BY OLUYEMI OLUSUSI • 2020-04-20

Build a Secure API in PHP Using Laravel Passport

There is no way to avoid the topic of RESTful APIs when building backend resources for a mobile application or using any of the modern JavaScript frameworks. If by chance you are unaware, an API is an interface or code in this case, that allows two software programs to communicate with each other. Notably, it does not maintain session state between requests, hence, you will need to use tokens to authenticate and authorize users of your application.

Laravel makes building such a resource easy with a predefined provision for you to secure it appropriately. This tutorial will teach you how to build and secure your Laravel back-end API using Laravel passport. When we are finished, you will have learned how to secure your new Laravel API or provide an extra layer of security to existing ones.

Prerequisites

Basic knowledge of building applications with Laravel will be of help in this tutorial. Also, you need to ensure that you have installed Composer globally to manage dependencies. Lastly, Postman will be needed to test our endpoints.

What We'll Build

To demonstrate how to build a secure Laravel API, we'll build an API that will be used to create a list of top tech CEOs. This application will list the following about each CEO:

- Name
- The year they became CEO
- Headquarters of their company and
- What their company does

To secure this application, we will install Laravel Passport and generate an access token for each user after authentication. This will allow such users to have access to some of the secured endpoints.

Getting Started

To begin, you can either use Composer or Laravel installer to quickly scaffold a new Laravel application on your computer. Follow the instructions here on Laravel's official website to set up the Laravel installer. Once you are done, run the following command:

```
$ laravel new laravel-backend-api
```

To install the same application using Composer run the following command:

```
$ composer create-project --prefer-dist laravel/laravel laravel-backend
```

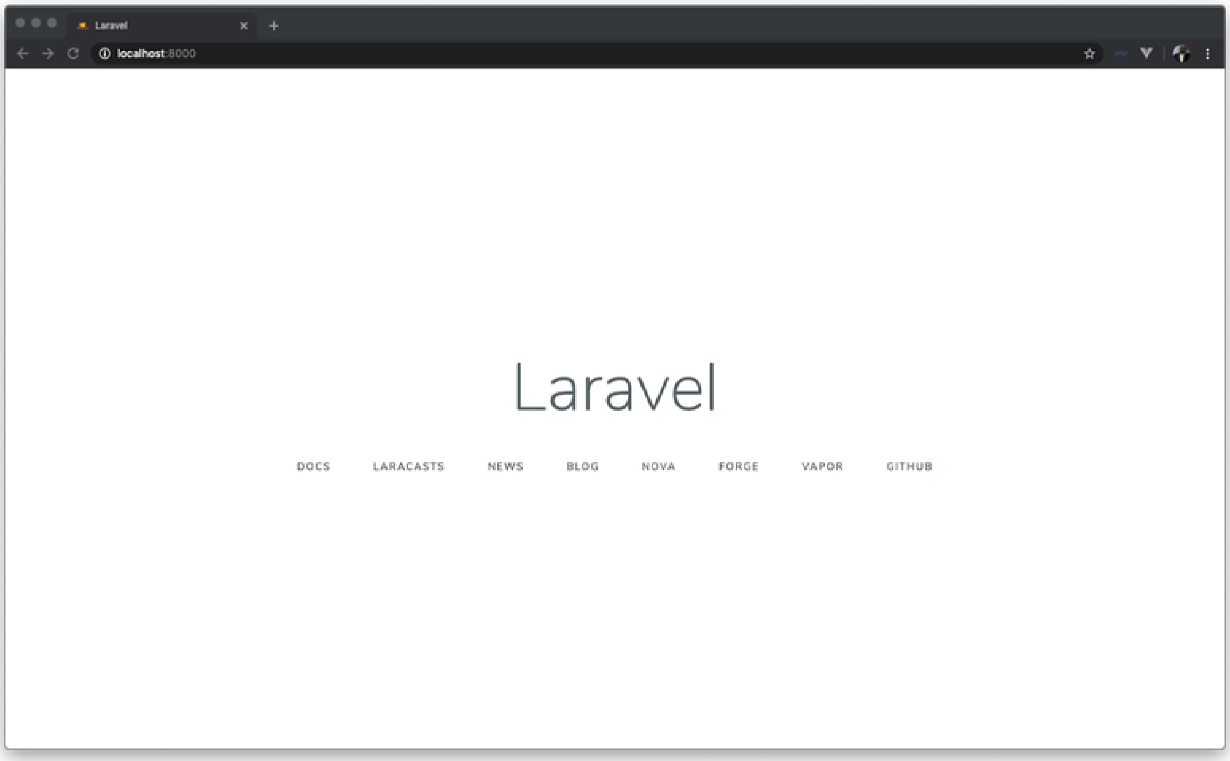
Depending on your preferred choice, the preceding commands will create a new folder named `laravel-backend-api` within the development folder, you installed Laravel and its dependencies in.

You can move into the newly created folder and run the application using the in built-in Laravel Artisan command as shown here:

```
// move into the project
$ cd laravel-backend-api

// run the application
$ php artisan serve
```

Navigate to http://localhost:8000 from your browser to view the welcome page:



Create a Database and Connect to It

Now that Laravel is installed and running, the next step is to create a connection to your database. First, ensure that you have created a database and then update the values of the following variables within the `.env` file:

- DB_DATABASE
- DB_USERNAME
- DB_PASSWORD

The database is all set, but before we start building our API, we need to install and configure Laravel Passport.

Install And Configure Laravel Passport

Laravel Passport provides a full 0Auth2 server implementation for Laravel applications. With it, you can easily generate a personal access token to uniquely identify a currently authenticated user. This token will then be attached to every request allowing each user access protected routes. To begin, stop the application from running by hitting `CTRL + C` on your computer's keyboard and install Laravel Passport using Composer as shown here:

```
$ composer require laravel/passport
```

Once the installation is complete, a new migration file containing the tables needed to store clients and access tokens will have been generated for your application. Run the following command to migrate your database:

```
$ php artisan migrate
```

Next, to create the encryption keys needed to generate secured access tokens, run the command below:

```
$ php artisan passport:install
```

Immediately after the installation process from the preceding command is finished, add the `Laravel\Passport\HasApiTokens` trait to your `App\User` model as shown here:

```php
// app/User.php

<?php

namespace App;

...
use Laravel\Passport\HasApiTokens; // include this

class User extends Authenticatable
{
    use Notifiable, HasApiTokens; // update this line

    ...
}
```

One of the benefits of this trait is the access to a few helper methods that your model can use to inspect the authenticated user's token and scopes.

Now, to register the routes necessary to issue and revoke access tokens (personal and client), you will call the `Passport::routes` method within the `boot` method of your `AuthServiceProvider`. To do this, open the `app/Providers/AuthServiceProvider` file and update its content as shown below:

```
// app/Providers/AuthServiceProvider.php

<?php

namespace App\Providers;

use Illuminate\Foundation\Support\Providers\AuthServiceProvider as Serv
use Illuminate\Support\Facades\Gate;
use Laravel\Passport\Passport; // add this

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
         'App\Model' => 'App\Policies\ModelPolicy', // uncomment this
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();

        Passport::routes(); // Add this
    }
}
```

After registering `Passport::routes()` , Laravel Passport is almost ready to handle all authentication and authorization processes within your application.

Finally, for your application to be ready to use Passport’s `TokenGuard`

to authenticate any incoming API requests, open the `config/auth` configuration file and set the `driver` option of the `api` authentication guard to `passport` :

```php
// config/auth

<?php

return [
    ...

    'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],

        'api' => [
            'driver' => 'passport', // set this to passport
            'provider' => 'users',
            'hash' => false,
        ],
    ],

    ...
];
```

Create a Migration File for the Company

Every new installation of Laravel comes with a pre-generated *User* model and migration file. This is useful for maintaining a standard database structure for your database. Open the `app/User.php` file and ensure that it is similar to this:

```php
// app/User.php

<?php

namespace App;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Passport\HasApiTokens;

class User extends Authenticatable
{
    use Notifiable, HasApiTokens;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];
}
```

Also for the user migration file in `database/migrations/***_create_users_table.php` :

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

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

The fields specified in the file above will suffice for the credentials required from the users of our application, hence there will be no need to modify it.

Next, we will use the `artisan` command to create a model instance and generate a database migration file for the *CEO* table:

```
$ php artisan make:model CEO -m
```

The preceding command will create a model within the `app` directory and a new migration file in `database/migrations` folder. The `-m` option is short for `--migration` and it tells the artisan command to create a migration file for our model. Next, open the newly created migration file and update its content as shown here:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateCEOSTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('c_e_o_s', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('company_name');
            $table->year('year');
            $table->string('company_headquarters');
            $table->string('what_company_does');
            $table->timestamps();
        });
    }

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

Here, we included `name`, `company_name`, `year`, `company_headquarters` and `what_company_does` fields.

Now open the `app/CEO.php` file and use the following content for it:

```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class CEO extends Model
{
    protected $fillable = [
        'name', 'company_name', 'year', 'company_headquarters', 'what_
    ];
}

```

Here, we specified the attributes that should be mass assignable, as all Eloquent models protect against mass-assignment by default.

Run the migration command again to update the database with the newly created table and fields using the following command:

```
$ php artisan migrate
```

Now that the database is updated, we will proceed to create controllers for the application. We will also create a couple of endpoints that will handle registration, login, and creating the details of a CEO as explained earlier.

Create controllers

Controllers accept incoming HTTP requests and redirect them to the appropriate action or methods to process such requests and return the appropriate response. Since we are building an API, most of the responses will be in JSON format. This is mostly considered the standard format for RESTful APIs.

Authentication controller

We will start by using the `artisan` command to generate an *Authentication Controller* for our application. This controller will process and handle requests for registration and login for a user into the application.

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

This will create a new `API` folder within `app/Http/Controllers` and then creates a new file named `AuthController.php` within it. Open the newly created controller file and use the following content for it:

```php
<?php

namespace App\Http\Controllers\API;

use App\Http\Controllers\Controller;
use App\User;
use Illuminate\Http\Request;

class AuthController extends Controller
{
    public function register(Request $request)
    {
        $validatedData = $request->validate([
            'name' => 'required|max:55',
            'email' => 'email|required|unique:users',
            'password' => 'required|confirmed'
        ]);

        $validatedData['password'] = bcrypt($request->password);

        $user = User::create($validatedData);

        $accessToken = $user->createToken('authToken')->accessToken;

        return response([ 'user' => $user, 'access_token' => $accessTo
    }

    public function login(Request $request)
    {
        $loginData = $request->validate([
            'email' => 'email|required',
            'password' => 'required'
        ]);

        if (!auth()->attempt($loginData)) {
            return response(['message' => 'Invalid Credentials']);
        }

        $accessToken = auth()->user()->createToken('authToken')->acces

        return response(['user' => auth()->user(), 'access_token' => $a

    }
}
```

The `register` method above handled the registration process for users of our application. To handle validation and ensure that all the required fields for registration are filled, we used Laravel's validation method. This validator will ensure that the `name`, `email`, `password` and `password_confirmation` fields are required and return the appropriate feedback.

Lastly, the `login` method ensures that the appropriate credentials are inputted before authenticating a user. If authenticated successfully, an `accessToken` is generated to uniquely identify the logged in user and send a JSON response. Any subsequent HTTP requests sent to a secured or protected route will require that the generated `accessToken` be passed as an Authorization header for the process to be successful. Otherwise, the user will receive an unauthenticated response.

Creating the CEO Controller

Here, you will use the same `artisan` command to automatically create a new controller. This time around we will create an API resource controller. Laravel resource controllers are controllers that handle all HTTP requests for a particular Model. In this case, we want to create a controller that will handle all requests for the *CEO* model, which include creating, reading, updating, and deleting. To achieve this, run the following command:

```
$ php artisan make:controller API/CEOController --api --model=CEO
```

The command above will generate an API resource controller that does not include the `create` and `edit` view since we are only building APIs. Navigate to `app/Http/Controllers/API/CEOController.php` and update its contents as shown below:

```
<?php

namespace App\Http\Controllers\API;

use App\CEO;
use App\Http\Controllers\Controller;
use App\Http\Resources\CEOResource;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;

```

```
class CEOController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $ceos = CEO::all();
        return response([ 'ceos' => CEOResource::collection($ceos), 'me
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $data = $request->all();

        $validator = Validator::make($data, [
            'name' => 'required|max:255',
            'year' => 'required|max:255',
            'company_headquarters' => 'required|max:255',
            'what_company_does' => 'required'
        ]);

        if($validator->fails()){
            return response(['error' => $validator->errors(), 'Validat
        }

        $ceo = CEO::create($data);

        return response([ 'ceo' => new CEOResource($ceo), 'message' =>
    }

    /**
     * Display the specified resource.
     *
     * @param  \App\CEO  $ceo
     * @return \Illuminate\Http\Response
     */
    public function show(CEO $ceo)
    {
            return response([ 'ceo' => new CEOResource($ceo), 'message' =>

        }

        /**
         * Update the specified resource in storage.
         *
         * @param  \Illuminate\Http\Request  $request
         * @param  \App\CEO  $ceo
         * @return \Illuminate\Http\Response
         */
        public function update(Request $request, CEO $ceo)
        {

            $ceo->update($request->all());

            return response([ 'ceo' => new CEOResource($ceo), 'message' =>
        }

        /**
         * Remove the specified resource from storage.
         *
         * @param \App\CEO $ceo
         * @return \Illuminate\Http\Response
         * @throws \Exception
         */
        public function destroy(CEO $ceo)
        {
            $ceo->delete();

            return response(['message' => 'Deleted']);
        }
    }
```

A quick view at the code snippet above shows that we created five different methods; each housing logic to carry out a particular function. Here is an overview of what each method does:

- `index` : This method retrieves the entire list of CEOs from the database and returns it as a resource collection in a JSON structure. More details about the Laravel resource collection will be shared in the next section.

  - `store` : This method receives the instance of the HTTP request to create new CEO details via dependency injection using `$request` to retrieve all the values posted. It also ensures that all the required fields are not submitted as an empty request. Lastly, it returns the information of the newly created CEO details as a JSON response.
  - `show` : This method uniquely identifies a particular CEO and returns the details as a JSON response.
  - `update` : This receives the HTTP request and the particular item that needs to be edited as a parameter. It runs an update on the instance of the model passed into it and returns the appropriate response.
  - `` `destroy `` : This method receives the instance of a particular item that needs to be deleted and removes it from the database.

Some methods require a specific model `ID` to uniquely verify an item such as `show()` , `update()` , and `destroy()` . One thing to note here is that we were able to inject the model instances directly. This is due to using implicit route model binding in Laravel.

Once in place, Laravel will help to inject the instance `CEO` into our methods and return a `404` status code if not found. This makes it very easy to use the instance of the model directly, without necessarily running a query to retrieve the model that corresponds to that `ID` .

You can read more about implicit route model binding here on the official documentation of Laravel.

Create a Resource

Laravel Eloquent resources allow you to convert your models and collections into JSON format. It works as a data transformation layer between the database and the controllers. This helps provide a uniform interface that can be used wherever you need it within your application. Let’s create one for our *CEO* model by using the following command:

```
$ php artisan make:resource CEOResource
```

This will create a new resource file named `CEOResource.php` within the `app/Http/Resources` directory. Go ahead and open the newly created file. The content will look like this:

```php
<?php

namespace App\Http\Resources;

use Illuminate\Http\Resources\Json\JsonResource;

class CEOResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function toArray($request)
    {
        return parent::toArray($request);
    }
}
```

The `parent::toArray($request)` inside the `toArray()` method will automatically convert all visible model attributes as part of the JSON response.

Gain a deeper understanding of the benefits offered by Eloquent resources here.

Update Routes File

To complete the set up of the endpoints for the methods created within our controllers, update the `routes.api.php` file with the following contents:

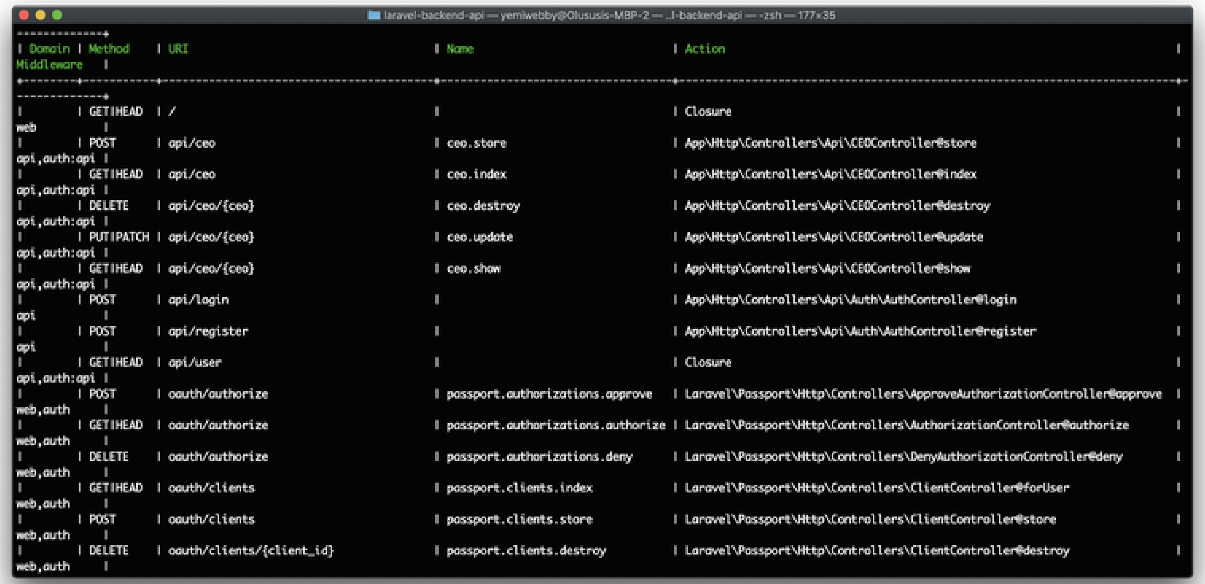
```php
Route::post('/register', 'Api\AuthController@register');
Route::post('/login', 'Api\AuthController@login');

Route::apiResource('/ceo', 'Api\CEOController')->middleware('auth:api'
```

To view the entire list of the routes created for this application, run the following command from the terminal:

```
$ php artisan route:list
```

You will see similar contents as shown below:


laravel-backend-api — yemiwebby@Olususis-MBP-2 — ..l-backend-api — -zsh — 177×35

```
-------------+
| Domain | Method    | URI                                     | Name                               | Action                                                                 |
Middleware   |
+--------+-----------+-----------------------------------------+------------------------------------+------------------------------------------------------------------------+
-------------+
|        | GET|HEAD  | /                                       |                                    | Closure                                                                |
web          |
|        | POST      | api/ceo                                 | ceo.store                          | App\Http\Controllers\Api\CEOController@store                           |
api,auth:api |
|        | GET|HEAD  | api/ceo                                 | ceo.index                          | App\Http\Controllers\Api\CEOController@index                           |
api,auth:api |
|        | DELETE    | api/ceo/{ceo}                           | ceo.destroy                        | App\Http\Controllers\Api\CEOController@destroy                         |
api,auth:api |
|        | PUT|PATCH | api/ceo/{ceo}                           | ceo.update                         | App\Http\Controllers\Api\CEOController@update                          |
api,auth:api |
|        | GET|HEAD  | api/ceo/{ceo}                           | ceo.show                           | App\Http\Controllers\Api\CEOController@show                            |
api,auth:api |
|        | POST      | api/login                               |                                    | App\Http\Controllers\Api\Auth\AuthController@login                     |
api          |
|        | POST      | api/register                            |                                    | App\Http\Controllers\Api\Auth\AuthController@register                  |
api          |
|        | GET|HEAD  | api/user                                |                                    | Closure                                                                |
api,auth:api |
|        | POST      | oauth/authorize                         | passport.authorizations.approve    | Laravel\Passport\Http\Controllers\ApproveAuthorizationController@approve |
web,auth     |
|        | GET|HEAD  | oauth/authorize                         | passport.authorizations.authorize  | Laravel\Passport\Http\Controllers\AuthorizationController@authorize    |
web,auth     |
|        | DELETE    | oauth/authorize                         | passport.authorizations.deny       | Laravel\Passport\Http\Controllers\DenyAuthorizationController@deny     |
web,auth     |
|        | GET|HEAD  | oauth/clients                           | passport.clients.index             | Laravel\Passport\Http\Controllers\ClientController@forUser             |
web,auth     |
|        | POST      | oauth/clients                           | passport.clients.store             | Laravel\Passport\Http\Controllers\ClientController@store               |
web,auth     |
|        | DELETE    | oauth/clients/{client_id}               | passport.clients.destroy           | Laravel\Passport\Http\Controllers\ClientController@destroy             |
web,auth     |
```

This will give you an idea of all the routes for our application.

NOTE: *Please bear in mind that all the routes inside the* `routes/api.php` *file will be prefixed with* `/api/` *, hence you will need to include that when sending HTTP requests to the endpoints created earlier.*

Run the Application

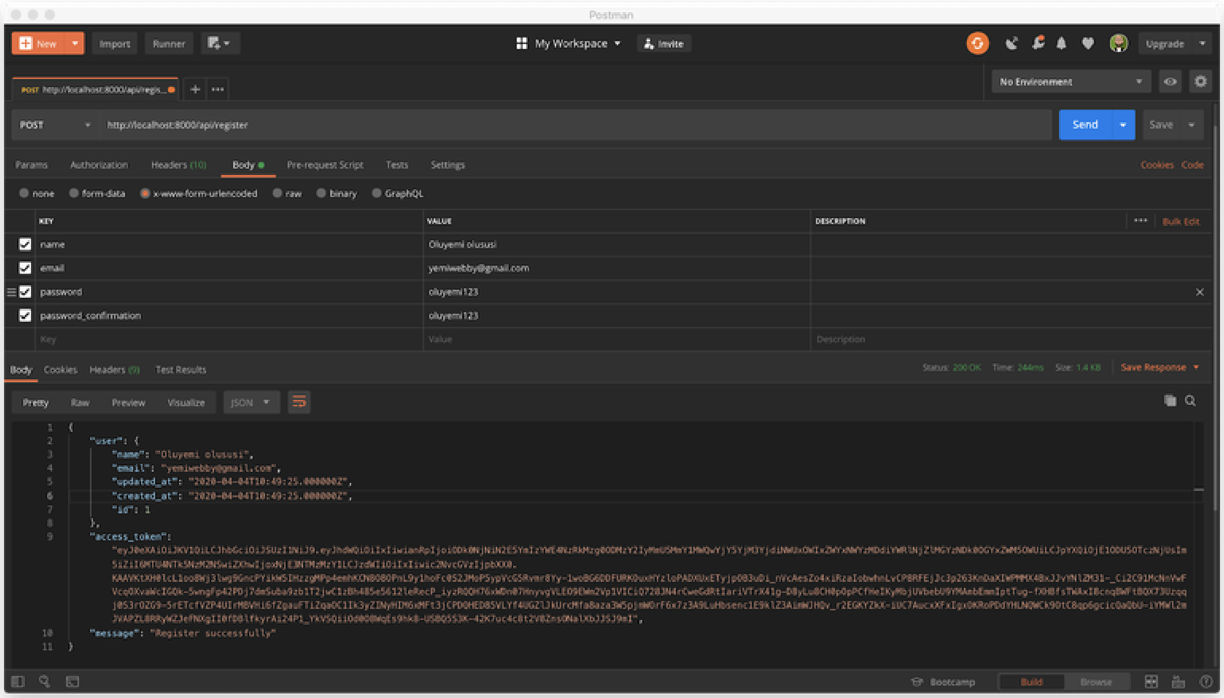
Now, test all the logic implemented so far by running the application with:

```
$ php artisan serve
```

We will use Postman for the remainder of this tutorial to test the endpoints. Download it here if you don’t have it installed on your machine.

Register user

To register a user, send a POST HTTP request to this endpoint `http://localhost:8000/api/register` and input the appropriate details as shown here:



POST http://localhost:8000/api/register

KEY	VALUE	DESCRIPTION
name	Oluyemi olususi	
email	yemiwebby@gmail.com	
password	oluyemi123	
password_confirmation	oluyemi123	

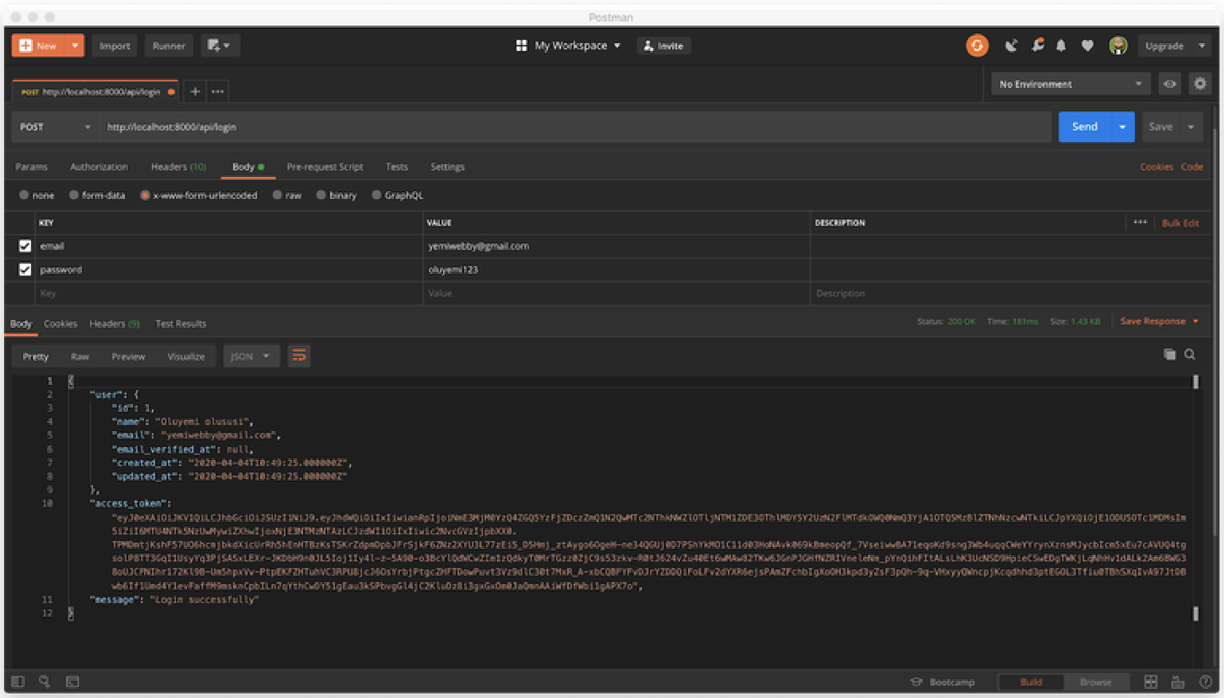
```
{
    "user": {
        "name": "Oluyemi olususi",
        "email": "yemiwebby@gmail.com",
        "updated_at": "2020-04-04T10:49:25.000000Z",
        "created_at": "2020-04-04T10:49:25.000000Z",
        "id": 1
    },
    "access_token":
        "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.[illegible]
        5iZiI6MTU4NTk5NzM2NSwiZXhwIjoxNjE3NTMzMzY1LCJzdWIiOiIxIiwic2NvcGVzIjpbXX0.
        [illegible]",
    "message": "Register successfully"
}
```

Now, your details might not be similar to this, but here is what the key and value of your requests should look like:

KEY	VALUE
name	John Doe
email	john@me.com
password	password
password_confirmation	password

Login

Once the registration is successful, you can proceed to `http://localhost:8000/api/login` and enter your details to get authenticated:



POST http://localhost:8000/api/login

none | form-data | ● x-www-form-urlencoded | raw | binary | GraphQL

	KEY	VALUE	DESCRIPTION
✓	email	yemiwebby@gmail.com	
✓	password	oluyemi123	
	Key	Value	Description

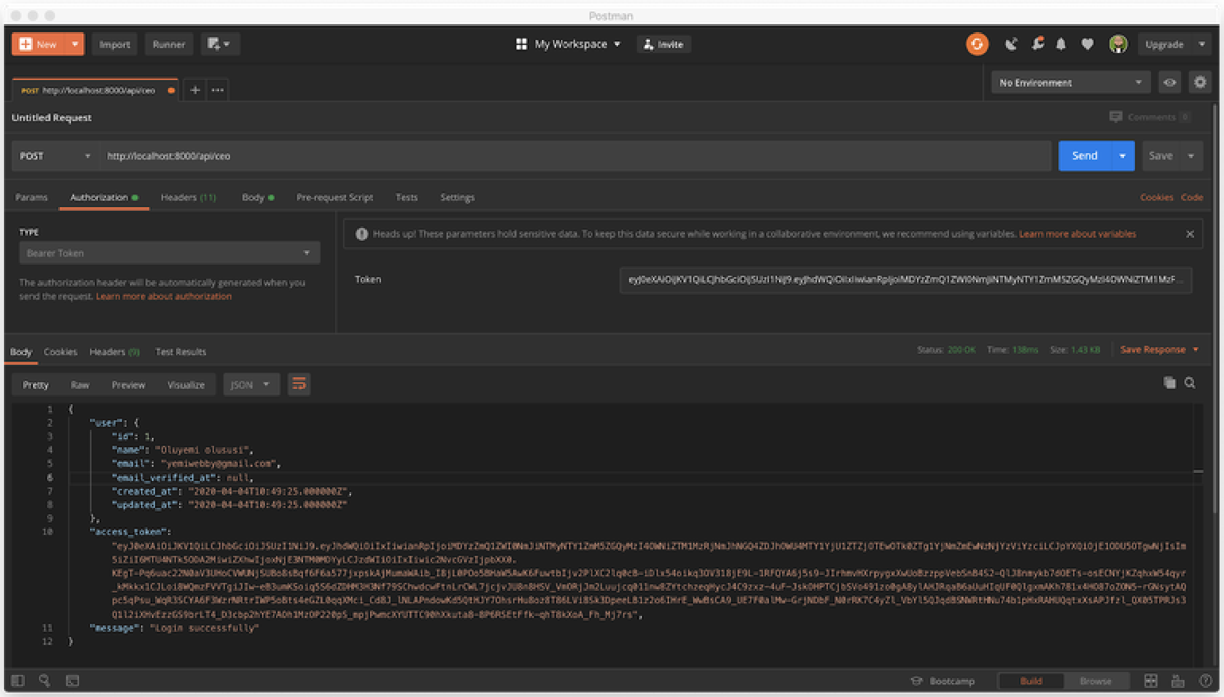
```
{
    "user": {
        "id": 1,
        "name": "Oluyemi olususi",
        "email": "yemiwebby@gmail.com",
        "email_verified_at": null,
        "created_at": "2020-04-04T10:49:25.000000Z",
        "updated_at": "2020-04-04T10:49:25.000000Z"
    },
    "access_token":
        "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.[illegible]",
    "message": "Login successfully"
}
```

Now, assuming that you used the same key and value specified in the previous section, your request should be similar to this:

```
KEY                                         VALUE
email                                      john@me.com
password                                    password
```

Add Bearer token

After the login, copy the value of the `access_token` from the response and click on the `Authorization` tab and select `Bearer Token` from the dropdown and paste the value of the `access_token` copied earlier:



POST http://localhost:8000/api/ceo

TYPE: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about authorization

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables

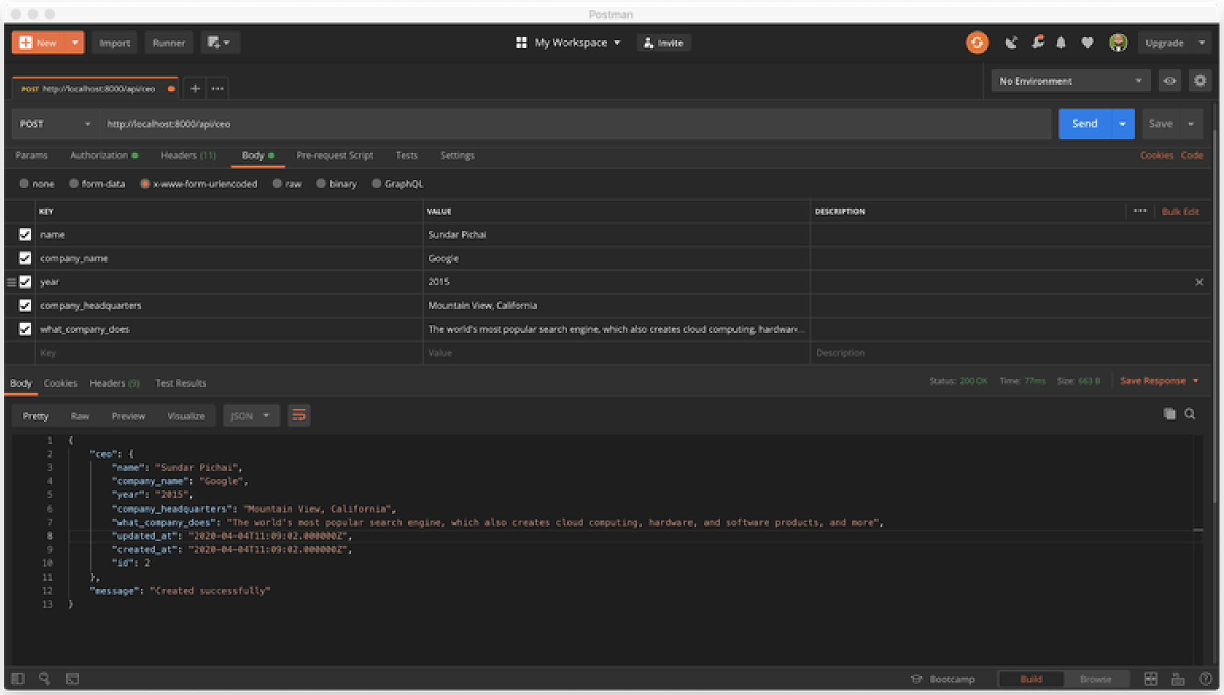
Token: eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIxIiwianRpIjoiMDYzZmQ1ZWI0NmJiNTMyNTY1ZmM5ZGQyMzI4OWNiZTM1MzF...

Body · Cookies · Headers (9) · Test Results — Status: 200 OK Time: 138ms Size: 1.43 KB

```
{
    "user": {
        "id": 1,
        "name": "Oluyemi olususi",
        "email": "yemiwebby@gmail.com",
        "email_verified_at": null,
        "created_at": "2020-04-04T10:49:25.000000Z",
        "updated_at": "2020-04-04T10:49:25.000000Z"
    },
    "access_token":
        "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIxIiwianRpIjoiMDYzZmQ1ZWI0NmJiNTMyNTY1ZmM5ZGQyMzI4OWNiZTM1MzRjNmJhNGQ4ZDJhOWU4MTY1YjU1ZTZjOTEwOTk0ZTg1YjNmZmEwNzNjYzViYzciLCJpYXQiOjE1ODU5OTgwNjIsIm
        5iZiI6MTU4NTk5ODA2MiwiZXhwIjoxNjE3NTM0MDYyLCJzdWIiOiIxIiwic2NvcGVzIjpbXX0.
        KEgT-Pq6uac22N0aV3UHoCVWUNjSUBo8sBqf6F6a577jxpskAjMumaWAib_I8jL0POo58HaW5AwK6FuwtbIjv2PlXC2lq0cB-iDlx54oikq3OV318jE9L-1RFQYA6j5s9-JIrhmvHXrpygxXwUoBzzppVebSnB4S2-Ql38nmykb7d0ETs-osECNYjKZqhxW54qyr
        _kMkkx1CJLoi8WQmzFVVTgiJIw-eB3umKSoiq5S6dZDMM3H3Nf79SChwdcwFtnLrCWL7jcjvJU8n8HSV_VmORjJm2Luujcq011nw8ZYtchzeqHycJ4C9zxz-4uF-JskDHPTCjbSVo491zo0gA8ylAHJRqaB6aUuHIqUF0QlgxmAKh781x4HO87oZON5-rGNsytAQ
        pc5qPsu_WqR3SCYA6F3WzrNRtrIWP5oBts4eGZL0qqXMci_CdBJ_UNLAPndowKd5QtHJY7OhsrHuBoz8T86LVi8Sk3DpeeL81z2o6IHrE_WwBsCA9_UE7F0alMw-GrjNDbF_N0rRK7C4yZl_VbYl5QJqdBSNWRtHNu74b1pHxRAHUQqtxXsAPJfzl_QX05TPRJs3
        Q1l2iXHvEzrGS9brLT4_D3cbp2hYE7ADh1MzOP220pS_mpjPwmcXYUTTC90hXkuta8-8P6RSEtFfk-qhT8kXoA_Fh_Mj7rs",
    "message": "Login successfully"
}
```

Create CEO

Next, create a new CEO with the details similar to the one shown below:



POST http://localhost:8000/api/ceo

none · form-data · [✓ x-www-form-urlencoded] · raw · binary · GraphQL

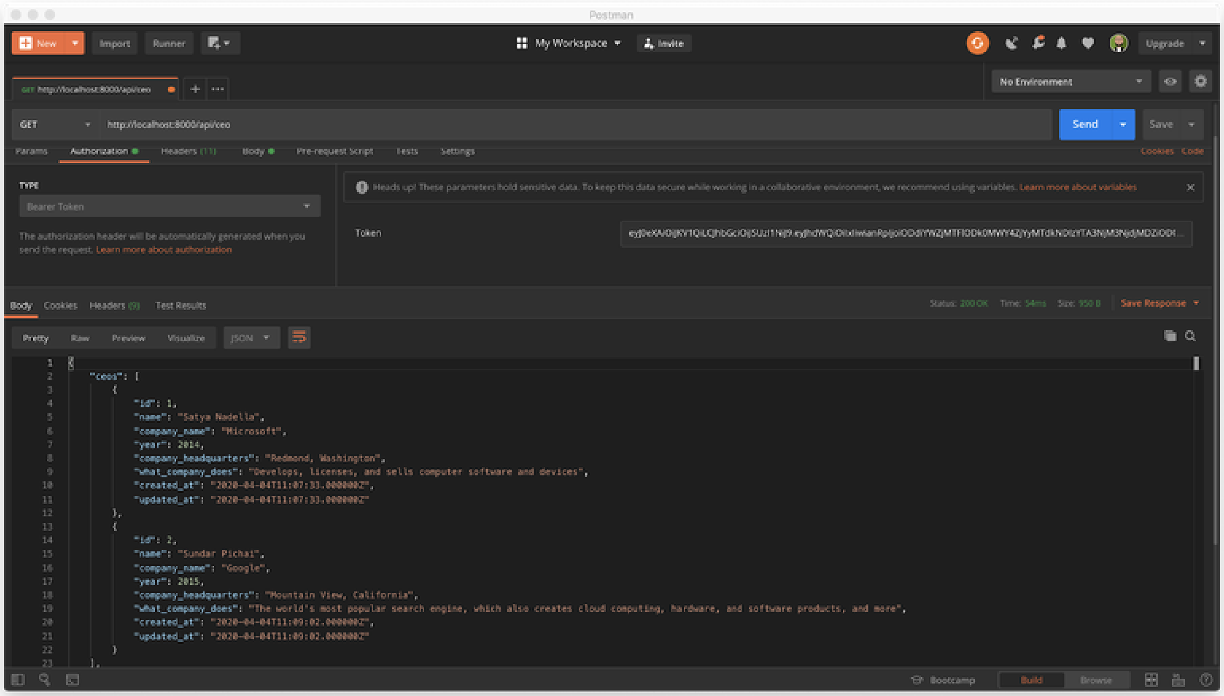
KEY	VALUE	DESCRIPTION
name	Sundar Pichai	
company_name	Google	
year	2015	
company_headquarters	Mountain View, California	
what_company_does	The world's most popular search engine, which also creates cloud computing, hardware...	
Key	Value	Description

Body · Cookies · Headers (9) · Test Results — Status: 200 OK Time: 77ms Size: 663 B

```
{
    "ceo": {
        "name": "Sundar Pichai",
        "company_name": "Google",
        "year": "2015",
        "company_headquarters": "Mountain View, California",
        "what_company_does": "The world's most popular search engine, which also creates cloud computing, hardware, and software products, and more",
        "updated_at": "2020-04-04T11:09:02.000000Z",
        "created_at": "2020-04-04T11:09:02.000000Z",
        "id": 2
    },
    "message": "Created successfully"
}
```

Fetch the list of CEOs

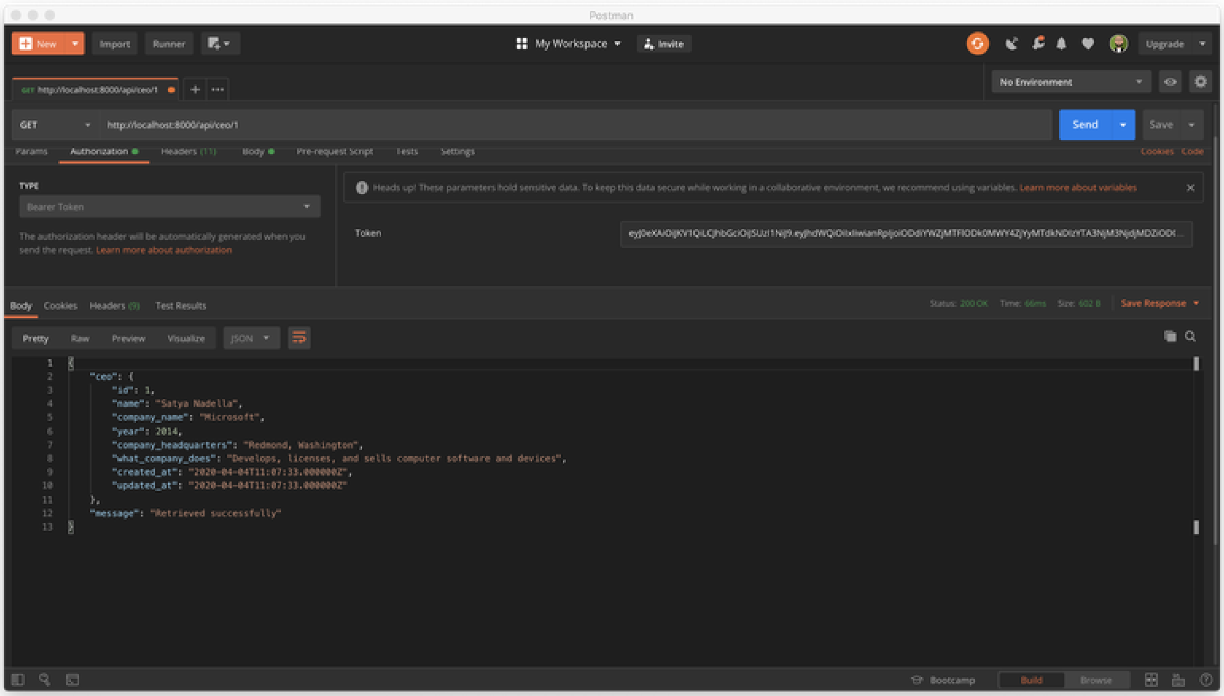
To retrieve the list of CEOs created so far, send a `GET` HTTP request to this endpoint `http://localhost:8000/api/ceo` as shown below:



```
{
    "ceos": [
        {
            "id": 1,
            "name": "Satya Nadella",
            "company_name": "Microsoft",
            "year": 2014,
            "company_headquarters": "Redmond, Washington",
            "what_company_does": "Develops, licenses, and sells computer software and devices",
            "created_at": "2020-04-04T11:07:33.000000Z",
            "updated_at": "2020-04-04T11:07:33.000000Z"
        },
        {
            "id": 2,
            "name": "Sundar Pichai",
            "company_name": "Google",
            "year": 2015,
            "company_headquarters": "Mountain View, California",
            "what_company_does": "The world's most popular search engine, which also creates cloud computing, hardware, and software products, and more",
            "created_at": "2020-04-04T11:09:02.000000Z",
            "updated_at": "2020-04-04T11:09:02.000000Z"
        }
    ],
```

Show CEO

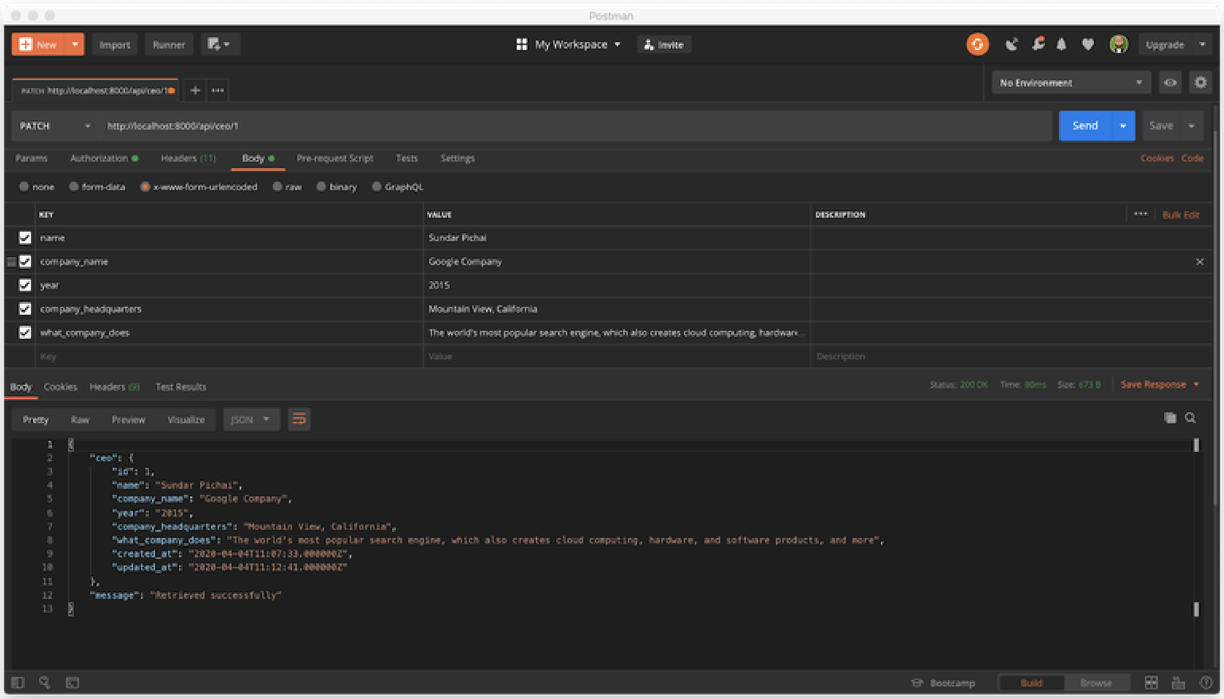
View the details of a particular CEO by sending a `GET` HTTP request to this URL `http://localhost:8000/api/ceo/1` :



```
{
    "ceo": {
        "id": 1,
        "name": "Satya Nadella",
        "company_name": "Microsoft",
        "year": 2014,
        "company_headquarters": "Redmond, Washington",
        "what_company_does": "Develops, licenses, and sells computer software and devices",
        "created_at": "2020-04-04T11:07:33.000000Z",
        "updated_at": "2020-04-04T11:07:33.000000Z"
    },
    "message": "Retrieved successfully"
}
```

Please note that `1` is used for the endpoint here to specify the `id` of a particular record that needs to be retrieved from the database. You can target any record here using the `id`.

Edit CEO



PATCH http://localhost:8000/api/ceo/1

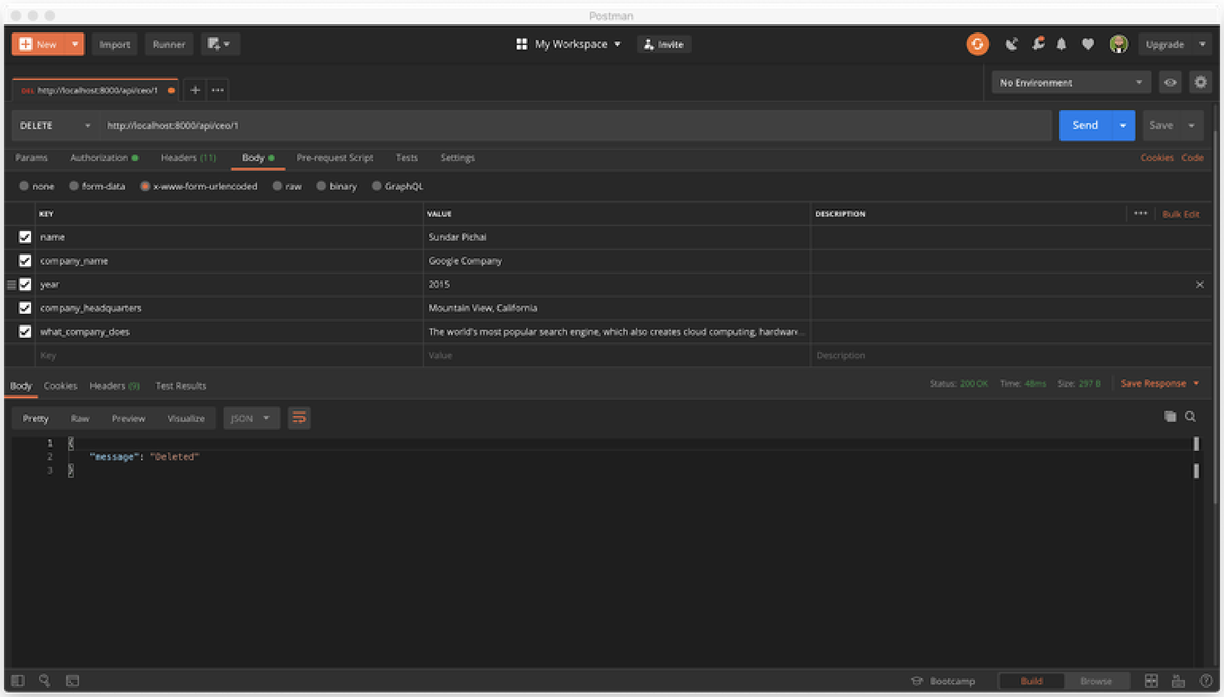
Body: x-www-form-urlencoded

KEY	VALUE	DESCRIPTION
name	Sundar Pichai	
company_name	Google Company	
year	2015	
company_headquarters	Mountain View, California	
what_company_does	The world's most popular search engine, which also creates cloud computing, hardware…	

Status: 200 OK Time: 80ms Size: 673 B

```
{
    "ceo": {
        "id": 1,
        "name": "Sundar Pichai",
        "company_name": "Google Company",
        "year": "2015",
        "company_headquarters": "Mountain View, California",
        "what_company_does": "The world's most popular search engine, which also creates cloud computing, hardware, and software products, and more",
        "created_at": "2020-04-04T11:07:33.000000Z",
        "updated_at": "2020-04-04T11:12:41.000000Z"
    },
    "message": "Retrieved successfully"
}
```

Delete CEO



DELETE http://localhost:8000/api/ceo/1

Body: x-www-form-urlencoded

KEY	VALUE	DESCRIPTION
name	Sundar Pichai	
company_name	Google Company	
year	2015	
company_headquarters	Mountain View, California	
what_company_does	The world's most popular search engine, which also creates cloud computing, hardware…	

Status: 200 OK Time: 48ms Size: 297 B

```
{
    "message": "Deleted"
}
```

Conclusion

In this tutorial, we have learned how to secure any RESTful API built with Laravel using Laravel Passport. The example created in this tutorial covers the basic CRUD (create, read, update and delete) processes required by most applications.

I hope this gives you a solid foundation that can be improved on for your existing project and new ones. Please feel free to explore the codebase of this application by downloading it here on GitHub.

For more information about Laravel and Laravel Passport, don’t hesitate to visit the official documentation.

Olususi Oluyemi is a tech enthusiast, programming freak, and a web development junkie who loves to embrace new technology.

- Twitter: https://twitter.com/yemiwebby
- GitHub: https://github.com/yemiwebby
- Website: https://yemiwebby.com.ng/

RATE THIS POST ★★★★★

AUTHORS Oluyemi Olususi

POSTS BY PRODUCT

SMS AUTHY VOICE TWILIO CLIENT MMS VIDEO TASK ROUTER FLEX SIP IOT

PROGRAMMABLE CHAT STUDIO

CATEGORIES

Code, Tutorials and Hacks

Customer Highlights

Developers Drawing The Owl

News

Stories From The Road

The Owl's Nest: Inside Twilio

TWITTER FACEBOOK

Developer stories to your inbox.

Subscribe to the Developer Digest, a monthly dose of all things code.

Enter your email…

You may unsubscribe at any time using the unsubscribe link in the digest email. See our privacy policy for more information.

NEW!

Tutorials

Sample applications that cover common use cases in a variety of languages. Download, test drive, and tweak them yourself.

Get started

SIGN UP AND START BUILDING

Not ready yet? Talk to an expert.