**Aula 4 – CRUD-Produto em Laravel**

**Implementando o Cadastro de Produto**

**Objetivos:**

* Conhecer aspectos da programação back-end da construção de uma aplicação web com Laravel;
* Implementar o Cadastro de Produtos da Aplicação.

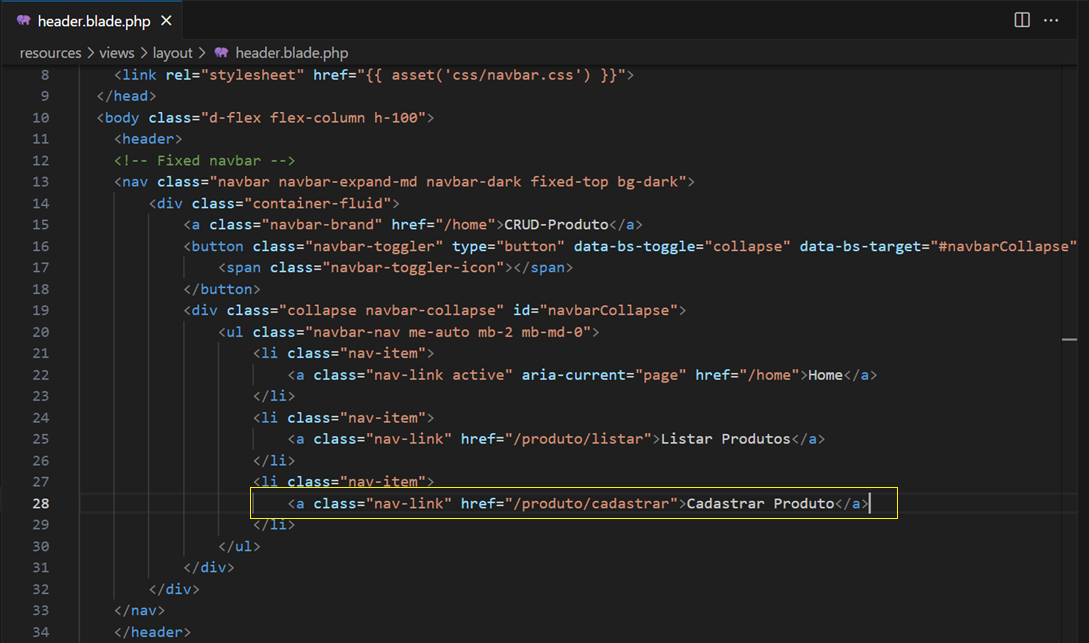
Ao final desta Aula devemos ter já em funcionamento o acesso ao Banco de Dados MySQL para a operação de Cadastro de Produto(Create-Insert) para a nossa Aplicação.

1. **Implementando a função Cadastrar Produto**

Os passos para implementar o Cadastro de Produto podem ser resumidos em:

1. Ativar o Link no menu navbar que carregará a página do Formulário de Entrada de dados;
2. Implementar o Formulário de Entrada de Dados;
3. Receber os dados do Formulário e validá-los;
4. Conectar com o Banco e salvar os dados na Tabela Produtos;
5. Mostrar uma mensagem de Feedback para o usuário.
   1. **Ativar o link para a página do Formulário no menu navbar.**

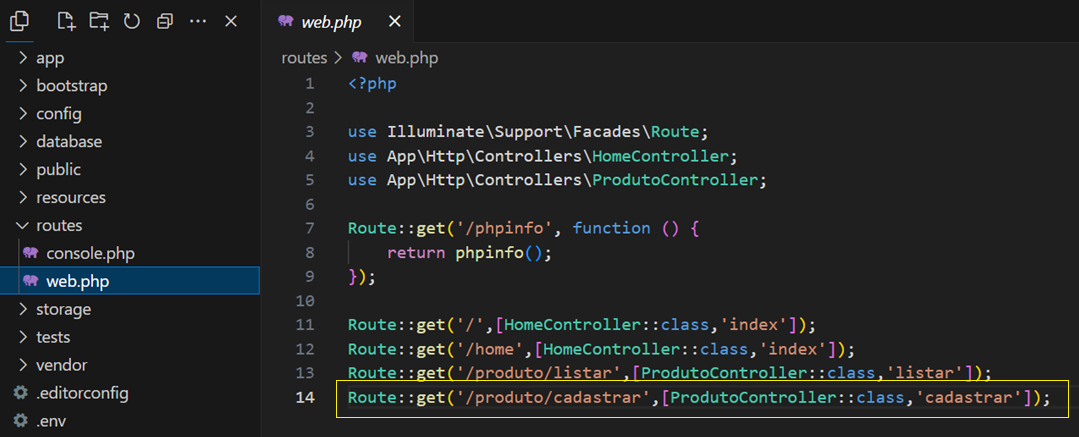
Edite o arquivo resources\views\layout\header.blade.php e acrescente o link para chamar o método cadastrar() do ProdutoController.



* 1. **Criar a Rota para o Controller e implementar o método cadastrar()**

Agora é preciso definir a rota no arquivo routes\web.php, acrescentando a seguinte linha:

Route::get('/produto/cadastrar',[ProdutoController::class,'cadastrar']);



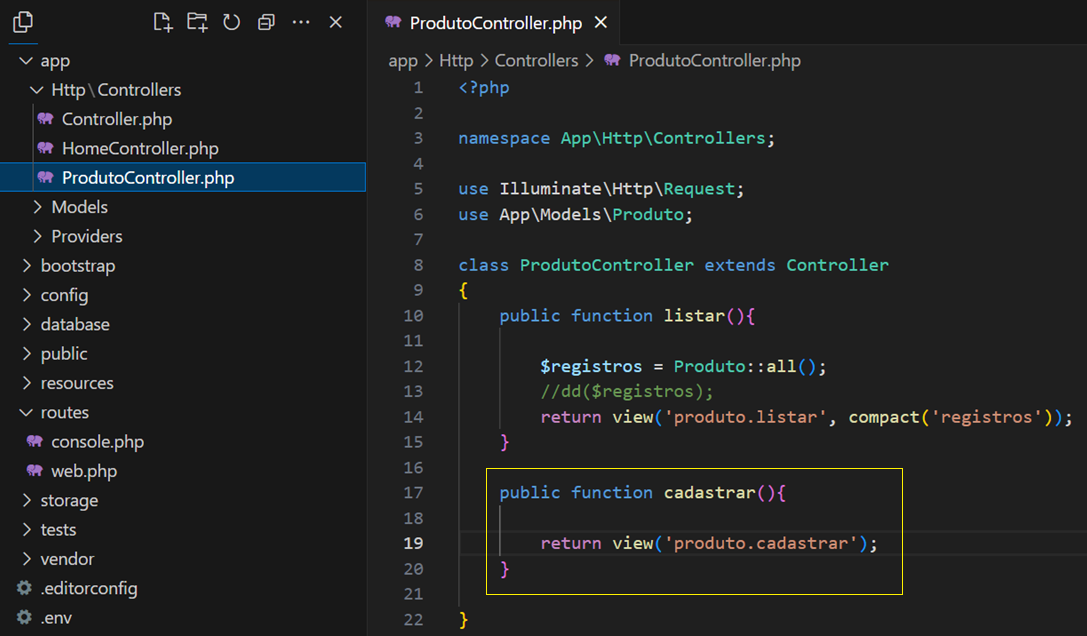
Acrescente o método cadastrar() no ProdutoController.php o qual simplesmente chama a view que monta o formulário de entrada de dados do novo Produto:

\app\Http\Controllers\ProdutoController.php

public function cadastrar(){

return view('produto.cadastrar');

}



* 1. **Criar a view com o formulário de cadastro do produto**

Agora vamos criar a view que vai mostrar o Formulário de cadastro de produto. Crie o arquivo cadastrar.blade.php em app\resources\views\produto com o seguinte código:

app\resources\views\produto\cadastrar.blade.php

@extends('layout.site')

@section('main')

<div class="container mt-4">

<h1 class="text-center">Cadastrar Produto</h2>

<div class="row">

<div class="col-md-8 offset-md-2">

@if ($errors->any())

<div class="alert alert-danger">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<form action="/produto/salvar" method="post">

@csrf

<div class="form-group">

<label for="nome">Nome do Produto</label>

<input type="text" class="form-control" name="nome" value="{{ old('nome') }}" required>

</div>

<div class="form-group">

<label for="descricao">Descrição</label>

< input type="textarea" class="form-control" name="descricao" value="{{ old('descricao') }}" required></textarea>

</div>

<div class="form-group">

<label for="preco">Preço</label>

R$ <input type="text" class="form-control" name="preco" value="{{ old('preco') }}">

</div>

<div class="form-group">

<label for="quantidade">Quantidade</label>

<input type="number" class="form-control" name="quantidade" value="{{ old('quantidade') }}" required>

</div>

<button type="submit" class="btn btn-success btn-sm mt-2">Salvar</button>

</form>

</div>

</div>

</div>

@endsection

Texto

Descrição gerada automaticamente

O código anterior implementa o Formulário de entrada de dados do produto com as classes do Bootstrap. No site do bootstrap, em Docs, procure por forms. Podemos tomar como base os formulários dos exemplos apresentados, todos usando elementos da classe “form group” do Bootstrap.

O trecho de código:

Texto

Descrição gerada automaticamente

Utiliza a diretiva @if ...@endif do Laravel para exibir, caso existam, mensagens de tratamento de erros de validação, cujo processo será explicado mais adiante.

Dentro da tag <form> note o uso da diretiva **@csrf** do Laravel:

Texto

Descrição gerada automaticamente

é um mecanismo do Laravel para proteção contra ataques Cross-site Request Forgeries. A diretiva @csrf gera automaticamente um token CSRF para cada sessão do usuário e é usado para verificar se o usuário autenticado é a pessoa que realmente fez a solicitação à Aplicação (<https://laravel.com/docs/11.x/csrf>). Sempre que definir um formulário HTML "POST", "PUT", "PATCH" ou "DELETE" no aplicativo, deve-se incluir um campo token CSRF oculto no formulário para que o middleware de proteção CSRF possa validar a solicitação. A diretiva @csrf do Blade gera esse campo de entrada de token oculto, por exemplo, como mostrado na documentação:

Texto

Descrição gerada automaticamente

Salve todos os arquivos e teste. Já será possível ver o Formulário.

Interface gráfica do usuário

Descrição gerada automaticamente

* 1. **Configurar a rota e receber os dados do Formulário**

Edite o arquivo routes\web.php, acrescentando a seguinte linha:

Route::post('/produto/salvar',[ProdutoController::class,'salvar']);

Texto

Descrição gerada automaticamente

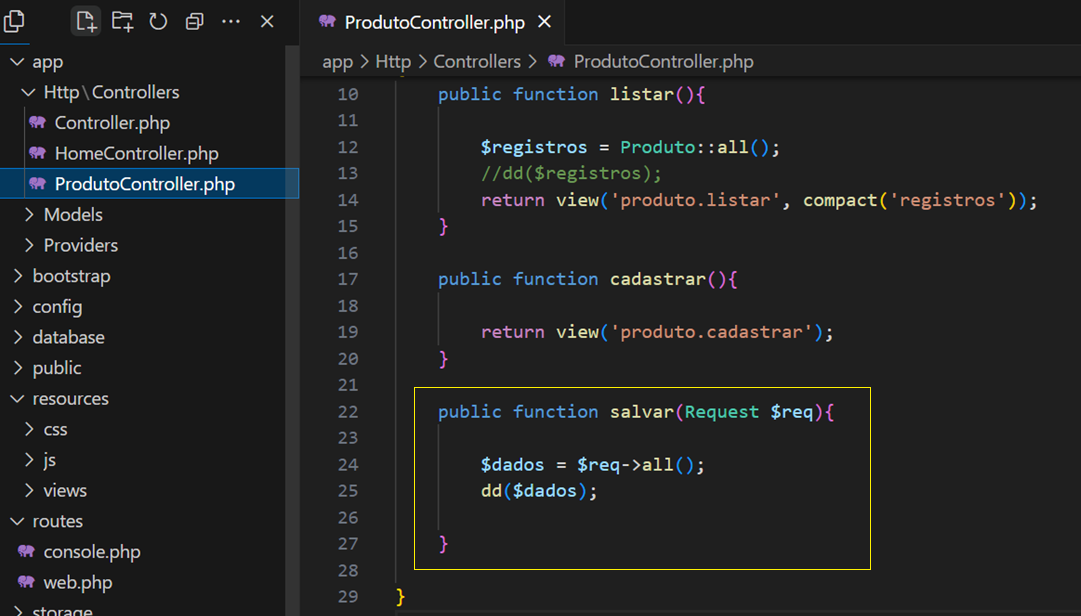
Como especificamos no Form action, o método que vai receber os dados do Formulário e gravá-los no banco é salvar. Assim, acrescente o método salvar() na classe ProdutoController, arquivo app\Http\Controlles\ProdutoController.php com o seguinte código:

public function salvar(Request $req){

$dados = $req->all();

dd($dados);

}



Note que neste código simplesmente estamos recebendo os dados e exibindo em uma página, isso para verificar o funcionamento do envio dos dados. Se tudo tiver correto, implementamos as demais etapas. Salve e teste.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

* 1. **Validando os dados**

Validação dos dados é verificar se os dados seguem regras (ou critérios) pré-estabelecidas que, em geral, estão relacionados com as regras de negócio da Aplicação. Por exemplo, um cadastro de usuário que solicita a data de nascimento para validar se o usuário está na faixa etária exigida. Outro exemplo é a validação de números de documentos, como por exemplo o CPF. Assim, cada campo pode ser validado de acordo com regras de negócio específicas. Para exemplificar, vamos supor que em nossa Aplicação o campo preço do produto não pode ser vazio. Assim, vamos aplicar uma validação de campo vazio (note que isto poderia ser feito pelo atributo required do html no Form, o que seria mais recomendado).

O Laravel oferece vários mecanismos de validação de dados com várias regras de validação prontas. O mais comum é utilizar o método validate() oferecido pelo objeto Request.

Troque o código do método salvar() na classe ProdutoController, arquivo app\Http\Controlles\ProdutoController.php, pelo seguinte código:

public function salvar(Request $req){

$valida = $req->validate([

'nome' => 'required|max:200',

'descricao' => 'required',

'preco' => 'required|decimal:2',

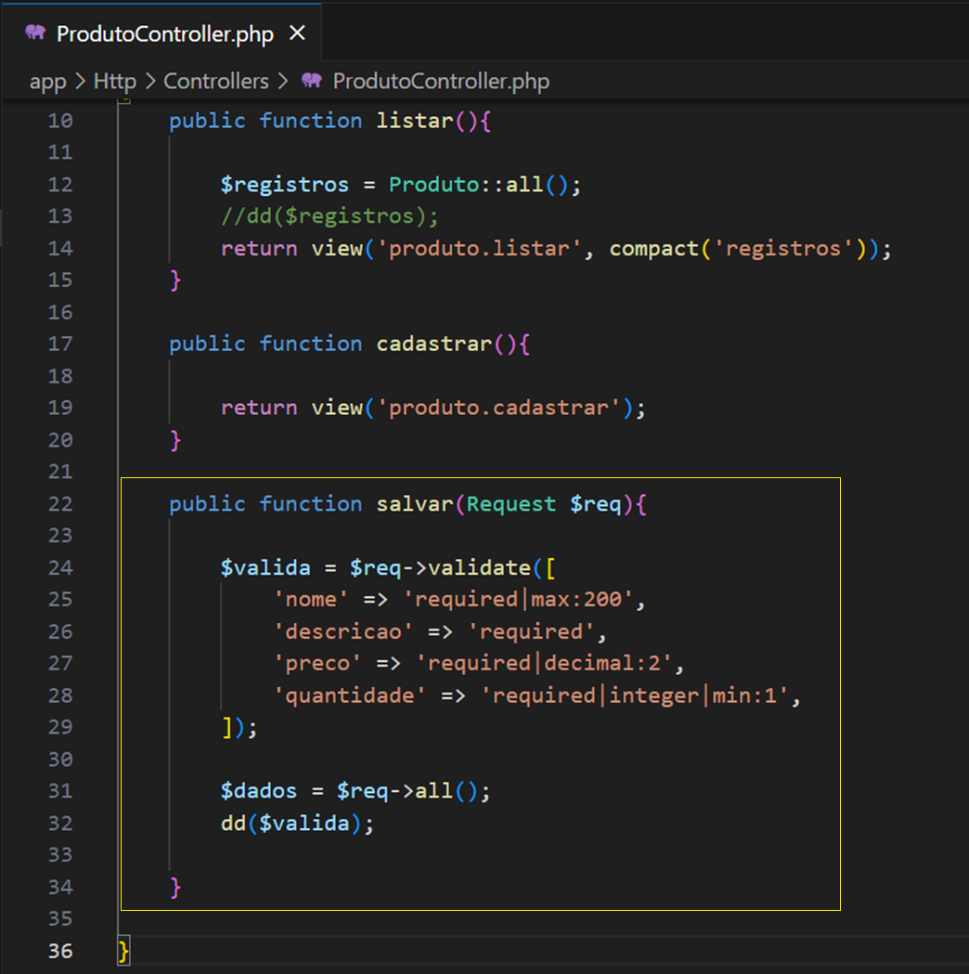
'quantidade' => 'required|integer|min:1',

]);

$dados = $req->all();

dd($valida);

}



**Explicando o Código:**

Texto

Descrição gerada automaticamente

Passamos para o método validate() o nome de cada campo que será recebido e as regras de validação já pré-estabelecidas pelo Laravel. Neste código o campo nome será validado por duas regras:

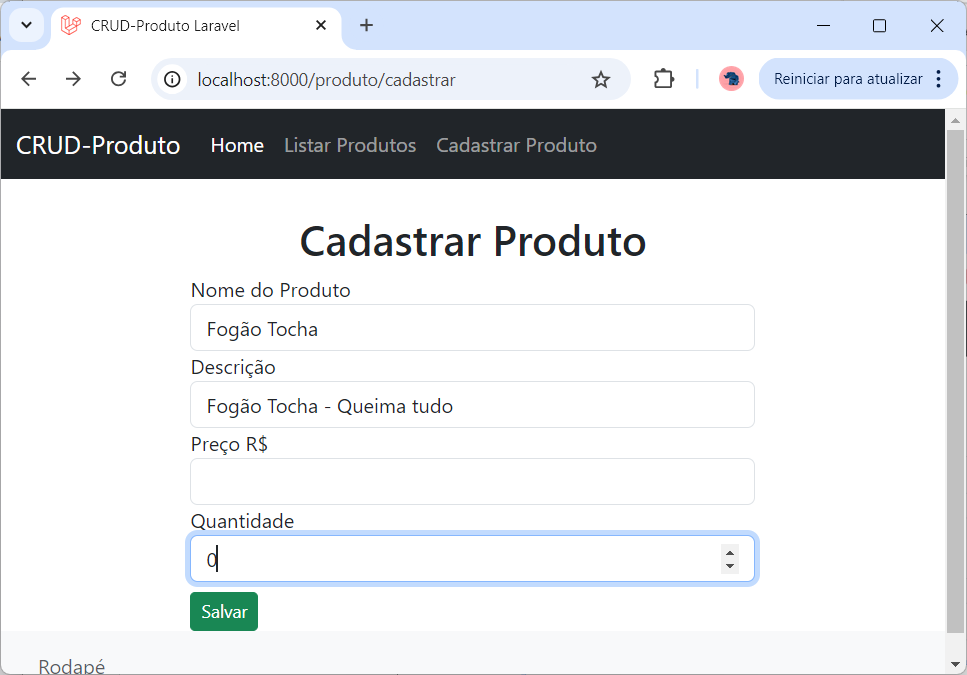
* required: o qual determina que este campo não pode ser vazio;
* max: o qual determina o tamanho máximo deste campo.

No caso do campo preco, além da regra required, está sendo aplicada a regra de que o campo deve ser digitado com duas casas decimais.

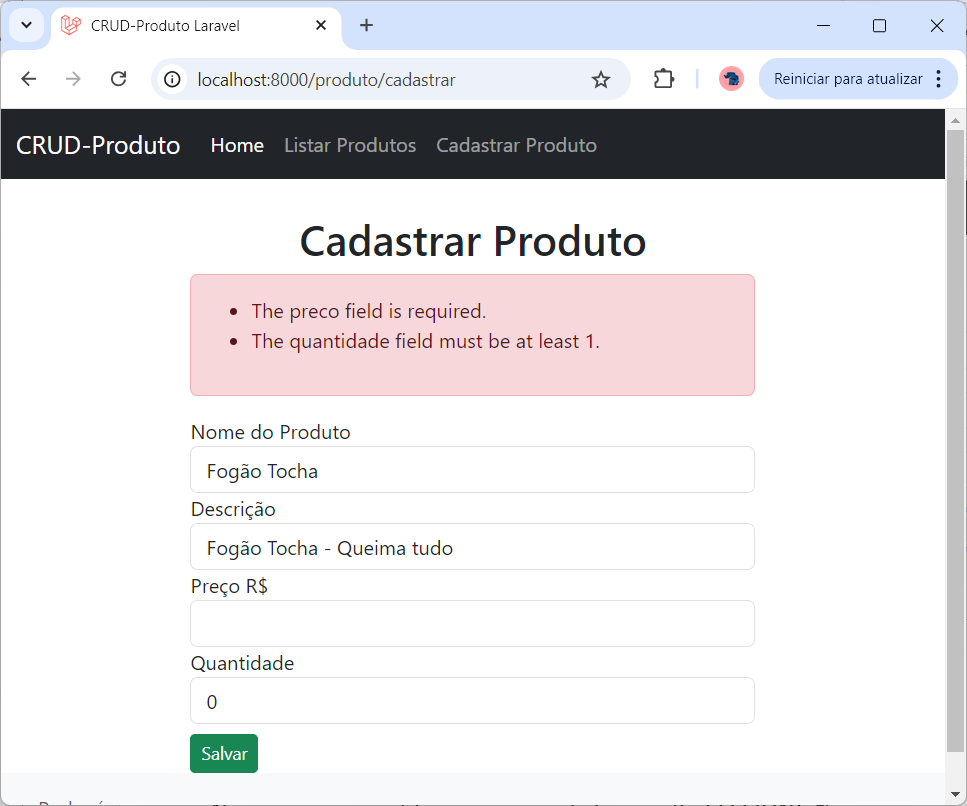
Já para o campo quantidade, não pode ser vazio (required), tem que ser um número inteiro (integer) e o valor deve ser maior que zero (min:1).

Todas as regras pré-estabelecidas do Laravel estão documentadas em https://laravel.com/docs/11.x/validation#available-validation-rules

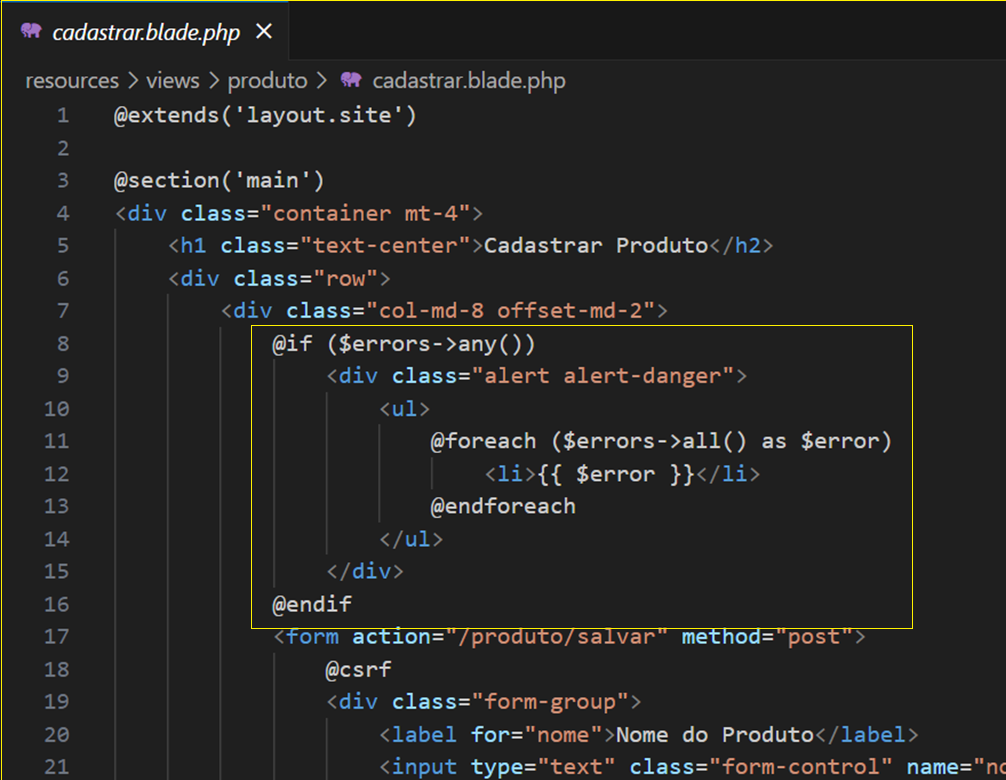
Para testar, no cadastro de um produto, deixe o campo preço vazio e coloque o valor zero para a quantidade.



Após enviar o formulário, este será reapresentado com as mensagens de erro de validação:

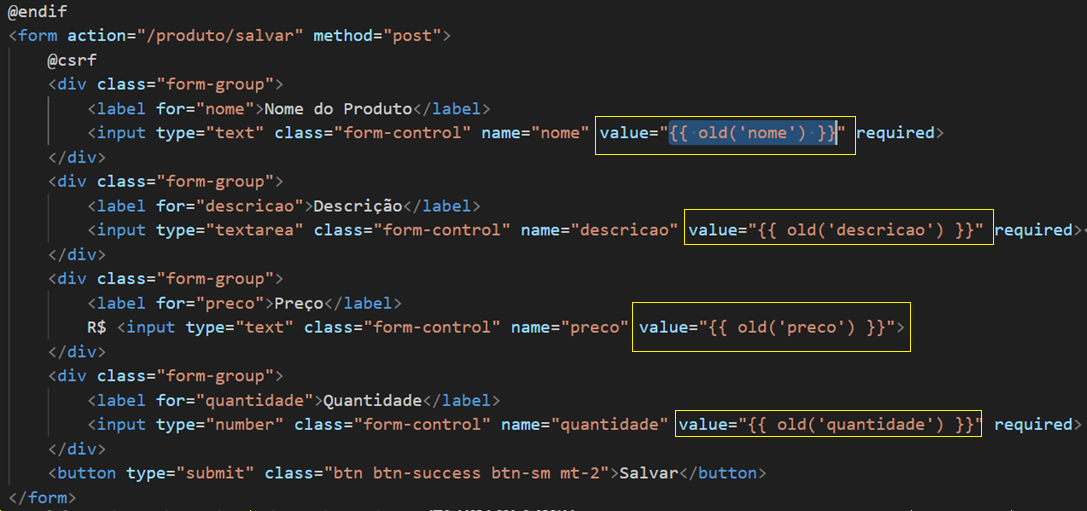


O método validate() aplica as regras de validação nos dados do formulário e, caso alguma regra não seja satisfeita, será lançada uma exceção e o Laravel redirecionará automaticamente o usuário de volta ao seu local anterior (o formulário de cadastro). Cada regra não satisfeita gera uma mensagem de erro. Assim, é possível preparar o script do formulário de cadastro para reapresentar o formulário e as mensagens de erro. Isso foi feito no código da view cadastrar.blade.php:



Note na figura anterior que o objeto $errors está sendo testado e, caso tenha algum erro, serão exibidas as mensagens de erro.

Também é preciso preencher o formulário sendo reapresentado com os dados já digitados anteriormente. Isso é feito utilizando-se o método old() no script que monta o formulário (em nosso exemplo cadastrar.blade.php), como na figura a seguir



O único inconveniente nesse mecanismo de validação automática é que as mensagens de erro são padrões e estão em inglês.

Para personalizar as mensagens de erro de validação, altere o código do método salvar() no ProdutoController para o seguinte:

public function salvar(Request $req){

// Validação com mensagens personalizadas

$req->validate([

'nome' => 'required|max:200',

'descricao' => 'required',

'preco' => 'required|decimal:2',

'quantidade' => 'required|integer|min:1',

], [

'nome.\*' => 'O campo Nome é obrigatório e deve ter no máximo 200 caracteres!',

'descricao.required' => 'O campo Descrição é obrigatório!',

'preco.\*' => 'O campo Preço é obrigatório e deve ter duas casas decimais!',

'quantidade.\*' => 'O campo Quantidade é obrigatório e deve ser maior que zero!'

]

);

$dados = $req->all();

dd($dados);

}

Texto

Descrição gerada automaticamente

Salve e teste novamente cadastrando um produto sem colocar o preço e com a quantidade igual a zero.

Interface gráfica do usuário, Texto, Email

Descrição gerada automaticamente

* 1. **Salvando o Produto no Banco de Dados**

Para salvar os dados do Produto no Banco vamos utilizar o método create() do Eloquent do Laravel. Troque o código do método salvar() do ProdutoController por:

public function salvar(Request $req){

// Validação com mensagens personalizadas

$req->validate([

'nome' => 'required|max:200',

'descricao' => 'required',

'preco' => 'required|decimal:2',

'quantidade' => 'required|integer|min:1',

], [

'nome.\*' => 'O campo Nome é obrigatório e deve ter no máximo 200 caracteres!',

'descricao.required' => 'O campo Descrição é obrigatório!',

'preco.\*' => 'O campo Preço é obrigatório e deve ter duas casas decimais!',

'quantidade.\*' => 'O campo Quantidade é obrigatório e deve ser maior que zero!'

]

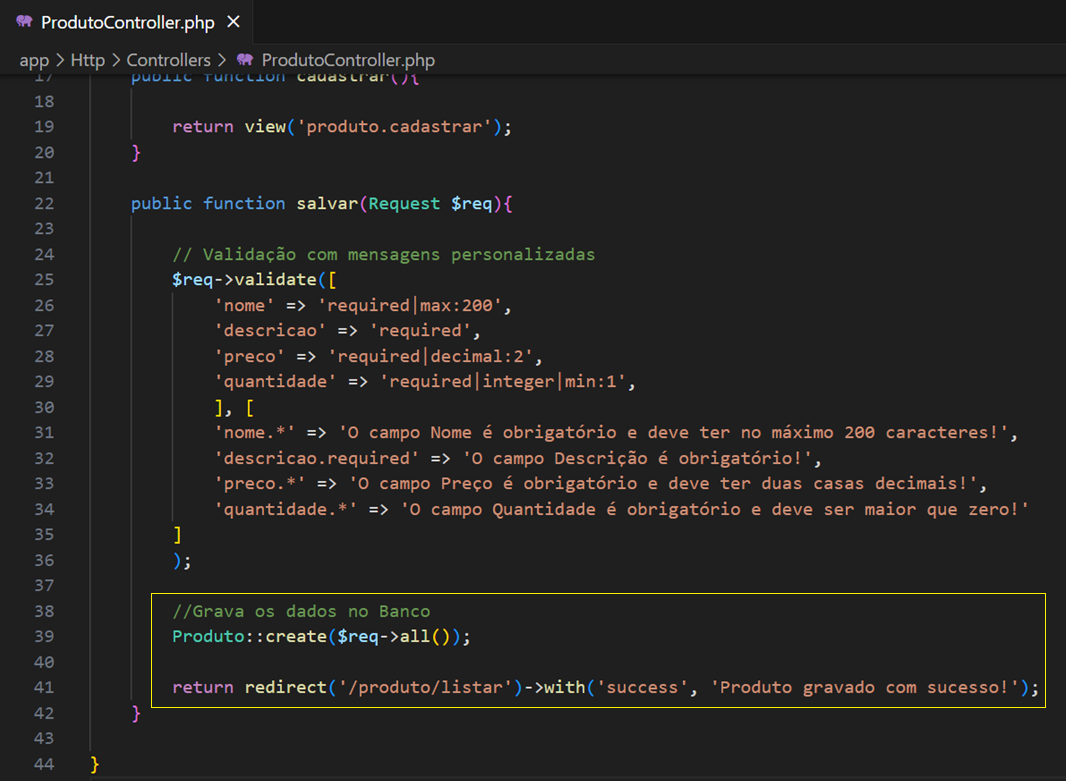
);

//Grava os dados no Banco

Produto::create($req->all());

return redirect('/produto/listar')->with('success', 'Produto gravado com sucesso!');

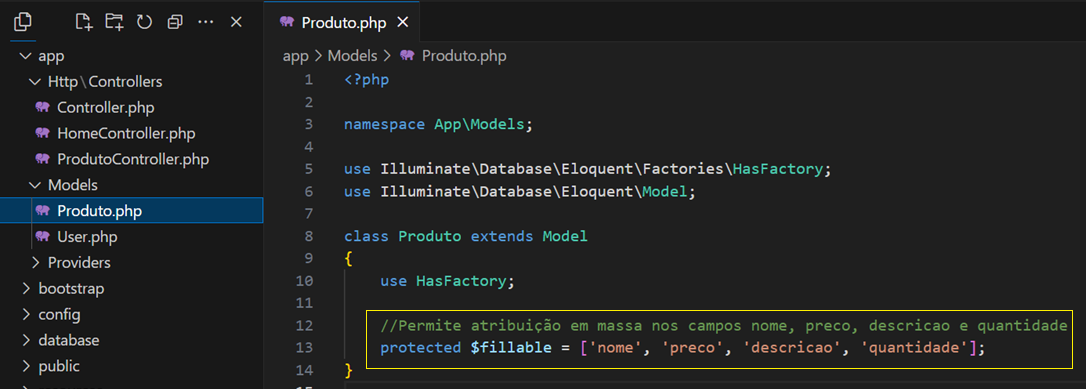
}



Para que seja possível gravar os dados em massa, é preciso permitir isso no modelo do objeto Produto. Assim, acrescente a seguinte linha no arquivo app\Models\Produto.php:

//Permite atribuição em massa nos campos nome, preco, descricao e quantidade

protected $fillable = ['nome', 'preco', 'descricao', 'quantidade'];



Para mostrar a mensagem de sucesso no Listar, acrescente o seguinte código na view listar.blade.php em resources\views\produto

@if (session('success'))

<div class="alert alert-success">

{{ session('success') }}

</div>

@endif

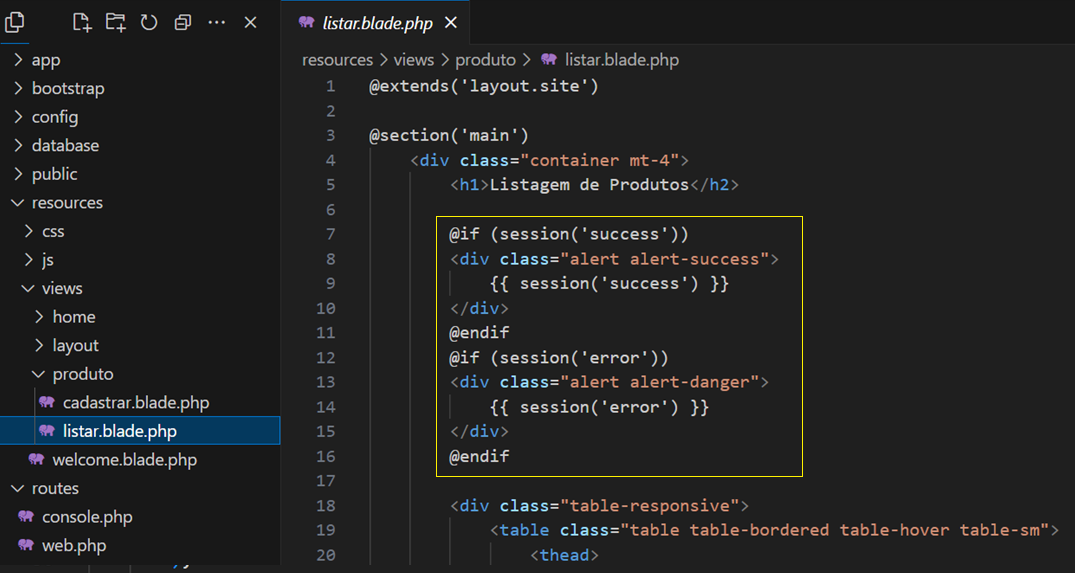
@if (session('error'))

<div class="alert alert-danger">

{{ session('error') }}

</div>

@endif



Salve tudo e teste novamente o cadastro de um produto.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente com confiança média