Laravel 5.5 - Desde Cero

Duilio Palacios (Styde)

https://www.youtube.com/watch?v=7YvBOOSqM8k&list=PL1r3w0C4ClYRbiTB4o70CyJEW6hUWJ39X

Resumo do curso feito por Roberto Pinheiro

Aula 01 - Instalação de Laravel

Para desenvolver aplicativos PHP com o Laravel, precisamos primeiro instalar e configurar um conjunto de ferramentas que facilite a criação de novos aplicativos. Por um lado, precisamos ter um ambiente de desenvolvimento em nossa equipe que atenda aos requisitos da estrutura e, por outro lado, é aconselhável configurar e conhecer as maneiras de acessar um aplicativo criado nesse ambiente. Nesta primeira aula, vamos orientá-lo a preparar sua equipe e, assim, começar a desenvolver com o Laravel.

Preparação do ambiente de desenvolvimento

Para desenvolver com o Laravel 5.5, você pode fazer isso do Windows, Linux ou MacOs sempre que tiver um servidor web com PHP 7 ou superior.

Chamamos isso de ambiente de desenvolvimento e há uma grande variedade deles, cada um com um nível de complexidade diferente do outro, do mais básico, instalando manualmente o Apache ou Nginx, PHP, MySQL, etc., bem como instalando ferramentas como o XAMPP, WAMP, MAMP, etc, para os mais complicados como Laravel Homestead

No entanto, recomendamos as seguintes opções para quem está iniciando, por sua facilidade de instalação e uso:

- No Windows pode-se usar: Laragon, ambiente de desenvolvimento para o Laravel no Windows
- No Linux: Instalando o Laravel Valet no Linux
- Em MacOs: Laravel Valet

Por outro lado, o Laravel usa o Composer para gerenciar suas dependências. O que significa isto? Para o Laravel, o framework utiliza uma coleção de pacotes ou componentes próprios ou de terceiros para adicionar funcionalidade aos aplicativos.

Portanto, precisamos de um gerenciador de dependências que seja automaticamente responsável pela criação de projetos, instalação, atualização ou exclusão de componentes e, por sua vez, pelas dependências deles. Esta ferramenta é o Composer, o gerenciador de dependências do PHP.

Instalá-lo depende do sistema operacional que você usa se for MacOs ou Linux você pode seguir os passos descritos no vídeo: baixe o arquivo composer.phar com as instruções do site oficial e então mova este arquivo para que seja global e assim use a ferramenta de qualquer diretório, seguindo as instruções oficiais do Composer.

Para o Windows, você pode baixar o instalador oferecido pelo Composer no seu site que irá instalá-lo para que você possa usá-lo em qualquer lugar do sistema. Se você decidiu trabalhar com o Laragon, não precisa executar esta instalação, já que ela é incluída por padrão neste ambiente de desenvolvimento.

Você pode confirmar se instalou o Composer executando no console a partir de qualquer diretório: composer e, se estiver instalado, mostrará uma lista de todos os comandos disponíveis.

Instalação do Laravel

Quando o ambiente de desenvolvimento estiver pronto, usaremos o Composer para instalar o Laravel desta maneira:

```
composer create-project --prefer-dist laravel/laravel mi-proyecto
```

Isso significa que estamos criando um novo projeto chamado create-project de Composer à partir do pacote laravel/laravel, usando a opção --prefer-dist para baixar o repositório de arquivos de distribuição Composer.

Há uma alternativa para instalar laravel e seu instalador, que também é um pacote, portanto, nós também usará Composer para instalá-lo globalmente com o comando:

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

Então nós temos que assegurar que o ambiente PATH sistema operacional variável ter incluído o diretório onde os pacotes instalados globalmente ficar e assim pode ser executado sem problemas, para isso temos de acrescentar sua rota:

Para MacOS e Linux a variável PATH pode ser definida em ~ / .bashrc ou o ~/.bash_profile, onde a rota a ser adicionado ao final do que tem atribuído a variável é: $ HOME/.composer/vendor/bin

Para o Windows tem de modificar a variável de ambiente PATH para adicionar o caminho,

C:\Users\seu usuário\AppData\Roaming\Composer\vendor\bin

Bem, desta forma já temos o instalador do Laravel disponível, portanto, podemos rodar a partir de qualquer diretório:

```
laravel new nome-projeto
```

e ele será instalado como foi feito com o comando:

```
composer create-project.
```

Abrir a aplicação criada

Para ver e navegar no aplicativo recém-criado, podemos executar o comando:

```
php artisan serve
```

Esse comando executa o aplicativo em um servidor de desenvolvimento incluído por padrão no Laravel. Portanto, devemos clicar no URL que nos mostra para explorar o aplicativo no navegador. Para parar a execução, pressione Ctrl + C

Aula 02 - Introdução ao Laravel 5.5

O Laravel é um framework para o desenvolvimento de aplicações web com PHP que nos permite construir aplicativos modernos com uma sintaxe elegante e expressiva. Foi criado por Taylor Otwell em 2011 e a partir da data de publicação deste curso passa pela versão 5.5. Para começar a trabalhar com o Laravel, você precisa preparar seu ambiente de desenvolvimento, IDE ou editor de texto e outras ferramentas, como o Composer, que o ajudarão a instalar e desenvolver aplicativos da Web com essa estrutura. Nesta lição, faremos uma breve introdução à estrutura e aprenderemos sobre o padrão de design do Front Controller, que é usado pelo Laravel para processar todas as solicitações feitas ao aplicativo.

Para instalar um novo projeto do Laravel, você pode fazê-lo de duas maneiras: com o comando create-project no Composer ou com o laravel new como explicado na lição anterior. Mas há algo mais que você deveria saber. O instalador do Laravel só nos permite instalar a versão atual do framework (opção default) e a versão em desenvolvimento com o comando laravel new mi-proyecto --dev

Portanto, se você quiser instalar o Laravel em uma versão diferente da versão mais recente, use o comando create-project que nos dá a opção de especificar a versão que queremos usar. Desta maneira:

```
composer create-project laravel/laravel nombre-proyecto "5.5.*"
```

Com isso estaríamos instalando um projeto do Laravel 5.5

OBS.: Se você vai começar a seguir as lições deste curso, recomendamos que você instale a versão 5.5 para poder reproduzir os exemplos como eles estão no vídeo. Não se preocupe, se houver uma versão posterior do Laravel, forneceremos as lições e notas para que você possa aprender as pequenas diferenças entre um e outro.

O uso de um sistema de controle de versão como o Git é primordial quando você quer se desenvolver profissionalmente. Para realizar este curso, o Git é opcional, pois será usado apenas para compartilhar o código.

Por outro lado, algo importante para desenvolver de forma eficiente é ter um editor de texto ou IDE bem configurado para facilitar o trabalho de escrever o código de nossa aplicação. Existem dois grupos principais: IDE (Integrated Development Environment) e editores de texto.

A principal diferença é que os primeiros vêm, por padrão, com várias ferramentas, como: autocomplete inteligente, destaque de sintaxe, sistema de controle de versão, depurador, entre outras ferramentas configuradas e prontas para começar a funcionar. Em contraste, os editores de texto são mais leves e não vêm com todas as ferramentas ou opções listadas por padrão, mas você deve instalá-los e configurá-los por meio de plugins e extensões.

Entre os IDEs para PHP temos: PHPStorm, Zend Studio, Eclipse, NetBeans, Aptana Studio, etc.

Entre os editores de texto estão: Sublime Text, Atom, Visual Studio Code, NotePad ++, etc. Escolha um com o qual você se sinta confortável e comece a se desenvolver!

O padrão de design do Front Controller é que um único componente do aplicativo é responsável por manipular todas as solicitações HTTP que ele recebe. Ou seja, há apenas um ponto de acesso para todas as solicitações.

No Laravel esta função é preenchida pelo arquivo index.php encontrado no diretório public. junto com o arquivo .htaccess. Bem, quando você usa o servidor web Apache, este último arquivo é responsável por redirecionar todos os pedidos para index.php

O diretório public também contém as imagens, arquivos CSS e arquivos Javascript que serão públicos para usuários e visitantes da aplicação, o resto dos arquivos onde se encontra a lógica da aplicação está inacessível para eles, ou seja, são privados.

Aula 03 - Criação de rotas no Laravel 5.5

O sistema de Rotas do Laravel é bastante intuitivo e fácil de manusear, mas ao mesmo tempo muito poderoso, com ele podemos criar todos os tipos de rotas na aplicação: simples ou complexas. Nesta lição, você aprenderá a criar suas primeiras rotas em um projeto do Laravel.

As rotas são uma camada muito importante no Laravel, é por isso que o Framework aloca um diretório na pasta raiz, chamado de rotas, para localizar todas as rotas do aplicativo. Por padrão, ele possui 2 arquivos de rotas web.php e api.php. Como seus nomes expressam em web.php as rotas para a web são definidas e em api.php as rotas para criar APIs para a aplicação.

Podemos definir rotas de várias maneiras nesta lição, usando uma função anônima, que segue o seguinte formato:

```
Route::get('/esta-es-la-url', function () {
   return 'Hola mundo';
});
```

Escreva a classe Route que chama o método relacionado ao verbo HTTP, neste caso, get que aceita dois parâmetros: o primeiro é o URL que será chamado do navegador e o segundo é uma função anônima que retorna o que queremos mostrar.

Mais tarde, você conhecerá outras maneiras de definir rotas, como usar ações do controlador.

Para ver a rota em operação, devemos escrever no navegador algo como: http://tu-proyecto.dev/esta-es-la-url, dependendo do VirtualHost que você criou para o seu projeto. Se você não tem, você pode usar o php artisan ou revisar a primeira lição deste curso, onde explicamos como preparar seu ambiente de desenvolvimento.

Rotas com parâmetros

Também com o sistema de rotas do Laravel, você pode criar rotas mais complexas que precisam de parâmetros dinâmicos. Eles podem ser definidos da seguinte maneira:

```
Route::get('/usuarios/detalles/{id}', function ($id) {
   return "Detalles del usuario: {$id}";
});
```

Neste caso, o Laravel é responsável por capturar o segmento da rota que é dinâmico (identifica-o porque está entre chaves). Portanto, na URL nós passamos a identificação do parâmetro entre chaves e na função anônima passamos isso como um argumento para que possa ser acessado e usado dentro da função.

Você pode usar quantos parâmetros forem necessários, é importante que eles sejam colocados entre chaves {} e os nomes possam ser alfanuméricos, mas você não pode usar o hífen (-) e o sublinhado(_). Além disso, importa a ordem dos parâmetros passados para a função anônima, mas não os nomes dados a eles. Por exemplo:

```
Route::get('posts/{post_id}/comments/{comment_id}', function ($postId, $commentId) {
   return "Este el comentario {$commentId} del post {$postId}";
});
```

Rotas com filtros ou restrições de expressões regulares nos parâmetros

Quando um usuário faz uma solicitação HTTP, o Laravel procura nos arquivos de roteamento uma definição que corresponda ao padrão de URL de acordo com o método HTTP usado e, na primeira correspondência, mostre o resultado para o usuário. Portanto, a ordem de precedência das definições de rota é muito importante.

Para resolver possíveis conflitos com a similaridade na URL de diferentes rotas, você pode fazer isso de duas maneiras:

Usando o método where para adicionar condições de expressão regulares à rota. Você pode conferir nossas rotas de tutorial com filtros no Laravel onde explicamos em detalhes o uso do método where.

Classificando as rotas de forma que as mais específicas estejam no início e as mais gerais no final do arquivo de rota.

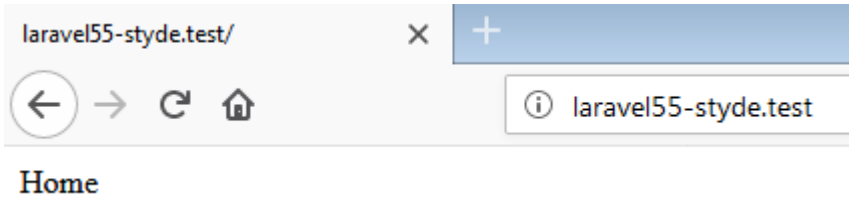
Rotas com parâmetros opcionais

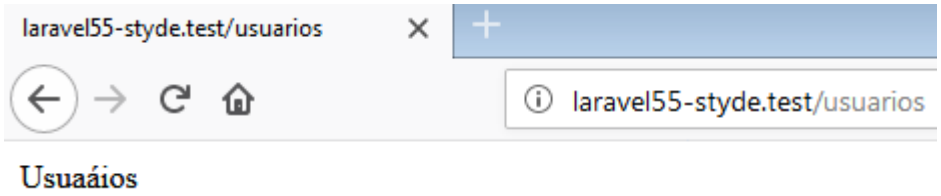
Quando o uso de um parâmetro não é obrigatório, podemos usar o caractere? após o nome do parâmetro para indicar que é opcional. No entanto, um valor padrão deve ser adicionado ao parâmetro quando colocá-lo na função, por exemplo:

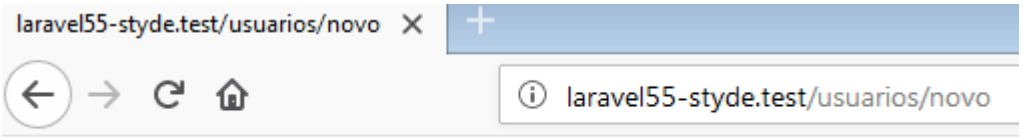
```
Route::get('saudar/{name}/{nickname?}', function ($name, $nickname = null) {
   if ($nickname) {
      return "Bem-vindo {$name}, teu apelido é {$nickname}";
   } else {
      return "Bem-vindo {$name} ";
   }
});
```

routes/web.php

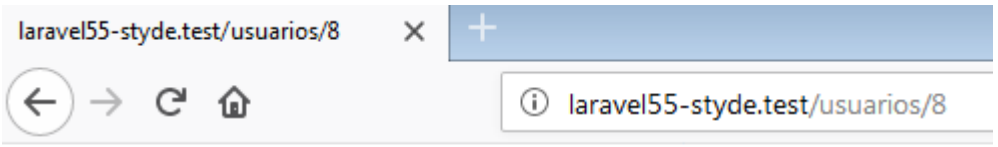
```php
<?php

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

Route::get('/usuarios', function () {
    return 'Usuários';
});

Route::get('/usuarios/novo', function () {
    return 'Criar novo usuário';
});

Route::get('/usuarios/{id}', function ($id) {
    return "Exibindo detalhes do usuário: {$id}";
});

Route::get('/saudar/{name}/{nickname?}', function ($name, $nickname = null) {

    $name = ucfirst($name);
    $nickname = ucfirst($nickname);

    if ($nickname) {
        return "Bem-vindo {$name}, teu apelido é {$nickname}";
    } else {
        return "Bem-vindo {$name}";
    }
});
```



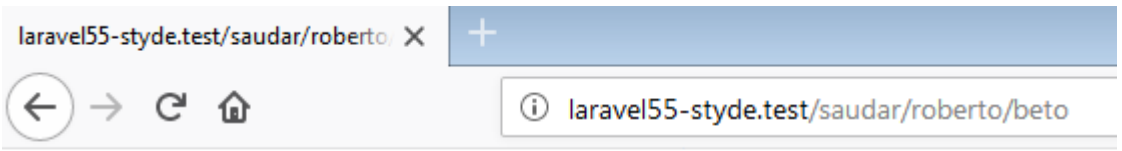




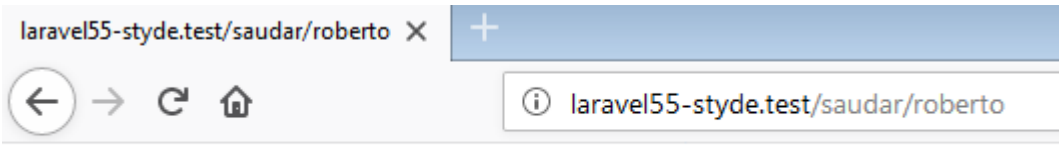
Criar novo usuário



Exibindo detalhes do usuário: 8



Bem-vindo Roberto, teu apelido é Beto



Bem-vindo Roberto

Aula 04 - Introdução à provas automatizadas no Laravel 5.5

Na lição anterior, onde aprendemos a escrever as primeiras rotas do nosso aplicativo, usamos o navegador para testar essas rotas e URLs. O problema desses testes no navegador é que eles não duram no tempo e não podem ser executados rápida/automaticamente. Então hoje vamos ver como podemos testar o código que desenvolvemos de forma mais inteligente, usando o componente de teste automatizado que vem com o Laravel.

Diretório de teste

O Laravel inclui um diretório chamado /tests no diretório principal do seu projeto. Neste diretório, escreveremos código que será responsável por testar o código do restante do aplicativo. Este diretório é separado em dois subdiretórios: O diretório Feature onde escrevemos testes que emulam solicitações HTTP para o servidor. O diretório Unit onde escrevemos testes que são responsáveis por testar partes individuais do aplicativo (como classes e métodos).

Escrevendo um teste

No teste a seguir, simularemos uma solicitação HTTP GET para o URL do módulo do usuário. Com o asserStatus, verificamos se o URL é carregado corretamente, verificando se o status HTTP é 200. Com o método assertSee, verificamos se podemos ver o texto "Usuários":

```
/** @test */
function it_loads_the_users_list_page()
{
  $this->get('/usuarios')
    ->assertStatus(200)
    ->assertSee('Usuarios');
}
```

Para que o PHPUnit execute o método como um teste, você deve colocar a anotação / ** @test * / antes da declaração do método ou colocar o prefixo test_ no nome do método como tal:

```
function test_it_loads_the_users_list_page
{
  //...
}
```

Caso contrário, o método NÃO será executado como um teste, o que é útil porque nos permite adicionar métodos auxiliares em nossa classe de teste.

Notas

O comando para gerar novos testes é:

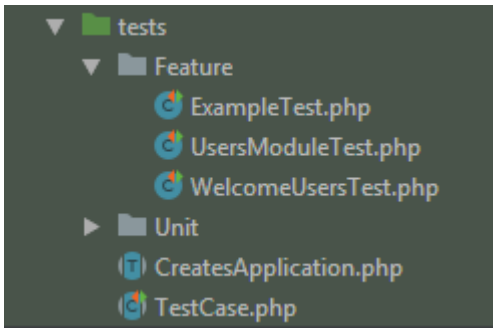
php artisan make:test NomeTesteDeProva

Você pode executar os testes com vendor/bin/phpunit ou criar um alias para o console. Você pode ler sobre os métodos de asserção disponíveis na documentação do Laravel.

Iniciando os testes de prova

No terminal, entre com o seguinte comando:

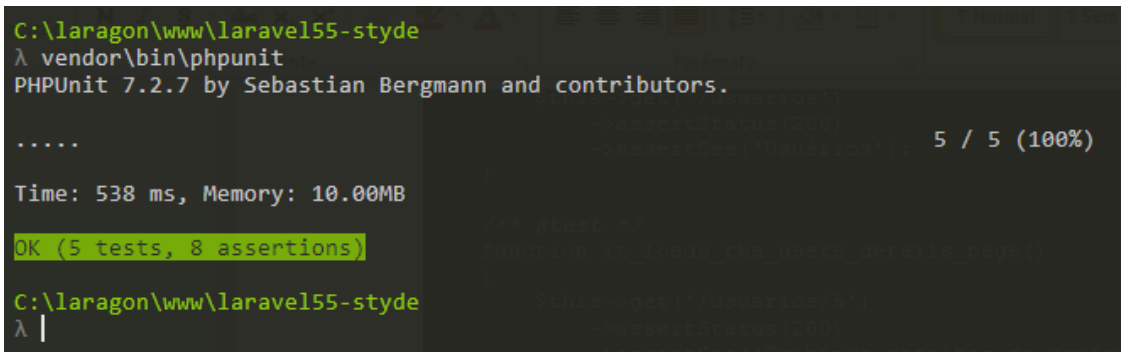
php artisan make:test UsersModuleTest



tests/Feature/UsersModuleTest.php

```
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{
   /** @test */
   function it_loads_the_users_list_page()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Usuários');
   }

   /** @test */
   function it_loads_the_users_details_page()
   {
      $this->get('/usuarios/5')
         ->assertStatus(200)
         ->assertSee('Exibindo detalhes do usuário: 5');
   }

   /** @test */
   function it_loads_the_new_users_page()
   {
      $this->get('/usuarios/novo')
         ->assertStatus(200)
         ->assertSee('Criar novo usuário');
   }
}
```

Entre com o comando: vendor\bin\phpunit

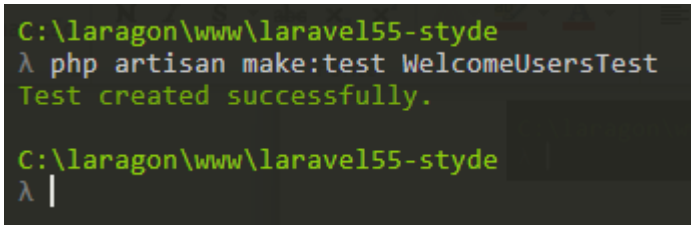


Criando um alias:

alias t=vendor\bin\phpunit

No terminal, entre com o seguinte comando:

php artisan make:test WelcomeUsersTest

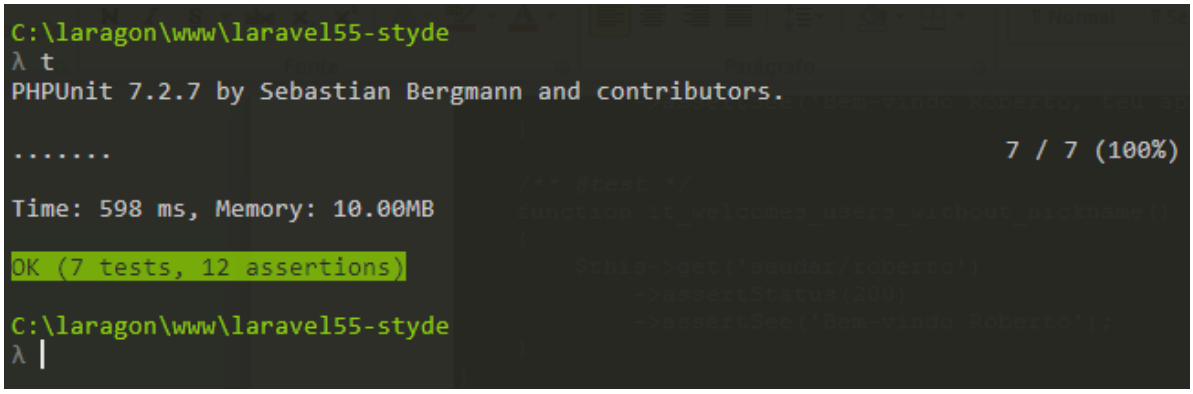


tests/Feature/WelcomeUsersTest.php

```
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class WelcomeUsersTest extends TestCase
{
   /** @test */
   function it_welcomes_users_with_nickname()
   {
      $this->get('saudar/roberto/beto')
         ->assertStatus(200)
         ->assertSee('Bem-vindo Roberto, teu apelido é Beto');
   }

   /** @test */
   function it_welcomes_users_without_nickname()
   {
      $this->get('saudar/roberto')
         ->assertStatus(200)
         ->assertSee('Bem-vindo Roberto');
   }
}
```

Entre com o comando:

vendor\bin\phpunit



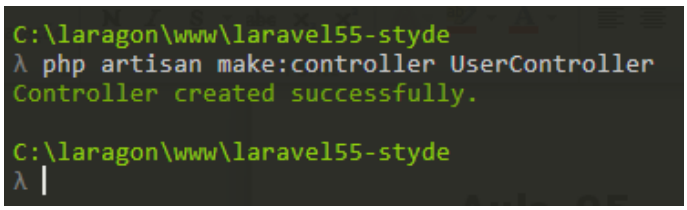
Aula 05 - Criação e uso de Controllers no Laravel 5.5

Os controladores são um mecanismo que nos permite agrupar a lógica de solicitações HTTP relacionadas e, assim, organizar melhor nosso código. Nesta quinta lição do Curso Laravel vamos aprender a fazer uso deles e também veremos como os testes unitários nos permitem verificar as mudanças que introduzimos em nosso código de maneira fácil e rápida.

Criando um controlador

Geramos um novo controlador com o comando do Artisan make: controller, passando o nome que queremos dar. No exemplo, o nome é UserController:

php artisan make:controller UserController



Feito isso, no diretório app/Http/Controllers teremos nosso controlador UserController.

Métodos no controlador

Um controller nada mais é do que um arquivo .php com uma classe que se estende da classe App\Http\Controllers\Controller:

```
<?php

namespace App\Http\Controllers;

class UserController extends Controller {
    // ...
}
```

Estender a classe Controller é opcional, no entanto, é recomendado porque fornecerá vários métodos úteis que veremos mais adiante. Dentro desta classe (no nosso caso UserController) nós adicionamos nossos métodos públicos (chamados ações), que podemos então vincular a uma rota:

```
public function index()
{
   return 'Usuários';
}
```

Vincular uma rota a um controlador

Para vincular uma rota a um controlador, passamos como argumento o nome do controlador e o método que queremos vincular, separados por um @. Neste caso, queremos vincular o caminho /usuários ao método de indexação do driver UserController:

```
Route::get('/usuarios', 'UserController@index');
```

Controlador de método único

Se você quiser ter um controlador que tenha apenas uma ação, você pode fazê-lo chamando o método __invoke, por exemplo:

```
public function __invoke($name, $nickname = null)
{
   // ...
}
```

Em nossa rota, agora podemos vincular diretamente ao controlador:

```
Route::get('/saudar/{name}/{nickname}', 'WelcomeUserController');
```

Detectar erros nos testes

Na lição, ensinei a você uma maneira rápida de "revelar" erros de programação nos testes. Depois você pode ver a lição Tratamento de exceções nos testes do Laravel 5.5 para aprender mais sobre o assunto. Lembre-se de que você também pode abrir o navegador para ver os erros, conforme explicado em Instalando o Laravel 5.5 e o retorno do componente Whoops. A ideia é combinar testes automatizados e manuais para verificar o funcionamento do código que você está desenvolvendo em tempo hábil.

routes/web.php

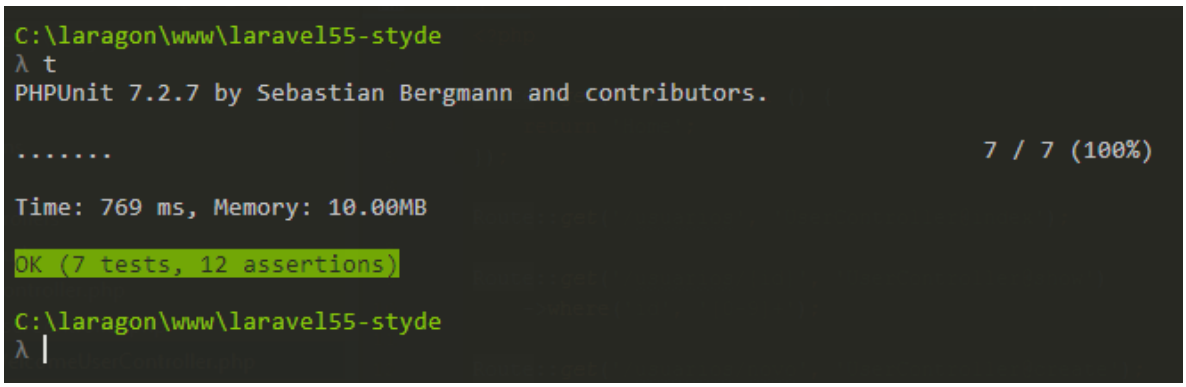
```php
<?php

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

Route::get('/usuarios', 'UserController@index');

Route::get('/usuarios/{id}', 'UserController@show')
   ->where('id', '[0-9]+');

Route::get('/usuarios/novo', 'UserController@create');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

app/Http/Controllers/UserController

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
   public function index()
   {
      return 'Usuários';
   }

   public function show($id)
   {
      return "Exibindo detalhes do usuário: {$id}";
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```

app/Http/Controllers/WelcomeUserController

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class WelcomeUserController extends Controller
{
   public function __invoke($name, $nickname = null)
   {
      $name = ucfirst($name);
      $nickname = ucfirst($nickname);

      if ($nickname) {
         return "Bem-vindo {$name}, teu apelido é {$nickname}";
      } else {
         return "Bem-vindo {$name}";
      }
   }
}
```



Aula 06 - Criação e uso de Views no Laravel 5.5

O Laravel também nos permite separar a lógica de apresentação (ou seja, como vamos "apresentar" o conteúdo ao usuário) da lógica de nossa aplicação (por exemplo, como vamos obter o conteúdo do banco de dados, validar os dados da requisição, etc.) através da camada de "views".

Criar uma view

As visualizações geralmente são encontradas no diretório /resources/views da pasta principal do nosso projeto. Criar uma visão com o Laravel é muito simples, nós simplesmente precisamos criar um arquivo .php (ou .blade.php como veremos na próxima lição) no diretório /views. Dentro deste arquivo nós escrevemos o HTML da visão.

Retornar uma view

Para retornar uma view, retornamos a chamada para a função de visualização auxiliar passando o nome da view como um argumento. O nome do arquivo é relativo à pasta resources /views e não é necessário indicar a extensão do arquivo:

```
public function index()
{
   return view('users');
}
```

Passar dados para a view

Podemos passar dados para a view por meio de um arranjo associativo, onde as chaves são o nome das variáveis que queremos passar para a view e o valor são os dados que queremos associar:

```
$users = [
   'Joel',
   'Ellie',
   'Tess',
   //...
];
```

```
return view('users', ['users' => $users]);
```

Também podemos usar o método with vinculando-o à chamada para a função view para passar dados para a view no formato de array associativo:

```
return view('users')->with(['users' => $users]);
```

Com with podemos também passar as variáveis individualmente:

```
return view('users')
   ->with('users', $users)
   ->with('title', 'Listagem de usuarios');
```

Se os dados que queremos ver estiverem dentro das variáveis locais, podemos usar a função compact, que aceita os nomes das variáveis como argumentos e os converte em um array associativo:

```
$users = [
   ...
];

$title = 'Listagem de usuários';

return view('users', compact('users', 'title'));
```

Escape do código HTML

O Laravel nos oferece um helper chamado e que nos permite escapar do HTML que poderia ser inserido pelos usuários do nosso aplicativo, a fim de evitar possíveis ataques XSS:

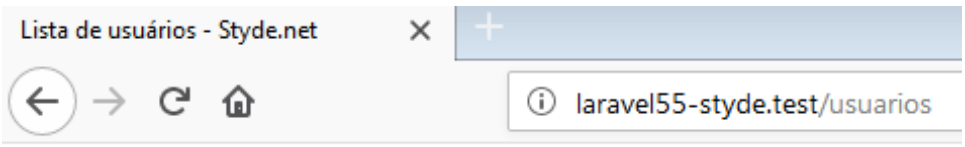
```
<li> <? php echo e($usuário) ?> </ li>
```

app/Http/Controllers/UserController

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
   public function index()
   {
      $users = [
         'Joel',
         'Ellie',
         'Tess',
         'Tommy',
         'Bill',
         '<script>alert("Clicker")</script>'
      ];

      $title = 'Lista de usuários';

      return view('users', compact('title', 'users'));
   }

   public function show($id)
   {
      return "Exibindo detalhes do usuário: {$id}";
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```

resources/views/users.php

```
<!doctype html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Lista de usuários - Styde.net</title>
</head>
<body>
<h1><?= e($title) ?></h1>

<ul>
   <?php foreach ($users as $user): ?>
      <li><?= e($user) ?></li>
   <?php endforeach; ?>
</ul>
</body>
</html>
```



Lista de usuários

- Joel
- Ellie
- Tess
- Tommy
- Bill
- <script>alert("Clicker")</script>

tests/Feature/UsersModuleTest.php

```php
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{
    /** @test */
    function it_loads_the_users_list_page()
    {
        $this->get('/usuarios')
            ->assertStatus(200)
            ->assertSee('Lista de usuários')
            ->assertSee('Joel')
            ->assertSee('Ellie');
    }

    /** @test */
    function it_loads_the_users_details_page()
    {
        $this->get('/usuarios/5')
            ->assertStatus(200)
            ->assertSee('Exibindo detalhes do usuário: 5');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar novo usuário');
    }
}
```

```
C:\laragon\www\laravel55-styde
λ t
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.......                                                             7 / 7 (100%)

Time: 603 ms, Memory: 10.00MB

OK (7 tests, 14 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 07 - Templates com Blade no Laravel 5.5

Nesta lição, começaremos a aprender sobre o Blade, o sistema de templates Laravel, que nos fornece muitos recursos que devemos ter em uma linguagem de template, como a capacidade de escapar dados automaticamente.

Anotações

Para usar o sistema de templates Laravel, devemos renomear nossas visualizações para que elas tenham a extensão .blade.php.

Variáveis de impressão

Se quisermos imprimir uma variável, podemos fazê-lo usando a sintaxe de chave dupla {{ }}

{{ $user }}

Ciclos e estruturas

Se quisermos usar ciclos e estruturas condicionais, podemos usar diretivas. As diretivas Blade são precedidas por um sinal de arroba (@) e, em seguida, o nome da diretiva:

```
@foreach ($users as $user)
    <li>{{ $user }}</li>
@endforeach
```

Nós também podemos usar a diretiva @if:

```
@if (!empty($users))
    ...
@endif
```

A diretiva @if pode ser usada junto com um bloco else (usando @else):

```
@if (!empty($users))
    ...
@else
    <p>Não há usuários registrados.</p>
@endif
```

Podemos usar a diretiva @elseif, que como o próprio nome sugere nos permite usar um bloco else if:

```
@if (!empty($users))
   ...
@elseif ($users < 3)
   <p>Há menos de três usuários registrados.</p>
@else
   <p>Não há usuários registrados.</p>
@endif
```

Blade também possui a diretiva @unless, que funciona como uma condição inversa:

```
@unless (empty($users))
   <ul>
      @foreach ($users as $user)
         <li>{{ $user }}</li>
      @endforeach
   </ul>
@else
   <p>Não há usuários registrados.</p>
@endunless
```

No exemplo anterior, queremos mostrar a lista de usuários, a menos que a lista esteja vazia. Caso contrário, queremos mostrar a mensagem do bloco else.

Também podemos usar a diretiva @empty, que é uma maneira mais curta de escrever @if (vazio (...))

```
@empty($users)
   <p>Não há usuários registrados.</p>
@else
   <ul>
      @foreach ($users as $user)
         <li>{{ $user }}</li>
      @endforeach
   </ul>
@endempty
```

Neste caso, é claro, nós investimos os blocos dentro do condicional.

Além de @foreach, também podemos usar @for:

```
@for ($i = 0; $i < 10; $i++)
    O valor atual é {{ $i }}
@endfor
```

Com a diretiva @forelse, podemos atribuir uma opção padrão a um ciclo foreach sem usar blocos aninhados:

```
@forelse ($users as $user)
    <li>{{ $user }}</li>
@empty
    <li>Não há usuários registrados.</li>
@endforelse
```

Quando usamos a sintaxe de duas chaves, o Blade nos protege automaticamente contra ataques XSS.

Visualizações em cache

O Laravel compila e armazena em cache nossas visualizações, portanto, o uso do mecanismo de modelo Blade não afeta o desempenho de nosso aplicativo. Você pode ver essas visualizações compiladas no diretório /storage/framework/views. Usando o comando view:clear podemos deletar as views em cache do terminal:

```
php artisan view:clear
```

Na pasta resources/views renomeie o nome do arquivo users.php para users.blade.php

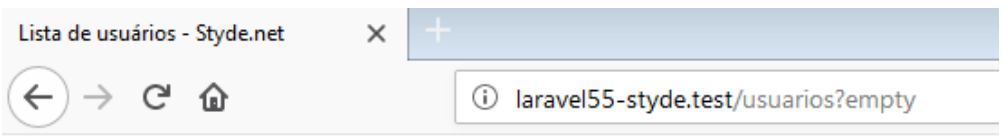
E altere o seu conteúdo para:

resources/views/users.blade.php

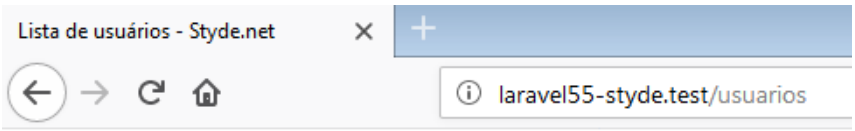
```
<!doctype html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Lista de usuários - Styde.net</title>
</head>
<body>
<h1> {{ $title }} </h1>

<ul>
   @foreach ($users as $user)
      <li>{{ $user }}</li>
   @endforeach
</ul>
</body>
</html>
```

app/Http/Controllers/UserController

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
   public function index()
   {

      if (request()->has('empty')) {
         $users = [];
      } else {
         $users = [
            'Joel', 'Ellie', 'Tess', 'Tommy', 'Bill',
         ];
      }

      $title = 'Lista de usuários';

      return view('users', compact('title', 'users'));

   }

   public function show($id)
   {
      return "Exibindo detalhes do usuário: {$id}";
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```



Lista de usuários

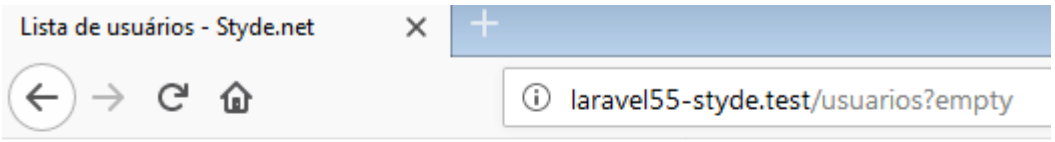


Lista de usuários

- Joel
- Ellie
- Tess
- Tommy
- Bill

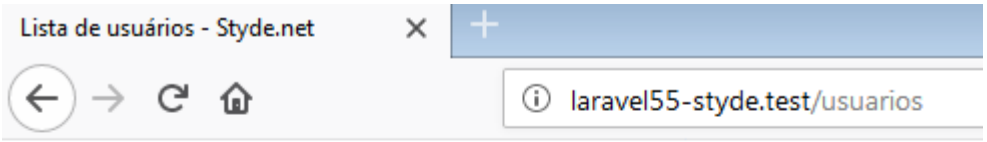
resources/views/users.blade.php

```
<!doctype html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Lista de usuários - Styde.net</title>
</head>
<body>
<h1> {{ $title }} </h1>

<hr>

<ul>
   @forelse ($users as $user)
      <li>{{ $user }}</li>
   @empty
      <li>Não há usuários registrados.</li>
   @endforelse
</ul>

</body>
</html>
```



Lista de usuários

- Não há usuários registrados.



Lista de usuários

- Joel
- Ellie
- Tess
- Tommy
- Bill

tests/Feature/UsersModuleTest.php

```php
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{
    /** @test */
    function it_shows_the_users_list()
    {
        $this->get('/usuarios')
            ->assertStatus(200)
            ->assertSee('Lista de usuários')
            ->assertSee('Joel')
            ->assertSee('Ellie');
    }

    /** @test */
    function it_shows_a_default_message_if_the_users_list_is_empty()
    {
        $this->get('/usuarios?empty')
            ->assertStatus(200)
            ->assertSee('Não há usuários registrados.');
    }

    /** @test */
    function it_loads_the_users_details_page()
    {
        $this->get('/usuarios/5')
            ->assertStatus(200)
            ->assertSee('Exibindo detalhes do usuário: 5');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar novo usuário');
    }
}
```


C:\laragon\www\laravel55-styde
λ t
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.
.... 8 / 8 (100%)
Time: 663 ms, Memory: 10.00MB
OK (8 tests, 16 assertions)
C:\laragon\www\laravel55-styde
λ

Aula 08 - Introdução ao manejo de base de dados no Laravel

As bases de dados são um dos aspectos mais importantes de um projeto. No entanto, o processo de projetar, criar e assumir o controle pode ser bastante entediante. Felizmente, o Laravel fornece um mecanismo chamado Migrations, com o qual podemos projetar a estrutura de nosso banco de dados e manter seu histórico de mudanças durante o desenvolvimento do projeto.

O que são migrations?

As migrations são um mecanismo fornecido pelo Laravel com o qual podemos ter um tipo de controle de versão sobre as mudanças na estrutura do nosso banco de dados. Com as migrations, podemos projetar essa estrutura usando PHP e programação orientada a objetos, sem a necessidade de escrever código SQL.

As migrações são agnósticas para o mecanismo de banco de dados que seu projeto usa. Ao criar um esquema, as migrations criarão as tabelas para o mecanismo de banco de dados que configuramos, mas essas mesmas migrações podem ser usadas com outros mecanismos de bancos de dados diferentes. Ou seja, podemos usar o mesmo esquema em vários mecanismos de banco de dados (desde que o mecanismo seja suportado pelo Laravel).

Por padrão, as migrations estão no diretório database/migrations. Cada migration é um arquivo .php que inclui no nome do arquivo a data e a hora em que a migration foi criada (no formato de registro de data e hora) e o nome da migration. Os mecanismos de banco de dados suportados por padrão são MySQL, PostgreSQL, SQLite e SQL Server. Além disso, existem componentes de terceiros que suportam outros tipos.

Migrations default

Ao criar um novo projeto, o Laravel inclui duas migrations por padrão:

- 2014_10_12_000000_create_users_table.php
- 2014_10_12_100000_create_password_resets_table.php

Uma migração é apenas uma classe PHP que se estende da classe Migration. O nome da classe corresponde ao nome do arquivo, no caso de 2014_10_12_000000_create_users_table.php, encontramos, mas com o formato "studly case" (a primeira letra de cada palavra em letras maiúsculas, começando com uma letra maiúscula) em vez de separadas por hífens:

```
class CreateUsersTable extends Migration
{
   // ...
}
```

Métodos de uma migration

Dentro da classe da migration, encontramos dois métodos, up () e down ():

```
class CreateUsersTable extends Migration
{
   public function up()
   {
      // ...
   }

   public function down()
   {
      // ...
   }
}
```

No método up(), vamos especificar o que queremos que nossa migration faça. Para colocá-lo de alguma forma, de que maneira queremos que nosso banco de dados "evolua" quando essa migration é executada. Normalmente, adicionaremos tabelas ao banco de dados, mas também podemos adicionar colunas a uma tabela existente ou podemos até gerar uma migration para remover uma tabela ou coluna que não precisamos mais.

Para criar uma tabela, chamamos o método create do Facade Schema, passando como primeiro argumento o nome da tabela que queremos criar (neste caso, users) e como segundo argumento, uma função anônima que recebe um objeto da classe Blueprint como argumento. Com os métodos que este objeto fornece, nós projetamos a estrutura da tabela:

```
public function up()
{
   Schema::create('users', function (Blueprint $table) {
      $table->increments('id');
      $table->string('name');
      $table->string('email')->unique();
      $table->string('password');
      $table->rememberToken();
      $table->timestamps();
   });
}
```

É uma boa prática que o nome do arquivo de migration corresponda ao que estamos fazendo na classe.

O método down() nos permite reverter ou "retornar" a operação executada no método up(). No caso de CreateUsersTable, down() nos permitirá excluir a tabela de usuários (usando o método dropIfExists do Facade Schema):

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

Construtor de esquema (Schema Builder)

A classe Blueprint nos permite construir nossas tabelas com uma interface orientada a objeto, através de diferentes métodos, por exemplo:

- $table->string('nome_da_coluna') permite criar una coluna do tipo VARCHAR (cadeia de texto).
- $table->integer('nome_da_coluna') permite criar una coluna do tipo INTEGER (inteiro).

Podemos encadear métodos para especificar características adicionais, por exemplo:

$table->integer('nome_coluna')->unsigned()->default(0);

cria uma coluna de inteiro sem sinal cujo valor padrão é 0.

Métodos helpers (auxiliares)

Uma das muitas vantagens de usar o construtor de esquema do Laravel é que ele inclui métodos helpers que facilitam tarefas comuns e evitam a necessidade de duplicar código, por exemplo, o método:

$table->timestamps();

adicionará 2 colunas: created_at e updated_at do tipo timestamp (data e hora) à tabela em questão. Essas colunas são bastante típicas para armazenar o momento em que um registro foi criado ou atualizado.

Em programação um helper não é nada mais do que uma função de suporte que permite resolver tarefas genéricas / comuns.

Configurar o banco de dados

A configuração do banco de dados é feita no arquivo .env que está dentro do diretório principal do nosso projeto:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel55-styde
DB_USERNAME=root
DB_PASSWORD=
```

Em DB_DATABASE colocamos o nome do nosso banco de dados, no DB_USERNAME o usuário do banco de dados e no DB_PASSWORD a senha do banco de dados.

Executando as migrations

Com o comando do Artisan migrate, podemos executar todas as migrations:

```
php artisan migrate
```

Ao fazer isso, o Laravel criará automaticamente nossas tabelas no banco de dados. Ao executar nossas migrations pela primeira vez, o Laravel criará uma tabela chamada migrations onde salvará as informações das migrações que foram executadas.

Na lição, mencionei que Schema and Route são Facades. No Laravel as Facades são uma forma de acessar as classes internas do framework com uma interface estática e fácil de usar. Mas isso não significa que o Laravel é formado apenas por classes estáticas, pelo contrário, o Laravel usa práticas muito boas de programação orientada a objetos como a injeção de dependência, ao mesmo tempo que inclui auxiliares Facades e funções helpers para facilitar a interação com as classes do framework.

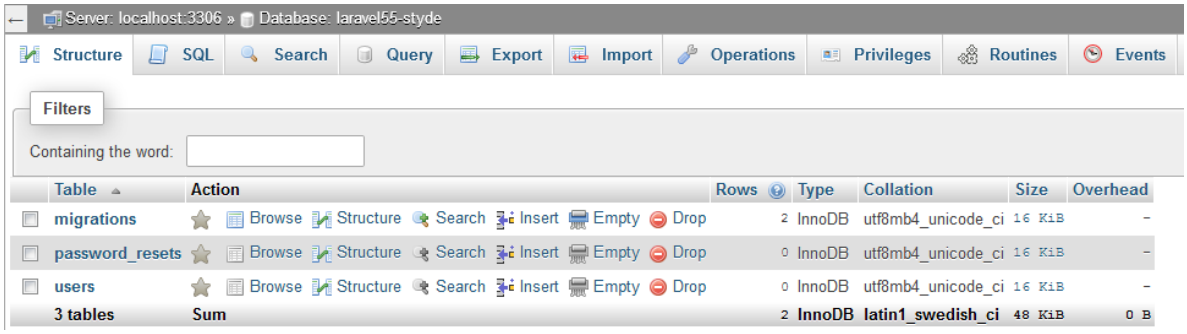
.env

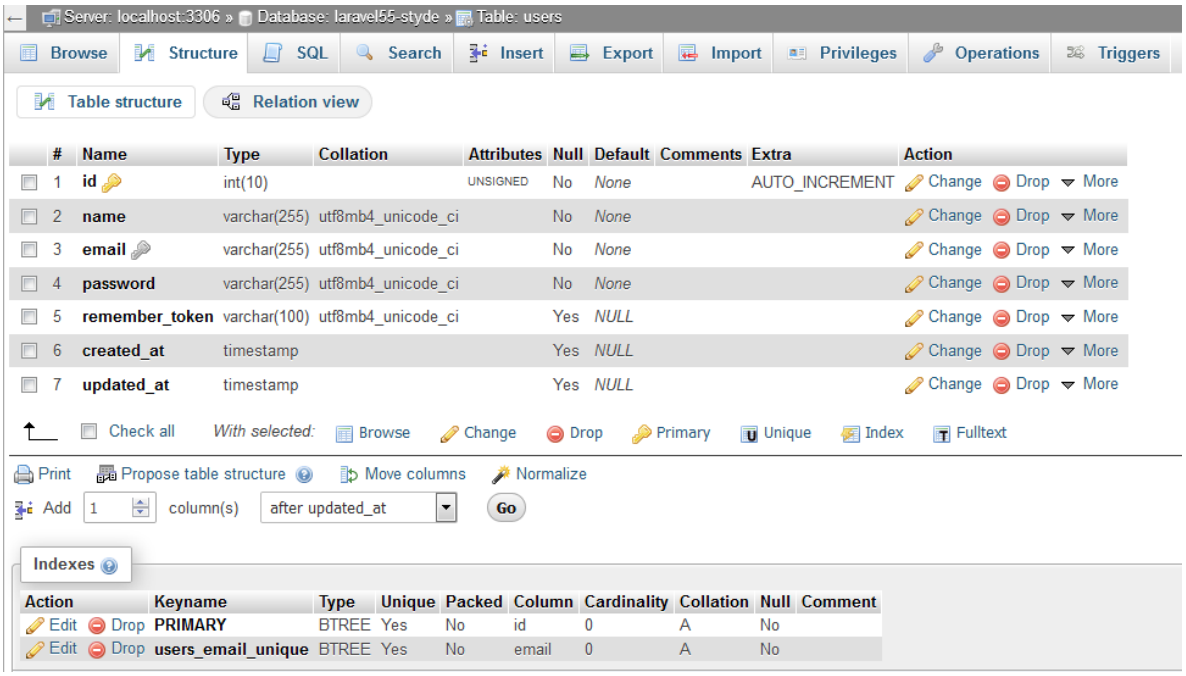
```
APP_NAME=Styde
APP_ENV=local
APP_KEY=base64:M29FqBdy1ir30TerdGYjv3YPRsMN+3k8vDvIgM+stTw=
APP_DEBUG=true
APP_URL=http://laravel55-styde.test/

LOG_CHANNEL=stack

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel55-styde
DB_USERNAME=root
DB_PASSWORD=
```

Para criar as tabelas, entre com o comando:

php artisan migrate

```
C:\laragon\www\laravel55-styde
λ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table

C:\laragon\www\laravel55-styde
λ
```




Server: localhost:3306 » Database: laravel55-styde » Table: users
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Triggers
Table structure
Relation view
Name Type Collation Attributes Null Default Comments Extra Action
1 id int(10) UNSIGNED No None AUTO_INCREMENT Change Drop More
2 name varchar(255) utf8mb4_unicode_ci No None Change Drop More
3 email varchar(255) utf8mb4_unicode_ci No None Change Drop More
4 password varchar(255) utf8mb4_unicode_ci No None Change Drop More
5 remember_token varchar(100) utf8mb4_unicode_ci Yes NULL Change Drop More
6 created_at timestamp Yes NULL Change Drop More
7 updated_at timestamp Yes NULL Change Drop More
Check all
With selected:
Browse
Change
Drop
Primary
Unique
Index
Fulltext
Print
Propose table structure
Move columns
Normalize
Add 1 column(s) after updated_at Go
Indexes
Action Keyname Type Unique Packed Column Cardinality Collation Null Comment
Edit Drop PRIMARY BTREE Yes No id 0 A No
Edit Drop users_email_unique BTREE Yes No email 0 A No

Aula 09 - Modificar tabelas já existentes com migrations do Laravel

Nesta lição, você aprenderá como pode modificar tabelas existentes através do sistema de migrations do Laravel de duas maneiras: modificando uma migration existente ou criando uma nova migration. Para conseguir isso, usaremos vários comandos do Artisan, como migrate: refresh, migrate: reset e migrate: rollback e, claro, criaremos a nova migration com o comando make: migration.

Modificando uma tabela

Para adicionar uma coluna a uma tabela, modificamos a migration existente. No nosso caso, adicionamos um campo de profissão com um limite de 100 caracteres aos usuários da tabela:

```
<?php
//...
Schema::create('users', function (Blueprint $table) {
    $table->increments('id');
    $table->string('name');
    $table->string('email')->unique();
    $table->string('profession', 100)->nullable(); //Adicionamos uma nova coluna
    $table->string('password');
    $table->rememberToken();
    $table->timestamps();
});
```

Encadear nullable() indica que a coluna pode conter valores nulos, ou seja, seu valor é opcional.

Comando reset

O comando do Artisan migrate:reset irá reverter as migrações executadas anteriormente. Em nosso exemplo, você excluirá as tabelas users e password_resets.

```
php artisan migrate:reset
```

Depois de executar este comando, podemos executar o comando php artisan migrate para recriar as tabelas.

Modificando um campo existente

Para modificar um campo existente, por exemplo, o limite de caracteres do campo profissão, adicionamos o novo valor na migration:

```
<?php
//...
Schema::create('users', function (Blueprint $table) {
   // ...
   $table->string('profession', 50)->nullable(); // Alteramos o límite de 100 a 50
   // ...
});
```

Para que essas mudanças entrem em vigor, devemos executar o comando do Artisan:

```
php artisan migrate:refresh
```

Comando migrate:refresh

O comando migrate:refresh primeiro executará uma redefinição de todas as migrations (chamando o método down()) e, em seguida, executará novamente as migrações (chamando o método up()):

```
php artisan migrate:refresh
```

Modificar migrations

Ao fazer uma modificação na migration original, temos o problema de que os comandos reset e refresh eliminarão o conteúdo das tabelas no banco de dados. Para evitar isso, podemos (usando o comando make:migration) criar uma nova migração e adicionar as modificações de que precisamos a partir daí:

```
php artisan make:migration add_profession_to_users
```

É uma boa prática que o nome da migration seja descritivo e se refira ao que vamos fazer. Nesse caso, add_profession_to_users indica que queremos incluir o campo profession na tabela users.

Dentro do método up() da migration, em vez de usar o método create do Facade Schema, usaremos table e passaremos como primeiro argumento o nome da tabela que queremos modificar:

```php
<?php
//...
Schema::table('users', (Blueprint $table)) {
   $table->string('profession', 50)->nullable();
});
```

Dentro da função, indicamos o campo que queremos adicionar.

No método down() especificamos a ação inversa, nesse caso, eliminamos a coluna que adicionamos no método up():

```php
<?php
//...
Schema::table('users', function (Blueprint $table) {
   $table->dropColumn('profession');
});
```

Com o método dropColumn, eliminamos a coluna especificada como um argumento da tabela.

Indicando a posição de uma coluna

Ao modificar uma migration usando Schema::table e adicionando uma coluna, ela será adicionada ao final da tabela. Podemos conseguir que a coluna seja criada na posição que indicamos usando o método after:

```php
<?php
//...
Schema::table('users', function (Blueprint $table)) {
   $table->string('profession', 50)->nullable()->after('password');
});
```

Neste caso, indicamos que queremos adicionar a coluna profession depois do campo password.

Comando Rollback

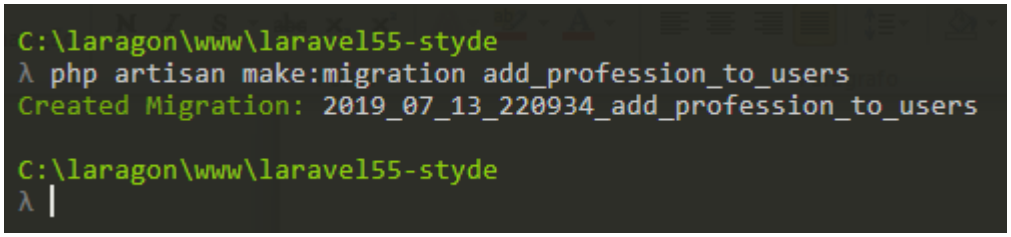
Usando o comando de Artisan migrate:rollback do Laravel retornará o último lote de migrations executadas:

```
php artisan migrate:rollback
```

Criando o campo profession na tabela users

Entre com o comando:

php artisan make:migration add_profession_to_users



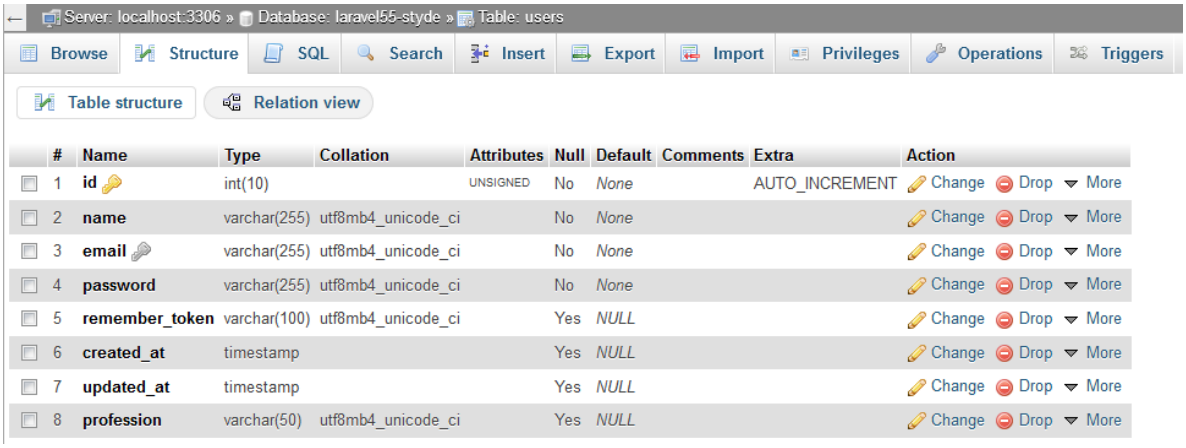
database/migrations/2019_07_13_220934_add_profession_to_users.php

```
<?php

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

class AddProfessionToUsers extends Migration
{
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('profession', 50)->nullable();
        });
    }

    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('profession');
        });
    }
}
```

Agora, entre com o comando:

php artisan migrate

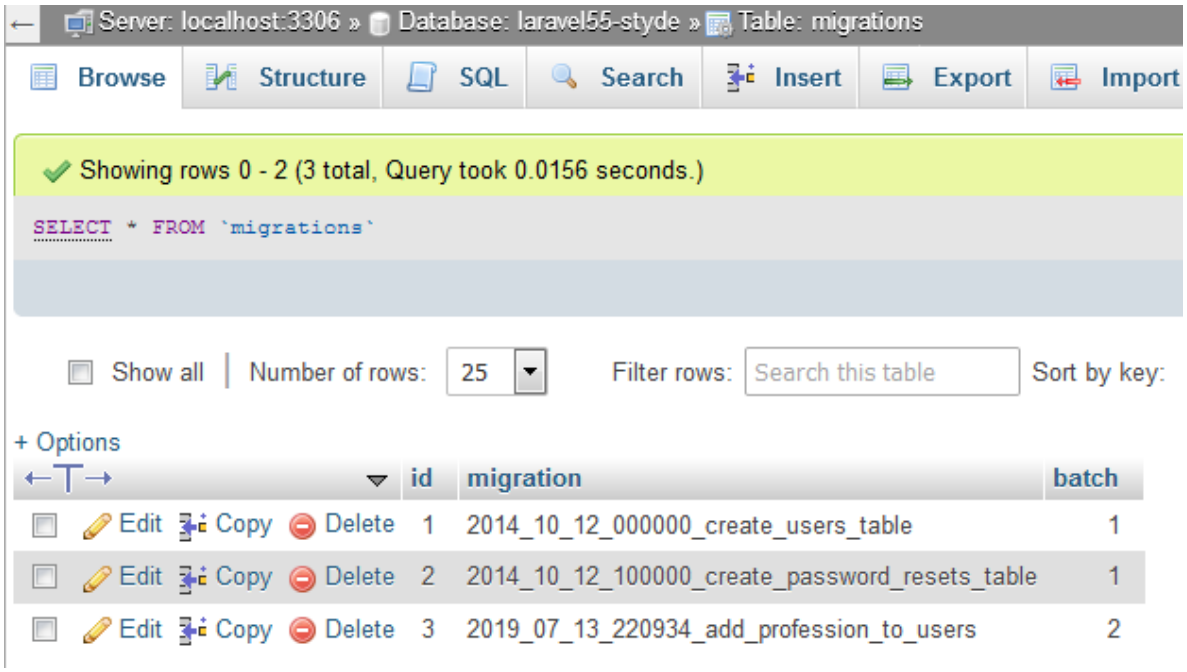
```
C:\laragon\www\laravel55-styde
λ php artisan migrate
Migrating: 2019_07_13_220934_add_profession_to_users
Migrated:  2019_07_13_220934_add_profession_to_users

C:\laragon\www\laravel55-styde
λ
```



Alterando a posição do campo na tabela

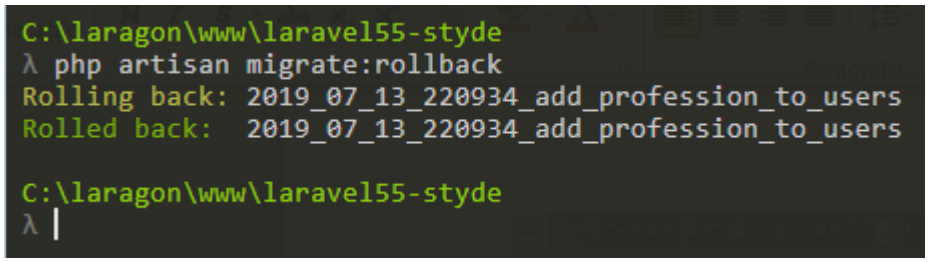
database/migrations/2019_07_13_220934_add_profession_to_users.php

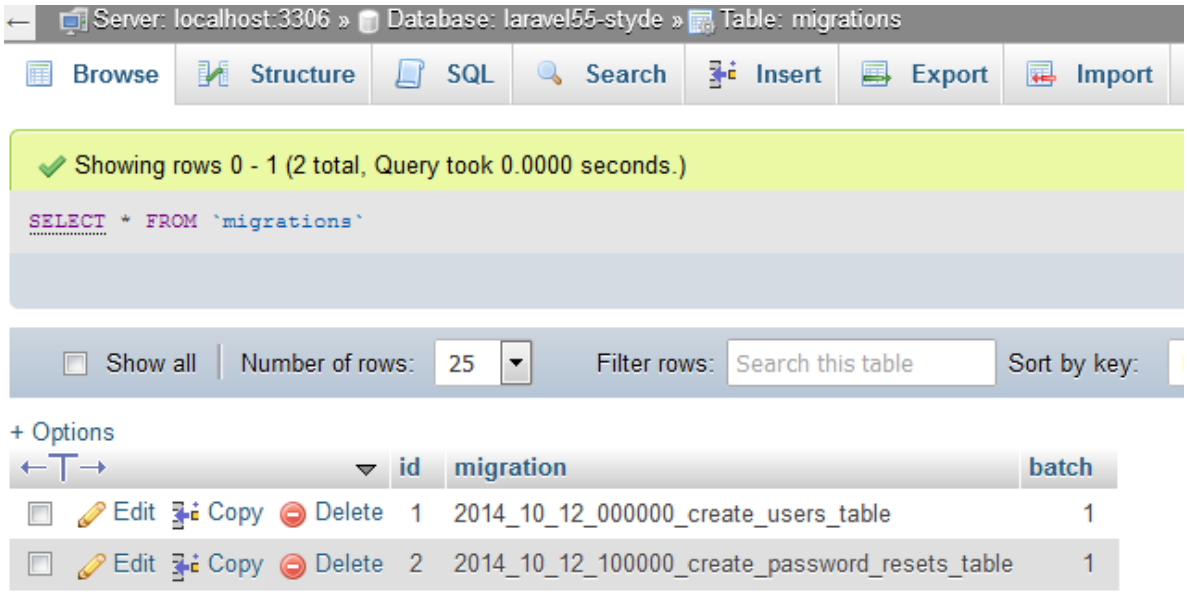
```php
<?php

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

class AddProfessionToUsers extends Migration
{
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->string('profession', 50)->nullable()->after('password');
        });
    }

    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('profession');
        });
    }
}
```



Entre com o comando:

php artisan migrate:rollback

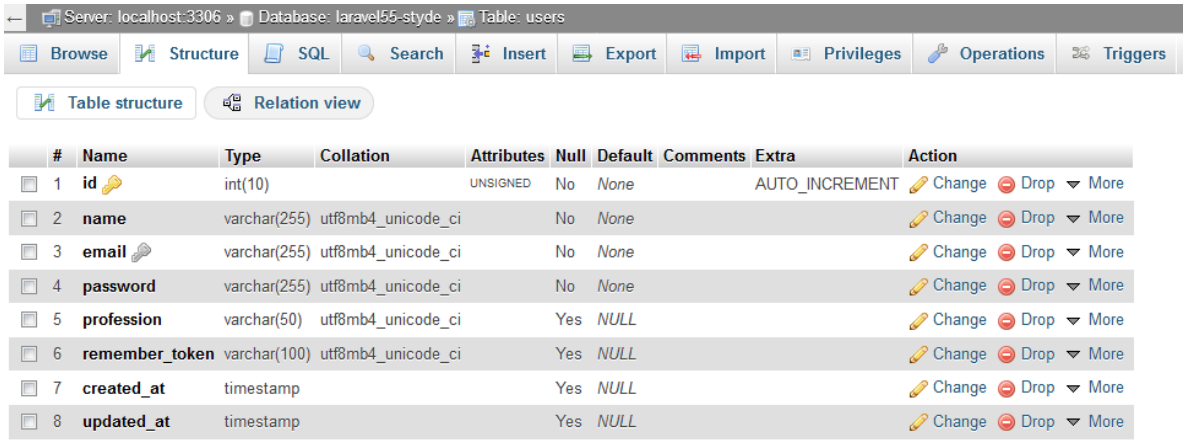




Agora, entre novamente com o comando:

php artisan migrate

```
C:\laragon\www\laravel55-styde
λ php artisan migrate
Migrating: 2019_07_13_220934_add_profession_to_users
Migrated:  2019_07_13_220934_add_profession_to_users

C:\laragon\www\laravel55-styde
λ
```



Aula 10 - Criação e associação de tabelas com uso de migrations e chaves estrangeiras no Laravel

Nesta lição veremos em profundidade o sistema de migrations do Laravel. Nesta oportunidade, mostrarei como criar novas tabelas e como criar associações entre uma tabela e outra usando chaves estrangeiras. Você também aprenderá um pouco mais sobre as convenções que podemos usar ao gerar novas migrations e sobre outros métodos para definir campos.

Convenções ao executar migrations

Ao gerar uma migration com o comando php artisan make:migration, se usarmos o seguinte formato para o nome: create_NAME OF TABLE_table, o Laravel irá gerar automaticamente o código necessário para criar a tabela:

```
php artisan make:migration create_professions_table
```

Será produzido o seguinte código boilerplate:

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

Agora só temos que definir as colunas.

Também podemos passar a opção --create caso não desejemos usar a convenção:

```
php artisan make:migration new_professions_table --create=professions
```

Para a opção create passamos como valor o nome da tabela.

Restrição de chave estrangeira

Podemos adicionar uma restrição de chave estrangeira ao nosso campo usando o método foreign():

```
Schema::create('users', function (Blueprint $table) {
   // ...
   $table->unsignedInteger('profession_id');
   $table->foreign('profession_id')->references('id')->on('professions');
   // ...
});
```

Nesse caso, indicamos que o campo profession_id vai se referir ao campo id na tabela profession. Aqui é importante que o tipo do campo profession_id corresponda ao campo id na tabela profession. Ou seja, o campo profession_id deve ser definido como um inteiro positivo, para isso usamos o método:

$table->unsignedInteger('nombre_del_campo_aqui'); ou

$table->integer('nombre_del_campo')->unsigned();

Chaves primárias

Ao projetar um banco de dados, geralmente é importante ter um campo (ou combinação de campos) que possa identificar exclusivamente cada linha. Assim como você tem um número de passaporte que é único, cada usuário ou profissão terá um identificador único (id). Neste banco de dados, usaremos identificadores do tipo de incremento automático, ou seja, a primeira linha obterá o identificador 1, o segundo 2 e assim por diante. Esses identificadores serão gerados pelo mecanismo do banco de dados.

Chaves Estrangeiras

Para associar uma tabela a outra, usaremos uma chave estrangeira. Por exemplo, na tabela de usuários, usaremos o campo profession_id, cujo valor será um identificador válido (id) de um dos registros na tabela de profissões. Desta forma, associaremos um registro (linha) à tabela de usuários com um registro (linha) da tabela de profissões. Neste tipo de relacionamento, costumamos dizer que um usuário pertence a uma profissão. Também podemos dizer que uma profissão tem muitos usuários.

Como podem haver 100 usuários que são desenvolvedores de backend ou 50 usuários que são desenvolvedores front-end, cada profissão receberá muitos usuários. Por outro lado, cada usuário só pode receber uma profissão (embora na vida real haja pessoas que tenham mais de uma profissão, em nosso sistema isso não é relevante).

Criar migrations ou modificar as existentes?

Para impedir que o número de migrations cresça sem controle, você pode modificar as migrations existentes. Lembre-se de que isso geralmente é possível em estágios iniciais de desenvolvimento em que o banco de dados não existe na produção e todos os dados são testados (não importa destruir e reconstruir o banco de dados toda vez). Se, após 3 meses do lançamento do seu projeto, você precisar adicionar um campo à tabela do usuário, nesse caso, recomendo criar uma nova migração. Porque desta forma você não só será capaz de modificar a tabela de usuários (sem excluí-la e recriá-la), mas também manterá um registro das alterações que ocorreram em diferentes versões de seu aplicativo.

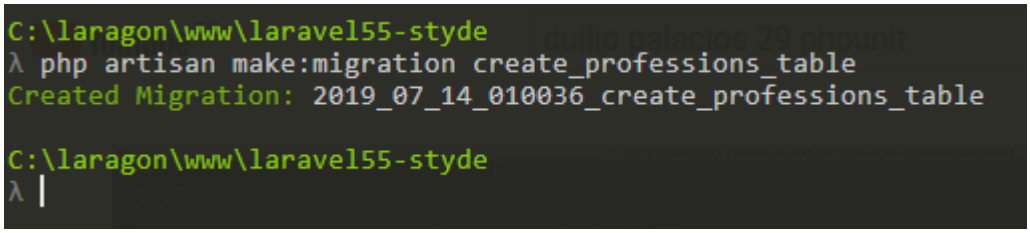
Eu mudei uma migration e nada funciona...

Nos casos em que você executar o php artisan migrate:refresh e comandos similares, sempre produza um erro, você pode corrigi-lo excluindo todas as tabelas no seu banco de dados ou executando o php artisan migrate:fresh.

Note que executar o comando php artisan migrate:fresh irá deletar todas as tabelas. Faça isso apenas se você não se importar em perder os dados (porque eles são apenas testes, etc).

Entre com o comando:

php artisan make:migration create_professions_table



database/migrations/2019_07_14_010036_create_professions_table.php

```php
<?php

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

class CreateProfessionsTable extends Migration
{
    public function up()
    {
        Schema::create('professions', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title', 100);
            $table->timestamps();
        });
    }

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

Entre com o comando:

php artisan make:migration add_profession_id_to_users --create=users

```
C:\laragon\www\laravel55-styde
λ php artisan make:migration add_profession_id_to_users --create=users
Created Migration: 2019_07_14_014456_add_profession_id_to_users

C:\laragon\www\laravel55-styde
λ
```

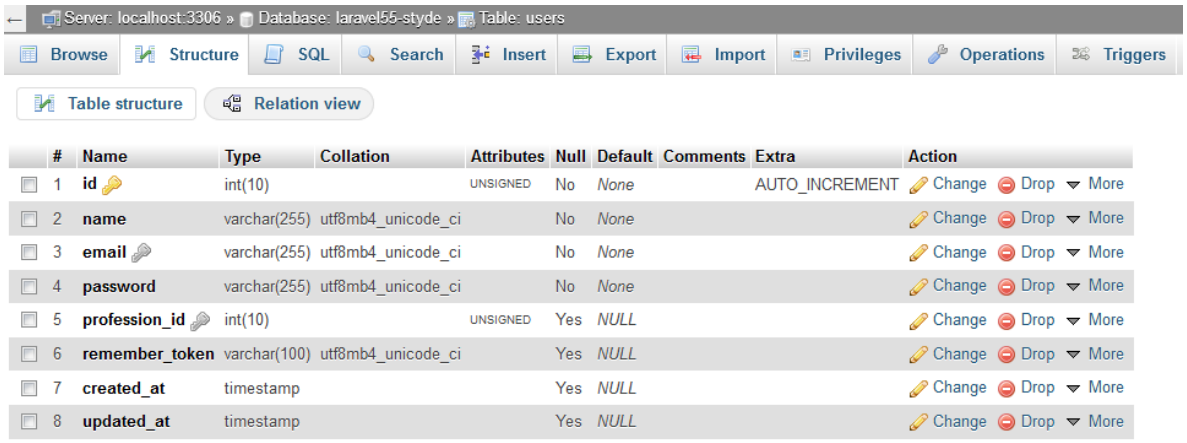
database/migrations/2019_07_14_014456_add_profession_id_to_users.php

```
<?php
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class AddProfessionIdToUsers extends Migration
{
   public function up()
   {
      Schema::table('users', function (Blueprint $table) {
         $table->unsignedInteger('profession_id')->after('password')->nullable();
         $table->foreign('profession_id')->references('id')->on('professions');
      });
   }

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

Entre com o comando:

php artisan migrate:fresh

```
C:\laragon\www\laravel55-styde
λ php artisan migrate:fresh
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
Migrating: 2019_07_14_010036_create_professions_table
Migrated:  2019_07_14_010036_create_professions_table
Migrating: 2019_07_14_014456_add_profession_id_to_users
Migrated:  2019_07_14_014456_add_profession_id_to_users

C:\laragon\www\laravel55-styde
λ
```



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**id**	int(10)		UNSIGNED	No	*None*		AUTO_INCREMENT	Change Drop More
2	**name**	varchar(255)	utf8mb4_unicode_ci		No	*None*			Change Drop More
3	**email**	varchar(255)	utf8mb4_unicode_ci		No	*None*			Change Drop More
4	**password**	varchar(255)	utf8mb4_unicode_ci		No	*None*			Change Drop More
5	**profession_id**	int(10)		UNSIGNED	Yes	*NULL*			Change Drop More
6	**remember_token**	varchar(100)	utf8mb4_unicode_ci		Yes	*NULL*			Change Drop More
7	**created_at**	timestamp			Yes	*NULL*			Change Drop More
8	**updated_at**	timestamp			Yes	*NULL*			Change Drop More

Aula 11 - Inserção de dados com o uso de seeders no Laravel 5.5

Na lição anterior, nós carregamos dados de teste usando um administrador de banco de dados, no entanto, isso tem a consequência de que toda vez que executarmos migrate:refresh ou migrate:fresh, perdemos esses dados. Nesta lição, veremos uma maneira melhor de carregar dados de teste usando os Seeders do Laravel.

Criando um seeder

Para criar um seeder, usamos o comando da artisan make:seeder seguido do nome do seeder:

```
php artisan make:seeder ProfessionSeeder
```

Ao executar este comando, um arquivo ProfessionSeeder.php será criado dentro do diretório database/seeds.

Código do seeder

Dentro do método run() do arquivo ProfessionSeeder.php nós escrevemos o código do nosso seeder:

```
class ProfessionSeeder extends Seeder
{
   public function run()
   {
      DB::table('professions')->insert([
         'title' => 'Desenvolvedor back-end',
      ]);
   }
}
```

Para inserir dados, usaremos o Query Builder (construtor de consulta SQL do Laravel). Isso inclui uma interface fluida para criar e executar consultas ao banco de dados. Para isso, vamos chamar o método de tabela do Facade DB, passando como argumento o nome da tabela com a qual queremos interagir. O método insert aceita um array associativo que representará as colunas e valores que queremos salvar na tabela.

Para usar o Facade DB:: devemos importar \Illuminate\Support\Facades\DB no início do arquivo:

```
<?php

// (namespace opcional aqui)

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

Registrando um seeder

Os seeders são registrados na classe DatabaseSeeder dentro de:

database/seeds/DatabaseSeeder.php

Dentro do método run(), chamamos o método call() passando como argumento o nome da classe do nosso seeder:

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

Neste caso, ProfessionSeeder::class retornará o nome da classe. Em vez de usar ::class também podemos passar o nome da classe como uma string de texto 'ProfessionalSeeder'.

Excluir registros

É possível que antes de executar um seeder, precisamos excluir o conteúdo existente. Para fazer isso, podemos usar o método truncate, que é responsável por esvaziar a tabela:

```
class ProfessionSeeder extends Seeder
{
    public function run()
    {
        DB::table('professions')->truncate();

        // ..
    }
}
```

Executando um seeder

Para executar seeders usamos o comando db:seed a partir do terminal:

```
php artisan db:seed
```

Caso você tenha vários seeders, você pode passar a opção --class que permite que você execute apenas o seeder passado como um argumento:

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

Você também pode executar o comando migrate:fresh ou migrate:refresh junto com os seeders passando a opção --seed:

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

Desativar revisão de chaves estrangeiras

Se uma tabela tiver uma referência de chave estrangeira, você precisará desativar a revisão de chaves estrangeiras usando uma instrução antes de esvaziar essa tabela (por exemplo, usando o método truncate).

Podemos fazer isso com a instrução:

SQL SET FOREIGN_KEY_CHECKS = [0 | 1].

No Laravel podemos executar a declaração usando o método DB::statement da seguinte maneira:

```
class ProfessionSeeder extends Seeder
{
   public function run()
   {

         // Desativamos a revisão de claves estrangeiras
         DB::statement('SET FOREIGN_KEY_CHECKS = 0;');

         DB::table('professions')->truncate();

         // Reativamos a revisão de chaves estrangeiras
         DB::statement('SET FOREIGN_KEY_CHECKS = 1;');       // ..
   }
}
```

Usando a mesma sentença, mas com o valor 1, reativamos a revisão de chaves estrangeiras após a execução do nosso seeder. Caso você queira esvaziar várias tabelas ao mesmo tempo, você pode usar o seguinte código dentro da classe DatabaseSeeder:

```
class DatabaseSeeder extends Seeder
{
   public function run()
   {
      $this->truncateTables([
         'nome_da_tabela_aqui',
         'nome_de_outra_tabela',
      ]);

      // Executar os seeders:
      $this->call(ProfessionSeeder::class);
   }

   public function truncateTables(array $tables)
   {
      DB::statement('SET FOREIGN_KEY_CHECKS = 0;');

      foreach ($tables as $table) {
         DB::table($table)->truncate();
      }

      DB::statement('SET FOREIGN_KEY_CHECKS = 1;');
   }
}
```

Agora você pode chamar o método truncateTables passando uma matriz com os nomes das tabelas que deseja esvaziar.

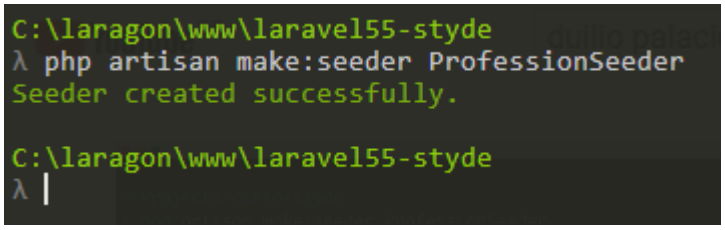
Senhas dentro de um seeder

Caso você queira inserir usuários dessa maneira, lembre-se de criptografar as senhas usando o helper bcrypt:

```
DB::table('users')->insert([
   // ..
   'password' => bcrypt('laravel')
]);
```

Entre com o comando:

php artisan make:seeder ProfessionSeeder

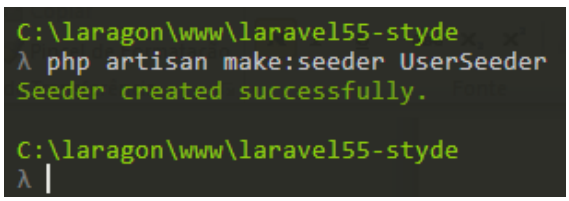


database/seed/ProfessionSeeder

```
<?php

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

class ProfessionSeeder extends Seeder
{
   public function run()
   {
      DB::table('professions')->insert([
         'title' => 'Desenvolvedor back-end',
      ]);

      DB::table('professions')->insert([
         'title' => 'Desenvolvedor front-end',
      ]);

      DB::table('professions')->insert([
         'title' => 'Web designer',
      ]);
   }
}
```

Entre com o comando:

php artisan make:seeder UserSeeder



database/seed/UserSeeder

```
<?php

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

class UserSeeder extends Seeder
{
    public function run()
    {
        DB::table('users')->insert([
            'name' => 'Duilio Palacios',
            'email' => 'duilio@styde.net',
            'password' => bcrypt('laravel'),
        ]);
    }
}
```

database/migrations/2019_07_14_010036_create_professions_table.php

```
<?php

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

class CreateProfessionsTable extends Migration
{
    public function up()
    {
        Schema::create('professions', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title', 100)->unique();
            $table->timestamps();
        });
    }

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

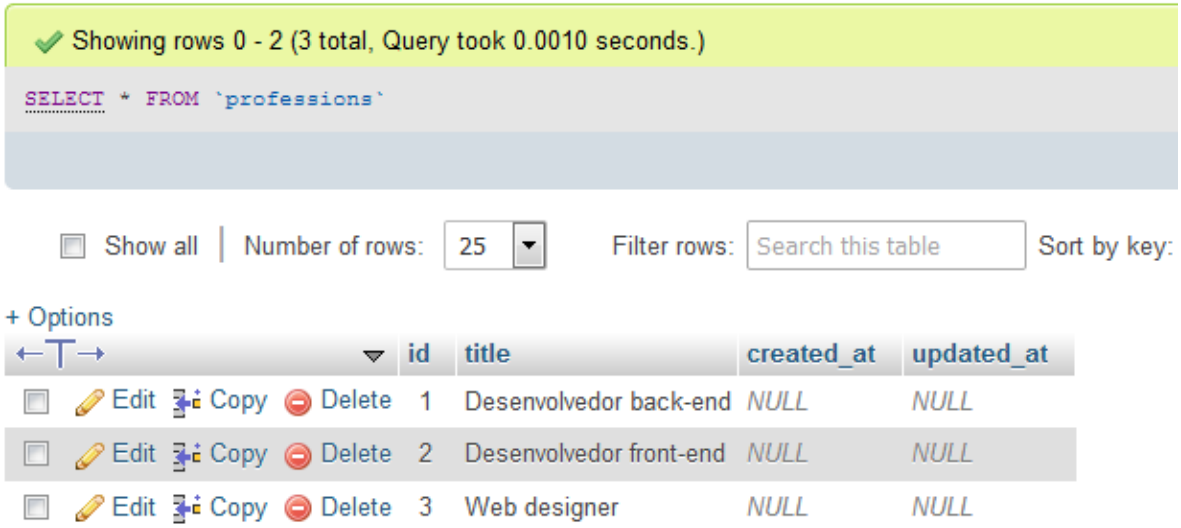
database/seed/DatabaseSeeder

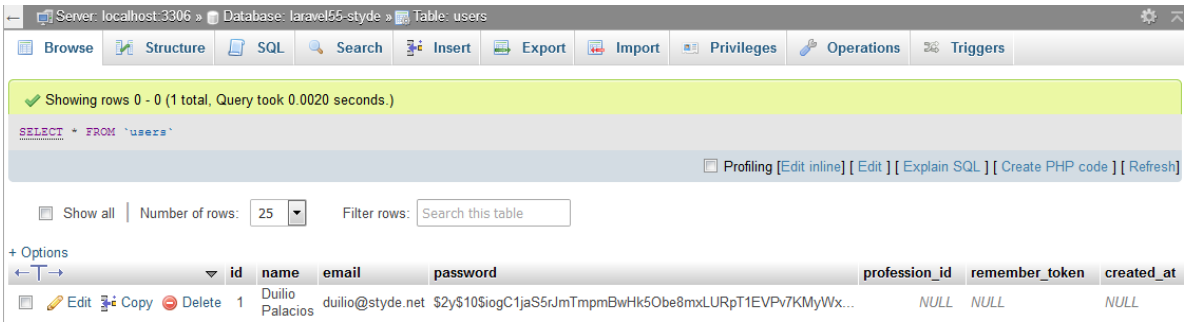
```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
   public function run()
   {
      $this->truncateTables([
         'users',
         'professions'
      ]);

      // $this->call(UsersTableSeeder::class);
      $this->call(ProfessionSeeder::class);
      $this->call(UserSeeder::class);
   }

   protected function truncateTables(array $tables)
   {
      DB::statement('SET FOREIGN_KEY_CHECKS = 0;');

      foreach ($tables as $table) {
         DB::table($table)->truncate();
      }

      DB::statement('SET FOREIGN_KEY_CHECKS = 1;');
   }
}
```

Entre com o comando:

php artisan migrate:fresh --seed

```
C:\laragon\www\laravel55-styde
λ php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
Migrating: 2019_07_14_010036_create_professions_table
Migrated:  2019_07_14_010036_create_professions_table
Migrating: 2019_07_14_014456_add_profession_id_to_users
Migrated:  2019_07_14_014456_add_profession_id_to_users
Seeding: ProfessionSeeder
Seeding: UserSeeder

C:\laragon\www\laravel55-styde
λ
```





Aula 12 - Construção de consultas SQL com Laravel

Nesta lição, veremos mais detalhadamente como o construtor Laravel SQL funciona e aproveitarei a oportunidade para mostrar como executar consultas SQL manualmente usando a estrutura e como você pode se proteger do ataque de injeção SQL ao usar o Laravel.

Método insert

Com o método DB::insert, podemos escrever consultas SQL manualmente para inserir conteúdo em nosso banco de dados. Por exemplo, o código do seeder ProfessionSeeder.php que usa o método DB::table:

```
DB::table('professions')->insert([
 'title' => 'Desenvolvedor back-end',
]);
```

Pode ser reescrito usando o método DB::insert, diretamente com o código SQL:

```
DB::insert('INSERT INTO professions (title) VALUES ("Desenvolvedor back-end")');
```

Embora o DB::insert nos forneça o mesmo resultado que o DB::table, quando realizamos consultas desse tipo e recebemos dados inseridos por um usuário, devemos ter muito cuidado, já que nos expomos a ataques de injeção de SQL.

Injeção SQL

Nosso código é vulnerável a injeções SQL quando inserimos variáveis em uma solicitação SQL. Por exemplo, se você tivesse uma consulta em que selecionasse uma série de artigos, dependendo de um autor:

```
$sql = "SELECT * FROM articles WHERE id_author = $id";
```

Esta consulta traz todos os artigos escritos por um determinado autor. No entanto, dentro dessa consulta, estamos inserindo o conteúdo diretamente ao colocar a variável $ id.

Suponha que inserimos os artigos do autor a partir de um URL, passando como argumento o id do autor (?Id=1 ou artigos/{id}), neste caso retornaríamos todos os artigos escritos pelo autor cujo id é igual a 1. No entanto, como nosso código é vulnerável, um usuário mal-intencionado poderia alterar a URL para ?id=1 UNION SELECT password FROM users. A consulta seria realmente feita desta maneira:

```
SELECT * FROM articles WHERE id_author = 1 UNION SELECT password FROM users;
```

Essa consulta seleciona todos os artigos e, em seguida, seleciona todas as senhas armazenadas na tabela users.

Ao armazenar senhas em um banco de dados, certifique-se sempre de criptografá-las.

Parâmetros dinâmicos

Para evitar ataques de injeção de SQL, podemos usar parâmetros dinâmicos. O Laravel usa o componente PHP PDO internamente e, por isso, podemos colocar marcadores em nossa consulta. O Laravel nos permite passar seus valores em uma matriz como o segundo argumento do método:

```
DB::insert('INSERT INTO professions (title) VALUES (?)', ['Desenvolvedor  back-end']);
```

Outra maneira de passar os parâmetros é usando como marcador um parâmetro de substituição com nome e como segundo argumento passamos um array associativo dos respectivos parâmetros com seus valores:

```
DB::insert('INSERT INTO professions (title) VALUES (:title)', ['title' => 'Desenvolvedor back-end']);
```

Ao fazer isso, ficaremos protegidos contra ataques de injeção de SQL, já que os parâmetros dinâmicos serão automaticamente evitados e salvos.

Método select

Usando o método DB::select, podemos construir uma consulta SQL SELECT manualmente:

```
DB::select('SELECT id FROM professions WHERE title = ?', ['Desenvolvedor back-end']);
```

Por outro lado, usando o construtor de consulta, podemos realizar uma consulta SQL do tipo SELECT, da seguinte maneira:

```
$professions = DB::table('professions')->select('id')->take(1)->get();
```

O resultado dessa consulta é um objeto da classe Illuminate\Support\Collection que encapsula a matriz de dados e isso nos traz algumas vantagens extras: uma interface orientada a objeto com a qual trabalhar e muitas funções extras. Por exemplo, podemos usar o método first para obter o primeiro resultado da consulta (no caso deste exemplo, a primeira profissão):

```
$professions->first(); // en vez de $professions[0]
```

Consultas com condicionais (WHERE)

O construtor de consulta também nos permite executar consultas condicionais usando:

```
$profession = DB::table('professions')->select('id')->where('title', '=', 'Desenvolvedor back-end')->first();
```

O operador = dentro do método where é opcional. Passando o nome da coluna junto com o valor, o Laravel assumirá que você quer usar o operador de comparação de igualdade (=):

```
where('title', 'Desenvolvedor back-end')
```

O método where também aceita um array associativo, onde indicamos o nome da coluna e o valor que esperamos encontrar:

```
where(['title' => 'Desenvolvedor back-end'])
```

Métodos dinâmicos

Nós também podemos usar métodos dinâmicos:

```
$profession = DB::table('professions')->whereTitle('Desenvolvedor back-end')->first();
```

Nesse caso, whereTitle é o equivalente a escrever where ('title', '=', 'Desenvolvedor Back-end').

Omitir o método select de DB::table

Ao omitir o método select ao usar o DB::table, podemos retornar todas as colunas:

```
$profession = DB::table('professions')->where('title', '=', 'Desenvolvedor  back-end')->first();
```

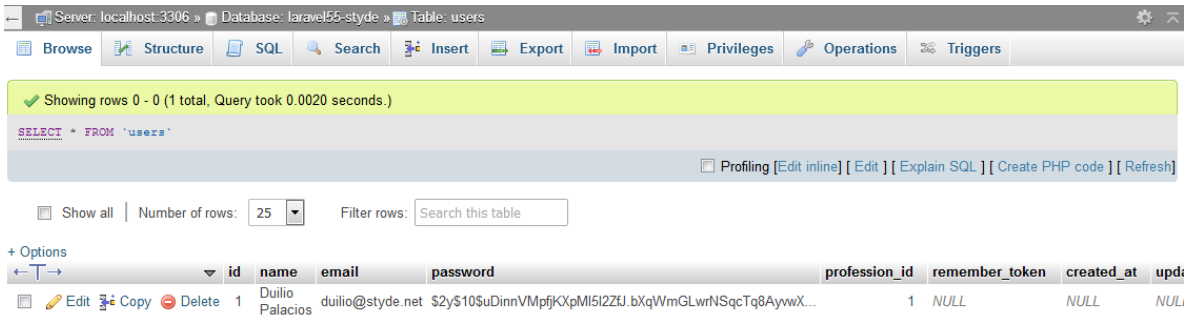
database/seed/UserSeeder

```
<?php

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

class UserSeeder extends Seeder
{
   /**
    * Run the database seeds.
    *
    * @return void
    */
   public function run()
   {
      //$professions = DB::select('SELECT id FROM professions WHERE title = ?', ['Desenvolvedor
back-end']);

      $professionId = DB::table('professions')
         ->where('title', 'Desenvolvedor back-end')
         ->value('id');

      DB::table('users')->insert([
         'name' => 'Duilio Palacios',
         'email' => 'duilio@styde.net',
         'password' => bcrypt('laravel'),
         'profession_id' => $professionId
      ]);
   }
}
```

Entre com o comando:

php artisan migrate:fresh --seed

```
C:\laragon\www\laravel55-styde
λ php artisan migrate:fresh --seed
Dropped all tables successfully.
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table
Migrating: 2019_07_14_010036_create_professions_table
Migrated:  2019_07_14_010036_create_professions_table
Migrating: 2019_07_14_014456_add_profession_id_to_users
Migrated:  2019_07_14_014456_add_profession_id_to_users
Seeding: ProfessionSeeder
Seeding: UserSeeder

C:\laragon\www\laravel55-styde
λ
```


Server: localhost:3306 » Database: laravel55-styde » Table: users
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Triggers
Showing rows 0 - 0 (1 total, Query took 0.0020 seconds.)
SELECT * FROM `users`
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Number of rows: 25
Filter rows: Search this table
+ Options
id
name
email
password
profession_id
remember_token
created_at
upda
Edit
Copy
Delete
1
Duilio Palacios
duilio@styde.net
$2y$10$uDinnVMpfjKXpMl5l2ZfJ.bXqWmGLwrNSqcTq8AywwX...
1
NULL
NULL
NUL

Aula 13 - Introdução a Eloquent, a ORM do framework Laravel

Para trabalhar com bancos de dados no Laravel, não precisamos escrever SQL manualmente nem usar o construtor de consultas. O Laravel nos permite interagir com o banco de dados em um nível muito mais alto através do Eloquent ORM. Usando o Eloquent, podemos trabalhar com modelos, que são classes que representam nossas tabelas no banco de dados e nos fornecem métodos com os quais podemos interagir de uma forma mais "eloquente" e orientada a objetos.

Criar um modelo

Podemos criar os modelos a partir do console usando o comando make:model da Artisan:

```
php artisan make:model Profession
```

A convenção é nomear o modelo com sua primeira letra em maiúscula e escrevê-lo no singular (por exemplo: "Profession" em vez de "professions"). Se quisermos adicionar duas palavras no nome, a convenção é colocar a primeira letra de cada palavra em maiúscula:

```
php artisan make:model ProfessionCategory
```

Este formato é conhecido como «Studly Caps».

Por padrão modelos serão criados no diretório app de nossa aplicação, com seu nome mais a extenão .php. No nosso caso Profession.php ou ProfessionCategory.php. Também podemos criar o modelo dentro de outro directorio, especificando o nome do diretório antes do modelo:

```
php artisan make:model Models/Profession
```

Neste caso o modelo Profession.php se encontrará em app/Models/. Se não existir o diretório especificado, Laravel irá nos criar.

Especificar a tabela relacionada ao modelo manualmente

Ao usar um modelo, não é obrigatório especificar o nome da tabela se seguirmos as convenções do Eloquent. Por exemplo, se usarmos o nome do modelo, Profession Laravel consultará a tabela professions. Se usarmos User. Laravel consultará a tabela users. Finalmente, se nosso modelo fosse ProfessionCategory, a tabela padrão seria profession_categories.

Caso o nome da tabela não seja igual ao nome do modelo, devemos especificá-lo no modelo definindo a propriedade $table:

```
class Profession extends Model
{
    protected $table = 'my_professions';
}
```

Inserir dados utilizando um modelo

Podemos inserir dados chamando ao método create do modelo.

```
\App\Profession::create([
    'title' => 'Desenvolvedor back-end',
]);
```

Importando o modelo no início do arquivo, evitamos fazer referência a \App\Profession sempre que trabalhamos com o modelo:

```
use App\Profession;
```

Depois de importar o modelo, podemos referenciá-lo diretamente:

```
Profession::create([
    'title' => 'Desenvolvedor back-end',
]);
```

Eliminar campos timestamps

Ao inserir dados usando um modelo, o Eloquent será responsável por carregar os valores dos campos created_at e updated_at da tabela. Se não quisermos usar esses campos, dentro do nosso modelo, devemos adicionar a propriedade pública $timestamps e atribuir a ela o valor false:

```
public $timestamps = false;
```

Realizar consultas

Podemos usar os modelos para fazer consultas ao banco de dados. Usando o método all(), obtemos todo o conteúdo da tabela:

```
$professions = Profession::all();
```

Também podemos encadear métodos do construtor de consultas para obter resultados mais específicos:

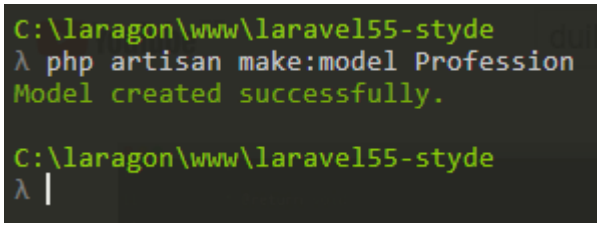
```
$professionId = Profession::where('title', 'Desenvolvedor back-end')->value('id');
```

Podemos retonar um resultado dependendo de seu id mediante o método find():

```
$profession = Profession::find(1);
```

Entre com o comando:

```
php artisan make:model Profession
```

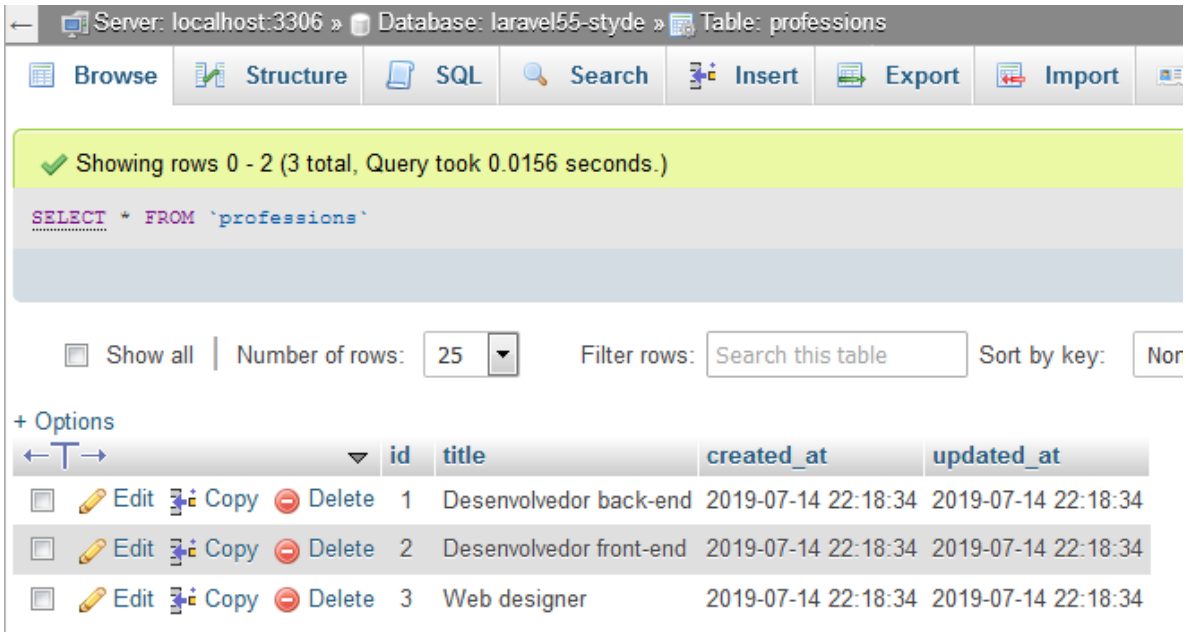


database/seed/ProfessionSeeder

```
<?php

use App\Profession;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class ProfessionSeeder extends Seeder
{
   public function run()
   {
//       DB::insert('INSERT INTO professions (title) VALUES (:title)', [
//           'title' => 'Desenvolvedor back-end',
//       ]);

//       DB::table('professions')->insert([
//           'title' => 'Desenvolvedor back-end',
//       ]);

       Profession::create([
          'title' => 'Desenvolvedor back-end',
       ]);

       Profession::create([
          'title' => 'Desenvolvedor front-end',
       ]);

       Profession::create([
          'title' => 'Web designer',
       ]);
   }
}
```

Entre com o comando:

php artisan migrate:fresh --seed


Server: localhost:3306 » Database: laravel55-styde » Table: professions
Browse
Structure
SQL
Search
Insert
Export
Import
Showing rows 0 - 2 (3 total, Query took 0.0156 seconds.)
SELECT * FROM `professions`
Show all
Number of rows: 25
Filter rows: Search this table
Sort by key:
+ Options
id
title
created_at
updated_at
Edit Copy Delete 1 Desenvolvedor back-end 2019-07-14 22:18:34 2019-07-14 22:18:34
Edit Copy Delete 2 Desenvolvedor front-end 2019-07-14 22:18:34 2019-07-14 22:18:34
Edit Copy Delete 3 Web designer 2019-07-14 22:18:34 2019-07-14 22:18:34

Aula 14 - Usando Eloquent ORM de forma interativa com Tinker

Nesta lição, aprenderemos como interagir a partir do terminal com nosso aplicativo usando o console dinâmico do Tinker. Também veremos novos métodos do Eloquent para interagir com nossos modelos e aprenderemos como criar nossos próprios métodos dentro dos modelos.

Acessando Tinker

Podemos acessar o ambiente com o comando do Artisan Tinker:

```
php artisan tinker
```

A partir do ambiente interativo podemos executar expressões PHP e também teremos as classes do nosso projeto e o framework disponível.

Retornar todos os registros com o método all()

Utilizando o método all() retornamos todos os registros associados a um modelo:

```
$professions = Profession::all();
```

Os métodos all() e get() em Eloquent retornam coleções (objetos da classe Illuminate\Database\Eloquent\Collection) as quais "envolvem" o array de resultados e nos fornecem funções e métodos adicionais, por exemplo:

Podemos obter o primeiro resultado usando o método first() e o último usando o método last(). Nós também podemos obter um resultado aleatório usando o método random():

```
$profession->first(); // Obtemos o primeiro resultado

$profession->last(); // Obtemos o último resultado

$profession->random(1); // Obtemos um resultado aleatório
```

Esses métodos da classe Collection não geram novas consultas SQL, mas operam nos resultados já encontrados.

Selecionar um campo com o método pluck()

Usando o método pluck(), podemos retornar uma nova coleção que contém uma lista de um único campo em vez de uma lista de objetos. Por exemplo, podemos obter apenas o campo de título da seguinte maneira:

```
$professions->pluck('title');
```

Devolve:

```
=> Illuminate\Support\Collection {#736
    all: [
      "Desenvolvedor back-end",
      "Desenvolvedor front-end",
      "Web designer",
    ],
  }
```

Criar uma coleção de forma manual

Con o helper collect podemos criar uma coleção de forma manual:

```
collect(['Duilio', 'Styde', 'Laravel']);
```

No entanto este será um objeto da classe:

```
Illuminate\Support\Collection
```

que representa a coleção «base» en vez de

```
Illuminate\Database\Eloquent\Collection
```

.

As coleções Eloquent se estendem da coleção base, mas fornecem alguns métodos extras associados ao ORM.

Declarar métodos no modelo

Podemos declarar métodos dentro de um modelo e usá-los ao interagir com os objetos desses modelos:

Declarando métodos não estáticos

Associamos um método para ser usado em um objeto (que representa um registro do banco de dados):

```
public function isAdmin()
{
   return $this->email === 'duilio@styde.net';
}
```

Neste caso isAdmin() devolve true se o email do usuário é igual ao valor com o qual ele está sendo comparado.

```
$user = User::find(1);

$user->isAdmin(); // Devolve true ou false
```

Declarando métodos estáticos

Associamos um método à classe do modelo como tal, que representa o acesso a uma tabela no banco de dados. Esses métodos são normalmente usados para consultas:

```
public static function findByEmail($email)
{
   return static::where('email', $email)->first();
}
```

Então você pode usar User::findbyEmail('email@aqui.com ') para procurar um usuário através do campo de email e obter um objeto User como resultado (ou null se nenhum registro for encontrado).

```
C:\laragon\www\laravel55-styde
λ php artisan tinker
Psy Shell v0.9.6 (PHP 7.2.11 — cli) by Justin Hileman
>>> use App\Profession;
>>> Profession::all();
=> Illuminate\Database\Eloquent\Collection {#2881
     all: [
       App\Profession {#2882
         id: 1,
         title: "Desenvolvedor back-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2883
         id: 2,
         title: "Desenvolvedor front-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2884
         id: 3,
         title: "Web designer",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
     ],
   }
>>>
```

```
>>> Profession::get();
=> Illuminate\Database\Eloquent\Collection {#2875
     all: [
       App\Profession {#2885
         id: 1,
         title: "Desenvolvedor back-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2886
         id: 2,
         title: "Desenvolvedor front-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2887
         id: 3,
         title: "Web designer",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
     ],
   }
>>>
```

```
>>> $professions = Profession::all();
=> Illuminate\Database\Eloquent\Collection {#2869
     all: [
       App\Profession {#2888
         id: 1,
         title: "Desenvolvedor back-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2889
         id: 2,
         title: "Desenvolvedor front-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2890
         id: 3,
         title: "Web designer",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
     ],
   }
>>> $professions->first();
=> App\Profession {#2888
     id: 1,
     title: "Desenvolvedor back-end",
     created_at: "2019-07-14 22:18:34",
     updated_at: "2019-07-14 22:18:34",
   }
>>> $professions->last();
=> App\Profession {#2890
     id: 3,
     title: "Web designer",
     created_at: "2019-07-14 22:18:34",
     updated_at: "2019-07-14 22:18:34",
   }
```

```
>>> $professions->random(2);
=> Illuminate\Database\Eloquent\Collection {#2886
     all: [
       App\Profession {#2888
         id: 1,
         title: "Desenvolvedor back-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
       App\Profession {#2889
         id: 2,
         title: "Desenvolvedor front-end",
         created_at: "2019-07-14 22:18:34",
         updated_at: "2019-07-14 22:18:34",
       },
     ],
   }
>>>
```

```
>>> $professions->pluck('title');
=> Illuminate\Support\Collection {#2887
     all: [
       "Desenvolvedor back-end",
       "Desenvolvedor front-end",
       "Web designer",
     ],
   }
>>>
```

```
>>> DB::table('professions')->where('title', 'Desenvolvedor back-end')->first();
=> {#2879
     +"id": 1,
     +"title": "Desenvolvedor back-end",
     +"created_at": "2019-07-14 22:18:34",
     +"updated_at": "2019-07-14 22:18:34",
   }
>>>
```

```
>>> use App\User;
>>> $user = User::first();
=> App\User {#2882
     id: 1,
     name: "Duilio Palacios",
     email: "duilio@styde.net",
     profession_id: 1,
     created_at: null,
     updated_at: null,
   }
>>> $user = User::find(1);
=> App\User {#2867
     id: 1,
     name: "Duilio Palacios",
     email: "duilio@styde.net",
     profession_id: 1,
     created_at: null,
     updated_at: null,
   }
>>>
```

Declarando métodos na classe User

app/User.php

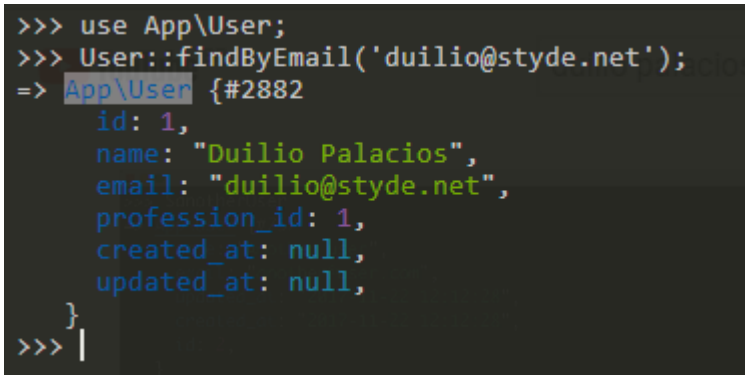
```php
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'name', 'email', 'password',
    ];

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

    public function isAdmin(){
        return $this->email === 'duilio@styde.net';
    }
}
```

```
>>> use App\User;
>>> $user = User::find(1);
=> App\User {#2892
     id: 1,
     name: "Duilio Palacios",
     email: "duilio@styde.net",
     profession_id: 1,
     created_at: null,
     updated_at: null,
   }
>>> $user->isAdmin();
=> true
>>>
```

```
>>> $user2 = User::create(['name' => 'Roberto Pinheiro', 'email' => 'roberto_pinheiro@oriontecinfo.com.br',
 'password' => bcrypt('123')]);
=> App\User {#2900
     name: "Roberto Pinheiro",
     email: "roberto_pinheiro@oriontecinfo.com.br",
     updated_at: "2019-07-15 03:08:05",
     created_at: "2019-07-15 03:08:05",
     id: 2,
   }
>>> $user2->isAdmin();
=> false
>>>
```

app/User.php

```php
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'name', 'email', 'password',
    ];

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

    public static function findByEmail($email){
        return static::where('email', $email)->first();
    }

    public function isAdmin(){
        return $this->email === 'duilio@styde.net';
    }
}
```



Aula 15 - Manejo de atributos em Eloquent ORM

Nesta lição, aprenderemos um pouco mais sobre como trabalhar com os atributos Eloquent, ou seja, como podemos ler e atribuir atributos a um modelo. A atribuição de atributos será feita usando atribuição individual e em massa, e veremos o que é o erro MassAssignmentException, como resolvê-lo e qual é a proteção que o Laravel nos oferece para evitar a injeção de atributos indesejados quando carregamos dados da solicitação de um usuário (por exemplo, por meio de um formulário ou API). Também mostrarei como converter atributos de um tipo para outro usando a propriedade $casts do Eloquent. Todos esses conceitos o aproximarão do que você precisa saber para começar a criar módulos CRUD com este excelente framework.

Ler e definir atributos

Ao trabalhar com o Eloquent, todos os atributos de um modelo (ou colunas de um registro) são tratados como se propriedades públicas da classe fossem tratadas. Por exemplo, se você deseja obter o nome de um usuário armazenado na variável $user, você pode escrever:

```
$user->name; // obtém o nome do usuário
```

Para reatribuir outro nome você pode escrever:

```
$user->name = 'Duilio';
```

Essas alterações não serão salvas automaticamente na tabela de usuários no banco de dados, você precisa executar o método save para fazer isso:

```
$user->save(); //insere ou atualza o usuário na base de dados
```

Eloquent é inteligente o suficiente para saber se deve executar um INSERT ou um UPDATE, dependendo se o usuário existe ou não, respectivamente.

A propriedade exists do Eloquent, nos permite descobrir se um modelo existe ou não, exemplo: $user->exists // retorna TRUE se o usuário já existe no banco de dados, FALSE caso contrário.

Evitar falhas de segurança devido à alocação maciça de dados

A exceção MassAssignmentException é uma maneira na qual o ORM nos protege. Uma vulnerabilidade de alocação em massa ocorre quando um usuário envia um parâmetro inesperado por meio de uma solicitação e esse parâmetro faz uma alteração no banco de dados que você não esperava. Por exemplo, um usuário poderia, usando o Chrome Developer Tools ou ferramentas semelhantes, adicionar um campo oculto chamado is_admin com o valor 1 e enviar a solicitação de registro dessa maneira. Se você não for cuidadoso com isso, qualquer usuário poderá se tornar administrador do seu aplicativo, com consequências desastrosas para o seu sistema.

Para evitar isso, dentro do modelo, adicionamos a propriedade $fillable e atribuímos como valor uma matriz com as colunas que queremos permitir que sejam carregadas em massa:

```
class User extends Model
{
   protected $fillable = ['name', 'password', 'email'];
}
```

Também temos a propriedade $guarded disponível. Como $fillable, essa propriedade terá como valor um array, mas nesse caso as colunas que indicamos são aquelas que não queremos que possam ser carregadas de forma massiva:

```
class User extends Model
{
   protected $guarded = ['is_admin'];
}
```

Atribuir um campo não preenchível

Para atribuir um valor a um campo que não esteja dentro de $fillable, podemos atribuir uma nova instância de um modelo a uma variável e, em seguida, atribuir o campo manualmente:

```
$user = new User(['name' => 'Duilio', 'password' => bcrypt('123')]);

$user->is_admin = true;
$user->save();
```

Observe que o novo usuário ($data) cria apenas um novo modelo sem persistir no banco de dados VS User::create ($data) que cria um novo modelo e o insere no banco de dados em uma única etapa.

Converter atributos

A propriedade $casts nos permite converter atributos em diferentes tipos de dados em um modelo. $casts recebe como valor um array onde a chave é o nome do atributo que será convertido e o valor do tipo para o qual queremos convertê-lo:

```
protected $casts = [
   'is_admin' => 'boolean'
];
```

Neste caso, convertemos a propriedade is_admin em boolean.

Erro de MassAssignment

```
C:\laragon\www\laravel55-styde
λ php artisan tinker
Psy Shell v0.9.6 (PHP 7.2.11 — cli) by Justin Hileman
>>> use App\Profession;
>>> Profession::create(['title'=>'Professor']);
Illuminate/Database/Eloquent/MassAssignmentException with message 'Add [title] to fillable property to allo
w mass assignment on [App/Profession].'
>>>
```

app/Profession.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Profession extends Model
{
   //protected $table = 'my_professions';
   //public $timestamps = false;
   protected $fillable = ['title'];
}
```

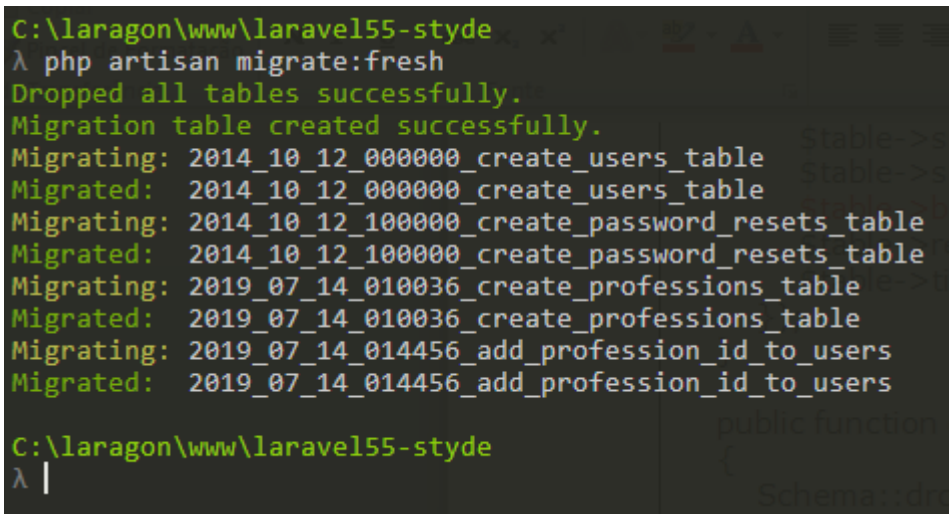
```
C:\laragon\www\laravel55-styde
λ php artisan tinker
Psy Shell v0.9.6 (PHP 7.2.11 — cli) by Justin Hileman
>>> use App\Profession;
>>> $data = ['title' => 'Profesor'];
=> [
     "title" => "Profesor",
   ]
>>> Profession::create($data);
=> App\Profession {#2873
     title: "Profesor",
     updated_at: "2019-07-15 17:39:13",
     created_at: "2019-07-15 17:39:13",
     id: 4,
   }
>>>
```

database/migrations/2014_10_12_000000_create_users_table.php

```php
<?php

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

class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->boolean('is_admin')->default(false);
            $table->rememberToken();
            $table->timestamps();
        });
    }

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

Entre com o comando:

php artisan migrate:fresh



app/User.php

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'name', 'email', 'password',
    ];

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

    protected $casts = [
        'is_admin' => 'boolean'
    ];

    public static function findByEmail($email){
        return static::where('email', $email)->first();
    }

    public function isAdmin(){
        return $this->email === 'duilio@styde.net';
    }

}
```

```
>>> $user = User::first();
=> App\User {#2874
     id: 1,
     name: "Roberto Pinheiro",
     email: "roberto_pinheiro@oriontecinfo.com.br",
     profession_id: null,
     is_admin: 0,
     created_at: "2019-07-15 18:37:58",
     updated_at: "2019-07-15 18:37:58",
   }
>>> $user->isAdmin();
=> false
>>> $user = new User(['name'=>'Duilio', 'email'=>'duilio@styde.net', 'password'=>bcrypt('123')]);
=> App\User {#2893
     name: "Duilio",
     email: "duilio@styde.net",
   }
>>> $user->is_admin = true;
=> true
>>> $user
=> App\User {#2893
     name: "Duilio",
     email: "duilio@styde.net",
     is_admin: true,
   }
>>> $user->save();
=> true
>>>
```

Server: localhost:3306 » Database: laravel55-styde » Table: users

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

Showing rows 0 - 1 (2 total, Query took 0.0020 seconds.)

```
SELECT * FROM `users`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

	id	name	email	password	profession_id	is_admin	remem
Edit Copy Delete	1	Roberto Pinheiro	roberto_pinheiro@oriontecinfo.com.br	$2y$10$Wq5fekeWr2fOUADjXy.GFulNq1Eue75oWdLifLsq5hM...	NULL	0	NULL
Edit Copy Delete	2	Duilio	duilio@styde.net	$2y$10$5vZJfHC87Pvb2u8v3ZW70uXyJr4YQ0MR/NfL592XJSO...	NULL	1	NULL

```
>>> $user->name = 'Duilio Palacios';
=> "Duilio Palacios"
>>> $user->save();
=> true
>>> $user->exists;
=> true
>>>
```

Server: localhost:3306 » Database: laravel55-styde » Table: users

Browse | Structure | SQL | Search | Insert | Export | Import | Privileges | Operations | Triggers

Showing rows 0 - 1 (2 total, Query took 0.0100 seconds.)

```
SELECT * FROM `users`
```

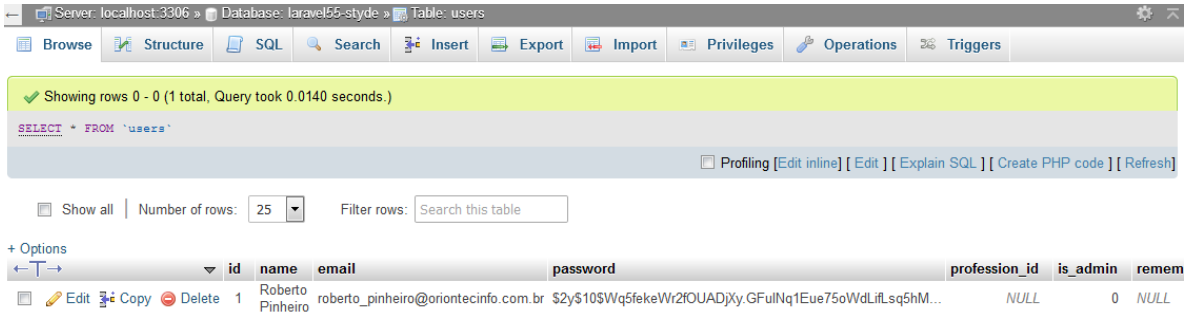
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

	id	name	email	password	profession_id	is_admin	remem
Edit Copy Delete	1	Roberto Pinheiro	roberto_pinheiro@oriontecinfo.com.br	$2y$10$Wq5fekeWr2fOUADjXy.GFulNq1Eue75oWdLifLsq5hM...	NULL	0	NULL
Edit Copy Delete	2	Duilio Palacios	duilio@styde.net	$2y$10$5vZJfHC87Pvb2u8v3ZW70uXyJr4YQ0MR/NfL592XJSO...	NULL	1	NULL

```
>>> $user->delete();
=> true
>>>
```



Aula 16 - Manejo de relações com Eloquent ORM

As tabelas em um banco de dados são freqüentemente relacionadas umas às outras. Por exemplo, um usuário pode estar associado a uma profissão e uma profissão pode ter muitos usuários associados. Usando o Eloquent ORM, podemos tornar esse processo muito mais fácil, trabalhando com relacionamentos diretamente em nossos modelos (usando programação orientada a objetos) e criando métodos personalizados que evitam a necessidade de criar consultas manualmente.

Relacionamentos «Pertence a»

O método belongsTo nos permite trabalhar com relacionamentos em que um registro pertence a outro registro. Esse método aceita como primeiro argumento o nome da classe que queremos vincular. Eloquent determina o nome da chave estrangeira a partir do nome do método (neste caso, profession) e adicionando o sufixo _id a ele:

```
public function profession()
{
   return $this->belongsTo(Profession::class);
}
```

Se no seu banco de dados o nome da chave estrangeira não seguir esta convenção, você pode passar o nome da coluna como o segundo argumento:

```
public function profession()
{
   return $this->belongsTo(Profession::class, 'id_profession');
}
```

Por outro lado, se o modelo pai não usar uma coluna id como sua chave primária ou se você quiser relacionar o modelo a uma coluna diferente, será possível passar um terceiro argumento especificando o nome da coluna que atuaria como a chave do modelo pai:

```
public function profession()
{
   return $this->belongsTo(Profession::class, 'profession_name', 'name');
}
```

Nesse caso, o Eloquent procurará a relação entre a coluna profession_name do modelo Users e a coluna name do modelo Profession.

Feito isso, usando qualquer uma das formas acima, podemos obter a profissão do usuário:

```
$user = User::first();
$user->profession;
```

Relacionamentos um-para-muitos com hasMany

Um relacionamento um-para-muitos é usado quando um modelo pode ter muitos outros modelos relacionados. Por exemplo, uma profissão pode ter um número indeterminado de usuários associados a ela. Dentro do modelo Profession, podemos dizer que uma profissão tem muitos usuários:

```
public function users()
{
   return $this->hasMany(User::class);
}
```

Agora podemos obter todos os usuários de uma profissão:

```
$profession = Profession:first();

$profession->users;
```

Os métodos que nos permitem relacionar um modelo a muitos outros sempre retornarão uma coleção, mesmo se ela estiver vazia e os métodos que nos permitem relacionar um modelo a outro retornarão o modelo ou null.

Construir consultas

Podemos construir uma consulta chamando o método de um relacionamento. Por exemplo, encadeando o método where() para users(), podemos obter todos os usuários associados a uma profissão, mas tendo a coluna is_admin como verdadeira:

```
$profession->users()->where('is_admin', true)->get();
```

Inicialmente, entre com o comando:

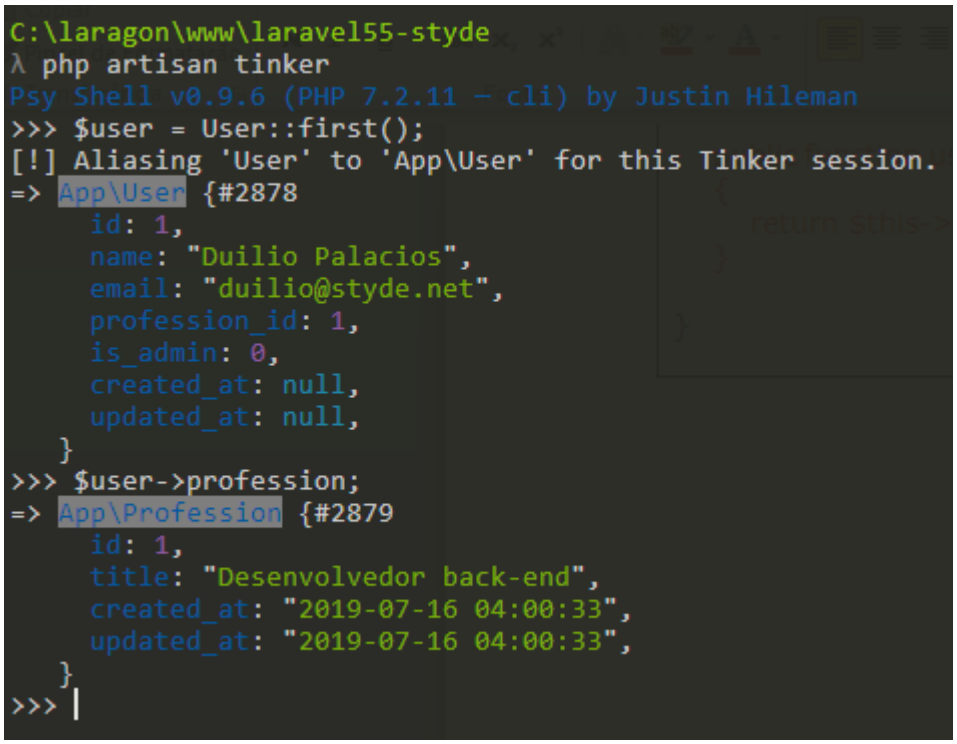
php artisan migrate:fresh --seed

```
C:\laragon\www\laravel55-styde
λ php artisan tinker
Psy Shell v0.9.6 (PHP 7.2.11 — cli) by Justin Hileman
>>> $user = User::first();
[!] Aliasing 'User' to 'App\User' for this Tinker session.
=> App\User {#2878
     id: 1,
     name: "Duilio Palacios",
     email: "duilio@styde.net",
     profession_id: 1,
     is_admin: 0,
     created_at: null,
     updated_at: null,
   }
>>> $user->profession;
=> null
>>>
```

app/User.php

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'name', 'email', 'password',
    ];

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

    protected $casts = [
        'is_admin' => 'boolean'
    ];

    public static function findByEmail($email){
        return static::where('email', $email)->first();
    }

    public function profession()
    {
        return $this->belongsTo(Profession::class);
    }

    public function isAdmin(){
        return $this->email === 'duilio@styde.net';
    }

}
```

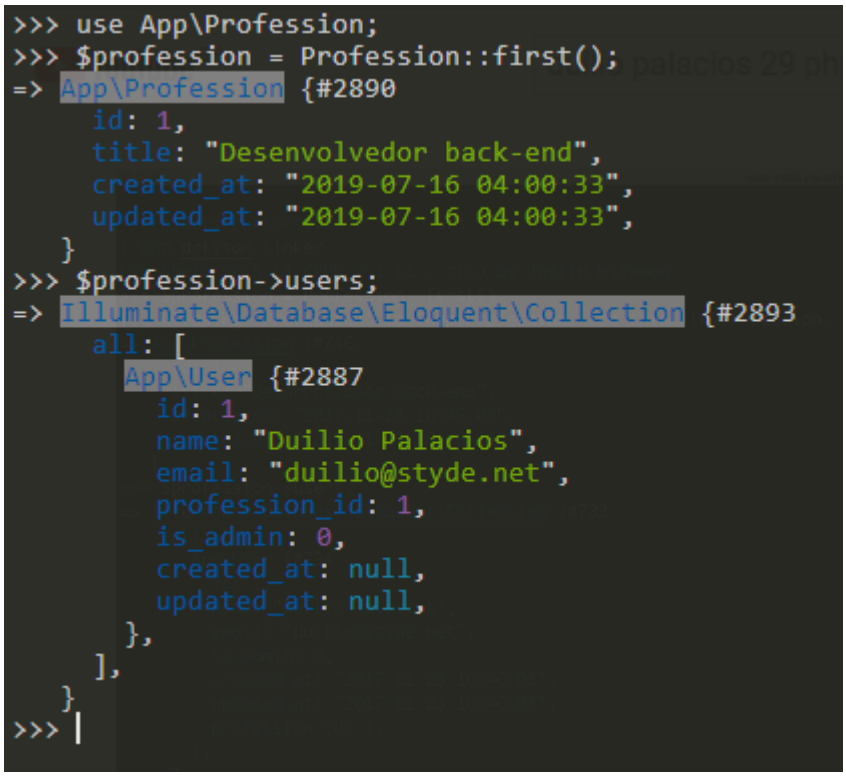
Vai retornar um **modelo**.



app/Profession.php

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Profession extends Model
{
   //protected $table = 'my_professions';
   //public $timestamps = false;
   protected $fillable = ['title'];

   public function users()
   {
      return $this->hasMany(User::class);
   }

}
```

Vai retornar uma **coleção**.


>>> use App\Profession;
>>> $profession = Profession::first();
=> App\Profession {#2890
id: 1,
title: "Desenvolvedor back-end",
created_at: "2019-07-16 04:00:33",
updated_at: "2019-07-16 04:00:33",
}
>>> $profession->users;
=> Illuminate\Database\Eloquent\Collection {#2893
all: [
App\User {#2887
id: 1,
name: "Duilio Palacios",
email: "duilio@styde.net",
profession_id: 1,
is_admin: 0,
created_at: null,
updated_at: null,
},
],
}
>>>

database/seeds/UserSeeder.php

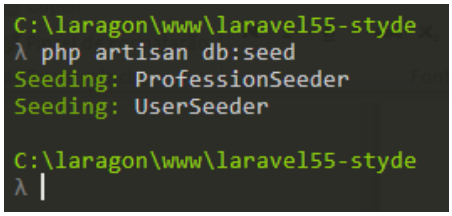
```
<?php

use App\User;
use App\Profession;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class UserSeeder extends Seeder
{
   public function run()
   {
      //$professions = DB::select('SELECT id FROM professions WHERE title = ?', ['Desaenvolvedor back-
end']);

      $professionId = DB::table('professions')
         ->where('title', 'Desenvolvedor back-end')
         ->value('id');

      User::create([
         'name' => 'Duilio Palacios',
         'email' => 'duilio@styde.net',
         'password' => bcrypt('laravel'),
         'profession_id' => $professionId,
         'is_admin' => true
      ]);

      User::create([
         'name' => 'Carlos Sanches',
         'email' => 'carlos_sanches@gmail.com',
         'password' => bcrypt('laravel'),
         'profession_id' => $professionId
      ]);
      User::create([
         'name' => 'Fernando Ribeiro',
         'email' => 'fernando_ribeiro@hotmail.com',
         'password' => bcrypt('laravel'),
         'profession_id' => null
      ]);

   }
}
```

app/User.php

```
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use Notifiable;

    protected $fillable = [
        'name', 'email', 'password',
    ];

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

    protected $casts = [
        'is_admin' => 'boolean'
    ];

    public static function findByEmail($email){
        return static::where('email', $email)->first();
    }

    public function profession()
    {
        return $this->belongsTo(Profession::class);
    }

    public function isAdmin(){
        return $this->is_admin;
    }

}
```

Entre com o comando:

php artisan db:seed



```
C:\laragon\www\laravel55-styde
λ php artisan tinker
Psy Shell v0.9.6 (PHP 7.2.11 — cli) by Justin Hileman
>>> use App\Profession;
>>> $profession = Profession::first();
=> App\Profession {#2880
     id: 1,
     title: "Desenvolvedor back-end",
     created_at: "2019-07-16 05:18:05",
     updated_at: "2019-07-16 05:18:05",
   }
>>> $profession->users;
=> Illuminate\Database\Eloquent\Collection {#2877
     all: [
       App\User {#2871
         id: 1,
         name: "Duilio Palacios",
         email: "duilio@styde.net",
         profession_id: 1,
         is_admin: 1,
         created_at: "2019-07-16 05:18:06",
         updated_at: "2019-07-16 05:18:06",
       },
       App\User {#2881
         id: 2,
         name: "Carlos Sanches",
         email: "carlos_sanches@gmail.com",
         profession_id: 1,
         is_admin: 0,
         created_at: "2019-07-16 05:18:06",
         updated_at: "2019-07-16 05:18:06",
       },
     ],
   }
>>>
```

```
>>> $profession->users()->where('is_admin', true)->get();
=> Illuminate\Database\Eloquent\Collection {#2873
     all: [
       App\User {#2888
         id: 1,
         name: "Duilio Palacios",
         email: "duilio@styde.net",
         profession_id: 1,
         is_admin: 1,
         created_at: "2019-07-16 05:18:06",
         updated_at: "2019-07-16 05:18:06",
       },
     ],
   }
>>>
```

Aula 17 - Gerar registros usando Model Factory

As Model Factories nos permitem criar registros de teste, seja para carregar nosso banco de dados com "informações falsas" ou "informações de teste" ou para criar as condições necessárias para executar testes automatizados. Nesta lição, praticaremos com a criação de modelos do Tinker e de nossos seeders usando Model Factories. Além disso, ensinarei como gerar suas Factories personalizadas, adaptá-las ao modelo correspondente e também veremos uma breve introdução ao uso do componente Faker.

Gerar um factory

Para usar um Model Factory, precisamos gerá-lo primeiro com o comando make:factory. O Factory será gerado no diretório database/factories.

```
php artisan make:factory ProfessionFactory
```

Dentro do nosso Factory especificamos o atributo ou atributos que queremos gerar de forma aleatória:

```
$factory->define(\App\Profession::class, function (Faker $faker) {
    return [
        'title' => $faker->sentence
    ];
});
```

Também é importante indicarmos o nome do modelo que queremos vincular ao Model Factory (neste caso, \App\Profession::class). Para evitar ter que adicionar o nome do modelo manualmente, podemos passar a opção --model para o comando make: factory:

```
php artisan make:factory ProfessionFactory --model=Profession
```

Ao gerar um modelo com o comando make:model, também podemos gerar um Model Factory e/ou uma migration, passando as opções -f e/ou -m, por exemplo:

```
php artisan make:model Skill -mf
```

Usando o componente PHP Faker, indicamos que o valor do título será uma sentença aleatória:

```
$factory->define(\App\Profession::class, function (Faker $faker) {
    return [
        'title' => $faker->sentence
    ];
});
```

Passando um número como o primeiro argumento a sentença, podemos indicar o número de palavras que queremos que a sentença contenha: $faker->sentence(3). Isso retornará frases com 2, 3 ou 4 palavras, ou se quisermos estritamente 3 palavras, podemos chamar o método desta forma: $faker->sentence(3, false)

Componente Faker

O componente Faker é uma biblioteca PHP que gera dados de teste para nós. Por exemplo, podemos gerar um nome:

```
$faker->name;
// 'Jazmyne Romaguera'
```

Ou um texto aleatório:

```
$faker->text;
// Dolores sit sint laboriosam dolorem culpa et autem. Beatae nam sunt fugit
// et sit et mollitia sed.
```

Inclusive números de telefone:

```
$faker->cellphone;
// 9432-5656
```

Utilizando um Model Factory

Para usar um Model Factory, devemos chamar o helper factory, especificando o nome do modelo como um argumento e, finalmente, encadeando a chamada para o método create.

```
factory(User::class())->create();
```

Isso criará um usuário no banco de dados e retornará o modelo em questão, nesse caso, um objeto da classe User:

```
App\User {
   name: "Jazmyne Romaguera",
   email: "ciara.willms@example.com",
   updated_at: "2017-11-24 15:55:32",
   created_at: "2017-11-24 15:55:32",
   id: 4,
}
```

Cada vez que executamos o método create(), criamos um novo registro aleatório. Se passarmos um array associativo para o método create(), podemos sobrescrever ou atribuir propriedades extras:

```
factory(User::class)->create([
   'profession_id' => $professionId
]);
```

Para carregar um determinado número de registros, passamos como segundo argumento a quantidade que queremos criar:

```
factory(User::class, 48)->create();
```

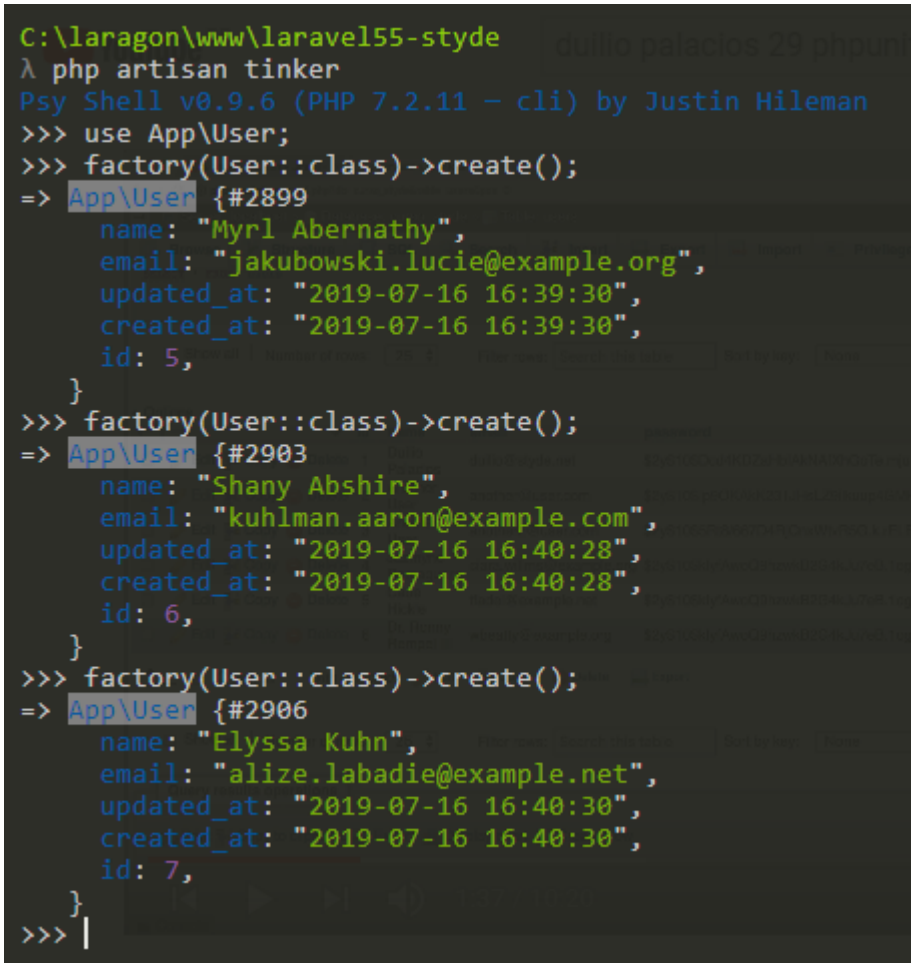
Também podemos conseguir isso usando o método times():

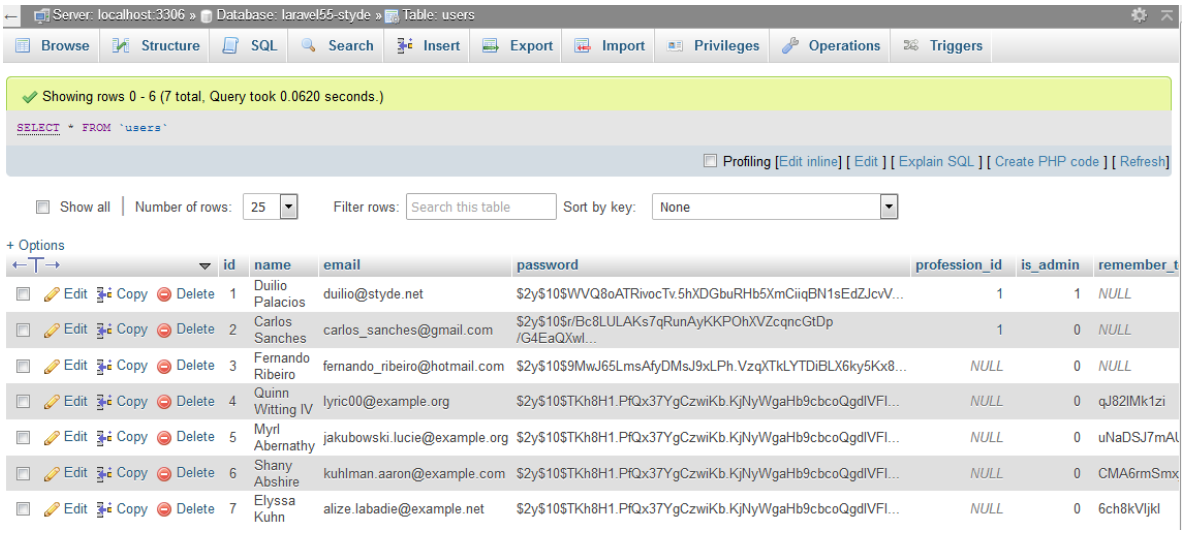
```
factory(User::class)->times(48)->create();
```

---

database/factories/UserFactory.php

```
<?php
use Faker\Generator as Faker;

$factory->define(App\User::class, function (Faker $faker) {
   static $password;
   return [
      'name' => $faker->name,
      'email' => $faker->unique()->safeEmail,
      'password' => $password ?: $password = bcrypt('secret'),
      'remember_token' => str_random(10),
   ];
});
```



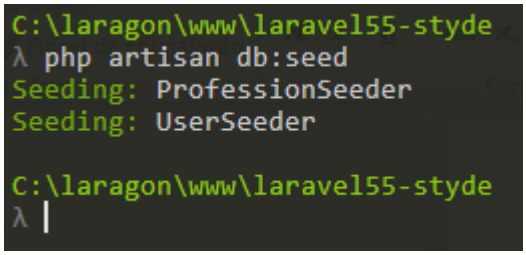


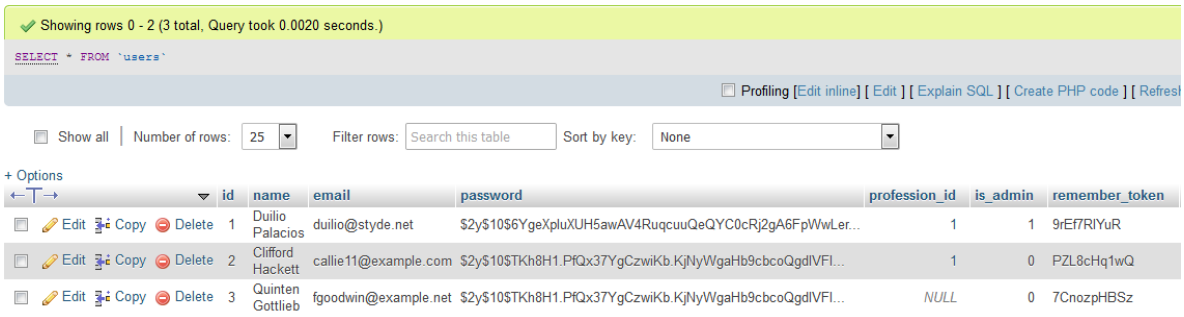
app/database/seeds/UserSeeder.php

```
<?php
use App\User;
use App\Profession;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
class UserSeeder extends Seeder
{
   public function run()
   {

      //$professions = DB::select('SELECT id FROM professions WHERE title = ?', ['Desenvolvedor back-
end']);

      $professionId = Profession::where('title', 'Desenvolvedor back-end')->value('id');

      factory(User::class)->create([
         'name' => 'Duilio Palacios',
         'email' => 'duilio@styde.net',
         'password' => bcrypt('laravel'),
         'profession_id' => $professionId,
         'is_admin' => true,
      ]);

      factory(User::class)->create([
         'profession_id' => $professionId
      ]);

      factory(User::class)->create();
   }
}
```

Entre com o comando:

php artisan db:seed



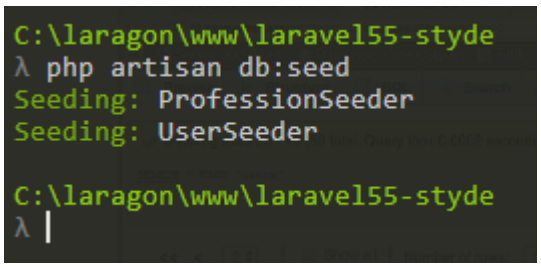


app/database/seeds/UserSeeder.php

```
<?php
use App\User;
use App\Profession;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
class UserSeeder extends Seeder
{
   public function run()
   {

      //$professions = DB::select('SELECT id FROM professions WHERE title = ?', ['Desenvolvedor back-
end']);

      $professionId = Profession::where('title', 'Desenvolvedor back-end')->value('id');

      factory(User::class)->create([
         'name' => 'Duilio Palacios',
         'email' => 'duilio@styde.net',
         'password' => bcrypt('laravel'),
         'profession_id' => $professionId,
         'is_admin' => true,
      ]);

      factory(User::class)->create([
         'profession_id' => $professionId
      ]);

      factory(User::class, 48)->create();
   }
}
```

Entre com o comando:

php artisan db:seed



	id	name	email	password	profession_id	is_admin	remem
Edit Copy Delete	1	Duilio Palacios	duilio@styde.net	$2y$10$1Cu4HGQZkfrNtme8/ChtsuY0si4sCWahpKwOHcMzAwg...	1	1	qv8M4I
Edit Copy Delete	2	Mr. Blake Yundt	emann@example.net	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	1	0	eTuTyz
Edit Copy Delete	3	Carolina Shields	grant.michelle@example.net	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	Q3AGa
Edit Copy Delete	4	Macy Weber	nwehner@example.org	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	qut5W
Edit Copy Delete	5	Willy Gutmann	hilpert.wade@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	PVB8c
Edit Copy Delete	6	Hanna Gorczany	towne.thomas@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	raCwbi
Edit Copy Delete	7	Mrs. Bettye Krajcik	larkin.salma@example.net	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	2ujrdG
Edit Copy Delete	8	Mrs. Alene Johnson III	ervin.jacobson@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	rNzRJk
Edit Copy Delete	9	Lilian Kuhn	nakia49@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	tshBg1
Edit Copy Delete	10	Monica Ziemann	jaqueline08@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	cOKDI
Edit Copy Delete	11	Dr. Anita McKenzie	esteban.luettgen@example.net	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	0tfuszU
Edit Copy Delete	12	Sylvia Frami	vanderson@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	KbjnIM
Edit Copy Delete	13	Jamarcus Strosin	gpollich@example.net	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	zbLv4A
Edit Copy Delete	14	Torrey Gibson PhD	white.sabrina@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	724QT
Edit Copy Delete	15	Manley Little	hegmann.sonia@example.com	$2y$10$TKh8H1.PfQx37YgCzwiKb.KjNyWgaHb9cbcoQgdlVFl...	NULL	0	66muY
Console Edit Copy Delete	16	Prof. Elwyn					

Entre com o comando:

php artisan make:factory ProfessionFactory --model=Profession

```
C:\laragon\www\laravel55-styde
λ php artisan make:factory ProfessionFactory
Factory created successfully.

C:\laragon\www\laravel55-styde
λ
```

database/factories/UserFactory.php

```
<?php

use Faker\Generator as Faker;

$factory->define(App\Profession::class, function (Faker $faker) {
    return [
        'title' => $faker->sentence(3, false)
    ];
});
```

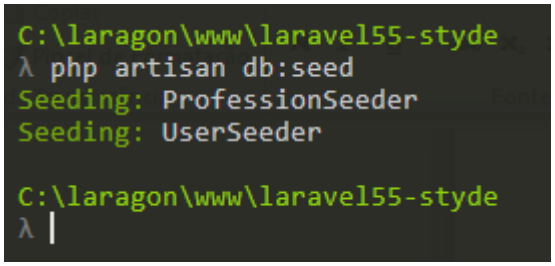
sentence(3, false): três palavras

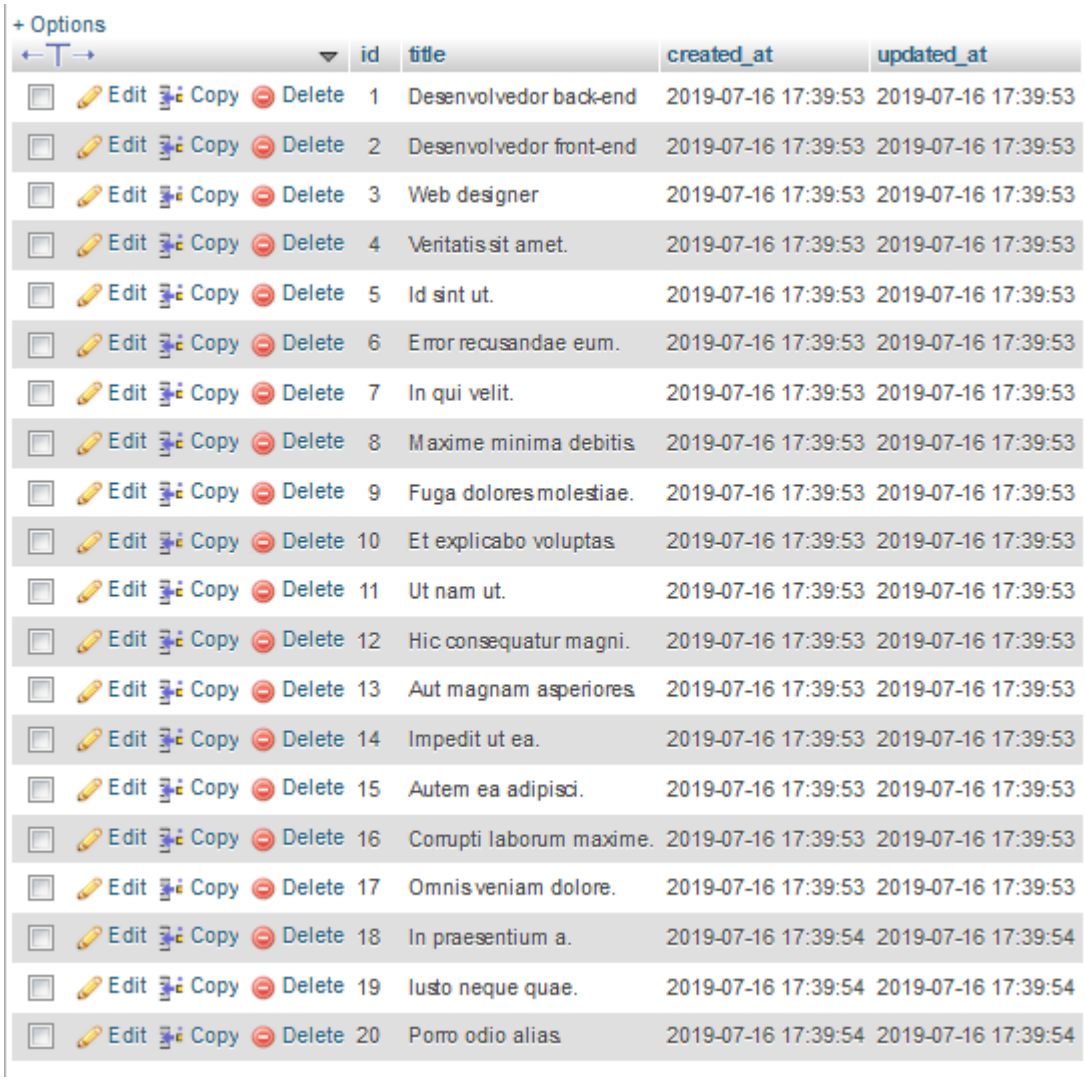
app/database/seeds/ProfessionSeeder.php

```
<?php

use App\Profession;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class ProfessionSeeder extends Seeder
{
   public function run()
   {
//       DB::insert('INSERT INTO professions (title) VALUES (:title)', [
//           'title' => 'Desenvolvedor back-end',
//       ]);

//       DB::table('professions')->insert([
//           'title' => 'Desenvolvedor back-end',
//       ]);

       Profession::create([
           'title' => 'Desenvolvedor back-end',
       ]);

       Profession::create([
           'title' => 'Desenvolvedor front-end',
       ]);

       Profession::create([
           'title' => 'Web designer',
       ]);

       factory(Profession::class)->times(17)->create();

   }
}
```

Entre com o comando:

php artisan db:seed





+ Options

	id	title	created_at	updated_at
Edit Copy Delete	1	Desenvolvedor back-end	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	2	Desenvolvedor front-end	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	3	Web designer	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	4	Veritatis sit amet.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	5	Id sint ut.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	6	Error recusandae eum.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	7	In qui velit.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	8	Maxime minima debitis.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	9	Fuga dolores molestiae.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	10	Et explicabo voluptas.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	11	Ut nam ut.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	12	Hic consequatur magni.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	13	Aut magnam asperiores.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	14	Impedit ut ea.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	15	Autem ea adipisci.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	16	Corrupti laborum maxime.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	17	Omnis veniam dolore.	2019-07-16 17:39:53	2019-07-16 17:39:53
Edit Copy Delete	18	In praesentium a.	2019-07-16 17:39:54	2019-07-16 17:39:54
Edit Copy Delete	19	Iusto neque quae.	2019-07-16 17:39:54	2019-07-16 17:39:54
Edit Copy Delete	20	Porro odio alias.	2019-07-16 17:39:54	2019-07-16 17:39:54

Entre com o comando:

php artisan make:model Skill -mf

```
C:\laragon\www\laravel55-styde
λ php artisan make:model Skill -mf
Model created successfully.
Factory created successfully.
Created Migration: 2019_07_16_174553_create_skills_table

C:\laragon\www\laravel55-styde
λ
```

Cria o modelo, a migration e o factory

Aula 18 - Introdução ao módulo CRUD de Usuários

A partir desta lição, usaremos o conhecimento que obtivemos sobre o Eloquent ORM e o gerenciamento de banco de dados com o Laravel para converter a lista estática de usuários em uma lista dinâmica.

Para continuar com esta terceira parte do Curso Laravel a partir do zero, recomendamos que você veja e analise as duas partes anteriores, onde explicamos como criar rotas, drivers, visualizações e também como criar e trabalhar com bancos de dados no Laravel.

Aula 19 - Lista dinâmica de usuários no Laravel

Nesta lição, iniciaremos o desenvolvimento do módulo CRUD de usuários, substituindo a lista estática de usuários pelos registros que estão no banco de dados. Para isso, usaremos o construtor de consultas do Laravel e o ORM Eloquent.

Obter registros do banco de dados

Substituímos a lista estática por uma lista dinâmica que carregaremos do banco de dados. Usando get() obtemos todos os registros que estão na tabela passados como um argumento para o método de tabela:

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

Necessitamos importar o facade DB no início do arquivo:

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

A variável $user dentro do ciclo @forelse não contém mais uma string de texto, mas um objeto. Não podemos imprimir esses objetos como se tratasse de uma cadeia de texto mas podemos imprimir as propriedades de cada objeto:

```
@forelse($users as $user)
  <li>{{ $user->name }}</li>
@empty
  <li>No hay usuarios registrados.</li>
@endforelse
```

Obter registros com o Eloquent

Podemos usar o Eloquent para obter os dados encontrados no banco de dados. Usando all() podemos obter todos os registros que estão na tabela:

```
$users = User::all();
```

Devemos importar o modelo no início do arquivo:

```
use App\User;
```

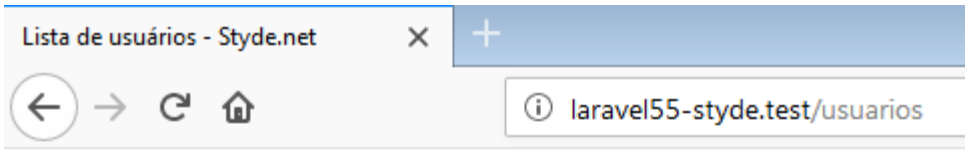
Quando você imprime diretamente $user na diretiva @forelse sem chamar qualquer propriedade, uma representação JSON de cada usuário é impressa. Isso se deve ao fato de que, ao usar o Eloquent, obtemos uma coleção de objetos em que cada objeto representa uma instância da classe User (um modelo Eloquent), que inclui essa funcionalidade.

app/http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{
   public function index()
   {

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

      $title = 'Lista de usuários';

      return view('users', compact('title', 'users'));

   }

   public function show($id)
   {
      return "Exibindo detalhes do usuário: {$id}";
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```

resources/views/users.blade.php

```
<!doctype html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Lista de usuários - Styde.net</title>
</head>
<body>
<h1> {{ $title }} </h1>

<hr>

<ul>
   @forelse ($users as $user)
      <li>{{ $user->name }}</li>
   @empty
      <li>Não há usuários registrados.</li>
   @endforelse
</ul>

</body>
</html>
```

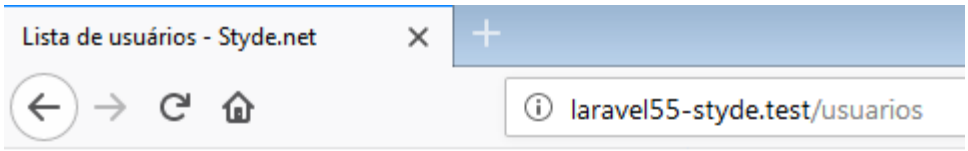


Lista de usuários

- Duilio Palacios
- Felipa Effertz II
- Suzanne Windler I
- Shanna Rogahn
- Mr. Jocelyn Beer
- Mrs. Ally Luettgen
- Paula Kunde
- Juston Jacobi
- Jesus Mante
- Zachariah McCullough
- Jayson Effertz Sr.
- Hillary Schulist
- Bonnie Buckridge
- Mr. Thad Marquardt PhD
- Rosella Bradtke
- Madalyn Quigley
- Hermina Jones
- Savanna Nader
- Chelsie Green
- Oren Harvey
- Phoebe Blanda
- Prof. Akeem Pouros DVM
- Eryn Bins
- Lazaro Reilly
- Iva Kub
- Prof. Jeramie Ratke IV
- Leonie Rath
- Ms. Anais Bayer
- Destini Homenick
- Shany Will

app/http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{
   public function index()
   {

      $users = User::all();

      $title = 'Lista de usuários';

      return view('users', compact('title', 'users'));

   }

   public function show($id)
   {
      return "Exibindo detalhes do usuário: {$id}";
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```



Lista de usuários

- Duilio Palacios
- Felipa Effertz II
- Suzanne Windler I
- Shanna Rogahn
- Mr. Jocelyn Beer
- Mrs. Ally Luettgen
- Paula Kunde
- Juston Jacobi
- Jesus Mante
- Zachariah McCullough
- Jayson Effertz Sr.
- Hillary Schulist
- Bonnie Buckridge
- Mr. Thad Marquardt PhD
- Rosella Bradtke
- Madalyn Quigley
- Hermina Jones
- Savanna Nader
- Chelsie Green
- Oren Harvey
- Phoebe Blanda
- Prof. Akeem Pouros DVM
- Eryn Bins
- Lazaro Reilly
- Iva Kub
- Prof. Jeramie Ratke IV
- Leonie Rath
- Ms. Anais Bayer
- Destini Homenick
- Shany Will

Aula 20 - Base de dados com Laravel e PHPUnit

Nesta nova lição do Curso Laravel a partir do zero, aprenderemos a configurar e executar operações de banco de dados no ambiente de teste automatizado (PHPUnit). Para isso, seguiremos uma série de etapas: criaremos um banco de dados adicional para o ambiente de teste automatizado, veremos como executar automaticamente as migrações de banco de dados de nossos testes e como executar os testes em um ambiente isolado para obter os resultados esperados.

Bases de dados para provas automatizadas

Certifique-se de estar usando o mecanismo InnoDB no banco de dados MySQL para que os testes possam ser executados nas transações do banco de dados.

Podemos ter dois bancos de dados, um para interagir com nosso aplicativo no navegador e outro para os testes que vamos executar. Desta forma, a execução dos testes automatizados não afetará nossa interação com o aplicativo localmente e vice-versa.

Configurar o banco de dados para testes automatizados

Definir as variáveis de ambiente no phpunit No arquivo phpunit.xml, vamos sobrescrever a variável que o Laravel usa para ler o nome do banco de dados por padrão. Fazemos isso adicionando uma variável de ambiente. No nome da propriedade, indicamos o nome da variável de ambiente (DB_DATABASE) e no valor da propriedade, indicamos o nome do banco de dados que queremos usar para os testes (no nosso exemplo, é curso_styde_tests):

```
<env name="DB_DATABASE" value="curso_styde_tests"/>
```

Crie o novo banco de dados

Claro que você precisará criar esse novo banco de dados (por exemplo, curso_styde_tests). Para fazer isso, você pode fazê-lo no console ou com o administrador de banco de dados de sua escolha (PHPMyAdmin, Sequel Pro, entre outros).

Use o trait RefreshDatabase

Inclua o trait RefreshDatabase em cada um dos testes que irão interagir com o banco de dados. Desta forma, o Laravel executará as migrations do banco de dados antes de executar os testes. Além disso, ele executará cada teste dentro de uma transação de banco de dados que será revertida após a execução de cada método de teste. Dessa forma, evitamos ter que migrar manualmente o banco de dados e nos preocupar com dados que possam "contaminar" o status de cada um dos nossos testes:

```
class UsersModuleTest extends TestCase
{
   use RefreshDatabase;

   // ...
```

Para usá-lo, devemos importar o trait no início do arquivo:

```
use Illuminate\Foundation\Testing\RefreshDatabase;
```

Usando RefreshDatabase os testes são executados em transações de banco de dados, ou seja, cada vez que um teste é executado, o banco de dados retorna ao seu estado original no final do teste. Devido a isso, no nosso exemplo ambos os testes irão acontecer, embora em it_shows_users_list() nós estamos criando dois usuários e então no próximo teste nós checamos que não há usuários no banco de dados:

```
/** @test */
function it_shows_the_users_list()
{
   $this->get('/usuarios')
      ->assertStatus(200)
      ->assertSee('Lista de usuários')
      ->assertSee('Duilio Palacios');
}

/** @test */
function it_shows_a_default_message_if_the_users_list_is_empty()
{
   $this->get('/usuarios')
      ->assertStatus(200)
      ->assertSee('Não há usuários registrados.');
}
```

Crie o banco de dados curso_styde_tests

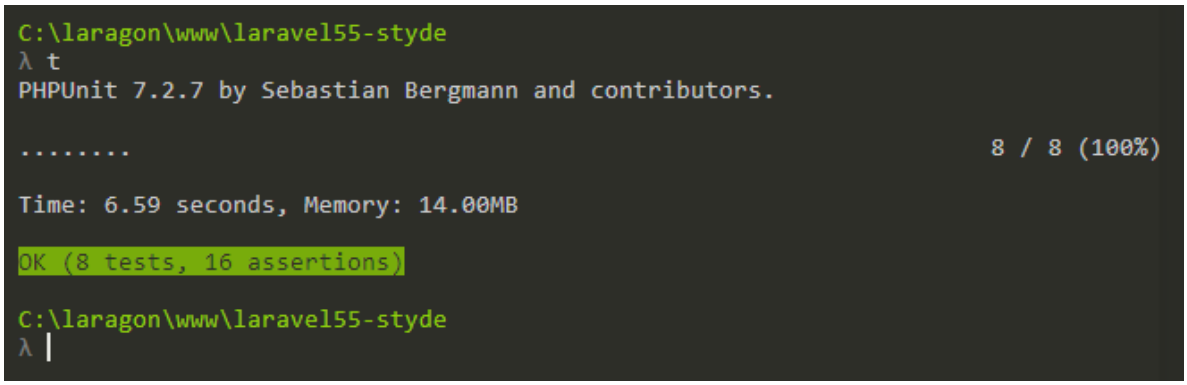
```
C:\laragon\www\laravel55-styde
λ mysql -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 605
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database curso_styde_tests;
Query OK, 1 row affected (0.19 sec)

mysql>
```

phpunit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit backupGlobals="false"
         backupStaticAttributes="false"
         bootstrap="vendor/autoload.php"
         colors="true"
         convertErrorsToExceptions="true"
         convertNoticesToExceptions="true"
         convertWarningsToExceptions="true"
         processIsolation="false"
         stopOnFailure="false">
    <testsuites>
        <testsuite name="Feature">
            <directory suffix="Test.php">./tests/Feature</directory>
        </testsuite>

        <testsuite name="Unit">
            <directory suffix="Test.php">./tests/Unit</directory>
        </testsuite>
    </testsuites>
    <filter>
        <whitelist processUncoveredFilesFromWhitelist="true">
            <directory suffix=".php">./app</directory>
        </whitelist>
    </filter>
    <php>
        <env name="APP_ENV" value="testing"/>
        <env name="BCRYPT_ROUNDS" value="4"/>
        <env name="CACHE_DRIVER" value="array"/>
        <env name="SESSION_DRIVER" value="array"/>
        <env name="QUEUE_DRIVER" value="sync"/>
        <env name="MAIL_DRIVER" value="array"/>
        <env name="DB_DATABASE" value="curso_styde_tests"/>
    </php>
</phpunit>
```

tests/Feature/UserModuleTest.php

```
<?php

namespace Tests\Feature;
use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{
   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {
      factory(User::class)->create([
         'name' => 'Joel'
      ]);
      factory(User::class)->create([
         'name' => 'Ellie',
      ]);
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_loads_the_users_details_page()
   {
      $this->get('/usuarios/5')
         ->assertStatus(200)
         ->assertSee('Exibindo detalhes do usuário: 5');
   }

   /** @test */
   function it_loads_the_new_users_page()
   {
      $this->get('/usuarios/novo')
         ->assertStatus(200)
         ->assertSee('Criar novo usuário');
   }
}
```


C:\laragon\www\laravel55-styde
λ t
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.
.... 8 / 8 (100%)
Time: 6.59 seconds, Memory: 14.00MB
OK (8 tests, 16 assertions)
C:\laragon\www\laravel55-styde
λ

Aula 21 - Detalhes ou perfil básico de um usuário com Laravel e TDD

Nesta lição, faremos o processo inverso à lição anterior, em vez de escrever o código e, em seguida, o teste, primeiro escreveremos o teste e depois o código para o teste passar, que é o que o "Desenvolvimento Orientado" refere. por testes automatizados ou TDD. Isso nos permitirá sermos capazes de nos guiar enquanto construímos as diferentes partes de nosso aplicativo, que no caso desta lição é o perfil do usuário.

Método Find() do Eloquent

Usando o método find(), podemos retornar um registro específico do banco de dados. Neste caso, find () retornará o usuário com um id igual a 1:

```
$user = User::find(1);
```

O método find() também aceita uma matriz de chaves primárias. Passando este array, find() retornará uma coleção com os registros encontrados:

```
$users = User::find([1, 2, 3]);
```

---

app/http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

//       return view('users.index')
//          ->with('users', User::all())
//          ->with('title', 'Lista de usuários');

      return view('users.index', compact('title', 'users'));
   }

   public function show($id)
   {

      $user = User::find($id);
      return view('users.show', compact('user'));

   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```

resources/views/users/index.blade.php

```
<!doctype html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Lista de usuários - Styde.net</title>
</head>
<body>
<h1> {{ $title }} </h1>

<hr>

<ul>
   @forelse ($users as $user)
      <li>{{ $user->name }}</li>
   @empty
      <li>Não há usuários registrados.</li>
   @endforelse
</ul>

</body>
</html>
```

resources/views/users/show.blade.php

```
@extends('layout')

@section('title', "Usuário {$user->id}")

@section('content')
   <h1>Usuário #{{ $user->id }}</h1>

   <p>Nome do usuário: {{ $user->name }}</p>
   <p>Email: {{ $user->email }}</p>
@endsection
```

resources/views/layout.blade.php

```
<!doctype html>
<html lang="pt-br">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <link rel="icon" href="favicon.ico">

  <title>@yield('title') - Styde.net</title>

  <!-- Bootstrap core CSS -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
beta.2/css/bootstrap.min.css" integrity="sha384-
PsH8R72JQ3SOdhVi3uxftmaW6Vc51MKb0q5P2rRUpPvrszuE4W1povHYgTpBfshb" crossorigin="anonymous">
  <!-- Custom styles for this template -->
  <link href="{{ asset('css/style.css') }}" rel="stylesheet">
</head>

<body>

<header>
  <!-- Fixed navbar -->
  <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
    <a class="navbar-brand" href="#">Curso de Laravel</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarCollapse"
aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarCollapse">
      <ul class="navbar-nav mr-auto">
        <li class="nav-item active">
          <a class="nav-link" href="{{ url('/usuarios') }}">Usuários</a>
        </li>
      </ul>
    </div>
  </nav>
</header>

<!-- Begin page content -->
<main role="main" class="container">
  <div class="row mt-3">
    <div class="col-8">
      @yield('content')
    </div>
    <div class="col-4">
      <p> </p>
    </div>
  </div>
</main>

<footer class="footer">
  <div class="container">
    <span class="text-muted">https://styde.net</span>
```

```
   </div>
</footer>


<!-- Bootstrap core JavaScript
================================================== -->
<!-- Placed at the end of the document so the pages load faster -->
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.3/umd/popper.min.js" integrity="sha384-
vFJXuSJphROIrBnz7yo7oB41mKfc8JzQZiCq4NCceLEaO4IHwicKwpJf9c9IpFgh"
crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js"
integrity="sha384-alpBpkh1PFOepccYVYDB4do5UnbKysX5WZXm3XxPqe5iKTfUKjNkCk9SaVuEZflJ"
crossorigin="anonymous"></script>
</body>
</html>
```

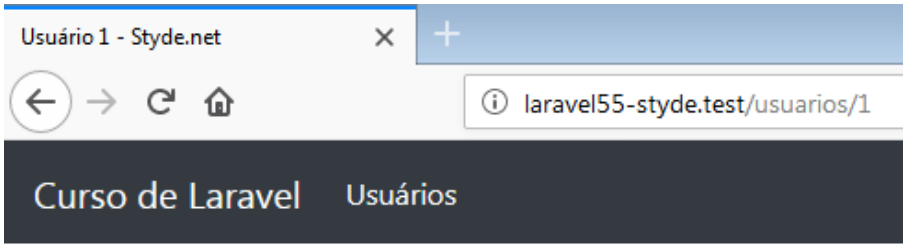
public/css/style.css

```
/* Sticky footer styles
-------------------------------------------------- */
html {
  position: relative;
  min-height: 100%;
}
body {
  /* Margin bottom by footer height */
  margin-bottom: 60px;
}
.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  /* Set the fixed height of the footer here */
  height: 60px;
  line-height: 60px; /* Vertically center the text there */
  background-color: #f5f5f5;
}


/* Custom page CSS
-------------------------------------------------- */
/* Not required for template or sticky footer method. */

body > .container {
  padding: 60px 15px 0;
}

.footer > .container {
  padding-right: 15px;
  padding-left: 15px;
}

code {
  font-size: 80%;
}
```

tests/Feature/UserModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);
      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }
```

```
    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar novo usuário');
    }
}
```



Usuário #1

Nome do usuário: Duilio Palacios

Email: duilio@styde.net

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

........                                                            8 / 8 (100%)

Time: 6.29 seconds, Memory: 14.00MB

OK (8 tests, 16 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 22 - Gerar URL's no Laravel

Uma das vantagens de uma página web é a possibilidade de navegar através de links. Nesta lição aprenderemos como criar links no Laravel usando os diferentes helpers que temos disponíveis, como url(), action() e route().

Criando links com o helper url()

O helper url() nos permite criar um link em nossa view:

```
<a href="{{ url('/usuarios/'.$user->id) }}">Ver detalhes</a>
```

Também podemos usar aspas duplas. Desta forma, em vez de concatenar, podemos colocar diretamente a propriedade $user->id:

```
<a href="{{ url("/usuarios/{$user->id}") }}">Ver detalhes</a>
```

Ao chamar o helper url() sem passar nenhum argumento, o Laravel retornará uma instância do objeto Illuminate\Routing\ UrlGenerator, com o qual podemos acessar diferentes métodos. Por exemplo, o método previous() que retorna o URL anterior:

```
<a href="{{ url()->previous() }}">Voltar</a>
```

Outros métodos disponíveis são current() e full() com os quais podemos acessar o URL atual e o URL atual com a string de consulta (Query String):

```
url()->current(); // URL completa

url()->full(); // URL completa com a cadeia de consulta
```

Também podemos utilizar o alias e facade URL para acessar a estes métodos:

```
URL::current();
```

Dentro de uma classe com um namespace, você precisaria importar o Facade: Illuminate\Support\Facades\URL

Criando links com o helper action()

Com o helper action() podemos também criar URLs, passando como argumento o nome do controller que queremos vincular seguido por uma arroba e o nome da action dentro do controller:

```
<a href="{{action('UserController@index')}}">Voltar à lista de usuários</a>
```

A rota, por padrão, é relativa ao namespace.

App\Http\Controllers.

Também podemos usar o helper action() quando precisamos passar um argumento para o URL. Isto é conseguido passando como um segundo argumento para o helper um array associativo com os parâmetros que precisamos passar para a URL:

```
<a href="{{ action('UserController@show', ['id' => $user->id]) }}">Ver detalhes</a>
```

Rotas com nome

Podemos atribuir um nome a uma rota, encadeando o método name() para a declaração da rota. Neste caso, a primeira rota terá o nome dos usuários e a segunda usuários.show

```
Route::get('/usuarios', 'UserController@index')->name('users.index');

Route::get('/usuarios/detalles/{id}', 'UserController@details')
   ->where('id', '[0-9]+')
   ->name('users.show');
```

Ao fazer isso, agora dentro das views podemos nos referir ao nome das rotas usando o helper route(), passando como primeiro argumento o nome da rota e como segundo argumento os parâmetros da rota, como com a ação da função:

```
<a href="{{ route('users.show', ['id' => $user->id]) }}">Ver detalhes</a>
```

Se você está passando como argumento uma chave primária de um modelo Eloquent, você pode passar o modelo diretamente para route(). O helper extrairá automaticamente a chave primária:

```
<a href="{{ route('users.show', ['id' => $user]) }}">Ver detalhes</a>
```

---

routes/web.php

```php
<?php

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

Route::get('/usuarios', 'UserController@index')
    ->name('users.index');

Route::get('/usuarios/{id}', 'UserController@show')
    ->where('id', '[0-9]+')
    ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

resources/views/users/index.blade.php

```
@extends('layout')

@section('title', 'Usuarios')

@section('content')
    <h1>{{ $title }}</h1>

    <ul>
        @forelse ($users as $user)
            <li>
                {{ $user->name }}, ({{ $user->email }})
                <a href="{{ route('users.show', ['id' => $user->id]) }}">Ver detalhes</a>
            </li>
        @empty
            <li>Não há usuários registrados.</li>
        @endforelse
    </ul>
@endsection

@section('sidebar')
    @parent
@endsection
```

resources/views/users/show.blade.php

```
@extends('layout')

@section('title', "Usuário {$user->id}")

@section('content')
  <h1>Usuário #{{ $user->id }}</h1>

  <p>Nome do usuário: {{ $user->name }}</p>
  <p>Email: {{ $user->email }}</p>

  <p>
    <a href="{{ route('users.index') }}">Voltar a lista de usuários</a>
  </p>
@endsection
```

```
C:\laragon\www\laravel55-styde
λ t
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

........                                                            8 / 8 (100%)

Time: 7.04 seconds, Memory: 14.00MB

OK (8 tests, 16 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 23 - Manejo de erros 404 no Laravel

Muitas vezes, quando realizamos uma consulta usando SQL ou API, o resultado esperado pode não ser obtido porque o conteúdo que o usuário tenta ver não existe. Nós, como desenvolvedores, devemos levar em conta situações como essa em nosso aplicativo, é por isso que nesta lição veremos como podemos retornar erros 404 manualmente e também como fazê-lo automaticamente quando um modelo não é encontrado.

Retornar uma visão com status 404

Quando retornamos a chamada para o helper view() de uma ação, o Laravel retornará o conteúdo da view com o status HTTP 200 (OK). Podemos retornar uma visão com o status 404 (não encontrado) usando o helper response e então encadeando a chamada para o método view. Para o método view passamos como primeiro argumento o nome da view, como segundo argumento os dados e como terceiro argumento o status HTTP:

```
return response()->view('errors.404', [], 404);
```

Uso de findOrFail

O método findOrFail tentará encontrar o registro correspondente à chave primária passada como argumento e, se não for encontrado, retornará uma exceção do tipo ModelNotFoundException:

```
$user = User::findOrFail($id);
```

Também podemos usar o método FirstOrFail, que retorna o primeiro resultado da consulta e, se nenhum registro for encontrado, retorna um ModelNotFoundException:

```
$user = User::where('posts', '>', 100)->firstOrFail();
```

---

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

//       return view('users.index')
//          ->with('users', User::all())
//          ->with('title', 'Lista de usuários');

      return view('users.index', compact('title', 'users'));
   }

   public function show($id)
   {
      $user = User::findOrFail($id);
      return view('users.show', compact('user'));
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```

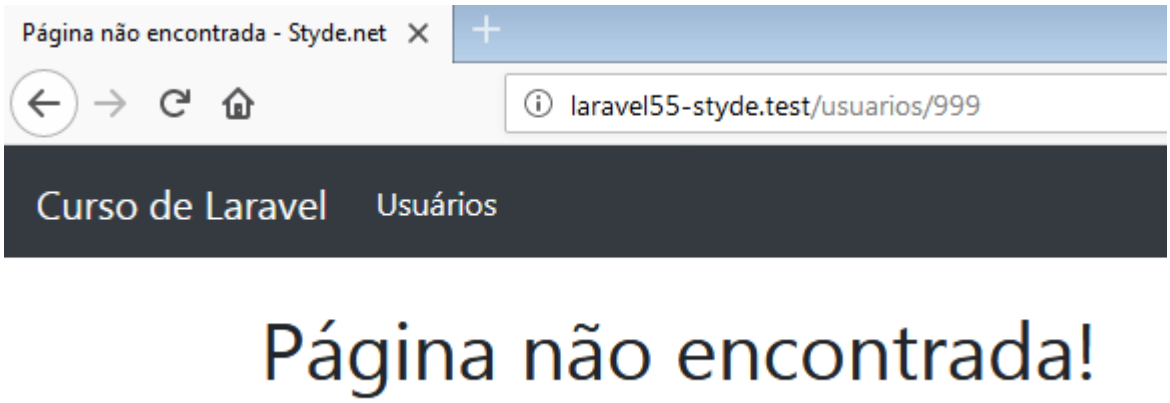
resources/views/errors/404.blade.php

```
@extends('layout')

@section('title', "Página não encontrada")

@section('content')
   <h1>Página não encontrada!</h1>
@endsection
```

tests/Feature/UserModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }

   /** @test */
   function it_displays_a_404_error_if_the_user_is_not_found()
```

```
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar novo usuário');
    }
}
```




```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.........                                                   9 / 9 (100%)

Time: 8.36 seconds, Memory: 14.00MB

OK (9 tests, 18 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 24 - Route Model Binding

O Laravel nos permite obter modelos diretamente nos parâmetros de nossas ações, sem precisar da chamada explícita para métodos Eloquent como find ou findOrFail, nesta lição veremos o uso desta característica conhecida como Route Model Binding:

Em vez de obter o usuário usando os métodos find ou findOrFail, podemos obtê-lo diretamente como parâmetro da ação:

```
public function show(User $user)
{
   return view('users.show', compact('user'));
}
```

Para que isso funcione, observe que o nome do parâmetro na declaração de rota deve corresponder ao nome do parâmetro na declaração do método:

```
Route::get('/usuarios/{user}', 'UserController@show');

// e no controller

use App\User;

class UserController {

   public function show(User $user)
   {
      //...
   }

}
```

Além disso, o tipo do parâmetro deve ser, obviamente, um modelo Eloquente (no nosso exemplo é App\User).

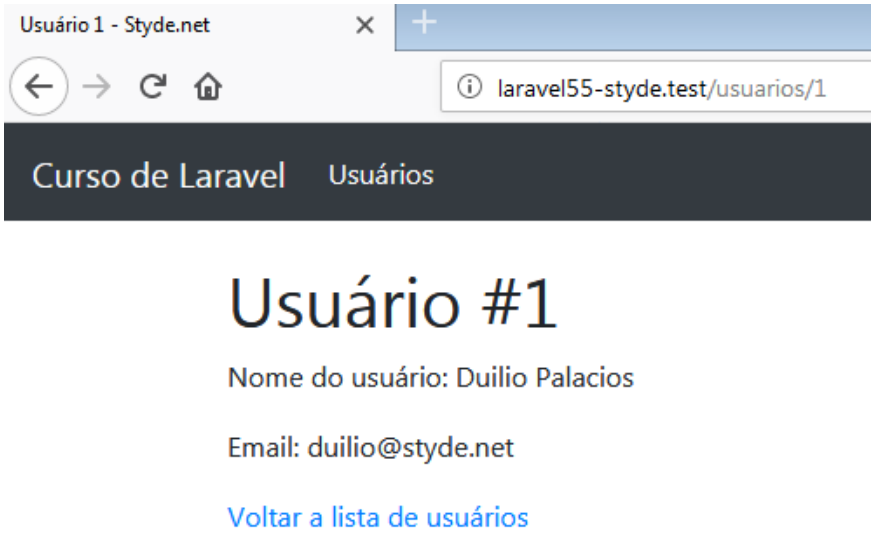
---

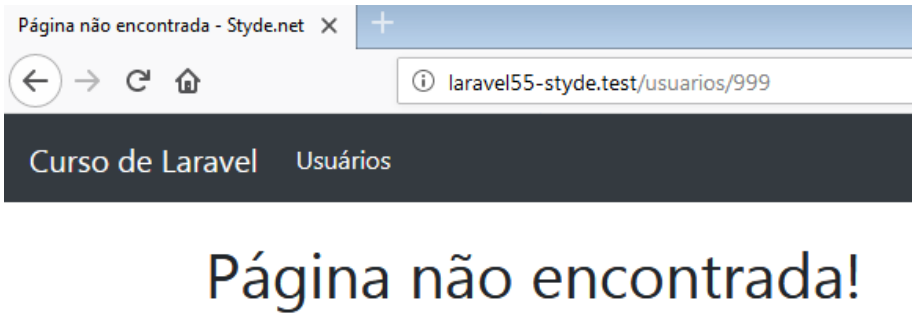
app/Http/Controllers/UserController.php

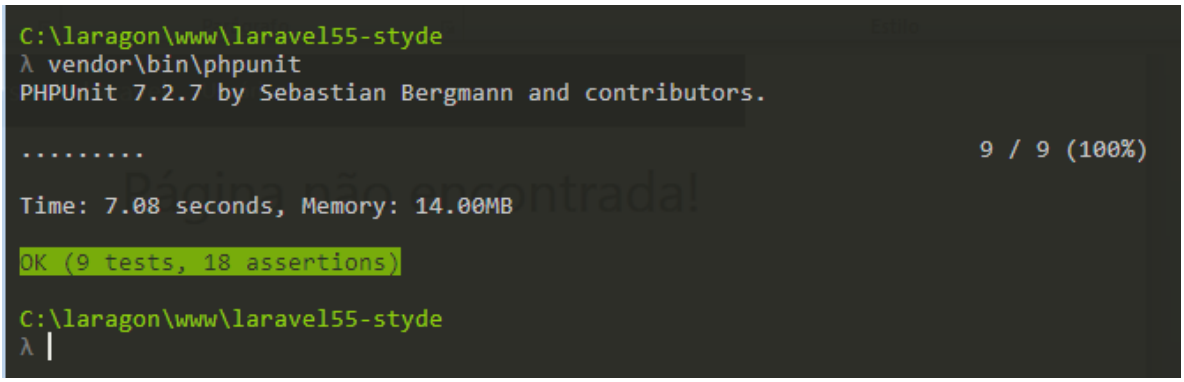
```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

//       return view('users.index')
//          ->with('users', User::all())
//          ->with('title', 'Lista de usuários');

      return view('users.index', compact('title', 'users'));
   }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }

   public function create()
   {
      return 'Criar novo usuário';
   }
}
```

routes/web.php

```
<?php

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

Route::get('/usuarios', 'UserController@index')
   ->name('users.index');

Route::get('/usuarios/{user}', 'UserController@show')
   ->where('user', '[0-9]+')
   ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```






C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.
....
9 / 9 (100%)
Time: 7.08 seconds, Memory: 14.00MB
OK (9 tests, 18 assertions)
C:\laragon\www\laravel55-styde
λ

Aula 25 - Rotas com POST e proteção contra ataques do tipo CSRF em Laravel

Nesta lição, aprenderemos a criar rotas do tipo POST, que se referirão a ações que normalmente alteram o estado de nosso aplicativo (por exemplo, para a criação de registros). Também veremos como evitar ataques de CSRF usando uma proteção que já está incluída na estrutura do Laravel.

Rotas com o método POST

Nós declaramos uma rota do tipo POST usando o facade Route e chamando o método post():

```
Route::post('/usuarios/create', 'UserController@store');
```

Podemos ter duas rotas que usam o mesmo URL, mas com métodos diferentes:

```
Route::get('/usuarios', 'UserController@index');

Route::post('/usuarios', 'UserController@create');
```

Para ver todas as rotas que você tem em seu aplicativo, você pode executar o comando route:list do Artisan no console:

```
php artisan route:list
```

Token (CSRF)

O middleware VerifyCsrfToken nos permite impedir que terceiros enviem solicitações POST para nosso aplicativo e executem ataques de falsificação de solicitações entre sites. Para adicionar um campo com o token dentro de nosso formulário, o que permitirá que o Laravel reconheça solicitações de formulário válidas, devemos chamar o método csrf_field():

```
<form method="POST" action="{{ url('usuarios/create') }}">
   {{ csrf_field() }}

   <button type="submit">Criar usuario</button>
</form>
```

O método csrf_field() adicionará um campo oculto chamado _token com o valor do token ao código HTML de nosso formulário. Podemos desativar a proteção CSRF comentando na linha 36 do arquivo Kernel.php, embora isso seja altamente desaconselhável:

```
protected $middlewareGroups = [
    'web' => [
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        // \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ]

    // ...
]
```

get X post

get: solicitar e obter informação
post: enviar e processar informação

---

routes/web.php

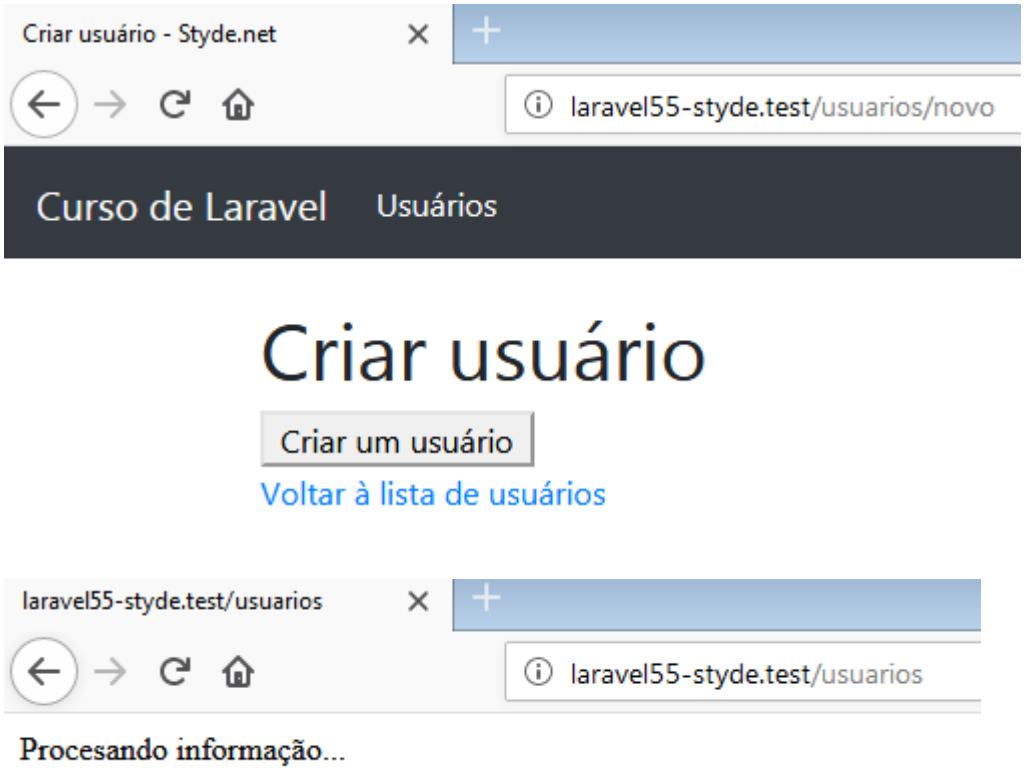
```
<?php

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

Route::get('/usuarios', 'UserController@index')
    ->name('users.index');

Route::get('/usuarios/{user}', 'UserController@show')
    ->where('user', '[0-9]+')
    ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::post('/usuarios', 'UserController@store');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

//       return view('users.index')
//          ->with('users', User::all())
//          ->with('title', 'Lista de usuários');

      return view('users.index', compact('title', 'users'));
   }

//    public function show($id)
//    {
//       $user = User::findOrFail($id);
//       return view('users.show', compact('user'));
//    }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }


   public function create()
   {
      return view('users.create');
   }

   public function store()
   {
      return 'Procesando informação...';
   }

}
```

resources/views/users/create.blade.php

```
@extends('layout')

@section('title', "Criar usuário")

@section('content')
  <h1>Criar usuário</h1>

  <form method="POST" action="{{ url('usuarios') }}">
    {{ csrf_field() }}

    <button type="submit">Criar um usuário</button>
  </form>

  <p>
    <a href="{{ route('users.index') }}">Voltar à lista de usuários</a>
  </p>
@endsection
```



tests/Feature/UserModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);
```

```
        $this->get('/usuarios/'.$user->id) // usuarios/5
        ->assertStatus(200)
            ->assertSee('Duilio Palacios');
    }

    /** @test */
    function it_displays_a_404_error_if_the_user_is_not_found()
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

}
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

..........                                                        10 / 10 (100%)

Time: 7.22 seconds, Memory: 16.00MB

OK (10 tests, 21 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 26 - Criação de usuários com Laravel e TDD

Verificar manualmente se nossos formulários funcionam corretamente pode ser complicado, ainda mais se nosso aplicativo crescer e acabarmos tendo formulários diferentes com vários campos em cada um. O que acontece se estivermos trabalhando em uma API e nosso aplicativo Laravel não tiver nenhum formulário? Embora existam aplicativos e plug-ins como o Postman, que nos permitem testar as solicitações POST, nesta lição veremos como podemos verificar a operação das rotas POST usando testes automatizados.

Usando $this->post(), podemos simular solicitações POST de um teste automatizado. Neste caso, adicionamos um novo teste para verificar se podemos acessar a rota para criar usuários e que também podemos criar novos usuários:

```
function it_creates_a_new_user()
{
   $this->post('/usuarios/', [
      'name' => 'Duilio',
      'email' => 'duilio@styde.net',
      'password' => '123456'
   ])->assertRedirect('usuarios');
}
```

Como primeiro argumento, passamos o URL (neste caso /usuarios) e como segundo argumento os dados do pedido. Com assertRedirect(), verificamos se o usuário é redirecionado para a URL fornecida.

Para o método assertRedirect() podemos passar como argumento o helper route para usar o nome de uma rota em vez de uma URL:

```
assertRedirect(route('users.index'));
```

Para verificar se um usuário foi criado com os dados que passamos na solicitação POST, usaremos o método assertDatabaseHas. Como primeiro argumento, passamos o nome da tabela e, como segundo argumento, os dados que esperamos encontrar:

```
$this->assertDatabaseHas('users', [
   'name' => 'Duilio',
   'email' => 'duilio@styde.net'
]);
```

Neste caso, esperamos encontrar dentro da tabela users, um registro com um campo de nome igual a Duilio e um campo de e-mail igual a duilio@styde.net. Também podemos usar o método assertCredentials(), que nos permite verificar a presença da senha correta. Como primeiro argumento, passamos uma matriz com os dados que esperamos encontrar. Esse método não precisa passar o nome da tabela, desde que estejamos usando a tabela de usuários padrão (users):

```
$this->assertCredentials([
   'name' => 'Duilio',
   'email' => 'duilio@styde.net',
   'password' => '123456'
]);
```

Para receber os dados da solicitação dentro do controlador, podemos usar o método request():

```
$data = request()->all();

// Depois disso, podemos acessar os dados:
$data['name'];
```

Usando redirect(), podemos redirecionar para outra parte do aplicativo a partir do controller:

```
return redirect('usuarios'); // Redirecionamos a URL "/users"

return redirect()->route('users.index'); // Redirecionamos à rota com o nome "users.index"
```

---

tests/Feature/UserModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }
```

```
    /** @test */
    function it_displays_a_404_error_if_the_user_is_not_found()
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

    /** @test */
    function it_creates_a_new_user()
    {
        $this->withoutExceptionHandling();
        $this->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect('usuarios');
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

}
```

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

    public function index()
    {
        //$users = DB::table('users')->get();
        $users = User::all();
        $title = 'Lista de usuários';

        return view('users.index', compact('title', 'users'));
    }

    public function show(User $user)
    {
        return view('users.show', compact('user'));
    }


    public function create()
    {
        return view('users.create');
    }

    public function store()
    {
        $data = request()->all();
        User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password'])
        ]);
        return redirect()->route('users.index');
    }

}
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit --filter it_creates_a_new_user
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.                                                                   1 / 1 (100%)

Time: 6.6 seconds, Memory: 14.00MB

OK (1 test, 3 assertions)

C:\laragon\www\laravel55-styde
λ
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

..........                                                         10 / 10 (100%)

Time: 7.88 seconds, Memory: 16.00MB

OK (10 tests, 21 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 27 - Criação de formulário para adicionar usuários com Laravel

Nesta lição 27 do Curso Laravel 5.5 do zero, criaremos o formulário para o cadastro de novos usuários no aplicativo, utilizando como base o teste e o código que escrevemos na lição anterior.

Lembre-se de adicionar a chamada à função csrf_field() dentro de seu formulário para gerar o campo oculto que contém o token para passar a proteção contra ataques do tipo CSRF que o Laravel fornece por padrão. Para gerar o token, você deve chamar o helper dentro do formulário:

```
<form action="" method="POST">
   {{ csrf_field() }}
   ...
</form>
```

É importante que os valores dos atributos name dos campos no formulário HTML correspondam aos atributos que você está obtendo do objeto Request no driver do Laravel:

```
// No formulário:
<input type="email" name="email">

// Dentro do controller:
$data = request()->all();

User::create([
   'email' => $data['email'], // Coincide com o atributo name
]);
```

Obter dados do objeto Request

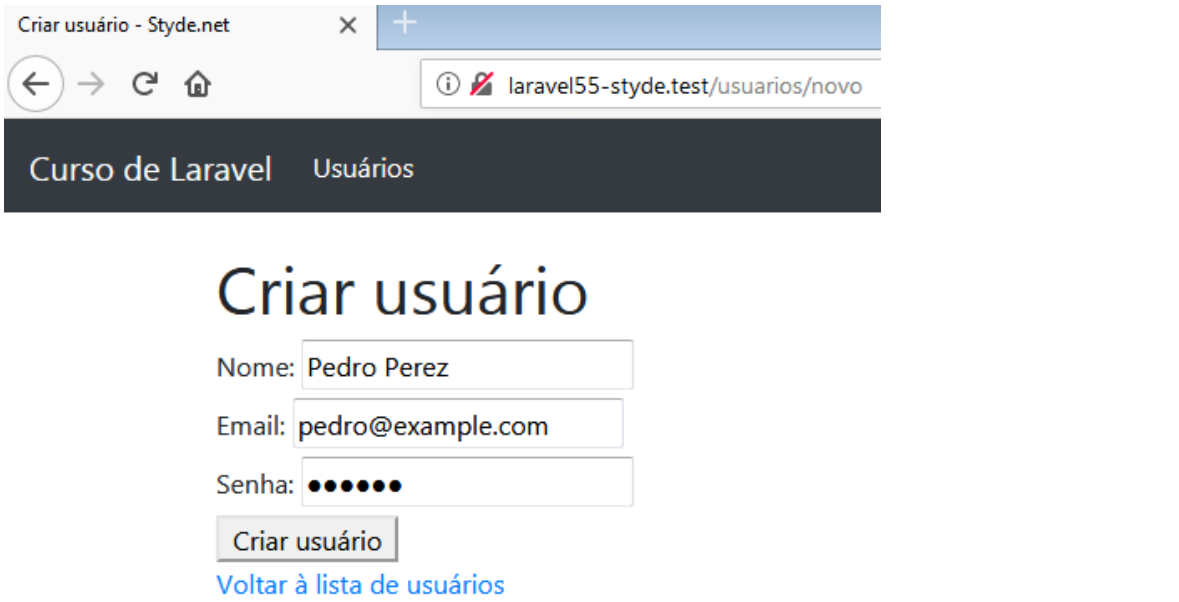
O Laravel oferece várias maneiras de obter os dados dos formulários com o objeto Request, no exemplo abaixo, eu mostro 3 deles, todos usando o helper request():

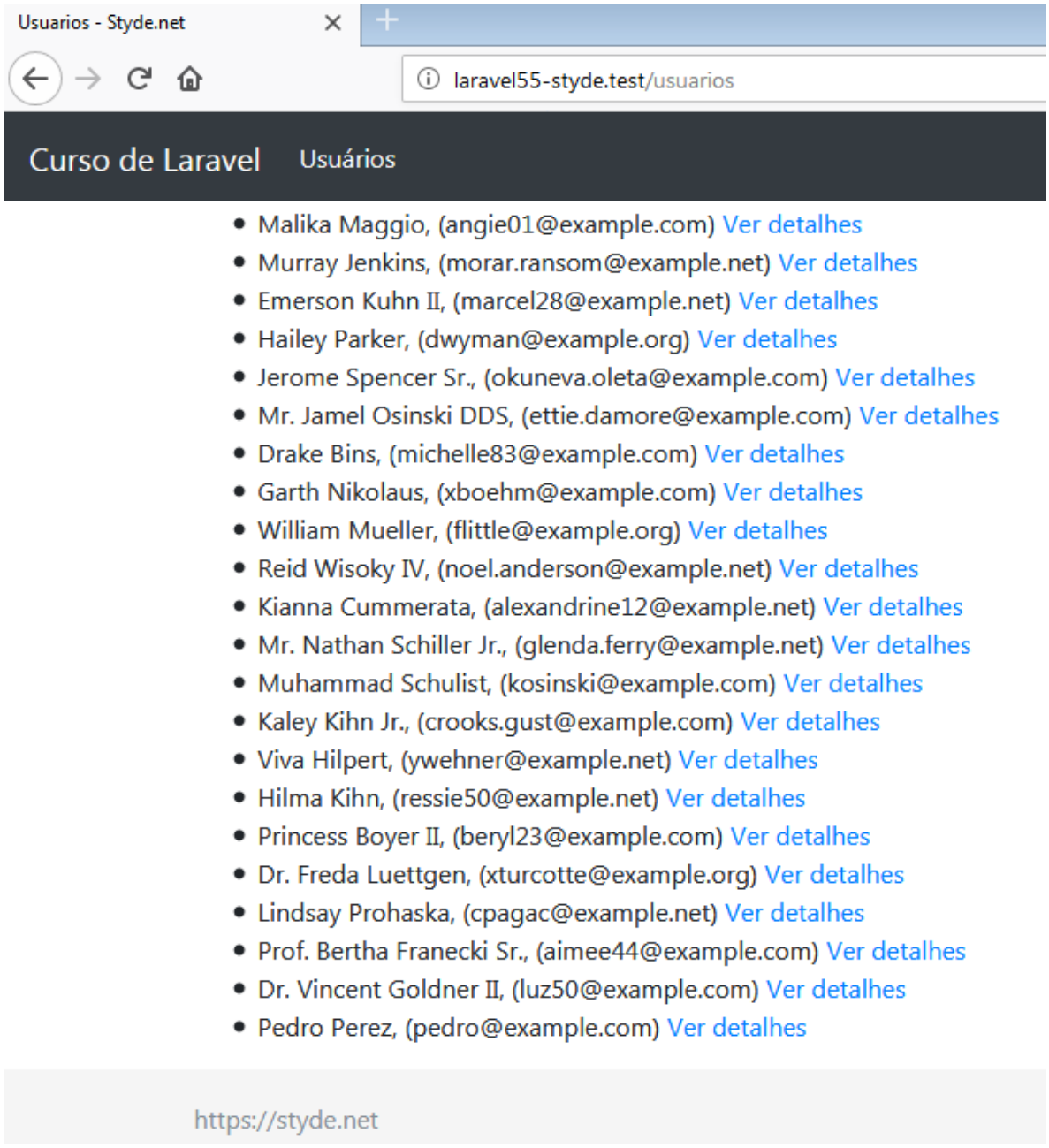
```
User::create([
   'name' => request('name'),
   'email' => request()->email,
   'password' => bcrypt(request()->get('password'))
]);
```

---

resources/views/users/create.blade.php

```
@extends('layout')

@section('title', "Criar usuário")

@section('content')
  <h1>Criar usuário</h1>

  <form method="POST" action="{{ url('usuarios') }}">
    {{ csrf_field() }}

    <label for="name">Nome:</label>
    <input type="text" name="name" id="name" placeholder="Pedro Perez">
    <br>
    <label for="email">Email:</label>
    <input type="email" name="email" id="email" placeholder="pedro@example.com">
    <br>
    <label for="password">Senha:</label>
    <input type="password" name="password" id="password" placeholder="Maior que 6 caracteres">
    <br>
    <button type="submit">Criar usuário</button>
  </form>

  <p>
    <a href="{{ route('users.index') }}">Voltar à lista de usuários</a>
  </p>
@endsection
```




Usuarios - Styde.net
laravel55-styde.test/usuarios
Curso de Laravel
Usuários
Malika Maggio, (angie01@example.com) Ver detalhes
Murray Jenkins, (morar.ransom@example.net) Ver detalhes
Emerson Kuhn II, (marcel28@example.net) Ver detalhes
Hailey Parker, (dwyman@example.org) Ver detalhes
Jerome Spencer Sr., (okuneva.oleta@example.com) Ver detalhes
Mr. Jamel Osinski DDS, (ettie.damore@example.com) Ver detalhes
Drake Bins, (michelle83@example.com) Ver detalhes
Garth Nikolaus, (xboehm@example.com) Ver detalhes
William Mueller, (flittle@example.org) Ver detalhes
Reid Wisoky IV, (noel.anderson@example.net) Ver detalhes
Kianna Cummerata, (alexandrine12@example.net) Ver detalhes
Mr. Nathan Schiller Jr., (glenda.ferry@example.net) Ver detalhes
Muhammad Schulist, (kosinski@example.com) Ver detalhes
Kaley Kihn Jr., (crooks.gust@example.com) Ver detalhes
Viva Hilpert, (ywehner@example.net) Ver detalhes
Hilma Kihn, (ressie50@example.net) Ver detalhes
Princess Boyer II, (beryl23@example.com) Ver detalhes
Dr. Freda Luettgen, (xturcotte@example.org) Ver detalhes
Lindsay Prohaska, (cpagac@example.net) Ver detalhes
Prof. Bertha Franecki Sr., (aimee44@example.com) Ver detalhes
Dr. Vincent Goldner II, (luz50@example.com) Ver detalhes
Pedro Perez, (pedro@example.com) Ver detalhes
https://styde.net

Aula 28 - Validar dados de petições http com Laravel e TDD

Nesta lição do Curso Laravel 5.5 do zero, aprenderemos como validar dados de uma solicitação HTTP (por exemplo, dados enviados por meio de um formulário), usando o componente de validação que inclui a estrutura do Laravel, e orientaremos o desenvolvimento por meio do uso de testes automatizados.

Dentro do teste, verificamos que a sessão contém um erro para o campo de nome usando assertSessionHasErrors:

```
$this->from('usuarios/novo')
   ->post('/usuarios/', [
      'name' => '',
      'email' => 'duilio@styde.net',
      'password' => '123456'
   ])
   ->assertRedirect('usuarios/novo')
   ->assertSessionHasErrors(['name']);
```

O método assertSessionHasErrors espera que haja um campo name na lista de erros da sessão, independentemente do conteúdo ou da mensagem desse erro. No entanto, podemos indicar que esperamos que o conteúdo ou a mensagem esteja passando uma matriz de pares de valores-chave em vez de apenas o nome do campo:

```
$this->post('...')->assertSessionHasErrors(['name' => 'O campo nome é obrigatório'])
```

Com assertDatabaseMissing, verificamos que o registro não está sendo salvo no banco de dados, no final da requisição executada durante o método de teste:

```
$this->assertDatabaseMissing('users', [
   'email' => 'duilio@styde.net'
]);
```

Também podemos usar o método assertEquals em vez de assertDatabaseMissing para verificar se o usuário não está sendo salvo no banco de dados. Para este método nós passamos o valor esperado como primeiro argumento e como segundo argumento passamos o número de usuários no banco de dados, usando o método count fornecido pelo Eloquent:

```
$this->assertEquals(0, User::count());
```

Nesse caso, o número de usuários no banco de dados retornado por User::count() deve ser zero, já que, dentro do teste, estamos usando o atributo RefreshDatabase.

Passar erros para a sessão

Do controller, usando o método withErrors, podemos passar uma matriz de erros ao executar um redirecionamento. Esses erros serão armazenados na sessão:

```
return redirect('usuarios/novo')->withErrors([
    'name' => 'O campo nome é obrigatório'
]);
```

Validação automática com validate()

Podemos chamar o método validate() no objeto request, passando como um valor uma matriz associativa onde cada chave será o nome de cada campo esperado e o valor uma string com as regras de validação:

```
$data = request()->validate([
    'name' => 'required'
]);
```

Passando outro array como segundo argumento, podemos especificar as mensagens dos erros. Para o nome do campo, precisamos acrescentar o nome da regra de validação (neste caso, required e notar que "name" e "required" são separados por um ponto):

```
$data = request()->validate([
    'name' => 'required'
], [
    'name.required' => 'O campo nome é obrigatório'
]);
```

Aula 29 - Mostrar mensagem de erros de validação no Laravel

Nesta lição, veremos como podemos mostrar mensagens de validação em nossas visualizações, usando a variável $errors e os diferentes métodos que o Laravel nos fornece. Também veremos como podemos preservar o valor de um campo caso ocorram erros de validação.

Mostrar mensagens de validação

Para trabalhar com mensagens de validação nas views, usamos a variável $errors. Esta variável é automaticamente transmitida pelo Laravel, seja ou não um erro registrado na sessão. Esta variável contém uma instância de Illuminate\Support\MessageBag e esta classe tem métodos diferentes que podemos usar para obter e exibir mensagens de erro para o usuário.

Um desses métodos é $errors->any(), com o qual podemos verificar se há um erro ou não. Este método retornará verdadeiro se houver um erro e falso se não houver nenhum:

```
@if ($errors->any())
   <p>Há erros!</p>
@endif
```

Outro método é $errors->all(). Chamando este método dentro de um foreach() podemos iterar através de todas as mensagens:

```
<ul>
   @foreach ($errors->all() as $error)
      <li>{{ $error }}</li>
   @endforeach
</ul>
```

Também podemos obter todas as mensagens de validação de um campo específico. Neste caso, usamos o método get() passando o nome do campo como um argumento:

```
<ul>
   @foreach ($errors->get('email') as $error)
      <li>{{ $error }}</li>
   @endforeach
</ul>
```

Se quisermos obter apenas a primeira mensagem de validação de um campo específico, usamos o método first(), passando como argumento o nome do campo:

```
<p>{{ $errors->first('email') }}</p>
```

Para determinar se há mensagens de validação para um campo, usamos o método has():

```
@if ($errors->has('email'))
   <p>{{ $errors->first('email') }}</p>
@endif
```

Observe como usamos o método first() junto com o método has()

Preservar o valor de um campo

Para preservar o valor de um campo, caso ocorram erros de validação e o usuário seja enviado de volta ao formulário, podemos usar a função helper old() passando como primeiro argumento o nome do campo, para atribuir o valor do atributo valor da tag input:

```
<input type="email" name="email" id="email"  placeholder="pedro@example.com" value="{{
old('email') }}">
```

Aula 30 - Uso de múltiplas regras de validação no Laravel

Nesta lição, continuaremos a adicionar regras de validação ao nosso formulário de registro, dessa vez nos guiando por meio de testes automatizados. Também veremos como podemos usar várias regras, como e-mail ou unique, para validar o mesmo campo.

Retornar todos os campos de validate()

Se quisermos executar a validação, mas também retornar os campos que não estamos validando, precisamos adicionar estes últimos ao array. O método validate() retornará todos os campos da matriz, independentemente de terem ou não regras de validação sempre que estiverem presentes:

```
$data = request()->validate([
    'name' => 'required',
    'email' => '',
    'password' => '',
], [
    'name.required' => 'O campo nome é obrigatório'
]);
```

Múltiplas regras de validação

Podemos especificar várias regras de validação, se as dividirmos com um caractere de barra vertical: |. No exemplo a seguir, não queremos apenas que o campo email seja obrigatório, mas também que ele contenha um endereço válido. Para isso, usamos a regra de validação de email:

```
$data = request()->validate([
    'name' => 'required',
    'email' => 'required|email',
    'password' => '',
], [
    'name.required' => 'O campo nome é obrigatório'
]);
```

Em vez de separar as regras com uma barra vertical, podemos passar uma matriz onde cada valor será o nome de uma regra diferente:

```
'email' => ['required', 'email'],
```

No caso de algumas regras de validação, como unique, devemos passar parâmetros diferentes:

'email' => ['required', 'email', 'unique:users,email'],

Nesse caso, especificamos que o campo de email deve ser único em relação ao campo de email da tabela users, para impedir que um usuário seja criado com um email que já pertence a outro usuário. Observe como os parâmetros são especificados com: seguido por cada parâmetro separado por vírgulas.

Personalizar mensagens de validação

Podemos passar mensagens personalizadas para cada uma das regras:

```
[
   'name.required' => 'O campo nome é obrigatório',
   'email.email' => 'Por favor entre com um email válido',
   'email.unique' => 'Já existe um usuário com esse email'
]
```

---

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;
use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      $users = User::all();
      $title = 'Lista de usuários';

      return view('users.index', compact('title', 'users'));
   }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }

   public function create()
   {
      return view('users.create');
   }

   public function store()
   {

      $data = request()->validate([
         'name' => 'required',
         'email' => ['required', 'email', 'unique:users,email'],
         'password' => ['required', 'min:6'],
      ], [
         'name.required' => 'O campo nome é obrigatório',
         'email.required' => 'O campo email é obrigatório',
         'password.required' => 'O campo senha é obrigatório',
         'email.email' => 'O email é inválido',
         'email.unique' => 'O email deve ser único',
         'password.min' => 'A senha deve ter no mínimo 6 caracteres'
      ]);

      User::create([
         'name' => $data['name'],
         'email' => $data['email'],
         'password' => bcrypt($data['password'])
      ]);
      return redirect()->route('users.index');
   }
}
```

tests/Feature/UserModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }

   /** @test */
```

```
    function it_displays_a_404_error_if_the_user_is_not_found()
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

    /** @test */
    function it_creates_a_new_user()
    {
        $this->withoutExceptionHandling();
        $this->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect('usuarios');
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

    /** @test */
    function the_name_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => '',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
        $this->assertEquals(0, User::count());
//      $this->assertDatabaseMissing('users', [
//          'email' => 'duilio@styde.net',
//      ]);
    }

    /** @test */
    function the_email_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => '',
                'password' => '123456'
            ])
```

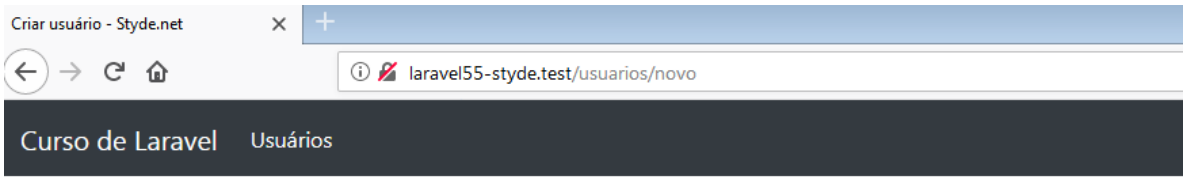
```
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_valid()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'correo-no-valido',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email é inválido']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_unique()
{
    factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email deve ser único']);
    $this->assertEquals(1, User::count());
}

/** @test */
function the_password_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => ''
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_password_must_contains_at_least_six_characters()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
```

```
            'password' => '12'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
    $this->assertEquals(0, User::count());
  }

}
```

resources/views/users/create.blade.php

```
@extends('layout')

@section('title', "Criar usuário")

@section('content')
  <h1>Criar usuário</h1>

  @if ($errors->any())
    <div class="alert alert-danger">
      <h6>Por favor corrija os erros abaixo:</h6>
      <ul>
        @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
        @endforeach
      </ul>
    </div>
  @endif

  <form method="POST" action="{{ url('usuarios') }}">
    {{ csrf_field() }}

    <label for="name">Nome:</label>
    <input type="text" name="name" id="name" placeholder="Pedro Perez" value="{{ old('name') }}">
    <br>
    <label for="email">Email:</label>
    <input type="email" name="email" id="email" placeholder="pedro@example.com" value="{{ old('email')
}}">
    <br>
    <label for="password">Senha:</label>
    <input type="password" name="password" id="password" placeholder="Maior que 6 caracteres">
    <br>
    <button type="submit">Criar usuário</button>
  </form>

  <p>
    <a href="{{ route('users.index') }}">Voltar à lista de usuários</a>
  </p>
@endsection
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.................                                                 16 / 16 (100%)

Time: 17.36 seconds, Memory: 16.00MB

OK (16 tests, 51 assertions)

C:\laragon\www\laravel55-styde
λ
```



Criar usuário

Por favor corrija os erros abaixo:

- O campo nome é obrigatório
- O campo email é obrigatório
- A senha deve ter no mínimo 6 caracteres

Nome: Pedro Perez

Email: pedro@example.com

Senha: Maior que 6 caracteres

Criar usuário

Voltar à lista de usuários

Aula 31 - Formulário para edição de usuários

Nesta lição, vamos começar a trabalhar no recurso para editar usuários, começando criando o formulário e obtendo os registros do banco de dados. É claro que seremos guiados pelo uso de testes automatizados e também aprenderemos a usar alguns métodos que não vimos antes, como assertViewIs.

Uso de assertViewIs e assertViewHas

Com assertViewIs podemos verificar que a view retornada do controller é o que esperamos. Neste caso, esperamos que a visão seja users.edit:

```
$this->get("/usuarios/{$user->id}/editar")
  ->assertStatus(200)
  ->assertViewIs('users.edit');
```

Com assertViewHas podemos verificar que a view contém a variável passada como argumento:

```
->assertViewHas('user');
```

Este método também aceita um valor (que pode ser uma função anônima) como um segundo argumento, caso precisemos ou desejemos ser mais explícitos na comparação:

```
->assertViewHas('user', function ($viewUser) use ($user) {
   return $viewUser->id === $user->id;
})
```

Mostrar os dados do usuário no formulário

Para mostrar os dados do usuário em forma de edição passamos como segundo argumento para a função old() os dados que estamos retornando do banco de dados:

```
<input type="text" name="name" id="name" placeholder="Pedro Perez" value="{{ old('name', $user->name) }}">
```

Nesse caso, o old() tentará carregar os dados referentes ao campo de nome que estão na sessão e, se não os obtiver, carregará o valor de $user->name.

---

tests/Feature/UsersModuleTest

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }

   /** @test */
   function it_displays_a_404_error_if_the_user_is_not_found()
```

```
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

    /** @test */
    function it_creates_a_new_user()
    {
        $this->withoutExceptionHandling();
        $this->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect('usuarios');
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

    /** @test */
    function the_name_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => '',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
        $this->assertEquals(0, User::count());
//      $this->assertDatabaseMissing('users', [
//          'email' => 'duilio@styde.net',
//      ]);
    }

    /** @test */
    function the_email_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => '',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
```

```
            ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
        $this->assertEquals(0, User::count());
    }

    /** @test */
    function the_email_must_be_valid()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'correo-no-valido',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['email' => 'O email é inválido']);
        $this->assertEquals(0, User::count());
    }

    /** @test */
    function the_email_must_be_unique()
    {
        factory(User::class)->create([
            'email' => 'duilio@styde.net'
        ]);
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['email' => 'O email deve ser único']);
        $this->assertEquals(1, User::count());
    }

    /** @test */
    function the_password_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => ''
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
        $this->assertEquals(0, User::count());
    }

    /** @test */
    function the_password_must_contains_at_least_six_characters()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => '12'
```

```
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
      $this->assertEquals(0, User::count());
   }

   /** @test */
   function it_loads_the_edit_user_page()
   {
      $this->withoutExceptionHandling();
      $user = factory(User::class)->create();
      $this->get("/usuarios/{$user->id}/editar") // usuarios/5/editar
      ->assertStatus(200)
         ->assertViewIs('users.edit')
         ->assertSee('Editar usuário')
         ->assertViewHas('user', function ($viewUser) use ($user) {
            return $viewUser->id === $user->id;
         });
   }

}
```

routes/web.php

```
<?php

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

Route::get('/usuarios', 'UserController@index')
   ->name('users.index');

Route::get('/usuarios/{user}', 'UserController@show')
   ->where('user', '[0-9]+')
   ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::get('/usuarios/{user}/editar', 'UserController@edit')->name('users.edit');

Route::post('/usuarios', 'UserController@store');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

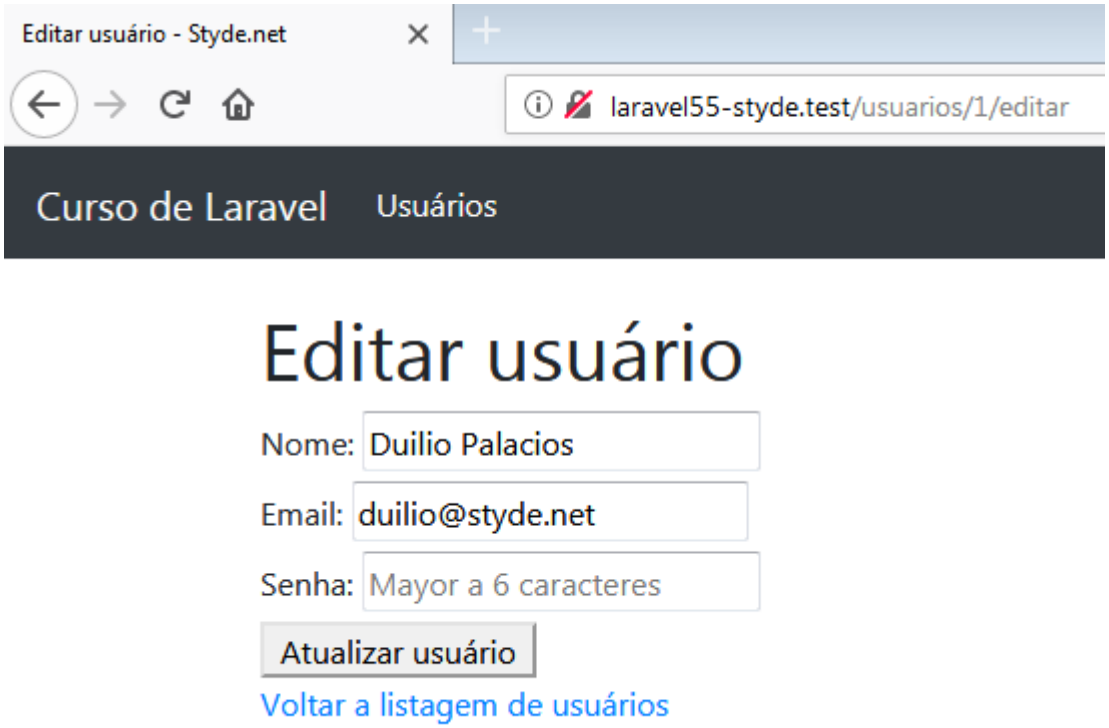
app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

//       return view('users.index')
//          ->with('users', User::all())
//          ->with('title', 'Lista de usuários');

      return view('users.index', compact('title', 'users'));
   }

//    public function show($id)
//    {
//       $user = User::findOrFail($id);
//       return view('users.show', compact('user'));
//    }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }


   public function create()
   {
      return view('users.create');
   }

   public function store()
   {

      $data = request()->validate([
         'name' => 'required',
         'email' => ['required', 'email', 'unique:users,email'],
         'password' => ['required', 'min:6'],
      ], [
         'name.required' => 'O campo nome é obrigatório',
         'email.required' => 'O campo email é obrigatório',
```

```
            'password.required' => 'O campo senha é obrigatório',
            'email.email' => 'O email é inválido',
            'email.unique' => 'O email deve ser único',
            'password.min' => 'A senha deve ter no mínimo 6 caracteres'
        ]);

        User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password'])
        ]);
        return redirect()->route('users.index');
    }

    public function edit(User $user)
    {
        return view('users.edit', ['user' => $user]);
    }

}
```

resources/views/users/edit.blade.php

```
@extends('layout')

@section('title', "Editar usuário")

@section('content')
  <h1>Editar usuário</h1>

  @if ($errors->any())
    <div class="alert alert-danger">
      <h6>Por favor corrija os erros abaixo:</h6>
      <ul>
        @foreach ($errors->all() as $error)
          <li>{{ $error }}</li>
        @endforeach
      </ul>
    </div>
  @endif

  <form method="POST" action="{{ url('usuarios') }}">
    {{ csrf_field() }}

    <label for="name">Nome:</label>
    <input type="text" name="name" id="name" placeholder="Pedro Perez" value="{{ old('name',
$user->name) }}">
    <br>
    <label for="email">Email:</label>
    <input type="email" name="email" id="email" placeholder="pedro@example.com" value="{{
old('email', $user->email) }}">
    <br>
    <label for="password">Senha:</label>
    <input type="password" name="password" id="password" placeholder="Mayor a 6 caracteres">
    <br>
    <button type="submit">Atualizar usuário</button>
  </form>

  <p>
    <a href="{{ route('users.index') }}">Voltar a listagem de usuários</a>
  </p>
@endsection
```


Editar usuário - Styde.net
laravel55-styde.test/usuarios/1/editar
Curso de Laravel
Usuários
Editar usuário
Nome: Duilio Palacios
Email: duilio@styde.net
Senha: Mayor a 6 caracteres
Atualizar usuário
Voltar a listagem de usuários

Aula 32 - Atualização de usuários com Laravel

Nesta lição você aprenderá a atualizar os registros usando o Laravel e o Eloquent ORM, enquanto trabalha no módulo do usuário. Para isso, mais uma vez, vamos contar com testes automatizados e a metodologia TDD.

Utilizando o método PUT

Devemos enviar a solicitação para editar usuários usando o método put() em vez de post():

```
$this->put("/usuarios/{$user->id}", [
   'name' => 'Duilio',
   'email' => 'duilio@styde.net',
   'password' => '123456'
])->assertRedirect('usuarios');
```

A URL é semelhante a URL para mostrar um usuário /users/{$user->id}, que varia em cada um deles é o nome do método:

- O método GET usando $this->get() mostra a página de detalhes.
- O método PUT usando $this->put() executa a ação para atualizar.

No web.php devemos definir a rota usando o método Route::put():

```
Route::put('/usuarios/{user}', 'UserController@update');
```

Lembre-se de que você deve criptografar a senha usando o bcrypt antes de passar os dados para o método update():

```
$data = request()->all();

$data['password'] = bcrypt($data['password']);

$data->update($data);
```

Na próxima lição, vamos trabalhar na questão da validação.

---

tests/Feature/UsersModuleTest

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }
```

```
    /** @test */
    function it_displays_a_404_error_if_the_user_is_not_found()
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

    /** @test */
    function it_creates_a_new_user()
    {
        $this->withoutExceptionHandling();
        $this->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect('usuarios');
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

    /** @test */
    function the_name_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => '',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
        $this->assertEquals(0, User::count());
//        $this->assertDatabaseMissing('users', [
//            'email' => 'duilio@styde.net',
//        ]);
    }

    /** @test */
    function the_email_is_required()
    {
        $this->from('usuarios/novo')
```

```
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => '',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_valid()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'correo-no-valido',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email é inválido']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_unique()
{
    factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email deve ser único']);
    $this->assertEquals(1, User::count());
}

/** @test */
function the_password_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => ''
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
    $this->assertEquals(0, User::count());
}
```

```
    /** @test */
    function the_password_must_contains_at_least_six_characters()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => '12'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
        $this->assertEquals(0, User::count());
    }

    /** @test */
    function it_loads_the_edit_user_page()
    {
        $this->withoutExceptionHandling();
        $user = factory(User::class)->create();
        $this->get("/usuarios/{$user->id}/editar") // usuarios/5/editar
        ->assertStatus(200)
            ->assertViewIs('users.edit')
            ->assertSee('Editar usuário')
            ->assertViewHas('user', function ($viewUser) use ($user) {
                return $viewUser->id === $user->id;
            });
    }

    /** @test */
    function it_updates_a_user()
    {
        $user = factory(User::class)->create();
        $this->withoutExceptionHandling();
        $this->put("/usuarios/{$user->id}", [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect("/usuarios/{$user->id}");
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

}
```

routes/web.php

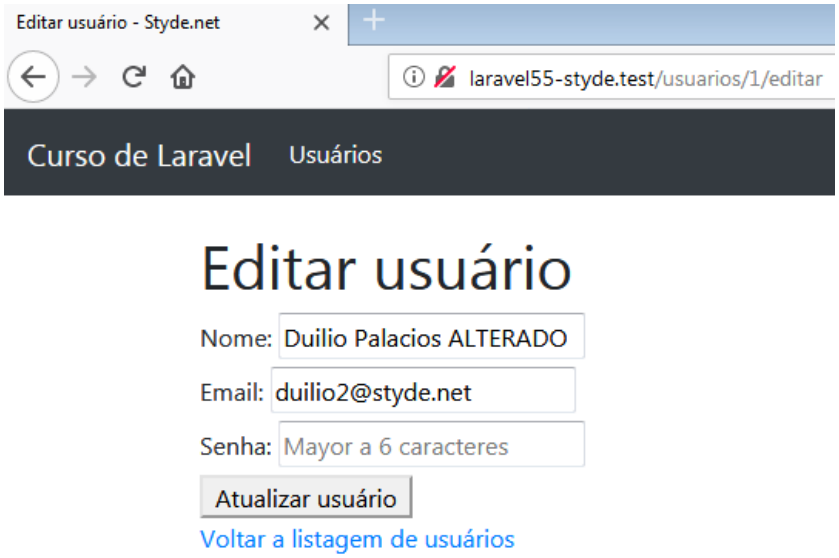
```php
<?php

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

Route::get('/usuarios', 'UserController@index')
   ->name('users.index');

Route::get('/usuarios/{user}', 'UserController@show')
   ->where('user', '[0-9]+')
   ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::get('/usuarios/{user}/editar', 'UserController@edit')->name('users.edit');

Route::put('/usuarios/{user}', 'UserController@update');

Route::post('/usuarios', 'UserController@store');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

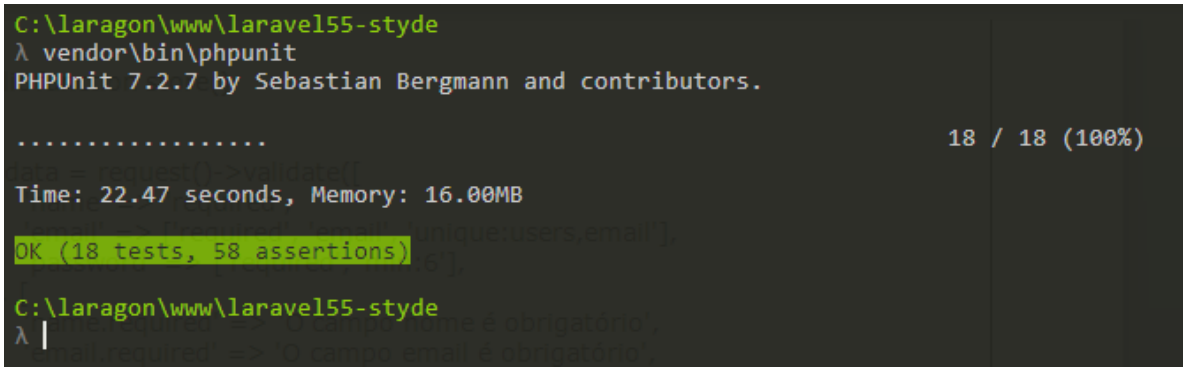
GET/usuarios/{id} pág. de detalhes
PUT/usuarios/{id} ação para atualizar

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

      return view('users.index', compact('title', 'users'));
   }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }

   public function create()
   {
      return view('users.create');
   }

   public function store()
   {

      $data = request()->validate([
         'name' => 'required',
         'email' => ['required', 'email', 'unique:users,email'],
         'password' => ['required', 'min:6'],
      ], [
         'name.required' => 'O campo nome é obrigatório',
         'email.required' => 'O campo email é obrigatório',
         'password.required' => 'O campo senha é obrigatório',
         'email.email' => 'O email é inválido',
         'email.unique' => 'O email deve ser único',
         'password.min' => 'A senha deve ter no mínimo 6 caracteres'
      ]);

      User::create([
         'name' => $data['name'],
         'email' => $data['email'],
         'password' => bcrypt($data['password'])
      ]);
```

```
        return redirect()->route('users.index');
    }

    public function edit(User $user)
    {
        return view('users.edit', ['user' => $user]);
    }

    public function update(User $user)
    {
        $data = request()->all();
        $data['password'] = bcrypt($data['password']);

        $user->update($data);
        return redirect()->route('users.show', ['user' => $user]);
    }

}
```





Aula 33 - Regras de validação para atualização de usuários com Laravel e TDD

Nesta lição, continuaremos com a criação da funcionalidade para editar usuários, para isso, adicionaremos as regras de validação à ação do controlador e mostraremos os erros de validação no formulário para editar os usuários. Também veremos como podemos enviar solicitações PUT de um formulário usando o helper method_field do Laravel.

Lembre-se de que devemos colocar o método from() com a URL do formulário antes do método put() para provar que o redirecionamento foi bem-sucedido:

```
$this->from("usuarios/{user->id}/editar")->put("usuarios/{$user->id}", [
   // ...
]);
```

Receber os dados da petição

Com request()->validate(), recebemos os dados da petição e verificamos que eles estão em conformidade com as regras de validação especificadas na matriz associativa:

```
$data = request()->validate([
   'name' => 'required',
   'email' => '',
   'password' => '',
]);
```

Indicar o método PUT no formulário

Os formulários padrão suportam apenas os métodos GET ou POST. No Laravel podemos usar o helper method_field() para gerar um campo oculto onde podemos indicar o tipo de requisição. Como um argumento para method_field() nós passamos o nome do método HTTP:

```
<form method="POST" action="{{ url("usuarios/{$user->id}") }}">
   {{ method_field('PUT') }}

   ...
</form>
```

---

tests/Feature/UsersModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }
```

```
    /** @test */
    function it_displays_a_404_error_if_the_user_is_not_found()
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

    /** @test */
    function it_creates_a_new_user()
    {
        $this->withoutExceptionHandling();
        $this->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect('usuarios');
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

    /** @test */
    function the_name_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => '',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
        $this->assertEquals(0, User::count());
//        $this->assertDatabaseMissing('users', [
//            'email' => 'duilio@styde.net',
//        ]);
    }

    /** @test */
    function the_email_is_required()
    {
        $this->from('usuarios/novo')
```

```
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => '',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_valid()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'correo-no-valido',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email é inválido']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_unique()
{
    factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email deve ser único']);
    $this->assertEquals(1, User::count());
}

/** @test */
function the_password_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => ''
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
    $this->assertEquals(0, User::count());
}
```

```
/** @test */
function the_password_must_contains_at_least_six_characters()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '12'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
    $this->assertEquals(0, User::count());
}

/** @test */
function it_loads_the_edit_user_page()
{
    $this->withoutExceptionHandling();
    $user = factory(User::class)->create();
    $this->get("/usuarios/{$user->id}/editar") // usuarios/5/editar
    ->assertStatus(200)
        ->assertViewIs('users.edit')
        ->assertSee('Editar usuário')
        ->assertViewHas('user', function ($viewUser) use ($user) {
            return $viewUser->id === $user->id;
        });
}

/** @test */
function it_updates_a_user()
{
    $user = factory(User::class)->create();
    $this->withoutExceptionHandling();
    $this->put("/usuarios/{$user->id}", [
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456'
    ])->assertRedirect("/usuarios/{$user->id}");
    $this->assertCredentials([
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456',
    ]);
}

/** @test */
function the_name_is_required_when_updating_the_user()
{
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => '',
            'email' => 'duilio@styde.net',
```

```
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
        ->assertSessionHasErrors(['name']);
    $this->assertDatabaseMissing('users', ['email' => 'duilio@styde.net']);
  }

  /** @test */
  function the_email_must_be_valid_when_updating_the_user()
  {
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
      ->put("usuarios/{$user->id}", [
        'name' => 'Duilio Palacios',
        'email' => 'email não válido',
        'password' => '123456'
      ])
      ->assertRedirect("usuarios/{$user->id}/editar")
      ->assertSessionHasErrors(['email']);
    $this->assertDatabaseMissing('users', ['name' => 'Duilio Palacios']);
  }

  /** @test */
  function the_users_email_can_stay_the_same_when_updating_the_user()
  {
    $user = factory(User::class)->create([
      'email' => 'duilio@styde.net'
    ]);
    $this->from("usuarios/{$user->id}/editar")
      ->put("usuarios/{$user->id}", [
        'name' => 'Duilio Palacios',
        'email' => 'duilio@styde.net',
        'password' => '12345678'
      ])
      ->assertRedirect("usuarios/{$user->id}"); // (users.show)
    $this->assertDatabaseHas('users', [
      'name' => 'Duilio Palacios',
      'email' => 'duilio@styde.net',
    ]);
  }

}
```

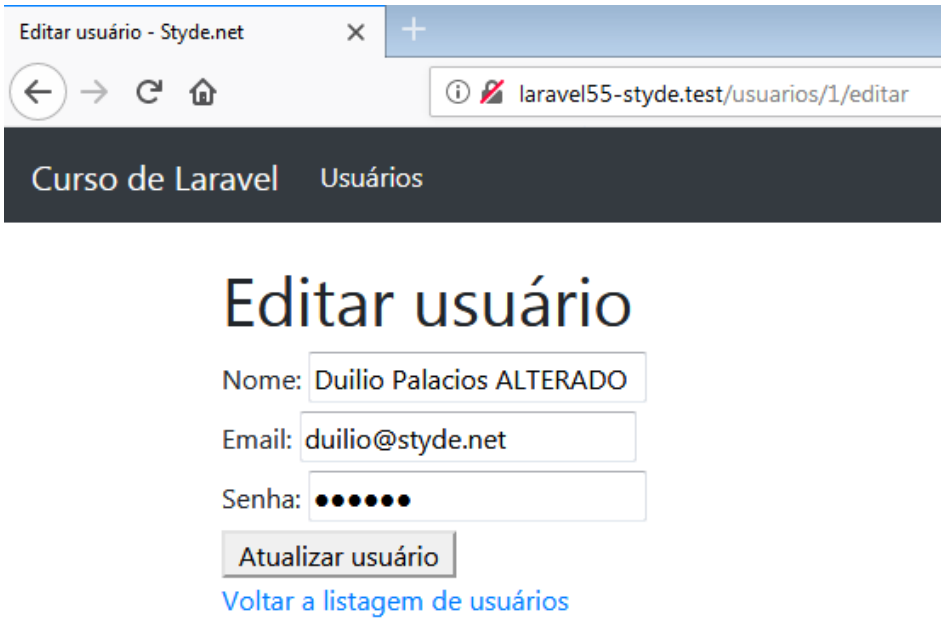
app/Http/Controllers/UserController.php

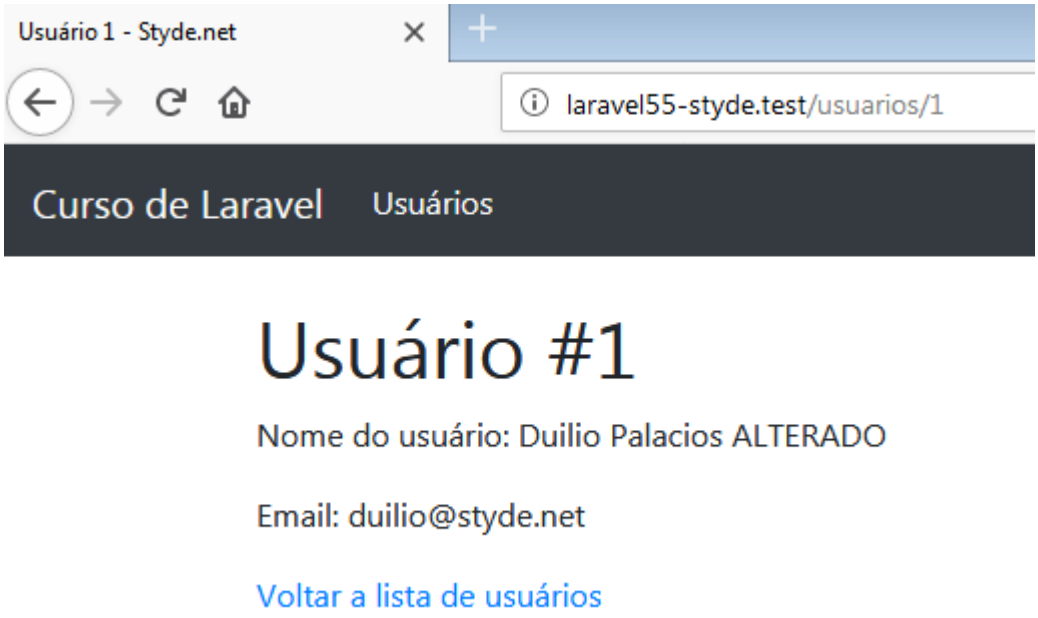
```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';
      return view('users.index', compact('title', 'users'));
   }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }


   public function create()
   {
      return view('users.create');
   }

   public function store()
   {

      $data = request()->validate([
         'name' => 'required',
         'email' => ['required', 'email', 'unique:users,email'],
         'password' => ['required', 'min:6'],
      ], [
         'name.required' => 'O campo nome é obrigatório',
         'email.required' => 'O campo email é obrigatório',
         'password.required' => 'O campo senha é obrigatório',
         'email.email' => 'O email é inválido',
         'email.unique' => 'O email deve ser único',
         'password.min' => 'A senha deve ter no mínimo 6 caracteres'
      ]);

      User::create([
         'name' => $data['name'],
         'email' => $data['email'],
         'password' => bcrypt($data['password'])
      ]);
```

```
        return redirect()->route('users.index');
    }

    public function edit(User $user)
    {
        return view('users.edit', ['user' => $user]);
    }

    public function update(User $user)
    {

        $data = request()->validate([
            'name' => 'required',
            'email' => ['required', 'email'],
            'password' => 'required',
        ]);

        $data['password'] = bcrypt($data['password']);

        $user->update($data);
        return redirect()->route('users.show', ['user' => $user]);
    }

}
```

resources/views/users/edit.blade.php

```
@extends('layout')

@section('title', "Editar usuário")

@section('content')
  <h1>Editar usuário</h1>

  @if ($errors->any())
    <div class="alert alert-danger">
      <h6>Por favor corrija os erros abaixo:</h6>
      <ul>
        @foreach ($errors->all() as $error)
          <li>{{ $error }}</li>
        @endforeach
      </ul>
    </div>
  @endif

  <form method="POST" action="{{ url("usuarios/{$user->id}") }}">
    {{ method_field('PUT') }}
    {{ csrf_field() }}
    <label for="name">Nome:</label>
    <input type="text" name="name" id="name" placeholder="Pedro Perez" value="{{ old('name',
$user->name) }}">
    <br>
    <label for="email">Email:</label>
    <input type="email" name="email" id="email" placeholder="pedro@example.com" value="{{
old('email', $user->email) }}">
    <br>
    <label for="password">Senha:</label>
    <input type="password" name="password" id="password" placeholder="Mayor a 6 caracteres">
    <br>
    <button type="submit">Atualizar usuário</button>
  </form>

  <p>
    <a href="{{ route('users.index') }}">Voltar a listagem de usuários</a>
  </p>
@endsection
```






```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.....................                                             21 / 21 (100%)

Time: 10.46 seconds, Memory: 16.00MB

OK (21 tests, 71 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 34 - Campo opcional de senha em Laravel

Nesta lição continuaremos com a construção da ação para editar usuários e aprenderemos como podemos tornar o campo opcional para a senha.

Lembre-se que você pode traduzir mensagens de erro passando um array associativo como um segundo argumento para o método validate():

```
$data = request()->validate([
   'name' => 'required'
], [
   'name.required' => 'O campo nome é obrigatório'
]);
```

Com assertCredentials podemos verificar as credenciais do usuário no banco de dados. Este método aceita um array com as credenciais como o primeiro argumento:

```
$this->assertCredentials([
   'name' => 'Duilio',
   'email' => 'duilio@styde.net',
   'password' => $oldPassword
]);
```

Dentro do controlador, verificamos se a senha não é nula e se não está criptografada, caso contrário, usamos unset() para eliminar o índice password $data. (O Laravel converte automaticamente campos de formulário vazios em nulos)

```
if ($data['password'] != null) {
   $data['password'] = bcrypt($data['password']);
} else {
   unset($data['password']);
}
```

---

tests/Feature/UsersModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }

   /** @test */
```

```
function it_displays_a_404_error_if_the_user_is_not_found()
{
    $this->get('/usuarios/999')
        ->assertStatus(404)
        ->assertSee('Página não encontrada!');
}

/** @test */
function it_loads_the_new_users_page()
{
    $this->get('/usuarios/novo')
        ->assertStatus(200)
        ->assertSee('Criar usuário');
}

/** @test */
function it_creates_a_new_user()
{
    $this->withoutExceptionHandling();
    $this->post('/usuarios/', [
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456'
    ])->assertRedirect('usuarios');
    $this->assertCredentials([
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456',
    ]);
}

/** @test */
function the_name_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => '',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => '',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
    $this->assertEquals(0, User::count());
```

```
    }

    /** @test */
    function the_email_must_be_valid()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'correo-no-valido',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['email' => 'O email é inválido']);
        $this->assertEquals(0, User::count());
    }

    /** @test */
    function the_email_must_be_unique()
    {
        factory(User::class)->create([
            'email' => 'duilio@styde.net'
        ]);
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['email' => 'O email deve ser único']);
        $this->assertEquals(1, User::count());
    }

    /** @test */
    function the_password_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => ''
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
        $this->assertEquals(0, User::count());
    }

    /** @test */
    function the_password_must_contains_at_least_six_characters()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => '12'
            ])
            ->assertRedirect('usuarios/novo')
```

```
        ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
    $this->assertEquals(0, User::count());
}

/** @test */
function it_loads_the_edit_user_page()
{
    $this->withoutExceptionHandling();
    $user = factory(User::class)->create();
    $this->get("/usuarios/{$user->id}/editar") // usuarios/5/editar
    ->assertStatus(200)
        ->assertViewIs('users.edit')
        ->assertSee('Editar usuário')
        ->assertViewHas('user', function ($viewUser) use ($user) {
            return $viewUser->id === $user->id;
        });
}

/** @test */
function it_updates_a_user()
{
    $user = factory(User::class)->create();
    $this->withoutExceptionHandling();
    $this->put("/usuarios/{$user->id}", [
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456'
    ])->assertRedirect("/usuarios/{$user->id}");
    $this->assertCredentials([
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456',
    ]);
}

/** @test */
function the_name_is_required_when_updating_the_user()
{
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => '',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
        ->assertSessionHasErrors(['name']);
    $this->assertDatabaseMissing('users', ['email' => 'duilio@styde.net']);
}

/** @test */
function the_email_must_be_valid_when_updating_the_user()
{
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => 'Duilio Palacios',
```

```
            'email' => 'email não válido',
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
        ->assertSessionHasErrors(['email']);
    $this->assertDatabaseMissing('users', ['name' => 'Duilio Palacios']);
  }

  /** @test */
  function the_users_email_can_stay_the_same_when_updating_the_user()
  {
    $user = factory(User::class)->create([
      'email' => 'duilio@styde.net'
    ]);
    $this->from("usuarios/{$user->id}/editar")
      ->put("usuarios/{$user->id}", [
        'name' => 'Duilio Palacios',
        'email' => 'duilio@styde.net',
        'password' => '12345678'
      ])
      ->assertRedirect("usuarios/{$user->id}"); // (users.show)
    $this->assertDatabaseHas('users', [
      'name' => 'Duilio Palacios',
      'email' => 'duilio@styde.net',
    ]);
  }

  /** @test */
  function the_password_is_optional_when_updating_the_user()
  {
    $oldPassword = 'CLAVE_ANTERIOR';
    $user = factory(User::class)->create([
      'password' => bcrypt($oldPassword)
    ]);
    $this->from("usuarios/{$user->id}/editar")
      ->put("usuarios/{$user->id}", [
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => ''
      ])
      ->assertRedirect("usuarios/{$user->id}"); // (users.show)
    $this->assertCredentials([
      'name' => 'Duilio',
      'email' => 'duilio@styde.net',
      'password' => $oldPassword // VERY IMPORTANT!
    ]);
  }

}
```

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';
      return view('users.index', compact('title', 'users'));
   }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }

   public function create()
   {
      return view('users.create');
   }

   public function store()
   {

      $data = request()->validate([
         'name' => 'required',
         'email' => ['required', 'email', 'unique:users,email'],
         'password' => ['required', 'min:6'],
      ], [
         'name.required' => 'O campo nome é obrigatório',
         'email.required' => 'O campo email é obrigatório',
         'password.required' => 'O campo senha é obrigatório',
         'email.email' => 'O email é inválido',
         'email.unique' => 'O email deve ser único',
         'password.min' => 'A senha deve ter no mínimo 6 caracteres'
      ]);

      User::create([
         'name' => $data['name'],
         'email' => $data['email'],
         'password' => bcrypt($data['password'])
      ]);
      return redirect()->route('users.index');
   }
```

```
    public function edit(User $user)
    {
        return view('users.edit', ['user' => $user]);
    }

    public function update(User $user)
    {

        $data = request()->validate([
            'name' => 'required',
            'email' => ['required', 'email'],
            'password' => ''
        ]);

        if ($data['password'] != null) {
            $data['password'] = bcrypt($data['password']);
        } else {
            unset($data['password']);
        }

        $user->update($data);
        return redirect()->route('users.show', ['user' => $user]);
    }

}
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

......................                                          22 / 22 (100%)

Time: 9.33 seconds, Memory: 16.00MB

OK (22 tests, 74 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 35 - Validar que o email seja único quando se edita um usuário com Laravel

Você aprenderá como é possível anexar a regra de validação para que o campo de e-mail do usuário possa ser único de forma correta na ação para editar usuários.

Podemos excluir um usuário da lista de usuários onde faremos a validação. Nesse caso, queremos verificar se o valor é único na tabela de usuários na coluna de email, mas excluindo o usuário que passamos com $user->id como o terceiro parâmetro da regra exclusiva:

```
$data = request()->validate([
    'name' => 'required',
    'email' => 'required|email|unique:users,email,'.$user->id,
    'password' => '',
]);
```

Também podemos usar a interface orientada a objetos fornecida pelo Laravel com a classe Rule:

```
'email' => ['required', 'email', Rule::unique('users')->ignore($user->id)]
```

Se a sua tabela não usa uma coluna chamada id como chave primária, você precisa especificar o nome da coluna ao chamar o método ignore, por exemplo:

```
'email' => ['required', 'email', Rule::unique('users')->ignore($user->id, 'user_id')]
```

---

tests/Feature/UsersModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }

   /** @test */
```

```
    function it_displays_a_404_error_if_the_user_is_not_found()
    {
        $this->get('/usuarios/999')
            ->assertStatus(404)
            ->assertSee('Página não encontrada!');
    }

    /** @test */
    function it_loads_the_new_users_page()
    {
        $this->get('/usuarios/novo')
            ->assertStatus(200)
            ->assertSee('Criar usuário');
    }

    /** @test */
    function it_creates_a_new_user()
    {
        $this->withoutExceptionHandling();
        $this->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])->assertRedirect('usuarios');
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456',
        ]);
    }

    /** @test */
    function the_name_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => '',
                'email' => 'duilio@styde.net',
                'password' => '123456'
            ])
            ->assertRedirect('usuarios/novo')
            ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
        $this->assertEquals(0, User::count());
//      $this->assertDatabaseMissing('users', [
//          'email' => 'duilio@styde.net',
//      ]);
    }

    /** @test */
    function the_email_is_required()
    {
        $this->from('usuarios/novo')
            ->post('/usuarios/', [
                'name' => 'Duilio',
                'email' => '',
                'password' => '123456'
            ])
```

```
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_valid()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'correo-no-valido',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email é inválido']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_unique()
{
    factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email deve ser único']);
    $this->assertEquals(1, User::count());
}

/** @test */
function the_password_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => ''
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_password_must_contains_at_least_six_characters()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
```

```
            'password' => '12'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
    $this->assertEquals(0, User::count());
}

/** @test */
function it_loads_the_edit_user_page()
{
    $this->withoutExceptionHandling();
    $user = factory(User::class)->create();
    $this->get("/usuarios/{$user->id}/editar") // usuarios/5/editar
    ->assertStatus(200)
        ->assertViewIs('users.edit')
        ->assertSee('Editar usuário')
        ->assertViewHas('user', function ($viewUser) use ($user) {
            return $viewUser->id === $user->id;
        });
}

/** @test */
function it_updates_a_user()
{
    $user = factory(User::class)->create();
    $this->withoutExceptionHandling();
    $this->put("/usuarios/{$user->id}", [
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456'
    ])->assertRedirect("/usuarios/{$user->id}");
    $this->assertCredentials([
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456',
    ]);
}

/** @test */
function the_name_is_required_when_updating_the_user()
{
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => '',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
        ->assertSessionHasErrors(['name']);
    $this->assertDatabaseMissing('users', ['email' => 'duilio@styde.net']);
}

/** @test */
function the_email_must_be_valid_when_updating_the_user()
{
    $user = factory(User::class)->create();
```

```
        $this->from("usuarios/{$user->id}/editar")
            ->put("usuarios/{$user->id}", [
                'name' => 'Duilio Palacios',
                'email' => 'email não válido',
                'password' => '123456'
            ])
            ->assertRedirect("usuarios/{$user->id}/editar")
            ->assertSessionHasErrors(['email']);
        $this->assertDatabaseMissing('users', ['name' => 'Duilio Palacios']);
    }

    /** @test */
    function the_email_must_be_unique_when_updating_the_user()
    {
        //$this->withoutExceptionHandling();
        factory(User::class)->create([
            'email' => 'existing-email@example.com',
        ]);
        $user = factory(User::class)->create([
            'email' => 'duilio@styde.net'
        ]);
        $this->from("usuarios/{$user->id}/editar")
            ->put("usuarios/{$user->id}", [
                'name' => 'Duilio',
                'email' => 'existing-email@example.com',
                'password' => '123456'
            ])
            ->assertRedirect("usuarios/{$user->id}/editar")
            ->assertSessionHasErrors(['email']);
        //
    }

    /** @test */
    function the_users_email_can_stay_the_same_when_updating_the_user()
    {
        $user = factory(User::class)->create([
            'email' => 'duilio@styde.net'
        ]);
        $this->from("usuarios/{$user->id}/editar")
            ->put("usuarios/{$user->id}", [
                'name' => 'Duilio Palacios',
                'email' => 'duilio@styde.net',
                'password' => '12345678'
            ])
            ->assertRedirect("usuarios/{$user->id}"); // (users.show)
        $this->assertDatabaseHas('users', [
            'name' => 'Duilio Palacios',
            'email' => 'duilio@styde.net',
        ]);
    }

    /** @test */
    function the_password_is_optional_when_updating_the_user()
    {
        $oldPassword = 'CLAVE_ANTERIOR';
        $user = factory(User::class)->create([
            'password' => bcrypt($oldPassword)
```

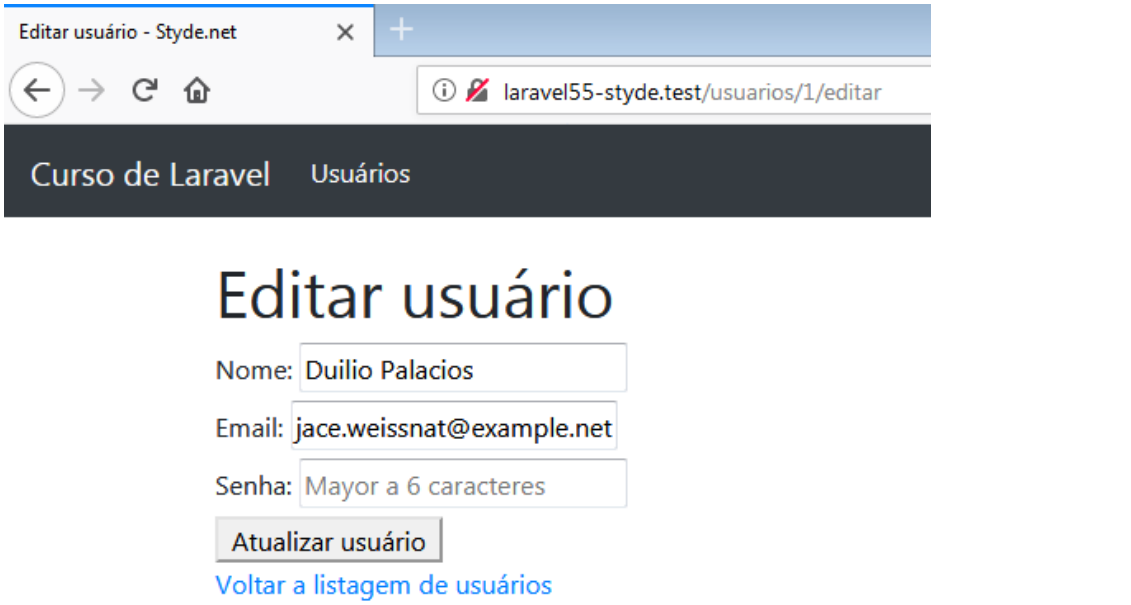
```
        ]);
        $this->from("usuarios/{$user->id}/editar")
            ->put("usuarios/{$user->id}", [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => ''
            ])
            ->assertRedirect("usuarios/{$user->id}"); // (users.show)
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => $oldPassword // VERY IMPORTANT!
        ]);
    }

}
```

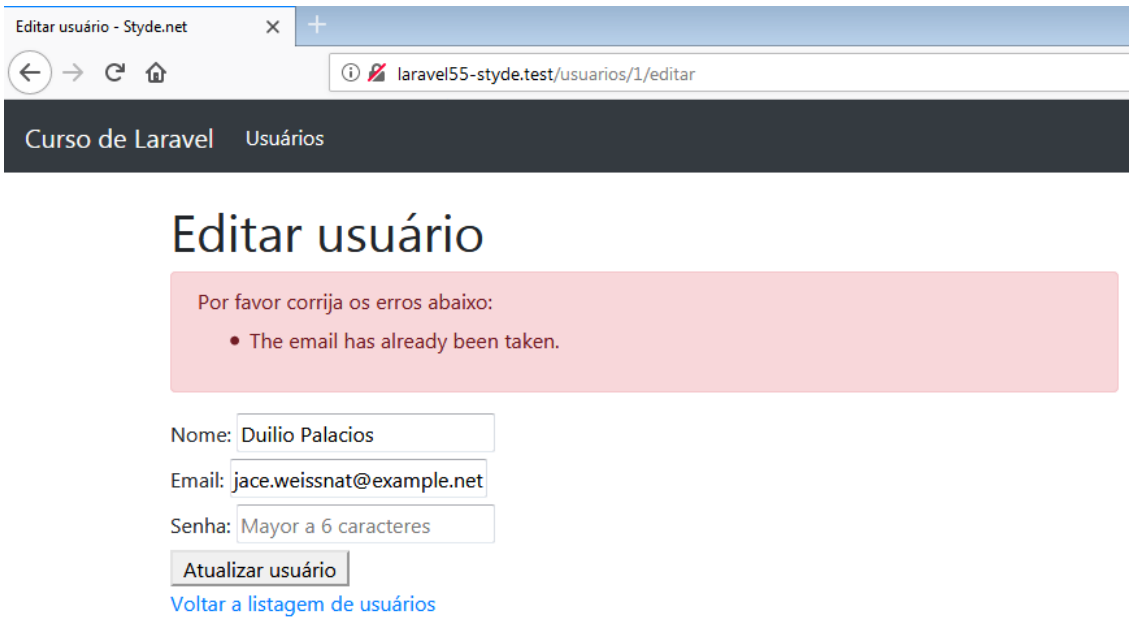
app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class UserController extends Controller
{

    public function index()
    {
        //$users = DB::table('users')->get();
        $users = User::all();
        $title = 'Lista de usuários';

        return view('users.index', compact('title', 'users'));
    }

    public function show(User $user)
    {
        return view('users.show', compact('user'));
    }


    public function create()
    {
        return view('users.create');
    }
```

```
    public function store()
    {

        $data = request()->validate([
            'name' => 'required',
            'email' => ['required', 'email', 'unique:users,email'],
            'password' => ['required', 'min:6'],
        ], [
            'name.required' => 'O campo nome é obrigatório',
            'email.required' => 'O campo email é obrigatório',
            'password.required' => 'O campo senha é obrigatório',
            'email.email' => 'O email é inválido',
            'email.unique' => 'O email deve ser único',
            'password.min' => 'A senha deve ter no mínimo 6 caracteres'
        ]);

        User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password'])
        ]);
        return redirect()->route('users.index');
    }

    public function edit(User $user)
    {
        return view('users.edit', ['user' => $user]);
    }

    public function update(User $user)
    {

        $data = request()->validate([
            'name' => 'required',
            'email' => ['required', 'email', Rule::unique('users')->ignore($user->id)],
            'password' => '',
        ]);

        if ($data['password'] != null) {
            $data['password'] = bcrypt($data['password']);
        } else {
            unset($data['password']);
        }

        $user->update($data);
        return redirect()->route('users.show', ['user' => $user]);
    }

}
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.......................                                         23 / 23 (100%)

Time: 29.68 seconds, Memory: 16.00MB

OK (23 tests, 78 assertions)

C:\laragon\www\laravel55-styde
λ
```





Aula 36 - Eliminar registros com Laravel e TDD

Continuaremos com a criação do módulo usuário, desta vez trabalharemos na ação de eliminar usuários do banco de dados utilizando desenvolvimento orientado por testes automatizados (TDD).

Se tentarmos acessar um URL que existe, mas usando um método diferente, receberemos uma exceção do tipo MethodNotAllowed.

No arquivo de rotas, usamos o método delete:

```
Route::delete('/usuarios/{user}', 'UserController@destroy');
```

Dentro do teste nós também usamos o método delete chamando $this->delete:

```
$this->delete("users/{$user->id}");
```

Com assertDatabaseMissing, verificamos que o registro não existe mais no banco de dados. Como primeiro argumento, passamos o nome da tabela e, como segundo, a matriz de atributos que não esperamos encontrar. Neste caso, não esperamos encontrar o registro com o último id:

```
$this->assertDatabaseMissing('users', [
   'id' => $user->id
]);
```

Também podemos usar assertSame, nós passamos o valor esperado como o primeiro argumento e o número atual de usuários como o segundo argumento:

```
$this->assertSame(0, User::count());
```

Esperaríamos zero usuários, já que eliminamos o único usuário criado.

Se estivermos carregando o modelo com a vinculação do modelo de rota, podemos chamar o método delete para excluir o registro diretamente:

```
function destroy(User $user) {
   $user->delete();

   return redirect()->route('users.index');
}
```

Caso contrário, teríamos que escrever algo como:

```
$user = User::findOrFail($id);

$user->delete();
```

Lembre-se de que, em vez de escrever a URL, você pode usar o nome da rota para redirecionar:

```
return redirect()->route('users.index');
```

---

tests/Feature/UsersModuleTest.php

```
<?php

namespace Tests\Feature;

use App\User;
use Tests\TestCase;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UsersModuleTest extends TestCase
{

   use RefreshDatabase;

   /** @test */
   function it_shows_the_users_list()
   {

      factory(User::class)->create([
         'name' => 'Joel'
      ]);

      factory(User::class)->create([
         'name' => 'Ellie',
      ]);

      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Lista de usuários')
         ->assertSee('Joel')
         ->assertSee('Ellie');
   }

   /** @test */
   function it_shows_a_default_message_if_the_users_list_is_empty()
   {
      $this->get('/usuarios')
         ->assertStatus(200)
         ->assertSee('Não há usuários registrados.');
   }

   /** @test */
   function it_displays_the_users_details()
   {
      $user = factory(User::class)->create([
         'name' => 'Duilio Palacios'
      ]);

      $this->get('/usuarios/'.$user->id) // usuarios/5
      ->assertStatus(200)
         ->assertSee('Duilio Palacios');
   }
```

```
/** @test */
function it_displays_a_404_error_if_the_user_is_not_found()
{
   $this->get('/usuarios/999')
      ->assertStatus(404)
      ->assertSee('Página não encontrada!');
}

/** @test */
function it_loads_the_new_users_page()
{
   $this->get('/usuarios/novo')
      ->assertStatus(200)
      ->assertSee('Criar usuário');
}

/** @test */
function it_creates_a_new_user()
{
   $this->withoutExceptionHandling();
   $this->post('/usuarios/', [
      'name' => 'Duilio',
      'email' => 'duilio@styde.net',
      'password' => '123456'
   ])->assertRedirect('usuarios');
   $this->assertCredentials([
      'name' => 'Duilio',
      'email' => 'duilio@styde.net',
      'password' => '123456',
   ]);
}

/** @test */
function the_name_is_required()
{
   $this->from('usuarios/novo')
      ->post('/usuarios/', [
         'name' => '',
         'email' => 'duilio@styde.net',
         'password' => '123456'
      ])
      ->assertRedirect('usuarios/novo')
      ->assertSessionHasErrors(['name' => 'O campo nome é obrigatório']);
   $this->assertEquals(0, User::count());
}

/** @test */
function the_email_is_required()
{
   $this->from('usuarios/novo')
      ->post('/usuarios/', [
         'name' => 'Duilio',
         'email' => '',
```

```
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O campo email é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_valid()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'correo-no-valido',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email é inválido']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_email_must_be_unique()
{
    factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['email' => 'O email deve ser único']);
    $this->assertEquals(1, User::count());
}

/** @test */
function the_password_is_required()
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => ''
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'O campo senha é obrigatório']);
    $this->assertEquals(0, User::count());
}

/** @test */
function the_password_must_contains_at_least_six_characters()
```

```
{
    $this->from('usuarios/novo')
        ->post('/usuarios/', [
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => '12'
        ])
        ->assertRedirect('usuarios/novo')
        ->assertSessionHasErrors(['password' => 'A senha deve ter no mínimo 6 caracteres']);
    $this->assertEquals(0, User::count());
}

/** @test */
function it_loads_the_edit_user_page()
{
    $this->withoutExceptionHandling();
    $user = factory(User::class)->create();
    $this->get("/usuarios/{$user->id}/editar") // usuarios/5/editar
    ->assertStatus(200)
        ->assertViewIs('users.edit')
        ->assertSee('Editar usuário')
        ->assertViewHas('user', function ($viewUser) use ($user) {
            return $viewUser->id === $user->id;
        });
}

/** @test */
function it_updates_a_user()
{
    $user = factory(User::class)->create();
    $this->withoutExceptionHandling();
    $this->put("/usuarios/{$user->id}", [
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456'
    ])->assertRedirect("/usuarios/{$user->id}");
    $this->assertCredentials([
        'name' => 'Duilio',
        'email' => 'duilio@styde.net',
        'password' => '123456',
    ]);
}

/** @test */
function the_name_is_required_when_updating_the_user()
{
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => '',
            'email' => 'duilio@styde.net',
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
```

```
        ->assertSessionHasErrors(['name']);
    $this->assertDatabaseMissing('users', ['email' => 'duilio@styde.net']);
}

/** @test */
function the_email_must_be_valid_when_updating_the_user()
{
    $user = factory(User::class)->create();
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => 'Duilio Palacios',
            'email' => 'email não válido',
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
        ->assertSessionHasErrors(['email']);
    $this->assertDatabaseMissing('users', ['name' => 'Duilio Palacios']);
}

/** @test */
function the_email_must_be_unique_when_updating_the_user()
{
    //$this->withoutExceptionHandling();
    factory(User::class)->create([
        'email' => 'existing-email@example.com',
    ]);
    $user = factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => 'Duilio',
            'email' => 'existing-email@example.com',
            'password' => '123456'
        ])
        ->assertRedirect("usuarios/{$user->id}/editar")
        ->assertSessionHasErrors(['email']);
    //
}

/** @test */
function the_users_email_can_stay_the_same_when_updating_the_user()
{
    $user = factory(User::class)->create([
        'email' => 'duilio@styde.net'
    ]);
    $this->from("usuarios/{$user->id}/editar")
        ->put("usuarios/{$user->id}", [
            'name' => 'Duilio Palacios',
            'email' => 'duilio@styde.net',
            'password' => '12345678'
        ])
        ->assertRedirect("usuarios/{$user->id}"); // (users.show)
    $this->assertDatabaseHas('users', [
```

```
            'name' => 'Duilio Palacios',
            'email' => 'duilio@styde.net',
        ]);
    }

    /** @test */
    function the_password_is_optional_when_updating_the_user()
    {
        $oldPassword = 'CLAVE_ANTERIOR';
        $user = factory(User::class)->create([
            'password' => bcrypt($oldPassword)
        ]);
        $this->from("usuarios/{$user->id}/editar")
            ->put("usuarios/{$user->id}", [
                'name' => 'Duilio',
                'email' => 'duilio@styde.net',
                'password' => ''
            ])
            ->assertRedirect("usuarios/{$user->id}"); // (users.show)
        $this->assertCredentials([
            'name' => 'Duilio',
            'email' => 'duilio@styde.net',
            'password' => $oldPassword // VERY IMPORTANT!
        ]);
    }

    /** @test */
    function it_deletes_a_user()
    {
        $this->withoutExceptionHandling();
        $user = factory(User::class)->create();
        $this->delete("usuarios/{$user->id}")
            ->assertRedirect('usuarios');
        $this->assertDatabaseMissing('users', [
            'id' => $user->id
        ]);
        //$this->assertSame(0, User::count());
    }

}
```

app/Http/Controllers/UserController.php

```
<?php

namespace App\Http\Controllers;

use App\User;
use Illuminate\Support\Facades\DB;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class UserController extends Controller
{

   public function index()
   {
      //$users = DB::table('users')->get();
      $users = User::all();
      $title = 'Lista de usuários';

      return view('users.index', compact('title', 'users'));
   }

   public function show(User $user)
   {
      return view('users.show', compact('user'));
   }


   public function create()
   {
      return view('users.create');
   }

   public function store()
   {

      $data = request()->validate([
         'name' => 'required',
         'email' => ['required', 'email', 'unique:users,email'],
         'password' => ['required', 'min:6'],
      ], [
         'name.required' => 'O campo nome é obrigatório',
         'email.required' => 'O campo email é obrigatório',
         'password.required' => 'O campo senha é obrigatório',
         'email.email' => 'O email é inválido',
         'email.unique' => 'O email deve ser único',
         'password.min' => 'A senha deve ter no mínimo 6 caracteres'
      ]);

      User::create([
         'name' => $data['name'],
         'email' => $data['email'],
```

```
            'password' => bcrypt($data['password'])
        ]);
        return redirect()->route('users.index');
    }

    public function edit(User $user)
    {
        return view('users.edit', ['user' => $user]);
    }

    public function update(User $user)
    {

        $data = request()->validate([
            'name' => 'required',
            'email' => ['required', 'email', Rule::unique('users')->ignore($user->id)],
            'password' => '',
        ]);

        if ($data['password'] != null) {
            $data['password'] = bcrypt($data['password']);
        } else {
            unset($data['password']);
        }

        $user->update($data);
        return redirect()->route('users.show', ['user' => $user]);
    }

    function destroy(User $user)
    {
        $user->delete();
        return redirect()->route('users.index');
    }

}
```

routes/web.php

```php
<?php

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

Route::get('/usuarios', 'UserController@index')
    ->name('users.index');

Route::get('/usuarios/{user}', 'UserController@show')
    ->where('user', '[0-9]+')
    ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::get('/usuarios/{user}/editar', 'UserController@edit')->name('users.edit');

Route::put('/usuarios/{user}', 'UserController@update');

Route::post('/usuarios', 'UserController@store');

Route::delete('/usuarios/{user}', 'UserController@destroy');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

Aula 37 - Agregando links ao módulo de usuários

Continuaremos com a criação do nosso módulo de usuário, desta vez, vamos trabalhar para adicionar os links HTML que faltam ao módulo, incluindo o botão para acionar a ação para eliminar os usuários que criamos na lição anterior.

Dentro de um link, podemos passar diretamente o modelo Eloquent em vez de $user->id:

```
// Passando a propriedade id:
route('users.edit', ['id' => $user->id])

// Passando diretamente o modelo de eloquent:
route('users.edit', ['id' => $user])
```

Também podemos passar diretamente o modelo Eloquent sem a correção:

```
route('users.edit', $user)
```

Isso torna nossos links mais limpos:

Enlace anterior:

```
<a href="{{ route('users.edit', ['id' => $user->id]) }}">Editar</a>
```

Enlace utilizando diretamente o modelo de Eloquent:

```
<a href="{{ route('users.edit', $user) }}">Editar</a>
```

Enviar petições PUT ou DELETE

Para enviar uma solicitação do tipo POST, PATCH, PUT ou DELETE, precisamos usar um formulário. Ao utilizar o formulário para requisições PATCH, PUT ou DELETE, o atributo method do campo form deve ser igual a POST e indicamos ao Laravel que o request é PATCH, PUT ou DELETE adicionando um campo oculto com method_field():

```
<form action="{{ route('users.destroy', $user) }}" method="POST">
   {{ csrf_field() }}
   {{ method_field('DELETE') }}
   <button type="submit">Eliminar</button>
</form>
```

---

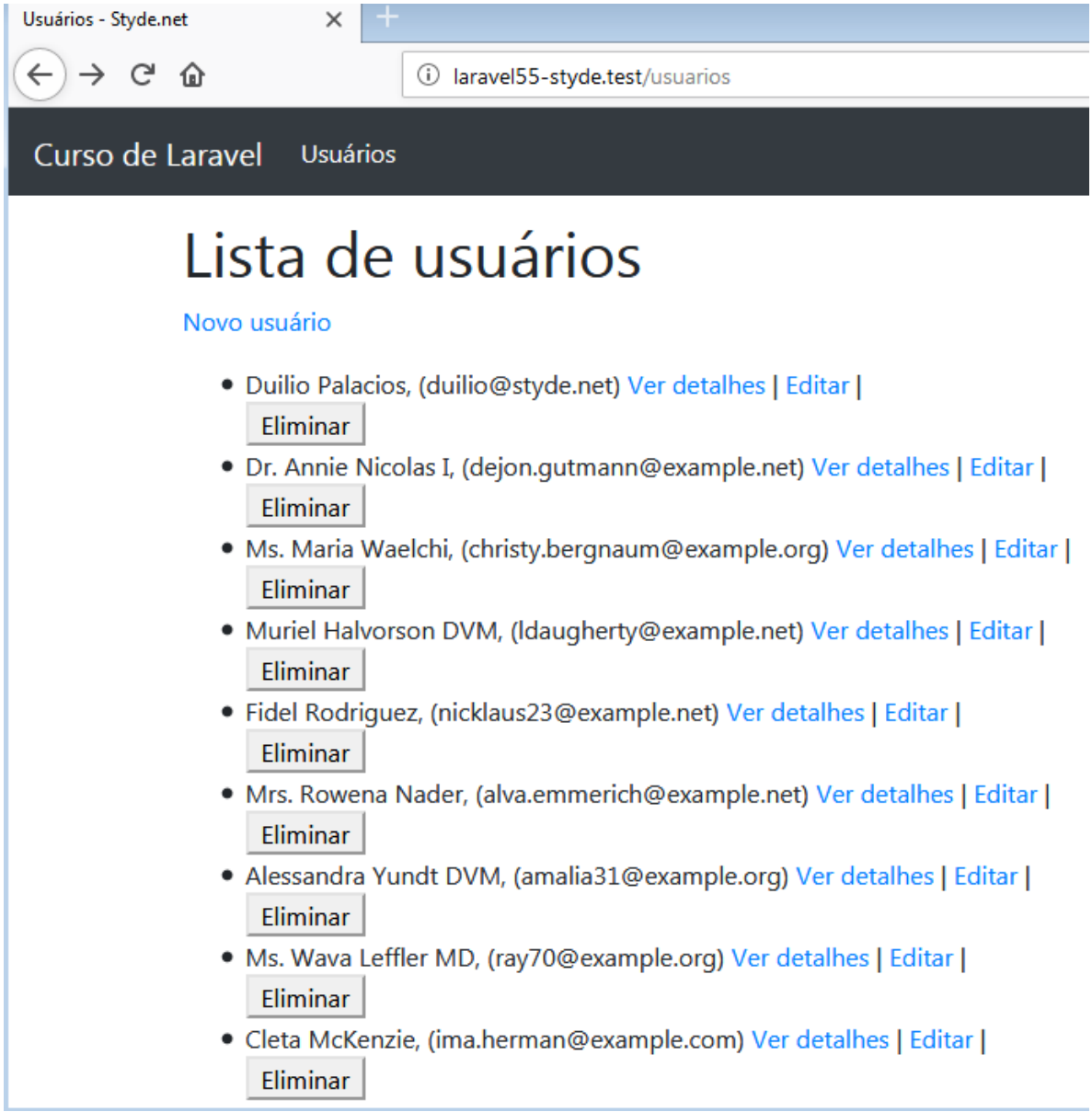
routes/web.php

```
<?php

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

Route::get('/usuarios', 'UserController@index')
   ->name('users.index');

Route::get('/usuarios/{user}', 'UserController@show')
   ->where('user', '[0-9]+')
   ->name('users.show');

Route::get('/usuarios/novo', 'UserController@create')->name('users.create');

Route::get('/usuarios/{user}/editar', 'UserController@edit')->name('users.edit');

Route::put('/usuarios/{user}', 'UserController@update');

Route::post('/usuarios', 'UserController@store');

Route::delete('/usuarios/{user}', 'UserController@destroy')->name('users.destroy');

Route::get('/saudar/{name}/{nickname?}', 'WelcomeUserController');
```

resources/views/users/index.blade.php

```
@extends('layout')

@section('title', 'Usuários')

@section('content')
  <h1>{{ $title }}</h1>

  <p>
    <a href="{{ route('users.create') }}">Novo usuário</a>
  </p>

  <ul>
    @forelse ($users as $user)
      <li>
        {{ $user->name }}, ({{ $user->email }})
        <a href="{{ route('users.show', $user) }}">Ver detalhes</a> |
        <a href="{{ route('users.edit', $user) }}">Editar</a> |
        <form action="{{ route('users.destroy', $user) }}" method="POST">
          {{ csrf_field() }}
          {{ method_field('DELETE') }}
          <button type="submit">Eliminar</button>
        </form>
      </li>
    @empty
      <li>Não há usuários registrados.</li>
    @endforelse
  </ul>@endsection

@section('sidebar')
  @parent
@endsection
```

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.......................                                           24 / 24 (100%)

Time: 9.94 seconds, Memory: 16.00MB

OK (24 tests, 81 assertions)

C:\laragon\www\laravel55-styde
λ
```


Usuários - Styde.net
laravel55-styde.test/usuarios
Curso de Laravel
Usuários
Lista de usuários
Novo usuário
Duilio Palacios, (duilio@styde.net) Ver detalhes | Editar |
Eliminar
Dr. Annie Nicolas I, (dejon.gutmann@example.net) Ver detalhes | Editar |
Eliminar
Ms. Maria Waelchi, (christy.bergnaum@example.org) Ver detalhes | Editar |
Eliminar
Muriel Halvorson DVM, (ldaugherty@example.net) Ver detalhes | Editar |
Eliminar
Fidel Rodriguez, (nicklaus23@example.net) Ver detalhes | Editar |
Eliminar
Mrs. Rowena Nader, (alva.emmerich@example.net) Ver detalhes | Editar |
Eliminar
Alessandra Yundt DVM, (amalia31@example.org) Ver detalhes | Editar |
Eliminar
Ms. Wava Leffler MD, (ray70@example.org) Ver detalhes | Editar |
Eliminar
Cleta McKenzie, (ima.herman@example.com) Ver detalhes | Editar |
Eliminar

Aula 38 - Adicionando estilo de Bootstrap 4 à listagem de usuários

Veremos como podemos adicionar estilos do Bootstrap 4 ao nosso aplicativo e como podemos usá-los para personalizar a lista de usuários. Desta forma, daremos uma melhor aparência ao módulo CRUD de usuários que desenvolvemos até agora.

Para usar o Laravel, não é obrigatório usar o Bootstrap. Você pode usar seus próprios estilos personalizados ou qualquer outra estrutura CSS, como Materialize ou Foundation.

Você pode verificar se uma coleção não está vazia usando o método isNotEmpty():

```
@if ($users->isNotEmpty())
   ...
@else
   <p>A coleção está vazia.</p>
@endif
```

Para mostrar um ícone, você pode usar a tag passando o nome do ícone como o valor da propriedade de classe:

```
<span class="oi oi-eye"></span>
```

---

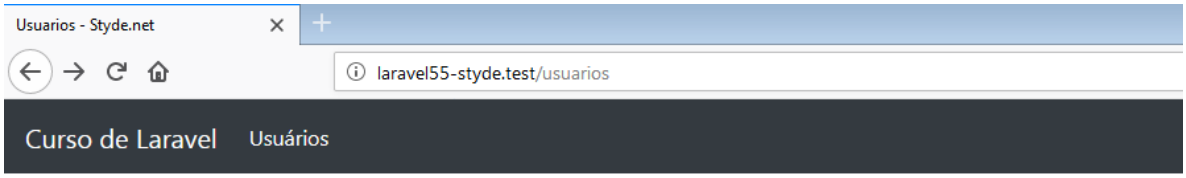
resources/views/layout.blade.php

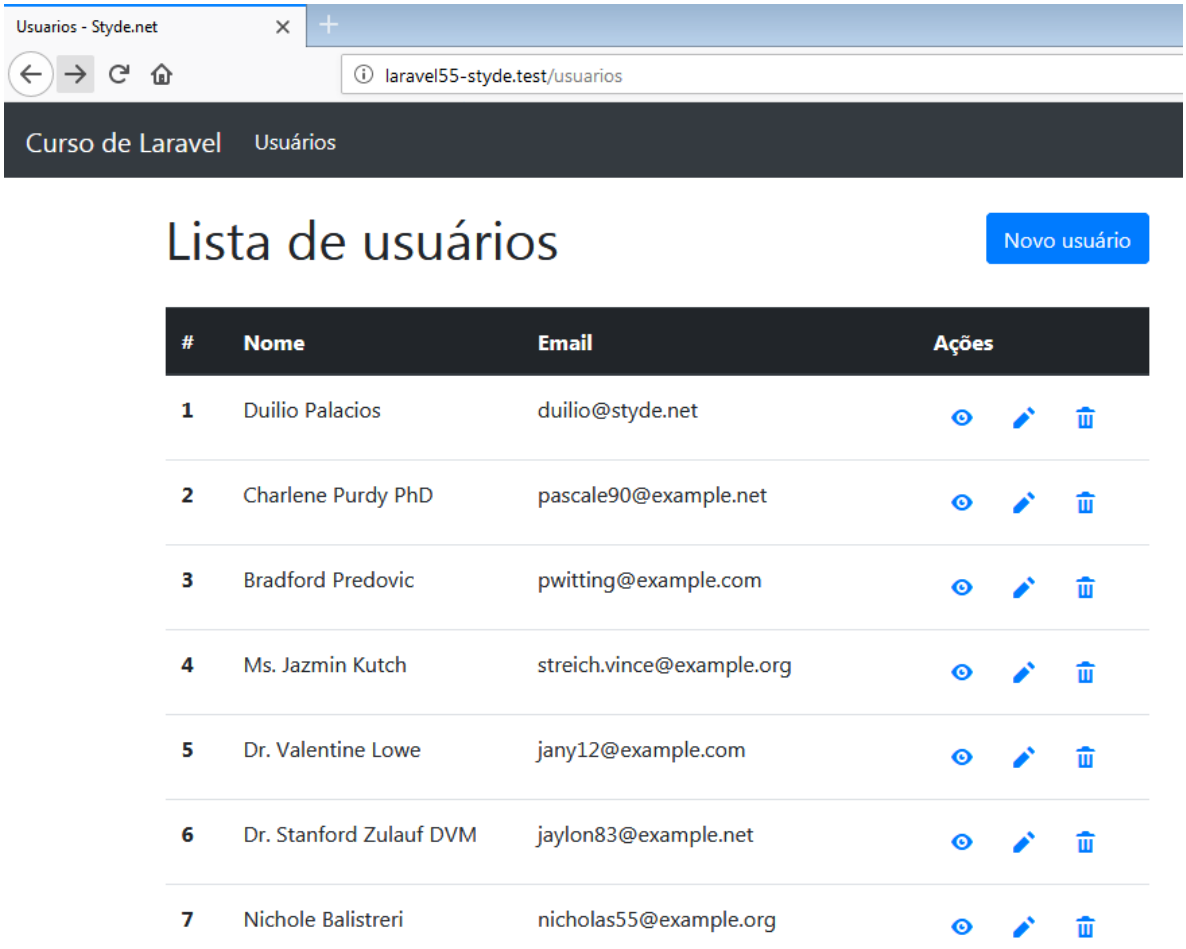
```
<!doctype html>
<html lang="pt-br">
<head>
   <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
   <meta name="description" content="">
   <meta name="author" content="">
   <link rel="icon" href="favicon.ico">

   <title>@yield('title') - Styde.net</title>

   <!-- Bootstrap core CSS -->
   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/open-iconic/1.1.1/font/css/open-iconic-
bootstrap.css" integrity="sha256-CNwnGWPO03a1kOlAsGaH5g8P3dFaqFqqGFV/1nkX5OU="
crossorigin="anonymous" />
   <!-- Custom styles for this template -->
   <link href="{{ asset('css/style.css') }}" rel="stylesheet">
</head>

<body>

<header>
   <!-- Fixed navbar -->
   <nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">
      <a class="navbar-brand" href="#">Curso de Laravel</a>
      <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarCollapse"
aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">
         <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarCollapse">
         <ul class="navbar-nav mr-auto">
            <li class="nav-item active">
               <a class="nav-link" href="{{ url('/usuarios') }}">Usuários</a>
            </li>
         </ul>
      </div>
   </nav>
</header>

<!-- Begin page content -->
<main role="main" class="container">
   <div class="row mt-3">
      <div class="col-8">
         @yield('content')
      </div>
      <div class="col-4">
         <p> </p>
      </div>
   </div>
</main>
```

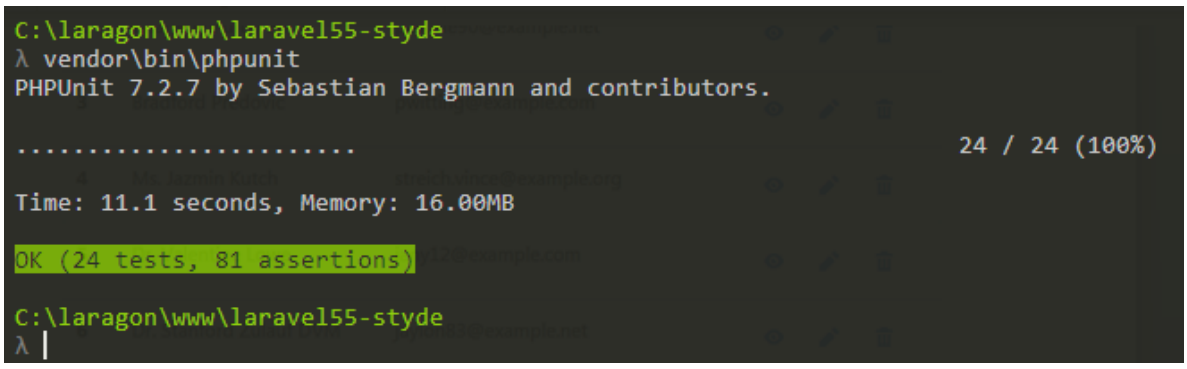
```
<footer class="footer">
  <div class="container">
    <span class="text-muted">https://styde.net</span>
  </div>
</footer>


<!-- Bootstrap core JavaScript
================================================== -->
<!-- Placed at the end of the document so the pages load faster -->
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
</body>
</html>
```

resources/views/users/index.blade.php

```
@extends('layout')

@section('title', 'Usuarios')

@section('content')
  <div class="d-flex justify-content-between align-items-end mb-3">
    <h1 class="pb-1">{{ $title }}</h1>
    <p>
      <a href="{{ route('users.create') }}" class="btn btn-primary">Novo usuário</a>
    </p>
  </div>

  @if ($users->isNotEmpty())
    <table class="table">
      <thead class="thead-dark">
      <tr>
        <th scope="col">#</th>
        <th scope="col">Nome</th>
        <th scope="col">Email</th>
        <th scope="col">Ações</th>
      </tr>
      </thead>
      <tbody>
      @foreach($users as $user)
        <tr>
          <th scope="row">{{ $user->id }}</th>
          <td>{{ $user->name }}</td>
          <td>{{ $user->email }}</td>
          <td>
            <form action="{{ route('users.destroy', $user) }}" method="POST">
              {{ csrf_field() }}
              {{ method_field('DELETE') }}
              <a href="{{ route('users.show', $user) }}" class="btn btn-link"><span class="oi oi-
eye"></span></a>
              <a href="{{ route('users.edit', $user) }}" class="btn btn-link"><span class="oi oi-
pencil"></span></a>
              <button type="submit" class="btn btn-link"><span class="oi oi-trash"></span></button>
            </form>
          </td>
        </tr>
      @endforeach
      </tbody>
    </table>
  @else
    <p>Não há usuários registrados.</p>
  @endif
@endsection

@section('sidebar')
  @parent
@endsection
```


Usuarios - Styde.net
laravel55-styde.test/usuarios
Curso de Laravel
Usuários
Lista de usuários
Novo usuário
Não há usuários registrados.


Usuarios - Styde.net
laravel55-styde.test/usuarios
Curso de Laravel
Usuários
Lista de usuários
Novo usuário
Nome Email Ações
1 Duilio Palacios duilio@styde.net
2 Charlene Purdy PhD pascale90@example.net
3 Bradford Predovic pwitting@example.com
4 Ms. Jazmin Kutch streich.vince@example.org
5 Dr. Valentine Lowe jany12@example.com
6 Dr. Stanford Zulauf DVM jaylon83@example.net
7 Nichole Balistreri nicholas55@example.org


C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.
.... 24 / 24 (100%)
Time: 11.1 seconds, Memory: 16.00MB
OK (24 tests, 81 assertions)
C:\laragon\www\laravel55-styde
λ

Aula 39 - Estilos de Bootstrap para formulários de Laravel

Esta é a lição final do Curso Laravel 5.5 do zero, no próximo vídeo continuaremos adicionando os estilos do Bootstrap ao nosso aplicativo, desta vez no formulário para criar novos usuários.

Muito obrigado por aprender conosco e parabéns por ter concluído a edição 2017-2018 do nosso Curso Laravel do zero. Para continuar aprendendo e melhorando suas habilidades de programação, inscreva-se hoje na Styde, cada pequena contribuição é muito valiosa para poder continuar com este projeto e continuar fazendo cursos como este para você e toda a comunidade em espanhol. No momento da publicação deste artigo, publicamos mais de 100 horas em vídeo (+500 aulas) em Styde e nossa comunidade em Slack tem mais de 1000 alunos. Este ano gostaríamos, com sua participação e seu apoio, de poder duplicar esses números.

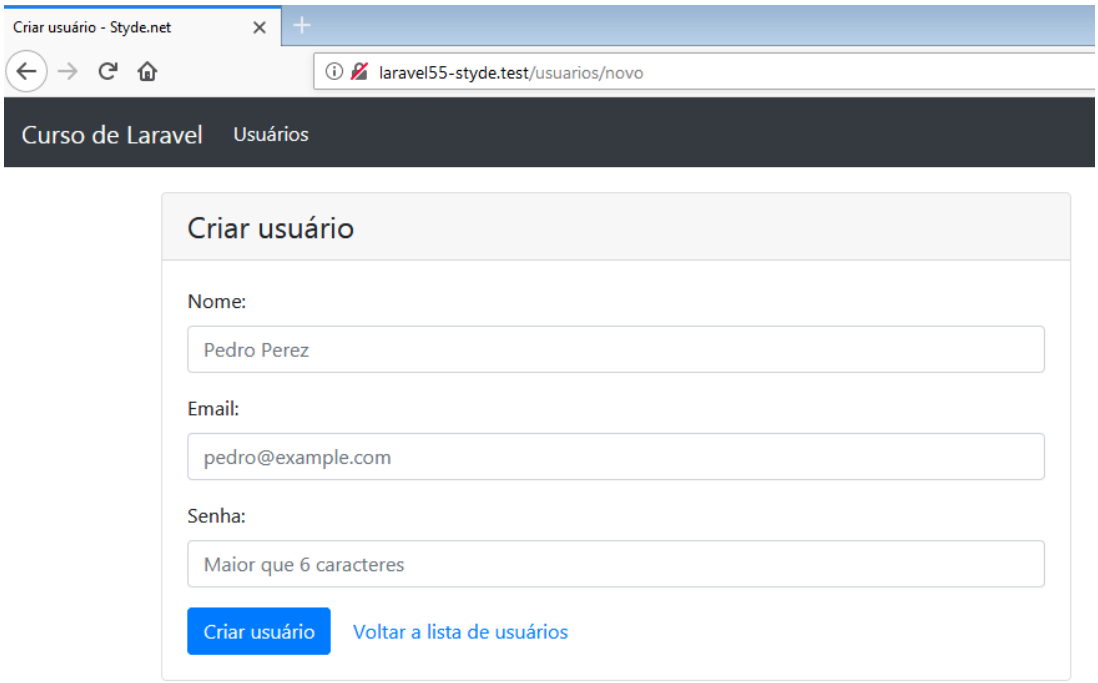
---

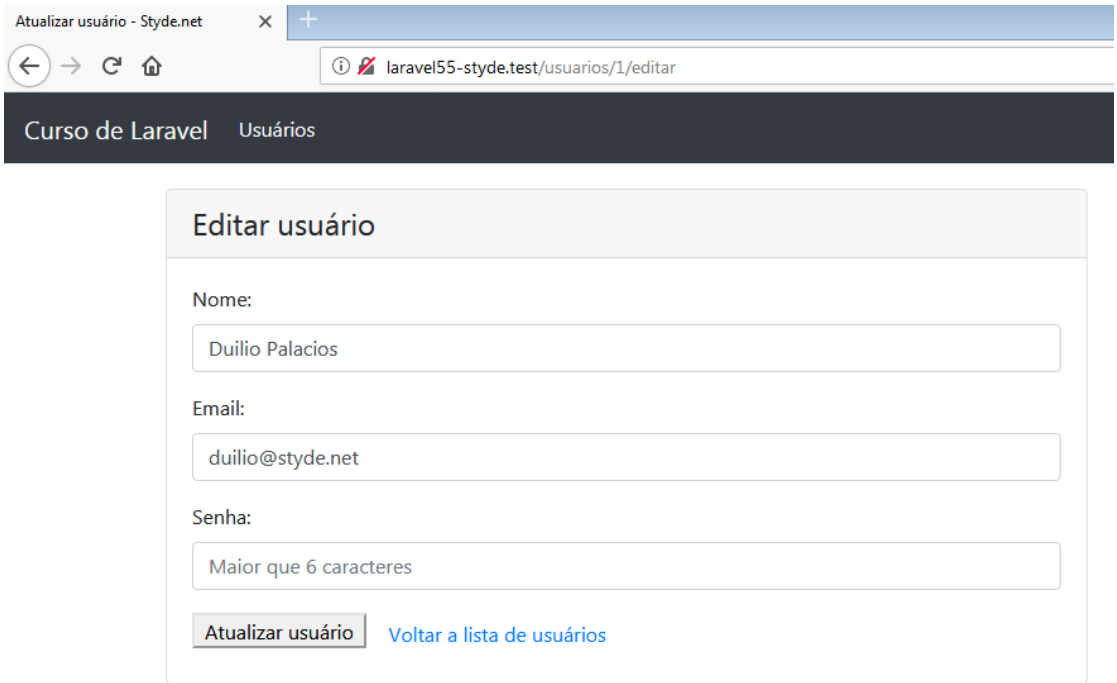
resources/views/users/create.blade.php

```
@extends('layout')

@section('title', "Criar usuário")

@section('content')
  <div class="card">
    <h4 class="card-header">Criar usuário</h4>
    <div class="card-body">

      @if ($errors->any())
        <div class="alert alert-danger">
          <h6>Por favor corrija os erros abaixo:</h6>
          <ul>
            @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
            @endforeach
          </ul>
        </div>
      @endif

      <form method="POST" action="{{ url('usuarios') }}">
        {{ csrf_field() }}

        <div class="form-group">
          <label for="name">Nome:</label>
          <input type="text" class="form-control" name="name" id="name" placeholder="Pedro Perez"
value="{{ old('name') }}">
        </div>

        <div class="form-group">
          <label for="email">Email:</label>
          <input type="email" class="form-control" name="email" id="email"
placeholder="pedro@example.com" value="{{ old('email') }}">
        </div>

        <div class="form-group">
          <label for="password">Senha:</label>
          <input type="password" class="form-control" name="password" id="password"
placeholder="Maior que 6 caracteres">
        </div>

        <button type="submit" class="btn btn-primary">Criar usuário</button>
        <a href="{{ route('users.index') }}" class="btn btn-link">Voltar a lista de usuários</a>
      </form>
    </div>
  </div>
@endsection
```

resources/views/users/edit.blade.php

```
@extends('layout')

@section('title', "Atualizar usuário")

@section('content')
  <div class="card">
    <h4 class="card-header">Editar usuário</h4>
    <div class="card-body">

      @if ($errors->any())
        <div class="alert alert-danger">
          <h6>Por favor corrija os erros abaixo:</h6>
          <ul>
            @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
            @endforeach
          </ul>
        </div>
      @endif

      <form method="POST" action="{{ url("usuarios/{$user->id}") }}">
        {{ method_field('PUT') }}
        {{ csrf_field() }}

        <div class="form-group">
          <label for="name">Nome:</label>
          <input type="text" class="form-control" name="name" id="name" placeholder="Pedro Perez"
value="{{ old('name', $user->name) }}">
        </div>

        <div class="form-group">
          <label for="email">Email:</label>
          <input type="email" class="form-control" name="email" id="email"
placeholder="pedro@example.com" value="{{ old('email', $user->email) }}">
        </div>

        <div class="form-group">
          <label for="password">Senha:</label>
          <input type="password" class="form-control" name="password" id="password"
placeholder="Maior que 6 caracteres">
        </div>

        <button type="submit">Atualizar usuário</button>

        <a href="{{ route('users.index') }}" class="btn btn-link">Voltar a lista de usuários</a>

      </form>

    </div>
  </div>
@endsection
```






```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.......................                                           24 / 24 (100%)

Time: 9.45 seconds, Memory: 16.00MB

OK (24 tests, 81 assertions)

C:\laragon\www\laravel55-styde
λ
```

Aula 40 - Gerando relatórios e documentação com PHPUnit

Nesta lição, explicarei um recurso do PHPUnit que nos permite gerar documentação e/ou relatar o progresso dos recursos de nossos aplicativos apenas executando os testes

Gerar relatórios em HTML com PHPUnit

Você pode gerar o relatório com o seguinte comando:

```
vendor/bin/phpunit --testdox-html diretório-aqui/nome-do-arquivo-aqui.html
```

Você pode colocar o relatório no diretório de sua preferência e depois abri-lo no navegador.

Gerar relatórios em XML com PHPUnit

Você pode gerar o relatório com o seguinte comando:

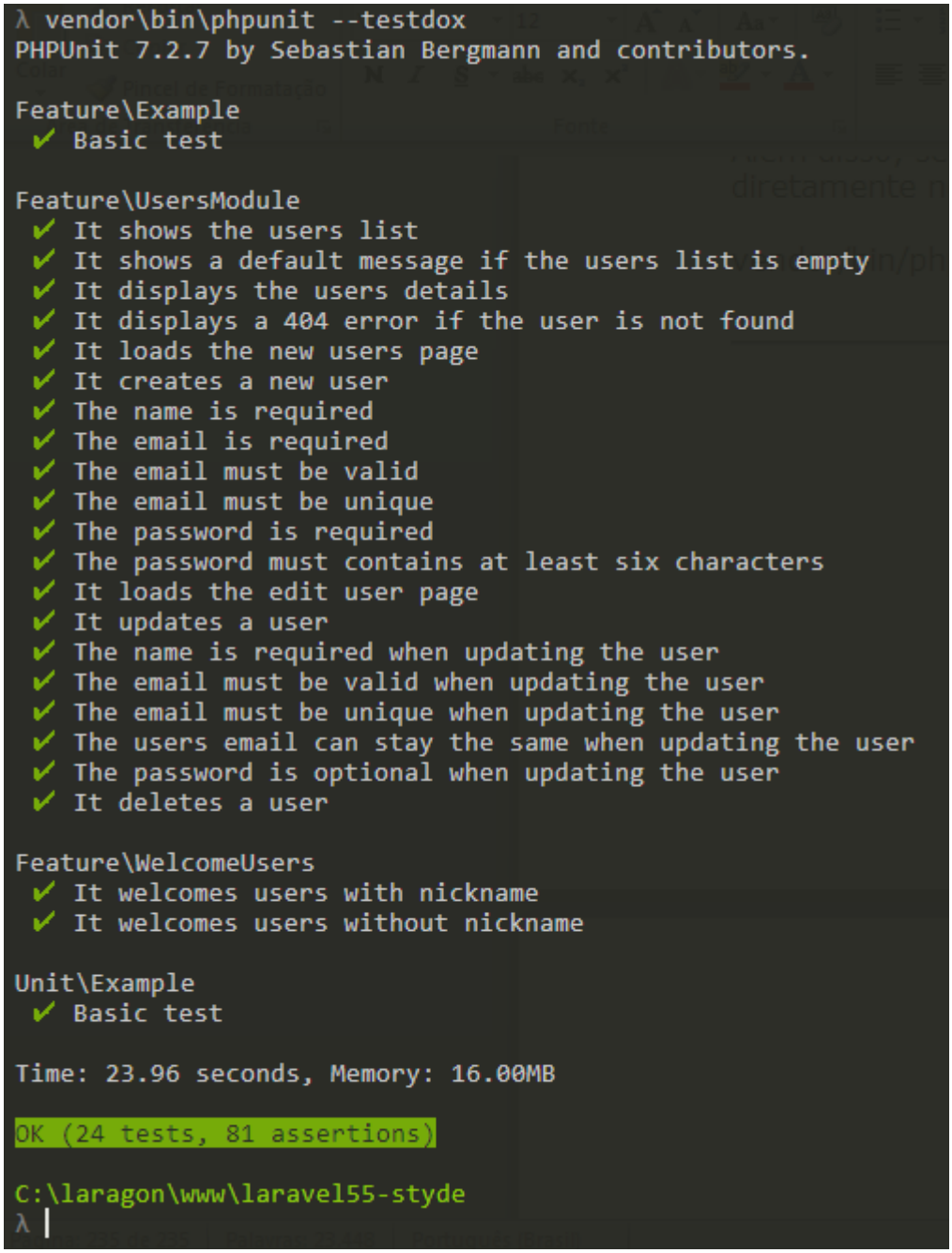
```
vendor/bin/phpunit --testdox-xml diretório-aqui/nome-do-arquivo-aqui.xml
```

Esse XML pode ser usado para gerar um relatório mais atraente em HTML ou em qualquer outro formato de sua preferência.

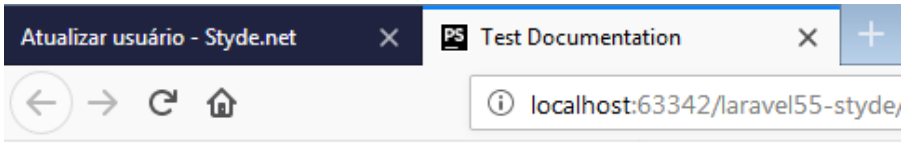
Visualizar relatório no console

Além disso, se você não precisar gerar um arquivo, poderá ver o resultado diretamente no console executando o seguinte:

```
vendor/bin/phpunit --testdox
```


λ vendor\bin\phpunit --testdox
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.
Feature\Example
✔ Basic test
Feature\UsersModule
✔ It shows the users list
✔ It shows a default message if the users list is empty
✔ It displays the users details
✔ It displays a 404 error if the user is not found
✔ It loads the new users page
✔ It creates a new user
✔ The name is required
✔ The email is required
✔ The email must be valid
✔ The email must be unique
✔ The password is required
✔ The password must contains at least six characters
✔ It loads the edit user page
✔ It updates a user
✔ The name is required when updating the user
✔ The email must be valid when updating the user
✔ The email must be unique when updating the user
✔ The users email can stay the same when updating the user
✔ The password is optional when updating the user
✔ It deletes a user
Feature\WelcomeUsers
✔ It welcomes users with nickname
✔ It welcomes users without nickname
Unit\Example
✔ Basic test
Time: 23.96 seconds, Memory: 16.00MB
OK (24 tests, 81 assertions)
C:\laragon\www\laravel55-styde
λ

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit --testdox-html reports/testes.html
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

.......................                                          24 / 24 (100%)

Time: 9.49 seconds, Memory: 16.00MB

OK (24 tests, 81 assertions)

C:\laragon\www\laravel55-styde
λ
```



Feature\Example

√ Basic test

Feature\UsersModule

√ It shows the users list
√ It shows a default message if the users list is empty
√ It displays the users details
√ It displays a 404 error if the user is not found
√ It loads the new users page
√ It creates a new user
√ The name is required
√ The email is required
√ The email must be valid
√ The email must be unique
√ The password is required
√ The password must contains at least six characters
√ It loads the edit user page
√ It updates a user
√ The name is required when updating the user
√ The email must be valid when updating the user
√ The email must be unique when updating the user
√ The users email can stay the same when updating the user
√ The password is optional when updating the user
√ It deletes a user

Feature\WelcomeUsers

```
C:\laragon\www\laravel55-styde
λ vendor\bin\phpunit --testdox-xml reports/testes.xml
PHPUnit 7.2.7 by Sebastian Bergmann and contributors.

........................                                          24 / 24 (100%)

Time: 10.51 seconds, Memory: 16.00MB

OK (24 tests, 81 assertions)

C:\laragon\www\laravel55-styde
λ
```

```
<?xml version="1.0" encoding="UTF-8"?>
- <tests>
    <test groups="default" size="-1" time="0.30701684951782" status="0" prettifiedMethodName="Basic test" prettifiedClassName="Feature\Example"
      methodName="testBasicTest" className="Tests\Feature\ExampleTest"/>
    <test groups="default" size="-1" time="6.0803468227386" status="0" prettifiedMethodName="It shows the users list" prettifiedClassName="Feature\UsersModule"
      methodName="it_shows_the_users_list" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.13700795173645" status="0" prettifiedMethodName="It shows a default message if the users list is empty"
      prettifiedClassName="Feature\UsersModule" methodName="it_shows_a_default_message_if_the_users_list_is_empty" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.1360080242157" status="0" prettifiedMethodName="It displays the users details" prettifiedClassName="Feature\UsersModule"
      methodName="it_displays_the_users_details" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.21701312065125" status="0" prettifiedMethodName="It displays a 404 error if the user is not found"
      prettifiedClassName="Feature\UsersModule" methodName="it_displays_a_404_error_if_the_user_is_not_found" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.099004983901978" status="0" prettifiedMethodName="It loads the new users page" prettifiedClassName="Feature\UsersModule"
      methodName="it_loads_the_new_users_page" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.19601202011108" status="0" prettifiedMethodName="It creates a new user" prettifiedClassName="Feature\UsersModule"
      methodName="it_creates_a_new_user" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.1360080242157" status="0" prettifiedMethodName="The name is required" prettifiedClassName="Feature\UsersModule"
      methodName="the_name_is_required" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.13600707054138" status="0" prettifiedMethodName="The email is required" prettifiedClassName="Feature\UsersModule"
      methodName="the_email_is_required" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.11900687217712" status="0" prettifiedMethodName="The email must be valid" prettifiedClassName="Feature\UsersModule"
      methodName="the_email_must_be_valid" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.15400910377502" status="0" prettifiedMethodName="The email must be unique" prettifiedClassName="Feature\UsersModule"
      methodName="the_email_must_be_unique" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.18901085853577" status="0" prettifiedMethodName="The password is required" prettifiedClassName="Feature\UsersModule"
      methodName="the_password_is_required" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.1620090007782" status="0" prettifiedMethodName="The password must contains at least six characters"
      prettifiedClassName="Feature\UsersModule" methodName="the_password_must_contains_at_least_six_characters" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.19501090049744" status="0" prettifiedMethodName="It loads the edit user page" prettifiedClassName="Feature\UsersModule"
      methodName="it_loads_the_edit_user_page" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.1620090007782" status="0" prettifiedMethodName="It updates a user" prettifiedClassName="Feature\UsersModule"
      methodName="it_updates_a_user" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.20601201057434" status="0" prettifiedMethodName="The name is required when updating the user"
      prettifiedClassName="Feature\UsersModule" methodName="the_name_is_required_when_updating_the_user" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.36102104187012" status="0" prettifiedMethodName="The email must be valid when updating the user"
      prettifiedClassName="Feature\UsersModule" methodName="the_email_must_be_valid_when_updating_the_user" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.23201298713684" status="0" prettifiedMethodName="The email must be unique when updating the user"
      prettifiedClassName="Feature\UsersModule" methodName="the_email_must_be_unique_when_updating_the_user" className="Tests\Feature\UsersModuleTest"/>
    <test groups="default" size="-1" time="0.46202707290649" status="0" prettifiedMethodName="The users email can stay the same when updating the user"
```