**Parâmetros da request e URL**

**Parâmetros da request e URL**

Agora que temos nosso código um pouco mais organizado, com a lógica de acesso ao banco no **controller** separada da lógica de apresentação na **view**, podemos adicionar um link em cada produto da tabela para que seja possível visualizar seus detalhes em uma nova tela.

**Exibindo detalhes do produto**

A mudança no HTML do arquivo listagem.php será pequena, só precisamos adicionar uma nova coluna com um link:

<h1>Listagem de produtos</h1>

<table class="table ...">

<?php foreach ($produtos as $p): ?>

<tr>

<td><?= $p->nome ?></td>

<!-- outras colunas omitidas -->

<td>

<a href="/produtos/mostra">

Visualizar

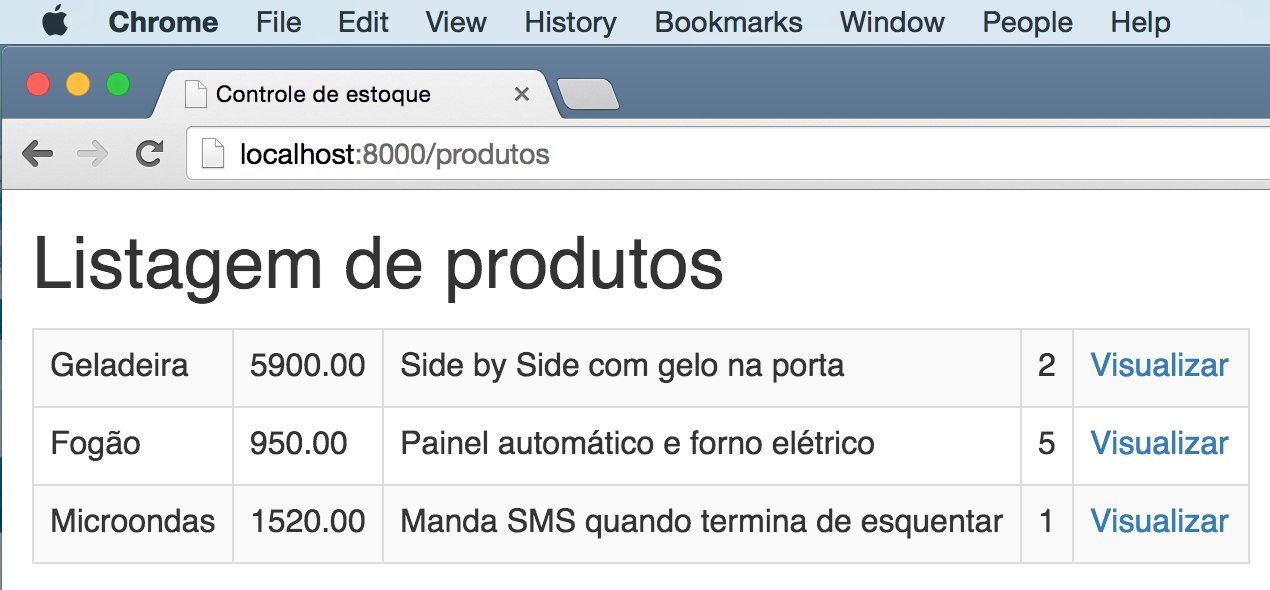
</a>

</td>

</tr>

<?php endforeach ?>

</table>/produtos/mostra>



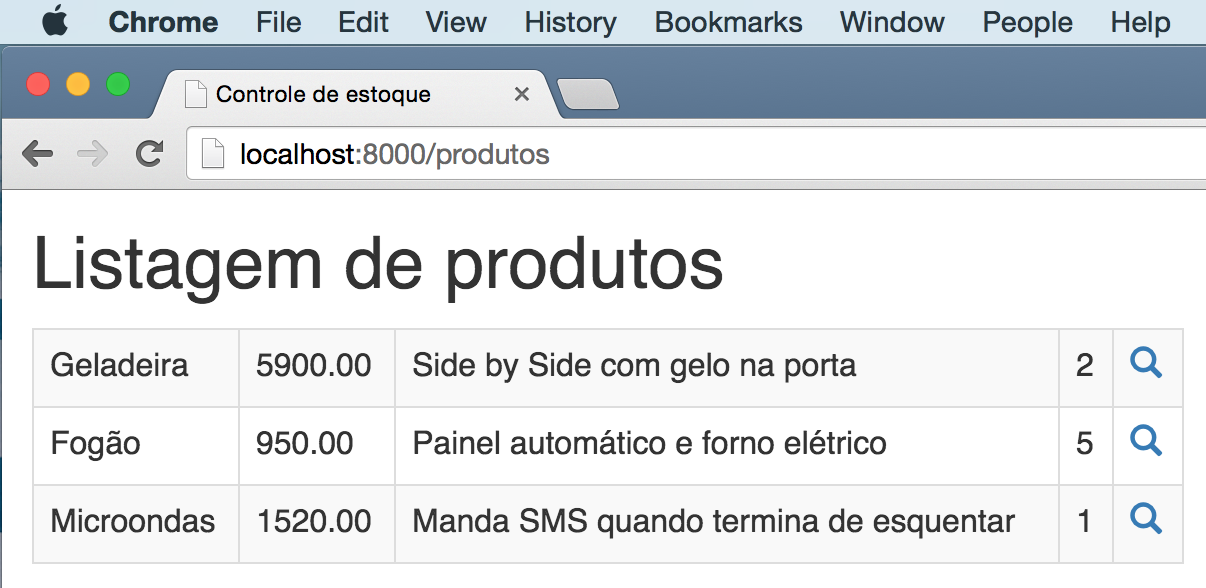
Ou, para deixar a aparência um pouco mais interessante, podemos usar um ícone de lupa no lugar desse texto. Como estamos usando ::bootstrap::, que já vem com o framework, só precisamos adicionar um ::span:: com a classe do ícone que queremos utilizar. Veja como é simples:

<a href="/produtos/mostra">

<span class="glyphicon glyphicon-search"></span>

</a>

Pronto, com isso nossa listagem ficou assim:



**Criando uma nova rota**

Veja que o link já aponta para a URL /produtos/mostra, que ainda não existe em nosso sistema. Vamos criá-la? Só precisamos acessar o arquivo routes.php e, da mesma forma como fizemos para a listagem, vincular essa URL a um método de nosso controller. O código ficará assim:

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

Agora, no ProdutoController, criamos o método mostra:

public function mostra(){

// retorna uma view com os detalhes

}

Você já deve imaginar qual o próximo passo, não é? Queremos retornar uma view com os dados do produto, portanto, vamos criar um novo arquivo chamado detalhes.php, por enquanto em branco, dentro da pasta/resources/views/.

Em nosso controller, precisamos adicionar o retorno do método mostra, que deve renderizar essa nova view de detalhes do produto:

public function mostra(){

return view('detalhes');

}

Até aqui nada novo, certo? Mas agora entraremos em um ponto interessante. A view, neste caso detalhes.php, precisa das informações do produto clicado para exibi-las em seu HTML. Algo muito parecido com o que fizemos na listagem, porém no lugar de fazer um forEach na lista de produtos, teremos só um:

<h1>Detalhes do produto: <?= $p->nome ?> </h1>

<ul>

<li>

<b>Valor:</b> R$ <?= $p->valor ?>

</li>

<li>

<b>Descrição:</b> <?= $p->descricao ?>

</li>

<li>

<b>Quantidade em estoque:</b> <?= $p->quantidade ?>

</li>

</ul>

Veja que utilizei um ul do HTML pra exibir as informações em formato de lista, mas fique à vontade para montar o HTML da forma que preferir. Utilizando a mesma estrutura da nossa view de listagem, incluindo ::header:: com ::css::, título etc., nosso arquivo detalhes.php completo pode ficar assim:

<html>

<head>

<link href="/css/app.css" rel="stylesheet">

<title>Controle de estoque</title>

</head>

<body>

<div class="container">

<h1>Detalhes do produto: <?= $p->nome ?> </h1>

<ul>

<li>

<b>Valor:</b> R$ <?= $p->valor ?>

</li>

<li>

<b>Descrição:</b> <?= $p->descricao ?>

</li>

<li>

<b>Quantidade em estoque:</b> <?= $p->quantidade ?>

</li>

</ul>

</div>

</body>

</html>

Agora, em nosso controller, precisaremos deixar a variável $p acessível na view. Não tem erro, afinal, já fizemos isso na listagem também! Só precisamos buscar o elemento do banco de dados e depois usar o método with para disponibilizá-lo no detalhes.php. Algo como:

public function mostra(){

$resposta = // busca do banco

return view('detalhes')->with('p', $resposta);

}

A busca será um ::select:: normal, que pode ser feito com o DB:select que já conhecemos, mas diferente de quando queremos listar todos os elementos, agora precisaremos passar o id do produto como parâmetro. Ele deve ficar assim:

$id = 1; // precisamos pegar o id de alguma forma

$resposta = DB::select('select \* from produtos where id = ?', [$id]);

Por enquanto, estamos passando o número 1 como id fixo, logo resolveremos isso. Veja que usamos um sinal de interrogação para mostrar o local onde o parâmetro passado no segundo argumento do método select (que é um array) deve ser aplicado.

**Não vamos focar no SQL... agora**

Não se preocupe em entender a fundo o SQL e sintaxe do DB:select agora, pois teremos uma aula dedicada a isso e também veremos uma forma bem mais interessante de fazer esse tipo de consulta sem nos preocuparmos tanto em entender esses detalhes.

Outra informação importante é que o método select sempre retorna um array, que neste caso pode:

* **ter 1 elemento**, caso seja encontrado;
* **ser vazio**, caso contrário.

Precisamos portanto verificar se o array de resposta é vazio, mostrando uma mensagem de erro, ou caso contrário incluir o elemento da primeira posição na view. Vale lembrar que a primeira posição de um array é o índice zero, portanto o código fica assim:

public function mostra(){

$id = 1; // precisamos pegar o id de alguma forma

$resposta = DB::select('select \* from produtos where id = ?', [$id]);

if(empty($resposta)) {

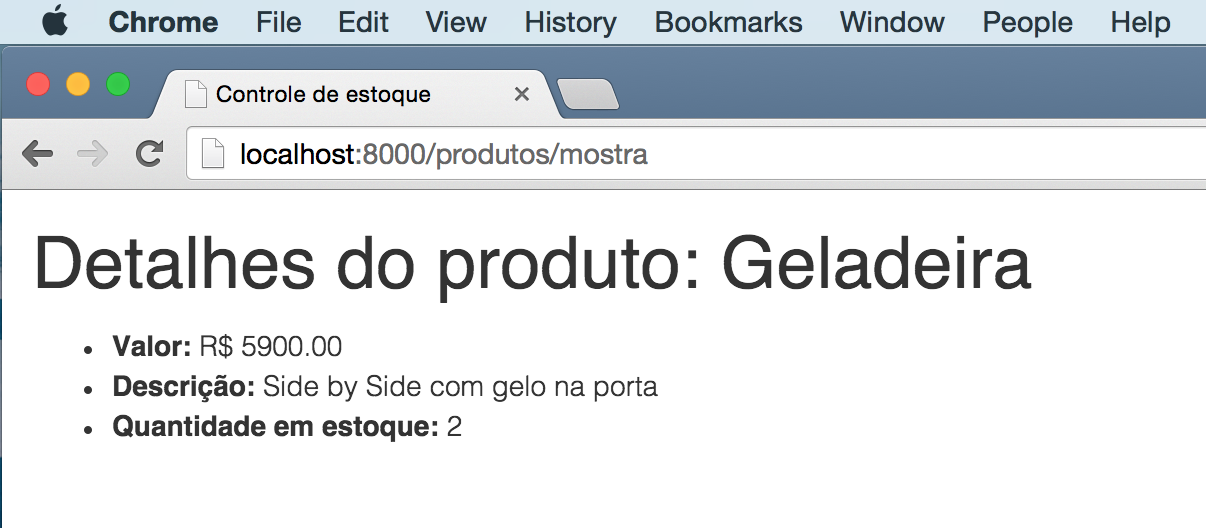
return "Esse produto não existe";

}

return view('detalhes')->with('p', $resposta[0]);

}

Vamos testar? Basta clicar na lupa de detalhes de qualquer produto da listagem, o resultado deverá ser parecido com:



Excelente, já sabemos que nosso controller e view estão funcionando como esperado, o SQL foi executado, o elemento com id 1 listado e o HTML foi exibido corretamente. Mas claro, independente do produto que eu clicar, sempre estamos mostrando os dados do produto com id 1, que no meu caso é a ::Geladeira::. Como resolver?

**Enviando parâmetros na request**

Como saber exatamente qual o id do produto que foi clicado no HTML? Uma forma bem tradicional de fazer isso seria passando o id do produto como um parâmetro na requisição, para isso bastaria mudar o link da listagem como a seguir:

<a href="/produtos/mostra?id=<?= $p->id ?>">

<span class="glyphicon glyphicon-search"></span>

</a>

Repare que logo após a URL adicionamos um ?id=<?= $p->id ?>, ou seja, estamos passando um parâmetro na requisição chamado id, com o valor do id do produto. Fazendo a alteração, você consegue verificar no código-fonte da listagem que os valores são substituídos e os links ficam assim:



**Recuperando parâmetros da request**

Agora que o parâmetro id está sendo enviado na requisição, podemos recuperar seu respectivo valor em nosso controller com uso da interface Request. Por sinal, esse tipo de interface estática é conhecida como ::facade::. Elas nos oferecem uma forma simples de acessar algumas das classes fundamentais do framework. Se quiser, você pode ler mais sobre esses facades em:

<http://laravel.com/docs/5.0/facades>

A interface Request tem métodos que nos auxiliam com esse trabalho, como o input, em que você passa a chave (::name::) do parâmetro que foi enviado na requisição e ele retorna o valor, ou ::null:: caso não seja encontrado.

Portanto, podemos fazer algo como:

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

Simples, não é? Você ainda pode passar um valor default, caso o parâmetro não seja encontrado. Um exemplo seria:

$id = Request::input('id', '0');

Agora o $id terá o valor passado, ou zero, caso nenhum valor de id tenha sido enviado na requisição. Vamos fazer essa alteração? Nosso método mostra pode ficar assim:

public function mostra(){

$id = Request::input('id', '0');

$resposta = DB::select('select \* from produtos where id = ?', [$id]);

if(empty($resposta)) {

return "Esse produto não existe";

}

return view('detalhes')->with('p', $resposta[0]);

}

Vale lembrar que, como estamos em um namespace diferente, temos que importar o Request no início de nosso controller, assim como fizemos com a interface DB:

<?php namespace estoque\Http\Controllers;

use Illuminate\Support\Facades\DB;

use Request;

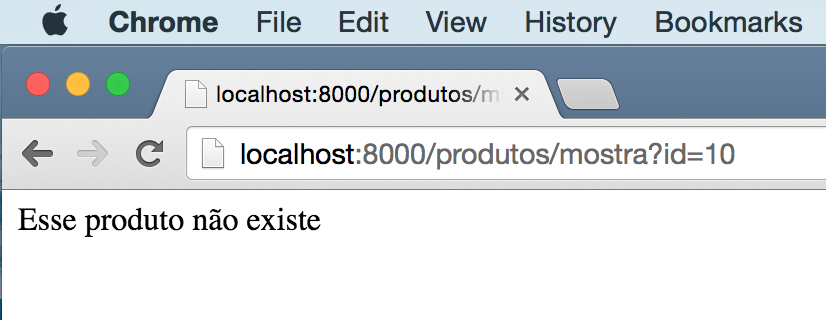
class ProdutoController extends Controller {

// ...

Tudo pronto, podemos voltar para o navegador e acessar o link de detalhes novamente que, desta vez, os detalhes do produto correto serão exibidos:



Veja que dessa vez eu selecionei o fogão, que tem id 2, e seus dados foram exibidos conforme esperado. Também podemos testar com um id que não existe, mudando o final da URL pra 10, por exemplo:



Excelente, a mensagem de erro do nosso if foi exibida como planejado.

**Conhecendo um pouco mais da Request**

Vimos como é simples recuperar os parâmetros da requisição com o método input, passando ou não um valor default. Mas além dele, a Request tem uma variedade bem grande de métodos que nos ajudam em trabalhos como esse. Por exemplo, quer saber se existe um parâmetro específico na requisição? Que tal fazer:

if (Request::has('id'))

{

// faz alguma coisa

}

Há ainda o método all, que retorna um array com todos os parâmetros da requisição, os métodos only eexcept, com que você pode restringir quais parâmetros quer listar.

// lista todos os params

$input = Request::all();

// apenas nome e id

$input = Request::only('nome', 'id');

// todos os params, menos o id

$input = Request::except('id');

Há também métodos como url, que retorna a URL da request atual, ou o path que retorna a URI. Por exemplo, em uma requisição para o método mostra, ao fazer:

$url = Request::url();

O valor da $url seria http://localhost:8000/produtos/mostra, mas já no caso do path:

$uri = Request::path();

O valor da $uri seria produtos/mostra.

Vamos usar mais alguns desses métodos no decorrer das aulas, mas se desde já quiser conhecer um pouco mais e dominar as possibilidades da interface Request, você pode dar uma olhada na sua documentação em:

<http://laravel.com/docs/5.0/requests>

**URLs mais semânticas**

Nossa estratégia atual de exibir o produto já resolve o problema, mas há uma alternativa bastante elegante que é adotada em situações como a nossa, onde queremos exibir detalhes de um recurso específico de nosso projeto. A ideia é que, no lugar de passar o parâmetro pela ::request::, dessa forma:

/produtos/mostra?id=2

você use o id como parte de sua rota, tendo uma URL mais semântica:

/produtos/mostra/2

Essas duas alternativas são conhecidas como parâmetro de busca (::query param::) e parâmetro de rota (::path param::), respectivamente. Normalmente a segunda estratégia é utilizada quando o parâmetro é obrigatório, como é o nosso caso, mas pode depender bastante do gosto do desenvolvedor.

Que tal aplicarmos essa alteração em nosso código? Não vai dar muito trabalho. Podemos começar pelo link, como de costume. No arquivo de listagem, no lugar de fazer:

<a href="/produtos/mostra?id=<?= $p->id ?>">

<span class="glyphicon glyphicon-search"></span>

</a>

Vamos mudar para:

<a href="/produtos/mostra/<?= $p->id ?>">

<span class="glyphicon glyphicon-search"></span>

</a>

Veja que só tiramos o ?id= e adicionamos uma barra no lugar. Simples, não é? Mas a alteração vai ser um pouco maior em nosso arquivo de rotas, o routes.php. Atualmente, a rota está assim:

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

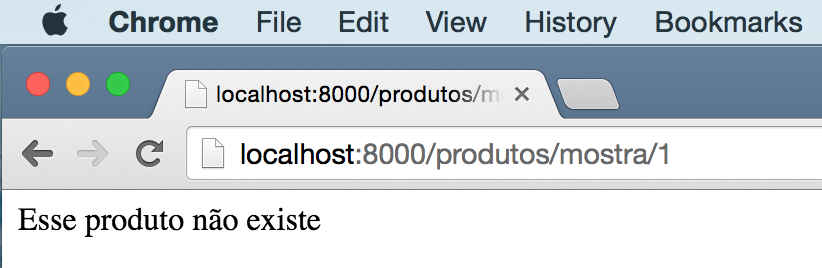
Mas, na verdade, agora queremos algo como:

Route::get('/produtos/mostra/1', 'ProdutoController@mostra');

Bem, já sabe o problema, não é? O id não pode ser fixo, se não a URI sempre seria /produtos/mostra/1. O id precisa ser uma variável! Veja como isso é feito:

Route::get('/produtos/mostra/{id}', 'ProdutoController@mostra');

É só isso, basta envolvermos o nome do parâmetro em um par de chaves e pronto! A rota já está feita. E agora, o que falta? Vamos testar para ver se está tudo ok? Basta abrir a listagem e clicar no ícone de detalhes de qualquer produto.



Ops, o produto não existe! Isso está acontecendo com todos os produtos, o que fizemos de errado?

**Recuperando parâmetros da URL**

O problema está no método do controller, que atualmente usa o Request:input para recuperar o valor do id:

public function mostra(){

$id = Request::input('id', '0');

$resposta = DB::select('select \* from produtos where id = ?', [$id]);

if(empty($resposta)) {

return "Esse produto não existe";

}

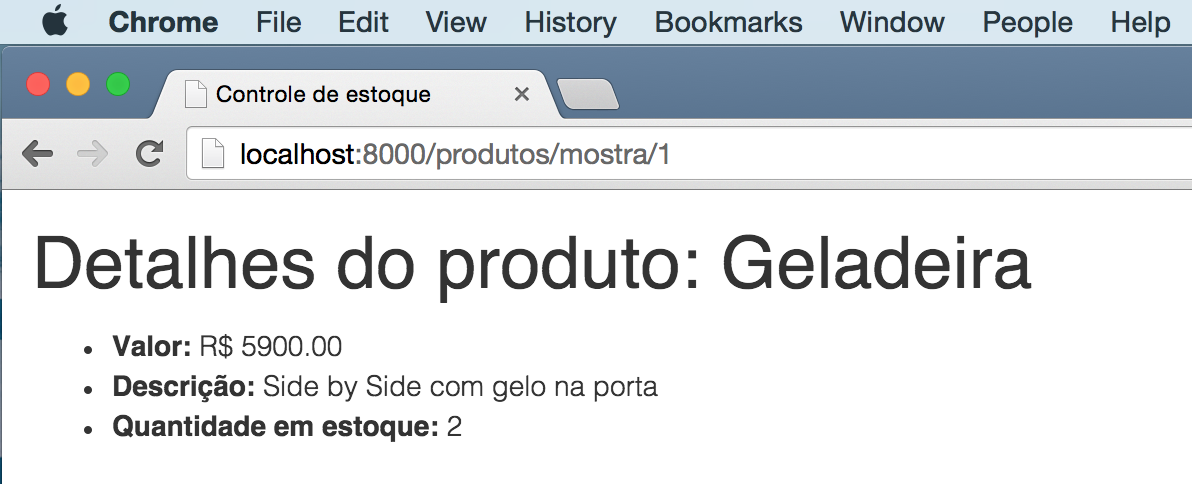
return view('detalhes')->with('p', $resposta[0]);

}

Como você já deve estar imaginando, o método input não busca parâmetros da URL, como é o caso do nosso idagora. No lugar disso, experimente mudar para:

$id = Request::route('id');

Agora sim, o resultado será o esperado!



Mas espera, e quanto ao valor default? Quando estávamos usando o método input, nosso código estava assim:

$id = Request::input('id', '0');

Por que não fizemos o mesmo com o route? A resposta é simples: o {id} da rota agora é obrigatório! Se não passar, não entra no método e ponto-final. Se você quiser que o id do final da url seja opcional, você precisará deixar isso explícito no momento de registrar sua rota.

Route::get('/produtos/mostra/{id?}', 'ProdutoController@mostra');

Repare que há uma ? após o id indicando que ele é opcional. Mas, claro, em nosso caso não queremos isso. O id sempre precisará ser passado.

Isso não é tudo, o Laravel nos oferece uma forma ainda mais legal de recuperar parâmetros da URL. Quer ver como? Basta adicionar um argumento **com o mesmo nome do parâmetro** na assinatura do seu método, ele vai ser populado como um passe de mágica. Veja como fica o método do controller:

public function mostra($id){

$resposta = DB::select('select \* from produtos where id = ?', [$id]);

if(empty($resposta)) {

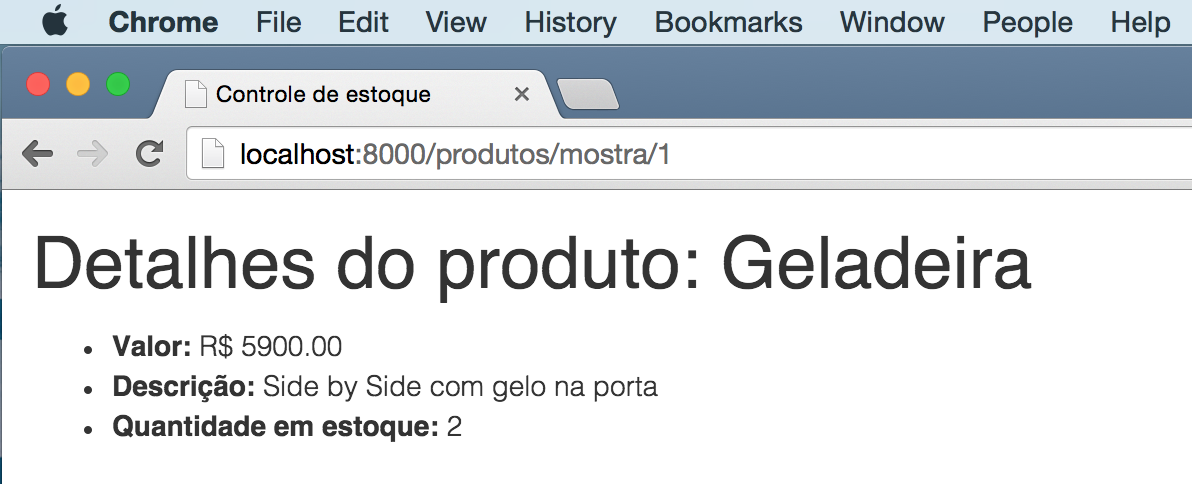
return "Esse produto não existe";

}

return view('detalhes')->with('p', $resposta[0]);

}

Note que em nenhum momento precisamos usar a Request, o framework faz esse trabalho por nós. Abra a página novamente para testar, o resultado será o mesmo:



**Alguns cuidados necessários**

Quando estamos usando parâmetros na URL, sempre precisamos nos atentar a alguns detalhes. Por exemplo, em algum momento falamos que o id precisaria ser um número? Não. Isso significa que eu consigo acessar o métodomostra pela seguinte URL:

http://localhost:8000/produtos/mostra/teste

Claramente isso não deveria acontecer. Por nossa sorte, a única consequência aqui será ver a mensagem de que o produto não foi encontrado. Mas existem problemas piores, como ambiguidade entre rotas.

Em nosso caso, isso seria um pouco difícil, mas imagine que para simplificar optássemos por remover o /mostra de nossa URL. Isso causaria sérios problemas, pois as seguintes URLs seriam equivalentes:

* <http://localhost:8000/produtos/1>
* <http://localhost:8000/produtos/adiciona>

Já que o {id} pode ser qualquer coisa, inclusive um texto, ele pode ser a palavra adiciona. Em outras palavras, dependendo da ordem em que eu registrar as rotas, pode acontecer de eu acessar /produtos/adiciona e a aplicação me responder que esse produto não existe... Ops!

Nesse caso, precisaríamos de alguma forma ensinar ao Laravel que o {id} da rota sempre será um número. Isso pode ser feito com auxilio do método where, como no exemplo:

Route::get(

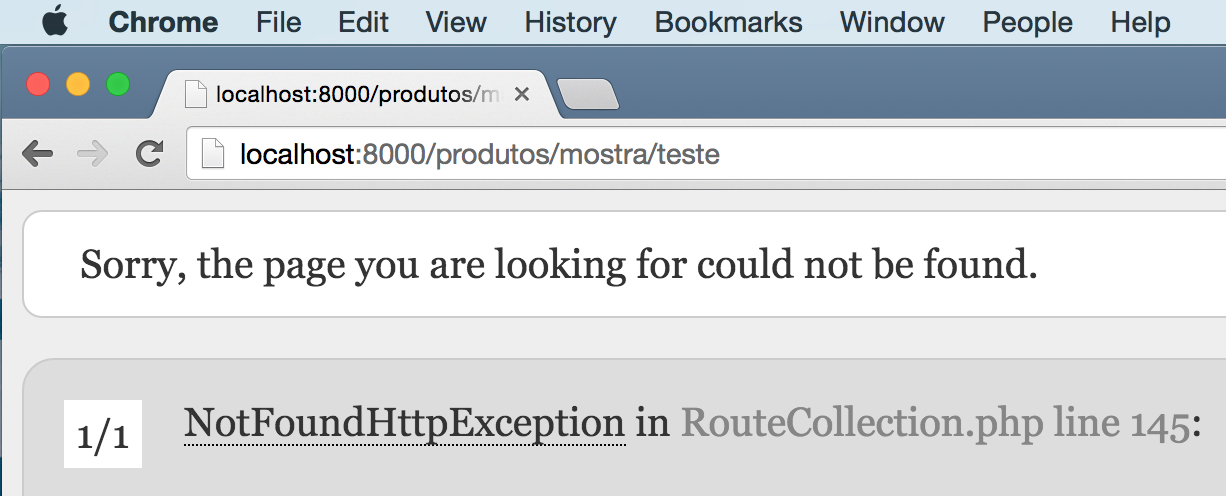
'/produtos/mostra/{id}',

'ProdutoController@mostra'

)

->where('id', '[0-9]+');

Observe que estamos passando para o método where o nome do parâmetro e uma expressão regular (::regex::) com o ::pattern:: que pode ser seguido. Faça a alteração e tente acessar a URL<http://localhost:8000/produtos/mostra/teste> novamente. O resultado dessa vez será umaNotFoundHttpException.



Justo, não é? Afinal, essa URL realmente não existe. Repare que, passando um número válido, tudo continua funcionando.