Criando Projetos com Laravel

Sumário

Justificativa deste livro

Porque sobre laravel?
Pra começar é o assunto que mais tenho estudado nos últimos tempos. O laravel não é apenas um framework em PHP. Basta olhar que o PHP é uma linguagem ainda muto popular pelos seus feitos mas bem impopular entre programadores de outras linguagens. Mas mesmo assim o laravel tem penetração entre estes. Sem contar que é um verdadeiro canivete suiço. Suas facilidades, somadas com as do PHP são muito importantes.
Só para dar uma ideia da sua versatilidade aqui, entre os projetos citados, está um que criei há uns 15 dias para servir de diário para mim. Ele substituiu um que usei por uns 5 anos feito no LibreOffice Writer. Tem mais vários aplicativos úteis, boa parte criada por mim, mas não somente.

Repositório com todos os projetos

https://github.com/ribafs2/laravel-projetos

Inclusive o PDF

Requisitos

- Conhecimento de PHP e de Laravel
- Ambiente de programação no desktop com PHP 8.2, pode ser num Linux ou no Windows com WSL2

Agradecimentos

Claro que em primeiro lugar quero agradecer à equipe que criou e que mantém esta excelente ferramenta.

Os melhores sites que conheço com conteúdo sobre laravel e a quem especialmente quero agradecer:

ItSolutionStuff.com | Web Development Tutorials & Solutions – ItSolutionStuff.com
All PHP Tricks - Web Development Tutorials and Demos
https://laravel.com/
https://laravel-news.com/

Também agradecer a todos que produzem conteúdo, tutoriais, dicas, vídeos, ao grande buscador e a todos os colegas dos grupos laravel e outros no Facebook, que ajudam a tirar dúvidas.

Sobre o autor

Ribamar FS (ribafs), engenheiro civil que virou programador PHP há mais de 20 anos, apaixonado pela programação e em particupar pelo PHP e pelo Laravel.

Já elaborei uns 20 livros na área e estarão em:

Minha primeira conta, abandonada, conta com mais de 1.2k repositórios, a maioria bons forks.
https://github.com/ribafs?tab=repositories

A conta atual e ainda sendo atualizada
https://github.com/ribafs2?tab=repositories

Um domínio ainda sem uso:
https://ribamar.net.br

Com conta na Contabo, aguardando uso.

Introdução

Tenho consciência de que cada programador é um universo, que tem seus gostos pessoais, seus interesses e necessidades. Então o que vou falar não é para influenciar, mas como depoimento pessoal. Durante mais de 20 anos estudando programação e em sua maioria web, eu tenho estudado muita coisa: Linux, PHP, PostgreSQL, MySQL, Joomla, Laravel, etc. Estive observando que o assunto que mais tenho estudado nos últimos tempos tem sido Laravel e uma boa referência para isso é minha antiga conta no Github, onde percebi isso pesquisando em meus repositórios por Laravel. Realmente uma excelente ferramenta e muito versãtil.

Preciso chamar a atenção para o fato de que o laravel tem uma enorme quantidade de recursos e que eu não estudo todos eles, me concentro em uma parte que me atende.

É importante conhecer os principais tipos de softwares existentes para quando chegar o momento de criar um certo software termos clareza de qual ferramenta usar para maior eficiência e produtividade.

Na programação web temos basicamente dois tipos de softwares principais, os CMS e os frameworks. Os CMS são especializados na criação de sites e blogs. Já os frameworks são especializados na criação de aplicativos, tipo CRUDs, cadastros, controles, etc.

Eu basicamente tenho estudado e trabalhado muito com Joomla (CMS) e Laravel (framework) e ultimamente estudei um pouco Wordpress para a criação de um site tipo landing page.

Muitas vezes eu preciso criar um CRUD. Neste caso me volto para o laravel, especialmente usando um gerador de CRUDs que automatiza muita coisa. Caso eu tenha que criar vários CRUDs com as tabelas relacionadas, então a ferramenta que irá me ajudar mais, fazendo muito por mm, será o framework CakePHP, que já vem nativamente com a ferramenta bake e esta gera CRUDs relacionados para cada tabela.

Se preciso criar um site, geralmente recorro ao Joomla, mas atualmente posso usar o Wordpress.

Como gosto de desafios, até que tento fazer com que um destes softwares faça tudo. Certa vez, quando o Joomla estava na versão 2.5, eu cheguei a criar um controle de estoque ou similar com o Joomla, até escrevi um pequeno livro sobre isso. Mas alterar o software de forma que na próxima atualização terá bastante trabalho para adaptar ou então poderá perder tudo, não é uma boa ideia. A minha conclusão é que eu use cada tipo de software para a atividade que ele mais entende.

Curiosidade: eu mantenho um diários há uns 5 anos no LibreOffice Writer. Há uns 15 dias eu pensei, porque não criar algo com uma linguagem desktop? Então saí pesquisando e não encontrei nada que me atendesse. Então lembrei que tenho forte experiência com programação web e que sempre tenho um servidor web em meu computador. Portanto posso criar um aplicativo para isso em meu desktop. Eu pensei em criar com PHP puro, mas logo mudei para o Laravel e usando o referido gerador. Então fiz algo no código que antess eu sempre fazia manualmente. Por default ele já usa o dia de hoje em cada novo registro do diário. Adicionei o CKEditor nos campos textarea e outros bons recursos. E continuo atualizando.

Esse gerador merece um destaque. Gosto muito deste gerador de CRUDs
https://github.com/sohelamin/crud-generator

Acontece que a versão atual usa o Taiwind e eu gosto de usar o Bootstrap. Dando uma olhada no código, resolvi fazer um fork e alterar para que trabalhe com o Bootstrap e publiquei no Packagist. E deu neste
https://github.com/ribafs2/gerador-cruds
Assim eu fui otimizando para que ele seja como eu gosto, criando novos releases.

Com isso acho importante ressaltar a importância do Github para a programação em geral e não somente. Uma excelente estrutura, com plano gratuito e bem generoso. Sinto como se trabalhasse em equipe, pois quando vejo um repositório que me agrada eu crio um fork. Algumas vezes faço correções, outras melhorias e muitos apenas guardo na minha conta. Sem contar que ter exemplos de nível que podem ser explorados e estutados a qualquer momento é muito importante.

Produtividade na programação

Especialmente com laravel mas não somente

Eu tenho como hábito melhorar minhas atividades, fazer mais rápido e melhor, especialmente atividades repetitivas.

Abaixo seguem algumas destas atividades. Assim ganho tempo a prazer, evitando atividades repetitivas. Meu tempo para atividades criativas e para resolver problemas consequentemente aumenta.

Vejamos apenas alguns exemplos que eu utilizo:

- **Organização**. Um programador organizado tem 24 horas em cada dia. Um programador que não se interessa por organização geralmente tem uma quantidade útil de horas no dia reduzidas. Perde parte procurando algo e geralmente perde muito.

- **Executar todas as migrations** juntamente com todos os seeeders no Laravel
php artisan migrate --seed

Ao invés de digitar tudo isso acima, sabe como faço?
ams

Somente mas, artisan mirgate seeder para lembrar.

Inicialmente eu criava scripts para todas estas ideias, depois aprendi que posso fazer com mais facilidade usando alias no .bashrc.

Seguem como exemplo os aliases que uso e alguns scripts na pasta scripts.

Algo sobre a História do Laravel

Taylor Otwell criou o Laravel como uma tentativa de fornecer uma alternativa mais avançada para o framework CodeIgniter, que não fornecia certos recursos, como suporte integrado para autenticação e autorização de usuário.

A 1º versão do Laravel foi lançada em junho de 2011, ela incluía suporte à localização de linguagem, a models e views, sessões, rotas e outros mecanismos.

O suporte aos controllers foi adicionado na versão 2º versão, onde o Laravel se tornou um framework MVC completo. Foi lançado também um sistema de templates chamado Blade e o Laravel passou a implementar os princípios da Inversão de controle (Inversion of Control ou IoC, em inglês).

Laravel 3 foi lançado em Fevereiro de 2012 com diversas funcionalidades, incluindo uma interface de linha de comando (command-line interface ou CLI, em inglês) chamado de Artisan, suporte a diversos Sistema Gerenciador de Banco de Dados – SGBDs, as chamadas migrations como uma forma para controle de versão dos bancos de dados.

Laravel 5 foi lançado em Fevereiro de 2015 como resultado de mudanças internas que acabaram na renumeração do então futuro lançamento do Laravel 4.3. O Laravel 5 criou uma nova estrutura de arvore de diretório interna para o desenvolvimento de aplicações.

Laravel 6 foi lançado em 3 de setembro de 2019, trazendo compatibilidade com Laravel Vapor[2] e novas funcionalidades como Versionamento Semântico, melhoria nas respostas de autorização, um novo recurso de middleware na classe Job – Job Middleware, as Lazy Collections, novos recursos de query Eloquent e o pacote Laravel UI.[3]

Laravel 7 foi lançado em 3 marco de 2020, com nova funcionalidades como o Laravel Sanctum, Custom Eloquent Casts, melhoria nas Tags de componentes Blade, uma API mínima e expressiva em torno do cliente HTTP Guzzle. Melhoria na velocidade de cache da rota.[4]

Laravel 8 foi lançado em 8 de setembro de 2020, com novas mudanças no esquema de versionamento do framework, agora os lançamentos primários serão a cada seis meses (Março e Setembro) enquanto versões secundárias ou patch de correção podem ser lançados frequentemente. A política de suporte também foi alterada, com a mudança o Laravel 6 passar a ser a versão LTS que contará com 2 anos de atualizações de novas funcionalidades e 3 anos de atualizações de segurança.

O Laravel 8 continuou as melhorias feitas na versão 7, suportando agora o Laravel JetStream, adicionando novas mudanças na Classe Factory, melhoria na queue, criação de componentes dinâmicos do Blade, criação de um novo recurso chamado de Migration Squashing para melhor organização das Migrations, criação do Job Batching que permite uma forma mais fácil de executar trabalhos em lote, entre diversas outras funcionalidades[5]

Laravel 9 foi lançado em 8 de fevereiro de 2022, trazendo uma mudança considerável no esquema de lançamento. A partir desta versão, o framework receberá uma versão a cada 1 ano. E com isso, cada versão terá suporte para correções de bugs por 18 meses, e correções de segurança por 2 anos. Deixando de lado a nomenclatura LTS.

O Laravel 9 trouxe diversas atualizações de pacotes da base do framework, como a troca do Swift Mailer, que não é mais mantido, para Symfony Mailer, atualização do Flysystem antes na versão 1, agora na versão mais atual (3). Também houve uma atualização no pacote Ignition, para a versão da Spatie.

Laravel 10 foi lançado em 14 de fevereiro de 2023. Esta versão atualizou completamente o esqueleto da aplicação e todos os stubs utilizados pelo framework para introduzir tipos de argumento e retorno em todas as assinaturas de método.

Além disso, houve diversas atualizações menores para produtividade, como novas funções str() e to_route() e maior suporte de Collections para IDEs.

Mas, mesmo com todas as novidades listadas acima, e muitas outras, o ponto chave é a versão do PHP que agora o mínimo é a versão 8.0.2 ou superior. Possibilitando o framework trazer diversas melhorias introduzidas nesta versão e posteriores. Inclusive Enums que não havia suporte nativo anteriormente. Apesar dessa funcionalidade só poder ser utilizada com a versão 8.1 do PHP.
Histórico de Versões[6]

Versões nomeadas de LTS possuem suporte a longo prazo, incluindo 2 anos de atualizações para resolver Bugs e 3 anos de atualizações de segurança. As outras versões incluem atualizações de Bugs por seis meses e de atualizações de segurança por 1 ano.

A partir da versão 9, o framework passa a ter os lançamentos anuais, e com isso cada versão passa a ter suporte para correções de segurança por 2 anos, resultando no fim da nomenclatura LTS para o framework, pois todas as versões terão suporte de longo prazo.

Versão	**Data de Lançamento**	**Versão do PHP**
1.0	Junho 2011	
2.0	Setembro 2011	
3.0	22 de Fevereiro 2012	
3.1	27 de Março, 2012	
3.2	22 de Maio, 2012	
4.0	28 de Maio, 2013	≥ 5.3.0
4.1	12 de Dezembro, 2013	≥ 5.3.0
4.2	1 de Junho, 2014	≥ 5.4.0
5.0	4 de Fevereiro, 2015	≥ 5.4.0
5.1 LTS	9 de julho, 2015	≥ 5.5.9
5.2	21 de Dezembro, 2015	≥ 5.5.9
5.3	23 de Agosto, 2016	≥ 5.6.4
5.4	24 de Janeiro, 2017	≥ 5.6.4
5.5 LTS	30 de Agosto, 2017	≥ 7.0.0
5.6	7 de Fevereiro, 2018	≥ 7.1.3
5.7	4 de Setembro, 2018	≥ 7.1.3
5.8	26 de Fevereiro, 2019	≥ 7.1.3
6 LTS	3 de Setembro, 2019	≥ 7.2.0
7	3 de Março, 2020	≥ 7.2.5
8	8 de Setembro, 2020	≥ 7.3.0
9	8 de Fevereiro, 2022	≥ 8.0.2
10	14 de Fevereiro, 2023	≥ 8.1.0

https://pt.wikipedia.org/wiki/Laravel
11
12

A Ascensão do Laravel: Da Origem à Fama

Laravel é um dos frameworks PHP mais populares e amplamente utilizados no mundo do desenvolvimento web. Criado por Taylor Otwell, este framework trouxe uma nova perspectiva para o desenvolvimento de aplicações web com PHP, combinando simplicidade, elegância e robustez em um único pacote. Neste artigo, exploraremos a história do Laravel, desde sua criação até como alcançou seu status atual de renome mundial.

A criação do Laravel

A história do Laravel começou em 2011, quando Taylor Otwell, um desenvolvedor web do Arkansas, Estados Unidos, decidiu criar um framework PHP mais simples e elegante. Ele estava insatisfeito com as opções disponíveis na época, como o CodeIgniter, que era popular mas possuía limitações e uma estrutura rígida.

Inspirado pelas filosofias do Ruby on Rails e do framework Sinatra, Taylor desenvolveu o Laravel com o objetivo de simplificar o desenvolvimento de aplicações web, facilitando a implementação de boas práticas e padrões de projeto. A primeira versão do Laravel foi lançada em junho de 2011 e recebeu elogios da comunidade PHP pela sua sintaxe clara, arquitetura bem organizada e ferramentas integradas.

A evolução do Laravel

Desde sua primeira versão, o Laravel passou por várias atualizações significativas, cada uma trazendo melhorias e novas funcionalidades. Algumas das atualizações mais notáveis incluem:

Laravel 3 (Fevereiro de 2012): introdução do sistema de migração de banco de dados, suporte a eventos e a geração de tarefas agendadas (Artisan);

Laravel 4 (Maio de 2013): reestruturação completa do código-base, adoção do Composer para gerenciamento de dependências e a introdução de Eloquent, um ORM (Object-Relational Mapping) altamente eficiente;

Laravel 5 (Fevereiro de 2015): inclusão de diretivas Blade, melhorias na estrutura de diretórios, suporte a Middleware e a introdução do Laravel Elixir, uma ferramenta para compilação de assets;

A popularização do Laravel

O Laravel ganhou popularidade rapidamente devido a uma combinação de fatores, entre eles:

Comunidade: a comunidade Laravel é incrivelmente ativa e acolhedora. Desde o início, houve um grande envolvimento dos desenvolvedores, que contribuíram com código, documentação e suporte. Além disso, foram criadas várias conferências e eventos específicos do Laravel, como o Laracon, que ajudaram a divulgar o framework e criar uma comunidade forte e unida.

Documentação e recursos educacionais: o Laravel é conhecido por sua documentação clara e completa, que facilita a aprendizagem e adoção do framework. Além disso, a quantidade de recursos educacionais disponíveis, como cursos online, tutoriais e livros, contribuiu para a popularização do Laravel.

Flexibilidade e desempenho: o Laravel oferece um equilíbrio perfeito entre flexibilidade e desempenho. Sua arquitetura permite aos desenvolvedores criar aplicações personalizadas e escaláveis com facilidade, e seu desempenho é comparável aos principais frameworks PHP disponíveis. Essa combinação atraiu desenvolvedores que buscavam uma solução eficiente e personalizável para suas necessidades.

Laravel hoje: reconhecimento e adoção

Hoje, o Laravel é amplamente considerado um dos melhores frameworks PHP para o desenvolvimento web. Ele é usado por empresas de todos os portes, desde startups até grandes corporações, para construir uma variedade de aplicações web, incluindo sites, sistemas de gerenciamento de conteúdo, aplicativos de comércio eletrônico e APIs.

Além disso, o Laravel tem sido reconhecido em várias ocasiões por sua excelência no design e na experiência do desenvolvedor, recebendo prêmios e honrarias em eventos e competições da indústria.

Conclusão

A trajetória do Laravel é um exemplo impressionante de como a visão e o trabalho de um desenvolvedor podem revolucionar a forma como os profissionais da área abordam a criação de aplicações web. Desde sua criação em 2011, o Laravel evoluiu constantemente, acompanhando as necessidades e demandas do mercado. A comunidade ativa, a documentação e os recursos educacionais de qualidade, e a flexibilidade e desempenho oferecidos pelo framework contribuíram para sua rápida ascensão e reconhecimento mundial.

Em resumo, o Laravel transformou o cenário do desenvolvimento PHP, oferecendo uma solução elegante, robusta e fácil de usar, e se tornou uma referência no mundo da tecnologia. Com sua crescente popularidade e adoção, o Laravel continua a impactar o desenvolvimento web e a vida de desenvolvedores em todo o mundo.

https://dolutech.com/a-ascensao-do-laravel-da-origem-a-fama/

Instalação

Existem várias maneiras de se instalar o Laravel.

Uma bem confortável é instalando o installer

Lembrando que esta é bem rápida porque armazena em cache os arquivos e os utiliza nas próximas vezes. Isso acarreta que assim ele não traz as novidades recentes.

composer global require laravel/installer

laravel new example-app

new [--dev] [--git] [--branch BRANCH] [--github [GITHUB]] [--organization ORGANIZATION] [--database DATABASE] [--stack [STACK]] [--breeze] [--jet] [--dark] [--typescript] [--ssr] [--api] [--teams] [--verification] [--pest] [--phpunit] [--prompt-breeze] [--prompt-jetstream] [-f|--force] [--] <name>

Usando alguns parâmetros

laravel new --git --database mysql --pest -f testelar

laravel new --database mysql --prompt-breeze --pest -f testelar

laravel new --git --database mysql --pest --breeze --stack blade --dark -f teste

alias laravel="/home/ribafs/.config/composer/vendor/bin/laravel"

laravel	**php (requer)**
11	- 8.2
10	- 81, 8.2
9	- 8.0, 8.2
8	- 7.3, 8.1

Instalando uma versão específica

composer create-project --prefer-dist laravel/laravel laravel11

composer create-project --prefer-dist laravel/laravel:^10.0 laravel10

composer create-project --prefer-dist laravel/laravel:^9.0 laravel9

composer create-project --prefer-dist laravel/laravel:^8.0 laravel8

Convenções no Laravel

Ao seguirmos as convenções definidas por um framework estamos garantindo boa performance, facilidade de manutenção, baixa curva de aprendizagem.

Ao não utilizar a convenção de nomeclatura 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** - no plural e tudo em minúsculas (snake case, products, my_products)
- **Model** - singular e inicial maiúscula (CamelCase, Product)
- **Controller** - singular e inicial maiúscula, CamelCase com sufixo Controller: ProductPcontroller
- **Métodos**/actions do Controller - todas os métodos em minúsculas
- **Views** - mesmos nomes das actions correspondentes, com pasta em minúsculas. products e index.blade.php
- Usa por padrão os timestamps: created_at e updated_at
- **PrimaryKey** - id (pode ser alterado no model)
- **Foreign Key** - tabela_id. Ex: product_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
- **Nomes das views básicas**: index.blade.php, create.blade.php, edit.blade.php e show.blade.php

Caso não queira seguir as convenções
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";

  //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.

Routes

resources no plural e minúsculas

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

A partir da versão 8 do Laravel ele não mais encontra o controller para nós e precisamos especificar, assim:

Route::resource('/products', 'App\Http\Controllers\ProductsController');

https://laravel.com/docs/7.x/eloquent#eloquent-model-conventions
https://laravel.com/docs/11.x/eloquent#eloquent-model-conventions
https://veesworld.com/laravel/laravel-naming-conventions

Configurações

As configurações básicas do Laravel se concentram no arquivo .env.

Quando instalamos o laravel ele cria o .env já com conteúdo básico.
Quando baixamos um exemplo em laravel geralmente o .env é removido, mas ainda assim vem uma cópia dele com nome .env.example. No caso renomeamos para .env e trabalhamos.

Vejamos parate do conteúdo do .env

```
APP_NAME='Diário do Ribamar'
APP_DEBUG=true
APP_TIMEZONE=America/Fortaleza
DB_CONNECTION=mariadb
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=diario
DB_USERNAME=root
DB_PASSWORD=
...
```

O laravel também tem outros arquivos de configuração, a maioria fica na pasta config, no raiz.

O Laravel trabalha com a filosofia do convenções sobre configurações, por isso se configura bem pouco em uma configuração. Para que ele nos ajude ao máximo precisamos conhecer e seguir suas convenções.

Novidades da versão 11

Obrigatório - PHP 8.2

Configuração simplificada

Laravel 11 consolida as configurações em uma única fonte: o arquivo .env. Esta é uma diferença notável em relação às versões anteriores do Laravel, que distribuíam as configurações em vários arquivos.

Remoção do Kernel

O Laravel 11 remove o kernel do Laravel. Em vez disso, você usa a classe Bootstrap/App, que vincula interfaces essenciais ao contêiner. Depois de configurar os componentes necessários, a classe Bootstrap/App retorna a instância do aplicativo. Essa separação entre a criação de instâncias e a execução do aplicativo e o tratamento de respostas proporciona uma arquitetura mais modular.

A remoção do kernel do Laravel é outra etapa para minimizar o código boilerplate. Como a classe Bootstrap/App cuida da configuração essencial, você pode se concentrar na criação das principais funcionalidades dos seus aplicativos. Quando uma instância do aplicativo é executada, a classe Bootstrap/App garante o tratamento necessário das solicitações web, simplificando o processo de desenvolvimento e promovendo uma base de código mais organizada.

Estrutura de diretórios

A remoção de determinados diretórios, como

app/Exceptions
app/Http/Middleware

Se a **personalização de middleware** for necessária, isso pode ser feito através do

app\Providers/AppServiceProvider.php

Aprimorando a Experiência de Depuração com o Novo Trait Dumpable

O Trait Dumpable é uma valiosa adição ao Laravel 11. Ele permite o uso de ferramentas de depuração familiares dentro das classes, incluindo os métodos dd() e dump(). Esses métodos são instrumentais durante a depuração, fornecendo visualizações instantâneas do objeto ou variável em questão.

Início básico de um controller no laravel 11

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

use App\Models\User;
use Illuminate\View\View;
...
    public function show(string $id): View
    {
        return view('user.profile', [
            'user' => User::findOrFail($id)
        ]);
    }
...
```

No controller agora incluimos

```
use Illuminate\View\View;
```

E nos métodos

Os métodos abaixo, que chamam uma view são do tipo View

```
index(): View
create(): View
show(Product $product): View
edit(Product $product): View
```

Demais métodos são do tipo : RedirectResponse

```
store(Request $request): RedirectResponse
update(Request $request, Product $product): RedirectResponse
destroy(Product $product): RedirectResponse
```

Obs: O laravel 11 continua aceitando quando criamos controllers sem seguir estas exigências, mas é bom lembrar delas na criação.

O projeto de CRUD para laravel 11 segue estas regras.

1 - Dicas

1.1 - Sobre as versões do laravel

Como a cada nova versão do laravel podemos ter várias mudanças, quando você procurar algum tutorial ou algum exemplo para estudar e/ou testar, precisa ficar atento para a versão que você tem instalada em seu desktop ou servidor e a versão do tutorial ou exemplo. Assim não correrá o risco de perder tempo testando algo que não é compatível com sua versão. Um indicativo de usar várias versões é o composer.json ter uma data mais recente que os demais arquivos.

Cada versão tem sua data de lançamento como pode ser visto aqui e isso o ajudará a ter uma ideia da versão apenas pela data dos arquivos.

https://laravel.com/docs/master/releases

Version	PHP (*)	Release	Bug Fixes Until	Security Fixes Until
9	8.0 - 8.2	February 8th, 2022	August 8th, 2023	February 6th, 2024
10	8.1 - 8.3	February 14th, 2023	August 6th, 2024	February 4th, 2025
11	8.2 - 8.3	March 12th, 2024	September 3rd, 2025	March 12th, 2026
12	8.2 - 8.3	Q1 2025	Q3, 2026	Q1, 2027

Alguns exemplos que encontramos são compatíveis com várias versões do laravel ou PHP, mas em geral apenas suportam uma versão.

1.2 - Abrir certa view por padrão ao executar artisan serve

```
use App\Http\Controllers\DiariosController;

Route::resource('diarios', DiariosController::class);

Route::get('/', function () {
    return redirect()->route('diarios.index');
});
```

1.3 - Adicionar CKEditor 5 para campos textarea

Adicionar ao resources/views/layouts/app-blade.php

```
<script src="https://cdn.ckeditor.com/ckeditor5/34.0.0/classic/ckeditor.js"></script>
```

Add para as views create e edit

```
    </div>
        <style>
        .ck-editor__editable_inline {
                min-height: 150px;
                font-size: 18px;
                line-height: 0.8;
        }
        </style>

    <script>
       // Initialize CKEditor
       ClassicEditor
          .create(document.querySelector('textarea'));
          //https://medium.com/@tutsmake.com/laravel-11-integrate-and-use-ckeditor-5-tutorial-
ceac8e1328ac
    </script>
x'@endsection
```

1.4 - Interceptar mensagem de erro de registro duplicado

Controller - store()

```
    public function store(Request $request)
    {
       $requestData = $request->all();
                try {
                        Diario::create($requestData);
                } catch (PDOException $e) {
                        $existingkey = "Integrity constraint violation: 1062 Duplicate entry";
                        if (strpos($e->getMessage(), $existingkey) !== FALSE) {
                                return redirect('diarios')->with('danger','Este dia já foi cadastrado.');

                        } else {
                                throw $e;
                        }
                }
       return redirect('diarios')->with('duplicate', 'Diario added!');
    }
```

View index

```
<div class="col-md-12">
   <div class="card">
      @session('duplicate')
         <div class="alert alert-danger" role="alert">
            {{ $value }}
         </div>
      @endsession
```

1.5 – Notificações

Vamos enviar uma notificação do controller para a view index sempre que tentar cadastrar um dia que já existe

Controller - store()

```
   public function store(Request $request)
   {
      $requestData = $request->all();
try {
   Diario::create($requestData);
} catch (PDOException $e) {
         $existingkey = "Integrity constraint violation: 1062 Duplicate entry";
   if (strpos($e->getMessage(), $existingkey) !== FALSE) {
                  return redirect('diarios')->with('danger','Este dia já foi cadastrado.');
   } else {
      throw $e;
   }
}

      return redirect('diarios')->with('flash_message', 'Diario added!');
   }
```

View index

```
<div class="col-md-12">
   <div class="card">
      @session('danger')
         <div class="alert alert-danger" role="alert">
            {{ $value }}
         </div>
      @endsession
```

1.6 - Collatioon no .env

Laravel 11 as vezes mostra erro por falta do collatioon no .env

Collation no.env

DB_COLLATION=utf8mb4_unicode_ci

1.7 – Corrigir Erro de permissão do storage

php artisan cache:clear
chmod -R 777 storage/
composer dump-autoload
php artisan route:clear

1.8 - Mostrar data numa rota

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

1.9 – Executando tags HTML em view

Para que seja executada, o campo precisa vir com

{!! $nomecampo !!}

1.10 – Mudando campo input em select

Tenho duas tabelas relacionadas posts e comments, um para vários. Posts tem os campos id e name.
Comments tem oo campo post_id que as relaciona
Quando cadastro um comentário, tenho o campo tipo caixa de texto post_id
Quero transformar p campo post_id em um select preenchido com os registros de posts, armazene post_id e mostre name.

No controller PostController alterei os métodos create() e edit() adicionando:

Adicionei ao início para que tivesse de comments acesso aos posts:
use App\Models\Post;

Mudei o create() assim:

```
public function create()
{
      $posts = Post::all();
   return view('comments.create', compact('posts'));
}
```

O edit assim:

```
public function edit($id)
{
      $posts = Post::all();
   $comment = Comment::findOrFail($id);
   return view('comments.edit', compact('comment', 'posts'));
}
```

O form-blade.php de comments

Removi o campo post_id que era um input e adicionei:

```
<select name="post_id" class="form-control" id="post_id" >
            @foreach ($posts as $key => $post)
      <option value="{{ $post->id }}" {{ (isset($comment->post_id) && $comment->post_id == $post->id) ? 'selected' : ''}}>{{ $post->name }}</option>
            @endforeach
      </select>
```

Isso resolveu para create e para edit

1.11 – Trocar id por name na view index

Na view index quero trocar post_id por post-name

Adicionei ao início do controller Comment
use Illuminate\Support\Facades\DB;

No mérodo index() do controller

O original era este criado pelo gerador

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

$comments = DB::table('comments')
->join('posts', 'posts.id', '=', 'comments.post_id')
->select('comments.*','posts.name')
->latest()->paginate(5);
```

Ne view index de comments

```
@foreach($comments as $item)
   <tr>
      <td>{{ $item->id }}</td>
            <td>{{ $item->name }}</td>
      <td>{{ $item->comment }}</td>
```

1.12 - Encurtar string mostrada na view index apenas para exibição

Encurtar tamanho de string mostrada na view index apenas para exibição

<td>{{ \Illuminate\Support\Str::limit($item->texto, 40, preserveWords: true) }}</td>
https://laravel.com/docs/11.x/strings#method-str-limit

1.13 - Limpara cache do laravel

php artisan cache:clear;php artisan config:cache;php artisan route:cache;php artisan optimize

Me parece que em produção não devemos estara limpando todos os caches assim.

2 – Projetos com Laravel 11

2.1 - Controle de estoque simplificado

Este é o projeto que considero mais importante daqui. Não pelo tipo de projeto, mas pelo código que aprendi sobre laravel ao criar.

Objetivo

O objetivo aqui não é o de entregar um sistema funcional, visto que o controle de estoque é um sistema bem grande, mas apenas mostrar algumas funcionalidades e dar uma ideia para que os interessados possam concluir o sistema. O sistema tem muito ainda por fazer para que fique funcional, mas estou compartilhando o que considero a ideia principal, que é a de efetuar certas operações via código, como a de adicionar ao estoque sempre que comprar e abater do estoque sempre que vender.

Codificação

A tabela estoques somente será cadastrada indiretamente, via código e somente nas operações create nas tabelas compras e vendas.

- Quando for inserido um registro em compras, via código o mesmo registro será inserido em estoques
- Quando for inserido um registro em vendas, via código será removido o respectivo registro de estoques

Simplificação do sistema

Para simplificação as tabelas compras e vendas somente terão as views index e create. Pois se usarmos update e delete nelas teremos também que ajustar o estoque, mas para este exemplo ficaremos apenas com index e create.

- A tabela produtos contará com um CRUD completo
- Compras e vendas terão somente index, create e search
- Estoques terá somente index e search

A criação inicial dos CRUDs será feita com o gerador de CRUDs abaixo

composer require ribafs/crud-generator --dev

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

php artisan crud:generate Products --fields='name#string; inventory_min#integer; inventory_max#integer;' --controller-namespace=App\\Http\\Controllers --form-helper=html

php artisan crud:generate Buys --fields='product_id#integer; quantity#integer; price#decimal; date#date;' --controller-namespace=App\\Http\\Controllers --form-helper=html

php artisan crud:generate Inventories --fields='product_id#integer; quantity#integer;' --controller-namespace=App\\Http\\Controllers --form-helper=html

php artisan crud:generate Sales --fields='product_id#integer; quantity#integer; price#decimal; date#date;' --controller-namespace=App\\Http\\Controllers --form-helper=html

Rotas

```
Route::resource('/products', 'App\Http\Controllers\ProductsController');
Route::resource('/inventories', 'App\Http\Controllers\InventoriesController');
Route::resource('/buys', 'App\Http\Controllers\BuysController');
Route::resource('/sales', 'App\Http\Controllers\SalesController');
```

Executar
php artisan route:clear

Seeder

php artisan make:seeder ProductsSeeder

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\Product;

class ProductsSeeder extends Seeder
{
    public function run(): void
    {
                Product::create([
                    'name' => 'Bananas',
                    'inventory_min' => 30,
                    'inventory_max' => 200,
                ]);
    }
}
```

php artisan db:seed --class=ProductsSeeder

Rotinas

- Antes de efetuar uma compra, antes de inserir no banco, checar:
  - o estoque_maximo em produtos do respectivo produto
  - e somente permitir a compra se a quantidade em estoque somada com a quantida que se pretende comprar, somar um valor menor ou igual ao estoque_maximo.

- Antes de efetuar uma venda, checar o estoque_minimo para o produto que se pretende vender.
  - Somente vender se existir quantidade suficiente em estoque.

- Caso, após a venda, a quantidade final do produto em estoque for igual ou inferior a quantidade do estoque_minimo em produtos
- então será disparado um aviso (usando o SwitchAlert) avisando para que o estoque seja reposto.

Obs.: num sistema em produção também devemos verificar a data de compra e a data de venda e nunca permitir que a data de venda de um produto seja menor que sua data de compra. Talvez inserir a data do sistema ao vender e ao comprar.

O preco de venda também precisa ser maior que o de compra. Idealmente estima-se um percentual de lucro e o sistema já faz isso automaticamente.

Vamos efetuar alguns ajustes iniciais

- Em compras o campo product_id deve mostrar uma combo com os produtos de products. Em vendas também.

Ver dicas sobre como implementar a troca de um input por um select

Inicialmente criarei um seeder que cadastra dois produtos, banana e manga.

Compras

- Quando for inserido um registro em compras, via código o mesmo registro será inserido em estoques

Como será isso via código?

Inicialmente o estoque está zerado

- Vamos comprar banabas (o único produto em produtos)
Clica em Compras - Add New

- A compra:
    - adiciona a quantidade do produto comprado ao atual de compras (Esta o laravel já faz por default)
    - Soma a quantidade comprada do produto para o estoque. Paraf azer isso precisarei:
    - Não vou considerar aqui, mas ao comprar, antes devemos verificar intentory_max, para ter certeza que tem
     onde armazenar

    - Adicionar ao controller Buys
        use App\Models\Inventory;
        No método store capturar o id do product_id e a quantity e somar ao inventories
        Vou verificar o que o store está recebendo, antes de enviar para o model:
            dd($requestData); Recebe todos os valores de compras

```
// Enviar a compra para o estoque
$inventories  = new Inventory();
$inventories->product_id = $request->product_id;
```

```
$inventories->quantity = $request->quantity;
$inventories->save();
```

Beleza. Salva em compras e também no estoque.

Também podemos fazer assim:

```
$requestData = $request->all();
Inventory::create($requestData);
```

Vendas

- Quando for inserido um registro em vendas, via código o mesmo product_id terá sua quantity reduzida no estoques

Inicialmente o estoque está zerado, então antes de reduzir precisamos verificar se sua quantity é > 0 e reduzir somente se sim

- Vamos vender banabas (o único produto cadastrado em produtos até o momento)
Clica em Vendas - Add New

- A venda:
    - abate a quantidade do produto vendido ao atual de vendas
    - Não vou considerar aqui, mas antes de efetuar uma venda, precisamos verificar inventory_min em products
    - Abate a quantidade vendida do produto no estoque. Paraf azer isso precisarei:
    Existem 100 bananas no estoque e vou vender 10

    - Adicionar ao controller Buys

```
use App\Models\Inventory; // Já fiz na compras
use Illuminate\Support\Facades\DB;

$requestData = $request->all();

    // Abater a quantidade vendida do estoque
$inventories = DB::table('inventories')->get();
foreach($inventories as $inventory){
    if($inventory->product_id == $request->product_id){
        $inventory->quantity = $inventory->quantity - $request->quantity;
                $inventories = DB::table('inventories')
                                ->where('product_id', $request->product_id)
                                ->update(['quantity' => $inventory->quantity]);

    }
}
Sale::create($requestData);
...
```

Beleza. Salva em vendas e abate no estoque.

Melhor código

```
$inventories = Inventory::find($request->product_id);
$inventories->quantity = $inventories->quantity - $request->quantity;
$inventories->save();
```

Migrations

As 4 tabelas serão relacionadas

```
Schema::create('products', function (Blueprint $table) {
   $table->id();
   $table->string('name')->notNllable();
   $table->integer('inventory_min')->notNullable();
   $table->integer('inventory_max')->notNullable();
   $table->timestamps();
});

Schema::create('buys', function (Blueprint $table) {
   $table->increments('id');
                $table->unsignedBigInteger('product_id');
   $table->integer('quantity')->notNullable();
   $table->decimal('price')->notNullable();
   $table->date('date')->nullable();
   $table->timestamps();

  $table->foreign('product_id')->references('id')->on('products')->onUpdate('cascade')-
>onDeletee('cascade');
});

Schema::create('inventories', function (Blueprint $table) {
   $table->increments('id');
                $table->unsignedBigInteger('product_id');
   $table->integer('quantity')->notNullable();
   $table->timestamps();

  $table->foreign('product_id')->references('id')->on('products')->onUpdate('cascade')-
>onDeletee('cascade');
});

Schema::create('sales', function (Blueprint $table) {
   $table->increments('id');
                $table->unsignedBigInteger('product_id');
   $table->integer('quantity')->notNullable();
   $table->decimal('price')->notNullable();
   $table->date('date')->nullable();
   $table->timestamps();

  $table->foreign('product_id')->references('id')->on('products')->onUpdate('cascade')-
>onDeletee('cascade');
```

});

Sequência

Lembrando que o cadastro de estoques/inventories será feito somente via código.
Quando eu finalizar estarei removendo os botões editar e deletar.

- Products já vem com um registro de bananas
- Adicionar 100 bananas em compras/buys.
- Então verifique Estoque que já mostrará as 100 compradas
- Cadastre 10 em vendas para bananas
- Verifique que o estoque foi reduzido de 10

Dicas sobre Controle de estoque

O estoque é um setor que exige muita atenção por parte das empresas, pois além de ser responsável pelo abastecimento interno, também é o fator que impacta nas finanças do negócio de forma positiva ou negativa, dependendo de como está sendo controlado.

A gestão de estoque é essencial para a redução de custos na empresa, bem como o aumento da lucratividade.

O gerenciamento de estoque é parte essencial no atendimento ao cliente, pois um bom controle dessa área evita possíveis frustrações, como permitir a compra produtos que não estão mais disponíveis.

É graças ao gerenciamento de estoque que a empresa mantém os níveis do setor sempre otimizado, para que não tenha problemas com o excesso de produtos, bem como a falta deles.

Quando o empreendimento ainda está no início, a quantidade de mercadorias para controle permite que a gestão seja feita utilizando cadernos e planilhas. Mas acontece que o crescimento repentino do negócio pode complicar um pouco esse processo.

Organize as informações de produtos e de fornecedores

Crie e envie pedidos de compra exatos

Receba seus pedidos com precisão

Identifique suas mercadorias

Faça Inventários periódicos

Organize seu espaço físico

O controle de estoque é usado para mostrar a quantidade de produtos que sua empresa tem, bem como a variedade de produtos disponíveis para venda ou para uso na prestação de serviços ou produção (dependendo de qual negócio você tem).

A papelada típica que deverá ser processada no controle de estoque é:

- Notas de entrega e fornecimento das mercadorias (entrada de produtos);
- Ordens de compra, recibos e notas fiscais
- Notas de retorno/devolução
- Requisições e notas para saída de produtos

Manter o controle de estoque em dia é de extrema importância para uma empresa apurar o seu movimento de entrada e saída de mercadorias. E, assim, ter informações precisas sobre a demanda que o mercado tem de seu produto, eventuais desvios e também sobre o lucro líquido obtido no final do mês contábil.

Ter um estoque controlado é saber que há a quantidade correta de produtos para que a empresa possa fluir corretamente e atender sua demanda do mercado, sem ter prejuízos com perdas.

O estoque pode ser, geralmente, de duas modalidades diferentes. Uma é de matérias-primas para a produção industrial de grande ou pequeno porte. A outra, de produtos finais prontos para a comercialização no varejo.

Sem controle de estoque eficiente, sua empresa pode ter dificuldade em identificar produtos que estão em falta ou baixa quantidade. Isso pode levar a perda de possíveis vendas. Já que, neste caso, seus potenciais clientes poderão procurar outra empresa que possua o produto desejado.

É péssimo para o vendedor conseguir fechar uma venda difícil e descobrir que não há o produto no estoque. E pior que ele pode demorar a chegar. A frustração não é só do vendedor, mas também do cliente que acaba procurando a concorrência.

O controle de vendas indica quais os produtos com maior procura. E até em que momento do mês há mais vendas sobre ele. Também fala quais os que têm pouca saída e os que são sazonais. Ou seja, os que vendem somente em determinada época. Através dessas observações, é possível compreender as estratégias de vendas desses produtos e como o consumidor tem reagido a elas. Assim como o que pode ser feito no futuro.

Muitos departamentos financeiros de empresas que não possuem um controle de estoque equilibrado e fazem compras em demasia de produtos, sem saber sua real necessidade ou mesmo deixam de comprar os que saem mais, acabam demorando para recuperar o capital investido e também perdem possíveis oportunidades de negócio.

Há ainda a possibilidade de armazenar muitos produtos que fiquem obsoletos ou que possam perder a data de vencimento e sejam desperdiçados. Essas questões são péssimas para a saúde financeira da empresa e são evitadas com um controle definido de estoque.

Com o estoque devidamente administrado, a produtividade aumenta, os custos diminuem, as perdas são diminuídas ou erradicadas e o capital de giro pode ser investido em outros recursos dentro da própria empresa ou em aplicações financeiras.

Fisicamente, utilize métodos para que seja fácil saber onde localizá-lo. Como colocar os produtos disponibilizados em ordem alfabética ou numérica, devidamente identificados em sua embalagem e no sistema.

É importante também verificar a data de validade do produto, mantendo os com data de vencimento mais próxima à frente dos outros. Isso evita que os produtos adquiridos mais recentes sejam vendidos e os mais antigos fiquem para trás. Fazendo com que percam a sua hora certa de venda.

Planeje compras fazendo projeções de demanda

Considere os prazos de entrega no planejamento

Escolha os fornecedores que melhor te atendam

Tenha ajuda de uma consultoria

Cadastre os seus produtos

Faça um inventário de estoque

Fique atento aos indicadores de resultados

Integre os setores de sua empresa

Categorize os seus produtos

Conte com ferramentas de gestão para controle de estoque

Organização e otimização do espaço

Integração com os setores de compras e vendas

Referências

https://blog.vhsys.com.br/sistema-de-controle-estoque/
https://blog.quantosobra.com.br/controle-de-estoque/
https://blog.egestor.com.br/controle-de-estoque/
https://blog.eficienciafiscal.com.br/controle-de-estoque/

Código extra
Pode ser que eu não tenha sido didático o suficiente para falar de como implementei o código, especialmente o cadastro na tabela de estoques ao mesmo tempo que na compra. Também abater em estoque ao mesmo tempo em que se vende. Mas se você tem a base necessária verá isso facilmente nos controllers Buys e Sales.

2.2 - Diário com bons recursos

Este projeto inicialmente eu queria criar usando uma linguagem tipo desktop, mas deu trabalho de encontrar e me deu tempo de refletir que eu tenho grande experiência com linguagem web, PHP e Laravel e que mantenho um servidor web ativo em meu computador. Então resolvi fazer o diário usando laravel.

Criar um aplicativo em Laravel 11
Usar o gerador de CRUDs
https://github.com/ribafs2/gerador-cruds

Para gerar o CRUD diarios
id, dia, texto

Ao abrir já deve abrir na view diarios.index

Adicionar o editor CKEditor para o campo textarea nas views create e edit.
O campo texto deve ter como valor default o dia de hoje e ser único
View index mostrar somente dia e texto reduzido

Criar script de backup para arquivos e banco
Usar o crontab para executar o script diariamente as 12 horas

```
cd /backup/usb/pessoais/diario
nano diario.sh
chmod +x  diario.sh
```

```
#!/bin/bash
zip -rq /backup/usb/pessoais/diario/diario_$(date +"%Y_%m_%d").zip /backup/usb/www/diario
mysqldump -uroot diario diarios > /backup/usb/pessoais/diario/diario_$(date +"%Y_%m_%d").sql
```

```
crontab -e
Adicionar ao final:
0 2 * * * /backup/usb/pessoais/diario/diario.sh
```

Testar com a cada minuto
```
* * * * * /backup/usb/pessoais/diario/diario.sh
```

Este backup torna desnecessário o pacote iseed

Ao final criar uma combinação de teclas para abrir o aplicativo no firefox, alias, apenas alterar a combinação atual que, ao invés de abrir o diário atual no LibreOffice irá abrir o aplicativo no firefox.

Existem diversos bons recursos que implementei e que não estão aqui, mas uma rápida análise no código e na execução do projeto podem mostrar.

2.3 - ACL com Spatie

Este aplicativo implementa o uso de ACL em aplicativos laravel 11 usando o famoso pacote de permissões Spatie Laravel.

O arquivo texto TUTORIAL.txt, no raiz do aplicativo, tem um resumo do tutorial oficial do criador em
https://www.allphptricks.com/laravel-11-spatie-user-roles-and-permissions/

2.4 – Autenticação

Muitos aplicativos da web fornecem uma maneira para seus usuários autenticarem com o aplicativo e "fazerem login". Implementar esse recurso em aplicativos da 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 forma rápida, segura e fácil.

Em seu núcleo, os recursos de autenticação do Laravel são compostos de "guards" e "providers". Os guardas definem como os usuários são autenticados para cada solicitação. Por exemplo, o Laravel vem com um guard de sessão que mantém o estado usando armazenamento de sessão e cookies.

Os providers/provedores definem como os usuários são recuperados do seu armazenamento persistente. O Laravel vem com suporte para recuperar usuários usando o Eloquent e o query builder/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.

Guards e provedores não devem ser confundidos com "roles/funções" e "permissões". Para saber mais sobre como autorizar ações do usuário por meio de permissões, consulte a documentação de autorização.

Starter Kits

Quer começar rápido? Instale um starter kit de aplicativo Laravel em um novo aplicativo Laravel. Após migrar seu banco de dados, navegue em seu navegador para /register ou qualquer outra URL que esteja atribuída ao seu aplicativo. Os starter kits cuidarão de todo o seu sistema de autenticação!

Mesmo se você optar por não usar um starter kit em seu aplicativo Laravel final, instalar o starter kit Laravel Breeze pode ser uma oportunidade maravilhosa para aprender como implementar todas as funcionalidades de autenticação do Laravel em um projeto Laravel real. Como o Laravel Breeze cria controladores de autenticação, rotas e visualizações para você, você pode examinar o código dentro desses arquivos para aprender como os recursos de autenticação do Laravel podem ser implementados.

Considerações sobre bancos de dados

Ao construir o esquema do banco de dados para o modelo App\Models\User, certifique-se de que a coluna password tenha pelo menos 60 caracteres de comprimento. Claro, a migração da tabela users que está incluída em novos aplicativos Laravel já cria uma coluna que excede esse comprimento.

Além disso, você deve verificar se sua tabela users (ou equivalente) contém uma coluna string remember_token nullable de 100 caracteres. Esta coluna será usada para armazenar um token para usuários que selecionarem a opção "lembrar de mim" ao efetuar login em seu aplicativo. Novamente, a migração padrão da tabela users que está incluída em novos aplicativos Laravel já contém esta coluna.

Visão geral do ecossistema

O Laravel oferece vários pacotes relacionados à autenticação. Antes de continuar, revisaremos o ecossistema geral de autenticação no Laravel e discutiremos a finalidade pretendida de cada pacote.

Primeiro, considere como a autenticação funciona. Ao usar um navegador da web, um usuário fornecerá seu nome de usuário e senha por meio de um formulário de login. Se essas credenciais estiverem corretas, o aplicativo armazenará informações sobre o usuário autenticado na sessão do usuário. Um cookie emitido para o navegador contém o ID da sessão para que solicitações subsequentes ao aplicativo possam associar o usuário à sessão correta. Após o cookie da sessão ser recebido, o aplicativo recuperará os dados da sessão com base no ID da sessão, observe que as informações de autenticação foram armazenadas na sessão e considerará o usuário como "autenticado".

Quando um serviço remoto precisa se autenticar para acessar uma API, os cookies normalmente não são usados para autenticação porque não há um navegador da web. Em vez disso, o serviço remoto envia um token de API para a API em cada solicitação. O aplicativo pode validar o token de entrada em uma tabela de tokens de API válidos e "autenticar" a solicitação como sendo realizada pelo usuário associado a esse token de API.

Pacotes de autenticação

- Laravel Breeze - https://laravel.com/docs/11.x/starter-kits#laravel-breeze
- Laravel Jetstream - https://laravel.com/docs/11.x/starter-kits#laravel-jetstream
- Laravel Fortify - https://laravel.com/docs/11.x/fortify
- Laravel UI - https://github.com/laravel/ui

O Laravel Breeze é uma implementação simples e mínima de todos os recursos de autenticação do Laravel, incluindo login, registro, redefinição de senha, verificação de e-mail e confirmação de senha. A camada de visualização do Laravel Breeze é composta de modelos Blade simples estilizados com Tailwind CSS. Para começar, confira a documentação sobre os kits iniciais de aplicativos do Laravel.

O Laravel Fortify é um backend de autenticação headless para o Laravel que implementa muitos dos recursos encontrados nesta documentação, incluindo autenticação baseada em cookie, bem como outros recursos como autenticação de dois fatores e verificação de e-mail. O Fortify fornece o backend de autenticação para o Laravel Jetstream ou pode ser usado independentemente em combinação com o Laravel Sanctum para fornecer autenticação para um SPA que precisa autenticar com o Laravel.

O Laravel Jetstream é um robusto kit inicial de aplicativo que consome e expõe os serviços de autenticação do Laravel Fortify com uma UI bonita e moderna, alimentada por Tailwind CSS, Livewire e/ou Inertia. O Laravel Jetstream inclui suporte opcional para autenticação de dois fatores, suporte de equipe, gerenciamento de sessão do navegador, gerenciamento de perfil e integração interna com o Laravel Sanctum para oferecer autenticação de token de API. As ofertas de autenticação de API do Laravel são discutidas abaixo.

Laravel UI - Embora o Laravel não dite quais pré-processadores JavaScript ou CSS você usa, ele fornece um ponto de partida básico usando Bootstrap, React e/ou Vue que será útil para muitos aplicativos. Por padrão, o Laravel usa o NPM para instalar ambos os pacotes de frontend.

Este pacote legado é um scaffolding de autenticação muito simples construído na estrutura CSS do Bootstrap. Embora ele continue a funcionar com a versão mais recente do Laravel, você deve considerar usar o Laravel Breeze para novos projetos. Ou, para algo mais robusto, considere o Laravel Jetstream.

Recuperando o usuário autenticado

```
use Illuminate\Support\Facades\Auth;

// Retrieve the currently authenticated user...
$user = Auth::user();

// Retrieve the currently authenticated user's ID...
$id = Auth::id();
```

Redirecionando usuários não autenticados

```
use Illuminate\Http\Request;

->withMiddleware(function (Middleware $middleware) {
    $middleware->redirectGuestsTo('/login');

    // Using a closure...
    $middleware->redirectGuestsTo(fn (Request $request) => route('login'));
})
```

2.4.1 – Autenticação com laravel/ui e Bootstrap

Instalar o laravel

composer require laravel/ui

php artisan ui bootstrap --auth
yes

php artisan migrate

php artisan serve

Veja que já temos login e register e toda a estrutura por traz

2.4.2 - Com Breeze

Instalar o laravel

composer require laravel/breeze --dev

php artisan breeze:install

php artisan migrate
npm install
npm run dev

Breeze e Blade

A "pilha" padrão do Breeze é a pilha Blade, que utiliza modelos Blade simples para renderizar o frontend do seu aplicativo. A pilha Blade pode ser instalada invocando o comando breeze:install sem outros argumentos adicionais e selecionando a pilha frontend Blade.

php artisan serve

http://127.0.0.1:8000

Em seguida, você pode navegar para as URLs /login ou /register do seu aplicativo no seu navegador da web. Todas as rotas do Breeze são definidas dentro do arquivo routes/auth.php.

2.4.3 - Com Jetstream

composer create-project --prefer-dist laravel/laravel authjs

cd authjs

composer require laravel/jetstream

php artisan jetstream:install livewire

npm install
npm run dev

php artisan migrate

config/jetstream.php
....
```
'features' => [
        Features::profilePhotos(),
        Features::api(),
        Features::teams(),
    ],
```

...

php artisan serve

Assim temos a estrutura de login e registro

2.5 - Sistema de comentários

Um projeto que baixei da deste site
https://www.itsolutionstuff.com/post/laravel-57-comment-system-tutorial-from-scratchexample.html

Execute com

composer install

Ajuste o .env

php artisan migrate

php artisan serve

http://127.0.0.1:8000/register

Posts

2.6 - Usando cron jobs

Por que precisamos usar um cron job? E quais são os benefícios de usar cron jobs no Laravel 11 e como configurar cron jobs no Laravel 11? Se você tem essas perguntas, então eu explicarei o porquê. Muitas vezes precisamos enviar notificações ou e-mails automaticamente para usuários para atualizar propriedades ou produtos. Então, naquele momento, você pode definir alguma lógica básica para cada dia, hora, etc., para executar e enviar notificações por e-mail.

Aqui, eu darei a você um exemplo muito simples. Nós criaremos um comando cron job para obter usuários de uma API e criar novos usuários em nosso banco de dados. Nós definiremos tarefas para serem feitas automaticamente a cada minuto. Você pode escrever sua própria lógica no comando. Eu também mostrarei como configurar um cron job no servidor com o Laravel 11. Então, vamos seguir os passos abaixo para fazer este exemplo.

composer create-project laravel/laravel cron-job

cd cron-job

php artisan make:command DemoCron --command=demo:cron

Ajustes

app/Console/Commands/DemoCron.php

```
<?php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use Illuminate\Support\Facades\Http;
use App\Models\User;

class DemoCron extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'demo:cron';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Execute the console command.
```

```
     */
    public function handle()
    {
        info("Cron Job running at ". now());

        /*-----------------------------------------
        -----------------------------------------
        Write Your Logic Here....
        I am getting users and create new users if not exist....
        -----------------------------------------
        -----------------------------------------*/
        $response = Http::get('https://jsonplaceholder.typicode.com/users');

        $users = $response->json();

        if (!empty($users)) {
            foreach ($users as $key => $user) {
                if(!User::where('email', $user['email'])->exists() ){
                    User::create([
                        'name' => $user['name'],
                        'email' => $user['email'],
                        'password' => bcrypt('123456789')
                    ]);
                }
            }
        }
    }
}
```

Cadastre-se como Agendador de Tarefas

Nesta etapa, precisamos definir nossos comandos no arquivo console.php com o horário em que você deseja executar seu comando, conforme mostrado nas funções a seguir:

->everyMinute(); Run the task every minute
->everyFiveMinutes(); Run the task every five minutes
->everyTenMinutes();Run the task every ten minutes
->everyFifteenMinutes(); Run the task every fifteen minutes
->everyThirtyMinutes(); Run the task every thirty minutes
->hourly(); Run the task every hour
->hourlyAt(17); Run the task every hour at 17 mins past the hour
->daily(); Run the task every day at midnight
->dailyAt(â€™13:00â€²); Run the task every day at 13:00
->twiceDaily(1, 13); Run the task daily at 1:00 & 13:00
->weekly(); Run the task every week
->weeklyOn(1, â€˜8:00â€™); Run the task every week on Tuesday at 8:00
->monthly(); Run the task every month
->monthlyOn(4, â€™15:00â€²); Run the task every month on the 4th at 15:00
->quarterly(); Run the task every quarter
->yearly(); Run the task every year

->timezone(â€˜America/New_Yorkâ€™); Set the timezone

routes/console.php

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

Schedule::command('demo:cron')->everyMinute();

Testar manualmente

php artisan schedule:run

2024-08-15 21:21:51 Running ['artisan' demo:cron] ...

Ver os logs em
cat storage/logs/laravel.log

Configurar no servidor

crontab -e

* * * * * cd /backup/usb/progecia/0Projetos/0SitesAppsCriacao/0Auxiliares/Laravel/0Laravel/0Projetos/cron-jobs & php artisan schedule:run >> /dev/null 2>&1

Créditos:
https://www.itsolutionstuff.com/post/laravel-11-cron-job-task-scheduling-tutorialexample.html

2.7 – CRUDs

2.7.1 - Passo a passo

Primeira etapa, criar uma migration, uma rota, um model, que serão a base de todas as fases e criaremos apenas uma vez

- Nesta fase criaremos um controller com um método index() e uma view, a index

Inicialmente criar

migration
model
rota

php artisan make:migration create_products_table
php artisan make:model Product

Route::resource('/products', 'App\Http\Controllers\ProductController');

Primeira fase

Criar o controller ProductController

php artisan make:controller ProductController

então criar método index()

Adicionar a view index.blade.php ao resources/views/products

php artisan make:view products.index

Testar

Ao abrir a index verá a listagem dos registros da tabela

Nossa próxima etapa será o create

Segunda fase

Criar o método create() no controller ProductController

Adicionar a view create.blade.php ao resources/views/products

Testar

Ao abrir a index clique no botão Add New

Veja quue ele abre o form, mas o create é apenas o form. Caso preencha os dados e clique em Create ele vai informar
Call to undefined method App\Http\Controllers\ProductController::store()

Ou seja, não encontrou o método store() que processaria os dados.

Nossa próxima etapa.

Terceira etapa

Criar o método store() no controller ProductController.php

Este método recebe os dados da view create e manda para o banco/model. Este não tem view.

Testar

Ao abrir a index clique no botão Add New

Preencha os dados e clique em Create
Veja que os dados aparecem na index, juntamente com os botões Edit e Delete.

Nossa próxima etapa é o edit.

Quarta etapa

Criar o método edit() no controller ProductController.php

Este método chama a view edit

Testar

Ao abrir a index clique no botão Edit na linha do registro existente

Ele abre o pqueno form com os dados. Altere algo e clique em Edit
Ele mostra
Call to undefined method App\Http\Controllers\ProductController::update()
Isso porque a view edit é apenas o form que chhama o método update() que nosso controller ainda não tem.

Vamos criar o update()

Quinta etapa

Criar o método update() no controller ProductController.php

Este método processa os dados da view edit.

Testar

Ao abrir a index clique no botão Edit na linha do registro existente

Ele abre o pqueno form com os dados. Altere algo e clique em Edit
Veja que ele já mostra na index com as alterações

Vamos criar o destroy()

Sexta etaapa

Criar o método destroy() no controller ProductController.php

Este método processa os dados da view index, quando clicamos em Delete.

Testar

Ao abrir a index clique no botão Delete na linha do registro existente
Veja que ele pede uma confirmação. Isso é algo que foi implementado pelo gerador que usei, não é padrão mas é desejável
Clique em Cancelar ou OK para testar

Veja que ele já mostra a index sem o registro

Assim concluímos nosso CRUD

2.7.2 - CRUD com Upload de imagem

Laravel 11 CRUD and Image Upload Application Tutorial

Build an Laravel 11 CRUD with Image Upload Step by Step example. you have to simply follow the below steps:

- Step 1: Install Laravel 11
- Step 2: MySQL Database Configuration
- Step 3: Create Migration
- Step 4: Create Controller and Model
- Step 5: Add Resource Route
- Step 6: Add Blade Files
- Run Laravel 11 App

https://www.itsolutionstuff.com/upload/laravel-11-crud-image

Run `php artisan serve` for a dev server.
Navigate to
http://localhost:8000/
The application will automatically reload if you change any of the source files.

2.7.3 – Gerador de CRUDs criado com commands

Outro bom material para estudo. Tem o código simples, inteiramente criado por mim. Por dois motivos é simples, porque é um dos meus princípios ao programar e porque meus conhecimentos não são tão elevados. Tem um tetrceiro motivo que esqueci, que sempre que faço algo tenho a intenção de repassar, então procuro a forma mais simples para me fazer entender.

CRUD generator - Laravel 8, 9, 10, 11

Testado nas versões 8, 9, 10 e 11
Em Wiindows 11 com Xampp e WSL2 e no Linux Debian 12

Criar aplicativo em laravel

composer create-project laravel/laravel crud
cd crud

Copiar a pasta Commands para app/Console
E executar:

php artisan crud:create products string#name,decimal#price

Ajustar a migration e o controller, caso queira e voltar para criar o CRUD.

Ajustar o banco no .env

Lembre que é coerente verificar o CreateSeeder antes de executar o comando abaixo para alguns ajustes, se for o caso.

AVISO: Fique atento aos tipos de dados dos campos, pois o command para o seeder tem limitações. Quando usar algum tipo que não esteja no switch precisará adicionar para que funcione. Aqui tem a relação de tipos de dados suportados pelo laravel 11 e que podem ser adaptados para uso com o fakerPHP:

https://laravel.com/docs/11.x/migrations#available-column-types
https://fakerphp.github.io/

php artisan migrate --seed

php artisan serve

http://127.0.0.1:8000

2.7.4 – Exemplo atualizado de CRUD para laravel 11

Este exemplo de CRUD segue as recomendações do laravel 11 bonitinho.

Simple Laravel 11 CRUD Application Tutorial

Learn how to develop a simple Laravel 11 CRUD application
Following the new rules/conventions of laravel 11

> The complete tutorial step by step guide is available on my blog. [Laravel 11 CRUD Application] (https://www.allphptricks.com/simple-laravel-11-crud-application-tutorial/)

Blog
https://www.allphptricks.com/

Installation
Make sure that you have setup the environment properly. You will need minimum PHP 8.2, MySQL/MariaDB, and composer.

1. Download the project (or clone using GIT)
2. Copy `.env.example` into `.env` and configure your database credentials
3. Go to the project's root directory using terminal window/command prompt
4. Run `composer install`
5. Set the application key by running `php artisan key:generate --ansi`
6. Run migrations `php artisan migrate`
7. Start local server by executing `php artisan serve`
8. Visit here http://127.0.0.1:8000/products to test the application

2.8 - Mensagens de erro de validações

O Laravel 11 fornece um objeto de request/solicitação para adicionar validação de formulário usando-o. Usaremos request validate() para adicionar regras de validação e mensagens personalizadas. Usaremos a variável $errors para exibir mensagens de erro.

composer create-project laravel/laravel errors-msg

cd errors-msg

php artisan make:controller FormController

app/Http/Controllers/FormController.php

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

use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\View\View;
use Illuminate\Http\RedirectResponse;

class FormController extends Controller
{
   /**
    * Show the application dashboard.
    *
    * @return \Illuminate\Http\Response
    */
   public function create(): View
   {
      return view('createUser');
   }

   /**
    * Show the application dashboard.
    *
    * @return \Illuminate\Http\Response
    */
   public function store(Request $request): RedirectResponse
   {
      $validatedData = $request->validate([
            'name' => 'required',
            'password' => 'required|min:5',
            'email' => 'required|email|unique:users'
         ], [
            'name.required' => 'Name field is required.',
            'password.required' => 'Password field is required.',
            'email.required' => 'Email field is required.',
```

```
            'email.email' => 'Email field must be email address.'
        ]);

        $validatedData['password'] = bcrypt($validatedData['password']);
        $user = User::create($validatedData);

        return back()->with('success', 'User created successfully.');
    }
}
```

routes/web.php

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

use App\Http\Controllers\FormController;

Route::get('users/create', [ FormController::class, 'create' ]);
Route::post('users/create', [ FormController::class, 'store' ])->name('users.store');
```

resources/views/createUser.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Laravel 11 Form Validation Example - ItSolutionStuff.com</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/
all.min.css" />
</head>
<body>
<div class="container">

    <div class="card mt-5">
        <h3 class="card-header p-3"><i class="fa fa-star"></i> Laravel 11 Form Validation
Example - ItSolutionStuff.com</h3>
        <div class="card-body">
            @session('success')
                <div class="alert alert-success" role="alert">
                    {{ $value }}
                </div>
            @endsession

            <!-- Way 1: Display All Error Messages -->
            @if ($errors->any())
                <div class="alert alert-danger">
                    <strong>Whoops!</strong> There were some problems with your input.<br><br>
                    <ul>
                        @foreach ($errors->all() as $error)
```

```
            <li>{{ $error }}</li>
        @endforeach
    </ul>
</div>
@endif

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

    {{ csrf_field() }}

    <div class="mb-3">
        <label class="form-label" for="inputName">Name:</label>
        <input
            type="text"
            name="name"
            id="inputName"
            class="form-control @error('name') is-invalid @enderror"
            placeholder="Name">

        <!-- Way 2: Display Error Message -->
        @error('name')
            <span class="text-danger">{{ $message }}</span>
        @enderror
    </div>

    <div class="mb-3">
        <label class="form-label" for="inputPassword">Password:</label>
        <input
            type="password"
            name="password"
            id="inputPassword"
            class="form-control @error('password') is-invalid @enderror"
            placeholder="Password">

        <!-- Way 3: Display Error Message -->
        @if ($errors->has('password'))
            <span class="text-danger">{{ $errors->first('password') }}</span>
        @endif
    </div>

    <div class="mb-3">
        <label class="form-label" for="inputEmail">Email:</label>
        <input
            type="text"
            name="email"
            id="inputEmail"
            class="form-control @error('email') is-invalid @enderror"
            placeholder="Email">

        @error('email')
```

```
                <span class="text-danger">{{ $message }}</span>
            @endif
        </div>

        <div class="mb-3">
            <button class="btn btn-success btn-submit"><i class="fa fa-save"></i>
Submit</button>
        </div>
    </form>
  </div>
 </div>
</div>
</body>
</html>
```

php artisan serve

http://localhost:8000/users/create

Efetue alguns testes
Apenas Enter ou clique em /submit sem digitar nada

https://www.itsolutionstuff.com/post/laravel-11-custom-validation-error-message-exampleexample.html

Traduzindo as mensagens

As mensagens originais estão no método store() do controller:

Exemplo:

Name field is required - O campo name é obrigatório

2.9 - Autenticação múltipla com Breeze

Neste exemplo, instalaremos um aplicativo Laravel 11 limpo para multi-auth. Então instalaremos o Laravel Breeze para o scaffold de auth. Então adicionaremos uma coluna "type" na tabela de usuários, onde definiremos três tipos (User, Agent e Admin) de usuários. Então, criaremos um middleware "role" para verificar a permissão de acesso.

composer create-project laravel/laravel multi-auth

composer require laravel/breeze --dev
php artisan breeze:install
Blade with Alpine
No
Pest
npm install
npm run build

php artisan make:migration add_type_to_users_table

Atualizar a mirgation

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->enum('role',['admin','agent','user'])->default('user');
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('users', function (Blueprint $table) {
            //
        });
    }
};
```

php artisan migrate

Ajustar

app/Models/User.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\Casts\Attribute;

class User extends Authenticatable
{
    use HasFactory, Notifiable;
    protected $fillable = [
        'name',
        'email',
        'password',
        'role'
    ];

    protected $hidden = [
        'password',
        'remember_token',
    ];

    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }
}
```

Criar Role Middleware

php artisan make:middleware Role

app/Http/Middleware/Role.php

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

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class Role
```

```
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\
Response)  $next
     */
    public function handle(Request $request, Closure $next, $role): Response
    {
        if ($request->user()->role != $role) {
            return redirect('dashboard');
        }

        return $next($request);
    }
}
```

register the Role middleware to the app.php

bootstrap/app.php

```
<?php

use Illuminate\Foundation\Application;
use Illuminate\Foundation\Configuration\Exceptions;
use Illuminate\Foundation\Configuration\Middleware;

return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
    ->withMiddleware(function (Middleware $middleware) {
        $middleware->alias([
            'role' => \App\Http\Middleware\Role::class,
        ]);
    })
    ->withExceptions(function (Exceptions $exceptions) {
        //
    })->create();
```

Criar

routes/web.php

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

use App\Http\Controllers\AdminController;
use App\Http\Controllers\AgentController;

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

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

Route::middleware(['auth', 'role:admin'])->group(function(){
    Route::get('/admin/dashboard', [AdminController::class, 'dashboard'])-
>name('admin.dashboard');
});

Route::middleware(['auth', 'role:agent'])->group(function(){
    Route::get('/agent/dashboard', [AgentController::class, 'dashboard'])-
>name('agent.dashboard');
});

require __DIR__.'/auth.php';
```

Criar dois controllers

php artisan make:controller AdminController
php artisan make:controller AgentController

Atualizar

app/Http/Controllers/AdminController.php

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

use Illuminate\Http\Request;

class AdminController extends Controller
{
    /**
     * Write code on Method
     *
     * @return response()
     */
    public function dashboard()
    {
        return view('admin.dashboard');
    }
}
```

app/Http/Controllers/AgentController.php

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

use Illuminate\Http\Request;

class AgentController extends Controller
{
    /**
     * Write code on Method
     *
     * @return response()
     */
    public function dashboard()
    {
        return view('agent.dashboard');
    }
}
```

Criar a view

resources/views/admin/dashboard.blade.php

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">
            {{ __('Dashboard') }}
```

```
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">
                <div class="p-6 text-gray-900 dark:text-gray-100">
                    You are admin!.
                </div>
            </div>
        </div>
    </div>
</x-app-layout>
```

resources/views/agent/dashboard.blade.php

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 dark:text-gray-200 leading-tight">
            {{ __('Dashboard') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white dark:bg-gray-800 overflow-hidden shadow-sm sm:rounded-lg">
                <div class="p-6 text-gray-900 dark:text-gray-100">
                    You are agent!.
                </div>
            </div>
        </div>
    </div>
</x-app-layout>
```

Atualizar

app/Http/Controllers/Auth/AuthenticatedSessionController.php

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

use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\LoginRequest;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\View\View;

class AuthenticatedSessionController extends Controller
{
```

```
    /**
     * Display the login view.
     */
    public function create(): View
    {
        return view('auth.login');
    }

    /**
     * Handle an incoming authentication request.
     */
    public function store(LoginRequest $request): RedirectResponse
    {
        $request->authenticate();

        $request->session()->regenerate();

        $url = "dashboard";

        if ($request->user()->role == "admin") {
            $url = "admin/dashboard";
        } else if($request->user()->role == "agent"){
            $url = "agent/dashboard";
        }

        return redirect()->intended($url);
    }

    /**
     * Destroy an authenticated session.
     */
    public function destroy(Request $request): RedirectResponse
    {
        Auth::guard('web')->logout();

        $request->session()->invalidate();

        $request->session()->regenerateToken();

        return redirect('/');
    }
}
```

Criar seeder

php artisan make:seeder UserSeeder

database/seeders/UserSeeder.php

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\User;
use Illuminate\Support\Facades\Hash;

class UserSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        User::create([
            'name' => 'Admin',
            'email' => 'admin@gmail.com',
            'password' => Hash::make('123456'),
            'role' => 'admin'
        ]);

        User::create([
            'name' => 'Agent',
            'email' => 'agent@gmail.com',
            'password' => Hash::make('123456'),
            'role' => 'agent'
        ]);

        User::create([
            'name' => 'User',
            'email' => 'user@gmail.com',
            'password' => Hash::make('123456'),
            'role' => 'user'
        ]);
    }
}
```

php artisan db:seed --class=UserSeeder

php artisan serve

http://localhost:8000/login

Logar com

User
Email: user@gmail.com
Password: 123456

Admin User:
Email: admin@gmail.com
Password: 123456

Agent User:
Email: agent@gmail.com
Password: 123456

Admin e agent conseguem logar mas user nçao

https://www.itsolutionstuff.com/post/laravel-11-breeze-multi-auth-tutorialexample.html

2.10 - Exemplo de notificações

composer require ribafs/crud-generator --dev

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

php artisan crud:generate Diarios --fields='dia#date; texto#text;' --controller-namespace=App\\Http\\Controllers --form-helper=html

php artisan route:clear

php artisan migrate

php artisan serve

http://127.0.0.1:8000

Criar diário com PHP, MySQL e Bootstrap

Tenho duas fortes alternativas para a criação:

- PHP puro, que será mais flexível mas me dará mais trabalho
- Usando laravel 11 com o gerador de cruds + ckeditor para o texto

Parece que mesmo deixando o projeto bem maior a opção com laravel é mais adequada. Vejamos

```
create table diario(
        id int primary key auto_increment,
        dia DATETIME DEFAULT CURRENT_TIMESTAMP,
        texto text not null
);
```

```
   <input class="form-control" name="dia" type="datetime" id="dia" value="{{ isset($diario->dia) ? $diario->dia : date('Y-m-d')}}" >

<textarea class="form-control" rows="5" name="texto" type="textarea" id="texto" ><strong>{{ isset($diario->texto) ? $diario->texto : date('Y-m-d')}}</strong></textarea>
composer require orangehill/iseed
php artisan iseed diarios

php artisan migrate –seed
```

2.11 - Paginação de resultados no laravel 11

Em outras estruturas, a paginação pode ser muito dolorosa. Esperamos que a abordagem do Laravel para paginação seja uma lufada de ar fresco. 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 a estrutura Tailwind CSS; no entanto, o suporte à paginação Bootstrap também está disponível.

Paginando Resultados do Query Builder

Existem várias maneiras de paginar itens. A mais simples é usar o método paginate no query builder ou uma consulta Eloquent. O método paginate cuida automaticamente da configuração do "limit" e "offset" da consulta com base na página atual que está sendo visualizada pelo usuário. Por padrão, a página atual é detectada pelo valor do argumento da string de consulta da página na solicitação HTTP. Esse valor é detectado automaticamente pelo Laravel e também é inserido automaticamente em links gerados pelo paginador.

composer create-project laravel/laravel pagination

cd pagination

Configurar banco no .env

php artisan migrate

Gerar 100 registros de teste com tinker e factory

php artisan tinker

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

Adicionar rota

Route::get('users', [UserController::class, 'index']);

Criar controller

app/Http/Controllers/UserController.php

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

use Illuminate\Http\Request;
use App\Models\User;
use Illuminate\View\View;

class UserController extends Controller
```

```
{
    public function index(Request $request): View
    {
        $users = User::paginate(5);
        return view('users', compact('users'));
    }
}
```

Criar

resources/views/users.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Paginação de Resultados no Laravel</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/5.0.1/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container">
    <h1>Paginação de Resultados no Laravel</h1>
    <table class="table table-bordered data-table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
            </tr>
        </thead>
        <tbody>
            @forelse($users as $user)
                <tr>
                    <td>{{ $user->id }}</td>
                    <td>{{ $user->name }}</td>
                    <td>{{ $user->email }}</td>
                </tr>
            @empty
                <tr>
                    <td colspan="3">There are no users.</td>
                </tr>
            @endforelse
        </tbody>
    </table>
    {!! $users->withQueryString()->links('pagination::bootstrap-5') !!}
</div>
</body>
</html>
```

php artisan serve

http://localhost:8000/users

https://www.itsolutionstuff.com/post/laravel-10-pagination-example-tutorialexample.html

Agora vamos traduzir as palavras da paginação

Editar
vendor\laravel\framework\src\elluminate\Pagination\resources\views\bootstrap-5.blade.php

Showing - Mostrando
to - até
of - de
results - resultados

Paginando resultados do Eloquent

Você também pode paginar consultas do Eloquent. Neste exemplo, paginaremos o modelo App\Models\User e indicaremos que planejamos exibir 15 registros por página. Como você pode ver, a sintaxe é quase idêntica à paginação de resultados do construtor de consultas:

use App\Models\User;

$users = User::paginate(15);

É claro que você pode chamar o método paginate após definir outras restrições na consulta, como cláusulas where:

$users = User::where('votes', '>', 100)->paginate(15);

Você também pode usar o método simplePaginate ao paginar modelos do Eloquent:

$users = User::where('votes', '>', 100)->simplePaginate(15);

Da mesma forma, você pode usar o método cursorPaginate para paginar modelos Eloquent com cursor:

$users = User::where('votes', '>', 100)->cursorPaginate(15);

Exibindo resultados de paginação

Ao chamar o método paginate, você receberá uma instância de Illuminate\Pagination\LengthAwarePaginator, enquanto chamar o método simplePaginate retorna uma instância de Illuminate\Pagination\Paginator. E, finalmente, chamar o método cursorPaginate retorna uma instância de Illuminate\Pagination\CursorPaginator.

Esses objetos fornecem vários métodos que descrevem o conjunto de resultados. Além desses métodos auxiliares, as instâncias do paginador são iteradores e podem ser colocadas em loop como

uma matriz. Então, depois de recuperar os resultados, você pode exibi-los e renderizar os links de página usando o Blade:

@foreach ($users as $user)
{{ $user->name }}
@endforeach

{{ $users->links() }}

O método links renderizará os links para o restante das páginas no conjunto de resultados. Cada um desses links já conterá a variável de string de consulta de página adequada. Lembre-se, o HTML gerado pelo método links é compatível com o framework Tailwind CSS.

Personalizando a Visualização de Paginação

Por padrão, as visualizações renderizadas para exibir os links de paginação são compatíveis com a estrutura Tailwind CSS. No entanto, se você não estiver usando o Tailwind, você está livre para definir suas próprias visualizações para renderizar esses links. Ao chamar o método links em uma instância do paginador, você pode passar o nome da visualização como o primeiro argumento para o método:

{{ $paginator->links('view.name') }}

<!-- Passando dados adicionais para a visualização... -->
{{ $paginator->links('view.name', ['foo' => 'bar']) }}

No entanto, a maneira mais fácil de personalizar as visualizações de paginação é exportando-as para seu diretório resources/views/vendor usando o comando vendor:publish:

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

Este comando colocará as visualizações no diretório resources/views/vendor/pagination do seu aplicativo. O arquivo tailwind.blade.php dentro deste diretório corresponde à visualização de paginação padrão. Você pode editar este arquivo para modificar o HTML de paginação.

Se você quiser designar um arquivo diferente como a visualização de paginação padrão, você pode invocar os métodos defaultView e defaultSimpleView do paginador dentro do método boot da sua classe

App\Providers\AppServiceProvider:

<?php
namespace App\Providers;

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

class AppServiceProvider extends ServiceProvider

```
{
        public function boot(): void
        {
                Paginator::defaultView('view-name');
                Paginator::defaultSimpleView('view-name');
        }
}
```

Usando Bootstrap

O 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 boot da sua classe

App\Providers\AppServiceProvider:

```
use Illuminate\Pagination\Paginator;

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

2.12 - Relacionamento um para vários

Relacionamento um para vários de models

Como criar migração com um esquema de chave estrangeira para relacionamentos um para muitos, usar sincronização com uma tabela dinâmica, criar registros, obter todos os dados, excluir, atualizar e tudo relacionado a relacionamentos um para muitos.

Neste exemplo, criarei uma tabela "posts" e uma tabela "comments". Ambas as tabelas estão conectadas entre si. Agora, criaremos relacionamentos um (post) para muitos (comments) entre si usando o modelo Eloquent do laravel.

O relacionamento um para muitos usará "hasMany()"/tem muitos e "belongsTo()"/pertence a para a relação.

Migrations

```
public function up(): void
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string("name");
        $table->timestamps();
    });
}

public function up(): void
{
    Schema::create('comments', function (Blueprint $table) {
        $table->id();
        $table->foreignId('post_id')->constrained('posts');
        $table->string("comment");
        $table->timestamps();
    });
}
```

Models

app/Models/Post.php

```
<?php
namespace App\Models;

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

class Post extends Model
{
```

```
    use HasFactory;

    /**
     * Get the comments for the blog post.
     *
     * Syntax: return $this->hasMany(Comment::class, 'foreign_key', 'local_key');
     *
     * Example: return $this->hasMany(Comment::class, 'post_id', 'id');
     *
     */
    public function comments(): HasMany
    {
        return $this->hasMany(Comment::class);
    }
}
```

app/Models/Comment.php

```
<?php
namespace App\Models;

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

class Comment extends Model
{
    use HasFactory;

    /**
     * Get the post that owns the comment.
     *
     * Syntax: return $this->belongsTo(Post::class, 'foreign_key', 'owner_key');
     *
     * Example: return $this->belongsTo(Post::class, 'post_id', 'id');
     *
     */
    public function post(): BelongsTo
    {
        return $this->belongsTo(Post::class);
    }
}
```

Retrieve Records:

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

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

```
class PostController extends Controller
{
   /**
    * Write code on Method
    *
    * @return response()
    */
   public function index(Request $request)
   {
      $comments = Post::find(1)->comments;

      dd($comments);
   }
}
```

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

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

class PostController extends Controller
{
   /**
    * Write code on Method
    *
    * @return response()
    */
   public function index(Request $request)
   {
      $post = Comment::find(1)->post;

      dd($post);
   }
}
```

Create Records:

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

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

class PostController extends Controller
{
   /**
    * Write code on Method
```

```
     *
     * @return response()
     */
    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
{
    /**
     * Write code on Method
     *
     * @return response()
     */
    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
```

```
{
    /**
     * Write code on Method
     *
     * @return response()
     */
    public function index(Request $request)
    {
        $comment = Comment::find(1);
        $post = Post::find(2);

        $comment->post()->associate($post)->save();
    }
}
```

https://www.itsolutionstuff.com/post/laravel-10-one-to-many-eloquent-relationship-tutorialexample.html

Relacionamento tipo um para vários

Entre products e sales

```
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name')->notNllable();
            $table->integer('inventory_min')->notNullable();
            $table->integer('inventory_max')->notNullable();
            $table->timestamps();
        });


        Schema::create('sales', function (Blueprint $table) {
            $table->increments('id');
                            $table->unsignedBigInteger('product_id');
            $table->integer('quantity')->notNullable();
            $table->decimal('price')->notNullable();
            $table->date('date')->nullable();
            $table->timestamps();

      $table->foreign('product_id')->references('id')->on('products')->onUpdate('cascade')->onDeletee('cascade');
        });
```

https://www.itsolutionstuff.com/post/laravel-11-one-to-many-eloquent-relationship-tutorialexample.html

https://www.iankumu.com/blog/laravel-one-to-many-relationship/

2.13 - Sweetalert2

Usando Sweetalert2 em aplicativos com Laravel 11

Aqui estão três maneiras de instalar o SweetAlert2 no seu projeto Laravel. Vamos dar uma olhada em cada exemplo, um por um.

Example 1: Laravel Add Sweetalert2 using CDN

We can use CDN to get SweetAlert2. Just add this script tag in the head section of your HTML file. This way, you don't need to download anything. Here is an example of the simple Blade file code.

resources/views/welcome.blade.php

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

    <!-- CSRF Token -->
    <meta name="csrf-token" content="{{ csrf_token() }}">
    <title>{{ config('app.name', 'Laravel') }}</title>
    <!-- Fonts -->
    <link rel="dns-prefetch" href="//fonts.bunny.net">
    <link href="https://fonts.bunny.net/css?family=Nunito" rel="stylesheet">

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
    <link href='https://cdn.jsdelivr.net/npm/sweetalert2@10.10.1/dist/sweetalert2.min.css'>
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@10.16.6/dist/sweetalert2.all.min.js"></
script>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css">
</head>
<body>
    <div id="app">
        <main class="container">
            <h1> How To Install Sweetalert2 in Laravel? - ItSolutionstuiff.com</h1>
            <button class="btn btn-success">Click Me!</button>
        </main>
    </div>
</body>
<script>
  $('button').click(function(){
      Swal.fire({
        title: 'Are you sure?',
        text: "You won't be able to revert this!",
        icon: 'warning',
        showCancelButton: true,
```

```
      confirmButtonColor: '#3085d6',
      cancelButtonColor: '#d33',
      confirmButtonText: 'Yes, delete it!'
    }).then((result) => {
      if (result.isConfirmed) {
        Swal.fire(
          'Deleted!',
          'Your file has been deleted.',
          'success'
        )
      }
    });
  });
</script>
</html>
```

Example 2: Laravel Vite Add Sweetalert2 using NPM

Here, we will add Sweetalert2 using an npm command. So, let's run this command:

```
npm install jquery
npm install sweetalert2
```

Next, we need to add jQuery to our app.js. So, let's put the following lines in your app.js file.

```
resources/js/app.js
import jQuery from 'jquery';
window.$ = jQuery;

import swal from 'sweetalert2';
window.Swal = swal;
```

Next, we need to add $ in your vite config file. so, let's add it.

```
vite.config.js
import { defineConfig } from 'vite';
import laravel from 'laravel-vite-plugin';

export default defineConfig({
    plugins: [
        laravel({
            input: [
                'resources/sass/app.scss',
                'resources/js/app.js',
            ],
            refresh: true,
        }),
    ],
    resolve: {
        alias: {
```

```
            '$': 'jQuery'
        },
    },
});
```

Next, create the npm JS and CSS files with this command:

npm run dev

Now, we are ready to use jQuery with Vite. You can see the simple Blade file code below.

resources/views/welcome.blade.php

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

    <!-- CSRF Token -->
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>{{ config('app.name', 'Laravel') }}</title>

    <!-- Fonts -->
    <link rel="dns-prefetch" href="//fonts.bunny.net">
    <link href="https://fonts.bunny.net/css?family=Nunito" rel="stylesheet">

    <!-- Scripts -->
    @vite(['resources/sass/app.scss', 'resources/js/app.js'])

    <script type="module">

        $('button').click(function(){
            Swal.fire({
              title: 'Are you sure?',
              text: "You won't be able to revert this!",
              icon: 'warning',
              showCancelButton: true,
              confirmButtonColor: '#3085d6',
              cancelButtonColor: '#d33',
              confirmButtonText: 'Yes, delete it!'
            }).then((result) => {
              if (result.isConfirmed) {
                Swal.fire(
                  'Deleted!',
                  'Your file has been deleted.',
                  'success'
                )
              }
```

```
            });
        });

    </script>

</head>
<body>
    <div id="app">

        <main class="container">
            <h1> How To Install Sweetalert2 in Laravel? - ItSolutionstuiff.com</h1>

            <button class="btn btn-success">Click Me</button>
        </main>
    </div>

</body>
</html>
```

Run Laravel App:

All the required steps have been done, now you have to type the given below command and hit enter to run the Laravel app:

php artisan serve

Now, Go to your web browser, type the given URL and view the app output:

http://localhost:8000/

Output:
I hope it can help you…
https://www.itsolutionstuff.com/post/how-to-install-sweetalert2-in-laravel-11-viteexample.html

3 - Tutoriais

3.1 - Fluxo de informações

O que chamo de fluxo de informações é o caminho em que as informações/dados percorrem no laravel. Um exemplo bem simples, efetuar as operações em um CRUD:

- Inicia chamando pela URL e desejado CRUD. Exemplo
http://127.0.0.1:8000/products
- Esta URL é recebida pela rota, caso ela exista
- A rota chamará o método default (index) no controller ProductController
- O controller efetua seu processamento e devolve o resultado para a view products/index.blade.php
- A view index mostra os registros na tela, caso existam

Existem também as demais operações: create, edit, show e delete/destroy

Quando criamos um CRUD no laravel, seguindo suas convenções, ele funciona praticamente sozinho, sem intervenção: Inserimos um novo registro apenas entrando as informações no create e as enviando. Assim com as outras operações. Mas e se eu quiser fazer uma operação diferente, como ao mesmo tempo em que cadastrar um produto, cadastrar este mesmo produto em outra janela? Se eu quiser ao mesmo tempo em que efetuar uma venda deduzir a quantidade da venda na tabela estoques? Isso estou fazendo no projeto Estoque simplificado.

O Eloquent e o QueryBuilder são os principais envolvidos quando lidamos com o código nos métodos do controller. Então trouxe informações sobre estes.

Partes do laravel conversando e trocando informações entre si

Mostrando como enviar e receber informações/dados

De um controller para uma view

De uma view para um conroller

O controller envia para o model

O model envia para a tabela do banco de dados

Controller para uma view

Do método index() o controller chama por padrão a view index e passa para ela algumas informações

No ProductController, método index()

```
$products = Product::latest()->paginate($perPage);
return view('products.index', compact('products'));
```

Ele chama a view products.index, passando para ela a variável $products, que recebeu o objeto Product.

Na view products/index.blade.php

Ela faz um laço pela variável $products para mostrar em uma tabela

```
@foreach($products as $item)
   <tr>
      <td>{{ $item->id }}</td>
      <td>{{ $item->name }}</td>
      <td>{{ $item->price }}</td>
      ...
```

Uma view pode enviar informações para um controller

A view create chama o método store() do controller, que através do Request pega os dados do form e os envia para o respectivo model, no caso, Product (use App\Models\Product), assim, através do model grava o registro na tabela.

```
   public function store(Request $request)
   {
      $requestData = $request->all();
      dd($requestData);
```

Assim podemos ver as informações vindas da view create e recebidas via Request

Enviando informações (array) do método index() para a view welcome

```
public function index()
{
   $data['name'] = 'Najmul Hasan';
   $data['age'] = 30;
   return view('welcome', $data);
}
```

Enviar da view para o controller

Vejamos como o laravel recebe as informações vindas da view create no método store() do controller

```
   public function store(Request $request)
   {
      $requestData = $request->all();
      dd($requestData);
```

Ao executar vemos no navegador:

```
array:3 [▼ // app/Http/Controllers/ProductsController.php:34
  "_token" => "V4DS3Piynt31YexmsvxpCMI50VxGelgWuWyMZKvY"
  "name" => "teste1"
  "price" => "1"
]
```

Um array com os dois campos do form mais o token

Então, para receber apenas oo name, podemos fazer assim:

```
dd($requestData['name']);
```

Receberemos:
"teste1" // app/Http/Controllers/ProductsController.php:34

Obs.: o request leva somente campos do form e o token, nada mais.
Adicionando mis um campo ao form ele também foi capturado pelo store().

Executar a view products/create

controller Product store
Adicionar ao método store

```
public function store(Request $request)
{
    dd('Vindo da view store');
    ...
```

Model Product
Ewmover a linha axima e criar o método no Model

```
function __construct(){
   dd('Vindo do controller');
}
```

Enviando informações do controller para a view

app/Http/Controllers/DemoController.php

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

use Illuminate\Http\Request;

class DemoController extends Controller
{
   public function index(Request $request)
   {
      $title = "Welcome to ItSolutionStuff.com";
      $subTitle = "Thank you";
```

```
        return view('demo', compact('title', 'subTitle'));
    }
}
```

resources/views/demo.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title></title>
</head>
<body>
    <h1>{{ $title }}</h1>
    <h2>{{ $subTitle }}</h2>
</body>
</html>
```

Example 2: Using Array

app/Http/Controllers/DemoController.php

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

use Illuminate\Http\Request;

class DemoController extends Controller
{
    public function index(Request $request)
    {
        $title = "Welcome to ItSolutionStuff.com";
        $subTitle = "Thank you";

        return view('demo', [
            'title' => $title,
            'subTitle' => $subTitle,
        ]);
    }
}
```

resources/views/demo.blade.php

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title></title>
```

```
</head>
<body>
   <h1>{{ $title }}</h1>
   <h2>{{ $subTitle }}</h2>
</body>
</html>
```

Example 3: Using With()

app/Http/Controllers/DemoController.php

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

use Illuminate\Http\Request;

class DemoController extends Controller
{
   public function index(Request $request)
   {
      $title = "Welcome to ItSolutionStuff.com";
      $subTitle = "Thank you";
      return view('demo')
            ->with('title', $title)
            ->with('subTitle', $subTitle);
   }
}
```

resources/views/demo.blade.php

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <title></title>
</head>
<body>
   <h1>{{ $title }}</h1>
   <h2>{{ $subTitle }}</h2>
</body>
</html>
```

https://www.itsolutionstuff.com/post/how-to-pass-data-from-controller-to-view-in-laravelexample.html

3.2 – Eloquent ORM

O Laravel inclui o Eloquent, um ORM (mapeador objeto-relacional que torna agradável interagir com seu banco de dados. Ao usar o Eloquent, cada tabela do banco de dados tem um "Modelo" correspondente que é usado para interagir com essa tabela. Além de recuperar registros da tabela do banco de dados, os modelos do Eloquent permitem que você insira, atualize e exclua registros da tabela também.

Antes de começar, certifique-se de configurar uma conexão de banco de dados no arquivo de configuração config/database.php do seu aplicativo ou no .env.

Os relacionamentos Eloquent são definidos como métodos em suas classes de modelo Eloquent. Como os relacionamentos também servem como poderosos construtores de consultas, definir relacionamentos como métodos fornece poderosos recursos de encadeamento e consulta de métodos. Por exemplo, podemos encadear restrições de consulta adicionais neste relacionamento de postagens

$user->posts()->where('active', 1)->get();

Timestamps

Por padrão, o Eloquent espera que as colunas created_at e updated_at existam na tabela de banco de dados correspondente do seu modelo. O Eloquent definirá automaticamente os valores dessas colunas quando os modelos forem criados ou atualizados. Se você não quiser que essas colunas sejam gerenciadas automaticamente pelo Eloquent, você deve definir uma propriedade $timestamps no seu modelo com um valor false:

No model

public $timestamps = false;

Se você precisar personalizar o formato dos timestamps do seu modelo, defina a propriedade $dateFormat no seu modelo. Essa propriedade determina como os atributos de data são armazenados no banco de dados, bem como seu formato quando o modelo é serializado para um array ou JSON:

```
    protected $dateFormat = 'U';
```

You can individually customize the format of Eloquent date and datetime casting:

```
protected $casts = [
    'birthday'  => 'date:Y-m-d',
    'joined_at' => 'datetime:Y-m-d H:00',
];
```

Insert

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

use App\Http\Controllers\Controller;
use App\Models\Flight;
use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;

class FlightController extends Controller
{
    /**
     * Store a new flight in the database.
     */
    public function store(Request $request): RedirectResponse
    {
        // Validate the request...

        $flight = new Flight;

        $flight->name = $request->name;

        $flight->save();

        return redirect('/flights');
    }
}
```

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::destroy(1);

Flight::destroy(1, 2, 3);
```

Flight::destroy([1, 2, 3]);

Flight::destroy(collect([1, 2, 3]));

Ordenando

```
use App\Models\Flight;

$flights = Flight::withTrashed()
            ->where('account_id', 1)
            ->get();
```

Relacionamentos Um para Um

Um relacionamento um-para-um é um tipo muito básico de relacionamento de banco de dados. Por exemplo, um modelo User pode ser associado a um modelo Phone. Para definir esse relacionamento, colocaremos um método phone no modelo User. O método phone deve chamar o método hasOne e retornar seu resultado. O método hasOne está disponível para seu modelo por meio da classe base Illuminate\Database\Eloquent\Model do modelo

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasOne;

class User extends Model
{
    /**
     * Get the phone associated with the user.
     */
    public function phone(): HasOne
    {
        return $this->hasOne(Phone::class);
    }
}
```

O primeiro argumento passado para o método hasOne é o nome da classe do modelo relacionado. Uma vez que o relacionamento é definido, podemos recuperar o registro relacionado usando as propriedades dinâmicas do Eloquent. As propriedades dinâmicas permitem que você acesse métodos de relacionamento como se fossem propriedades definidas no modelo

```
$phone = User::find(1)->phone;
```

O Eloquent determina a chave estrangeira do relacionamento com base no nome do modelo pai. Nesse caso, o modelo Phone é automaticamente assumido como tendo uma chave estrangeira user_id. Se você deseja substituir essa convenção, pode passar um segundo argumento para o método hasOne:

return $this->hasOne(Phone::class, 'foreign_key');

Além disso, o Eloquent assume que a chave estrangeira deve ter um valor que corresponda à coluna de chave primária do pai. Em outras palavras, o Eloquent procurará o valor da coluna id do usuário na coluna user_id do registro Phone. Se você quiser que o relacionamento use um valor de chave primária diferente de id ou da propriedade $primaryKey do seu modelo, você pode passar um terceiro argumento para o método hasOne:

return $this->hasOne(Phone::class, 'foreign_key', 'local_key');

Um para Muitos

Um relacionamento um-para-muitos é usado para definir relacionamentos em que um único modelo é o pai de um ou mais modelos filhos. Por exemplo, uma postagem de blog pode ter um número infinito de comentários. Como todos os outros relacionamentos Eloquent, os relacionamentos um-para-muitos são definidos pela definição de um método no seu modelo Eloquent:

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Post extends Model
{
    /**
     * Get the comments for the blog post.
     */
    public function comments(): HasMany
    {
        return $this->hasMany(Comment::class);
    }
}
```

Lembre-se, o Eloquent determinará automaticamente a coluna de chave estrangeira apropriada para o modelo Comment. Por convenção, o Eloquent pegará o nome "snake case" do modelo pai e o sufixará com _id. Então, neste exemplo, o Eloquent assumirá que a coluna de chave estrangeira no modelo Comment é post_id.

Uma vez que o método relationship tenha sido definido, podemos acessar a coleção de comentários relacionados acessando a propriedade comments. Lembre-se, como o Eloquent fornece "propriedades de relacionamento dinâmicas", podemos acessar métodos de relacionamento como se eles fossem definidos como propriedades no modelo:

```
use App\Models\Post;

$comments = Post::find(1)->comments;

foreach ($comments as $comment) {
    // ...
}
```

Como todos os relacionamentos também servem como construtores de consultas, você pode adicionar mais restrições à consulta de relacionamento chamando o método comments e continuando a encadear condições na consulta:

```
$comment = Post::find(1)->comments()
                ->where('title', 'foo')
                ->first();
```

Assim como o método hasOne, você também pode substituir as chaves estrangeiras e locais passando argumentos adicionais para o método hasMany:

```
return $this->hasMany(Comment::class, 'foreign_key');

return $this->hasMany(Comment::class, 'foreign_key', 'local_key');
```

Consultando Relacionamentos

Como todos os relacionamentos Eloquent são definidos por meio de métodos, você pode chamar esses métodos para obter uma instância do relacionamento sem realmente executar uma consulta para carregar os modelos relacionados. Além disso, todos os tipos de relacionamentos Eloquent também servem como construtores de consulta, permitindo que você continue a encadear restrições na consulta de relacionamento antes de finalmente executar a consulta SQL em seu banco de dados.

Por exemplo, imagine um aplicativo de blog no qual um modelo User tem muitos modelos Post associados:

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class User extends Model
{
    public function posts(): HasMany
    {
        return $this->hasMany(Post::class);
    }
}
```

Você pode consultar o relacionamento das postagens e adicionar restrições adicionais ao relacionamento, como esta:

```
use App\Models\User;

$user = User::find(1);

$user->posts()->where('active', 1)->get();
```

Encadeando cláusulas orWhere após relacionamentos

Conforme demonstrado no exemplo acima, você tem liberdade para adicionar restrições adicionais aos relacionamentos ao consultá-los. No entanto, tenha cuidado ao encadear cláusulas orWhere em um relacionamento, pois as cláusulas orWhere serão logicamente agrupadas no mesmo nível da restrição do relacionamento:

```
$user->posts()
    ->where('active', 1)
    ->orWhere('votes', '>=', 100)
    ->get();
```

Na maioria das situações, você deve usar grupos lógicos para agrupar as verificações condicionais entre parênteses:

```
use Illuminate\Database\Eloquent\Builder;

$user->posts()
    ->where(function (Builder $query) {
        return $query->where('active', 1)
                    ->orWhere('votes', '>=', 100);
    })
    ->get();
```

Similar a

```
select *
from posts
where user_id = ? and active = 1 or votes >= 100
```

```
use App\Models\Post;

// Recuperar todas as postagens que tenham pelo menos um comentário...
$posts = Post::has('comments')->get();
```

Você também pode especificar um operador e um valor de contagem para personalizar ainda mais a consulta:

```
// Recuperar todas as postagens que tenham três ou mais comentários...
$posts = Post::has('comments', '>=', 3)->get();
```

```
// Recuperar postagens que tenham pelo menos um comentário com imagens...
$posts = Post::has('comments.images')->get();

use Illuminate\Database\Eloquent\Builder;

// Retrieve posts with at least one comment containing words like code%...
$posts = Post::whereHas('comments', function (Builder $query) {
    $query->where('content', 'like', 'code%');
})->get();

// Retrieve posts with at least ten comments containing words like code%...
$posts = Post::whereHas('comments', function (Builder $query) {
    $query->where('content', 'like', 'code%');
}, '>=', 10)->get();

use App\Models\Post;

$posts = Post::whereRelation('comments', 'is_approved', false)->get();
```

Of course, like calls to the query builder's where method, you may also specify an operator:

```
$posts = Post::whereRelation(
    'comments', 'created_at', '>=', now()->subHour()
)->get();

use App\Models\Post;

$posts = Post::doesntHave('comments')->get();

use Illuminate\Database\Eloquent\Builder;

$posts = Post::whereDoesntHave('comments', function (Builder $query) {
    $query->where('content', 'like', 'code%');
})->get();

use Illuminate\Database\Eloquent\Builder;

$posts = Post::whereDoesntHave('comments.author', function (Builder $query) {
    $query->where('banned', 0);
})->get();


use Illuminate\Database\Eloquent\Builder;

$comments = Comment::whereHasMorph('commentable', '*', function (Builder $query) {
    $query->where('title', 'like', 'foo%');
})->get();
```

```
$posts = Post::select(['title', 'body'])
            ->withCount('comments')
            ->get();
```

```
use App\Models\Post;

$posts = Post::withSum('comments', 'votes')->get();

foreach ($posts as $post) {
   echo $post->comments_sum_votes;
}
```

If you wish to access the result of the aggregate function using another name, you may specify your own alias:

```
$posts = Post::withSum('comments as total_comments', 'votes')->get();

foreach ($posts as $post) {
   echo $post->total_comments;
}

$posts = Post::select(['title', 'body'])
            ->withExists('comments')
            ->get();
```

Save

```
use App\Models\Comment;
use App\Models\Post;

$comment = new Comment(['message' => 'A new comment.']);

$post = Post::find(1);

$post->comments()->save($comment);
```

Find

```
$post = Post::find(1);

$post->comments()->saveMany([
   new Comment(['message' => 'A new comment.']),
   new Comment(['message' => 'Another new comment.']),
]);
```

```
student_record::create(array(
   'first_name' => 'John',
   'last_name'  => 'Doe',
   'student_rank' => 1
));

Todos
$student = Students::all();

id = 1
$student = Students::find(1);

$JohnDoe = Students::where('name', '=', 'John Doe')->first();

$rankStudents = Student::where('student_rank', '>', 5)->get();

$JohnDoe = Bear::where('name', '=', 'John Doe')->first();
$JohnDoe->danger_level = 5;
$JohnDoe->save();

$student = Students::find(1);
$student->delete();

Students::destroy(1);
Students::destroy(1, 2, 3);

Students::where('student_rank', '>', 10)->delete();

https://stackify.com/laravel-eloquent-tutorial/

updateOrCreate
```

O método updateOrCreate tenta encontrar um Model que corresponda às restrições passadas como o primeiro parâmetro. Se um Model correspondente for encontrado, ele atualizará a correspondência com os atributos passados como o segundo parâmetro. Se nenhum Model correspondente for encontrado, um novo Model será criado com as restrições passadas como o primeiro parâmetro e os atributos passados como o segundo parâmetro.

Se 'product_id' = $request->product_id atualiza, caso contrário cria

In your use case, you should specify a second parameter. The first indicates the conditions for a match and second is used to specify which fields to update.

```
$inventories = Inventory::updateOrCreate(
   ['product_id' => $request->product_id],
   ['quantity' =>  'quantity' - $request->quantity],
);
```

```
$flight = Flight::updateOrCreate(
    ['departure' => 'Oakland', 'destination' => 'San Diego'],
    ['price' => 99, 'discounted' => 1]
);

Inventory::updateOrCreate(
    ['product_id' => $request->product_id],
    ['quantity' => $request->quantity]
);

$flight = Flight::updateOrCreate(
    ['departure' => 'Oakland', 'destination' => 'San Diego'],
    ['price' => 99, 'discounted' => 1]
);

User::updateOrCreate(
    [
        'email' => 'arthur@doyle.com',
        'name' => 'Arthur Conan Doyle',
    ],
    ['email_verified' => 0]
);

User::updateOrCreate(
    [
        'email' => 'e.e@hotmail.com',
        // will find even if case doesn't match
        // due to the case-insensitive collation on the table
        'name' => 'emilia earhart',
    ],
    ['email_verified' => 0]
);

User::updateOrCreate(
    [
        'email' => 'etan@fake.dev',
        'name' => 'Etan Hawkes',
    ],
    ['email_verified' => 1]
);

$user = User::updateOrCreate(
    ['email' =>  request('email')],
    ['name' => request('name')]
);

        $settings = $request->except(['_token', 'guest_share']);

        foreach ($settings as $key => $value) {
            Setting::updateOrCreate([
```

```
            'key' => $key,
            'user_id' => null,
        ], [
            'key' => $key,
            'value' => $value,
            'user_id' => null,
        ]);
    }

    $post = Post::updateOrCreate([
        'title' => 'Post 3'
    ], [
        'description' => 'Description for post 3.',
        'body' => 'body for post 3 - updated.'
    ]);

    print_r($post);die;

    $post = Post::updateOrCreate([
        'title' => 'Post 3'
    ], [
        'description' => 'Description for post 3.',
        'body' => 'body for post 3 - updated.'
    ]);

    print_r($post);die;

$flight = Flight::updateOrCreate(
    ['departure' => 'Oakland', 'destination' => 'San Diego'],
    ['price' => 99, 'discounted' => 1]
);


$matchThese = ['shopId'=>$theID,'metadataKey'=>2001];
ShopMeta::updateOrCreate($matchThese,['shopOwner'=>'New One']);

$matchThese = array('shopId'=>$theID,'metadataKey'=>2001);
DB::table('shop_metas')::updateOrCreate($matchThese,['shopOwner'=>'New One']);
```

Adiciona a facade DB:

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

// Instead of this
$flight = Flight::where('departure', 'Oakland')
    ->where('destination', 'San Diego')
    ->first();
if ($flight) {
    $flight->update(['price' => 99, 'discounted' => 1]);
} else {
```

```
    $flight = Flight::create([
        'departure' => 'Oakland',
        'destination' => 'San Diego',
        'price' => 99,
        'discounted' => 1
    ]);
}
// Do it in ONE sentence
$flight = Flight::updateOrCreate(
    ['departure' => 'Oakland', 'destination' => 'San Diego'],
    ['price' => 99, 'discounted' => 1]
);

    public function store(Request $request)
    {
        if ($request->ajax()) {
            $request->validate([
                'name'      => 'required',
                'email'     => 'required|email|unique:users,email,' .$request->user_id,
                'password'  => 'required',
                'role'      => 'required',
            ]);

            $user = User::updateOrCreate(['id' => $request->user_id],
            [
                'name'      => $request->name,
                'email'     => $request->email,
                'role'      => $request->role,
                'password'  => Hash::make($request->password)
            ]);
        }
    }

     $student = Student::updateOrCreate(
        ['name' => 'Arbaaz'],
        ['age'=> 40,'email'=>'arbaaz@gmail.com', 'address'=>'xyz']
     );

Post::where('id',3)->update(['title'=>'Updated title']);

Inventory::where('product_id',$request->product_id)->update(['quantity'=>$request->quantity]);

    $item = Item::find($id);
    $item->name = $request->input('name');
    $item->description = $request->input('description');

    $item->save();

        $user = User::findOrFail($request->id);
```

```
    if ($user->updateOrFail($request->all()) === false) {
        return response(
            "Couldn't update the user with id {$request->id}",
            Response::HTTP_BAD_REQUEST
        );
    }

    return response($user);

public function update(Request $request): Response
{
    return response(User::updateOrThrow($request->id, $request->all()));
}
```

3.3 - QueryBuilder

O construtor de consultas 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 seu aplicativo e funciona perfeitamente com todos os sistemas de banco de dados suportados pelo Laravel.

O construtor de consultas do Laravel usa vinculação de parâmetros PDO para proteger seu aplicativo contra ataques de injeção de SQL. Não há necessidade de limpar ou higienizar strings passadas para o construtor de consultas como vinculações de consulta.

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

use Illuminate\Support\Facades\DB;
use Illuminate\View\View;

class UserController extends Controller
{
   public function index(): View
   {
      $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;
}

$user = DB::table('users')->where('name', 'John')->first();

return $user->email;

$email = DB::table('users')->where('name', 'John')->value('email');

Receber o id=3
$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;
}

use Illuminate\Support\Collection;
use Illuminate\Support\Facades\DB;

DB::table('users')->orderBy('id')->chunk(100, function (Collection $users) {
    foreach ($users as $user) {
        // ...
    }
});

DB::table('users')->orderBy('id')->chunk(100, function (Collection $users) {
    // Process the records...

    return false;
});

DB::table('users')->where('active', false)
    ->chunkById(100, function (Collection $users) {
        foreach ($users as $user) {
            DB::table('users')
                ->where('id', $user->id)
                ->update(['active' => true]);
        }
    });

use Illuminate\Support\Facades\DB;

DB::table('users')->orderBy('id')->lazy()->each(function (object $user) {
    // ...
});

DB::table('users')->where('active', false)
    ->lazyById()->each(function (object $user) {
        DB::table('users')
            ->where('id', $user->id)
            ->update(['active' => true]);
    });

use Illuminate\Support\Facades\DB;

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

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

```
$price = DB::table('orders')
            ->where('finalized', 1)
            ->avg('price');

if (DB::table('orders')->where('finalized', 1)->exists()) {
   // ...
}

if (DB::table('orders')->where('finalized', 1)->doesntExist()) {
   // ...
}

use Illuminate\Support\Facades\DB;

$users = DB::table('users')
         ->select('name', 'email as user_email')
         ->get();

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

$query = DB::table('users')->select('name');

$users = $query->addSelect('age')->get();

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

$orders = DB::table('orders')
            ->selectRaw('price * ? as price_with_tax', [1.0825])
            ->get();

$orders = DB::table('orders')
            ->whereRaw('price > IF(state = "TX", ?, 100)', [200])
            ->get();

$orders = DB::table('orders')
            ->select('department', DB::raw('SUM(price) as total_sales'))
            ->groupBy('department')
            ->havingRaw('SUM(price) > ?', [2500])
            ->get();

$orders = DB::table('orders')
            ->orderByRaw('updated_at - created_at DESC')
            ->get();

$orders = DB::table('orders')
            ->select('city', 'state')
```

```
            ->groupByRaw('city, state')
            ->get();

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();

DB::table('users')
      ->join('contacts', function (JoinClause $join) {
          $join->on('users.id', '=', 'contacts.user_id')->orOn(/* ... */);
      })
      ->get();

DB::table('users')
      ->join('contacts', function (JoinClause $join) {
          $join->on('users.id', '=', 'contacts.user_id')
               ->where('contacts.user_id', '>', 5);
      })
      ->get();

$latestPosts = DB::table('posts')
               ->select('user_id', DB::raw('MAX(created_at) as last_post_created_at'))
               ->where('is_published', true)
               ->groupBy('user_id');

$users = DB::table('users')
      ->joinSub($latestPosts, 'latest_posts', function (JoinClause $join) {
          $join->on('users.id', '=', 'latest_posts.user_id');
      })->get();

$latestPosts = DB::table('posts')
               ->select('id as post_id', 'title as post_title', 'created_at as post_created_at')
               ->whereColumn('user_id', 'users.id')
               ->orderBy('created_at', 'desc')
               ->limit(3);

$users = DB::table('users')
        ->joinLateral($latestPosts, 'latest_posts')
```

```
        ->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)->get();
```

As previously mentioned, you may use any operator that is supported by your database system:

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

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

$users = DB::table('users')
            ->where('name', 'like', 'T%')
            ->get();

$users = DB::table('users')->where([
   ['status', '=', '1'],
   ['subscribed', '<>', '1'],
])->get();

$users = DB::table('users')
                ->where('votes', '>', 100)
                ->orWhere('name', 'John')
                ->get();

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

The example above will produce the following SQL:

```
select * from users where votes > 100 or (name = 'Abigail' and votes > 50)
```

```
$products = DB::table('products')
            ->whereNot(function (Builder $query) {
                $query->where('clearance', true)
                      ->orWhere('price', '<', 10);
            })
            ->get();

$users = DB::table('users')
        ->where('active', true)
        ->whereAny([
            'name',
            'email',
            'phone',
        ], 'like', 'Example%')
        ->get();
```

The query above will result in the following SQL:

```
SELECT *
FROM users
WHERE active = true AND (
    name LIKE 'Example%' OR
    email LIKE 'Example%' OR
    phone LIKE 'Example%'
)
```

Similarly, the whereAll method may be used to retrieve records where all of the given columns match a given constraint:

```
$posts = DB::table('posts')
        ->where('published', true)
        ->whereAll([
            'title',
            'content',
        ], 'like', '%Laravel%')
        ->get();
```

The query above will result in the following SQL:

```
SELECT *
FROM posts
WHERE published = true AND (
    title LIKE '%Laravel%' AND
    content LIKE '%Laravel%'
)
```

The whereNone method may be used to retrieve records where none of the given columns match a given constraint:

```
$posts = DB::table('albums')
        ->where('published', true)
        ->whereNone([
            'title',
            'lyrics',
            'tags',
        ], 'like', '%explicit%')
        ->get();
```

The query above will result in the following SQL:

```
SELECT *
FROM albums
WHERE published = true AND NOT (
   title LIKE '%explicit%' OR
   lyrics LIKE '%explicit%' OR
   tags LIKE '%explicit%'
)
```

```
$users = DB::table('users')
            ->where('preferences->dining->meal', 'salad')
            ->get();
```

You may use whereJsonContains to query JSON arrays:

```
$users = DB::table('users')
            ->whereJsonContains('options->languages', 'en')
            ->get();
```

If your application uses the MariaDB, MySQL, or PostgreSQL databases, you may pass an array of values to the whereJsonContains method:

```
$users = DB::table('users')
            ->whereJsonContains('options->languages', ['en', 'de'])
            ->get();
```

You may use whereJsonLength method to query JSON arrays by their length:

```
$users = DB::table('users')
            ->whereJsonLength('options->languages', 0)
            ->get();

$users = DB::table('users')
            ->whereJsonLength('options->languages', '>', 1)
            ->get();

$users = DB::table('users')
```

```
    ->whereLike('name', '%John%')
    ->get();
```

You can enable a case-sensitive search via the caseSensitive argument:

```
$users = DB::table('users')
    ->whereLike('name', '%John%', caseSensitive: true)
    ->get();
```

The orWhereLike method allows you to add an "or" clause with a LIKE condition:

```
$users = DB::table('users')
    ->where('votes', '>', 100)
    ->orWhereLike('name', '%John%')
    ->get();
```

The whereNotLike method allows you to add "NOT LIKE" clauses to your query:

```
$users = DB::table('users')
    ->whereNotLike('name', '%John%')
    ->get();
```

Similarly, you can use orWhereNotLike to add an "or" clause with a NOT LIKE condition:

```
$users = DB::table('users')
    ->where('votes', '>', 100)
    ->orWhereNotLike('name', '%John%')
    ->get();
```

The whereLike case-sensitive search option is currently not supported on SQL Server.
whereIn / whereNotIn / orWhereIn / orWhereNotIn
The whereIn method verifies that a given column's value is contained within the given array:

```
$users = DB::table('users')
        ->whereIn('id', [1, 2, 3])
        ->get();
```

The whereNotIn method verifies that the given column's value is not contained in the given array:

```
$users = DB::table('users')
        ->whereNotIn('id', [1, 2, 3])
        ->get();
```

You may also provide a query object as the whereIn method's second argument:

```
$activeUsers = DB::table('users')->select('id')->where('is_active', 1);

$users = DB::table('comments')
        ->whereIn('user_id', $activeUsers)
        ->get();
```

The example above will produce the following SQL:

```
select * from comments where user_id in (
   select id
   from users
   where is_active = 1
)
```

```
$users = DB::table('users')
        ->whereBetween('votes', [1, 100])
        ->get();
```

whereNotBetween / orWhereNotBetween

The whereNotBetween method verifies that a column's value lies outside of two values:

```
$users = DB::table('users')
                ->whereNotBetween('votes', [1, 100])
                ->get();

$patients = DB::table('patients')
                  ->whereBetweenColumns('weight', ['minimum_allowed_weight',
'maximum_allowed_weight'])
                ->get();
```

The whereNotBetweenColumns method verifies that a column's value lies outside the two values of two columns in the same table row:

```
$patients = DB::table('patients')
                ->whereNotBetweenColumns('weight', ['minimum_allowed_weight',
'maximum_allowed_weight'])
                ->get();

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

The whereNotNull method verifies that the column's value is not NULL:

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

$users = DB::table('users')
            ->whereDate('created_at', '2016-12-31')
            ->get();
```

The whereMonth method may be used to compare a column's value against a specific month:

```
$users = DB::table('users')
            ->whereMonth('created_at', '12')
            ->get();
```

The whereDay method may be used to compare a column's value against a specific day of the month:

```
$users = DB::table('users')
            ->whereDay('created_at', '31')
            ->get();
```

The whereYear method may be used to compare a column's value against a specific year:

```
$users = DB::table('users')
            ->whereYear('created_at', '2016')
            ->get();
```

The whereTime method may be used to compare a column's value against a specific time:

```
$users = DB::table('users')
            ->whereTime('created_at', '=', '11:20:45')
            ->get();

$users = DB::table('users')
            ->whereColumn('first_name', 'last_name')
            ->get();
```

You may also pass a comparison operator to the whereColumn method:

```
$users = DB::table('users')
            ->whereColumn('updated_at', '>', 'created_at')
            ->get();
```

You may also pass an array of column comparisons to the whereColumn method. These conditions will be joined using the and operator:

```
$users = DB::table('users')
            ->whereColumn([
               ['first_name', '=', 'last_name'],
               ['updated_at', '>', 'created_at'],
            ])->get();
```

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

select * from users where name = 'John' and (votes > 100 or title = 'Admin')

$users = DB::table('users')
        ->whereExists(function (Builder $query) {
            $query->select(DB::raw(1))
                ->from('orders')
                ->whereColumn('orders.user_id', 'users.id');
        })
        ->get();
```

Alternatively, you may provide a query object to the whereExists method instead of a closure:

```
$orders = DB::table('orders')
            ->select(DB::raw(1))
            ->whereColumn('orders.user_id', 'users.id');

$users = DB::table('users')
                ->whereExists($orders)
                →get();
```

Both of the examples above will produce the following SQL:

```
select * from users
where exists (
    select 1
    from orders
    where orders.user_id = users.id
)

use App\Models\User;
use Illuminate\Database\Query\Builder;

$users = User::where(function (Builder $query) {
    $query->select('type')
        ->from('membership')
        ->whereColumn('membership.user_id', 'users.id')
        ->orderByDesc('membership.start_date')
        ->limit(1);
}, 'Pro')->get();

use App\Models\Income;
use Illuminate\Database\Query\Builder;
```

```
$incomes = Income::where('amount', '<', function (Builder $query) {
    $query->selectRaw('avg(i.amount)')->from('incomes as i');
})->get();

$users = DB::table('users')
        ->whereFullText('bio', 'web developer')
        ->get();

$users = DB::table('users')
            ->orderBy('name', 'desc')
            ->get();
```

To sort by multiple columns, you may simply invoke orderBy as many times as necessary:

```
$users = DB::table('users')
            ->orderBy('name', 'desc')
            ->orderBy('email', 'asc')
            ->get();

$user = DB::table('users')
            ->latest()
            ->first();

$randomUser = DB::table('users')
            ->inRandomOrder()
            ->first();

$query = DB::table('users')->orderBy('name');

$unorderedUsers = $query->reorder()->get();
```

You may pass a column and direction when calling the reorder method in order to remove all existing "order by" clauses and apply an entirely new order to the query:

```
$query = DB::table('users')->orderBy('name');

$usersOrderedByEmail = $query->reorder('email', 'desc')->get();

$users = DB::table('users')
            ->groupBy('account_id')
            ->having('account_id', '>', 100)
            ->get();
```

You can use the havingBetween method to filter the results within a given range:

```
$report = DB::table('orders')
            ->selectRaw('count(id) as number_of_orders, customer_id')
            ->groupBy('customer_id')
            ->havingBetween('number_of_orders', [5, 15])
```

```
        ->get();
```

You may pass multiple arguments to the groupBy method to group by multiple columns:

```
$users = DB::table('users')
        ->groupBy('first_name', 'status')
        ->having('account_id', '>', 100)
        ->get();

$users = DB::table('users')->skip(10)->take(5)->get();
```

Alternatively, you may use the limit and offset methods. These methods are functionally equivalent to the take and skip methods, respectively:

```
$users = DB::table('users')
        ->offset(10)
        ->limit(5)
        ->get();

$role = $request->string('role');

$users = DB::table('users')
        ->when($role, function (Builder $query, string $role) {
            $query->where('role_id', $role);
        })
        ->get();

$sortByVotes = $request->boolean('sort_by_votes');

$users = DB::table('users')
        ->when($sortByVotes, function (Builder $query, bool $sortByVotes) {
            $query->orderBy('votes');
        }, function (Builder $query) {
            $query->orderBy('name');
        })
        ->get();

DB::table('users')->insert([
    'email' => 'kayla@example.com',
    'votes' => 0
]);

DB::table('users')->insert([
    ['email' => 'picard@example.com', 'votes' => 0],
    ['email' => 'janeway@example.com', 'votes' => 0],
]);

DB::table('users')->insertOrIgnore([
    ['id' => 1, 'email' => 'sisko@example.com'],
    ['id' => 2, 'email' => 'archer@example.com'],
```

```
]);
```

The insertUsing method will insert new records into the table while using a subquery to determine the data that should be inserted:

```
DB::table('pruned_users')->insertUsing([
    'id', 'name', 'email', 'email_verified_at'
], DB::table('users')->select(
    'id', 'name', 'email', 'email_verified_at'
)->where('updated_at', '<=', now()->subMonth()));

$id = DB::table('users')->insertGetId(
    ['email' => 'john@example.com', 'votes' => 0]
);
```

Upserts

```
DB::table('flights')->upsert(
    [
        ['departure' => 'Oakland', 'destination' => 'San Diego', 'price' => 99],
        ['departure' => 'Chicago', 'destination' => 'New York', 'price' => 150]
    ],
    ['departure', 'destination'],
    ['price']
);

$affected = DB::table('users')
              ->where('id', 1)
              ->update(['votes' => 1]);

DB::table('users')
    ->updateOrInsert(
        ['email' => 'john@example.com', 'name' => 'John'],
        ['votes' => '2']
    );
```

You may provide a closure to the updateOrInsert method to customize the attributes that are updated or inserted into the database based on the existence of a matching record:

```
DB::table('users')->updateOrInsert(
    ['user_id' => $user_id],
    fn ($exists) => $exists ? [
        'name' => $data['name'],
        'email' => $data['email'],
    ] : [
        'name' => $data['name'],
        'email' => $data['email'],
        'marketable' => true,
    ],
);
```

```
$affected = DB::table('users')
            ->where('id', 1)
            ->update(['options->enabled' => true]);

DB::table('users')->increment('votes');

DB::table('users')->increment('votes', 5);

DB::table('users')->decrement('votes');

DB::table('users')->decrement('votes', 5);
```

If needed, you may also specify additional columns to update during the increment or decrement operation:

```
DB::table('users')->increment('votes', 1, ['name' => 'John']);
```

In addition, you may increment or decrement multiple columns at once using the incrementEach and decrementEach methods:

```
DB::table('users')->incrementEach([
    'votes' => 5,
    'balance' => 100,
]);

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

$deleted = DB::table('users')->where('votes', '>', 100)->delete();
```

If you wish to truncate an entire table, which will remove all records from the table and reset the auto-incrementing ID to zero, you may use the truncate method:

```
DB::table('users')->truncate();

DB::table('users')->where('votes', '>', 100)->dd();

DB::table('users')->where('votes', '>', 100)->dump();

DB::table('users')->where('votes', '>', 100)->dumpRawSql();

DB::table('users')->where('votes', '>', 100)->ddRawSql();
```

3.4 – Migrations

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

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

Criando mirgations para tabelas

php artisan make:migration create_products_table

Relacionamentos

Criar arquivo com todas as migrations de um banco de dados
php artisan schema:dump --prune

Criará o arquivo
database/schema/mariadb-schema.sql

Arquivo básico de migration

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

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->decimal('price', 8,2);
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::drop('products');
    }
};
```

Executar migration
php artisan migrate

php artisan migrate:status

Desfazer e refazer todas as migrations
php artisan migrate:refresh

E rodar seeder
php artisan migrate:refresh --seed

Criando migration para modificar tabelas

Modificando uma tabela adicionando um campo
Mudar a tabela users adicionando o campo role

php artisan make:migration add_type_to_users_table

```
public function up(): void
{
    Schema::table('users', function (Blueprint $table) {
        $table->enum('role',['admin','agent','user'])->default('user');
    });
}

public function down(): void
{
    Schema::table('users', function (Blueprint $table) {
        //
    });
}
```

Tipos de campos nas migrations

O blueprint do schema builder oferece uma variedade de métodos que correspondem aos diferentes tipos de colunas que você pode adicionar às suas tabelas de banco de dados. Cada um dos métodos disponíveis está listado na tabela abaixo:

bigIncrements
bigInteger
binary
boolean
char
dateTimeTz
dateTime
date
decimal
double
enum
float

foreignId
foreignIdFor
foreignUlid
foreignUuid
geography
geometry
id
increments
integer
ipAddress
json
jsonb
longText
macAddress
mediumIncrements
mediumInteger
mediumText
morphs
nullableMorphs
nullableTimestamps
nullableUlidMorphs
nullableUuidMorphs
rememberToken
set
smallIncrements
smallInteger
softDeletesTz
softDeletes
string
text
timeTz
time
timestampTz
timestamp
timestampsTz
timestamps
tinyIncrements
tinyInteger
tinyText
unsignedBigInteger
unsignedInteger
unsignedMediumInteger
unsignedSmallInteger
unsignedTinyInteger
ulidMorphs
uuidMorphs
ulid
uuid
year
https://laravel.com/docs/11.x/migrations#available-column-types33

3.5 - Seeders no Laravel 11

O Laravel introduz um seeder para criar dados de teste ou padrões. Se você tem um pequeno projeto de administração, pode criar um usuário administrador e definir dados padrão para tabelas usando seeders.

- Seeder manual

- Seeder usando faker

php artisan make:seeder AdminUserSeeder

Com isso ele criará o arquivo

database/seeders;AdminUserSeeder.php

```
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\User;

class AdminUserSeeder extends Seeder
{
   public function run()
   {
                User::create([
                   'name' => 'Hardik',
                   'email' => 'admin@gmail.com',
                   'password' => bcrypt('123456'),
                ]);
   }
```

No DatabaseSeeder

```
   $this->call([
      UserSeeder::class,
   ]);
```

Com a estrutura básica de um seeder. Então edite e altere assim:

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use App\Models\User;

class AdminUserSeeder extends Seeder
{
   /**
    * Run the database seeds.
```

```
     */
    public function run(): void
    {
        // Se for apenas um único registro
                User::create([
                    'name' => 'Hardik',
                    'email' => 'admin@gmail.com',
                    'password' => bcrypt('123456'),
                ]);
    }
}
```

Criará um novo user na tabela users.

```
    public function run(): void
    {
                // Para vários registros
                $users = [
                    [
                                'name'=>'Javed Ur Rehman',
                                'email'=>'javed@allphptricks.com',
                                'password'=> Hash::make('javed1234')
                    ],
                    [
                                'name'=>'Syed Ahsan Kamal',
                                'email'=>'ahsan@allphptricks.com',
                                'password'=> Hash::make('ahsan1234')
                    ],
                    [
                                'name'=>'Abdul Muqeet',
                                'email'=>'a.muqeet@allphptricks.com',
                                'password'=> Hash::make('muqeet1234')
                ]
                ];

                // Looping and Inserting Array's Users into User Table
                foreach($users as $user){
                    User::create($user);
                }
```

Precisamos declarar nosso seeder na classe DatabaseSeeder

database/seeders/DatabaseSeeder.php

```
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;

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

Executar este seeder

php artisan db:seed --class=AdminUserSeeder

Executar todos os seeders

php artisan db:seed

https://www.itsolutionstuff.com/post/laravel-11-database-seeder-example-tutorialexample.html

Outra forma é mudar diretamente o DatabaseSeeder

Como exemplo, vamos modificar a classe DatabaseSeeder padrão e adicionar uma instrução de inserção de banco de dados ao método run

```
<?php
namespace Database\Seeders;

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

class DatabaseSeeder extends Seeder
{
   public function run(): void
   {
                DB::table('users')->insert([
                   'name' => Str::random(10),
                   'email' => Str::random(10).'@example.com',
                   'password' => Hash::make('password'),
                ]);
   }
}
```

Seeder com faker

```
use Illuminate\Database\Seeder;
use Faker\Factory as Faker;

class YourSeederName extends Seeder
{
   public function run()
   {
                $faker = Faker::create();

                foreach (range(1, 20) as $index) {
                   DB::table('users')->insert([
                                'name' => $faker->name,
                                'email' => $faker->unique()->safeEmail,
                                'password' => bcrypt('password'),
                   ]);
                }
   }
}

php artisan db:seed --class=YourSeederName

                return [
                   "name" => $this->faker->name(),
                   "email" => $this->faker->unique()->safeEmail(),
                   "username" => $this->faker->unique()->userName(),
                   "mobile_no" => $this->faker->phoneNumber(),
                   "password" => Hash::make("Password"),
                   "email_verified_at" => now(),
                   "remember_token" => Str::random(10)
                ];

namespace Database\Seeders;

use Faker\Factory;
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use DB;

class ArticlesTableSeeder extends Seeder
{
   /**
    * Run the database seeds.
    */
   public function run(): void
   {
                $faker = Factory::create();
      for ($i = 0; $i < 50; $i++) {
         DB::table('articles')->insert([
```

```
            [
                'title' => $faker->sentence,
                'body' => $faker->paragraph
            ]
        ]);
    }
  }
}
```

Faker

“Seeding” e “Faker” são conceitos relacionados usados para preencher suas tabelas de banco de dados com dados de amostra ou fictícios, especialmente para fins de teste e desenvolvimento.

Seeding

Seeding é o processo de inserir dados predefinidos em suas tabelas de banco de dados. Ele permite que você preencha seu banco de dados com dados consistentes e específicos, facilitando a configuração de um ambiente consistente para teste ou desenvolvimento. O Laravel fornece uma maneira conveniente de criar e executar seeders, que são classes que definem os dados que você deseja inserir em suas tabelas.

Você pode usar seeders para criar usuários iniciais, funções, produtos ou quaisquer outros dados necessários para seu aplicativo. Os seeders são úteis para preencher seu banco de dados com dados padrão, facilitando o início do seu aplicativo.

Faker

Faker é uma biblioteca PHP que gera dados falsos para você. Ela fornece uma maneira de criar dados aleatórios e de aparência realista, como nomes, endereços, e-mails e muito mais. No Laravel, a biblioteca Faker é frequentemente usada em conjunto com seeders para gerar esses dados falsos para preencher suas tabelas de banco de dados. O Faker facilita a criação de uma variedade de dados de teste que simulam cenários do mundo real. Por exemplo, você pode usar o Faker para gerar nomes aleatórios, endereços de e-mail, datas e outros atributos para preencher suas tabelas de banco de dados com informações diversas e realistas.

Ao combinar seeding e Faker no Laravel, você pode criar seeders que usam o Faker para preencher suas tabelas de banco de dados com dados aleatórios, mas realistas, facilitando o desenvolvimento e o teste de seus aplicativos. Isso ajuda a garantir que seu aplicativo funcione conforme o esperado ao encontrar diferentes tipos de dados. Veja como usar seeding e Faker no Laravel:

faker atualmente já vem instalado nativamente no laravel

Alguns dos métodos do Faker

```
'name' => $faker->name,
'email' => $faker->unique()->safeEmail,
'password' => bcrypt('password'),

'age' => $faker->numberBetween(18, 60),
'address' => $faker->address,
'notifications' => $faker->boolean,

$arabicName = $faker->name;
$arabicEmail = $faker->email;
$arabicAddress = $faker->address;
$arabicPhoneNumber = $faker->phoneNumber;

'title' => $this->faker->words(5, true),
'body' => $this->faker->sentence(45),
'likes' => $this->faker->randomNumber(5),

'name' => $faker->company,
'country' => $faker->country,
'website' => $faker->domainName,
'foundeddate' => $faker->dateTimeThisDecade,
'crmregistered' => $faker->dateTimeThisYear,

'comments' => $faker->realText($maxNbChars = 300, $indexSize = 2),
      'insurancenumber' => $faker->numberBetween($min = 80000, $max = 150000),
      'data1' => $faker->boolean($chanceOfGettingTrue = 50),
      'type_id' => $faker->numberBetween($min = 1, $max = 2),
      'user_id' => $faker->numberBetween($min = 3, $max = 15),

     'guests_no' => $faker->numberBetween($min = 50, $max = 500),
      'days_no' => $faker->numberBetween($min = 3, $max = 7),
      'value' => $faker->ean8,
'arrival_date' => $faker->dateTimeInInterval($startDate = '+1 years', $interval = '+ 14 days',
$timezone = null),
      'departure_date' => $faker->dateTimeInInterval($startDate = '+1 years', $interval = '+ 21 days',
$timezone = null),
      'file_name' => $faker->numerify('dossier ######'),
      'facture_number' => $faker->creditCardNumber,
      'guide' => $faker->boolean($chanceOfGettingTrue = 50),
      'rentacar' => $faker->boolean($chanceOfGettingTrue = 50),
      'aerial' => $faker->boolean($chanceOfGettingTrue = 50),
      'user_id' => $faker->numberBetween($min = 3, $max = 15),
      'client_id' => $faker->numberBetween($min = 1, $max = 1000),
      'transport_type_id' => $faker->numberBetween($min = 1, $max = 3),
      'created_at' => $faker->dateTimeThisYear,
      'payment_status_id' => $faker->numberBetween($min = 1, $max = 3),
      'query_status_id' => $faker->numberBetween($min = 1, $max = 7),
      'query_type_id' => $faker->numberBetween($min = 1, $max = 3),
```

```
'isActive' => '1',
'notified' => $faker->boolean($chanceOfGettingTrue = 50),
'text' => $faker->realText($maxNbChars = 300, $indexSize = 2),

    'code' => fake()->unique()->bothify('?????-#####'),
    'name' => fake()->name(),
    'quantity' => fake()->randomNumber(2, true),
    'price' => fake()->randomFloat(2, 20, 90),
    'description' => fake()->text(),
```

3.6 – Console Artisan

Artisan é a interface de linha de comando incluída no Laravel. Artisan existe na raiz do seu aplicativo como o script artisan e fornece uma série de comandos úteis que podem ajudar você enquanto constrói seu aplicativo. Para visualizar uma lista de todos os comandos Artisan disponíveis, você pode usar:

php artisan list

Cada comando também inclui uma tela de "ajuda" que exibe e descreve os argumentos e opções disponíveis do comando. Para visualizar uma tela de ajuda, preceda o nome do comando com help:

php artisan help migrate

php artisan make:migration create_flights_table

Tinker

Laravel Tinker é um REPL poderoso para o framework Laravel. Já vem instalado por default no laravel 11

Uso

O Tinker permite que você interaja com todo o seu aplicativo Laravel na linha de comando, incluindo seus modelos Eloquent, jobs, eventos e muito mais. Para entrar no ambiente Tinker, execute o comando tinker Artisan:

php artisan tinker

Você pode publicar o arquivo de configuração do Tinker usando o comando vendor:publish:

php artisan vendor:publish --provider="Laravel\Tinker\TinkerServiceProvider"

Criando comandos

php artisan make:command SendEmails

php artisan make:model ModelName -a

-a or — all Generate a migration, seeder, factory, and resource controller for the model
-c or — controller Create a new controller for the model
— force Create the class even if the model already exists
-m or — migration Create a new migration file for the model

php artisan make:model Todo -mcr

if you run php artisan make:model --help you can see all the available options

-m, --migration Create a new migration file for the model.

-c, --controller Create a new controller for the model.
-r, --resource Indicates if the generated controller should be a resource controller

Updated

Laravel 6 or Later

Through the model

To Generate a migration, seeder, factory and resource controller for the model

php artisan make:model Todo -a

Or

php artisan make:model Todo -all

Other Options

-c, --controller Create a new controller for the model

-f, --factory Create a new factory for the model

--force Create the class even if the model already exists

-m, --migration Create a new migration file for the model

-s, --seed Create a new seeder file for the model

-p, --pivot Indicates if the generated model should be a custom intermediate table model

-r, --resource Indicates if the generated controller should be a resource controller

For More Help

php artisan make:model Todo -help

php artisan make:model --migration --controller test

php artisan make:model --migration --controller test --resource

hp artisan make:model Author -cfmsr

-c, --controller Create a new controller for the model

-f, --factory Create a new factory for the model

-m, --migration Create a new migration file for the model

-s, --seed Create a new seeder file for the model

-r, --resource Indicates if the generated controller should be a resource controller

php artisan make:model Product -mcrf

It will create:

Model: Product model in app/Models.
Migration: Migration file for products table in database/migrations.
Controller: ProductController in app/Http/Controllers.
Factory: ProductFactory in database/factories.

php artisan make:migration create_table_name

php artisan migrate

php artisan make:controller ModelNameController — resource

3.7 – Tinker

Esta ferramenta poderosa permite que você interaja com seu aplicativo Laravel diretamente da linha de comando, tornando-a uma ferramenta inestimável para depuração, teste e exploração da funcionalidade do seu aplicativo.

O que é php artisan tinker?

php artisan tinker é uma interface de linha de comando incluída no Laravel que permite que você interaja com o código e os dados do seu aplicativo diretamente da linha de comando. É essencialmente um REPL (read-eval-print loop) que permite que você execute código PHP dentro do contexto do seu aplicativo.

Por que usar php artisan tinker?

Existem vários benefícios em usar tinker, incluindo:

Acesso rápido e fácil ao código e aos dados do seu aplicativo
Recursos poderosos de depuração e teste
A capacidade de criar e modificar dados rapidamente no seu banco de dados
Interface simples e intuitiva para explorar a funcionalidade do seu aplicativo

Abrindo a console do tinker

php artisan tinker

>>> echo "Olá, Tinker!";

A partir daqui, você pode começar a interagir com o código e os dados do seu aplicativo usando uma variedade de comandos. Alguns dos comandos mais comumente usados incluem:

help: exibe uma lista de comandos disponíveis
exit: sai do ambiente tinker
dd(): despeja a saída de uma variável e encerra a execução
DB::table('table_name')->get(): consulta dados da tabela especificada no seu banco de dados

Interagindo com Eloquent.

> App\Models\User::count();
= 0

Interagindo com helpers.

> env('PLATAFORM1_URL');
= "http://localhost:8001/plataform1/api"

Interagindo com facades e queues.

> $response = Http::get('http://localhost:8000/plataform1/api/v1/pr

```
oducts');
= Illuminate\Http\Client\Response {#6467
    +cookies: GuzzleHttp\Cookie\CookieJar {#6437},
    +transferStats: GuzzleHttp\TransferStats {#6515},
  }

> $response->status();
= 401
```

Interagindo com funções.

```
function isPrime($number) {
    if ($number <= 1) {
        return false;
    }

    for ($i = 2; $i <= sqrt($number); $i++) {
        if ($number % $i === 0) {
            return false;
        }
    }

    return true;
}

isPrime(97);
```

Criar

```
$user=App\Models\
User::create(['name'=>'Ribamar','email'=>'ribafs@gmail.com','password'=>'minhanpass']);
= App\Models\User {#5988
    name: "Ribamar",
    email: "ribafs@gmail.com",
    #password: "$2y$12$HigOyVUUhLmyLrRCxcpnPelvc6kyBupX/eKrwReRtfta72.6WQDiy",
    updated_at: "2024-08-15 21:42:50",
    created_at: "2024-08-15 21:42:50",
    id: 1,
  }
```

Receber

```
App\Models\User::find(1)
```

Atualizar

```
$user = App\Models\User::find(1);
$user->name = 'Elias EF';
$user->email = 'elias@gmail.com';
$user->password = 'minhasenha';
```

```
$user->save();

$user::find(1)
```

Excluir

```
$user->delete()

$user::find(1)

>>> $name = "John Doe";
>>> strtoupper($name); // Test the strtoupper function

>>> $user = App\Models\User::find(1); // Find a specific user
>>> $user->name = "Jane Smith"; // Change the user's name
>>> $user->save(); // Save the changes

>>> DB::table('posts')->where('id', 10)->get(); // Get a specific post
>>> DB::insert('users', ['name' => 'New User']); // Insert a new user record
```

Dados fake

```
php artisan tinker

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

php artisan make:factory MovieFactory --model=Movie

database/factories/MovieFactory.php

<?php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;

class MovieFactory extends Factory
{
   public function definition(): array
   {
      return [
         'title' => $this->faker->sentence,
         'country' => $this->faker->country,
         'release_date' => $this->faker->dateTimeBetween('-40 years', 'now'),
      ];
   }
}

php artisan make:seeder MovieSeeder
```

```
<?php
namespace Database\Seeders;

use App\Models\Movie;
use Illuminate\Database\Seeder;

class MovieSeeder extends Seeder
{
    public function run(): void
    {
        Movie::factory()->count(10000)->create();
    }
}
```

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

3.8 - Blade

Blade é o mecanismo de template simples, mas poderoso, que está incluso no Laravel. Ao contrário de alguns mecanismos de template PHP, o Blade não o restringe de usar código PHP simples em seus templates. Na verdade, todos os templates Blade são compilados em código PHP simples e armazenados em cache até serem modificados, o que significa que o Blade adiciona essencialmente zero overhead ao seu aplicativo. Os arquivos de template Blade usam a extensão de arquivo .blade.php e são normalmente armazenados no diretório resources/views.

As views Blade podem ser retornadas de rotas ou controladores usando o helper de view global. Claro, como mencionado na documentação sobre views, os dados podem ser passados para a view Blade usando o segundo argumento do helper de view na rota ou controller

Na rota default

```
Route::get('/', function () {
   return view('welcome', ['name' => 'Riba']);
});
```

Ele enviará a segunda parte ['name' => 'Riba'], para a view

Na view welcome.blade.php

No body

```
   <div>Meu  nome é {{ $name }} </div>
```

Por padrão, as instruções Blade {{ }} são enviadas automaticamente por meio da função htmlspecialchars do PHP para evitar ataques XSS. Se você não quiser que seus dados sejam escapados, você pode usar a seguinte sintaxe:

As instruções de eco {{ }} do Blade são enviadas automaticamente por meio da função htmlspecialchars do PHP para evitar ataques XSS.

O registro de data e hora UNIX atual é {{ time() }}.

Comentários no Blade

{!! comentário !!}

Se queremos que o HTML em um campo seja processado, então usamos:

<td> {!! $post->content !!} </td>

Diretivas Blade

```
@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

Blade also provides an @**unless** directive:

```
@unless (Auth::check())
    You are not signed in.
@endunless
```

```
@isset($records)
    // $records is defined and is not null...
@endisset
```

```
{{ isset($post->content) ? $post->content : ''}}
```

```
@empty($records)
    // $records is "empty"...
@endempty
```

```
@auth
    // The user is authenticated...
@endauth

@guest
    // The user is not authenticated...
@endguest
```

Ou você pode determinar se o aplicativo está sendo executado em um ambiente específico usando a diretiva @env:

```
@env('staging')
    // The application is running in "staging"...
@endenv

@env(['staging', 'production'])
    // The application is running in "staging" or "production"...
@endenv
```

```
@session('status')
    <div class="p-4 bg-green-100">
        {{ $value }}
    </div>
@endsession


@switch($i)
    @case(1)
        First case...
        @break

    @case(2)
        Second case...
        @break

    @default
        Default case...
@endswitch


@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile


@foreach ($users as $user)
    @if ($user->type == 1)
        @continue
    @endif

    <li>{{ $user->name }}</li>

    @if ($user->number == 5)
        @break
    @endif
```

```
@endforeach


@foreach ($users as $user)
    @continue($user->type == 1)

    <li>{{ $user->name }}</li>

    @break($user->number == 5)
@endforeach


@foreach ($users as $user)
    @if ($loop->first)
        This is the first iteration.
    @endif

    @if ($loop->last)
        This is the last iteration.
    @endif

    <p>This is user {{ $user->id }}</p>
@endforeach


@foreach ($users as $user)
    @foreach ($user->posts as $post)
        @if ($loop->parent->first)
            This is the first iteration of the parent loop.
        @endif
    @endforeach
@endforeach


@php
    $isActive = false;
    $hasError = true;
@endphp

<span @class([
    'p-4',
    'font-bold' => $isActive,
    'text-gray-500' => ! $isActive,
    'bg-red' => $hasError,
])></span>

<span class="p-4 text-gray-500 bg-red"></span>

@php
    $isActive = true;
```

```
@endphp

<span @style([
   'background-color: red',
   'font-weight: bold' => $isActive,
])></span>

<span style="background-color: red; font-weight: bold;"></span>


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


<select name="version">
   @foreach ($product->versions as $version)
      <option value="{{ $version }}" @selected(old('version') == $version)>
         {{ $version }}
      </option>
   @endforeach
</select>


<input type="email"
       name="email"
       value="email@laravel.com"
       @readonly($user->isNotAdmin()) />


<input type="text"
       name="title"
       value="title"
       @required($user->isAdmin()) />
```

Including Subviews

```
<div>
   @include('shared.errors')

   <form>
      <!-- Form Contents -->
   </form>
</div>


@once
   @push('scripts')
      <script>
         // Your custom JavaScript...
      </script>
```

```
    @endpush
@endonce


@pushOnce('scripts')
    <script>
        // Your custom JavaScript...
    </script>
@endPushOnce
```

PHP

```
@php
    $counter = 1;
@endphp


@use('App\Models\Flight', 'FlightModel')


@push('scripts')
    <script src="/example.js"></script>
@endpush


<head>
    <!-- Head Contents -->

    @stack('scripts')
</head>
```

3.9 - FileSystem

```
use File;

File::makeDirectory($path);

Ou

$path = public_path().'/images/article/imagegallery/' . $galleryId;
File::makeDirectory($path, $mode = 0777, true, true);

use Illuminate\Support\Facades\Storage;
if(!Storage::exists($path)) {
   Storage::makeDirectory($path); //creates directory
}

use Illuminate\Support\Facades\File;
if(!File::exists($path)) {
   File::makeDirectory($path, 0777, true); //creates directory
}

if (!\File::exists( storage_path("profile/" . $user->id . "/avatar") )) {
   \File::makeDirectory($invoice_path, 0755, true);
}

Storage::disk($disk)->delete('path/to/file.txt');

use Illuminate\Http\Request;

Route::post('process', function (Request $request) {
   $path = $request->file('photo')->store('photos');

   dd($path);
});

Create a new file with contents
Storage::put('file.txt', 'Contents');

Check if file exists
Storage::exists('file.txt')

Get file size
Storage::size('file.txt');

Last modified date
Storage::lastModified('file.txt')

Copy files
Storage::copy('file.txt', 'shared/file.txt');
```

```
Move files
Storage::move('file.txt', 'secret/file.txt');

Delete files
Storage::delete('file.txt');

// to delete multiple files
Storage::delete(['file1.txt', 'file2.txt']);

Make a directory
Storage::makeDirectory($directory_name);

Delete a directory
Storage::deleteDirectory($directory_name);

use Illuminate\Support\Facades\File

static bool copy(string $path, string $target)

static bool copyDirectory(string $directory, string $destination, int|null $options = null)

static bool delete(string|array $paths)

static bool deleteDirectories(string $directory)

static bool deleteDirectory(string $directory, bool $preserve = false)

static bool exists(string $path)

static bool isDirectory(string $directory)

static bool isFile(string $file)

static bool isReadable(string $path)

static bool isWritable(string $path)

static bool move(string $path, string $target)

static bool moveDirectory(string $from, string $to, bool $overwrite = false)

static string basename(string $path)

static string dirname(string $path)

static string extension(string $path)

static string hash(string $path)

use Illuminate\Support\Facades\File;
```

```
\File::copyDirectory( public_path . 'to/the/app', resource_path('to/the/app'));

File::copyDirectory(__DIR__.'/form-directory', resource_path('to-directory'));

File::copy(from_path, to_path);
```

PHP puro

```
copy($sourceFilePath, $destinationFilePath);

rename($oldFilePath, $newFilePath);

bool unlink ( string $filename [, resource $context ] )
unlink('test.php');

   $filename2 = $filename . '.old';
   $result = rename($filename, $filename2);
   if ($result) {
         print "$filename has been renamed to $filename2.\n";
   } else {
         print "Error: couldn't rename $filename to $filename2!\n";
   }
```

3.10 - Middleware

Middleware fornece um mecanismo conveniente para inspecionar e filtrar solicitações HTTP que entram em seu aplicativo. Por exemplo, o Laravel inclui um middleware que verifica se o usuário do seu aplicativo está autenticado. Se o usuário não estiver autenticado, o middleware redirecionará o usuário para a tela de login do seu aplicativo. No entanto, se o usuário estiver autenticado, o middleware permitirá que a solicitação prossiga para o aplicativo.

Middleware adicional pode ser escrito para executar uma variedade de tarefas além da autenticação. Por exemplo, um middleware de registro pode registrar todas as solicitações recebidas em seu aplicativo. Uma variedade de middleware está incluída no Laravel, incluindo middleware para autenticação e proteção CSRF; no entanto, todos os middlewares definidos pelo usuário estão normalmente localizados no diretório do seu aplicativo

app/Http/Middleware

Um exemplo pode ser visto no aplicativo multi-auth

Role.php

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

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class Role
{
   public function handle(Request $request, Closure $next, $role): Response
   {
      if ($request->user()->role != $role) {
         return redirect('dashboard');
      }
      return $next($request);
   }
}
```

Criando com artisan

php artisan make:middleware MeuMiddle

Um middleware pode executar tarefas antes ou depois de passar a solicitação mais profundamente no aplicativo. Por exemplo, o seguinte middleware executaria alguma tarefa antes que a solicitação fosse manipulada pelo aplicativo

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

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class BeforeMiddleware
{
   public function handle(Request $request, Closure $next): Response
   {
      // Perform action
      return $next($request);
   }
}
```

Entretanto, esse middleware executaria sua tarefa depois que a solicitação fosse tratada pelo aplicativo:

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

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class AfterMiddleware
{
   public function handle(Request $request, Closure $next): Response
   {
      $response = $next($request);

      // Perform action
      return $response;
   }
}
```

3.11 - Request

A classe Illuminate\Http\Request do Laravel fornece uma maneira orientada a objetos de interagir com a solicitação HTTP atual que está sendo manipulada pelo seu aplicativo, bem como recuperar a entrada, os cookies e os arquivos que foram enviados com a solicitação.

Acessando o Request/solicitação

Para obter uma instância da solicitação/Request HTTP atual por meio de injeção de dependência, você deve dar uma dica de tipo para a classe Illuminate\Http\Request no seu método de fechamento de rota ou controlador. A instância da solicitação/Request de entrada será injetada automaticamente pelo contêiner de serviço do Laravel:

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

use Illuminate\Http\RedirectResponse;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public function store(Request $request): RedirectResponse
    {
        $name = $request->input('name');
        // Store the user...
        return redirect('/users');
    }
}
```

Vejamos como o laravel recebe as informações vindas da view create no método store() do controller

```
    public function store(Request $request)
    {
        $requestData = $request->all();
        dd($requestData);
```

Ao executar vemos no navegador:

```
array:3 [▼ // app/Http/Controllers/ProductsController.php:34
  "_token" => "V4DS3Piynt31YexmsvxpCMI50VxGelgWuWyMZKvY"
  "name" => "teste1"
  "price" => "1"
]
```

Um array com os dois campos do form mais o token

Então, para receber apenas oo name, podemos fazer assim:

```
dd($requestData['name']);
```

Receberemos:
"teste1" // app/Http/Controllers/ProductsController.php:34

Obs.: o request leva somente campos do form e o token, nada mais.
Adicionando mis um campo ao form ele também foi capturado pelo store().

Recuperando o Request Path

O método path retorna as informações do caminho da solicitação. Então, se a solicitação de entrada for direcionada para http://example.com/products, o método path retornará products:

```
$uri = $request->path();
```

Inspecting the Request Path / Route
The is method allows you to verify that the incoming request path matches a given pattern. You may use the * character as a wildcard when utilizing this method:

```
if ($request->is('admin/*')) {
    // ...
}
```

Using the routeIs method, you may determine if the incoming request has matched a named route:

```
if ($request->routeIs('admin.*')) {
    // ...
}
```

Recuperando a Request URL

Para recuperar a URL completa da solicitação recebida, você pode usar os métodos url ou fullUrl. O método url retornará a URL sem a string de consulta, enquanto o método fullUrl inclui a string de consulta:

```
$url = $request->url(); -- http://127.0.0.1:8000/products

$urlWithQueryString = $request->fullUrl();
```

Se você quiser anexar dados da string de consulta à URL atual, você pode chamar o método fullUrlWithQuery. Este método mescla a matriz fornecida de variáveis de string de consulta com a string de consulta atual:

$request->fullUrlWithQuery(['type' => 'phone']);
Se você quiser obter a URL atual sem um parâmetro de string de consulta fornecido, você pode utilizar o método fullUrlWithoutQuery:
$request->fullUrlWithoutQuery(['type']);

Retrieving the Request Host
You may retrieve the "host" of the incoming request via the host, httpHost, and schemeAndHttpHost methods:

```
$request->host(); // 127.0.0.1
$request->httpHost(); // 127.0.0.1:8000
$request->schemeAndHttpHost(); // http://127.0.0.1:8000
```

Retrieving the Request Method

The method method will return the HTTP verb for the request. You may use the isMethod method to verify that the HTTP verb matches a given string:

```
$method = $request->method();

if ($request->isMethod('post')) {
    true
}
```

Request IP Address

The ip method may be used to retrieve the IP address of the client that made the request to your application:

```
$ipAddress = $request->ip();
```

If you would like to retrieve an array of IP addresses, including all of the client IP addresses that were forwarded by proxies, you may use the ips method. The "original" client IP address will be at the end of the array:

```
$ipAddresses = $request->ips();
```

In general, IP addresses should be considered untrusted, user-controlled input and be used for informational purposes only.

Input

Retrieving Input

Retrieving All Input Data

You may retrieve all of the incoming request's input data as an array using the all method. This method may be used regardless of whether the incoming request is from an HTML form or is an XHR request:

```
$input = $request->all();
```

Using the collect method, you may retrieve all of the incoming request's input data as a collection:

```
$input = $request->collect();
```

The collect method also allows you to retrieve a subset of the incoming request's input as a collection:

```
$request->collect('users')->each(function (string $user) {
    // ...
});
```

Retrieving an Input Value

Using a few simple methods, you may access all of the user input from your Illuminate\Http\Request instance without worrying about which HTTP verb was used for the request. Regardless of the HTTP verb, the input method may be used to retrieve user input:

```
$name = $request->input('name');
```

You may pass a default value as the second argument to the input method. This value will be returned if the requested input value is not present on the request:

```
$name = $request->input('name', 'Sally');
```

When working with forms that contain array inputs, use "dot" notation to access the arrays:

```
$name = $request->input('products.0.name');

$names = $request->input('products.*.name');
```

You may call the input method without any arguments in order to retrieve all of the input values as an associative array:

```
$input = $request->input();
```

Retrieving a Portion of the Input Data

If you need to retrieve a subset of the input data, you may use the only and except methods. Both of these methods accept a single array or a dynamic list of arguments:

```
$input = $request->only(['username', 'password']);

$input = $request->only('username', 'password');

$input = $request->except(['credit_card']);

$input = $request->except('credit_card');
```

The only method returns all of the key / value pairs that you request; however, it will not return key / value pairs that are not present on the request.

Cookies
Retrieving Cookies From Requests

All cookies created by the Laravel framework are encrypted and signed with an authentication code, meaning they will be considered invalid if they have been changed by the client. To retrieve a cookie value from the request, use the cookie method on an Illuminate\Http\Request instance:

```
$value = $request->cookie('name');
```

Files
Retrieving Uploaded Files

You may retrieve uploaded files from an Illuminate\Http\Request instance using the file method or using dynamic properties. The file method returns an instance of the Illuminate\Http\UploadedFile class, which extends the PHP SplFileInfo class and provides a variety of methods for interacting with the file:

```
$file = $request->file('photo');

$file = $request->photo;
```

You may determine if a file is present on the request using the hasFile method:

```
if ($request->hasFile('photo')) {
    // ...
}
```

File Paths and Extensions

The UploadedFile class also contains methods for accessing the file's fully-qualified path and its extension. The extension method will attempt to guess the file's extension based on its contents. This extension may be different from the extension that was supplied by the client:

```
$path = $request->photo->path();

$extension = $request->photo->extension();
```

Mais detalhes
https://laravel.com/docs/11.x/requests#introduction

3.12 - Response

Todas as rotas e controladores devem retornar uma Response a ser enviada de volta ao navegador do usuário. O Laravel fornece várias maneiras diferentes de retornar Responses/respostas. A resposta/Response mais básica é retornar uma string de uma rota ou controlador. O framework converterá automaticamente a string em uma resposta HTTP completa:

```
Route::get('/', function () {
    return 'Hello World';
});
```

Além de retornar strings de suas rotas e controladores, você também pode retornar arrays. O framework converterá automaticamente o array em uma resposta JSON:

```
Route::get('/', function () {
    return [1, 2, 3];
});
```

Objetos de resposta

Normalmente, você não retornará apenas strings ou arrays simples de suas ações de rota. Em vez disso, você retornará instâncias ou visualizações completas de

Illuminate\Http\Response

```
Route::get('/home', function () {
    return response('Hello World', 200)->header('Content-Type', 'text/plain');
});
```

```
use App\Models\User;

Route::get('/user/{user}', function (User $user) {
    return $user;
});
```

Redirecionamentos

Respostas de redirecionamento são instâncias da classe Illuminate\Http\RedirectResponse e contêm os cabeçalhos apropriados necessários para redirecionar o usuário para outra URL. Há várias maneiras de gerar uma instância RedirectResponse. O método mais simples é usar o auxiliar de redirecionamento global:

```
Route::get('/dashboard', function () {
    return redirect('/home/dashboard');
});
```

Às vezes, você pode desejar redirecionar o usuário para o local anterior, como quando um formulário enviado é inválido. Você pode fazer isso usando a função global back helper. Como esse

recurso utiliza a sessão, certifique-se de que a rota que chama a função back esteja usando o grupo de middleware da web:

```
Route::post('/user/profile', function () {
    // Validate the request...

    return back()->withInput();
});

return redirect()->route('login');

// For a route with the following URI: /profile/{id}

return redirect()->route('profile', ['id' => 1]);

// For a route with the following URI: /profile/{id}

return redirect()->route('profile', [$user]);

return redirect()->away('https://www.google.com');
```

Redirecionando com dados de sessão flasheados

Redirecionar para uma nova URL e flashear dados para a sessão geralmente são feitos ao mesmo tempo. Normalmente, isso é feito após executar com sucesso uma ação quando você flasheia uma mensagem de sucesso para a sessão. Para sua conveniência, você pode criar uma instância RedirectResponse e flashear dados para a sessão em uma única cadeia de métodos fluentes:

```
Route::post('/user/profile', function () {
    // ...

    return redirect('/dashboard')->with('status', 'Profile updated!');
});
```

After the user is redirected, you may display the flashed message from the session. For example, using Blade syntax:

```
@if (session('status'))
    <div class="alert alert-success">
        {{ session('status') }}
    </div>
@endif
```

Respostas de arquivo

O método file pode ser usado para exibir um arquivo, como uma imagem ou PDF, diretamente no navegador do usuário em vez de iniciar um download. Este método aceita o caminho absoluto para o arquivo como seu primeiro argumento e uma matriz de cabeçalhos como seu segundo argumento:

```
return response()->file($pathToFile);

return response()->file($pathToFile, $headers);
```

3.13 – Rotas no Laravel 11

No Laravel, roteamento se refere ao processo de definição das rotas às quais seu aplicativo responderá. Quando uma solicitação é feita ao seu aplicativo, o sistema de roteamento do Laravel determina qual rota corresponde à solicitação e chama o método do controlador ou fechamento correspondente para gerar a resposta.

As rotas no Laravel podem ser definidas usando a fachada Route, que fornece métodos para definir vários tipos de rotas.

As rotas mais básicas do Laravel aceitam um URI e um fechamento, fornecendo um método muito simples e expressivo de definir rotas e comportamento sem arquivos de configuração de roteamento complicados:

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
        return 'Hello World';
});
```

As rotas de aplicativos no Laravel ficam em

routes/web.php

As rotas de APIs ficam em
routes/api.php

Neste exemplo, o método Route::get é usado para definir uma rota para o caminho raiz (/) do aplicativo. Quando uma solicitação GET é feita para esse caminho, a função anônima fornecida como o segundo argumento será chamada e seu valor de retorno será usado como a resposta.

Métodos de roteador disponíveis

O roteador permite que você registre rotas que respondem a qualquer verbo HTTP:

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Rotas no Laravel também podem ser definidas para responder a outros métodos HTTP, como POST, PUT e DELETE. Aqui está um exemplo de uma definição de rota para uma solicitação POST:

```
Route::post('/users', 'UserController@store');
```

Neste exemplo, o método Route::post é usado para definir uma rota para o caminho /users que responde a solicitações POST. Quando uma solicitação POST é feita para este caminho, o método store da classe UserController será chamado para manipular a solicitação.

O sistema de roteamento do Laravel também suporta parâmetros de rota dinâmicos, que permitem que você defina rotas com marcadores de posição que podem ser usados para corresponder a uma variedade de padrões de URL diferentes. Aqui está um exemplo de uma definição de rota com um parâmetro dinâmico:

```
Route::get('/users/{id}', function ($id) {
    return "User with ID {$id}";
});
```

Neste exemplo, o método Route::get é usado para definir uma rota para o caminho /users/{id}, onde {id} é um parâmetro dinâmico que pode corresponder a qualquer valor. Quando uma solicitação GET é feita para este caminho com um valor específico para o parâmetro id, a função anônima fornecida como o segundo argumento será chamada com o valor do parâmetro id como seu argumento.

No geral, o roteamento é um conceito fundamental no Laravel que fornece uma maneira poderosa de definir os pontos de extremidade do seu aplicativo e manipular solicitações HTTP de entrada.

No Laravel, o arquivo de rota padrão é routes/web.php. Este arquivo é onde você pode definir as rotas para seu aplicativo que respondem a solicitações HTTP.

O arquivo routes/web.php é carregado pelo Laravel automaticamente quando o aplicativo é iniciado e é normalmente usado para definir rotas para as partes públicas do seu aplicativo, como páginas da web, pontos de extremidade de API e rotas de autenticação.

Rotas para views

Se sua rota só precisa retornar uma visualização, você pode usar o método Route::view. Assim como o método redirect, esse método fornece um atalho simples para que você não precise definir uma rota ou controlador completo. O método view aceita um URI como seu primeiro argumento e um nome de visualização como seu segundo argumento. Além disso, você pode fornecer uma matriz de dados para passar para a visualização como um terceiro argumento opcional:

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

Aqui está um exemplo de como o arquivo padrão routes/web.php pode se parecer:

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

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

Route::get('/about', function () {
```

```
    return view('about');
});

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

// ... other routes for your application
```

Listando as rotas

php artisan route:list

Neste exemplo, o método Route::get é usado para definir rotas para o caminho raiz (/), o caminho /about e o caminho /contact. Cada uma dessas rotas retorna uma visualização quando é acessada, usando a função auxiliar de visualização para carregar o modelo blade correspondente.

Embora o arquivo routes/web.php seja o arquivo de rota padrão no Laravel, você também pode criar arquivos de rota adicionais para diferentes partes do seu aplicativo. Por exemplo, você pode criar um arquivo routes/api.php para definir rotas para seus endpoints de API ou um arquivo routes/admin.php para definir rotas para a seção admin do seu aplicativo.

Para registrar um novo arquivo de rota, você pode usar o método Route::middleware para especificar qualquer middleware que deve ser aplicado às rotas no arquivo, como este:

```
Route::middleware('api')->group(function () {
require __DIR__.'/api.php';
});
```

Neste exemplo, o método Route::middleware é usado para agrupar as rotas no arquivo api.php sob o middleware api, que aplica qualquer middleware necessário às rotas para lidar com autenticação, limitação de taxa ou outras preocupações.

No geral, o arquivo routes/web.php é o arquivo de rota padrão no Laravel e fornece uma maneira conveniente de definir as rotas para seu aplicativo que respondem a solicitações HTTP.

Métodos de roteador

O Laravel fornece vários métodos para definir rotas no arquivo routes/web.php do aplicativo. Aqui está uma visão geral de alguns dos métodos de roteador mais comumente usados:

Route::get($uri, $callback) - Define uma rota para o método HTTP GET.
Route::post($uri, $callback) - Define uma rota para o método HTTP POST.
Route::put($uri, $callback) - Define uma rota para o método HTTP PUT.
Route::patch($uri, $callback) - Define uma rota para o método HTTP PATCH.
Route::delete($uri, $callback) - Define uma rota para o método HTTP DELETE.
Route::options($uri, $callback) - Define uma rota para o método HTTP OPTIONS.
Route::any($uri, $callback) - Define uma rota que responde a qualquer método HTTP.
Route::match(['get', 'post'], $uri, $callback) - Define uma rota que responde a métodos HTTP específicos.

Além desses métodos de roteador, o Laravel também fornece vários outros métodos para trabalhar com rotas e parâmetros de rota. Aqui estão alguns exemplos:

Parâmetros de rota: você pode definir parâmetros de rota colocando um nome de parâmetro entre chaves {} no URI da rota. Por exemplo, Route::get('/users/{id}', function ($id) {...}) define uma rota que responde a /users/1, /users/2 e assim por diante, onde 1 e 2 são os valores do parâmetro id.

Rotas nomeadas: você pode atribuir um nome a uma rota usando o método name, como este: Route::get('/users', function () {...})->name('users.index'). Isso permite que você faça referência à rota pelo nome em vez de seu URI, o que pode facilitar a atualização da rota no futuro.

Grupos de rotas: você pode agrupar rotas relacionadas usando o método Route::group, como este: Neste exemplo, o middleware auth é aplicado às rotas /dashboard e /profile, o que garante que apenas usuários autenticados possam acessá-las.

Prefixos de rota: Você pode adicionar um prefixo a um grupo de rotas usando o método prefix, como este: Route::prefix('admin')->group(function () {...}). Isso adicionará o prefixo /admin a todas as rotas definidas no grupo.

Fallbacks de rota: Você pode definir uma rota fallback que é executada quando nenhuma outra rota corresponde à solicitação usando o método Route::fallback, como este: Route::fallback(function () {...}).

No geral, esses métodos de roteador fornecem uma maneira poderosa e flexível de definir as rotas para seu aplicativo Laravel e lidar com solicitações HTTP de entrada.

O que é injeção de dependência em rotas?

O Laravel suporta injeção de dependência em rotas por meio do uso de fechamentos de rota ou controladores. A injeção de dependência é um padrão de design poderoso que permite que objetos sejam injetados em outro objeto, em vez de esse objeto criar suas dependências. Isso permite um melhor desacoplamento de componentes, tornando o código mais sustentável e testável.

No Laravel, você pode usar injeção de dependência em fechamentos de rota por meio de dicas de tipo de um parâmetro na função de fechamento. Por exemplo, se você tem uma classe UserController com um método show que recebe um parâmetro $id e retorna uma visualização, você pode definir uma rota que injeta uma instância da classe UserController assim:

```
use App\Http\Controllers\UserController;

Route::get('/users/{user}', [UserController::class, 'show']);
```

Neste exemplo, estamos definindo uma rota que chama o método show na classe UserController quando a URL /users/{user} é solicitada. Observe que estamos passando um array como o segundo argumento para o método get, que especifica o método do controlador a ser chamado (UserController::class e 'show').

Agora, digamos que o método show precisa usar uma instância da classe UserRepository para buscar os dados do usuário. Podemos injetar a dependência UserRepository no método show usando a injeção automática de dependência do Laravel:

```
use App\Http\Controllers\UserController;
use App\Repositories\UserRepository;

Route::get('/users/{user}', function (UserRepository $userRepository, $user) {
    $user = $userRepository->find($user);
    return view('users.show', compact('user'));
});
```

Neste exemplo, estamos usando um closure como o manipulador de rota em vez de um método controlador. Adicionamos a dependência UserRepository à lista de parâmetros do closure, e o contêiner de serviço do Laravel resolverá automaticamente uma instância da classe UserRepository e a passará para o closure quando a rota for chamada.

Essa técnica de injetar dependências em rotas pode ajudar a manter seu código limpo e modular, e facilitar o teste unitário de seus controladores e closures.

Rotas Nomeadas

Rotas nomeadas são uma maneira de dar um nome a uma rota específica em seu aplicativo Laravel. Em vez de usar a sequência de URL de uma rota no código do seu aplicativo, você pode usar o nome da rota para gerar a URL.

No Laravel, você pode atribuir um nome a uma rota encadeando o método name na fachada Route ao definir a rota. Aqui está um exemplo:

```
Route::get('/users', 'UserController@index')->name('users.index');
```

Neste exemplo, estamos definindo uma rota para a URL /users que mapeia para o método index da classe UserController. Também estamos dando à rota um nome de users.index usando o método name.

Depois de atribuir um nome a uma rota, você pode gerar a URL para essa rota no código do seu aplicativo usando a função route. Aqui está um exemplo:

```
$url = route('users.index');
```

Neste exemplo, estamos usando a função route para gerar a URL para a rota users.index. O Laravel mapeará automaticamente o nome para a sequência de URL da rota, então você não precisa se preocupar em codificar a sequência de URL no código do seu aplicativo.

Rotas nomeadas são úteis de várias maneiras. Por exemplo, elas facilitam a refatoração do seu código quando você precisa alterar a URL de uma rota, já que você só precisa atualizar a definição da rota e as chamadas de rota no seu código. Eles também tornam seu código mais legível, já que o nome da rota fornece uma maneira clara e concisa de se referir à URL no código do seu aplicativo. Além disso, eles podem ajudar a evitar erros causados por erros de digitação ou erros de sintaxe ao codificar URLs no código do seu aplicativo.

Redirecionamento de rotas

No Laravel, podemos redirecionar uma solicitação para uma nova URL usando a função redirect(). Esta função retorna uma instância da classe Illuminate\Routing\Redirector, que fornece vários métodos para criar e gerenciar redirecionamentos. Aqui estão alguns exemplos de como podemos usar rotas de redirecionamento no Laravel:

Redirecionamento básico

```
Route::get('/old-url', function () {
return redirect('/new-url');
});
```

Neste exemplo, estamos definindo uma rota que redireciona solicitações de /old-url para /new-url. Quando um usuário visita /old-url, o Laravel criará uma resposta de redirecionamento com um código de status 302 e um cabeçalho Location que aponta para /new-url.

Redirecionamento com uma rota nomeada

```
Route::get('/old-url', function () {
return redirect()->route('new-url');
});

Route::get('/new-url', function () {
// ...
})->name('new-url');
```

Neste exemplo, estamos usando uma rota nomeada para definir a URL de destino para o redirecionamento. A função redirect() é chamada sem argumentos, o que cria uma instância do redirecionador sem URL de destino. Em seguida, chamamos o método route() na instância do redirecionador para especificar a URL de destino pelo nome da rota.

Redirecionar com dados flash

```
Route::post('/form-submit', function () {
// Processar dados do formulário...

return redirect('/thank-you')->with('message', 'Obrigado por enviar o formulário!');
});

Route::get('/thank-you', function () {
$message = session('message');

return view('thank-you', compact('message'));
});
```

Neste exemplo, estamos usando o método with() para anexar dados flash à resposta de redirecionamento. Dados flash são dados que estão disponíveis apenas para a próxima solicitação e, em seguida, são removidos automaticamente da sessão. Neste caso, estamos anexando uma

mensagem "Obrigado por enviar o formulário!" à resposta de redirecionamento e, em seguida, exibindo-a na página /thank-you.

Redirecionar com um código de status

```
Route::get('/maintenance', function () {
return redirect('/home')->status(503);
});
```

Neste exemplo, estamos criando uma resposta de redirecionamento com um código de status 503. Isso é útil quando você deseja tirar uma página temporariamente do ar para manutenção ou atualizações.

No geral, as rotas de redirecionamento no Laravel fornecem uma maneira flexível de lidar com redirecionamentos HTTP em seu aplicativo. Você pode usá-las para redirecionar usuários para novas URLs, nomes de rotas ou até mesmo para outros domínios. Você também pode anexar dados flash à resposta de redirecionamento, personalizar o código de status HTTP e muito mais.

Redirecionamento permanente

Um redirecionamento permanente é uma resposta de redirecionamento HTTP com um código de status 301, indicando que um recurso foi movido permanentemente para uma nova URL. Quando um cliente (por exemplo, um navegador da web ou um bot de mecanismo de busca) recebe uma resposta 301, ele sabe que o recurso solicitado foi movido permanentemente para um novo local e deve atualizar seus registros de acordo.

No Laravel, você pode criar um redirecionamento permanente chamando o método permanentRedirect() na instância do redirecionador. Aqui está um exemplo:

```
Route::get('/old-url', function () {
return redirect()->permanent('/new-url');
});
```

Neste exemplo, estamos definindo uma rota que cria um redirecionamento permanente de /old-url para /new-url. Quando um usuário visita /old-url, o Laravel criará uma resposta de redirecionamento com um código de status 301 e um cabeçalho Location que aponta para /new-url. Isso informa ao cliente que o recurso foi movido permanentemente e que ele deve atualizar seus registros de acordo.

Redirecionamentos permanentes são úteis quando você deseja redirecionar o tráfego de uma URL antiga para uma nova URL, preservando a classificação do mecanismo de busca e o valor do tráfego da URL antiga. Isso é especialmente importante para sites que já existem há algum tempo e acumularam uma quantidade significativa de links de entrada e tráfego. Ao usar um redirecionamento permanente, você pode garantir que o tráfego e a classificação do mecanismo de busca da URL antiga sejam transferidos para a nova URL, em vez de serem perdidos na transição.

Parâmetros de rota

Os parâmetros de rota do Laravel permitem capturar partes da URL como variáveis, que podem ser passadas como argumentos para seus métodos de controlador ou fechamentos. Os parâmetros de rota são especificados colocando um nome de parâmetro entre chaves {} na definição de rota.

Por exemplo, digamos que queremos capturar o ID de um usuário da URL em nosso aplicativo Laravel. Podemos definir uma rota com um parâmetro como este:

```
Route::get('/users/{id}', 'UserController@show');
```

Neste exemplo, estamos definindo uma rota para a URL /users/{id} que mapeia para o método show da classe UserController. O parâmetro {id} na URL é um parâmetro de rota, que capturará o valor de ID da URL e o passará para o método show como um argumento.

Você pode definir quantos parâmetros de rota forem necessários em uma única definição de rota, e eles serão passados para seu método de controlador ou fechamento na ordem em que forem definidos na URL.

Aqui está um exemplo que captura tanto o ID de um usuário quanto o ID de uma postagem da URL:

```
Route::get('/users/{userId}/posts/{postId}', function ($userId, $postId) {
// Seu código aqui
});
```

Neste exemplo, estamos definindo uma rota para a URL /users/{userId}/posts/{postId} que captura tanto o ID de um usuário quanto o ID de uma postagem da URL e os passa para um fechamento como argumentos separados.

Os parâmetros de rota são úteis quando você precisa passar valores dinâmicos para seus métodos de controlador ou fechamentos, como IDs de usuário, IDs de postagem ou outros identificadores de recursos. Eles podem ajudar a tornar o código do seu aplicativo mais flexível e sustentável, permitindo que você capture diferentes tipos de dados da URL e os use em seu código.

O Laravel permite que você defina parâmetros opcionais em suas rotas usando o símbolo ?. Isso torna possível definir rotas que podem corresponder a diferentes URLs com base na presença ou ausência de certos parâmetros.

Aqui está um exemplo de uma rota que tem um parâmetro opcional:

```
Route::get('/users/{id}/{name?}', function ($id, $name = null) {
// Seu código aqui
});
```

Neste exemplo, definimos uma rota que pode corresponder a URLs com um ou dois parâmetros. O primeiro parâmetro, id, é obrigatório e sempre estará presente na URL. O segundo parâmetro, name, é opcional e pode ser incluído ou excluído da URL. Se o parâmetro name não estiver incluído na URL, seu valor será definido como null por padrão.

Você pode ter quantos parâmetros opcionais forem necessários em uma única definição de rota, e eles serão passados para seu método controlador ou fechamento como null se não estiverem presentes na URL.

Parâmetros opcionais são úteis quando você precisa definir rotas que podem corresponder a URLs com diferentes conjuntos de parâmetros. Por exemplo, você pode ter uma rota que pode corresponder a URLs para uma página de perfil de usuário com ou sem um parâmetro username. Ao tornar o parâmetro username opcional, você pode evitar ter que definir duas rotas separadas para esses casos.
https://ashutosh.dev/routing-in-laravel-10/

Rotas tipo Resource

php artisan make:controller PhotoController --resource

use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class);

Actions Handled by Resource Controllers

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

php artisan route:list

```
  GET|HEAD       products
  POST          products
  GET|HEAD       products/create
  GET|HEAD       products/{product}
  PUT|PATCH      products/{product}
  DELETE        products/{product}
  GET|HEAD       products/{product}/edit
```

Constraints/Restrições de Expressão Regular

Você pode restringir o formato dos seus parâmetros de rota usando o método where em uma instância de rota. O método where aceita o nome do parâmetro e uma expressão regular definindo como o parâmetro deve ser restringido:

```
Route::get('/user/{name}', function (string $name) {
// ...
})->where('name', '[A-Za-z]+');

Route::get('/user/{id}', function (string $id) {
```

```
// ...
})->where('id', '[0-9]+');

Route::get('/user/{id}/{name}', function (string $id, string $name) {
// ...
})->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Para conveniência, alguns padrões de expressão regular comumente usados têm métodos auxiliares que permitem que você adicione rapidamente restrições de padrão às suas rotas:

```
Route::get('/user/{id}/{name}', function (string $id, string $name) {
// ...
})->whereNumber('id')->whereAlpha('name');

Route::get('/user/{name}', function (string $name) {
// ...
})->whereAlphaNumeric('name');

Route::get('/user/{id}', function (string $id) {
// ...
})->whereUuid('id');

Route::get('/user/{id}', function (string $id) {
// ...
})->whereUlid('id');

Route::get('/category/{category}', function (string $category) {
// ...
})->whereIn('category', ['movie', 'song', 'painting']);

Route::get('/category/{category}', function (string $category) {
// ...
})->whereIn('category', CategoryEnum::cases());
```

Se a solicitação de entrada não corresponder às restrições do padrão de rota, uma resposta HTTP 404 será retornada.

Grupos de Rotas

Grupos de rotas permitem que você compartilhe atributos de rota, como middleware, em um grande número de rotas sem precisar definir esses atributos em cada rota individual.

Grupos aninhados tentam "mesclar" atributos de forma inteligente com seu grupo pai. Middleware e where condições são mescladas enquanto nomes e prefixos são anexados. Delimitadores de namespace e barras em prefixos de URI são adicionados automaticamente quando apropriado.

Middleware

Para atribuir middleware a todas as rotas dentro de um grupo, você pode usar o método middleware antes de definir o grupo. Middleware são executados na ordem em que são listados na matriz:

```
Route::middleware(['first', 'second'])->group(function () {
Route::get('/', function () {
// Usa o primeiro e o segundo middleware...
});

Route::get('/user/profile', function () {
// Usa o primeiro e o segundo middleware...
});
});
```

Controladores

Se um grupo de rotas utilizar o mesmo controlador, 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 elas invocam:

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

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

Roteamento de subdomínio

Grupos de rotas também podem ser usados para manipular o roteamento de subdomínio. Subdomínios podem receber parâmetros de rota assim como URIs de rota, permitindo que você capture uma parte do subdomínio para uso em sua rota ou controlador. O subdomínio pode ser especificado chamando o método de domínio antes de definir o grupo:

```
Route::domain('{account}.example.com')->group(function () {
        Route::get('/user/{id}', function (string $account, string $id) {
        // ...
        });
});
```

Para garantir que suas rotas de subdomínio sejam alcançáveis, você deve registrar rotas de subdomínio antes de registrar rotas de domínio raiz. Isso evitará que rotas de domínio raiz substituam rotas de subdomínio que tenham o mesmo caminho de URI.

Prefixos de rota

O método de prefixo pode ser usado para prefixar cada rota no grupo com um determinado URI. Por exemplo, você pode querer prefixar todos os URIs de rota dentro do grupo com admin:

```
Route::prefix('admin')->group(function () {
        Route::get('/users', function () {
        // Corresponde à URL "/admin/users"
        });
});
```

Prefixos de Nome de Rota

O método name pode ser usado para prefixar cada nome de rota no grupo com uma determinada string. Por exemplo, você pode querer prefixar os nomes de todas as rotas no grupo com admin. A string fornecida é prefixada ao nome da rota exatamente como é especificada, então teremos certeza de fornecer o caractere . final no prefixo:

```
Route::name('admin.')->group(function () {
        Route::get('/users', function () {
        // Nome atribuído à rota "admin.users"...
        })->name('users');
});
```

Limpando o cache das rotas

php artisan route:clear

4 – Referências

Melhor site com conteúdo sobre laravel ao meu ver

https://www.itsolutionstuff.com/laravel-tutorial
https://github.com/savanihd?tab=repositories

Muito bom
https://www.allphptricks.com/category/laravel/

Playlists do João Ribeiro sobre laravel
https://www.youtube.com/watch?v=1nPkIjIXe3E&list=PLXik_5Br-zO9xlSwhhEDUGF81M5mgMUFQ&pp=iAQB
https://www.youtube.com/watch?v=0T5gM1WRNsY&list=PLXik_5Br-zO893qVjjP7a4qg4NYrl33w1&pp=iAQB

Using PHP Codesniffer With Laravel
https://laravel-news.com/php-codesniffer-with-laravel

https://laravel-news.com/

https://laravel-news.com/your-first-laravel-application

Ótimo
https://www.tutsmake.com/category/laravel-tutorial/

Gerador de seeder
https://laravelseedergenerator.james-nock.co.uk/

https://www.w3schools.in/laravel

How to Integrate ChatGPT API with Laravel 11?
https://www.itsolutionstuff.com/post/how-to-integrate-chatgpt-api-with-laravel-11example.html

How to Integrate Admin Template in Laravel 11?
https://www.itsolutionstuff.com/post/how-to-integrate-admin-template-in-laravel-11example.html

Laravel 11 User Roles and Permissions Tutorial
https://www.itsolutionstuff.com/post/laravel-11-user-roles-and-permissions-tutorialexample.html

https://medium.com/@Adekola_Olawale/beginners-guide-to-object-oriented-programming-a94601ea2fbd

https://www.itsolutionstuff.com/post/laravel-11-crud-with-image-upload-tutorialexample.html

https://www.itsolutionstuff.com/post/laravel-11-multiple-image-upload-tutorial-exampleexample.html