

Laravel

Jaime R Vendrame Filho

 /jaimevendrame

 /in/jaimevendrame

 /jaime.vendrame

 jaime.vendrame@gmail.com

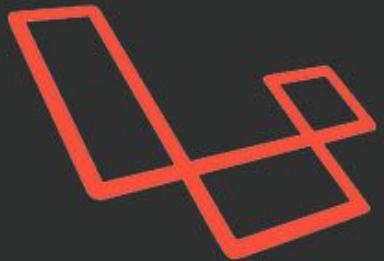


+55 44 99974-8008

Agenda

- Apresentação do Módulo
- Ambiente de Desenvolvimento
 - Requisitos
- Instalação e Configuração Laravel
 - Criando nosso Virtual Host
- Introdução ao Laravel
 - Estrutura de pastas
 - Routes
 - Controllers
 - Views & Templates
- Conhecendo projeto do Blog
- Integração layout Home & Dashboard
- Laravel avançado
 - Models & Migrations
- CRUD Usuários
- CRUD Categorias
- Auth (Autenticação personalizada)
- CRUD Post
- Site

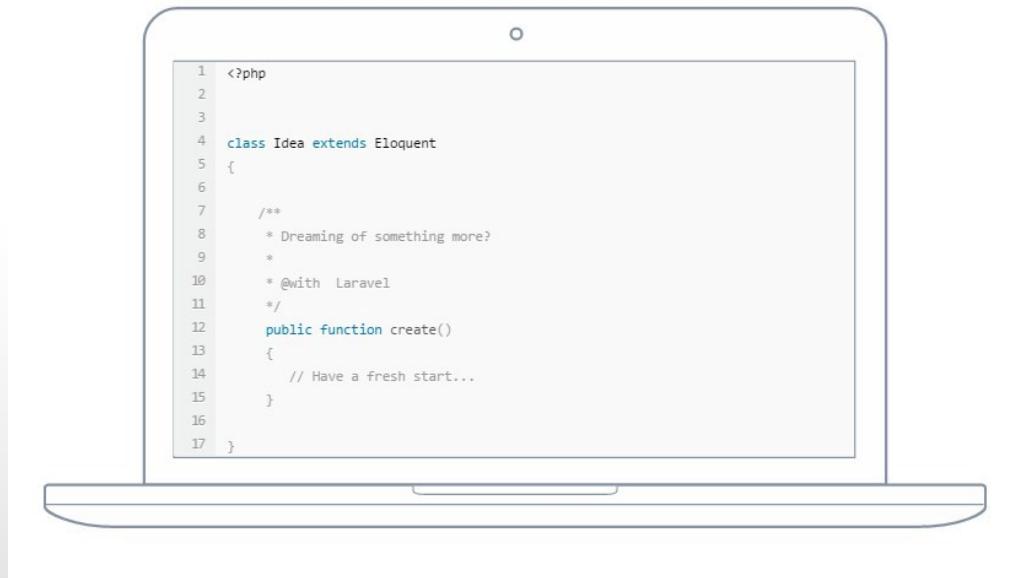
Apresentação do Módulo



laravel

Apresentação do Módulo

- O **Laravel** é um framework de desenvolvimento rápido para PHP, livre e de código aberto. Cuja o principal objetivo é permitir que você trabalhe de forma estruturada e rápida.
- **Gosta de código bonito?** Quem utiliza Laravel também gosta! Como o próprio slogan diz: “*O framework PHP para artesões da web*”.



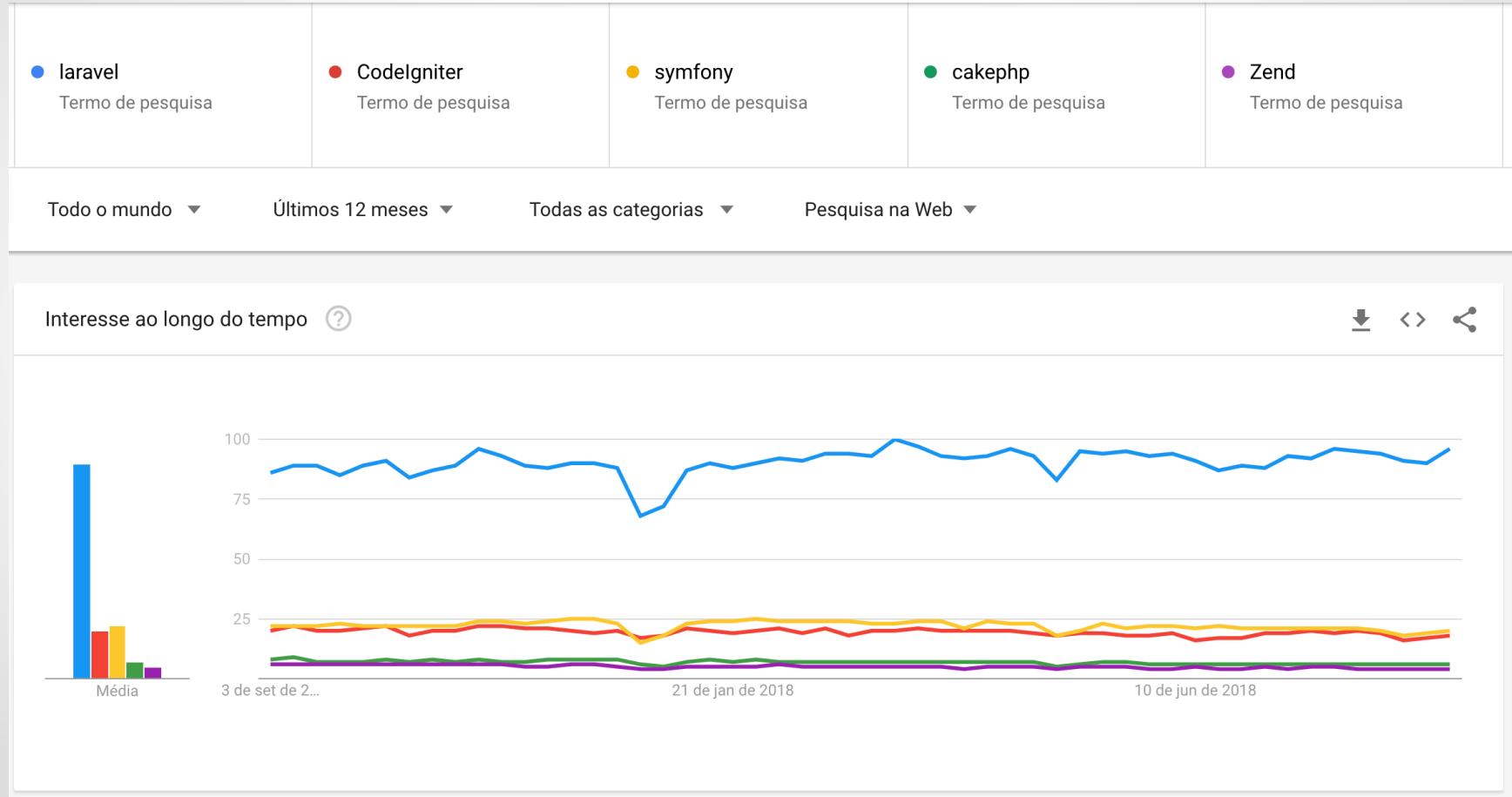
Apresentação do Módulo

- O Laravel tira a monotonia do desenvolvimento web. Ele fornece todas as ferramentas necessárias para que você possa começar programando o que for necessário, ele foi construído para ser simples e fácil de aprender.
- O Laravel é sem sombras de dúvidas, um dos frameworks mais populares da atualidade (*não somente no mundo PHP*), possui uma equipe de desenvolvedores ativa e extremamente competente, além de ter uma gigantesca comunidade e bastante aceitação no mercado.
- Com uma rápida pesquisa no [Google Trends](#), podemos comprovar a crescente popularidade do Laravel.

Apresentação do Módulo

- O Laravel tira a monotonia do desenvolvimento web. Ele fornece todas as ferramentas necessárias para que você possa começar programando o que for necessário, ele foi construído para ser simples e fácil de aprender.
- O Laravel é sem sombras de dúvidas, um dos frameworks mais populares da atualidade (*não somente no mundo PHP*), possui uma equipe de desenvolvedores ativa e extremamente competente, além de ter uma gigantesca comunidade e bastante aceitação no mercado.
- Com uma rápida pesquisa no [Google Trends](#), podemos comprovar a crescente popularidade do Laravel.

Apresentação do Módulo



Apresentação do Módulo

- **Mas, porque devo usar Laravel? O que torna ele tão bom assim?**
- Ok! Já sabemos que ele é o framework mais badalado da atualidade, para muitos, isso já é um baita argumento para começar a utilizá-lo, para outros, nem tanto... Sendo assim, vou apresentar alguns pontos:
- **Composer**
- O Laravel utiliza o [Composer](#) para gerenciar suas dependências, algo que praticamente toda aplicação PHP moderna faz. O Composer é um incrível gerenciador de dependências, uma ferramenta que permite gerenciar, de forma fácil, os pacotes de terceiros da sua aplicação.
- Para fazer a instalação, basta executar o comando no terminal:
composer create-project laravel/laravel
- **Documentação**
- Até na documentação, o Laravel se destaca! Ela é bastante intuitiva e você consegue encontrar de forma fácil praticamente tudo que precisa saber para começar e se aprofundar em todos os recursos disponíveis pelo framework.
- <http://laravel.com/docs>

Ambiente de Desenvolvimento

- Existem diversas opções para criação de um ótimo ambiente de desenvolvimento Laravel, como por exemplo:
 - Containers: Docker e Laradocker
 - Boxs: Vagrant e Homestead
 - Servidores Independentes: XAMP, MAMP, LAMP ...
- Você deve levar em consideração o seu sistema operacional e as limitações de seu hardware.

Ambiente de Desenvolvimento

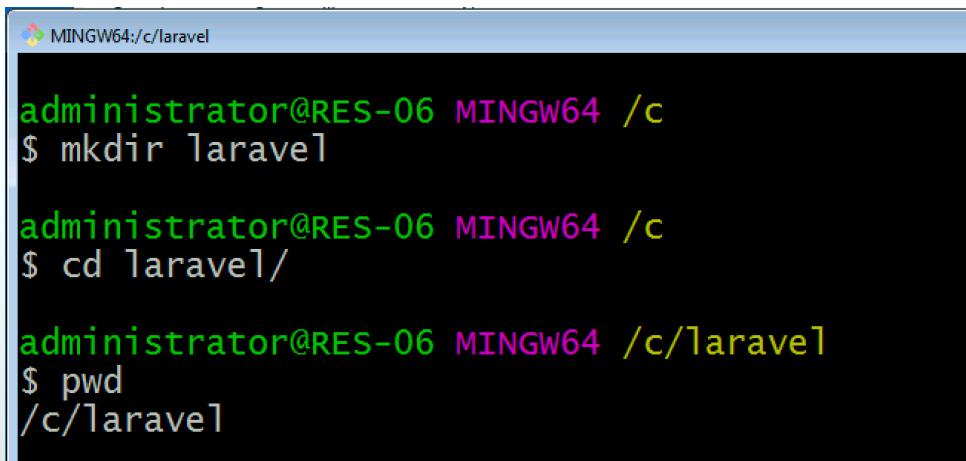
- Neste projeto utilizaremos o **VAGRANT**, por uma questão exclusivamente de padronização.
- **VAGRANT** é uma ferramenta que permite que criemos rapidamente ambientes virtuais para fazermos testes, desenvolvimento ou provisionamento de ambientes utilizando as soluções de virtualização mais comuns como o **Virtualbox** e o **VMWare**, sendo também compatível com os principais provedores cloud como AWS, **Rackspace** e **Digitalocean**.
- Outros aspecto bacana é que você tem a portabilidade de criar e recriar esses ambientes em qualquer lugar de forma simples e descomplicada bastando apenas ter a ‘receita mágica’ do vagrant.

Ambiente de Desenvolvimento

- **REQUISITOS:**
 - Vagrant <https://www.vagrantup.com/downloads.html>
 - VirtualBox <https://www.virtualbox.org/wiki/Downloads>
 - GIT <https://git-scm.com/download/>
 - VISUAL STUDIO CODE <https://code.visualstudio.com/>
 - PUTTY somente para usuários Windows.
<https://www.putty.org/>
- **Obs: Devem ser instaladas a ultima versão, desde que não exista outro ambiente desenvolvimento baseado em box instalando anteriormente.**

Ambiente de Desenvolvimento

- CRIANDO SUA WORKSPACE
 - Através seu terminal ou git-bash(windows) acessem a raiz de sua unidade disco e crie o diretório laravel.



```
MINGW64:/c/laravel
administrator@RES-06 MINGW64 /c
$ mkdir Laravel

administrator@RES-06 MINGW64 /c
$ cd Laravel/

administrator@RES-06 MINGW64 /c/laravel
$ pwd
/c/laravel
```

Ambiente de Desenvolvimento

- Configurações para criar o ambiente perfeito para o desenvolvimento web com PHP.
 - <https://github.com/jaimevendrame/vagrant-setup-php>

Ambiente de Desenvolvimento

- Clonando repositório:

```
administrator@RES-06 MINGW64 /c/laravel
$ git clone https://github.com/jaimevendrame/vagrant-setup-php.git webdev
```

- Rodando ambiente de desenvolvimento pela primeira vez.
 - Acesse o novo diretório webdev
 - Digite o comando: vagrant up
 - **Obs: O processo demora bastante na primeira vez.**
- Testando ...
 - Em seu browser acesse: <http://192.168.33.10> ou <http://localhost:8080>.

Instalação e Configuração Laravel

- Qual versão escolher do Laravel 5.x?
 - Quanto mais recente a versão mais recursos ela possui, e claro, melhor fica o desenvolvimento. Certo?
 - Em partes. Um ponto muito importante no momento de escolher uma versão é avaliar se ela é L.T.S., ou seja, se o tempo de suporte pra essa versão é longa. O Laravel atualmente possui 2 duas versões L.T.S, a 5.1 e a 5.5 ([veja a tabela de versões L.T.S. do Laravel 5.x](#))
 - **NOTA:** Em resumo, uma versão L.T.S. significa que a equipe responsável pelo projeto Laravel irá dar manutenção, ou seja, corrigir eventuais bugs e falhas de segurança.



O que é Composer

Dependency Manager for PHP

- **Gerenciador de dependências, o que é isso?**
- Imagine que você vai desenvolver uma aplicação web e em algum momento precisa gerar logs, por exemplo.
- O que fazer neste caso?
- Umas das alternativas é desenvolver a funcionalidade do zero, ou seja, criar o seu próprio sistema para gerenciar log. Porém isso seria trabalhoso e desnecessário, porque já existem inúmeros pacotes (bibliotecas) prontos que implementam essa funcionalidade.
- A alternativa mais simples quando se precisa de algo é verificar no [Packgist](#) se já existe algo pronto sobre isso, e se tiver algum pacote pronto você pode utilizar o mesmo, sem a necessidade de escrever tudo do zero, simplesmente aproveitando algo que outro desenvolvedor(a) já criou.
- Partindo do cenário onde você já encontrou algum pacote agora o que precisa é reutilizar o mesmo no seu projeto. Agora que entra a mágica do composer, pois através deste é possível baixar este pacote e todas as dependências do mesmo.
- Se um pacote depende de outro pacote para funcionar (algo comum) o próprio composer se encarrega de fazer o download tanto do pacote quanto dos pacotes requeridos para o funcionamento do pacote escolhido.
Vamos a um exemplo!
- Estamos desenvolvendo uma aplicação em algum momento precisamos de algo para gerar log, neste caso vamos utilizar um pacote pronto o monolog (um dos melhores). Neste caso vamos fazer o download deste pacote.



O que é Composer

- Como fazer o download de um pacote?
- Basta entrar no [packgist \(clique aqui\)](#) e pesquisar pelo pacote desejado “monolog”. Ao encontrar este pacote pode observar que o mesmo possui algumas dependências.
- php: >=5.3.0
- psr/log: ~1.0
- Quando rodar o comando sugerido para fazer o download do pacote, automaticamente o composer vai baixar tanto o monolog quanto os pacotes que este depende, como por exemplo psr/log

composer require monolog/monolog

Instalação e Configuração Laravel

- **Via Composer Create-Project**

Alternativamente, você também pode instalar o Laravel emitindo o comando Composer no seu terminal: **create-project**

```
composer create-project --prefer-dist laravel/laravel blog "5.5.*"
```

Instalação e Configuração Laravel

- **Criando nosso Virtual Host**

Com nosso projeto criado, agora precisamos criar e configurar nosso host virtual.

Primeiramente usando o Putty, abra o arquivo **000-default.conf**, que se encontra no diretório **/etc/apache2/sites-available**.

Instalação e Configuração Laravel

```
$> sudo nano /etc/apache2/sites-available/000-default.conf

<VirtualHost *:80>
    ServerName blog.webdev
    DocumentRoot /var/www/html/blog/public
        <Directory /var/www/html/blog/public>
            AllowOverride all
        </Directory>
</VirtualHost>
```

Ctrl X – Sair e Salvar
Y – Confirma
Enter

```
$> sudo nano /etc/hosts
127.0.0.1 blog.webdev
```

```
$> sudo service apache2 restart
```

Instalação e Configuração Laravel

- Configurar host local
- Windows ->
C:\Windows\System32\drivers\etc\hosts
- Mac/Linux -> /etc/hosts

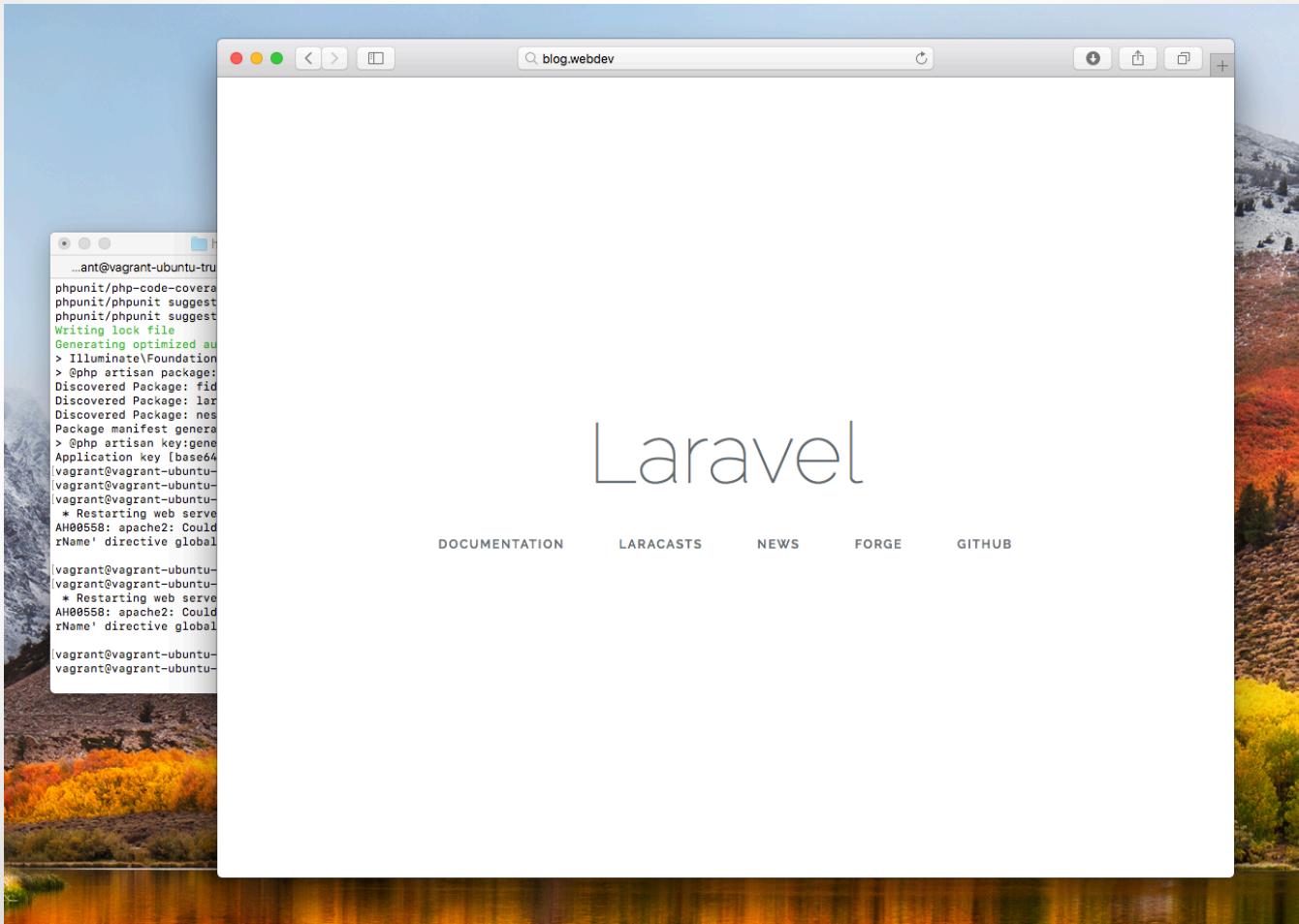
Instalação e Configuração Laravel

GNU nano 2.0.6

File: /etc/hosts

```
##  
# Host Database  
#  
# localhost is used to configure the loopback interface  
# when the system is booting. Do not change this entry.  
##  
127.0.0.1      localhost  
255.255.255.255 broadcasthost  
::1            localhost  
192.168.33.10  blog.webdev
```

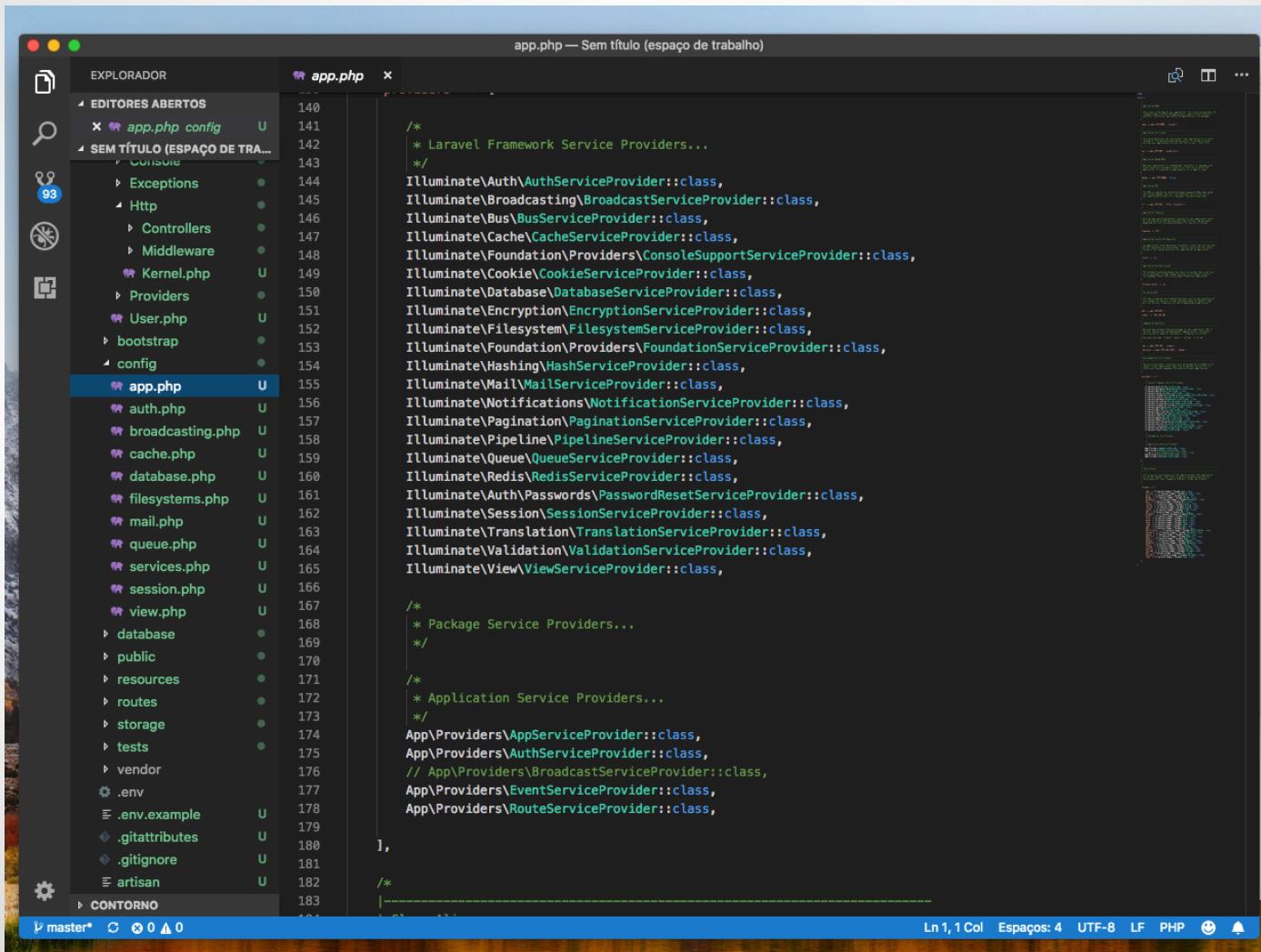
Instalação e Configuração Laravel



Introdução ao Laravel

Estrutura de pastas

Estrutura de pastas



The screenshot shows a code editor window titled "app.php — Sem título (espaço de trabalho)". The left sidebar displays a file tree with the following structure:

- EDITORES ABERTOS
- SEM TÍTULO (ESPAÇO DE TRABALHO)
- CONSOLE
- Exceptions
- Http
- Controllers
- Middleware
- Kernel.php
- Providers
- User.php
- bootstrap
- config
- app.php
- auth.php
- broadcasting.php
- cache.php
- database.php
- filesystems.php
- mail.php
- queue.php
- services.php
- session.php
- view.php
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- CONTORNO

The main editor area contains the following PHP code, which is the bootstrapping code for a Laravel application:

```
/*
 * Laravel Framework Service Providers...
 */

Illuminate\Auth\ServiceProvider::class,
Illuminate\Broadcasting\BroadcastServiceProvider::class,
Illuminate\Bus\BusServiceProvider::class,
Illuminate\Cache\CacheServiceProvider::class,
Illuminate\Foundation\Providers\ConsoleServiceProvider::class,
Illuminate\Cookie\CookieServiceProvider::class,
Illuminate\Database\DatabaseServiceProvider::class,
Illuminate\Encryption\EncryptionServiceProvider::class,
Illuminate\Filesystem\FilesystemServiceProvider::class,
Illuminate\Foundation\Providers\FoundationServiceProvider::class,
Illuminate\Hashing\HashServiceProvider::class,
Illuminate\Mail\MailServiceProvider::class,
Illuminate\Notifications\NotificationServiceProvider::class,
Illuminate\Pagination\PaginationServiceProvider::class,
Illuminate\Pipeline\PipelineServiceProvider::class,
Illuminate\Queue\QueueServiceProvider::class,
Illuminate\Redis\RedisServiceProvider::class,
Illuminate\Auth\Passwords\PasswordResetServiceProvider::class,
Illuminate\Session\SessionServiceProvider::class,
Illuminate\Translation\TranslationServiceProvider::class,
Illuminate\Validation\ValidationServiceProvider::class,
Illuminate\View\ViewServiceProvider::class,

/*
 * Package Service Providers...
 */

/*
 * Application Service Providers...
 */
App\Providers\AppServiceProvider::class,
App\Providers\AuthServiceProvider::class,
// App\Providers\BroadcastServiceProvider::class,
App\Providers\EventServiceProvider::class,
App\Providers\RouteServiceProvider::class,
```

Estrutura de pastas

- No diretório raiz podemos ver que temos vários arquivos e outros diretórios;
- **Diretório “app”** - nele temos todos arquivos de nossa aplicação, podemos ver vários outros diretórios, mas iremos focar no diretório Http, pois nele temos o nosso arquivo routes.php, que é onde definimos todas as rotas de nossa aplicação. Temos também vários diretórios, entre eles, o “Controllers”, que é onde ficam todos os controllers de nossa aplicação.
- **Diretório config** - onde configuramos todo nosso projeto. Nele temos o arquivo app.php, que é onde podemos configurar várias variáveis de nossa aplicação como local, fuso-horário, os providers e definir os aliases de nossa aplicação. Ainda no diretório config temos o arquivo database.php que é onde definimos todas as configurações sobre a conexão com o Banco de Dados. Ainda temos vários outros arquivos como o mail.php para definir configurações de e-mail de nossa aplicação;
- **Diretório database** - onde definimos três tipos de arquivos importantes:
 - **Migrations:** uma das ferramentas mais poderosas do Laravel para definir, através de arquivos PHP, como nosso Banco de Dados deve ser criado. Através do Artisan, que é a interface de linha de comando do Laravel, criamos, alteramos e excluímos tabelas do nosso Banco de Dados de forma fácil, rápida e intuitiva;
 - **Seeds:** com esses arquivos podemos popular as tabelas do Banco de Dados com os dados que queremos para testes de forma fácil e rápida;
 - **Factories:** essa ferramenta foi introduzida na versão 5.1 do Framework, para popular as tabelas do [Banco de Dados](#) com dados criados de forma automática e randômica, permitindo incluir uma grande massa de dados de forma bem rápida para criação de testes;

Estrutura de pastas

- **Diretório “public”** - possui os arquivos .htaccess e index.php, que é o roteador de nossa aplicação. Ele recebe as requisições, as trata através do kernel e retorna para os usuários as respostas.
- **Diretório “resources”** - nele temos três diretórios importantes:
 - **Assets**: usado para armazenarmos todos arquivos de estilo (CSS, LESS, SASS, etc.), scripts (JavaScript, etc.), imagens e outros recursos necessários para nossa aplicação
 - **Lang**: usado para armazenarmos os arquivos de tradução para nossa aplicação;
 - **Views**: usado para armazenar os arquivos de nossa camada de visualização;
- **Diretório “tests”** - nele salvamos todos os testes de nossa aplicação;
- **Diretório “vendor”** - nele temos todos os arquivos de terceiros utilizados em nosso projeto, ou seja, bibliotecas, plugins, etc;
- **Arquivo “.env”** - nele definimos várias configurações de nossa aplicação, como os dados de configuração da conexão do banco de dados e a configuração de e-mails.
- **Arquivo gulpfile.js** - nele temos acesso a API do Elixir, que é usado para definir tarefas do Gulp para nossa aplicação.

Introdução ao Laravel

Sistema de Rotas

Sistema de Rotas

Com o Laravel, podemos trabalhar facilmente com o conceito de rotas. De forma bem simplória, as rotas fazem o mapeamento da URL digitada no navegador para alguma ação dentro da sua aplicação.

```
Route::get('hello', function () {  
    return 'Hello World';  
});
```

Podemos registrar rotas que respondem a qualquer verbo HTTP:

```
Route::get($uri, $callback);  
Route::post($uri, $callback);  
Route::put($uri, $callback);  
Route::patch($uri, $callback);  
Route::delete($uri, $callback);  
Route::options($uri, $callback);
```

Sistema de Rotas

- Além disso, o sistema de roteamento nos permite trabalhar com:
 - **Parâmetros de rotas(Opcionais ou não);**

```
//Exemplo de rota com parâmetro opcional.  
Route::get('welcome/{name?}', function ($name = 'visitante') {  
    return "Seja bem vindo $name!";  
});
```

Restrições de parâmetros com expressões regulares

```
Route::get('welcome/{name}', function ($name) {  
    //  
})->where('name', '[A-Za-z]+');
```

Sistema de Rotas

- Agrupamento de rotas (*Prefixo, Subdomínio, Namespace, Middleware*);

```
//Rotas agrupadas por subdomínio
Route::group(['domain' => '{account}.myapp.com'], function () {
    Route::get('user/{id}', function ($account, $id) {
        //
    });
});
```

Introdução ao Laravel

Controllers

Controllers

- **Artisan Console**

- O Artisan é uma interface de linha de comando que fornece vários comandos para facilitar o desenvolvimento da aplicação. Para visualizar todos os comandos disponíveis basta digitar no terminal:

```
php artisan list
```

Controllers

- **O Fluxo de Nossa Aplicação**

As aplicações criadas com o Framework Laravel seguem um fluxo baseado na arquitetura MVC. Primeiramente devemos definir as rotas para que nossos usuários possam fazer requisições e criar os Controllers necessários para tratar as requisições de nossos usuários.

```
Route::get('/', 'PagesController@index');
```

Assim definimos que quando nosso usuário acessar a rota /, ou seja, a rota inicial de nossa aplicação, o Controller chamado PagesController.php ficará responsável por tratar a requisição de nosso usuário e retornar uma resposta a ele.

```
php artisan make:controller PagesController
```

Controllers

Depois de executar o código, abra o diretório Controllers em app\Http e veja que um novo arquivo foi criado: o PagesController.php. Como padrão do Framework, quando criamos um novo Controller, damos um nome a ele e logo à frente colocamos a palavra “Controller”.

Abra o arquivo que foi criado e verá que o Laravel já cria, por padrão, vários métodos em nosso Controller, que você pode apagar, com exceção do método index().

Até agora já temos configurado a rota, o Controller e o método que irá cuidar da requisição de nosso usuário.

Controllers

```
public function index()
{
    return view('welcome');
}
```

Controllers

- Pronto para aplicações RESTful

O Laravel já foi feito pensado para facilitar a criação de serviços RESTful, portanto, quando devolvemos alguma variável, objeto, array ou qualquer outro dado em nossos Controllers, eles já são convertidos automaticamente para formato JSON. Para exemplificar vamos criar mais uma rota em nosso arquivo routes.php

```
Route::get('/amigos', 'PagesController@amigos');
```

Controllers

- Pronto para aplicações RESTful

Agora vamos criar mais um método em nosso arquivo PagesController.php para tratar a requisição de nossa nova rota.

```
/*
 * Retorna JSON com lista de amigos
 *
 * @return Response
 */
public function amigos()
{
    $amigos = [
        ['nome' => 'José Silva', 'idade' => 22],
        ['nome' => 'Maria José', 'idade' => 20],
        ['nome' => 'João Pinheiro', 'idade' => 35]
    ];
    return $amigos;
}
```

Introdução ao Laravel

Views e Templates

Views e Templates

- Blade — Sistema de templates
 - O Blade é o compilador de templates do Laravel (template engine). A diferença dele para outros templates é a sua flexibilidade, o Blade não restringe o uso de PHP puro misturado a syntaxe do template. Os arquivos blade devem utilizar a extensão .blade.php .
 - O grande objetivo do Blade é reduzir a quantidade de código PHP inserido no meio do HTML e aumentar o reúso, para isso, ele disponibiliza uma serie de diretivas que são inseridas junto ao código HTML de acordo com a necessidade da página.
 - Os dois principais benefícios do uso do Blade são a herança e as seções, permitindo trabalhar facilmente com o conceito de master page. Vejamos o código:

Views e Templates

- <!-- master.blade.php -->

<html>
 <head>
 <title>Exemplo de arquivo Blade</title>
 </head>
 <body>
 <div class="container">
 @yield('conteudo')
 </div>
 </body>
</html>

Esse código representa a estrutura da nossa master page.

Views e Templates

Agora vamos implementar página *dashboard.blade.php* que irá herdar esse layout.

- ```
<!-- dashboard.blade.php -->

@extends('master')

@section('conteudo')
 <p>O conteúdo do nosso dashboard vem aqui!</p>
@endsection
```

Nessa página estamos herdando toda a estrutura da nossa master page utilizando a diretiva `@extends` , além de injetar o conteúdo específico da página através da diretiva `@section` .

Para renderizar esse exemplo fictício, poderíamos criar a seguinte rota:

- ```
Route::get('dashboard', function () {
    return view('dashboard');
});
```

Views e Templates

- **Passando Informações para a Camada de Visualização**
 - Além de retornar os dados como JSON para nossos usuários, podemos também retornar uma página com os dados de nossas variáveis. Vamos criar mais uma rota para nossa aplicação com o seguinte código:

```
• Route::get('/sobre', 'PagesController@sobre');
```

Agora vamos criar um método em nosso arquivo PagesController.php para tratar as requisições de nossa nova rota

```
public function sobre() {  
    $eu = [  
        'nome' => 'Wendell Adriel',  
        'idade' => 23  
    ];  
    return view('sobre', compact('eu'));  
}
```

Views e Templates

Agora vamos criar nosso arquivo sobre.blade.php no diretório resources/views

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Sobre mim</title>
  </head>
  <body>
    <h1>Meu nome é {{ $eu['nome'] }}</h1>
    <h2>Tenho {{ $eu['idade'] }} anos</h2>
  </body>
</html>
```

Conhecendo o Projeto

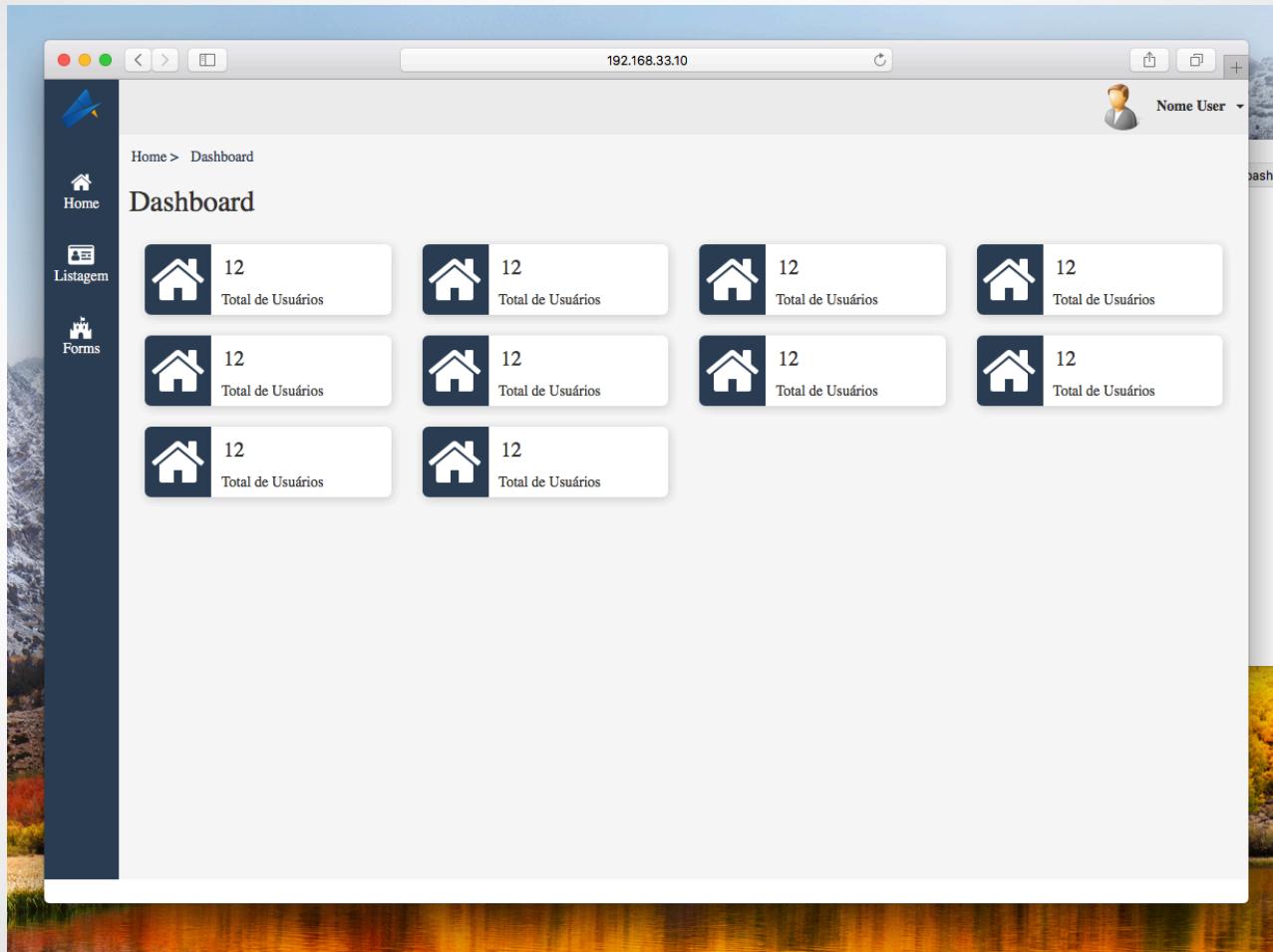
Home

Dashboard

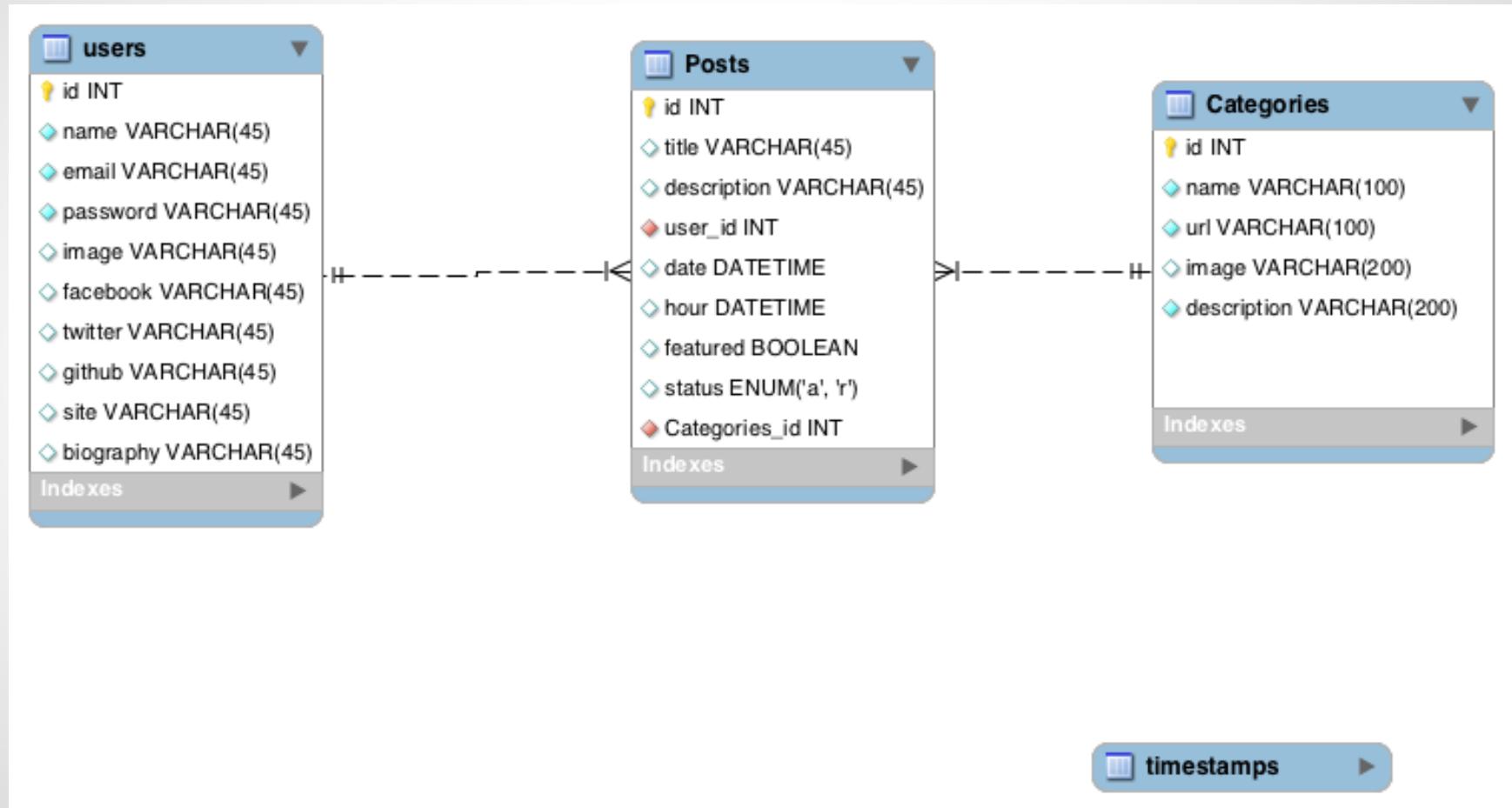
Modelo de dados

Home

Dashboard



Modelo de dados



Integração layout Home & Dashboard

Laravel avançado

Models & Migrations

Models & Migrations

- Os bancos de dados relacionais foram criados para prover um acesso facilitado aos dados, veremos como conseguir criar essa relação e acessar esses dados. Aprenderemos a rodar o comando para criarmos a Model e a Migration.
- Na função up criamos a tabela e na função down dropamos a tabela caso ela exista, também há um método de drop, mas DropIndexIfExists é com certeza a melhor prática possível. Existe uma lista grande de tipos de colunas que podem ser usados além desses a cima, você pode ver os tipos disponíveis na documentação.

Models & Migrations

- *php artisan make:model Models\\Category -m*
- *php artisan make:model Models\\Post –m*
- *Vamos configurar os arquivos de Migrations*

```
<?php

use ...

class CreateCategoriesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name', 100);
            $table->string('url', 100);
            $table->string('image', 200)->nullable();
            $table->text('description' );
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('categories');
    }
}
```

```

<?php
use ...
class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned();
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            $table->integer('category_id')->unsigned();
            $table->foreign('category_id')->references('id')->on('categories')->onDelete('cascade');
            $table->string('title', 250);
            $table->text('description');
            $table->date('date');
            $table->time('hour');
            $table->boolean('featured')->default(false);
            $table->enum('status', ['A', 'R'])->default('A')->comment('A-> Ativo postado, R-> Rascunho, não postado');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('posts');
    }
}

```

```

<?php

use ...
class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->string('image', 200)->nullable();
            $table->string('facebook', 100)->nullable();
            $table->string('twitter', 100)->nullable();
            $table->string('github', 100)->nullable();
            $table->string('site', 100)->nullable();
            $table->text('biography')->nullable();
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}

```

Models & Migrations

- *php artisan migrate*
- Ops! Antes temos que configurar o Banco de Dados.

Models & Migrations

<?php

- Criando usuário através de Seeders
- php artisan make:seed UserTableSeeder
- php artisan db:seed

```
use Illuminate\Database\Seeder;
use App\User;

class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        User::created([
            'name'      => 'Anonymo',
            'email'     =>
            'anonymo@email.com',
            'password'  => bcrypt('secret'),
            'biography' => 'Usuário Anonymous',
        ]);
    }
}
```

To Be
Continued?