**Request e métodos HTTP**

**Criando formulário de novos produtos**

Já somos capazes de visualizar uma lista com todos os produtos e os detalhes de um produto específico, mas ainda não temos uma função de adicionar novos produtos em estoque. Vamos implementá-la?

Podemos começar pela rota, no arquivo routes.php. Vamos registrar uma URI /produtos/novo apontando para o método novo, do ProdutoController. Ela pode ficar assim:

Route::get('/produtos/novo', 'ProdutoController@novo');

Vale lembrar que, antes de adicionar um produto no banco de dados, precisamos abrir uma página com formulário para que o usuário possa preencher os dados, portanto é isso o que o método novo fará. Ele retornará uma view chamada formulario, que será criada dentro da pasta produto:

public function novo(){

return view('produto.formulario');

}

Para fechar esse passo inicial, precisamos criar o arquivo formulario.blade.php dentro de /resources/view/produtos/. Como queremos que ele use o layout padrão, no início do arquivo declaramos que ele herda do layout principal e declaramos a section de conteúdo, assim como fizemos com os demais arquivos. Já vamos adicionar um form também. O arquivo deve ficar assim:

@extends('layout.principal')

@section('conteudo')

<h1>Novo produto</h1>

<form>

<label>Nome</label>

<input/>

<label>Descricao</label>

<input/>

<label>Valor</label>

<input/>

<label>Quantidade</label>

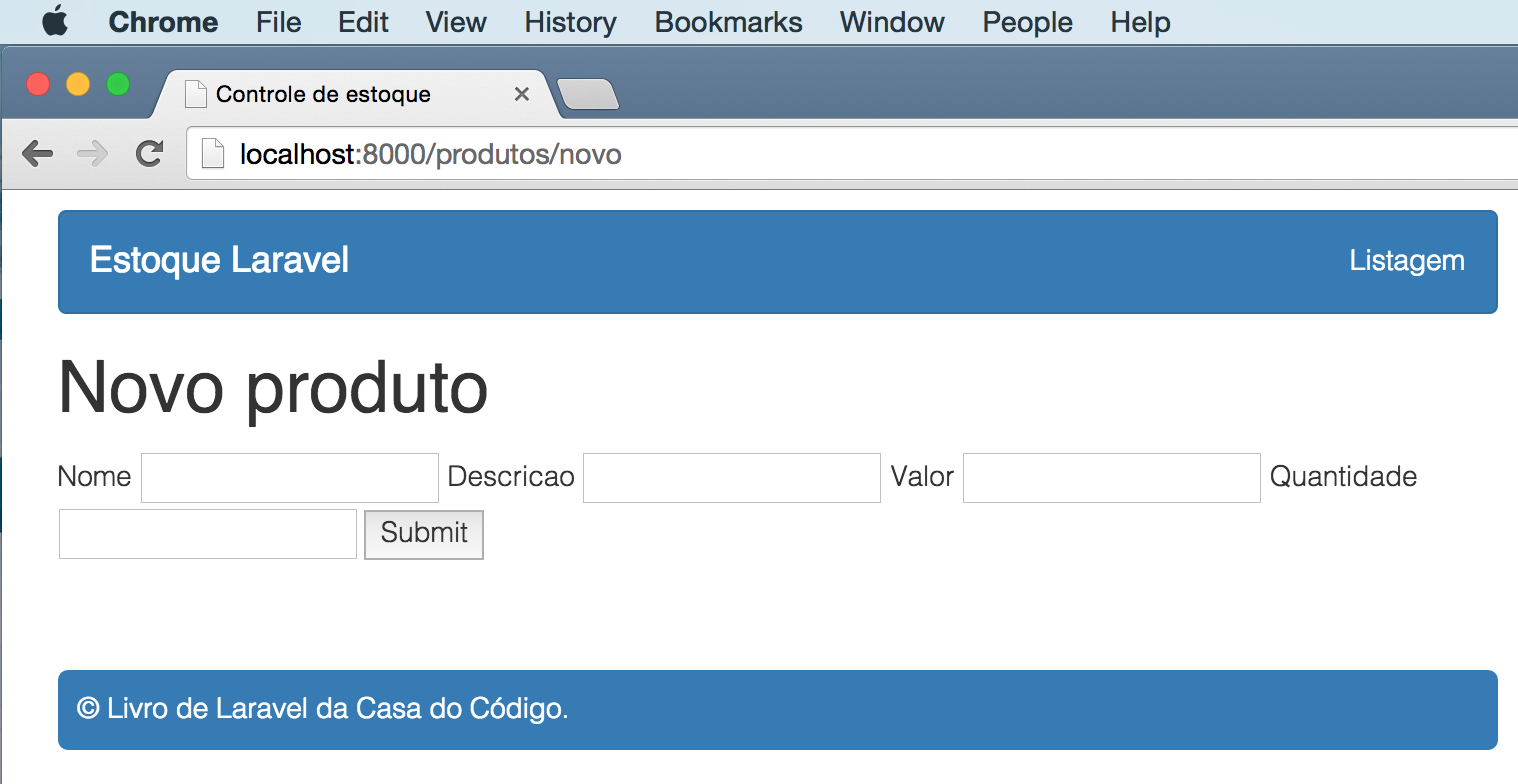
<input type="number"/>

<button type="submit">Submit</button>

</form>

@stop

Vamos testar para garantir que tudo funcionou até aqui? Só precisamos acessar a URL <http://localhost:8000/produtos/novo>. O resultado será:



O formulário está um pouco estranho, com as *labels* e *inputs* todos bagunçados em uma mesma linha. Já que estamos usando bootstrap, podemos usar um de seus modelos de formulários que pode ser encontrado em sua documentação:

<http://getbootstrap.com/css/#forms>

Fazendo a alteração, o HTML poderá ficar assim:

@extends('layout.principal')

@section('conteudo')

<h1>Novo produto</h1>

<form>

<div class="form-group">

<label>Nome</label>

<input class="form-control">

</div>

<div class="form-group">

<label>Descricao</label>

<input class="form-control">

</div>

<div class="form-group">

<label>Valor</label>

<input class="form-control">

</div>

<div class="form-group">

<label>Quantidade</label>

<input type="number" class="form-control">

</div>

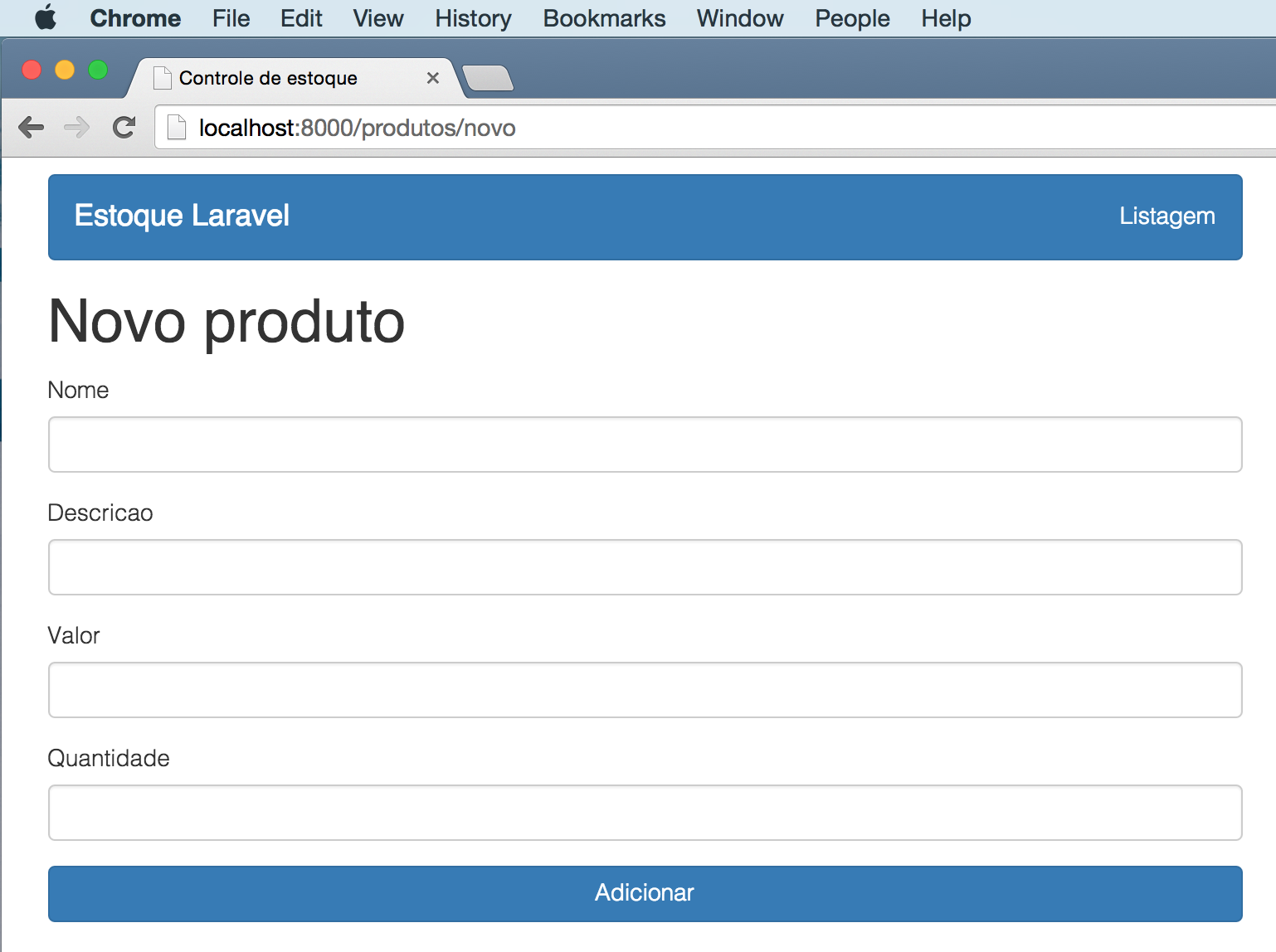
<button type="submit" class="btn

btn-primary btn-block">Submit</button>

</form>

@stop

Vamos ver o resultado?



Excelente! Agora que temos uma nova página, podemos adicionar um novo link no cabeçalho do arquivo principal.blade.php. O HTML com o novo link ficará assim:

<ul class="nav navbar-nav navbar-right">

<li><a href="/produtos">Listagem</a></li>

<li><a href="/produtos/novo">Novo</a></li>

</ul>

Com isso boa parte do trabalho está pronto, mas agora precisamos implementar a função que efetivamente adicionará um novo produto ao banco.

**Criando o método adiciona**

Como de costume, podemos começar pelo mapeamento desse método, que deve ser feito no routes.php. Podemos registrar uma nova rota /produtos/adiciona, que apontará para o método adiciona do ProdutoController. O código ficará assim:

Route::get('/produtos/adiciona', 'ProdutoController@adiciona');

Agora que já temos a rota, precisamos criar esse novo método no controller. Sua assinatura ficará desta forma:

public function adiciona(){

// deve adicionar os produtos no banco

}

E agora, qual o próximo passo? Antes mesmo de pensar no SQL que deve ser executado no banco de dados, ou na view que deve ser retornada, precisamos pegar os parâmetros digitados no formulário de alguma maneira! Os passos necessários para o método adiciona serão:

public function adiciona(){

// pegar dados do formulario

// salvar no banco de dados

// retornar alguma view

}

Já vimos que é a classe Request quem cuida desse trabalho, de pegar os parâmetros da requisição. Portanto podemos fazer algo como:

$nome = Request::input('o\_que\_passar\_aqui\_?');

Eis a questão. Quando estávamos buscando o parâmetro enviado pelo link de visualizar, sabíamos que a chave era id, pois ela era enviada pela URL dessa forma:

/produtos/mostra?id=1

Veja que estamos deixando explícito que o parâmetro se chama id, e que ele recebe o valor 1. Como agora temos um formulário, e não um link, precisaremos adicionar um name em cada *input* do formulário para que seja possível recuperar suas informações pelo controller. O HTML completo ficará assim:

@extends('layout.principal')

@section('conteudo')

<h1>Novo produto</h1>

<form action="/produtos/adiciona">

<div class="form-group">

<label>Nome</label>

<input name="nome" class="form-control"/>

</div>

<div class="form-group">

<label>Descricao</label>

<input name="descricao" class="form-control"/>

</div>

<div class="form-group">

<label>Valor</label>

<input name="valor" class="form-control"/>

</div>

<div class="form-group">

<label>Quantidade</label>

<input type="number" name="quantidade" class="form-control"/>

</div>

<button type="submit"

class="btn btn-primary btn-block">Submit</button>

</form>

@stop

Com isso, em nosso controller, podemos recuperar cada um dos valores:

public function adiciona(){

$nome = Request::input('nome');

$descricao = Request::input('descricao');

$valor = Request::input('valor');

$quantidade = Request::input('quantidade');

// salvar no banco de dados

// retornar alguma view

}

**Outras alternativas**

Podemos sim utilizar o método all da Request para pegar todos os valores de uma única vez em um array, ou ainda o método only deixando explícito quais parâmetros queremos buscar:

$all = Request::all();

$only = Request::only('nome', 'valor', '...');

Mas para deixar bem explícito, e garantir a ordem dos dados, por enquanto faremos dessa forma. Voltaremos a falar sobre esse método para melhorar este e outros pontos que serão detalhados mais à frente.

Ótimo, com isso já temos cada um dos valores digitados no formulário. Vamos testar? Podemos mudar o método adiciona para retornar esses valores:

public function adiciona(){

$nome = Request::input('nome');

$descricao = Request::input('descricao');

$valor = Request::input('valor');

$quantidade = Request::input('quantidade');

return implode( ', ', array($nome,

$descricao, $valor, $quantidade));

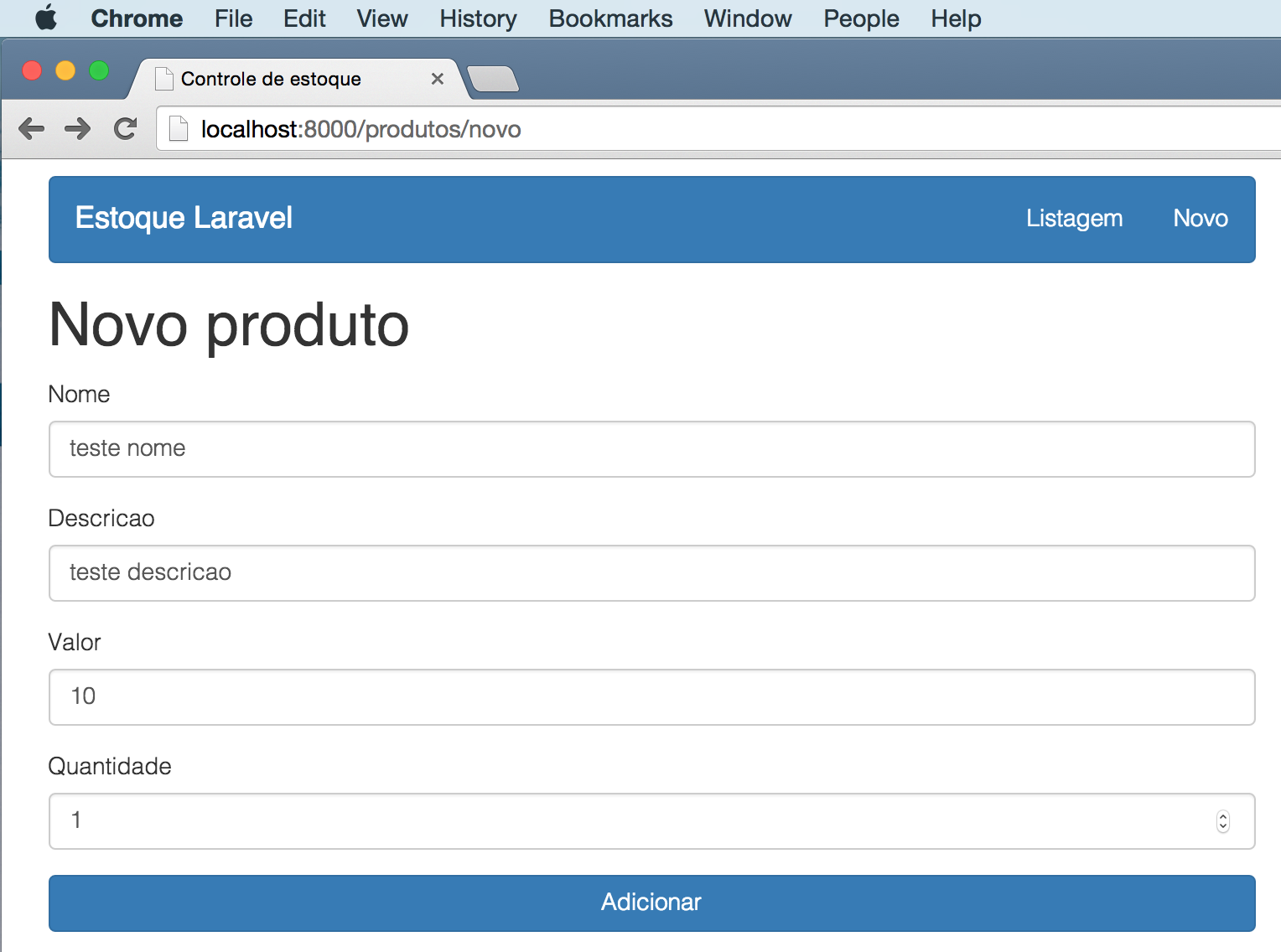
}

O implode fará com que cada um dos valores seja impresso separado por vírgula. Agora só falta um detalhe, precisamos adicionar a action do formulário, para que ele envie para o método adiciona:

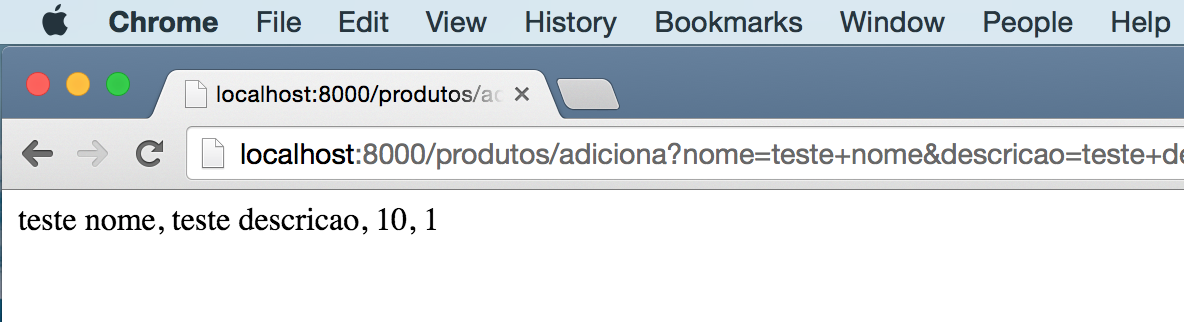
<form action="/produtos/adiciona">

<!-- restante do html omitido -->

Vamos testar? Basta abrir o formulário no navegador e preencher os campos com algum valor de teste:



Ao clicar no botão *Adicionar*, a saída será:



**Inserindo os produtos no BD**

Boa parte do trabalho está pronta, mas ainda precisamos persistir o produto no banco de dados. Por enquanto, faremos isso com a função insert da interface DB, que já usamos no método lista. O código de adicionar ficará parecido com:

DB::insert('insert into produtos values (null, ?, ?, ?, ?)',

array($nome, $valor, $descricao, $quantidade));

Veja que, assim como fizemos no *select* do método mostra, utilizamos uma ? no lugar dos valores e passamos um array na ordem em que esses elementos devem ser substituídos. Não tem muito segredo, mas se você não está muito acostumado com SQL pode se assustar um pouco. Muito em breve veremos como o Laravel nos ajuda neste trabalho.

**Retornando uma view de confirmação**

Agora que já estamos persistindo, podemos retornar uma *view* com a mensagem de confirmação, dizendo que o produto foi adicionado com sucesso. Podemos chamá-la de adicionado.blade.php.

@extends('layouts.principal')

@section('conteudo')

O produto foi adicionado com sucesso!

@stop

Vale lembrar que essa view será criada dentro de /resources/views/produto, assim como as demais, portanto, no retorno do controller faremos produto.adicionado:

public function adiciona(){

$nome = Request::input('nome');

$valor = Request::input('valor');

$descricao = Request::input('descricao');

$quantidade = Request::input('quantidade');

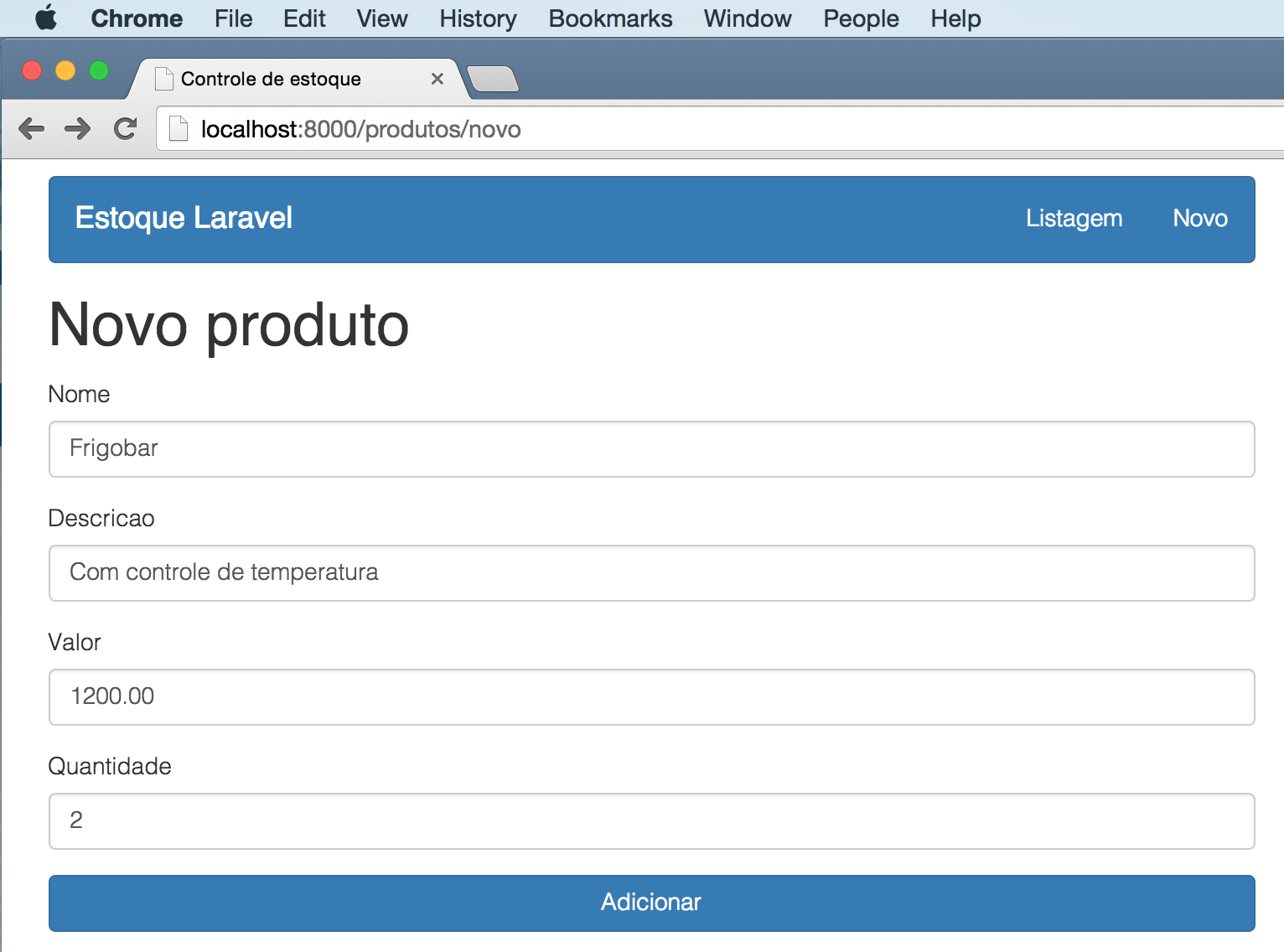
DB::insert('insert into produtos values (null, ?, ?, ?, ?)',

array($nome, $valor, $descricao, $quantidade));

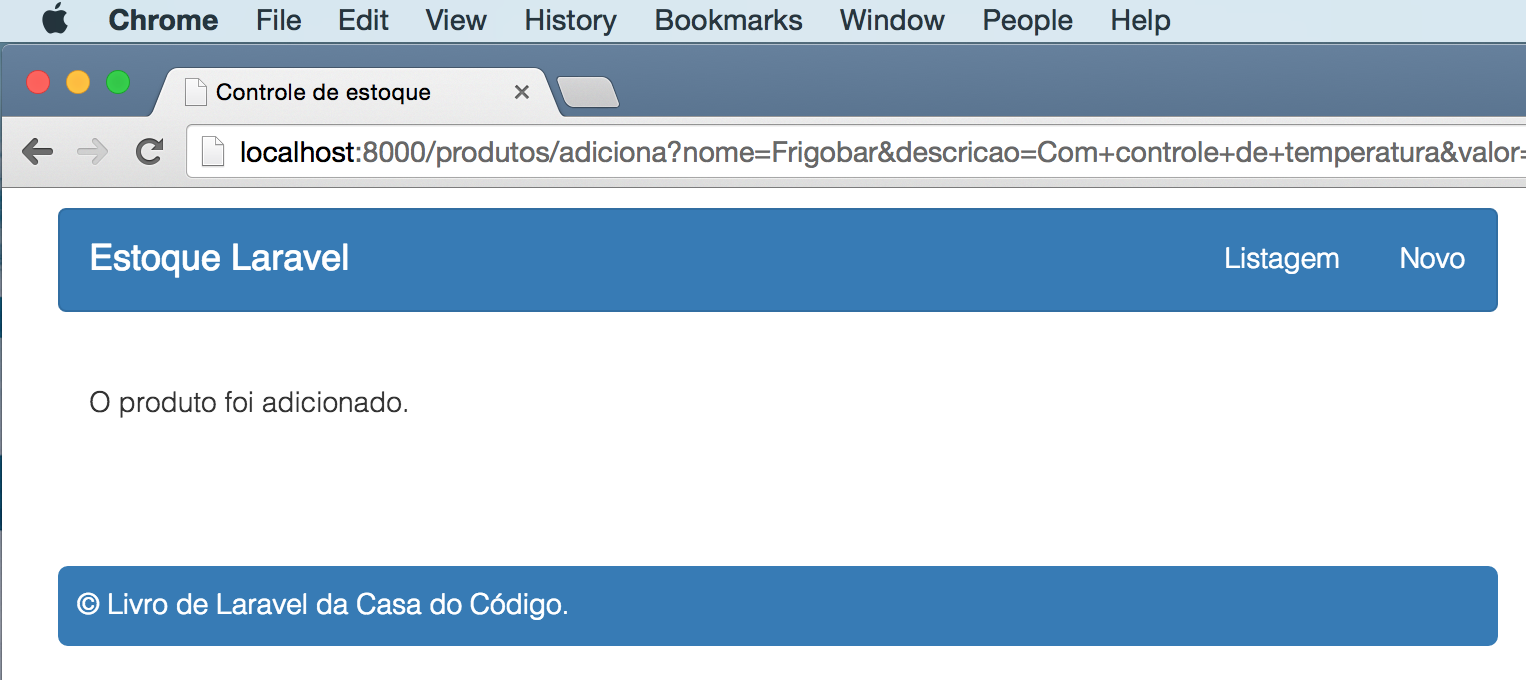
return view('produto.adicionado');

}

Já podemos fazer um novo teste, garantindo que a mensagem será exibida e que o produto será persistido no banco.



Ao submeter o formulário, recebemos o seguinte resultado:



Tudo funcionou como esperado, mas ainda assim alguns detalhes podem ser melhorados. Seria legal, por exemplo, exibir o nome do produto que foi adicionado, não é? Isso é bem simples, basta incluir o parâmetro pelo controller, com o método with que já vimos. O método adiciona completo ficará como o seguinte:

public function adiciona(){

$nome = Request::input('nome');

$valor = Request::input('valor');

$descricao = Request::input('descricao');

$quantidade = Request::input('quantidade');

DB::insert('insert into produtos values (null, ?, ?, ?, ?)',

array($nome, $valor, $descricao, $quantidade));

return view('produto.adicionado')->with('nome', $nome);

}

Na view, exibimos o $nome incluído:

@extends('layout.principal')

@section('conteudo')

O produto {{$nome}} foi adicionado com sucesso!

@stop

Além disso, podemos adicionar uma *div* com as classes do bootstrap para deixar a mensagem com uma aparência um pouco mais amigável. Uma alternativa seria:

@extends('layout.principal')

@section('conteudo')

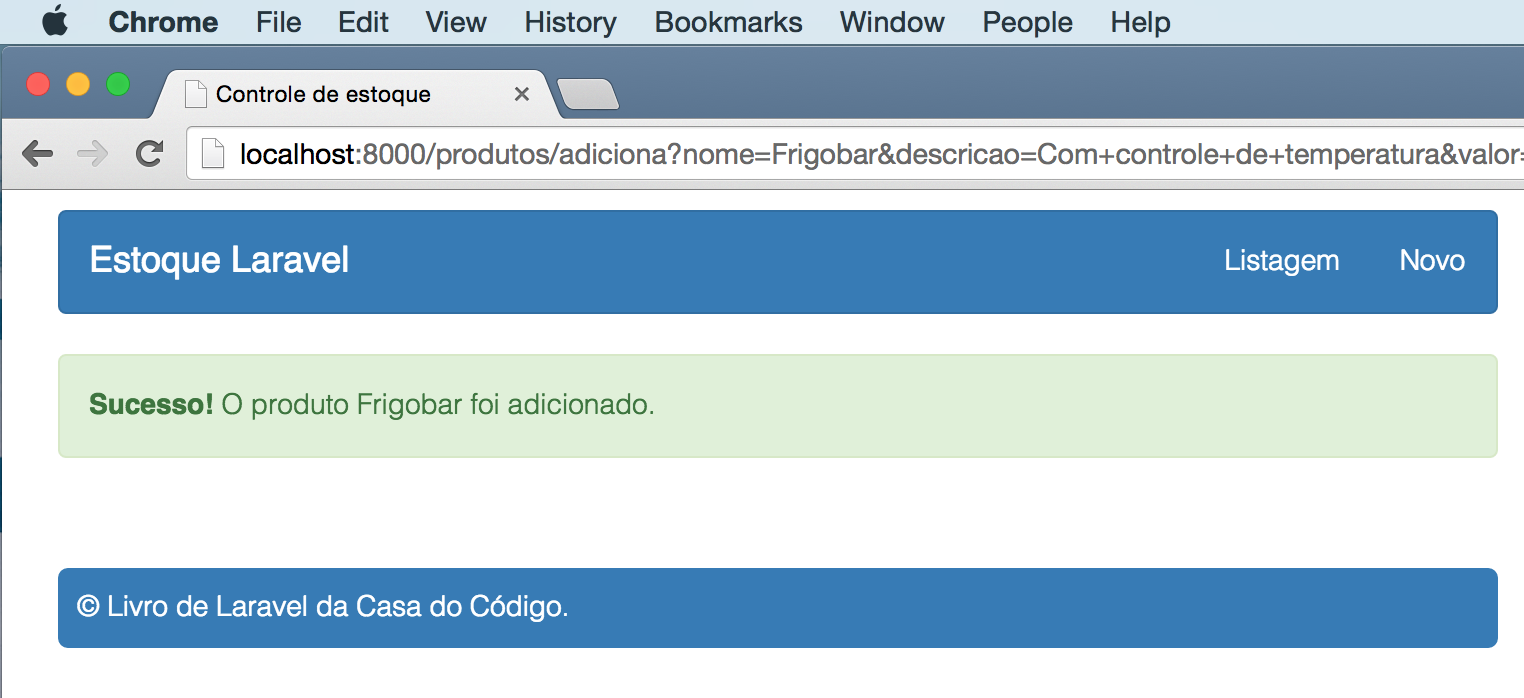
<div class="alert alert-success">

<strong>Sucesso!</strong> O produto {{$nome}} foi adicionado.

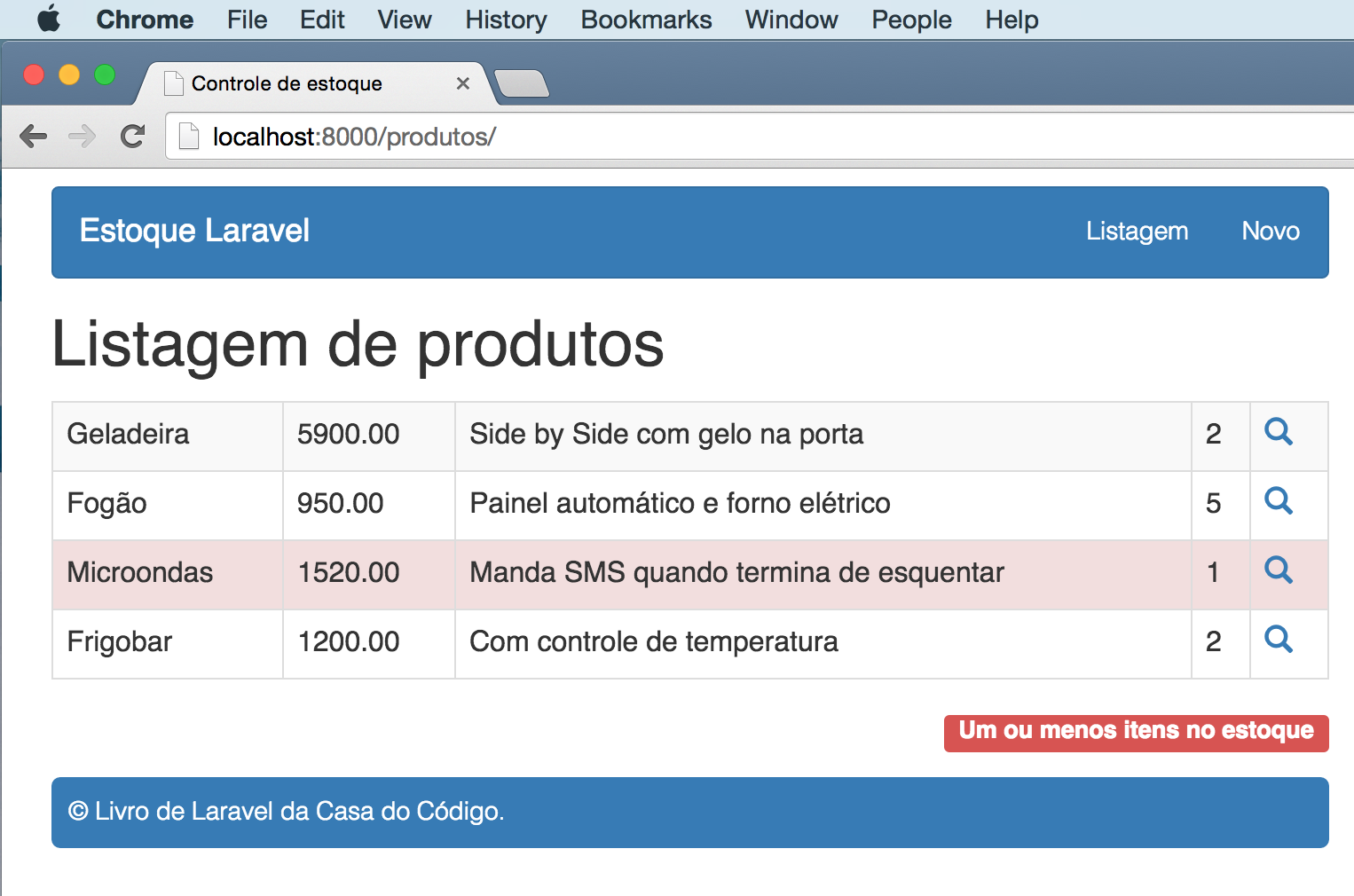
</div>

@stop

Agora sim, o resultado será:



Vamos acessar também a listagem, para garantir que o produto foi persistido no banco de dados como deveria.



Perfeito, deu tudo certo!

**Utilizando os métodos do HTTP**

O cadastro de produtos está funcionando bem, ele é adicionado ao banco, uma página de confirmação bonita é exibida, tudo parece correto. Só que há um ponto muito importante que precisa ser corrigido.

Veja que, ao adicionar um produto, todos os parâmetros da requisição estão sendo enviados pela URL:

http://localhost:8000/produtos

/adiciona?nome=Frigobar

&descricao=Com+controle+de+temperatura

&valor=1200.00&quantidade=2

Isso acontece porque, por padrão, o método HTTP utilizado no form do HTML é o GET. Mas nem sempre queremos deixar tudo visível dessa forma, imagine por exemplo se esse fosse um formulário de login, a URL ficaria assim:

http://localhost:8000

/login?usuario=Rodrigo

&senha=mamae+querida

Um caos! Além disso, existe um limite para o tamanho da URL, ou seja, se fosse um formulário muito grande, ou pequeno com muito texto, correríamos o risco de perder informações se esse conteúdo ultrapassasse esse limite.

No lugar de enviar por GET, podemos modificar o formulário para utilizar um outro método HTTP, o POST. Dessa forma, os dados não ficarão mais visíveis na URL, tudo será passado dentro do corpo do protocolo HTTP.

A URL ficará <http://localhost:8000/produtos/adiciona>, sem os dados visíveis, independente da quantidade de parâmetros enviados. É muito importante lembrar que isso não é uma medida de segurança, afinal os dados ainda podem ser consultados pelo navegador do usuário. O ganho dessa abordagem é que o corpo do POST é muito maior que a URI do GET, e evitamos deixar os dados sempre tão expostos.

Tudo o que precisamos fazer para que os dados sejam enviados via POST no *form* de novo de produto é adicionar o atributo method="post", como a seguir:

<h1>Novo produto</h1>

<form action="/produtos/adiciona" method="post">

<div class="form-group">

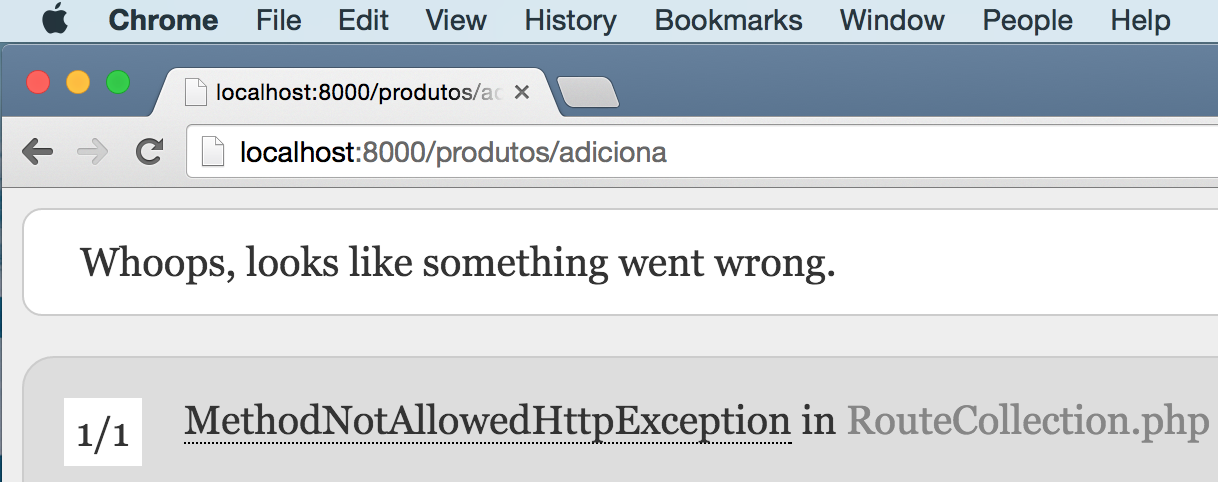
<label>Nome</label>

<input name="nome" class="form-control"/>

</div>

<!-- continuação do html -->

Mas, com essa alteração, nosso formulário para de funcionar! Tente adicionar um novo produto para ver o resultado.



Recebemos uma MethodNotAllowedHttpException, mas o que isso significa? O que acontece é que no momento de mapear uma rota para o método adiciona, fizemos:

Route::get('/produtos/adiciona', 'ProdutoController@adiciona');

Repare que, como utilizamos o método get, definimos explicitamente que apenas requisições desse tipo serão aceitas. É justo. Como resolver? Uma possibilidade seria utilizando o método any, ou seja, liberando qualquer tipo de requisição para essa URL. Mas não é isso que queremos, não é? Queremos enviar os dados por POST; portanto, utilizaremos o método post:

Route::post('/produtos/adiciona', 'ProdutoController@adiciona');

**Para saber mais: O método match**

Além de get, post e any, a interface Route ainda nos oferece um método chamado match, que nos permite passar um array especificando exatamente quais métodos HTTP devem ser aceitos. Um exemplo seria:

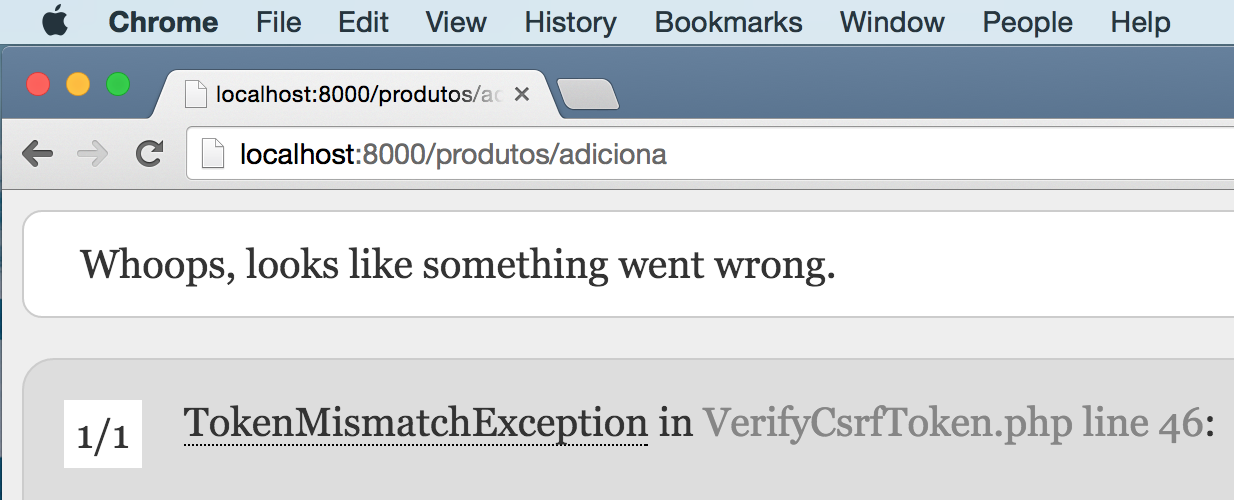
Route::match(array('GET', 'POST'),

'/produtos/adiciona',

'ProdutoController@adiciona');

Neste caso, requisições de tipo GET ou POST seriam aceitas.

Vamos testar? É só abrir o formulário novamente no navegador, preencher os campos e clicar em adicionar. Para nossa surpresa, ou não, ainda não funcionará. Mas desta vez o erro será outro:



Recebemos um TokenMismatchException, que acontece sempre que enviamos uma requisição do tipo POST sem mandar um parâmetro chamado \_token. Mas por que isso é necessário? Por padrão, o Laravel tem um mecanismo de proteção contra **CSRF** (*Cross-site request forgery*, ou falsificação de solicitação entre sites). Esse é um tipo de ataque, bastante conhecido no mundo web, que tira proveito da relação de confiança entre seu usuário legítimo e um aplicativo web. Você pode ler um pouco mais sobre ele aqui:

<http://pt.wikipedia.org/wiki/Cross-site_request_forgery>

A solução é simples, o Laravel tem um *helper method* chamado csrf\_token() que nos retorna esse token. Só precisamos adicionar um novo *input*, que pode e deve ficar escondido, portanto será type="hidden". Além disso, o name desse input precisa ser \_token, que é esperado pelas requisições desse tipo. O *input* ficará assim:

<input type="hidden"

name="\_token" value="{{{ csrf\_token() }}}" />

O código no nosso formulario.blade.php completo fica:

@extends('layout.principal')

@section('conteudo')

<h1>Novo produto</h1>

<form action="/produtos/adiciona" method="post">

<input type="hidden"

name="\_token" value="{{{ csrf\_token() }}}" />

<div class="form-group">

<label>Nome</label>

<input name="nome" class="form-control"/>

</div>

<div class="form-group">

<label>Descricao</label>

<input name="descricao" class="form-control"/>

</div>

<div class="form-group">

<label>Valor</label>

<input name="valor" class="form-control"/>

</div>

<div class="form-group">

<label>Quantidade</label>

<input type="number"

name="quantidade" class="form-control"/>

</div>

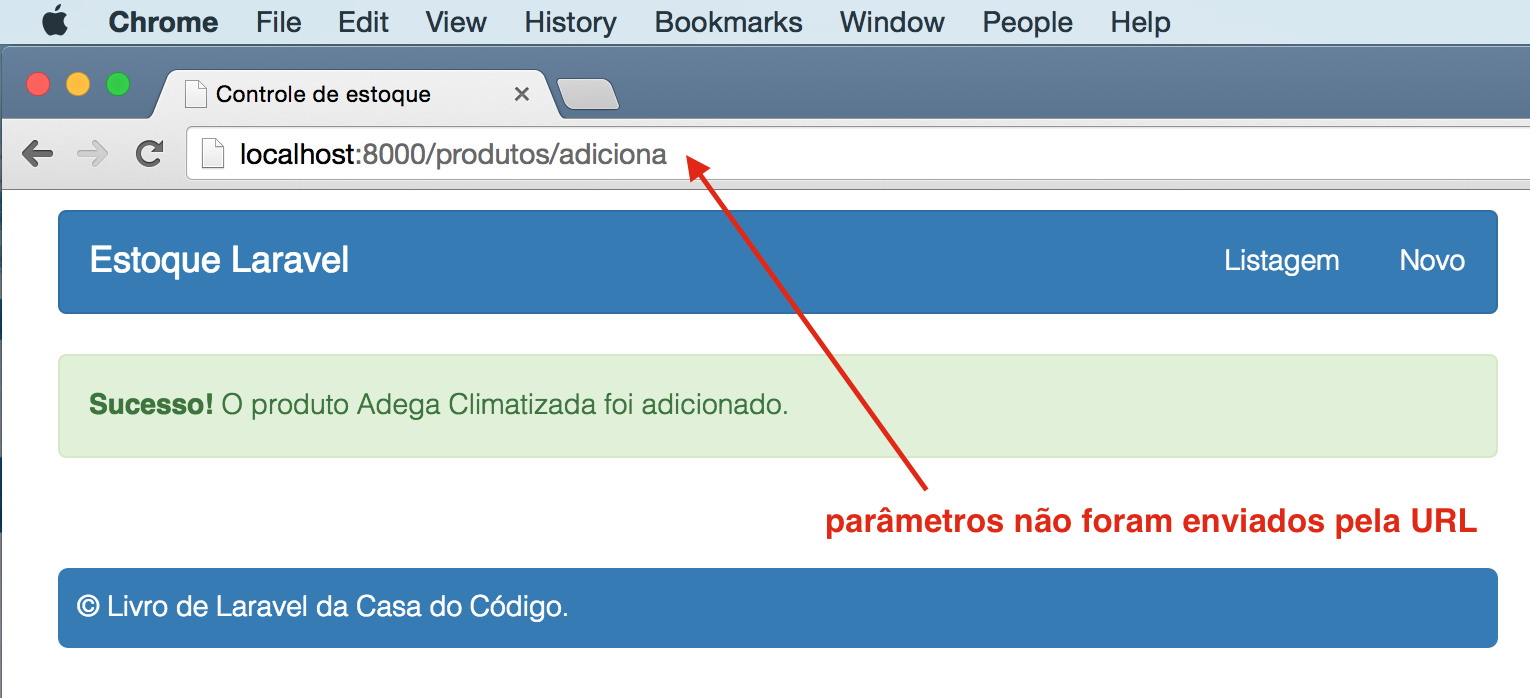
<button type="submit"

class="btn btn-primary btn-block">Adicionar</button>

</form>

@stop

Agora sim, vamos testar?



Sucesso!

**Mais métodos HTTP e quando utilizá-los**

Mas, afinal, quando devemos usar cada método HTTP? É muito comum utilizarmos o método GET para listagens ou visualização de algum de nossos recursos, sempre em operações que **não modificam os valores do sistema**.

O POST deve ser utilizado para adição de informações a um recuso existente ou **adição de um novo recurso**, como estamos fazendo agora com os produtos.