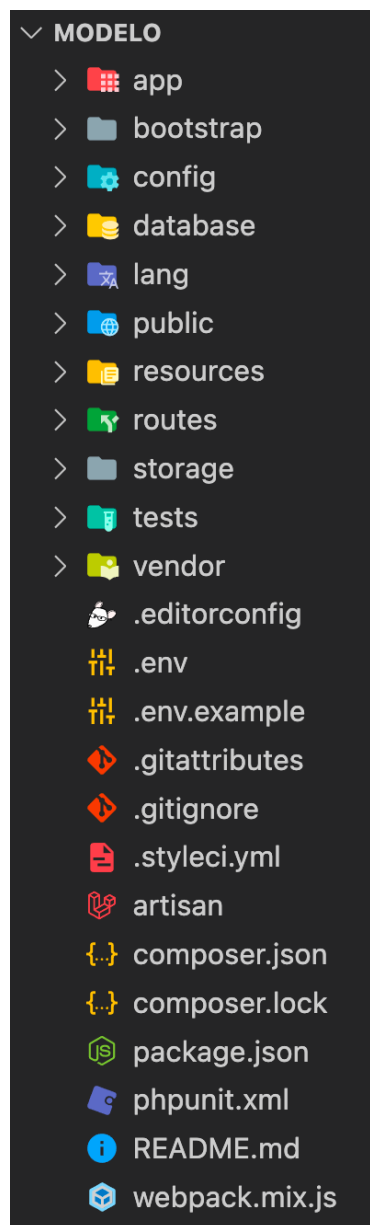
	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Para começar um projeto Laravel é necessário termos alguns softwares instalados no computador. O principal é o PHP versão 8 ou superior.

Existem diversas formas de fazer a instalação do Framework vamos utilizar o composer que é um gerenciador de pacotes PHP. A documentação do laravel está disponível no endereço <https://laravel.com/>. Para criar um projeto Laravel na pasta que você está, digite no terminal:

```
composer create-project laravel/laravel .
```



Este comando irá criar um projeto laravel que contem as pastas ao lado.

Feito isso, já será possível executar este projeto utilizando o terminal e executando o comando abaixo:

```
php artisan serve
```

Este comando irá inicializar o servidor abrindo a porta 8000. Caso apareça o texto abaixo, já será possível acessar a página inicial do seu projeto Laravel:

Starting Laravel development server: <http://127.0.0.1:8000>

Acesse o endereço indicado para verificar se seu projeto está rodando.

Feito isso, vamos configurar o acesso ao banco de dados. As configurações estão localizadas no arquivo .env. As configurações do Banco de dados são:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=
DB_USERNAME=
DB_PASSWORD=
```

Outra configuração importante que deve ser realizada é no arquivo config/app.php. Onde está escrito

```
'timezone' => 'UTC',
```

Alterar para:


```
'timezone' => 'America/Sao_Paulo',
```

Feito isso, vamos instalar alguns pacotes para realizarmos a criação e a autenticação de usuários. Para isso, vamos instalar os pacotes Laravel Jetstream com Livewire + Blade. O Blade é o recurso de templates do Laravel e vamos utilizar ele com o Livewire para rodar o Jetstream. A documentação destes pacotes está disponível em <https://jetstream.laravel.com/>. Para fazer a instalação deve-se executar o comando abaixo no terminal, dentro da pasta do projeto:

```
composer require laravel/jetstream
```

Para instalar a o jetstream com o livewire, deve-se executar o comando abaixo:

```
php artisan jetstream:install livewire
```

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Caso queira dividir seus usuários em times, altere o comando acima para:

```
php artisan jetstream:install livewire --teams
```

Esta instalação utiliza pacotes nodejs que precisam ser instalados. Para isso, execute os comandos abaixo:

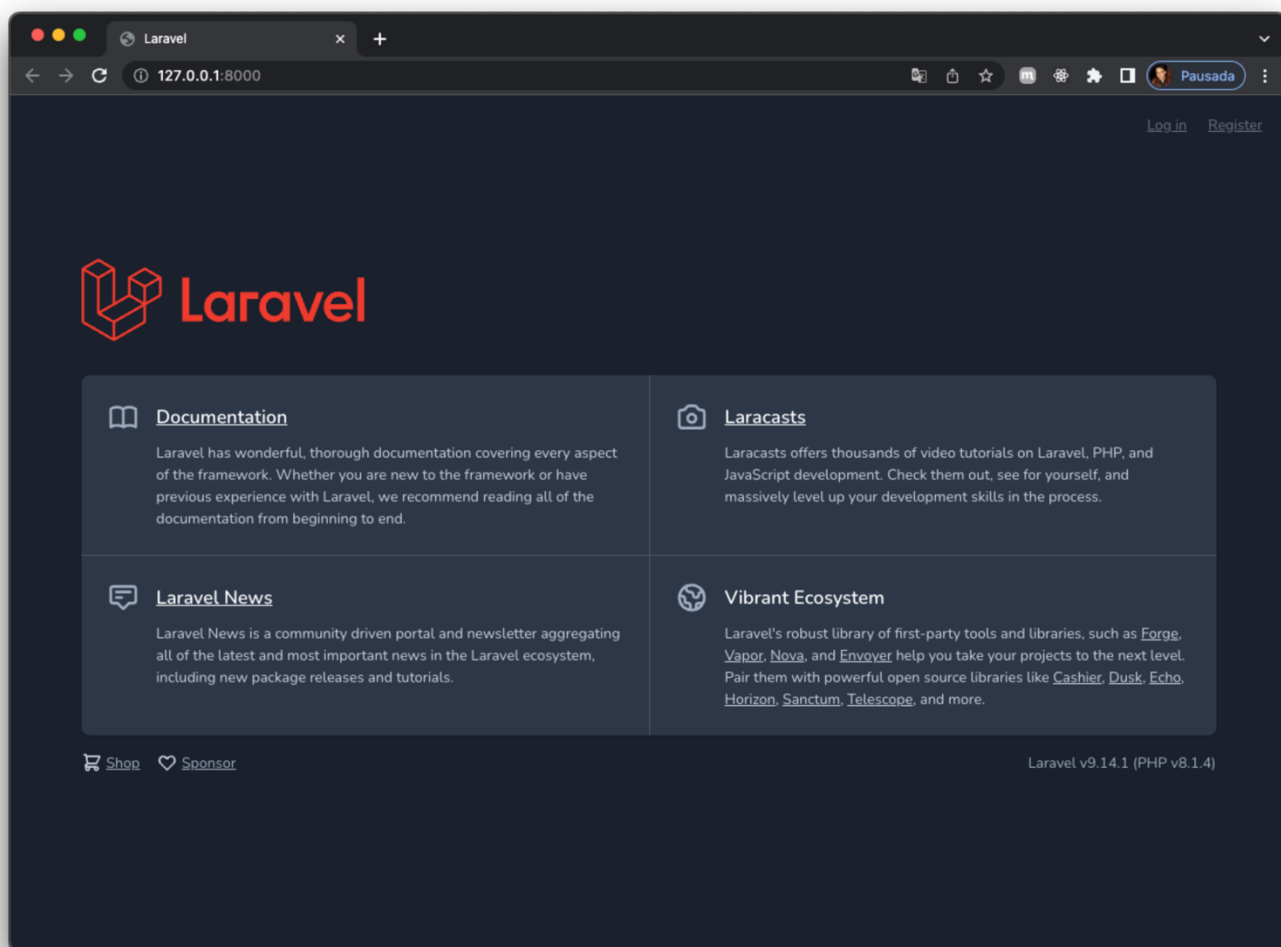
```
yarn
yarn dev
```

Depois de instalar os pacotes é necessário rodar as migrations para que as tabelas do banco de dados utilizado para registro dos usuários sejam criadas.

```
php artisan migrate
```


Confira no seu banco de dados se as tabelas foram criadas.

Coloque seu projeto para rodar novamente. Você perceberá que os links Login e Register foram acrescentados comparando a execução anterior.



Clique em Register e crie um usuário pra você.

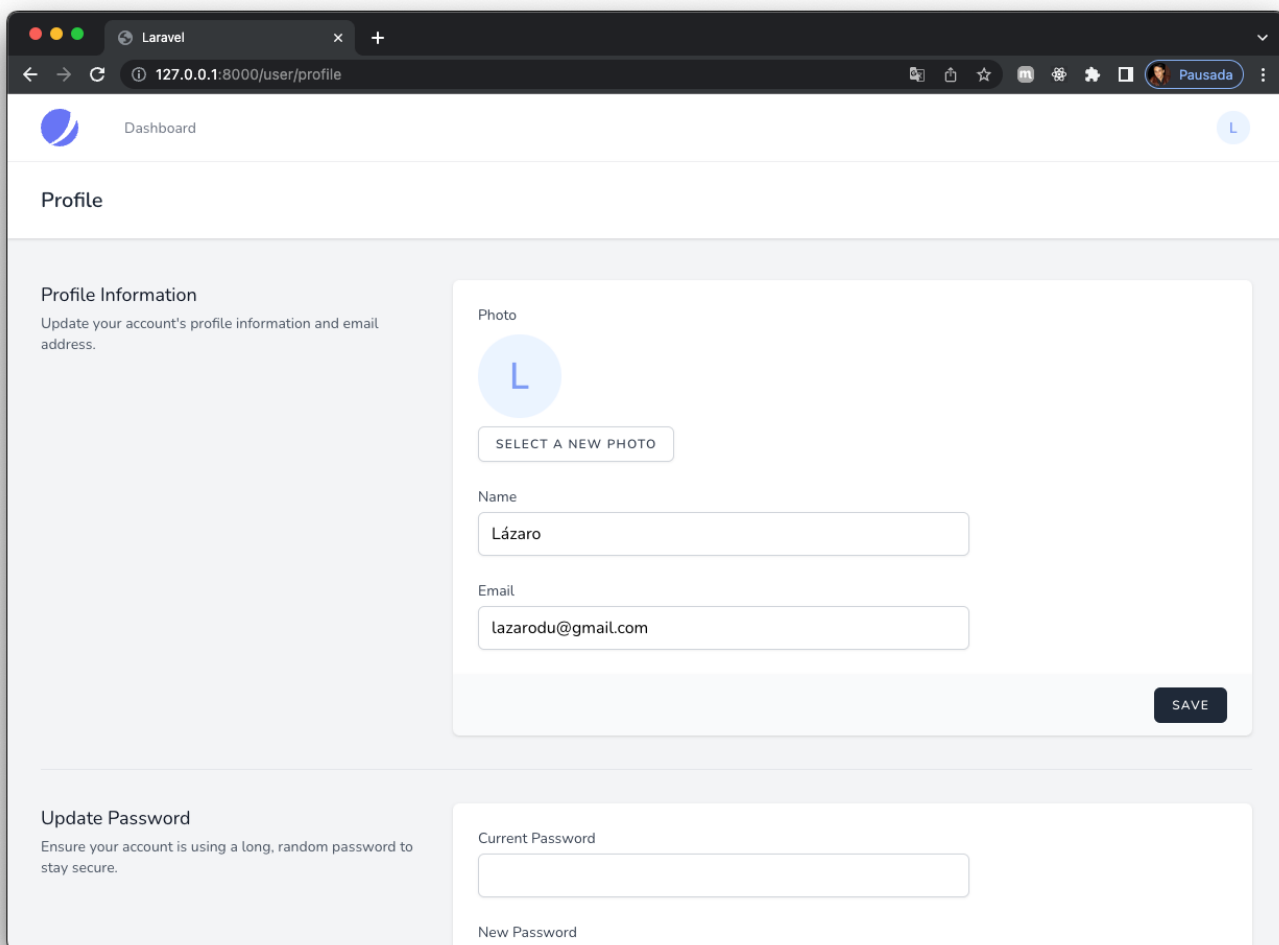
Após criar, faça o login, você verá que um dashboard e uma página de Profile do seu usuário já foi criada. Para habilitarmos alguns recursos que utilizaremos para autenticar o usuário via API, vamos editar o arquivo config/jetstream.php.

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Você deve encontrar as configurações de features da ferramenta, ela deve estar como a abaixo:


```
'features' => [
    // Features::termsAndPrivacyPolicy(),
    // Features::profilePhotos(),
    // Features::api(),
    // Features::teams(['invitations' => true]),
    Features::accountDeletion(),
],
```

Vamos habilitar a profilePhotos() e a api(). A interface do profile deve ficar como a abaixo:



Para utilizarmos esta API, precisamos fazer algumas configurações. A primeira delas é retirar o comentário da linha no arquivo que está na pasta app/Http/Kernel.php

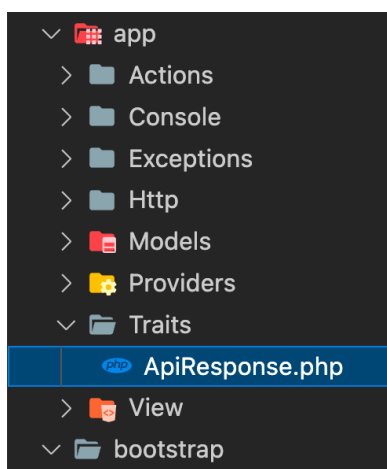
```
'api' => [
    // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
    'throttle:api',
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
],
```

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Ficando assim

```
'api' => [
    \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
    'throttle:api',
    \Illuminate\Routing\Middleware\SubstituteBindings::class,
],
```

Vamos criar uma Trait para formatar a saída da nossa API. Crie uma pasta Traits na pasta app e crie um arquivo chamado ApiResponse.php como na imagem abaixo:



Este arquivo deve ter o conteúdo abaixo:


```
<?php
namespace App\Traits;
```

```
trait ApiResponse
{
    protected function success($data, string $message = null, int $code = 200)
    {
        return response()->json([
            'status' => "Sucesso",
            'message' => $message,
            'data' => $data
        ], $code);
    }

    protected function error(string $message = null, int $code, $data = null)
    {
        return response()->json([
            'status' => "Erro",
            'message' => $message,
            'data' => $data
        ], $code);
    }
}
```

Vamos criar o arquivo do controller que terá o código para registrar, fazer login e logout do usuário executando o comando abaixo no terminal:

```
php artisan make:controller API/AuthController
```

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

O conteúdo deste arquivo deve ser:

```
<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;
use App\Models\User;
use App\Traits\ApiResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;

class AuthController extends Controller
{
    use ApiResponse;

    public function register(Request $request)
    {
        try {
            $validatedData = Validator::make($request->all(), [
                'name' => 'required|string|max:255|unique:users',
                'email' => 'required|string|max:255|unique:users',
                'password' => 'required|string|min:8'
            ]);


            if ($validatedData->fails()) {
                return $this->error("Falha ao registrar!!!", 403, $validatedData->errors());
            }

            $user = User::create([
                'name' => $request->get('name'),
                'email' => $request->get('email'),
                'password' => Hash::make($request->get('password')),
            ]);

            $token = $user->createToken('auth_token')->plainTextToken;

            return $this->success([
                'access_token' => $token,
                'user' => $user,
            ], "Cadastro realizado com sucesso!!!");
        } catch (\Throwable $th) {
            return $this->error("Falha ao registrar!!!", 401, $th->getMessage());
        }
    }

    public function login(Request $request)
    {
```

	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

```

try {
    if (!Auth::attempt($request->only('email', 'password'))) {
        return $this->error('Dados de autenticação inválidos!!!', 401);
    }

    $user = User::where('email', $request['email'])->firstOrFail();

    $token = $user->createToken('auth_token')->plainTextToken;

    return $this->success([
        'access_token' => $token,
        'user' => $user
    ], "Login realizado!!!");
} catch (\Throwable $th) {
    return $this->error("Falha ao realizar o login!!!", 401, $th->getMessage());
}

}

public function logout(Request $request)
{
    try {
        $request->user()->currentAccessToken()->delete();

        return $this->success(null, "Até a próxima!!!");
    } catch (\Throwable $th) {
        return $this->error("Falha ao realizar o logout!!!", 401, $th->getMessage());
    }
}

}

```

Precisamos acrescentar ainda as rotas da API. Elas ficam na pasta routes/api.php. Precisamos fazer algumas alterações neste arquivo. Vou deixar o código alterado comentado.

```


<?php

use App\Http\Controllers\API\AuthController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

/*
|-----
| API Routes
|-----
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

Route::post('/register', [AuthController::class, 'register']);
Route::post('/login', [AuthController::class, 'login']);

```

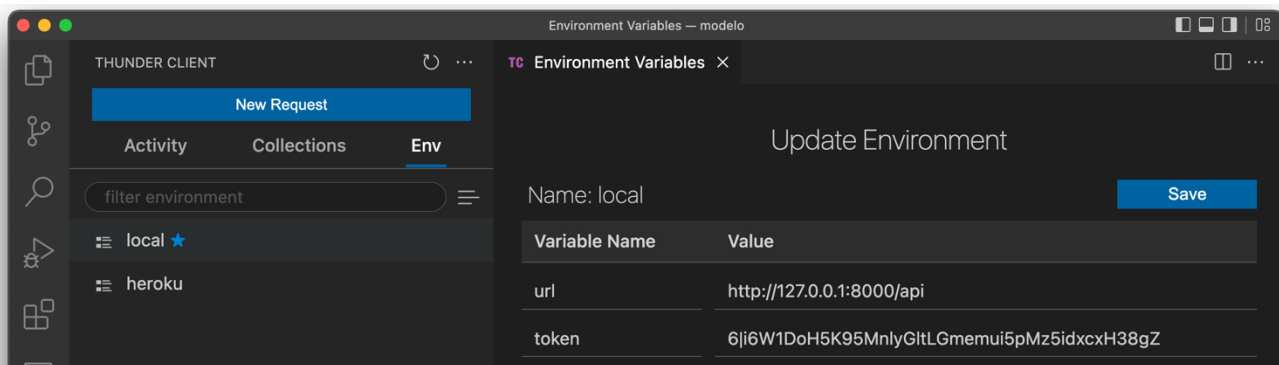
	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

```
// Route::middleware('auth:sanctum')->get('/user', function (Request $request) {
//     return $request->user();
// });
Route::middleware('auth:sanctum')->group(function () {
    Route::get('/logout', [AuthController::class, 'logout']);
});
```

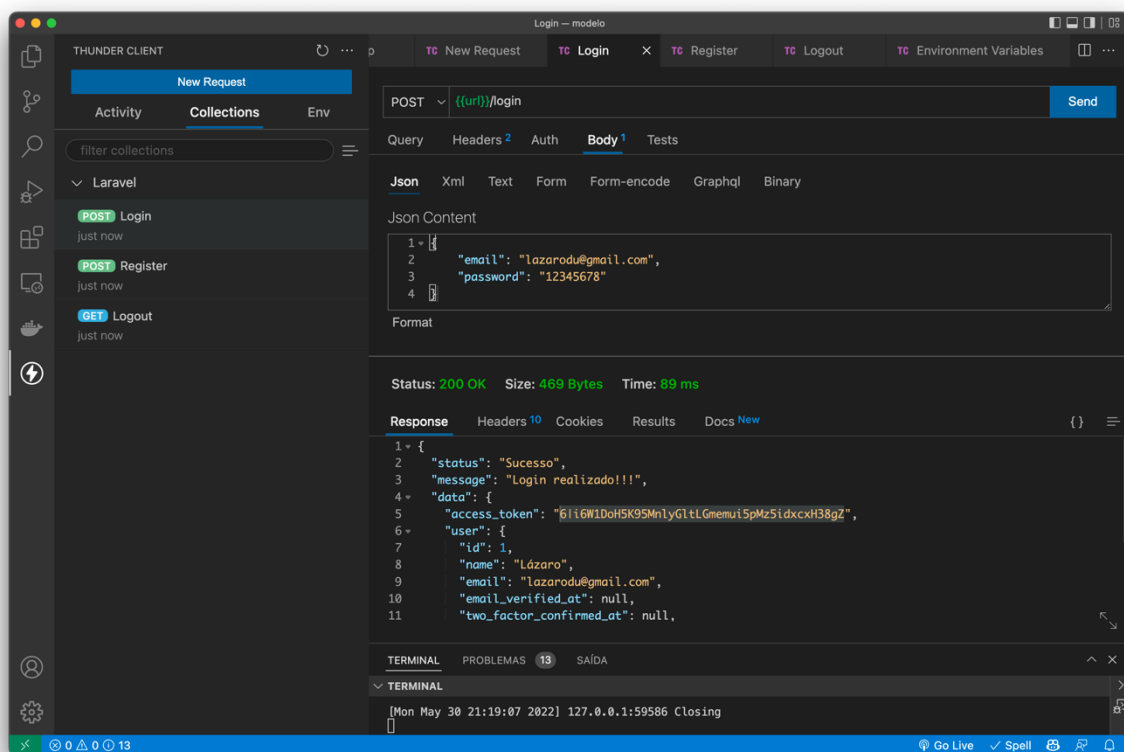
Feito isso, execute o seu projeto novamente com o comando abaixo:

php artisan serve

Vamos testar a API. Instale a extensão Thunder Client no Visual Studio Code e configure inicialmente o Environment.



Posteriormente a Collectionm Laravel e as Requests Login, Register e Logout conforme imagens abaixo:





Centro Federal de Educação Tecnológica de Minas Gerais
Campus VIII – Varginha
Curso Técnico em Informática

Disciplina
Aplicações para Web II

Instalação e Configuração do
Laravel

Professor
Lázaro Eduardo da Silva

Thunder Client interface showing a POST request to `register` with a JSON body:

```
{  "name": "Eduardo",  "email": "lazarodu@hotmail.com",  "password": "12345678"}
```

The response status is **200 OK** (377 Bytes, 108 ms). The response body is:


```
{  "status": "Sucesso",  "message": "Cadastro realizado com sucesso!!!",  "data": {    "access_token": "711CNfaZn8zDEKzd29QHwMUsb48YypPa5LeLEK0Cscp",    "user": {      "name": "Eduardo",      "email": "lazarodu@hotmail.com",      "updated_at": "2022-05-31T00:18:10.000000Z",      "created_at": "2022-05-31T00:18:10.000000Z",      "id": 2,    }  }}
```

Thunder Client interface showing a GET request to `logout` with a Bearer token:

```
Bearer Token: {{token}}
```

The response status is **200 OK** (71 Bytes, 32 ms). The response body is:

```
{  "status": "Sucesso",  "message": "Até a próxima!!!",  "data": null}
```


	<p align="center">Centro Federal de Educação Tecnológica de Minas Gerais Campus VIII – Varginha Curso Técnico em Informática</p>	
<p><i>Disciplina</i> Aplicações para Web II</p>	<p align="center">Instalação e Configuração do Laravel</p>	<p><i>Professor</i> Lázaro Eduardo da Silva</p>

Por fim, vamos subir esta aplicação no heroku.

Para subir no heroku, é necessário criar um arquivo chamado Procfile na raiz do seu projeto.

O conteúdo deste arquivo deve ser:

```
release: php artisan migrate --force
web: vendor/bin/heroku-php-apache2 public/
```

Outra configuração muito importante deve ser realizada no arquivo app/Providers/AppServiceProvider.php deve ser acrescentado no método boot o seguinte código:

```
public function boot()
{
    if ($this->app->environment('production')) {
        URL::forceScheme('https');
    }
}
```

A URL utilizada neste método é uma Facade que deve ser importada no começo do arquivo

```
use Illuminate\Support\Facades\URL;
```

Feitas estas configurações, antes de qualquer configuração no heroku, suba seu código no github.

Observe que o arquivo .env e o diretório vendor não vão para o github. O arquivo .env são as configurações do Banco de Dados, que depende de cada local que a aplicação estiver sendo executada. A pasta vendor são os pacotes PHP que o Laravel utiliza, portanto, devem estar adequados ao local de execução também.

No heroku, crie uma nova aplicação.

Dê um nome a ela e clique em create app.

Após criar, acesse a aba Resource e digite na barra de busca postgres. Selecione Heroku Postgres.

Deixe marcado Hobby Dev – Free no plano e clique em submit order form.

Clique na aba Settings e clique no botão Reveal Config Vars.

Vamos acrescentar algumas variáveis no formato chave valor. Estas configurações correspondem às configurações do arquivo .env. Vamos acrescentar as configurações abaixo presentes no seu arquivo .env:

APP_KEY (valor que está no .env)

DB_CONNECTION (pgsql)

Após acrescentar estas variáveis, acesse a aba Deploy.

Logo abaixo clique na opção GitHub.

Clique em Connect, conceda a devida autorização.

Na barra de busca, digite o nome do repositório github que está com o seu projeto Laravel e clique em Search.

Ao localizar clique em connect.

Após conectar clique em Deploy Branch.

Clique em Open App para ver seu site funcionando.

Registre um usuário para posteriormente testá-lo na API.

Crie um novo environment com a URL do Heroku.

Clique nos ... ao lado do environment e vá em Set Active para ativá-lo.

Teste os endpoints utilizando a API.