**Novo projeto com Laravel**

**Novo projeto com Laravel**

Nosso projeto será de controle de estoque. Como usuário final, seremos capazes de gerenciar os produtos que serão persistidos em um banco de dados MySQL, visualizar com facilidade os que estão em falta no estoque, e mais. O contexto é simples, mas será uma boa base para explorar os poderosos recursos e facilidades que o Laravel oferece.

**Como criar cinco telas de uma aplicação web?**

Imagine uma aplicação que tenha as funções de adicionar, remover, listar, enviar e-mail, entre diversas outras que são essenciais para toda aplicação web. Para cada uma, devemos executar um código de lógica, buscar ou atualizar informações do banco, mostrar um HTML como resposta. Bastante coisa repetitiva, não é? Será que alguém não pode nos ajudar?

**Framework, pra que te quero?**

Independente da linguagem ou tecnologia que estamos usando, um conceito global é: **não queremos ficar nos preocupando com infraestrutura**. É aí que os frameworks entram. Eles nos ajudam e muito a agilizar o processo de desenvolvimento, de forma organizada, evitando repetições de código e muito mais.

Quem nunca foi criar um projeto novo e acabou copiando a estrutura de algum outro projeto que já tinha criado antes? Isso acontece porque boa parte dessa estrutura será igual, você não precisa reinventar a roda a cada novo projeto. Essa é uma das ideias dos frameworks, oferecer essa estrutura padrão entre os projetos, de forma bem organizada e fácil de manter, segundo as melhores práticas do mercado. Essa reutilização de código entre vários projetos vai lhe poupar muito tempo e trabalho. Precisa conectar com o banco? Enviar um e-mail? Migrar seu banco de dados? Você perceberá que o *Laravel*, assim como diversos outros frameworks do mercado, já tem tudo isso pronto e pré-configurado.

Ao longo desse curso você perceberá que não precisa usar frameworks, não é obrigatório. Mas mesmo assim você não vai querer mais viver sem eles, que nos tornam muito mais produtivos e simplificam bastante o processo de desenvolvimento.

**Algumas outras opções famosas**

Além do *Laravel*, que vamos aprender, existem diversas opções bastante interessantes no mercado. Uma das mais populares é o *Zend Framework 2*, da própria **Zend Technologies**. Além dele, há também o *CodeIgniter*, *Symphony*,*CakePHP*, *Phalcon*, entre diversos outros.

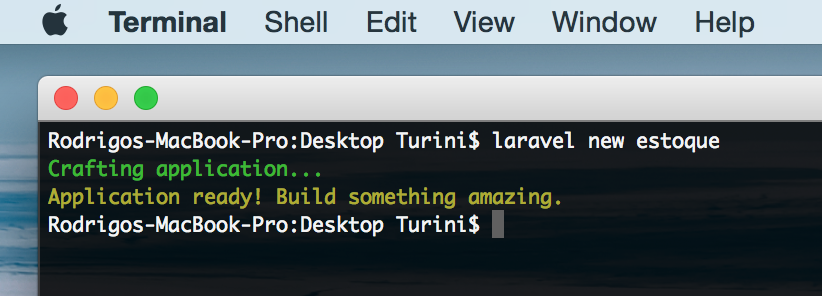
O *Laravel* é uma das maiores apostas da atualidade. Muito se deve à sua simplicidade, sintaxe, flexibilidade e rica documentação. Além de seu site oficial, que é o [http://laravel.com](http://laravel.com/), você também conta com bastante conteúdo, discussões, exemplos de código, perguntas e respostas em um site de receitas, o <http://laravel-recipes.com/>. Eu recomendo que, além do curso, você use e abuse desses sites para dominar completamente a ferramenta.

**Novo projeto: Uma app de controle de estoque**

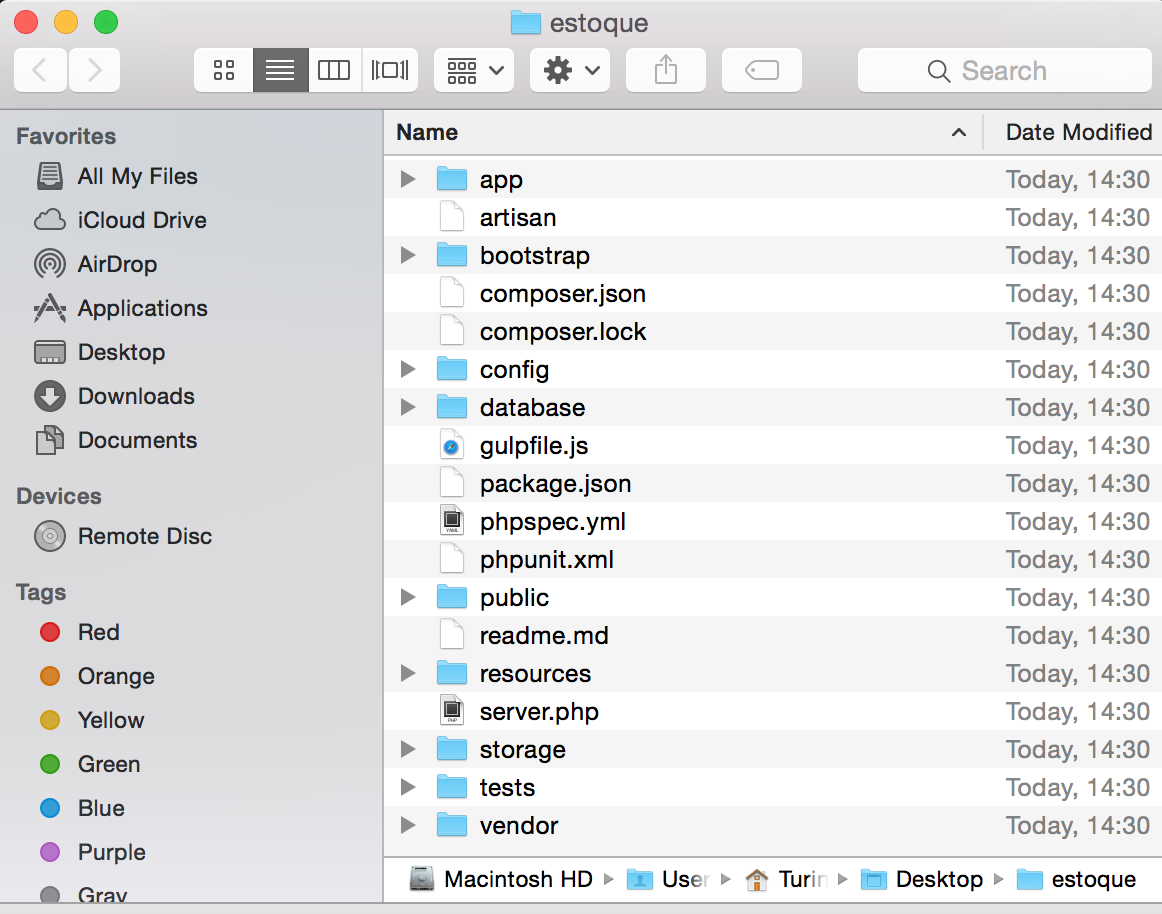
Para começar a aprender Laravel, vamos criar um novo projeto chamado estoque. Fazer isso é verdadeiramente simples, desde o começo você já percebe as vantagens em usar o framework. Quer ver como é fácil? Se você já tem o Laravel instalado, tudo que precisa fazer é rodar o comando laravel new pelo terminal, passando o nome do projeto que queremos criar. Em nosso caso será:

laravel new estoque

A saída será parecida com:

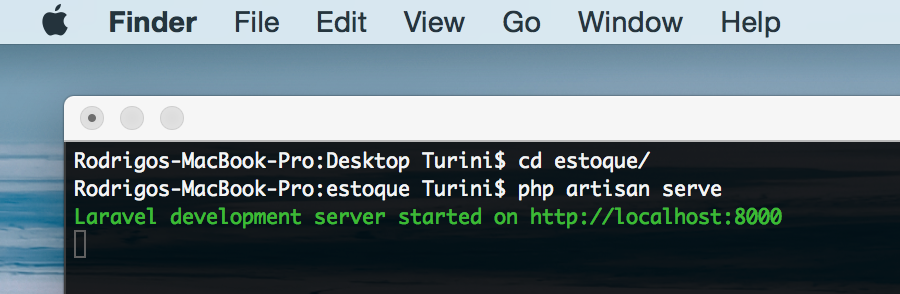


O texto *Application ready! Build something amazing* será exibido e pronto, ele cuidou de todo o trabalho pesado. Note que uma pasta com o nome do projeto (estoque, neste caso) foi criada no mesmo diretório em que você executou o comando. Ela já tem toda a estrutura de pastas, algumas classes e dependências configuradas.



Legal, não é? Se você está se perguntando o que significa cada uma dessas pastas, não se preocupe: em breve vamos falar mais sobre essa estrutura inicial e a entenderemos mais a fundo no decorrer do curso. Logo você estará dominando tudo isso. Mas por agora, vamos rodar o projeto estoque para garantir que tudo está funcionando conforme o esperado?

Normalmente usamos o próprio *Apache* do servidor em que fizermos *deploy*, mas em ambiente de desenvolvimento, podemos servir nossa aplicação utilizando o comando php artisan serve, que executa a aplicação no servidor de desenvolvimento do PHP. Basta rodar esse comando de dentro da pasta do projeto:



**Navegando nas pastas do OS pelo terminal**

Você pode utilizar o comando cd (*change directory*) para navegar pelas suas pastas via terminal. Um exemplo em Windows seria:

D:\> cd Desktop\estoque

D:\Desktop\estoque>

O mesmo comando pode ser utilizado em um ambiente Unix (Linux ou Mac OS). Para listar, há uma diferença. Em Windows utilizamos o comando dir:

D:\Desktop\estoque> dir

// vai mostrar todos os arquivos

Porém, nos outros sistemas que foram citados o comando será ls. Repare:

turini ~ $ cd Desktop/estoque

turini/Desktop/estoque ~ $ ls

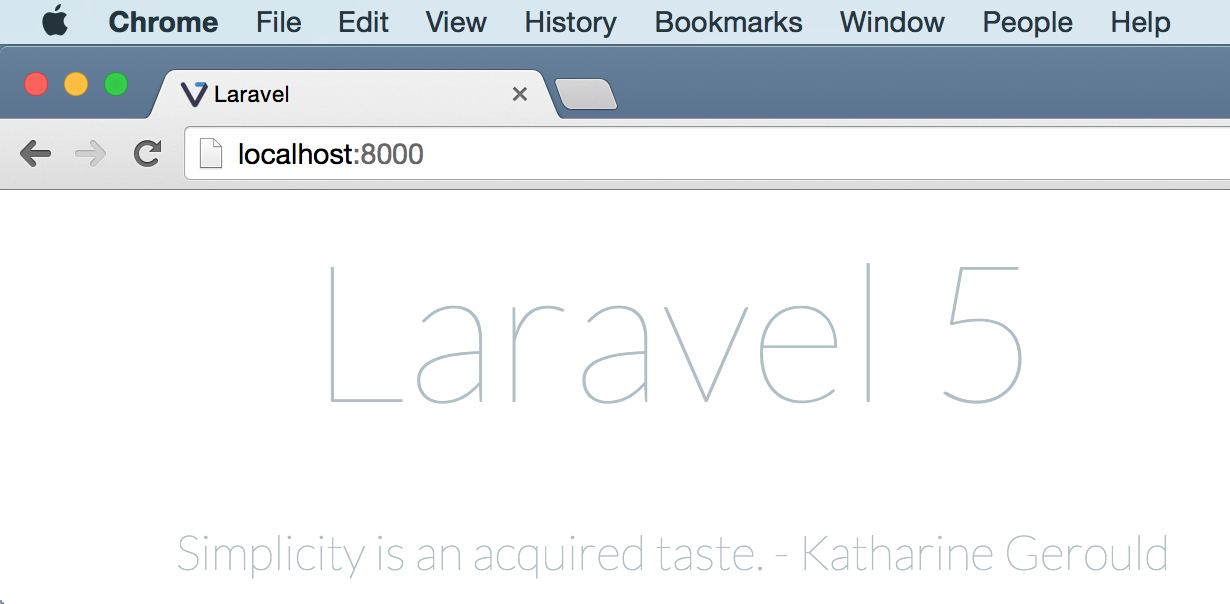
// vai mostrar todos os arquivos

Considerando, portanto, que meu projeto se chama estoque e foi criado dentro da pasta Desktop, para executar o comando que inicia o servidor você precisará executar os dois comandos:

cd Desktop/estoque/

php artisan serve

Se tudo correu bem, a mensagem Laravel development server started on <a href="http://localhost">http://localhost</a>:8000 deve ter aparecido no seu terminal. Vamos testar? Basta acessar essa URL no navegador de sua preferência. O resultado será:



Note que, para criar um projeto e executar o Laravel, nós não precisamos de nenhuma configuração extra. Basta criar o projeto em alguns segundos, com o comando laravel new, e sair usando! Isso é muito conveniente quando estamos começando um novo projeto. Podemos gastar nosso tempo com o que realmente importa: nossas regras de negócio.

Precisa fazer alguma configuração adicional? O framework faz todo o possível para que você não precise ficar configurando nada, mas em alguns momentos isso pode ser necessário. Se algo de que você precisa não está configurado por *default*, como por exemplo o locale, você pode fazer isso facilmente pelo arquivo de configurações presente em estoque/config/app.php.

Quer saber mais sobre essas configurações adicionais? Talvez você queira dar uma olhada na página de configurações do framework, disponível em:

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

Mas, por enquanto, não precisaremos de nenhuma configuração adicional em nosso projeto. Se ele executou sem nenhum problema, já estamos prontos para prosseguir.

**Entendendo a estrutura de pastas**

Como vimos, ao criar um novo projeto, diversas classes e arquivos foram criados. O objetivo dessa estrutura inicial é oferecer um padrão e o mínimo de esforço possível para começar o seu projeto. Isso é bem legal, mas claro, caso preferir, você também pode renomear as classes e mudar a estrutura de pastas para ficar como melhor lhe agrada, o framework não impõe muitas restrições quanto a isso.

Durante todo o curso conheceremos um pouco mais sobre as pastas e essa estrutura inicial, mas desde já podemos ter uma breve noção do que vai em cada lugar. Essas são algumas das principais pastas:

* **app**: nela ficam seus **m**odelos, **v**iews e **c**ontrollers, que serão bem detalhados no próxima aula. Em poucas palavras, é onde boa parte do seu código vai ficar. Ela possui uma série de subdiretórios, como Commands,Console, Http, Events, entre outros. Não se preocupe em entender o significado de cada um deles agora, vamos vê-los melhor conforme formos precisando.
* **config**: como o nome já indica, é onde ficam os arquivos de configuração do seu projeto. Se você precisar alterar as configurações de cache, e-mail, banco de dados, entre outras, já sabe onde encontrar.
* **public**: é a pasta pra onde seu web server vai apontar. Lá fica o arquivo index.php, que aponta para sua aplicação. Além disso, é comum colocarmos os arquivos css, imagens, javascript e todos os demais arquivos públicos nesse diretório.
* **vendor**: é onde fica o *source code* do Laravel, plugins e outras dependências. Tudo que você usar de terceiros (bibliotecas, frameworks etc.) deve ficar nela.

Esse é só um pouco, claro, mas já é o bastante por agora. Vamos entrar mais a fundo nesse conteúdo no decorrer do curso, mas se quiser adiantar, talvez queira dar uma olhada na página do Laravel que explica essa estrutura inicial:

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

**Alterando o namespace padrão com Artisan**

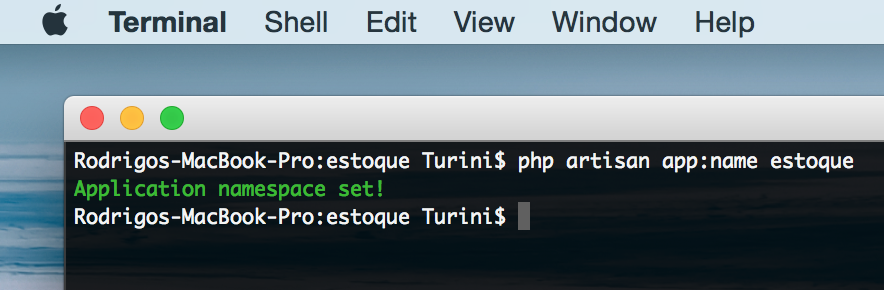
O *namespace* padrão de toda aplicação com Laravel é App, mas é muito comum e bastante recomendado que você altere o namespace para o nome da sua aplicação. Como fazer isso? É muito fácil, basta rodar um simples comando e pronto.

Pra subir o server e testar, usamos o php artisan serve, lembra? Esse **Artisan** é uma ferramenta de linha de comando já inclusa no framework. Ela nos oferece uma série de comandos úteis para tornar nosso desenvolvimento mais produtivo.

Para mudar o *namespace*, por exemplo, podemos usar o php artisan app:name. Vamos mudá-lo para **estoque**, que é o nome do projeto. Basta executar o seguinte comando pelo terminal, dentro da pasta de seu projeto:

php artisan app:name estoque

A saída será parecida com:



Tudo pronto, *namespace* alterado!

**Criando nossa primeira lógica**

Agora que já conhecemos um pouco mais sobre o Laravel, queremos ensiná-lo como queremos que ele reaja quando alguém acessar determinada URL, isto é, criar as nossas próprias rotas. Mas quão complicado é fazer isso?

Quando acessamos <http://localhost:8000/>, ou seja, a URL / da nossa aplicação, em algum lugar foi configurado que a página padrão do Laravel deveria ser exibida, não é? Esse trabalho é feito no o arquivo routes.php, que fica dentro da pasta app/Http/. Abra o arquivo para conferir seu conteúdo, que deve estar parecido com:

<?php

// comentário omitido

Route::get('/', 'WelcomeController@index');

Route::get('home', 'HomeController@index');

Route::controllers([

'auth' => 'Auth\AuthController',

'password' => 'Auth\PasswordController',

]);

Como você pode ver, ele já vem com algumas rotas configuradas, como a / que nos leva para a página *default* do Laravel. O código pode parecer diferente no começo, mas não é complicado. Vamos entender em detalhes cada linha de código desse arquivo, mas antes disso, apague todo esse código para criarmos nossa própria rota. Que acha? É bem fácil, basta deixar seu arquivo vazio, só com a tag do php:

<?php

// nosso código vai aqui

E agora, usar o Route:get para definir uma nova rota. Podemos fazer algo como:

<?php

Route::get('/', function()

{

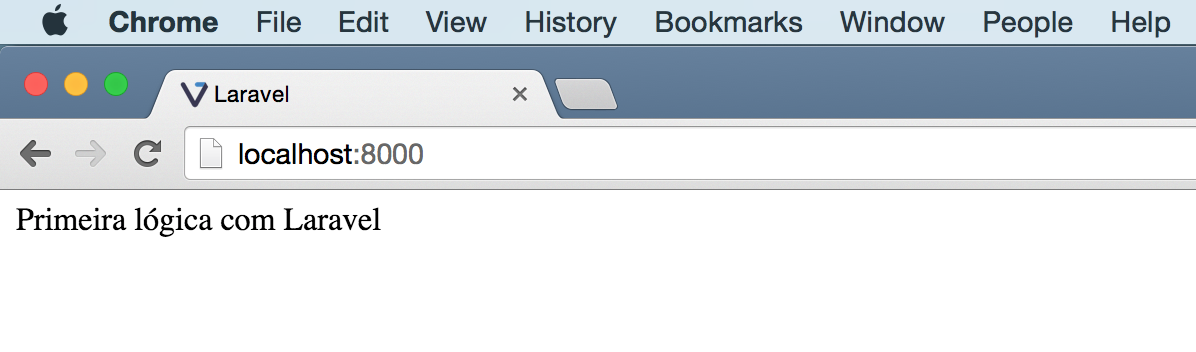
return 'Primeira lógica com Laravel';

});

Sucesso, já criamos nossa primeira rota. Vamos entender o que aconteceu?

Usamos o método estático get, da classe Route, passando dois parâmetros. O primeiro é a rota (caminho, ou *path*- como comumente é chamado) que será acessado pelo navegador. O segundo parâmetro é uma função com a resposta que deverá ser enviada. Em poucas palavras, esse código ensina que, quando alguém acessar a URL /, o Laravel deve retornar o texto Primeira lógica com Laravel para o navegador.

Vamos testar? Basta rodar o comando php artisan serve para subir o servidor novamente (caso ainda não esteja startado) e acesse <http://localhost:8000/> em seu navegador. Veja a resposta:



Perfeito, o texto foi exibido no navegador! Mandamos um texto comum, mas podemos responder com qualquer HTML válido. Por exemplo, envolvendo esse texto em um h1 para ter um destaque maior na página:

<?php

Route::get('/', function()

{

return '<h1>Primeira lógica com Laravel</h1';

});

Você não precisa restartar o servidor, basta acessar a página novamente para ver o resultado:



**Conflito entre rotas do Laravel**

É importante perceber desde já que você pode criar quantas rotas quiser no arquivo routes.php:

<?php

Route::get('/', function()

{

return '<h1>Primeira lógica com Laravel</h1';

});

Route::get('/outra', function()

{

return '<h1>Outra lógica com Laravel</h1';

});

Agora temos duas rotas, uma para a URL / e outra para /outra. Mas o que aconteceria se as duas fossem registradas na mesma URL? Na /, por exemplo.

<?php

Route::get('/', function()

{

return '<h1>Primeira lógica com Laravel</h1';

});

Route::get('/', function()

{

return '<h1>Outra lógica com Laravel</h1';

});

Tente rodar esse código para ver o resultado, a segunda rota vai sobrescrever a primeira e o texto Outra lógica com Laravel será exibido. Ou seja, em caso de ambiguidade sempre a última rota é quem será registrada. Há outras formas de lidar com ambiguidade, como quando usamos diferentes métodos HTTP, mas entraremos nesse assunto um pouco mais à frente.

Agora que já sabemos o essencial, podemos partir para as regras de negócio do nosso sistema de estoque. Está preparado?