo que a coleção resultante deve usar como sua chave fornecendo um segundo argumento para o método pluck:

```
$titles = DB::table('users')->pluck('title', 'name');
```



```
foreach ($titles as $name => $title) {
    echo $title;
}
```

Resultados de fragmentação/Chunking

Se você precisar trabalhar com milhares de registros de banco de dados, considere usar o método `chunk` fornecido pela classe `facade` do banco de dados. Esse método recupera um pequeno bloco de resultados de cada vez e alimenta cada parte em um encerramento para processamento. Por exemplo, vamos recuperar toda a tabela de usuários em blocos de 100 registros por vez:

```
use Illuminate\Support\Facades\DB;

DB::table('users')->orderBy('id')->chunk(100, function ($users) {
    foreach ($users as $user) {
        //
    }
});
```

Se você estiver atualizando os registros do banco de dados enquanto agrupa/chunking os resultados, os resultados do bloco podem mudar de maneira inesperada. Se você planeja atualizar os registros recuperados durante o agrupamento, é sempre melhor usar o método `chunkById`. Este método paginará automaticamente os resultados com base na chave primária do registro:

```
DB::table('users')->where('active', false)
->chunkById(100, function ($users) {
    foreach ($users as $user) {
        DB::table('users')
            ->where('id', $user->id)
            ->update(['active' => true]);
    }
});
```

Como obter registros aleatórios em Laravel

Registros aleatórios do banco de dados em laravel usando o método `inRandomOrder()`.

Se você estiver construindo um aplicativo de blog no framework laravel e deseja obter postagens/dados aleatórios do banco de dados em laravel. E deseja exibir esses dados aleatórios em qualquer lugar do seu blog laravel. Então, nesse momento, você precisa usar o método `inRandomOrder()` para obter registros aleatórios em laravel.

Este artigo irá guiá-lo sobre como usar obter registros aleatórios do banco de dados em laravel usando o método `inRandomOrder()`. Assim como obter registros aleatórios da coleção em laravel.

Observe que o método Laravel `inRandomOrder` pode ser usado para classificar os resultados da consulta aleatoriamente. Por exemplo, se você deseja obter postagens aleatórias do banco de dados em laravel.

Portanto, você também pode usar os seguintes métodos abaixo.

Laravel recupera registros aleatórios do banco de dados com o Eloquent

O exemplo a seguir irá buscar os 5 posts aleatórios da tabela DB em laravel:

```
public function index()
{
    $data = DB::table('posts')
        ->inRandomOrder()
        ->limit(5)
        ->get();
}
```

<https://www.tutsmake.com/how-to-get-random-records-in-laravel/>

Popular select com registros de outra tabela

Tenho um campo em um form, que se relaciona com outra tabela.

Quero preencher este campo do tipo select com todos os registros da tabela relacionada, apenas os campos id e descricao:

No meu caso tenho um CRUD criado pelo crud-generator

Mudar o método create no controller

```
public function create()
{
    $produtos = Produto::all(['id', 'descricao']);
    return view('compras.create', compact('produtos'));
    // https://stackoverflow.com/questions/29508297/laravel-5-how-to-populate-select-box-from-database-with-id-value-and-name-value
}
```

Mudar a view create, no form

```
<div class="form-group {{ $errors->has('produto_id') ? 'has-error' : '' }}">
    <label for="produto_id" class="control-label">{{ 'Produto Id' }}</label>
    <select class="form-select mb-3" aria-label=".form-select-lg example" name="produto_id">
        @foreach($produtos as $produto)
            <option value="{{ $produto->id }}"> {{ $produto->descricao }}</option> <!--&#x29E9; -->
        @endforeach
    </select>
    {!! $errors->first('produto_id', '<p class="help-block">:message</p>') !!}
</div>
```

Assim no campo produto-id de compras aparece um select com todos os registros de produtos

Tinker

Criar registros faker usando o factory

php artisan tinker

User::factory()->count(20)->create()

Para criar para a tabela clientes, copie o

UserFactory.php

ClientFactory.php

E adapte assim:

```
<?php
namespace Database\Factories;

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

class ClientFactory extends Factory
{
    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'email' => $this->faker->unique()->safeEmail(),
        ];
    }
}
```

Mudar o nome de um user

```
$user = User::where('email', 'trycia.koelpin@example.org')->first();
$user->name = 'Ribamar FS';
$user->save();
```

Em um único comando

```
User::where('email', 'trycia.koelpin@example.org')->first()->update(['name' => 'Ribamar FS']);
```

Pegando a versão do Laravel

```
app()->version()
```

<https://beyondco.de/blog/the-ultimate-guide-to-php-artisan-tinker>

```
$user = new User
$user->name = "John Doe"
$user->email = "john.doe@foo.bar"
$user->password = 'password'
$user->save()
```

Retornar todos os users

```
User::all()
```

Adicionar dois users

```
$user1 = new User
$user2 = new User
```

```
$user1->name = "John Doe2"
$user1->email = "john.doe2@foo.bar"
$user1->password = 'password'
```

```
$user2->name = "John Doe3"
$user2->email = "john.doe3@foo.bar"
$user2->password = 'password'
```

```
$user1->save()
$user2->save()
```

User 1

```
$user1 = User::find(1)
```

<https://medium.com/@leandroembu/testando-relacionamentos-com-o-laravel-tinker-cf1fe028608f>

<https://www.javatpoint.com/laravel-tinker>

<https://www.educba.com/laravel-tinker/>

<https://therichpost.com/php-artisan-tinker-database-commands-which-makes-life-easy/>

<https://www.tech-prastish.com/blog/how-to-use-laravel-tinker/>

Trabalhando com datas

Data atual, semana, mês, ano

Current date, current week, current, month, current year

php artisan tinker

Inserir um registro para testes

```
$user = new User()
$user->name = 'Ribamar FS'
$user->email = 'ribafs@gmail.com'
$user->password = 'abir'
$user->save()
```

Retornar name com data created_at de hoje

```
use Carbon\Carbon;
User::whereDate('created_at', Carbon::today())->get(['name','created_at']);
```

Retorno

```
=> Illuminate\Database\Eloquent\Collection {#4437
  all: [
    App\Models\User {#4188
      name: "ribafs",
      created_at: "2022-05-01 17:19:25",
    },
  ],
}
```

Receber semana atual

```
$current_week = User::whereBetween('created_at', [Carbon::now()->startOfWeek(),
Carbon::now()->endOfWeek()])->get();
```

Mês atual

```
User::whereMonth('created_at', date('m'))->whereYear('created_at', date('Y'))-
>get(['name','created_at']);
```

Obter dados mensais do ano atual

```
User::select(DB::raw("(COUNT(*) as count"),DB::raw("MONTHNAME(created_at) as monthname"))->whereYear('created_at', date('Y'))->groupBy('monthname')->get();
```

Obter dia atual na semana atual

```
User::select(DB::raw("(COUNT(*) as count"),DB::raw("DAYNAME(created_at) as dayname"))->whereBetween('created_at', [Carbon::now()->startOfWeek(), Carbon::now()->endOfWeek()])->whereYear('created_at', date('Y'))->groupBy('dayname')->get();
```

Ano atual

```
User::select(DB::raw("(COUNT(*) as count"),DB::raw("YEAR(created_at) as year"))->groupBy('year')->get();
```

<https://www.tutsmake.com/laravel-get-current-date-week-month-wise-year-data/>

6 – MVC

MVC – É um padrão arquitetura de software para separar o código em camadas, a model, a view e o controller.

Model

Model é a camada que cuida dos dados e se comunica diretamente com o banco de dados. No laravel 9 a os models ficam na pasta app/Models

Abaixo um exemplo simples de model

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    protected $table = 'posts';

    protected $primaryKey = 'id';

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

Relacionamentos

Relacionamentos entre tabelas no Laravel 8

Um para um

Exemplos de relacionamentos tipo um para um

- um user pode ter somente uma única conta bancária
- uma conta pode ter somente um dono/user
- marido somente pode ter uma esposa
- esposa somente pode pertencer a um marido

Exemplo

O relacionamento um para um é feito pela PK de ambas as tabelas

```
CREATE TABLE users(
    id INT NOT NULL AUTO_INCREMENT,
    email VARCHAR(100) NOT NULL,
    password VARCHAR(100) NOT NULL,
    PRIMARY KEY(id)
);
```



```
CREATE TABLE accounts(
  id INT NOT NULL AUTO_INCREMENT,
  role VARCHAR(50) NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(id) REFERENCES users(id)
);
```

Um para muitos

- users e orders
- categories e subcategories
- Items and Categories (many items from one category)

```
CREATE TABLE users(
  id INT NOT NULL AUTO_INCREMENT,
  email VARCHAR(100) NOT NULL,
  password VARCHAR(100) NOT NULL,
  PRIMARY KEY(id)
);
```

```
CREATE TABLE orders(
  id INT NOT NULL AUTO_INCREMENT,
  user_id INT NOT NULL,
  description VARCHAR(50) NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(user_id) REFERENCES users(id)
);
```

O relacionamento é formado pela PK da tabela principal com a FK da secundária

Muitos para muitos

- authors and books
- orders e itens

Um relacionamento muitos para muitos resolve-se adicionando uma terceira tabela entre as duas com relacionamento: M -> 1 -- 1 <- M

Relacionamento entre authors e books

authors authors_books books

id	name	author_id	book_id	id	name
1	John	1	11	11	Justice is Done
2	Peter	2	12	12	E o vento levou

Geralmente a tabela pivot (authors_books) contém somente dois campos, que são os ids de cada uma das duas tabelas, mas também podemos inserir um campo id na mesma

Criar duas migrations, para produtos e para compras, sendo compras relacionadas com produtos pelo produto-id

php artisan make:migration create_produtos_table

databases/migrations/xxxprodutosxxx

```
Schema::create('produtos', function (Blueprint $table) {
    $table->id();
    $table->string('descricao', 50)->nullable();
    $table->integer('estoque_minimo')->nullable();
    $table->integer('estoque_maximo')->nullable();
    $table->timestamps();
});
```

php artisan make:migration create_compras_table

```
Schema::create('compras', function (Blueprint $table) {
    $table->id();
    $table->foreignId('produto_id')->constrained('produtos');
    $table->integer('quantidade')->notNullable();
    $table->decimal('valor_unitario', 9,2);
    $table->date('data')->notNullable();
    $table->timestamps();
});
```

Coments e posts

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string("name");
        $table->timestamps();
    });
}
```

```

public function up()
{
    Schema::create('comments', function (Blueprint $table) {
        $table->id();
        $table->foreignId('post_id')->constrained('posts');
        $table->string("comment");
        $table->timestamps();
    });
}

```

app/Models/Post.php

```

class Post extends Model
{
    use HasFactory;

    public function comments()
    {
        return $this->hasMany(Comment::class);
    }
}

```

app/Models/Comment.php

```

class Comment extends Model
{
    use HasFactory;

    public function post()
    {
        return $this->belongsTo(Post::class);
    }
}

```

Receber registros do model num controller

```

class PostController extends Controller
{
    public function index(Request $request)
    {
        $comments = Post::find(1)->comments;
        dd($comments);
    }
}

use Illuminate\Http\Request;

```

```
use App\Models\Comment;

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = Comment::find(1)->post;

        dd($post);
    }
}
```

Criar registros

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = Post::find(1);

        $comment = new Comment;
        $comment->comment = "Hi ItSolutionStuff.com";

        $post = $post->comments()->save($comment);
    }
}
```

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = Post::find(1);

        $comment1 = new Comment;
```

```
$comment1->comment = "Hi ItSolutionStuff.com Comment 1";
```

```
$comment2 = new Comment;
```

```
$comment2->comment = "Hi ItSolutionStuff.com Comment 2";
```

```
$post = $post->comments()->saveMany([$comment1, $comment2]);
```

```
}
```

```
}
```

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```
use App\Models\Post;
```

```
use App\Models\Comment;
```

```
class PostController extends Controller
```

```
{
```

```
    public function index(Request $request)
```

```
{
```

```
        $comment = Comment::find(1);
```

```
        $post = Post::find(2);
```

```
        $comment->post()->associate($post)->save();
```

```
    }
```

```
}
```

<https://www.itsolutionstuff.com/post/laravel-9-one-to-many-eloquent-relationship-tutorialexample.html>

<https://www.itsolutionstuff.com/post/laravel-9-one-to-one-relationship-exampleexample.html>

<https://onlinewebtutorblog.com/laravel-9-has-many-through-eloquent-relationship-tutorial/>

Criação de um pequeno aplicativo com Laravel 9

Com posts e comments usando relacionamentos um para vários entre eles.

```
laravel new relac
cd relac
```

Criar banco e conf .env

Criar os CRUDs Posts e Comments usando o
crud-generator-laravel-br

Criar migration para posts

```
php artisan make:migration create_posts_table
```

Observe a convenção para criar uma migration para uma tabela
create_nometabela_table

Criará a migration em
database/migrations

Edite a migration criada e atualize:

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->string('name', 50);
    $table->timestamps();
});
```

Criar a migration para comments

```
php artisan make:migration create_comments_table
```

Edite e atualize

```
Schema::create('comments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('post_id')->constrained('posts');
    $table->text("comment");
    $table->timestamps();
});
```

Criar o model Post

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

Gera
app/Models/Post.php

Edite e atualize:

```
...
class Post extends Model
{
    use HasFactory;

    public function comments()
    {
        return $this->hasMany(Comment::class);
    }
}
```

Observe o método adicionado, que no model Post, retorna `$this->hasMany(Comment::class)`, o que nos informa que cada Post pode retornar vários comentários.

Criar o model Comment

php artisan make:model Comment

Gera
app/Models/Comment.php

Edite e atualize:

```
...
class Comment extends Model
{
    use HasFactory;

    public function post()
    {
        return $this->belongsTo(Post::class);
    }
}
```

Aqui o método `post()` nos informa que cada comment pertence a um post. Observe que em Comment método é no singular: `post()` e em Post o método é no plural: `comments`. Cada post pode ter vários comments e cada comment pertence a somente um único post.

Receber registros através de controllers

Criar o controller

php artisan make:controller PostController

Edite e atualize:

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $comments = Post::find(1)->comments;
        dd($comments);
    }
}
```

Observe que, se desejo receber registros da tabela posts, eu uso o model Post no controller.

Para testar os controllers vamos inserir manualmente um post e um comment nas respectivas tabelas:

Post 1

Comment 1 do Post 1

Para ambos em Timestamp escolha NOW

Atualizar as rotas assim:

```
Route::get('post', [App\Http\Controllers\PostController::class, 'index']);
```

Veja que o controller não chama nenhuma view. Ele usa o método de debug dd(), que mostra uma variável e encerra o processamento.

php artisan migrate

Adicionar um registro manualmente para a tabela posts e um em comments relacionado com o post com ID 1

php artisan serve

http://127.0.0.1:8000/post

Retorna no navegador:

```
^ Illuminate\Database\Eloquent\Collection {#1208 ▼
  #items: array:1 [ ]
  #escapeWhenCastingToString: false
}
```

Então clique na seta em items

Ele fica assim:

```
^ Illuminate\Database\Eloquent\Collection {#1208 ▼
  #items: array:1 [ ▼
    0 => App\Models\Comment {#958 }
  ]
  #escapeWhenCastingToString: false
}
```

Clique na seta em Comment

```
^ Illuminate\Database\Eloquent\Collection {#1208 ▼
  #items: array:1 [ ▼
    0 => App\Models\Comment {#958 ▼
      #connection: "mysql"
      #table: "comments"
      #primaryKey: "id"
      #keyType: "int"
      +incrementing: true
      #with: []
      #withCount: []
      +preventsLazyLoading: false
      #perPage: 15
      +exists: true
      +wasRecentlyCreated: false
      #escapeWhenCastingToString: false
      #attributes: array:5 [ ]
      #original: array:5 [ ]
      ...
    ]
  ]
  #escapeWhenCastingToString: false
}
```

Agora clique na seta em attributes ou em original:

```
#attributes: array:5 [ ▼
  "id" => 1
  "post_id" => 1
  "comment" => "Apenas um comentário."
  "created_at" => null
  "updated_at" => null
]
```

Se eu pretendo receber registros de comments, preciso usar o model Comment:

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = Comment::find(1)->post;
        dd($post);
    }
}
```

Veja que ele está requisitando os posts pelos comentários.

Criar registros

Use os models Post e Comment

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = Post::find(1); // Cria uma referência ao post com ID 1

        $comment = new Comment;
        $comment->comment = "Segundo comentário do post 1";

        $post = $post->comments()->save($comment); // Salva o comentário acima no post 1
        print 'Registro inserido. Confira no banco';
    }
}
```

Para inserir o comentário acima no banco, chame novamente pelo navegador:
<http://127.0.0.1:8000/post>

Criar o registro post com id 2

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = new Post;
        $post->name = "Post com id 2";

        $post = $post->save(); // Salva o post 2
        print 'Registro inserido. Confira no banco';
    }
}
```

Chame pelo navegador para executar

Salvando muitos registros de uma só vez

```
<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $post = Post::find(1);

        $comment3 = new Comment;
        $comment3->comment = "Terceito comentário do post 1";

        $comment4 = new Comment;
        $comment4->comment = "Quarto comentário do post 1";

        $post = $post->comments()->saveMany([$comment3, $comment4]);
        print 'Registros inseridos. Confira no banco';
    }
}
```

```

<?php
namespace App\Http\Controllers;

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

class PostController extends Controller
{
    public function index(Request $request)
    {
        $comment = Comment::find(3); // Cria em $comment referência para o comment com id 1
        $post = Post::find(2); // Cria em $post referência para o $post 2

        $comment->post()->associate($post)->save();
        print 'Muda a associação. O comment 3 era relacionado com o post 1. Assim ele será
relacionado com o post 2';
    }
}

```

Chame pelo navegador e confira no banco, tabela comments.

Crédito:

<https://www.itsolutionstuff.com/post/laravel-9-one-to-many-eloquent-relationship-tutorialexample.html>

Outros:

<https://www.itsolutionstuff.com/post/laravel-9-one-to-one-relationship-exampleexample.html>
<https://onlinewebtutorblog.com/laravel-9-has-many-through-eloquent-relationship-tutorial/>

Views

As views formam a camada do MVC que se encarrega de retornar as informações de volta para o usuário que as solicitou.

Obviamente, não é prático retornar strings em documentos HTML diretamente de suas rotas e controllers. Felizmente, as views fornecem uma maneira conveniente de colocar todo o nosso HTML em arquivos separados. As views separam a lógica do controller/aplicativo da lógica de apresentação e são armazenadas no diretório `resources/views`. Uma view simples pode ser algo assim:

```
<html>
  <body>
    <h1>Hello, {{ $name }}</h1>
  </body>
</html>
```

O template usado pelo laravel nas suas views é o Blade.

Blade é o mecanismo de modelagem simples, mas poderoso, incluído no Laravel. Ao contrário de alguns mecanismos de modelagem PHP, o Blade não restringe o uso de código PHP simples em seus modelos. Na verdade, todos os modelos do Blade são compilados em código PHP simples e armazenados em cache até serem modificados, o que significa que o Blade adiciona essencialmente zero sobrecarga ao seu aplicativo. Os arquivos de modelo blade usam a extensão de arquivo `.blade.php` e normalmente são armazenados no diretório `resources/views`.

As views de blade podem ser retornadas de rotas ou controladores usando o helper global de view.

Paginação

O paginador do Laravel é integrado ao Query Builder e ao Eloquent ORM e fornece paginação conveniente e fácil de usar de registros de banco de dados com configuração zero.

Por padrão, o HTML gerado pelo paginador é compatível com o framework Tailwind CSS; no entanto, o suporte à paginação do Bootstrap também está disponível.

Usando Bootstrap

Laravel inclui visualizações de paginação construídas usando Bootstrap CSS. Para usar essas visualizações em vez das visualizações padrão do Tailwind, você pode chamar os métodos `useBootstrapFour` ou `useBootstrapFive` do paginador dentro do método de inicialização de sua classe `App\Providers\AppServiceProvider`:

```
use Illuminate\Pagination\Paginator;

/**
 * Bootstrap any application services.
```

```

*
* @return void
*/
public function boot()
{
    Paginator::useBootstrapFive();
    Paginator::useBootstrapFour();
}

```

Validação de formulários

composer create-project --prefer-dist laravel/laravel formval

cd formval

Criar bd e conf .env

php artisan make:model Employee -m

Editar a migration e deixar assim:

```

Schema::create('employees', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email');
    $table->string('contact_no');
    $table->string('age');
    $table->timestamps();
});

```

php artisan migrate

Adicionar a rota

```
use App\Http\Controllers\FormController;
```

```
Route::get('form', [FormController::class, 'index']);
Route::post('store-form', [FormController::class, 'store']);
```

php artisan make:controller FormController

```

<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

```

```

use App\Models\Employee;

class FormController extends Controller
{
    public function index()
    {
        return view('form');
    }

    public function store(Request $request)
    {
        $validatedData = $request->validate([
            'name' => 'required',
            'email' => 'required|unique:employees|max:255',
            'age' => 'required',
            'contact_no' => 'required|unique:employees|max:255',
        ]);

        $emp = new Employee;

        $emp->name = $request->name;
        $emp->email = $request->email;
        $emp->age = $request->age;
        $emp->contact_no = $request->contact_no;

        $emp->save();

        return redirect('form')->with('status', 'Form Data Has Been Inserted');
    }
}

```

Criar a view
resources/views/form.blade.php

```

<!DOCTYPE html>
<html>
<head>
<title>Laravel 9 Form Validation</title>
<meta name="csrf-token" content="{{ csrf_token() }}">
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-4">
@if(session('status'))

```

```

<div class="alert alert-success">
{{ session('status') }}
</div>
@endif
<div class="card">
<div class="card-header text-center font-weight-bold">
<h2>Laravel 9 Form Validation</h2>
</div>
<div class="card-body">
<form name="employee" id="employee" method="post" action="{{url('store-form')}}">
{{ csrf_field() }}
<div class="form-group">
<label for="exampleInputEmail1">Name</label>
<input type="text" id="name" name="name" class="@error('name') is-invalid @enderror form-control">
@error('name')
<div class="alert alert-danger mt-1 mb-1">{{ $message }}</div>
@enderror
</div>
<div class="form-group">
<label for="exampleInputEmail1">Email</label>
<input type="email" id="email" name="email" class="@error('email') is-invalid @enderror form-control">
@error('email')
<div class="alert alert-danger mt-1 mb-1">{{ $message }}</div>
@enderror
</div>
<div class="form-group">
<label for="exampleInputEmail1">Age</label>
<input type="number" id="age" name="age" class="@error('age') is-invalid @enderror form-control">
@error('age')
<div class="alert alert-danger mt-1 mb-1">{{ $message }}</div>
@enderror
</div>
<div class="form-group">
<label for="exampleInputEmail1">Contact No</label>
<input type="number" id="contact_no" name="contact_no" class="@error('contact_no') is-invalid @enderror form-control">
@error('contact_no')
<div class="alert alert-danger mt-1 mb-1">{{ $message }}</div>
@enderror
</div>
<button type="submit" class="btn btn-primary">Submit</button>
</form>
</div>
</div>
</div>

```



```
</body>
</html>
```

Testar

php artisan serve

http://127.0.0.1:8000/form

<https://www.tutsmake.com/laravel-9-form-validation-tutorial-with-example/>

Customizar as mensagens de erro

```
<!DOCTYPE html>
<html>
<head>
  <title>Laravel 9 Custom Validation Error Message Example Tutorial -Tutsmake.com</title>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <h1>Laravel 9 Custom Validation Error Message Example</h1>

    @if(Session::has('success'))
    <div class="alert alert-success">
      {{ Session::get('success') }}
      @php
        Session::forget('success');
      @endphp
    </div>
    @endif

    <form method="POST" action="{{ route('store') }}">

      @csrf

      <div class="form-group">
        <label>Name:</label>
        <input type="text" name="name" class="form-control" placeholder="Name">
        @if ($errors->has('name'))
          <span class="text-danger">{{ $errors->first('name') }}</span>
```

```

        @endif
    </div>

    <div class="form-group">
        <label>Password: </label>
        <input type="password" name="password" class="form-control"
placeholder="Password">
        @if ($errors->has('password'))
            <span class="text-danger">{{ $errors->first('password') }}</span>
        @endif
    </div>

    <div class="form-group">
        <strong>Email: </strong>
        <input type="text" name="email" class="form-control" placeholder="Email">
        @if ($errors->has('email'))
            <span class="text-danger">{{ $errors->first('email') }}</span>
        @endif
    </div>

    <div class="form-group">
        <button class="btn btn-success btn-submit">Submit</button>
    </div>
</form>
</div>
</body>
</html>

```

<https://www.tutsmake.com/laravel-9-validation-custom-error-messages-example/>

Controller

Em vez de definir toda a sua lógica de manipulação de requisições como encerramentos em seus arquivos de rota, você pode organizar esse comportamento usando classes "controladoras/controllers". Os controladores podem agrupar a lógica de manipulação de requisições relacionadas em uma única classe. Por exemplo, uma classe UserController pode lidar com todas as requisições recebidas relacionadas a usuários, incluindo mostrar, criar, atualizar e excluir usuários. Por padrão, os controladores são armazenados na pasta app/Http/Controllers.

Paginação automática

Se você já lutou com paginação em seus aplicativos, você entenderá o valor de ter sua paginação ordenada por um framework embutida.

Laravel resolve o problema da paginação através da construção de paginação automática que sai da caixa. Esta característica é uma das mais reconhecidas e elimina o trabalho envolvido na resolução do mistério da paginação por si mesmo.

Rotas

As rotas mais básicas do Laravel aceitam um URI e um closure/encerramento, fornecendo um método muito simples e expressivo de definir rotas e behavior/comportamento sem arquivos de configuração de roteamento complicados:

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Todas as rotas do Laravel são definidas em seus arquivos de rotas, que estão localizados no diretório routes. Esses arquivos são carregados automaticamente pelo App\Providers\RouteServiceProvider do seu aplicativo. O arquivo routes/web.php define as rotas que são para sua interface web. Essas rotas são atribuídas ao grupo de middleware da Web, que fornece recursos como estado de sessão e proteção CSRF. As rotas em routes/api.php são stateless e são atribuídas ao grupo de middleware api.

Para a maioria dos aplicativos, você começará definindo rotas em seu arquivo routes/web.php. As rotas definidas em routes/web.php podem ser acessadas digitando a URL da rota definida em seu navegador. Por exemplo, você pode acessar a seguinte rota navegando para <http://example.com/user> em seu navegador:

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Veja abaixo um arquivo re rotas default do laravel 9, logo que é instalado, sem comentários:

```
<?php
use Illuminate\Support\Facades\Route;

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

Alguns exemplos

Rotas individuais para cada ação do CRUD

```
use Illuminate\Support\Facades\Route;

Route::get('/', [App\Http\Controllers\ProdutoController::class, 'index']);

use Illuminate\Support\Facades\Route;

Route::get('/', [App\Http\Controllers\ProdutoController::class, 'create']);
```

Rota resource para todas as ações de um CRUD

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\ProdutoController;

Route::resource('produtos', ProdutoController::class);
```

Exemplo com várias e usando um middleware

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\RoleController;
use App\Http\Controllers\UserController;

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

Auth::routes();

//Nomeada
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');
```

```
//Grupo com middleware
Route::group(['middleware' => ['auth']], function() {
    Route::resource('roles', RoleController::class);
    Route::resource('users', UserController::class);
        Route::resource('posts', 'App\Http\Controllers\PostController');
        Route::resource('comments', 'App\Http\Controllers\CommentController');
});
```

7 – Segurança

O Laravel já vem com alguns bons recursos que reforçam sua segurança: autenticação, autorização, gates, CSRF, proteção das rotas, lembrar e reset de senhas, etc.

É essencial examinar as medidas de segurança de qualquer aplicação web que você esteja considerando usar, pois a falta de diligência pode resultar em perda de fundos ou mesmo seqüestro de seu site ou produto.

Laravel vem com muitas medidas de segurança em vigor devido à sua adesão aos princípios de segurança da OWASP. Desde o pedido de falsificação cruzada (CSRF) até a injeção SQL, a Laravel tem uma solução integrada para tudo isso.

A classe Hash do Laravel fornece hash usando Bcrypt seguro:
`$password = Hash::make('secret');`

Autenticação

Muitos aplicativos web fornecem uma maneira de seus usuários se autenticarem com o aplicativo e fazerem o "login". A implementação desse recurso em aplicativos web pode ser um empreendimento complexo e potencialmente arriscado. Por esse motivo, o Laravel se esforça para fornecer as ferramentas necessárias para implementar a autenticação de maneira rápida, segura e fácil.

Em sua essência, as instalações de autenticação do Laravel são compostas por "gates/guardas" e "policies". Os guardas definem como os usuários são autenticados para cada solicitação. Por exemplo, o Laravel vem com um protetor de sessão que mantém o estado usando armazenamento de sessão e cookies.

Os policies definem como os usuários são recuperados de seu armazenamento persistente. O Laravel vem com suporte para recuperar usuários usando o Eloquent e o QueryBuilder/construtor de consultas de banco de dados. No entanto, você é livre para definir provedores adicionais conforme necessário para seu aplicativo.

O arquivo de configuração de autenticação do seu aplicativo está localizado em `config/auth.php`. Este arquivo contém várias opções bem documentadas para ajustar o comportamento dos serviços de autenticação do Laravel.

Referências

<https://codeanddeploy.com/blog/laravel/laravel-8-authentication-login-and-registration-with-username-or-email>

<https://www.positronx.io/create-multi-auth-authentication-in-laravel/>

<https://www.nicesnippets.com/blog/laravel-9-multiple-authentication-using-middleware>

<https://appdividend.com/2022/01/23/laravel-middleware/>

<https://www.mywebtuts.com/blog/laravel-9-multi-authauthentication-tutorial>

<https://codingdriver.com/laravel-multi-auth-authentication-tutorial-example.html>

<https://www.positronx.io/create-multi-auth-authentication-in-laravel/>

Autorização

A camada de autenticação verifica se o usuário está registrado e se sim dá acesso a ele. Então vem a camada de autorização e verifica quais as permissões do usuário para garantir um acesso somente onde o usuário for permitido.

Além de fornecer serviços de autenticação integrados, o Laravel também fornece uma maneira simples de autorizar ações do usuário em relação a um determinado recurso. Por exemplo, mesmo que um usuário seja autenticado, ele pode não estar autorizado a atualizar ou excluir determinados modelos do Eloquent ou registros de banco de dados gerenciados pelo seu aplicativo. Os recursos de autorização do Laravel fornecem uma maneira fácil e organizada de gerenciar esses tipos de verificações de autorização.

O Laravel fornece duas formas principais de autorizar ações: gates/portas e policies/políticas. Pense em portas e políticas como rotas e controladores. Os gates fornecem uma abordagem simples e baseada em fechamento para autorização, enquanto as políticas, como controladores, agrupam a lógica em torno de um model ou recurso específico.

Você não precisa escolher entre usar exclusivamente gates ou usar exclusivamente policies ao criar um aplicativo. A maioria dos aplicativos provavelmente conterá alguma mistura de ambos, e isso está bem! Os gates são mais aplicáveis a ações que não estão relacionadas a nenhum model ou recurso, como exibir um painel de administrador. Em contraste, as policies devem ser usadas quando você deseja autorizar uma ação para um model ou recurso específico.

Referências

<https://resources.infosecinstitute.com/topic/how-to-use-authorization-in-laravel-gates-policies-roles-and-permissions/>

<https://www.avyatech.com/role-based-authentication-authorization-in-laravel/>

<https://tutspack.com/laravel-authorization-gate-and-policy-with-example/>

Roles and Permissions

Roles são grupos ou funções para os usuários de um aplicativo. As roles definem permissões de usuários aos recursos do aplicativo. Em nosso aplicativo temos algumas rotas pré-definidas, que são Super (para usuários que podem acessar todos os recursos do aplicativo), Admin (para usuários administrativos, que podem acessar tudo de Users, Roles e Permissions), Manager (pode tudo nos CRUDs de negócio, Posts e Comments) e finalmente a role User, para usuários que somente acessam a view index de Posts.

ACL - Access Control List

Lista de Controle de Acesso

A solução de ACL mais popular para Laravel é a do pacote laravel-permission do Spatie

<https://spatie.be>

<https://github.com/spatie/laravel-permission>

<https://spatie.be/docs/laravel-permission/v5/introduction>

Veja um resumo da documentação caso queira um maior controle do acesso ao aplicativo

Este pacote permite gerenciar permissões e roles/funções de usuários em um banco de dados.

Roles (papeis/funções) e permissões são muito importantes para aplicações/sites web. User roles são uma forma útil de permitir ou controlar quem pode acessar um site/aplicativo,

O pacote Spatie é o melhor para roles e permissions, permitindo que gerencie permissions e roles no banco de dados.

Spatie oferece forma de atribuir roles para users, como atribuir diversas permissões para users e como atribuir permissões para roles. Com roles e permissions podemos criar vários tipos de usuários com diferentes roles e permissões.

Uma permissão é o direito de acessar uma página ou apenas uma região da página. Uma role é um grupo de permissões. Roles podem ser atribuídas para qualquer usuário ou grupo de usuários. Um usuário ou grupo pode ter mais que uma role. Uma role não contém outra role, somente permissões.

- Checar numa view se um user tem permissão de criar artigo:

```
@can('article-create')  
// pode  
@endcan
```

- Criar role e permission

```
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

$role = Role::create(['name' => 'Admin']);
$permission = Permission::create(['name' => 'article-edit']);
```

Uma permissão pode ser atribuída a uma função usando um destes métodos:

```
$role->givePermissionTo($permission);
$permission->assignRole($role);
```

Várias permissões podem ser sincronizadas com uma função usando um destes métodos:

```
$role->syncPermissions($permissions);
$permission->syncRoles($roles);
```

Uma permissão pode ser removida de uma função usando um destes métodos:

```
$role->revokePermissionTo($permission);
$permission->removeRole($role);
```

O trait HasRoles adiciona relacionamentos Eloquent aos seus modelos, que podem ser acessados diretamente ou usados como consulta base:

```
// get a list of all permissions directly assigned to the user
$permissionNames = $user->getPermissionNames(); // collection of name strings
$permissions = $user->permissions; // collection of permission objects

// get all permissions for the user, either directly, or from roles, or from both
$permissions = $user->getDirectPermissions();
$permissions = $user->getPermissionsViaRoles();
$permissions = $user->getAllPermissions();

// get the names of the user's roles
$roles = $user->getRoleNames(); // Returns a collection
```

O trait HasRoles também adiciona um escopo de função aos seus modelos para definir o escopo da consulta para determinadas funções ou permissões:

```
$users = User::role('writer')->get(); // Returns only users with the role 'writer'
```

O escopo da função pode aceitar uma string, um objeto `\Spatie\Permission\Models\Role` ou um objeto `\Illuminate\Support\Collection`.

A mesma característica também adiciona um escopo para obter apenas usuários que tenham uma determinada permissão.

```
$users = User::permission('edit articles')->get(); // Returns only users with the permission 'edit articles' (inherited or directly)
```

Eloquent

Como os models Role e Permission são estendidos dos models Eloquent, as chamadas básicas do Eloquent também podem ser usadas:

```
$all_users_with_all_their_roles = User::with('roles')->get();
$all_users_with_all_direct_permissions = User::with('permissions')->get();
$all_roles_in_database = Role::all()->pluck('name');
$users_without_any_roles = User::doesntHave('roles')->get();
$all_roles_except_a_and_b = Role::whereNotIn('name', ['role A', 'role B'])->get();
```

Uma permissão pode ser dada a qualquer usuário:

```
$user->givePermissionTo('edit articles');
```

Você também pode conceder várias permissões de uma só vez

```
$user->givePermissionTo('edit articles', 'delete articles');
```

Você também pode passar um array

```
$user->givePermissionTo(['edit articles', 'delete articles']);
```

Uma permissão pode ser revogada de um usuário:

```
$user->revokePermissionTo('edit articles');
```

Ou revogue e adicione novas permissões de uma só vez:

```
$user->syncPermissions(['edit articles', 'delete articles']);
```

Você pode checar se um user tem uma permission:

```
$user->hasPermissionTo('edit articles');
```

Ou você pode passar um inteiro representando o id da permission

```
$user->hasPermissionTo('1');
$user->hasPermissionTo(Permission::find(1)->id);
$user->hasPermissionTo($somePermission->id);
```

Você pode verificar se um usuário tem qualquer uma de uma matriz de permissões:

```
$user->hasAnyPermission(['edit articles', 'publish articles', 'unpublish articles']);
```

...ou se um usuário tiver todas as permissões de uma matriz:

```
$user->hasAllPermissions(['edit articles', 'publish articles', 'unpublish articles']);
```

Você também pode passar inteiros para pesquisa por id de permissão

```
$user->hasAnyPermission(['edit articles', 1, 5]);
```

As permissões salvas serão registradas com a classe Illuminate\Auth\Access\Gate para o protetor padrão. Então você pode verificar se um usuário tem permissão com o padrão do Laravel pode funcionar:

```
$user->can('edit articles');
```

Usando caracteres curinga

ALERTA: O * significa "ALL". Não significa "ANY".

```
Permission::create(['name'=>'posts.*']);
```

```
$user->givePermissionTo('posts.*');
```

// is the same as

```
Permission::create(['name'=>'posts']);
```

```
$user->givePermissionTo('posts');
```

// user can only do the actions create, update and view on both resources posts and users

```
$user->givePermissionTo('posts,users.create,update,view');
```

// user can do the actions create, update, view on any available resource

```
$user->givePermissionTo('*.create,update,view');
```

// user can do any action on posts with ids 1, 4 and 6

```
$user->givePermissionTo('posts.*.1,4,6');
```

A role can be assigned to any user:

```
$user->assignRole('writer');
```

// You can also assign multiple roles at once

```
$user->assignRole('writer', 'admin');
```

// or as an array

```
$user->assignRole(['writer', 'admin']);
```

A role can be removed from a user:

```
$user->removeRole('writer');
```

Roles can also be synced:

// All current roles will be removed from the user and replaced by the array given

```
$user->syncRoles(['writer', 'admin']);
```

You can determine if a user has a certain role:

```
$user->hasRole('writer');
```

// or at least one role from an array of roles:

```
$user->hasRole(['editor', 'moderator']);
```

You can also determine if a user has any of a given list of roles:

```
$user->hasAnyRole(['writer', 'reader']);  
// or  
$user->hasAnyRole('writer', 'reader');
```

You can also determine if a user has all of a given list of roles:

```
$user->hasAllRoles(Role::all());
```

You can also determine if a user has exactly all of a given list of roles:

```
$user->hasExactRoles(Role::all());
```

A permission can be given to a role:

```
$role->givePermissionTo('edit articles');
```

You can determine if a role has a certain permission:

```
$role->hasPermissionTo('edit articles');
```

A permission can be revoked from a role:

```
$role->revokePermissionTo('edit articles');
```

What Permissions Does A Role Have?

The permissions property on any given role returns a collection with all the related permission objects. This collection can respond to usual Eloquent Collection operations, such as count, sort, etc.

```
// get collection  
$role->permissions;
```

```
// return only the permission names:  
$role->permissions->pluck('name');
```

```
// count the number of permissions assigned to a role  
count($role->permissions);  
// or  
$role->permissions->count();
```

Assigning Direct Permissions To A User

Additionally, individual permissions can be assigned to the user too. For instance:

```
$role = Role::findByName('writer');  
$role->givePermissionTo('edit articles');
```

```
$user->assignRole('writer');
```

```
$user->givePermissionTo('delete articles');
```

You can check if the user has a Specific or All or Any of a set of permissions directly assigned:

```
// Check if the user has Direct permission
```

```
$user->hasDirectPermission('edit articles')
```

```
// Check if the user has All direct permissions
```

```
$user->hasAllDirectPermissions(['edit articles', 'delete articles']);
```

```
// Check if the user has Any permission directly
```

```
$user->hasAnyDirectPermission(['create articles', 'delete articles']);
```

You can examine all of these permissions:

```
// Direct permissions
```

```
$user->getDirectPermissions() // Or $user->permissions;
```

```
// Permissions inherited from the user's roles
```

```
$user->getPermissionsViaRoles();
```

```
// All permissions which apply on the user (inherited and direct)
```

```
$user->getAllPermissions();
```

Blade directives

#

Permissions

This package doesn't add any permission-specific Blade directives. Instead, use Laravel's native `@can` directive to check if a user has a certain permission.

```
@can('edit articles')
```

```
//
```

```
@endcan
```

or

```
@if(auth()->user()->can('edit articles') && $some_other_condition)
```

```
//
```

```
@endif
```

You can use `@can`, `@cannot`, `@canany`, and `@guest` to test for permission-related access.

#

Roles

As discussed in the Best Practices section of the docs, it is strongly recommended to always use permission directives, instead of role directives.

Additionally, if your reason for testing against Roles is for a Super-Admin, see the Defining A Super-Admin section of the docs.

If you actually need to test for Roles, this package offers some Blade directives to verify whether the currently logged in user has all or any of a given list of roles.

Optionally you can pass in the guard that the check will be performed on as a second argument.
#

Blade and Roles

Check for a specific role:

```
@role('writer')
    I am a writer!
@else
    I am not a writer...
@endrole
```

is the same as

```
@hasrole('writer')
    I am a writer!
@else
    I am not a writer...
@endhasrole
```

Check for any role in a list:

```
@hasanyrole($collectionOfRoles)
    I have one or more of these roles!
@else
    I have none of these roles...
@endhasanyrole
// or
@hasanyrole('writer|admin')
    I am either a writer or an admin or both!
@else
    I have none of these roles...
@endhasanyrole
```

Check for all roles:

```
@hasallroles($collectionOfRoles)
```

```

    I have all of these roles!
@else
    I do not have all of these roles...
@endhasallroles
// or
@hasallroles('writer|admin')
    I am both a writer and an admin!
@else
    I do not have all of these roles...
@endhasallroles

```

Alternatively, @unlessrole gives the reverse for checking a singular role, like this:

```

@unlessrole('does not have this role')
    I do not have the role
@else
    I do have the role
@endunlessrole

```

You can also determine if a user has exactly all of a given list of roles:

```

@hasexactroles('writer|admin')
    I am both a writer and an admin and nothing else!
@else
    I do not have all of these roles or have more other roles...
@endhasexactroles

```

Using permissions and roles with multiple guards

When creating new permissions and roles, if no guard is specified, then the first defined guard in auth.guards config array will be used.

```

// Create a manager role for users authenticating with the admin guard:
$role = Role::create(['guard_name' => 'admin', 'name' => 'manager']);

// Define a `publish articles` permission for the admin users belonging to the admin guard
$permission = Permission::create(['guard_name' => 'admin', 'name' => 'publish articles']);

// Define a *different* `publish articles` permission for the regular users belonging to the web guard
$permission = Permission::create(['guard_name' => 'web', 'name' => 'publish articles']);

```

Informações sobre o pacote

composer show spatie/laravel-permission

artisan commands**Criar role**

php artisan permission:create-role writer

Criar permissão

php artisan permission:create-permission "article-edit"

Criar role e permissão com guard

php artisan permission:create-role Admin web

php artisan permission:create-permission "article-edit" web

php artisan permission:show

php artisan permission:cache-reset

re-migrate and seed the database:

php artisan migrate:fresh --seed --seeder=PermissionsDemoSeeder

<https://spatie.be/docs/laravel-permission/v5/introduction>

Referências

<https://www.itsolutionstuff.com/post/laravel-9-user-roles-and-permissions-tutorialexample.html>

<https://websolutionstuff.com/post/laravel-9-user-role-and-permission>

<https://www.tutsmake.com/laravel-9-multi-user-authentication-example/>

<https://codingdriver.com/laravel-user-roles-and-permissions-tutorial-with-example.html>

<https://www.codecheef.org/article/laravel-9-roles-and-permissions-without-package-example>

<https://websolutionstuff.com/post/laravel-9-user-role-and-permission>

8 - Recursos usando Laravel

E-commerce usando Laravel

<https://bagisto.com/en/download/>

```
cd bagisto
```

```
composer install
```

Criar o banco e configurar o .env

```
php artisan bagisto:install
```

Alerta

Warning: Before going into production mode, we recommend you uninstall developer dependencies. To do that, run the command below:

```
composer install --no-dev
```

```
php artisan serve
```

Podemos acessar com

```
email:admin@example.com  
password:admin123
```

Ou registrar um novo user

<https://github.com/bagisto/bagisto>

9 – Dicas sobre Laravel

Saber a versão do laravel

```
php artisan --version
```

Via código

```
app()->version()
```

Numa view/blade

```
The current Laravel version is {{ app()->version() }}
```

Tornando a instalação mais amigável

No caso de transportar aplicativo para outro servidor, ou mesmo oferecer para download em repositórios, onde se baixa sem a pasta vendor e precisamos executar o comando `composer install` para que seja criado a pasta vendor. Idealmente é bom avisar o usuário que ele precisa fazer isso, que é melhor que ele ser recebido por várias mensagens de erro.

Editar o arquivo **artisan** que existe no raiz e adicionar isso:

```
#!/usr/bin/env php
<?php

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

/*
|-----
| Register The Auto Loader
|-----
|
| Composer provides a convenient, automatically generated class loader
| for our application. We just need to utilize it! We'll require it
| into the script here so that we do not have to worry about the
| loading of any of our classes manually. It's great to relax.
|
*/
// Adicionar estas linhas abaixo
if (!file_exists('vendor')) {
    print "\n\nPrecisa usar antes o comando:\ncomposer install\n\n";
    exit;
}
```

Também fazer isso no public/inde.php, caso tente acessar pelo navegador.

Então edite o public/index.php e adicione algo parecido:

```
<?php
use Illuminate\Contracts\Http\Kernel;
use Illuminate\Http\Request;

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

/*
|-----
| Check If The Application Is Under Maintenance
|-----
|
| If the application is in maintenance / demo mode via the "down" command
| we will load this file so that any pre-rendered content can be shown
| instead of starting the framework, which could cause an exception.
|
*/

if (!file_exists('vendor')) {
    print '<br><br><br>Precisa usar antes o comando:<br> composer
install<br><br><br>';
    exit;
}
```

Limpendo o cache de views

Geralmente quando efetuamos alguma alteração em uma view do laravel, não basta teclar F5 para atualizar a página no navegador, precisamos limpar o cache das views:

```
php artisan view:clear
```

Algumas funções

Como receber o path da pasta public do laravel?

```
echo public_path();
```

Da pasta storage:

```
storage_path();
```

Da pasta app:

```
app_path();
```

<https://www.itsolutionstuff.com/post/how-can-i-get-the-path-from-laravel-application-rootexample.html>

Ignorar requisições de plataforma

Ao encontrar o erro

Your Composer dependencies require a PHP version ">= 8.1.0".

Procurando na internet e após tentar várias coisas (sem sucesso) encontrei a seguinte solução, executar a linha abaixo na raiz do projeto:

```
composer install --ignore-platform-reqs
```

Um novo recurso no **Composer v2** permite que você ignore seletivamente os requisitos da plataforma.

```
composer install --ignore-platform-req=php
```

Composer agora tem a opção `--ignore-platform-reqs` (observe o `s` em `reqs`), mas ignora todos os requisitos da plataforma, incluindo versão do PHP, extensões (`ext-*`) e `composer-plugin-api`.

A nova opção `--ignore-platform-req` pode ser usado para definir requisitos específicos que o Composer pode ignorar.

10 – Ferramentas

Artisan

CLI Artisan

A CLI Artisan é outro aspecto vital de Laravel. Com ele, você pode criar ou modificar praticamente qualquer parte de Laravel a partir da linha de comando, sem ter que navegar por pastas e arquivos.

Com Artisan, você pode até interagir com sua base de dados diretamente de sua linha de comando usando Laravel Tinker – tudo isso sem instalar um cliente de base de dados.

Laravel Artisan Console Command Cheat Sheet

O comando da console Artisan é uma das partes mais úteis do Laravel Framework. Em laravel, uma linha de comando do console artisan é uma ferramenta para executar comandos do Laravel. Se você usar esses comandos, a velocidade de desenvolvimento será aprimorada ao fornecer scaffoldings prontos para uso e métodos obrigatórios. Você pode economizar muito tempo usando ele.

Check Your Laravel Application Version

```
php artisan --version OR -V
```

Laravel Composer dump-autoload:

Ele apenas regenera a lista de todas as classes que precisam ser incluídas no projeto (autoload_classmap.php), e é por isso que sua migrate está funcionando depois que você executa esse comando.

```
php artisan dump-autoload
```

Laravel lists artisan commands:

```
php artisan list
```

Laravel Application Serve:

O comando Laravel artisan serve é usado para executar o aplicativo usando o servidor de desenvolvimento PHP.

```
php artisan serve
```

Interact with the Laravel Application:

```
php artisan tinker
```

Publique os ativos de um pacote no diretório público:

```
php artisan asset:publish [--bench[="vendor/package"]] [--path[="..."]] [package]
```

Gere a chave do aplicativo no Laravel:

```
php artisan key:generate
```

Laravel Database migrations:

```
php artisan migrate
```

Create a new resourceful controller in Laravel:

```
php artisan controller:make [--bench="vendor/package"]
```

Generate database migration along with model in Laravel:

```
php artisan make:model User --migration OR -m
```

Generate database seeder in Laravel:

```
php artisan make:seeder [Table Seeder]
```

Create new middleware in Laravel:

```
Make:middleware [Middleware name]
```

Lists active routes in Laravel Application:

```
php artisan route:list
```

Create symbolic link in Laravel:

Create symbolic link from public/storage to storage/app/public

```
php artisan storage:link
```

Flush the Laravel Application cache:

```
php artisan cache:clear
```

Conclusion

In this post, you have learned uses of the laravel artisan commands.

<https://www.tutsmake.com/laravel-artisan-console-command-cheat-sheet-list/>

Alguns comandos

Generate a model and a FlightFactory class...

```
php artisan make:model Flight --factory
```

```
php artisan make:model Flight -f
```

Generate a model and a FlightSeeder class...

```
php artisan make:model Flight --seed  
php artisan make:model Flight -s
```

```
# Generate a model and a FlightController class...
```

```
php artisan make:model Flight --controller  
php artisan make:model Flight -c
```

```
# Generate a model, FlightController resource class, and form request classes...
```

```
php artisan make:model Flight --controller --resource --requests  
php artisan make:model Flight -crR
```

```
# Generate a model and a FlightPolicy class...
```

```
php artisan make:model Flight --policy
```

```
# Generate a model and a migration, factory, seeder, and controller...
```

```
php artisan make:model Flight -mfsc
```

```
# Shortcut to generate a model, migration, factory, seeder, policy, controller, and form requests...
```

```
php artisan make:model Flight --all
```

```
# Generate a pivot model...
```

```
php artisan make:model Member --pivot
```

```
Conectar ao banco atual
```

```
php artisan db mysql
```


Backup

<https://www.mywebtuts.com/blog/laravel-8-automatic-daily-database-backup-tutorial>
<https://www.tutsmake.com/laravel-9-daily-monthly-weekly-automatic-database-backup/>

DataTables no Laravel

<https://www.itsolutionstuff.com/post/laravel-9-yajra-datatables-example-tutorialexample.html>
<https://www.tutsmake.com/laravel-9-yajra-datatables-crud-example-tutorial/>
<https://onlinecode.org/laravel-9-ajax-crud-with-datatable/>
<https://laratutorials.com/laravel-9-ajax-crud-example-using-datatable-js/>

Aplicativo para Testes

É bem útil criar um aplicativo exclusivamente para testar pacotes desconhecidos.

Para a instalação de pacotes desconhecidos e também para efetuar diversos testes é bem útil criar um aplicativo somente para testes diversos.

Chega a um ponto em que está com alguns erros, então podemos removê-lo completamente e criar outro.

Para remover um pacote:

```
composer remove ribafs/crud-generator
```

11 – Deploy

A publicação de um aplicativo laravel num servidor requer alguns cuidados importantes.

Deploy do Laravel 9

O que precisamos fazer para colocar uma aplicação de forma correta em produção.

Server Requirements Laravel 9

O framework Laravel possui alguns requisitos de sistema. Você deve garantir que seu servidor web tenha a seguinte versão e extensões mínimas do PHP:

- PHP >= 8.0
- BCMath PHP Extension
- Ctype PHP Extension
- cURL PHP Extension
- DOM PHP Extension
- Fileinfo PHP Extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PCRE PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

No .env

```
APP_ENV=production  
APP_DEBUG=false
```

Em production - O Laravel tem uma serie de proteção quando ele está configurado assim.

O APP_DEBUG indica para o Laravel se ele deve mostrar erros no navegador. Exibir informações de erro é extremamente perigoso, um usuário mal intencionado pode obter diversas informações a partir dele.

Composer

Ao clonar a aplicação para nosso servidor em produção, a primeira coisa que precisamos fazer é executar o composer para baixar as dependências do projeto. Quando estamos em produção podemos passar dois parâmetros extras, veja como fica o comando:

```
composer install --optimize-autoload --no-dev
```

–optimize-autoload: gera uma versão das regras do PSR-4/PSR-0 em um arquivo PHP único, evitando que a linguagem tenha que olhar no sistema de arquivos. Esse arquivo de classmap pode ser facilmente cacheado pelo opcache tornando a obtenção dos caminhos muito mais rápido. Mais detalhes em autoloader-optimization

–no-dev: ignora as dependências exclusivas do ambiente de desenvolvimento

Remover pacotes para desenvolvimento

Antes de colocar em produção devemos remover os pacotes que instalamos para uso apenas no desktop. Lembrar de também remover suas configurações.
composer remove barryvdh/laravel-debugbar

Cacheando os arquivos de configuração

Acessar o arquivo .env toda hora é muito custoso, uma vez que ele é um arquivo de texto e não pode ser cacheado pelo opcache. Baseado nisso, o Laravel possui um comando que copia as configurações dele para um arquivo php único diminuindo assim o custo de acesso. Para isso temos o comando:
php artisan config:cache

Único detalhe que devemos ficar atentos quando executamos esse comando. Como as configurações do arquivo de configuração .env são carregados para o arquivo único, não é aconselhável usar o helper env() do Laravel que pega as configurações do arquivo .env já que ele pode não ser carregado.

Cacheando as rotas

O Laravel possui um comando que serializa todas as rotas da aplicação. Esses dados são passados para um único método em um arquivo cacheado. Isso diminui o tempo de carregamento das rotas da aplicação:

php artisan route:cache

O comando acima só funciona se não houver nenhuma chamada de função anônima nos arquivos de rota. A chamada de funções anônimas no arquivo de rota não é uma boa prática por padrão, o cache de rotas é mais um motivo para não usarmos.

php artisan view:cache

Veja abaixo de forma prática alguns modos de automatizar os processos mostrados nesse post:
https://www.youtube.com/watch?v=iKaPrEBoUZY&feature=emb_imp_woyt

<https://www.treinaweb.com.br/blog/como-colocar-uma-aplicacao-laravel-em-producao-e-automatizar-o-processo-de-deploy>

Caso o app esteja em um repositório

```
git clone http://[GIT_SERVER]/your-app.git
```

Criar um link simbólico

```
/ribamar.net.br/appfull/public/
```

```
para  
ribamar.net.br/public
```

```
cd appfull  
mv public public_bak  
cd ..  
ln -s appfull/public public
```

```
Editar o  
appfull/public/index.php
```

```
Trocar  
require __DIR__.'../bootstrap/autoload.php';  
$app = require_once __DIR__.'../bootstrap/app.php';
```

```
Por  
require __DIR__.'../bootstrap/autoload.php';  
$app = require_once __DIR__.'../bootstrap/app.php';
```

Lembre que a pasta /storage/ and /bootstrap/cache/ precisa ter permissão de escrita para o apache

```
chmod -R o+w storage  
chmod -R o+w bootstrap/cache
```

```
php artisan key:generate
```

<https://welcm.uk/blog/putting-a-laravel-app-into-production>

```
php artisan optimize
```

```
npm run production
```

Opcional

(Optional) Create a symbolic link from public/storage to storage/app/public (docs):

- php artisan storage:link

<https://stackoverflow.com/questions/59663762/laravel-what-steps-should-one-take-to-make-a-laravel-app-ready-for-production>

Optimize Image Size

use ssl certificate

set very strong database password

Forçando os seedes a funcionar em produção

Algumas operações de com seeders podem fazer com que você altere ou perca dados. Para protegê-lo da execução de comandos seeders em seu banco de dados em produção, você será solicitado a confirmar antes que os seeders sejam executados no ambiente de produção. Para forçar os seeders a serem executados sem uma solicitação de confirmação, use o sinalizador --force:

```
php artisan db:seed --force
```

<https://laratutorials.com/laravel-9-deploy-on-shared-hosting-cpanel/>

<https://www.cloudways.com/blog/install-laravel-on-digitalocean/>

<https://laravel.com/docs/9.x/deployment>

<https://www.treinaweb.com.br/blog/como-colocar-uma-aplicacao-laravel-em-producao-e-automatizar-o-processo-de-deploy>

12 – Boas Práticas

Adotar boas e consagradas práticas de programação, juntamente com bons padrões de projeto é importante.

- Estilos de codificação

Uma ótima ferramenta é o Visual Studio Code com suas várias extensões.

Uma delas é o

phpfmt – PHP formatter

- Basta selecionar todo o código a formatar

- Teclar Ctrl+Shift+P

- Selecionar: Format Document

Com isso ele já deixa o documento bem formatado: com indentação, linhas em branco para separação e melhor leitura, ...

É importante deixar o código bem legível:

```
class Nome{
```

```
    function  
    {  
        //  
    }  
}
```

Nomes de arquivos - tudo no singular e separados (compostos) por sublinhado

Nomes de classes - CamelCase. Exs: Clientes, MeusClientes, etc

Nomes de propriedades/variáveis, funções/métodos - camelCase. Exs: meuMetodo(), minhaVariavel

Constantes - tudo em maiúsculas. Ex.: MINHACONST

Prefira usar um bom framework para projetos importantes

Use um controle de versão, como o Git

Use o composer para instalação e controle de dependências

Use .gitignore para ignorar arquivos desnecessários

PHP

Uma das PSRs é a PSR 2, ou Guia de estilo de codificação (Coding Style Guide), que aborda como deve ser feita a formatação do nosso código para facilitar a leitura por outros desenvolvedores.

Algumas das indicações desse guia são:

- Devemos usar 4 espaços para indentação, não tabs.
- Não devemos fixar um numero de caracteres por linha, mas é bom que uma linha tenha menos de 80 caracteres.
- A abertura de colchetes de classes e métodos devem vir na proxima linha.
- A abertura de colchetes de estruturas de controle devem vir na mesma linha com um espaço em branco.

Busque boas referências sobre a programação:

- <https://php.net>
- <http://br.phptherightway.com/>
- <https://www.php-fig.org/>
- https://www.w3schools.com/php/php_ref_overview.asp
- <https://www.w3schools.com/php7/>
- Bons livros como o Modern PHP
- <https://www.eduardopires.net.br/2013/04/orientacao-a-objeto-solid/>
- <https://github.com/ziadoz/awesome-php>
- <https://github.com/jakzal/phpqa>
- Faça parte de bons grupos, foruns e listas de discussão para ajudar e ser ajudado
- Mantenha-se atualizado quanto ao PHP e a todo o conhecimento sobre as ferramentas que utiliza

Sempre crie um projeto para cada aplicativo ou site

Use notação do PHPDoc na documentação do seu código

Em qualquer dúvida consulte o manual em php.net

Ligue a exibição de erros no php.ini, com display_error = On e Error Reporting = E_All e nunca ocultar erros com @. Erros devem ser tratados e não ocultados

Use um bom editor de código ou uma boa IDE

Para trabalhos mais complexos use um bom framework

Fique atento para o DRY e KISS

Seja organizado com seu código

Utilize os operadores === e !==

Não use = em if: if(\$x = 3)

Zeros à esquerda e à direita: echo str_pad('9', 10, '0', STR_PAD_LEFT); // 0000000009

Use somente as tags

<?php e ?> e

<?= ... ?>

Use nomes consistentes e lógicos para variáveis, funções, métodos e classes. Evite nomes de variáveis com nomes muito curtos: \$a, \$ab, etc

Use bons comentários em locais que necessite, mas somente se bem necessário, pois é preferível que seu código seja tão claro e simples, que dispense comentários.

No trabalho com bancos de dados prefira o PDO

Sempre que possível use aspas simples ao invés de duplas

Armazene hashes de senhas criptografados no banco:

```
<?php
// Hash the password. $hashedPassword will be a 60-character string.
$hashedPassword = password_hash('my super cool password', PASSWORD_DEFAULT);

// You can now safely store the contents of $hashedPassword in your database!

// Check if a user has provided the correct password by comparing what they typed with our hash
password_verify('the wrong password', $hashedPassword); // false

password_verify('my super cool password', $hashedPassword); // true
```

Tenha bastante cuidado ao projetar seu banco de dados: tabelas, relacionamentos, tipos de dados dos campos, tamanhos, constraints, etc.

Use um sistema de cache com o PHP

Instale o Xdebug em seu php e habilite no seu editor/IDE, mas somente em desenvolvimento.

Use um editor com suporte a UTF8

Evite a criação de código complexo, como criação de funções dentro de loops

Evite criar variáveis desnecessárias. Ex:

```
$description = strip_tags($_POST['description']);

echo $description;
```

Se o arquivo contiver apenas código php (nada de html), então não use a tag de fechamento.

Sempre reutilize código. Conheça o código que em seu disco e também bibliotecas de terceiros para o assunto e reutilize sempre que puder.

Expressões regulares:

As funções `ereg_*` foram removidas no PHP 7, então essa fonte de confusão já passou por nós.

Sanitizar HTML:

- `strip_tags()`
- `filter_var()`
- `htmlentities()`
- `htmlspecialchars()`

Checar se null ou false com `===`

```
if($y === null){
    print('Perfect!');
}
if(strpos('abc', 'a') !== false){
    print('Found it for real this time!');
}
```

Use sempre a versão mais recente do php que puder

Reduza ao máximo a quantidade de consultas ao banco

<https://www.turbosite.com.br/blog/30-melhores-praticas-em-php-para-iniciantes>

Quanto melhor o desempenho do site/aplicativo, mais satisfeito ficam os usuários. Segundo pesquisas eles esperam que os sites carreguem em 2 segundos ou menos, sendo que eles abandonariam páginas que levassem mais de 3 segundos para carregar. E ainda: 79% dos clientes que tiveram problemas com o desempenho de um site não voltariam a fazer negócio. E 44% deles contariam aos amigos sobre a má experiência. Velocidade de carregamento é fundamental em qualquer site. E é o critério que mais pesa nas avaliações feitas por usuários. Afinal, ninguém poderá avaliar o design do site ou seu conteúdo antes dele carregar completamente. Muitas vezes, alguns desenvolvedores perdem alguns segundos de carregamento em prol de um efeito visual.

Bancos de dados

- Selecione apenas exatamente os campos que vai precisar usar e nunca `*`.
 - Modele zelosamente o banco de dados: tabelas, relacionamentos, nomes de campos, tipos de dados, constraints, etc.
 - Use bastante a cláusula `limit`.
 - Faça cache das consultas. Memcached ou APCu
 - Ao invés de sub-consultas use joins
 - Use union ao invés de OR
 - Utilize bastante índices, mas sem exagerar
 - Um servidor de buscas como o Elasticsearch pode trazer um desempenho de até 278 vezes maior.
- https://tableless.com.br/10-dicas-simples-para-acelerar-seu-site-ate-278-vezes/?utm_source=tablelessRelatedLink
- Geralmente quanto mais ações forem executadas no backend maior a performance

HTML

Estruture suas páginas com HTML semântico

Form design:

- Tenha certeza que o usuário entende o que e o porque você está pedindo ou perguntando algo
- Teste, teste novamente e quando achar que está tudo ok, teste de novo
- Escolha sabiamente os elementos de controle
 - tipos de dados/formato
 - máscaras
 - placeholder e dicas ao receber foco/tips
 - autocomplete
 - mensagens de erro

CSS

Nomes de classes compostos devem ser hifenizados: minha-classe

Use o VSCode para ajudar

Recursos modernos da programação com PHP:

Resumo do livro

Modern PHP

- Uso do SSH e do Git para envio do código para o servidor
- VirtualBox para criar máquinas virtuais
- Uso do Vagrant para ter localmente um servidor idêntico ao de produção otimizado
- Adoção de boas práticas e padrões de projeto: namespace (surgiu no PHP 5.3.0), autoload, Traits (PHP 5.4.0), MVC, Singleton, ActiveRecord, funções anônimas/closures (PHP 5.3.0), etc
- Uso do composer para gerenciar pacotes e dependências juntamente com GitHub e Packagist
- Uso do PHPUnit para testar o código
- Uso de cache embutido (Zend OP Cache - 5.5.5)
- Uso de migrations
- Uso de recursos do PHP na linha de comando/terminal
- Aparecimento de bons frameworks que implementam boas práticas e padrões de projeto, além de usar o PHPOO: classes, herança, interfaces, etc
- Use bastante o servidor web embutido: php -S localhost:4000

O ecossistema PHP moderno é um verdadeiro caldeirão de código que ajuda os desenvolvedores a criar aplicativos incríveis.

Últimas novidades

<https://www.php.net/manual/en/features.php>

Se você fizer referência a uma classe, interface, função ou constante sem um namespace, o PHP assume que a classe, interface, função ou constante vive no namespace atual.

Se você precisa fazer referência a uma classe, interface, função ou constante com namespace dentro de outra namespace, você deve usar o nome de classe PHP totalmente qualificado (namespace + nome de classe).

Você pode digitar o nome da classe PHP totalmente qualificado ou importar o código para o namespace atual com a palavra-chave use.

Carregamento automático

Os namespaces também fornecem a base para o padrão de autoloader PSR4 criado pelo Grupo de interoperabilidade do framework PHP (PHP-FIG). Este padrão de autoloader é usado pela maioria componentes PHP modernos e nos permite carregar automaticamente as dependências do projeto usando o Composer gerenciador de dependências.

Uma interface dissocia nosso código de suas dependências e permite que nosso código dependem de qualquer código de terceiros que implemente a interface esperada. Não nos importamos como o código de terceiros implementa a interface; nos importamos apenas que o código de terceiros implementar a interface. Aqui está um exemplo mais realista.

Este é exatamente o mesmo conceito em PHP orientado a objetos. Se eu escrever um código que espera um objeto de uma classe específica (e, portanto, uma implementação específica), a utilidade do meu código é inerentemente limitado porque só pode usar objetos dessa classe, para sempre. No entanto, se eu escrever código que espera uma interface, meu código imediatamente sabe como usar qualquer objeto que implementa essa interface. Meu código não se importa como a interface é implementada; meu código se importa apenas que a interface seja implementada.

Um trait se comportam como classes, mas se parecem com interfaces.

Um trait é uma implementação de classe parcial (ou seja, constantes, propriedades e métodos) que pode ser misturado em uma ou mais classes PHP existentes. Os traços funcionam em dupla função: eles dizem que classe pode fazer (como uma interface), e eles fornecem uma implementação modular (como uma classe).

Servidor web embutido no PHP

```
php -S localhost:4000
```

Recursos do PHP Moderno

- Composer
- Namespace
- Git
- GitHub, GitLab, BitBucket e Packagist
- PSRs - <http://www.php-fig.org/>
- XDebug
- Try catch
- Inúmeros pacotes disponíveis/componentes: Whoops, Migrations (Phinx)
- VSCode

- PHPUnit
- phpDoc
- PHPCodeSniffer
- Padrões de Projeto
- MVC
- Frameworks
- CMSs
- PDO
- ORMs
- Segurança
- Servidores tipo VPS
- Virtualização e Containerização: Virtualbox, Vagrant, Docker
- SOLID is a mnemonic to remind us of five key principles in good object-oriented software design.
- Novos recursos do PHP 7 e PHP 8

O perigo do extremismo

Um problema com regras e diretrizes na programação é que elas geralmente só servem a um propósito em um contexto específico. Saindo desse contexto, uma boa regra pode se tornar uma regra horrível. De fato, toda boa regra se torna ruim quando levada ao extremo.

O princípio KISS, que é um acrônimo para “Keep It Simple, Stupid”, é um bom e extremamente sábio princípio que geralmente é visto por pessoas experientes como um conselho muito bom a seguir, mas mesmo este grande princípio torna-se um perigo para um projeto, se levado ao extremo. Existe tal coisa como “muito simples” resultando em falta de funcionalidade necessária.

Use os frameworks atuais com moderação. Antes de usar verifique se é adequado.

Não somos obrigados a sempre usar padrões de projetos em nosso código. É importante usar quando eles nos trará vantagens e evitar quando eles nos trouxer grande complexidade.

Assim também vale para a orientação a objetos. Alguns pequenos projetos não justifica o uso da POO.

Siga o PHP-FIG mas com critérios, de forma a colher algo de útil para você e sua empresa. Não para procurar fazer tudo que o grupo criou.

Fique sempre atento aos bons comportamentos para deixar seu código e aplicativo mais seguros.

<http://br.phptherightway.com/>

https://phpthewrongway.com/pt_br/

<https://www.freecodecamp.org/news/this-is-what-modern-php-looks-like-769192a1320/>

<https://www.airpair.com/php/posts/best-practices-for-modern-php-development>

<https://github.com/odan/learn-php>

<https://medium.com/@FernandoDebrand/guia-pr%C3%A1tico-do-modern-php-desenvolvimento-e-ecossistema-c9715184e463>

https://phpthewrongway.com/pt_br/

Clean Code

<https://github.com/jupeter/clean-code-php>

<https://github.com/fernandowobeto/clean-code-php>

<https://github.com/VadimGut/clean-code-php>

<https://github.com/subrata6630/Clean-Code-PHP>

<https://github.com/jupeter/clean-code-php>

Porque usar um CMS

Formas principais de fazer um site:

1. Criá-lo com arquivos HTML estáticos;
2. Programar todo o site, com uma linguagem de programação e um banco de dados;
3. Utilizar um Sistema de Gerenciamento de Conteúdo (CMS), como o WordPress, ou Joomla;

Criar páginas estáticas tem vários problemas:

- Adicionar uma página nova pode exigir alterações manuais em todas as outras páginas;
- Mudar elementos como menus e headers, também vão exigir mudanças em todas as outras páginas (a não ser que você use iframes, mas se você faz isso, meus pêsames);
- Quando o cliente decidir mudar o visual e estrutura do site, vai haver muito retrabalho. É melhor jogar tudo fora e começar outro novo;
- Fica muito mais fácil de acontecerem erros, devido a links quebrados, ou tags escritas incorretamente (afinal, qualquer um pode errar uma tag, às 3h da manhã, tendo que entregar tudo às 8h);
- Segurança e técnicas novas de otimização serão difíceis de serem adicionadas, devido aos primeiros itens desta lista;

Os bons CMS's já trazem funções como:

- Gerenciamento de usuários, logins, permissões
- Categorização de conteúdos
- Construção de menus
- Temas visuais para todas as páginas
- Mecanismos de imagens e vídeos para as páginas
- Sistemas de blogs
- Formulários de contato
- Tradução e internacionalização

<https://webdevacademy.com.br/artigos/por-que-usar-cms/>

Porque usar um bom Framework

Um framework é um conjunto de códigos comuns abstraídos de vários projetos com o objetivo de prover funcionalidades genéricas em um projeto que utilize esse framework. Através de programação, ele permite a customização dessas funcionalidades para torná-las mais específicas de acordo com a exigência de cada aplicação.

De forma resumida o framework é uma estrutura, uma fundação para você criar a sua aplicação. Em outras palavras o framework te permite o desenvolvimento rápido de aplicações (RAD), o que faz economizar tempo, ajuda a criar aplicações mais sólidas e seguras além de reduzir a quantidade de código repetido.

Os desenvolvedores utilizam frameworks por vários motivos, e o maior deles é para agilizar o processo de desenvolvimento. A re-utilização de código em vários projetos vai economizar muito tempo e trabalho... Isso é garantido pois o framework já traz uma série de módulos pré-configurados (e funcionando) para fazer as mais variadas e comuns tarefas como envio de e-mails, conexão com o banco de dados, sanitização (limpeza) de dados e proteção contra ataques.

Não recomendado para:

- Programadores iniciantes
- Projetos pequenos

Pontos fortes:

- Qualidade do código do aplicativo
- Segurança
- Produtividade
- Fácil manutenção do código
- Documentação e material online fartos
- Pequeno tempo de desenvolvimento

13 – Laravel Skeleton

Chamamos de esqueleto uma instalação do laravel com alguns pacotes instalados e configurados e também com algumas customizações, de forma que nos facilite a criação de aplicativos e já traga recursos que costumamos usar.

Para que fique mais prático criamos um repositório público e o publicamos no packagist.org

```
laravel new skeleton
```

```
cd skeleton
```

Ajustar o composer.json

```
"name": "ribafs/laravel-skeleton",  
"type": "project",  
"description": "Laravel 9 Skeleton",
```

Customizar o README.md

```
# Laravel 9 Skeleton
```

```
## Pacotes instalados
```

```
crud-generator-laravel-br
```

```
e vários outros
```

```
## CRUD criado
```

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

```
## Security Vulnerabilities
```

If you discover a security vulnerability within Laravel, please send an e-mail to Taylor Otwell via taylor@laravel.com. All security vulnerabilities will be promptly addressed.

```
## License
```

The Laravel framework is open-sourced software licensed under the [MIT license](https://opensource.org/licenses/MIT).

Alguns pacotes

```
composer require laravel/ui
php artisan ui bootstrap --auth
```

```
composer require ribafs/crud-generator-laravel-br
php artisan vendor:publish --provider="Ribafs\CrudGenerator\CrudGeneratorServiceProvider"
```

```
composer require laravelcollective/html
```

Criar um CRUD

```
php artisan crud:generate Posts --fields='name#string;' --view-path= --controller-namespace=App\
Http\Controllers --route-group= --form-helper=html
```

- Seeders para posts

```
php artisan make:seeder PostsSeeder
```

Editar o databases/seeder/PostsSeeder.php e atualizar:

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class PostsSeeder extends Seeder
{
    public function run()
    {
        $faker = \Faker\Factory::create();// em config/app.php já está: 'fallback_locale' =>
        'pt_BR',
        for ($i=0; $i < 100; $i++) {
            DB::table('posts')->insert([
                'name' => $faker->name,
            ]);
        }
    }
}
```

Jogar na tabela:**Criar banco e configurar .env**

```
php artisan migrate
php artisan db:seed --class=PostsSeeder
```


- Configurar e melhorar a Paginação com Bootstrap

Editar app/Providers/AppServiceProvider.php e atualizar:

```
use Illuminate\Pagination\Paginator;

public function boot()
{
    Paginator::useBootstrap();
}
```

- Instalar para customização da paginação

php artisan vendor:publish --tag=laravel-pagination

Será Criada a pasta
/resources/views/vendor/pagination

Contendo os arquivos:

```
bootstrap-4.blade.php
bootstrap-5.blade.php
default.blade.php
semantic-ui.blade.php
simple-bootstrap-4.blade.php
simple-bootstrap-5.blade.php
simple-default.blade.php
simple-tailwind.blade.php
tailwind.blade.php
```

Copiei a
bootstrap-5.blade.php

Para
bootstrap-5custom.blade.php
E customizei para:

```
@if ($paginator->hasPages())
    <nav class="d-flex justify-items-center justify-content-between">
        <div class="d-flex justify-content-between flex-fill d-sm-none">
            <ul class="pagination">
                {{-- Previous Page Link --}}
                @if ($paginator->onFirstPage())
                    <li class="page-item disabled" aria-disabled="true">
                        <span class="page-link">@lang('pagination.previous')</span>
                    </li>
                @else
                    <li class="page-item">
```

```

        <a class="page-link" href="{{ $paginator->previousPageUrl() }}"
rel="prev">@lang('pagination.previous')</a>
    </li>
@endif

    {{ -- Next Page Link -- }}
    @if ($paginator->hasMorePages())
        <li class="page-item">
            <a class="page-link" href="{{ $paginator->nextPageUrl() }}"
rel="next">@lang('pagination.next')</a>
        </li>
    @else
        <li class="page-item disabled" aria-disabled="true">
            <span class="page-link">@lang('pagination.next')</span>
        </li>
    @endif
</ul>
</div>

<div class="d-none flex-sm-fill d-sm-flex align-items-sm-center justify-content-sm-between">
<div>
    <p class="small text-muted">
        {!! __('Showing') !!}
        <span class="font-medium">{{ $paginator->firstItem() }}</span>
        {!! __('to') !!}
        <span class="font-medium">{{ $paginator->lastItem() }}</span>
        {!! __('of') !!}
        <span class="font-medium">{{ $paginator->total() }}</span>
        {!! __('results') !!}
    </p>
</div>

<div>
    <ul class="pagination">
        {{ -- Previous Page Link -- }}
        @if ($paginator->onFirstPage())
            <li class="page-item disabled" aria-disabled="true" aria-
label="@lang('pagination.previous')">
                <span class="page-link" aria-hidden="true">Primeira</span>
            </li>
        @else
            <li class="page-item">
                <a class="page-link" href="{{ $paginator->previousPageUrl() }}" rel="prev" aria-
label="@lang('pagination.previous')">Anterior</a>
            </li>
        @endif

        {{ -- Pagination Elements -- }}

```

```

    @foreach ($elements as $element)
        {{-- "Three Dots" Separator --}}
        @if (is_string($element))
            <li class="page-item disabled" aria-disabled="true"><span class="page-
link">{{ $element }}</span></li>
        @endif

        {{-- Array Of Links --}}
        @if (is_array($element))
            @foreach ($element as $page => $url)
                @if ($page == $paginator->currentPage())
                    <li class="page-item active" aria-current="page"><span class="page-
link">{{ $page }}</span></li>
                @else
                    <li class="page-item"><a class="page-link"
href="{{ $url }}">{{ $page }}</a></li>
                @endif
            @endforeach
        @endif
    @endforeach

    {{-- Next Page Link --}}
    @if ($paginator->hasMorePages())
        <li class="page-item">
            <a class="page-link" href="{{ $paginator->nextPageUrl() }}" rel="next" aria-
label="@lang('pagination.next')">Próxima</a>
        </li>
    @else
        <li class="page-item disabled" aria-disabled="true" aria-
label="@lang('pagination.next')">
            <span class="page-link" aria-hidden="true">Última</span>
        </li>
    @endif
</ul>
</div>
</div>
</nav>
@endif

```

- Implementação do AdminLTE

```

composer require infyomlabs/laravel-ui-adminlte
php artisan ui adminlte --auth

```

Ícones encontrados aqui:

<https://fontawesome.com/v5/search>

Ele substituiu o `resources/views/layouts/sidebar.blade.php` por um dele, que chama o `menu.blade.php`, criado por ele.
 Editei o `menu.blade.php` e adaptei para que ficasse parecido com o antigo `sidebar.blade.php`.

Atualizar `resources/views/layouts/menu.blade.php` para:

```
<li class="nav-item">
  <a href="{{ route('home') }}" class="nav-link {{ Request::is('home') ? 'active' : '' }}">
    <i class="nav-icon fas fa-home"></i>
    <p>Home</p>
  </a>
  <a href="{{ url('posts') }}" class="nav-link {{ Request::is('posts') ? 'active' : '' }}">
    <i class="nav-icon fas fa-people-arrows"></i>
    <p>Posts</p>
  </a>
</li>
```

- Localization

```
composer require lucascudo/laravel-pt-br-localization --dev
php artisan vendor:publish --tag=laravel-pt-br-localization
```

Configurações para utilizar 'pt-BR' como linguagem padrão

```
// Altere Linha 83 do arquivo config/app.php para:
'locale' => 'pt-BR',
```

Caso deseje, configure o Framework para utilizar 'America/Sao_Paulo' como data hora padrão

```
// Altere Linha 70 do arquivo config/app.php para:
'timezone' => 'America/Fortaleza',
```

Customizar a paginação em

`app/Http/Controllers/PostController.php`

Mudando 25 para 7 no método `index()`

Exemplo de Customização de mensagens de erro

Criar a pasta
`resources/views/errors`

E dentro dela o arquivo
`403.blade.php`

Contendo:

```
@extends('layouts.app')

@section('content')
<div class="text-center">
    <h1>{{ 'Acesso negado' }}</h1>
    <a href="#" onClick="history.back()">Voltar</a>
</div>
@endsection
```

Gerar seed de tabelas existentes

composer require orangehill/iseed

php artisan iseed nomeTabela

<https://github.com/orangehill/iseed>

- Testando

```
npm install
npm run dev
npm install resolve-url-loader@^5.0.0 --save-dev --legacy-peer-deps
npm run dev
```

composer du

```
php artisan migrate
php artisan db:seed --class=PostsSeeder
```

php artisan serve

http://127.0.0.1:8000/posts

Criar repo no GH chamado

laravel-skeleton

Criado sem README.md e com licença MIT

Após publicar no packagist.org criar o primeiro release

Na prática, podemos pegar todo o aplicativo criado ou qualquer outro e criar um esqueleto.

Como instalar

```
composer create-project --prefer-dist ribafs/laravel-skeleton skeleton
```

14 – Criando um demo para o aplicativo

Criarei um usuário especificamente para o demo e um banco, sendo o user dono do banco demo. Aqui irei criar um banco e um user específico, que tem acesso somente a leitura dos registros, para que fique seguro e não comprometa o servidor

```
sudo mariadb
```

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'password';
flush privileges;
```

```
CREATE DATABASE demoapp_db;
```

```
CREATE USER 'demoapp_us'@'localhost' IDENTIFIED BY 'zmxnxxxxD@';
GRANT ALL PRIVILEGES ON demoapp_db.* TO 'demoapp_us'@'localhost';
\q
```

```
nano .env
```

```
php artisan migrate
php artisan db:seed --class=PermissionsSeeder
php artisan db:seed --class=RolesSeeder
php artisan db:seed --class=UsersSeeder
php artisan db:seed --class=PostsSeeder
php artisan db:seed --class=CommentsSeeder
```

```
php artisan serve
```

```
http://127.0.0.1:8000
```

Efetuar login com os 4 users para testar.

Efetuar logout e voltar ao terminal

```
Ctrl+C
```

Remover todos os privilégios do user demoapp_us e atribuir somente o privilégio de select em todas as tabelas e full na tabela sessions.

```
REVOKE ALL, GRANT OPTION FROM 'demoapp_us'@'localhost';
GRANT SELECT ON demoapp_db.* TO 'demoapp_us'@'127.0.0.1' IDENTIFIED BY
'zmxnxxxxD@';
GRANT ALL ON demoapp_db.sessions TO 'demoapp_us'@'127.0.0.1' IDENTIFIED BY
'zmxnxxxxD@';
```

Efetuar logout e depois logar novamente

Agora testar as operações edit, add e delete.

Tratando os erros de acesso:

Mudar no .env

Debug para false

APP_ENV para production

Criar o arquivo

resources/views/errors/500.blade.php

```
@extends('layouts.app')
```

```
@section('content')
```

```
<div class="text-center">
```

```
    <h3 align="center">500 - Esta operação é permitida somente na versão full do livro, link  
abaixo<br>
```

```
    <h3 class="text-center"><a href="https://ribamar.net.br/portal/livro/laravel9"  
target="_blank">Livro sobre Laravel 9</a></h3>
```

```
    <h3 class="text-center"><a href="/home">Voltar</a></h3>
```

```
</div>
```

```
@endsection
```

php artisan serve

Efetuar login para tastar

O demo está aqui:

<http://198.58.97.36>

Todos os arquivos do app estão no raiz (/var/www/html) e o virtualhost aponta para /var/www/html/public

15 – Customização do Aplicativo

Customizar nosso aplicativo central do Livro para usar para cadastrar os leitores

Para isso efetuei as seguintes customizações

- Removi os controllers que não usarei: Post e Comment
- Removi os models: Post e Comment
- Removi as views respectivas: posts e comments
- Removi as roles posts e comments
- Removi as migrations
- Removi o debugbar

Removi as respectivas entradas em config/app.php
composer remove barryvdh/laravel-debugbar

Ajustei os seeders:

Removi as permissões para posts e comments e adicionei reader

Roles removi todas e mudei uma para RibaFS

Removi todos os users e mudei Super para RibaFS

Veja a migration criada pelo gerador de cruds

```
Schema::create('readers', function (Blueprint $table) {  
    $table->increments('id');  
    $table->string('name', 50)->notNullable();  
    $table->string('email', 150)->notNullable();  
    $table->date('date')->notNullable();  
    $table->boolean('active')->notNullable();  
    $table->text('obs')->nullable();  
    $table->timestamps();  
});
```

O resultado está aqui

<https://ribamar.net.br/portal/livro/material-livro>

Ao final do artigo.

16 – Referências

<https://laravel.com/docs/9.x/releases>

<https://laravel.com/docs/9.x/upgrade>

<https://kinsta.com/pt/blog/laravel-9/>

<https://blog.logrocket.com/whats-new-laravel-9/>

<https://www.cloudways.com/blog/laravel-9/>

<https://masteringbackend.com/posts/laravel-9-tutorial-laravel-9-new-features>

<https://kinsta.com/pt/blog/tutoriais-laravel/>

<https://laravel.com/docs/9.x/installation>

<https://laravel-news.com/your-first-laravel-application>

<https://infyom.com/>

The Missing Laravel Admin

<https://voyager.devdojo.com/>

Class to dump a database using PHP. Currently MySQL, PostgreSQL, SQLite and MongoDB

<https://github.com/spatie/db-dumper>

Curso Gratuito de Laravel 9.x - Carlos Ferreira

<https://www.youtube.com/watch?v=9VYjk6cPRIw&list=PLVSNL1PHDWvS1e1aeoJV7VvaDZ9m67YPU&index=2>

Curso gratuito de laravel 9 - 34 aulas

<https://www.youtube.com/watch?v=h7pq2uc6e4&list=PLcoYAcR89n-reidRFA3XCivQPeKFt4dQU>

<https://www.cloudways.com/blog/best-laravel-packages/>

Controle de estoque com laravel 5.1

<https://www.youtube.com/watch?v=u9hSMixPau0&list=PLS2HpSNAcox-gzxKVSbWNwzvgrwdSBEDU>

<https://www.youtube.com/watch?v=h5ysXC0j3ZI>

<https://github.com/orchidsoftware/platform>

<https://github.com/Labs64/laravel-boilerplate>

<https://github.com/the-control-group/voyager>

[Live] PHP e Laravel na Prática: criando um projeto open source do zero, com Laravel 8 e Github

https://www.youtube.com/watch?v=UngvQxEsK_M

Sistema para gerenciamento para escola em Laravel com Angular opensource (grátis)

https://www.youtube.com/watch?v=kd0y0e_2dR4

<https://github.com/laradock/laradock>

<https://github.com/daniel0ferraz/ApiEstoque>

<https://github.com/daniel0ferraz/ControleDeEstoque>

Criando um Ecommerce com Laravel

<https://www.youtube.com/c/TheCoderBr/search?query=ecommerce>

COMO CRIAR UM APLICATIVO DE GERENCIAMENTO DE ESTOQUE

<https://www.youtube.com/playlist?list=PLDwHKy6FYybvfJnfm4ppSN6iPvGw1lIdb>

<https://www.cloudways.com/blog/best-laravel-packages/>

<https://spdload.com/blog/best-laravel-tools-and-resources/>

<https://www.bacancytechnology.com/blog/laravel-packages>

<https://github.com/spatie/laravel-permission>

<https://spatie.be/docs/laravel-permission>

<https://bestlaravel.com/t/packages> - 333 pacotes

<https://kinsta.com/blog/laravel-tutorial/>

<https://github.com/chiraggude/awesome-laravel>

<https://github.com/amranidev/scaffold-interface>

<https://amranidev.github.io/scaffold-interface/docs/>

<https://infyom.com/open-source/>

<https://github.com/ribafs/laravel-generator>

CMS

<https://github.com/octobercms/october>

<https://octobercms.com/>

<https://github.com/pyrocms/pyrocms>

<https://www.pyrocms.com/>

<https://github.com/microweber/microweber>

<https://microweber.org/>

E-commerce

<https://github.com/aimeos/aimeos-laravel>

<https://bagisto.com/en/> e <https://github.com/bagisto/bagisto>

<https://github.com/sandervanhooft/laravel-invoicable>

<https://codingcompiler.com/best-laravel-tools-packages/>

AdminLTE

<https://www.cafeteria.id/2022/02/cara-integrasi-laravel-9-dengan-laravel.html>

<https://websolutionstuff.com/post/laravel-9-user-role-and-permission>

Criar painel admin

<https://www.youtube.com/watch?v=CcuHFuv-fn8>

<https://www.tutsmake.com/laravel-9-ckeditor-image-upload-tutorial/>

<https://www.codecheef.org/article/how-to-use-ckeditor-in-laravel-9-application>

<https://laratutorials.com/how-to-install-and-use-ckeditor-in-laravel-9/>

<https://www.positronx.io/how-to-install-integrate-ckeditor-in-laravel/>

<https://techtutorial.net/laravel-tutorial/how-to-install-and-use-ckeditor-in-laravel/>

<https://www.tutsmake.com/laravel-9-generator-qr-code-tutorial-with-example/>

<https://www.tutsmake.com/laravel-9-livewire-pagination-with-search-example/>

<https://www.tutsmake.com/laravel-9-livewire-pagination-example/>

<https://onlinewebtutorblog.com/concept-of-trait-in-laravel-9-tutorial-with-example/>

<https://www.tutsmake.com/laravel-9-crud-application-tutorial-with-example/>

<https://www.itsolutionstuff.com/post/laravel-9-crud-application-tutorial-exampleexample.html>

<https://onlinecode.org/laravel-9-crud-example-laravel-9-beginners-tutorial/>

<https://www.nicesnippets.com/blog/laravel-9-crud-operation-example-tutorial>

<https://codingdriver.com/laravel-9-crud-example-tutorial.html>

<https://www.itsolutionstuff.com/post/laravel-9-crud-with-image-upload-tutorialexample.html>

<https://www.tutsmake.com/laravel-9-vue-js-crud-example-tutorial/>

<https://laravel.com/docs/9.x/deployment>

<https://www.tutsmake.com/laravel-9-rest-api-crud-example-tutorial/>

<https://www.itsolutionstuff.com/post/laravel-9-rest-api-authentication-using-sanctum-tutorialexample.html>

<https://larainfo.com/blogs/laravel-9-rest-api-crud-tutorial-example>

<https://onlinecode.org/laravel-9-rest-api-crud-tutorial-with-example/>

<https://onlinewebtutorblog.com/laravel-9-many-to-many-eloquent-relationship-tutorial/>

<https://kinsta.com/pt/blog/frameworks-php-populares/>

<https://gabrielcassimiro.medium.com/github-e-github-desktop-b%C3%A1sico-f439879cb087>

<https://docs.github.com/pt/desktop/installing-and-configuring-github-desktop/overview/getting-started-with-github-desktop>

<https://www.codecademy.com/article/what-is-git-and-github-desktop>

Estrutura de diretório do laravel 9

<https://learn2torials.com/a/laravel9-directory-structure>

PHP

Crea Tu Framework MVC con PHP

https://www.youtube.com/playlist?list=PLty0cFLf07jXQA5_P9rDMWjpEet2wTXN1

17 – Bônus

Criando aplicativo com ACL usando o CakePHP 3

O laravel é o melhor framework PHP ao meu ver, em geral, mas existe algo que não consigo fazer usando laravel, que é instalar dois ou mais aplicativos num mesmo servidor, num mesmo domínio. Usando laravel consigo apenas se instalar somente o aplicativo laravel no raiz do servidor web.

Com o CakePHP conseguimos isso de forma default, sem precisar de nenhum ajuste. Como criei um plugin para implantar ACL em aplicativos do CakePHP 3, por isso estou trazendo ele aqui como alternativa de framework, como o plano B. Não consegui atualizar o plugin para a versão 4 do Cake, por isso passo o plugin na versão 3.

<https://github.com/ribafs/admin-br>

O projeto está bem documentado, mas se sobrar alguma dúvida mande para o grupo, ok?

Precisa tomar o cuidado de usar a versão 3 do Cake e não a versão atual. Segue o comando

```
composer create-project --prefer-dist cakephp/app:~3.0 nomeAplicativo
```