**Aula 5 – CRUD-Produto em Laravel**

**Editar/Excluir Produto**

**Objetivos:**

* Implementar as funcionalidades de Editar e Excluir um Produto

Ao final desta Aula devemos ter em funcionamento o acesso ao Banco de Dados MySQL para a operação de Edição (update) e Exclusão(delete) de Produto para a nossa Aplicação.

O acesso a estas funções será feito através da interface de Listar Produtos, como mostrado a seguir.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**1. Editar Produto**

Os passos para implementar a Edição de Produtos podem ser resumidos em:

1. Inserir o botão Editar em cada linha da Tabela de Listagem de Produtos;
2. Ao clicar no Botão Editar, enviar o Id do produto para o script que vai acessar o banco e recuperar os dados do Produto;
3. Montar o Formulário de Edição do Produto com os dados atuais do produto;
4. Quando o usuário submeter, receber os dados do Formulário e validá-los;
5. Atualizar os dados na Tabela Produtos;
6. Mostrar uma mensagem de Feedback para o usuário.

**1.1. Inserir o botão Editar**

Edite o script resources\views\listar.blade.php e acrescente uma coluna na linha de cabeçalho (<thead>) na tabela que apresenta as informações de cada produto, bem como o botão Editar na coluna final de cada linha.

...

<th class="info"></th>

...

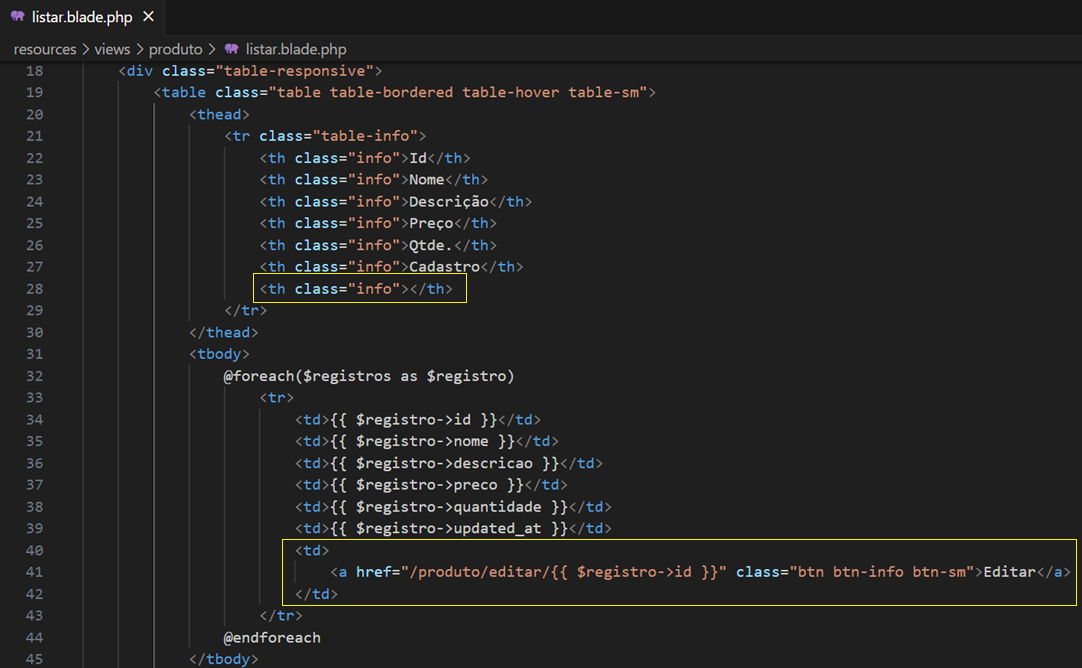
...

<td>

<a href="/produto/editar/{{ $registro->id }}" class="btn btn-info btn-sm">Editar</a>

</td>...

Note no quadro acima que foram mostrados somente as linhas que devem ser acrescentadas no referido arquivo (listar.blade.php). A figura a seguir mostra o local correto onde cada código deve ser inserido.



Salve as modificações e teste. O resultado deve ser parecido como a seguir.

Coloque o mouse sobre o botão Editar para ver se o link aparece corretamente na parte inferior da janela.

Note que deve ser apresentado o Id do produto da referida linha.

Tabela

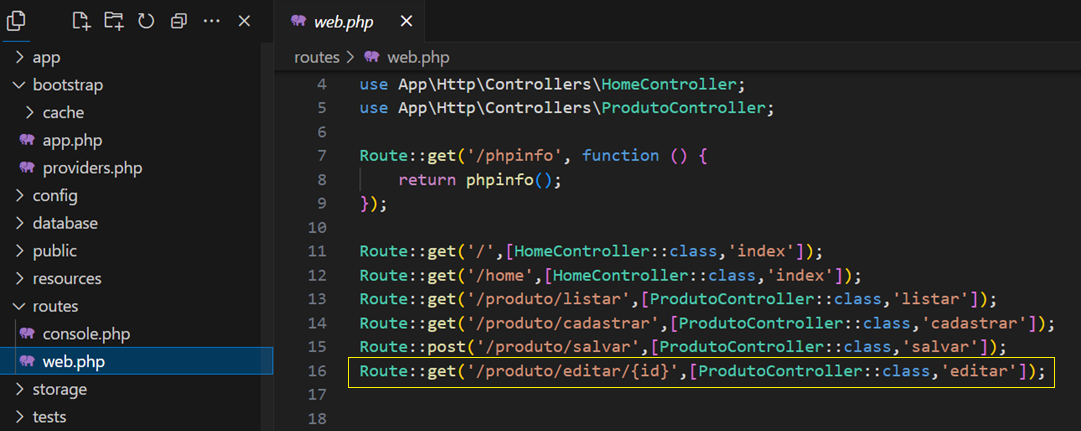
Descrição gerada automaticamente com confiança baixa

Posicione o mouse sobre o botão Editar de cada linha da Tabela e o link contendo o endereço seguido do ID do produto será exibido no canto inferior esquerdo da janela.

**1.2. Configurar a Rota para a função editar do Controlador Produto**

Acrescente a seguinte linha no arquivo de rotas routes\web.php:

Route::get('/produto/editar/{id}',[ProdutoController::class,'editar']);



Note acima na definição da rota a presença do parâmetro {id}. Isto significa que o que vier nesta posição deverá ser tratado como o ID do produto a ser editado.

**1.3. Implementar o método editar()do Controller**

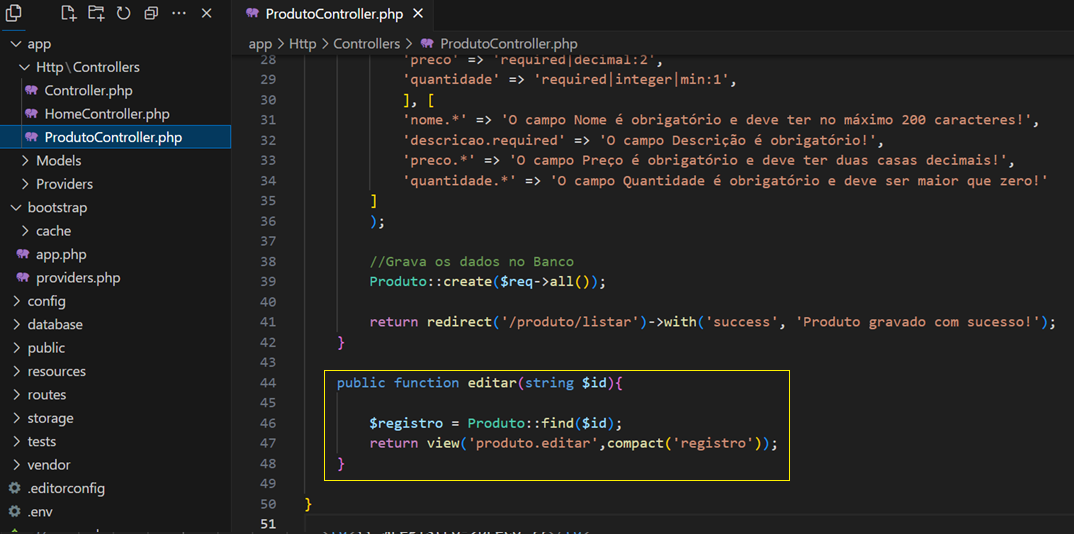
Acrescente o método editar() em app\Http\Controllers\ProdutoController.php

public function editar(string $id){

$registro = Produto::find($id);

return view('produto.editar',compact('registro'));

}



Note no código anterior o método find($id) do Eloquent do Laravel, o qual será responsável por buscar na tabela de produtos o produto com o ID $id, que foi recebido como parâmetro. Os dados são então enviados para a view.

**1.4. Montar o Formulário de Edição do Produto**

Este Formulário é praticamente o mesmo formulário do Cadastro de Produto. Entretanto, aqui temos que preencher os campos do formulário com os dados do produto sendo editado. Esses dados foram passados pelo controlador via Seção com o nome registro. Assim, crie o script resources\views\produto\editar.blade.php com o seguinte conteúdo:

@extends('layout.site')

@section('main')

<div class="container mt-4">

<h1 class="text-center">Editar 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/atualizar" method="post">

@csrf

<div class="form-group">

<input type="hidden" name="id" value="{{ isset($registro->id) ? $registro->id : '' }}">

</div>

<div class="form-group">

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

<input type="text" class="form-control" name="nome" value="{{ isset($registro->nome) ? $registro->nome : '' }}" required>

</div>

<div class="form-group">

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

<input type="textarea" class="form-control" name="descricao" value="{{ isset($registro->descricao) ? $registro->descricao : '' }}" required></textarea>

</div>

<div class="form-group">

<label for="preco">Preço</label>

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

</div>

<div class="form-group">

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

<input type="number" class="form-control" name="quantidade" value="{{ isset($registro->quantidade) ? $registro->quantidade : '' }}" required>

</div>

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

</form>

</div>

</div>

</div>

@endsection

No Código anterior, note a presença do campo hidden (oculto) id, o qual representa o ID do produto sendo editado. Este campo foi definido como hidden porque não pode ser alterado por se tratar de chave primária da tabela, porém, ele precisa ser enviado para o backend junto com os demais dados do produto para que seja possível identificar o produto sendo editado.

Salve todos os arquivos e teste a aplicação clicando no botão Editar de um produto. Note na saída o formulário preenchido e a falta do campo id.

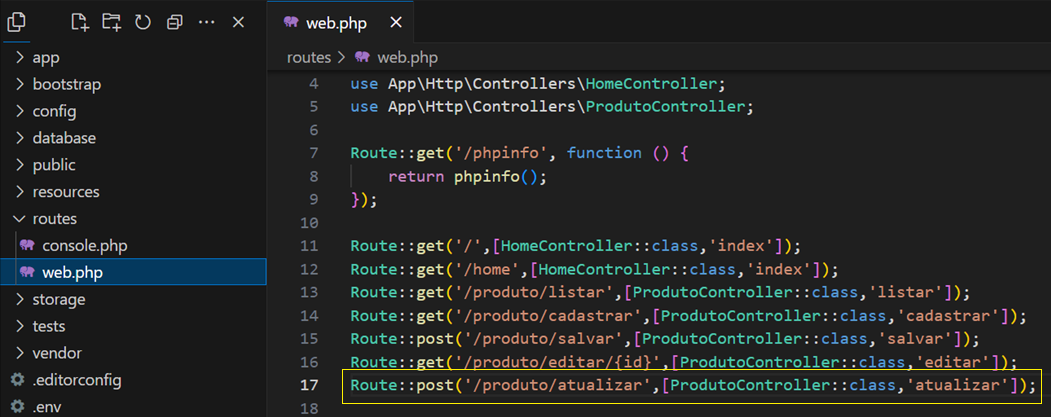
Interface gráfica do usuário

Descrição gerada automaticamente

**1.5. Configurar a Rota para a função atualizar do Controlador Produto**

Acrescente a seguinte linha no arquivo de rotas routes\web.php:

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



**1.6. Receber os dados e atualizar no Banco**

Acrescente o método atualizar() no final do Controlador do Produto em app\Http\Controllers\ProdutoController.php

public function atualizar(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!'

]

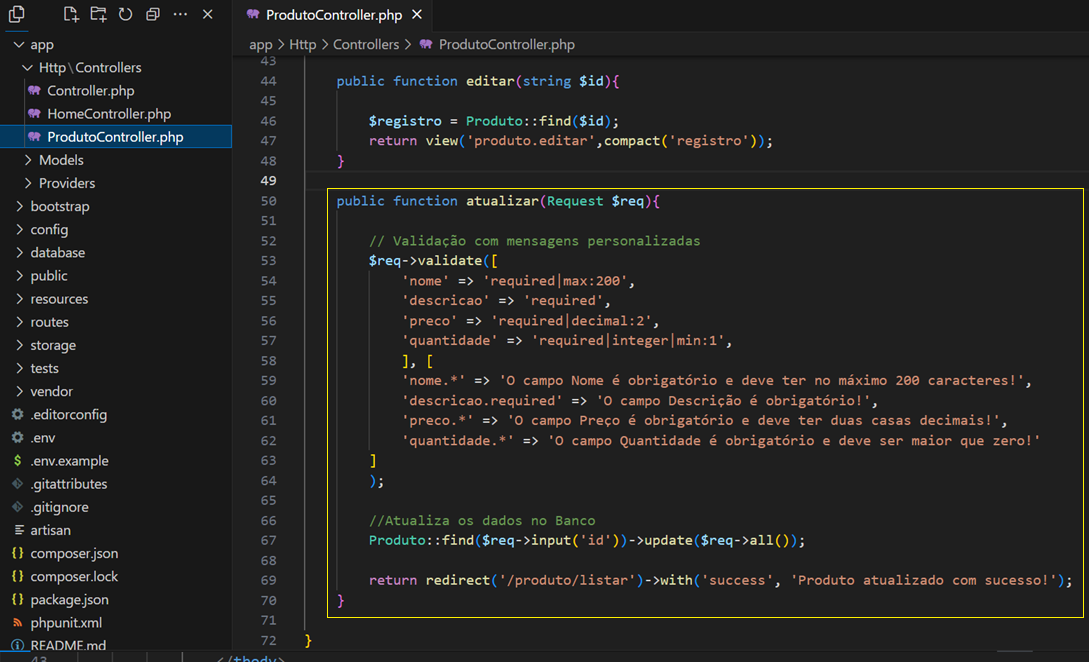
);

//Atualiza os dados no Banco

Produto::find($req->input('id'))->update($req->all());

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

}

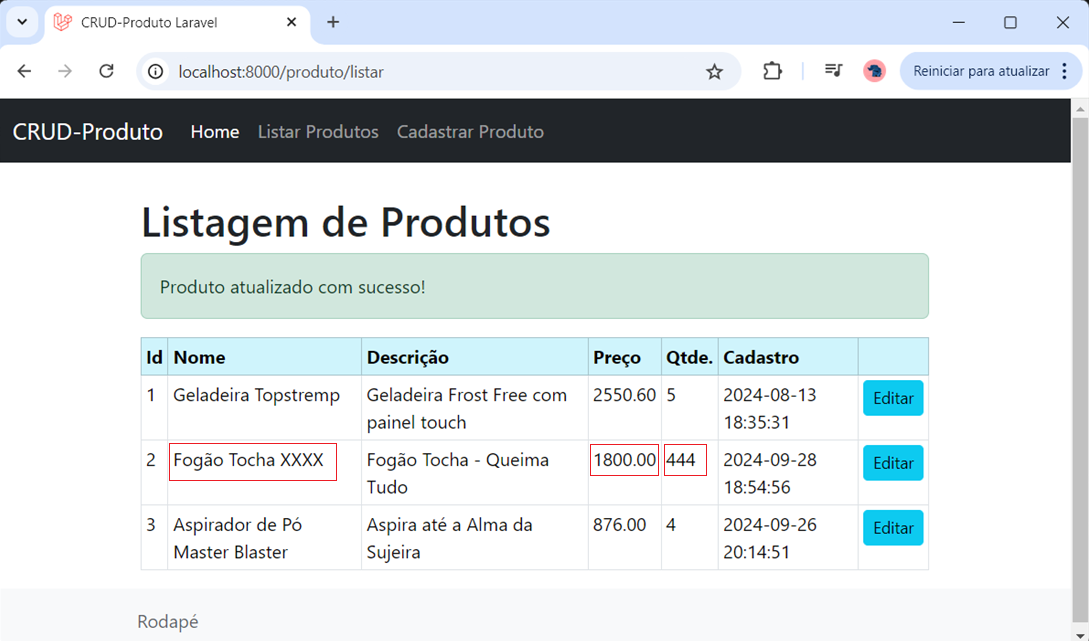


Salve tudo e teste. Edite um produto, altere seus dados e atualize.

Interface gráfica do usuário

Descrição gerada automaticamente

Note que foram alterados os campos Nome, Preço e Quantidade do produto.



**PROBLEMA NESTE PROCESSO**:

Estamos usando o mecanismo de validação de dados (validate) do Laravel, então se algum campo alterado NÃO passar na validação, o formulário será reapresentado porém populado com os dados originais do Produto, e NÃO com os dados já alterados. Veja:

Interface gráfica do usuário, Site

Descrição gerada automaticamente

Na edição anterior foram alterados o nome, o preço e a quantidade, porém a quantidade foi alterada para zero, o que não vai passar pela nossa regra de validação. Assim, ao clicar em Atualizar, o formulário será reapresentado da seguinte forma:

Interface gráfica do usuário, Texto, Email

Descrição gerada automaticamente

Note que as alterações foram perdidas!!!

**SOLUÇÃO: Vamos perguntar ao ChatGPT**

Fiz a seguinte pergunta ao ChatGPT:

Considerando o Laravel, na edição de um produto, no Controller, busquei os dados do produto no Banco e chamei a view através do seguinte código:

$registro = Produto::find($id);

return view('produto.editar',compact('registro'));

Na view populei os campos do form usando os dados do registro no atributo value como, por exemplo, no campo nome:

value="{{ isset($registro->nome) ? $registro->nome : '' }}"

Quando envio o formulário para o backend, estou utilizando o validate do Laravel para validar os dados da seguinte forma:

$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!' ]

);

O problema é que se alguma validação não passar, será retornado ao formulário de edição, porém o formulário será populado com as informações originais do produto e não as informações com as alterações feitas, o qual seria o correto. É possível resolver isso?

Resposta do ChatGPT:

Sim, é possível resolver esse problema no Laravel para que, se a validação falhar, o formulário de edição seja preenchido com os dados que o usuário tentou submeter, em vez de apenas os dados originais do produto.

### Como Funciona

Quando você utiliza o método `validate()`, o Laravel automaticamente redireciona o usuário de volta para a página anterior (no seu caso, a página de edição) e repopula o formulário com os dados que foram enviados, utilizando a função `old()`. Para que isso funcione, você deve utilizar a função `old()` para os valores dos campos do formulário.

### Passos para Resolver

1. \*\*Modifique a View para Usar `old()`\*\*: Altere os campos do formulário na view para usar `old()` em vez de `isset()` para repopular os valores.

#### Exemplo de Campo Nome

```blade

<input type="text" name="nome" value="{{ old('nome', $registro->nome) }}">

```

### Explicação:

- `old('nome', $registro->nome)`: O Laravel tenta recuperar o valor antigo da sessão usando `old('nome')`. Se não houver um valor anterior (ou seja, se a validação não falhar), ele usará o valor original do registro (`$registro->nome`).

### Exemplo Completo dos Campos

Aqui está como você poderia estruturar todos os campos do seu formulário:

```blade

<form method="POST" action="{{ route('produtos.atualizar', $registro->id) }}">

@csrf

@method('PUT')

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

<input type="text" name="nome" value="{{ old('nome', $registro->nome) }}">

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

<textarea name="descricao">{{ old('descricao', $registro->descricao) }}</textarea>

<label for="preco">Preço</label>

<input type="text" name="preco" value="{{ old('preco', $registro->preco) }}">

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

<input type="number" name="quantidade" value="{{ old('quantidade', $registro->quantidade) }}">

<button type="submit">Atualizar Produto</button>

</form>

```

### Comportamento Esperado

- Quando o formulário é enviado, se alguma validação falhar, o Laravel redirecionará de volta para a página de edição e os campos do formulário serão preenchidos com os dados que o usuário tentou submeter.

- Se a validação for bem-sucedida, os dados do produto serão atualizados normalmente.

### Conclusão

Usar `old()` nos campos do formulário é a maneira recomendada de manter os dados do usuário em caso de erro de validação. Isso proporciona uma melhor experiência ao usuário, permitindo que eles vejam o que foi digitado anteriormente e façam as correções necessárias sem precisar reescrever tudo.

Então, baseado nas sugestões do ChatGPT, altere o script resources\views\produto\editar.blade.php para o seguinte conteúdo:

@extends('layout.site')

@section('main')

<div class="container mt-4">

<h1 class="text-center">Editar 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/atualizar" method="post">

@csrf

<div class="form-group">

<input type="hidden" name="id" value="{{ isset($registro->id) ? $registro->id : '' }}">

</div>

<div class="form-group">

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

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

</div>

<div class="form-group">

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

<input type="textarea" class="form-control" name="descricao" value="{{ old('descricao',$registro->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',$registro->preco) }}">

</div>

<div class="form-group">

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

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

</div>

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

</form>

</div>

</div>

</div>

@endsection

Salve e refaça o teste. Veja que agora o Formulário será reapresentado com a mensagem de erro e os dados alterados.

Interface gráfica do usuário, Texto, Email

Descrição gerada automaticamente

**2. Excluir Produto**

Os passos para implementar a Exclusão de Produtos podem ser resumidos em:

1. Inserir o botão Excluir em cada linha da Tabela de Listagem de Produtos;
2. Ao clicar no Botão Excluir, enviar o Id e o nome do produto para o script que vai solicitar confirmação da Exclusão;
3. Se confirmada a Exclusão, excluir o produto da Tabela Produtos;
4. Mostrar uma mensagem de Feedback para o usuário.

**2.1. Inserir o botão Excluir**

Edite o script resources\views\listar.blade.php e troque a linha que programa o botão Editar pelo seguinte código:

...

<td>

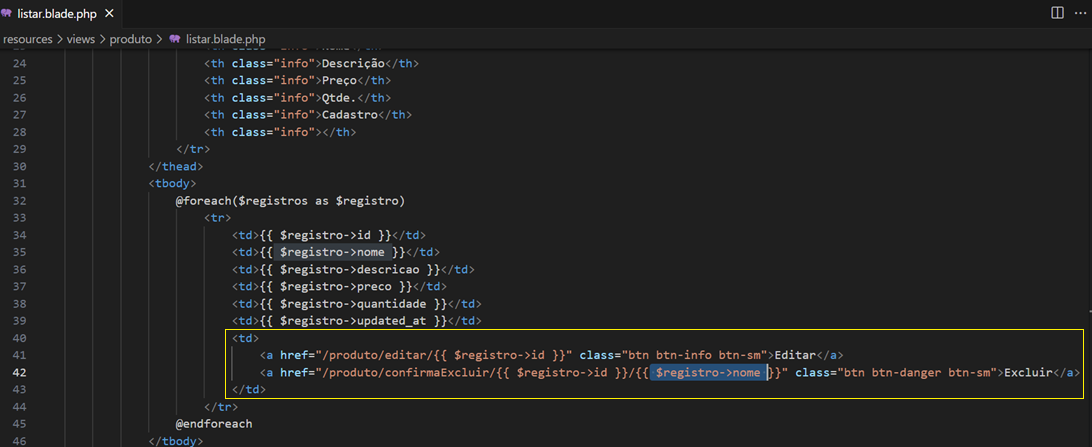
<a href="/produto/editar/{{ $registro->id }}" class="btn btn-info btn-sm">Editar</a>

<a href="/produto/conformaExcluir/{{ $registro->id }}/{{ $registro->nome }}

" class="btn btn-danger btn-sm">Excluir</a>

</td>

...



Salve e teste. Passe o mouse sobre o botão excluir e perceba que na parte inferior esquerda da janela será exibido o link para a função excluir, passando o ID e o Nome do produto.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

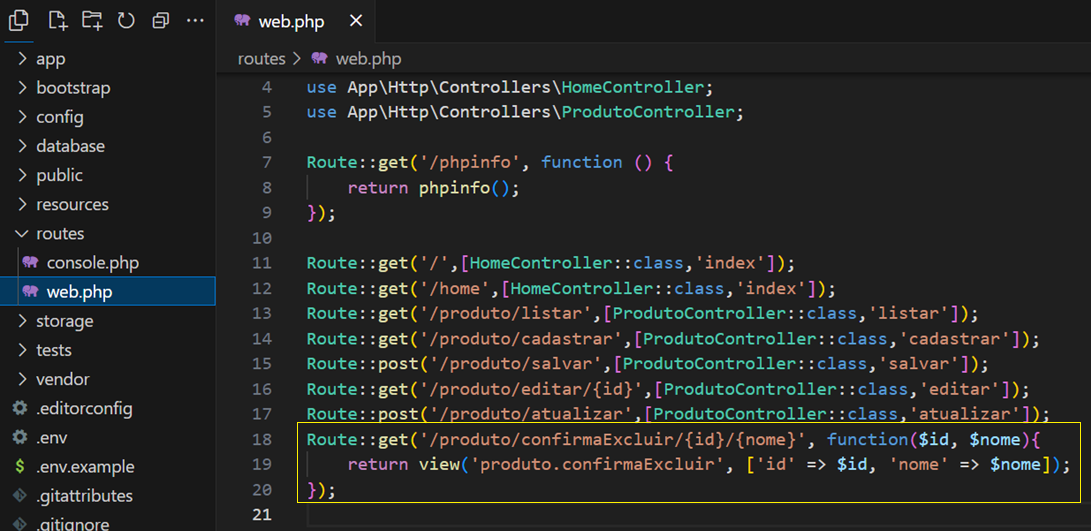
**2.2. Criar a Rota que chama o Formulário de confirmação da Exclusão**

Acrescente no final do routes\web.php a seguinte rota:

Route::get('/produto/confirmaExcluir/{id}/{nome}', function($id, $nome){

return view('produto.confirmaExcluir', ['id' => $id, 'nome' => $nome]);

});



Note que está sendo chamada a view na configuração da própria rota passando os parâmetros id e nome do produto a ser excluído.

**2.3. Criar a View com o Formulário de confirmação da Exclusão**

Crie o arquivo resources\views\produto\confirmaExcluir.blade.php com o seguinte conteúdo:

@extends('layout.site')

@section('main')

<div class="container mt-4">

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

<div class="row">

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

<div class="card bg-danger text-white">

<div class="card-header">

<h5>Confirmação de Exclusão do Produto</h5>

</div>

<div class="card-body">

<p>Tem certeza de que deseja excluir o produto: <strong>{{ $nome }}</strong>?</p>

</div>

<div class="card-footer text-end">

<form action="/produto/excluir/{{ $id }}" method="POST">

@csrf

@method('DELETE')

<button type="submit" class="btn btn-danger">Confirmar</button>

<a href="/produto/listar" class="btn btn-secondary">Cancelar</a>

</form>

</div>

</div>

</div>

</div>

</div>

@endsection

Salve e teste. Na listagem de produtos, clique no botão excluir de um produto.

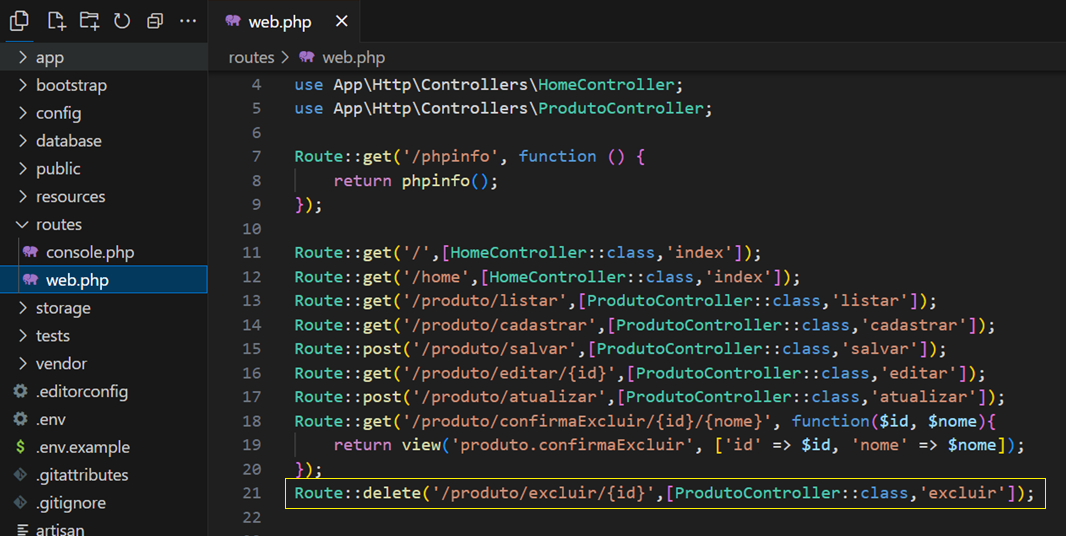
Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**2.4. Criar a Rota e o método para exclusão do Produto**

Acrescente no final do routes\web.php a seguinte rota:

Route::delete('/produto/excluir/{id}',[ProdutoController::class,'excluir']);



Acrescente o método excluir no Controlador do Produto:

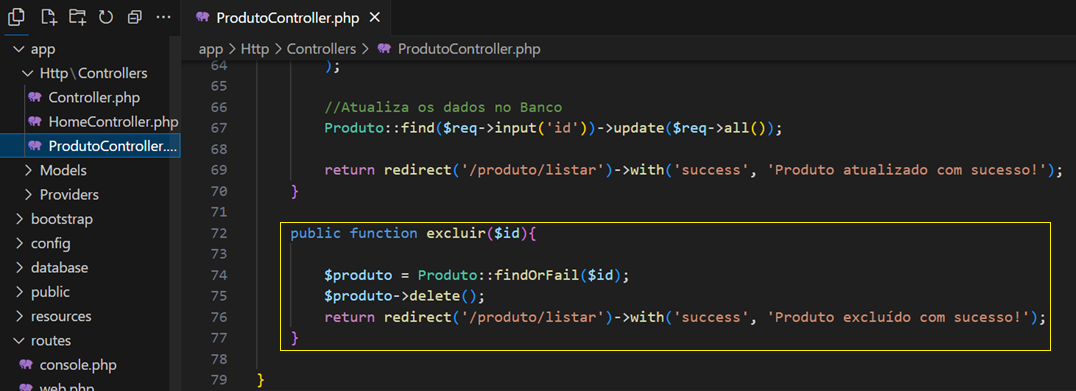
public function excluir($id){

$produto = Produto::findOrFail($id);

$produto->delete();

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

}



Salve e teste.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente