Oct 27, 2016 · 6 min read

Laravel RESTful API Development, A Step By Step Approach (PART 1)

Representational State Transfer (REST) or RESTful web services are one way of providing interoperability between computer systems on the internet. This services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations. Also in computer programming, an application programming interface (API) is a set of subroutine definitions, protocols and tools for building applications. A good API makes it easier to develop a program by providing all the building blocks, which are then put together by the programmer. Therefor a RESTful API is an application program interface that uses HTTP requests to GET, PUT, POST and DELETE data. For this article i would be using the Laravel 5.3 to develop the RESTful API project.

For this article I would be building a blog application using Laravel 5.3

Lets start by creating a new Laravel project.

Step 1

Open your terminal or command prompt and type the following commands. Make sure you have composer installed on your machine.

```
composer create-project --prefer-dist laravel/laravel
my-blog
```

cd into the “my-blog” project folder, open the composer.json file and add the following dependencies.

```
"tymon/jwt-auth": "0.5.*",
"barryvdh/laravel-cors": "^0.8.2"
```

Then run

```
composer update
```

Step 2

In the app.php config file, under the providers array, add the following

```
Tymon\JWTAuth\Providers\JWTAuthServiceProvider::class,
Barryvdh\Cors\ServiceProvider::class
```

Then also under the aliases array, add the following

```
'JWTAuth' => Tymon\JWTAuth\Facades\JWTAuth::class,
'JWTFactory' => Tymon\JWTAuth\Facades\JWTFactory::class,
```

On your terminal or command prompt run

```
php artisan vendor:publish
```

Add

```
"cors" => \Barryvdh\Cors\HandleCors::class,
```

to your route middle ware in the \App\Http\Kernel.php

Step 3

By default Laravel 5.3 comes with users table and password reset table migration files. Open the terminal or command prompt and run the command below

```
php artisan make:migration create_articles_table
```

This would create a file in the \database\migrations\ folder with the time stamp appended before the file name, e.g "2016_10_26_155720_create_articles_table.php".

Inside the file, add the following code block

```
<?php

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


class CreateArticlesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('articles', function (Blueprint
$table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned();
            $table->foreign('user_id')-
>references('id')->on('users');
```

```
            $table->string('title');
            $table->string('slug')->unique();
            $table->text('excerpts');
            $table->text('body');
            $table->timestamps();
        });
    }

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

Run the command below in the terminal or command prompt

```
php artisan migrate
```

This would creating the tables inside the database.

Step 4

Create a new controller called ApiController by running this command on the terminal or command prompt

```
php artisan make:controller ApiController
```

```
<?php

namespace App\Http\Controllers;
use Illuminate\Pagination\LengthAwarePaginator as
Paginator;
use Response;
use \Illuminate\Http\Response as Res;
```

```
/**
 * Class ApiController
 * @package App\Modules\Api\Lesson\Controllers
 */
class ApiController extends Controller{
```

```
    /**
     * Create a new authentication controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->beforeFilter('auth', ['on' => 'post']);
    }
```

```
    /**
     * @var int
     */
    protected $statusCode = Res::HTTP_OK;
```

```
    /**
     * @return mixed
     */
    public function getStatusCode()
    {
        return $this->statusCode;
    }


    /**
     * @param $message
     * @return json response
     */
    public function setStatusCode($statusCode)
    {
        $this->statusCode = $statusCode;
```

```
        return $this;
    }
```

```
    public function respondCreated($message, $data=null)
{
```

```
        return $this->respond([
```

```
            'status' => 'success',
            'status_code' => Res::HTTP_CREATED,
            'message' => $message,
            'data' => $data


        ]);
```

```
    }
```

```
    /**
     * @param Paginator $paginate
     * @param $data
```

```
     * @return mixed
     */
    protected function respondWithPagination(Paginator
$paginate, $data, $message){

        $data = array_merge($data, [
            'paginator' => [
                'total_count'  => $paginate->total(),
                'total_pages' => ceil($paginate->total()
/ $paginate->perPage()),
                'current_page' => $paginate-
>currentPage(),
                'limit' => $paginate->perPage(),
            ]
        ]);

        return $this->respond([

            'status' => 'success',
            'status_code' => Res::HTTP_OK,
            'message' => $message,
            'data' => $data

        ]);
    }


    public function respondNotFound($message = 'Not
Found!'){

        return $this->respond([

            'status' => 'error',
            'status_code' => Res::HTTP_NOT_FOUND,
            'message' => $message,

        ]);

    }


    public function respondInternalError($message){

        return $this->respond([

            'status' => 'error',
            'status_code' =>
Res::HTTP_INTERNAL_SERVER_ERROR,
            'message' => $message,

        ]);
```

```
    }

    public function respondValidationError($message,
$errors){

        return $this->respond([

            'status' => 'error',
            'status_code' =>
Res::HTTP_UNPROCESSABLE_ENTITY,
            'message' => $message,
            'data' => $errors

        ]);

    }

    public function respond($data, $headers = []){

        return Response::json($data, $this-
>getStatusCode(), $headers);

    }

    public function respondWithError($message){
        return $this->respond([

            'status' => 'error',
            'status_code' => Res::HTTP_UNAUTHORIZED,
            'message' => $message,

        ]);
    }
}
```

This controller extends the base Controller class, which would be used to handle all the API request responses.

Step 5

Create a Transformer class inside the App\Repository\Transformers folder.

```
<?php namespace App\Repository\Transformers;
```

```
abstract class Transformer {

    /*
     * Transforms a collection of lessons
     * @param $lessons
     * @return array
     */
    public function transformCollection(array $items){

        return array_map([$this, 'transform'], $items);

    }

    public abstract function transform($item);

}
```

Then also create a UserTransformer class that would extend the Transfomer class.

```
<?php

namespace App\Repository\Transformers;


class UserTransformer extends Transformer{

    public function transform($user){

        return [
            'fullname' => $user->name,
            'email' => $user->email,
            'api_token' => $user->api_token,
        ];

    }

}
```

This class abstracts the data coming from the model by hiding the structure of the database, so there is no direct query from the API to the model.

Step 6

Next step is creating the UserController class that extends the ApiController class

```
php artisan make:controller UserController
```

```
<?php namespace App\Http\Controllers;
```

```
use App\User;
use Illuminate\Http\Request;
use App\Http\Requests;
use JWTAuth;
use Response;
use App\Repository\Transformers\UserTransformer;
use \Illuminate\Http\Response as Res;
use Validator;
use Tymon\JWTAuth\Exceptions\JWTException;


class UserController extends ApiController
{
    /**
     * @var \App\Repository\Transformers\UserTransformer
     * */
    protected $userTransformer;
```

```
    public function __construct(userTransformer
$userTransformer)
    {
```

```
        $this->userTransformer = $userTransformer;
```

```
    }
```

```
    /**
     * @description: Api user authenticate method
     * @author: Adelekan David Aderemi
     * @param: email, password
     * @return: Json String response
     */
    public function authenticate(Request $request)
    {
```

```
        $rules = array (
```

```
            'email' => 'required|email',
            'password' => 'required',
```

```
        );

        $validator = Validator::make($request->all(),
$rules);

        if ($validator-> fails()){

            return $this->respondValidationError('Fields
Validation Failed.', $validator->errors());

        }

        else{

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

            if($user){
                $api_token = $user->api_token;

                if ($api_token == NULL){
                    return $this-
>_login($request['email'], $request['password']);
                }

                try{

                    $user = JWTAuth::toUser($api_token);

                    return $this->respond([

                        'status' => 'success',
                        'status_code' => $this-
>getStatusCode(),
                        'message' => 'Already logged
in',
                        'user' => $this-
>userTransformer->transform($user)

                    ]);

                }catch(JWTException $e){

                    $user->api_token = NULL;
                    $user->save();

                    return $this-
>respondInternalError("Login Unsuccessful. An error
occurred while performing an action!");
```

```
                }
            }
            else{
                return $this->respondWithError("Invalid
Email or Password");
            }

        }

    }


    private function _login($email, $password)
    {

        $credentials = ['email' => $email, 'password' =>
$password];


        if ( ! $token = JWTAuth::attempt($credentials))
{

            return $this->respondWithError("User does
not exist!");

        }

        $user = JWTAuth::toUser($token);

        $user->api_token = $token;
        $user->save();

        return $this->respond([

            'status' => 'success',
            'status_code' => $this->getStatusCode(),
            'message' => 'Login successful!',
            'data' => $this->userTransformer-
>transform($user)


        ]);
    }


    /**
     * @description: Api user register method
     * @author: Adelekan David Aderemi
     * @param: lastname, firstname, username, email,
password
     * @return: Json String response
     */
```

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

        $rules = array (

            'name' => 'required|max:255',
            'email' =>
'required|email|max:255|unique:users',
            'password' => 'required|min:6|confirmed',
            'password_confirmation' => 'required|min:3'

        );

        $validator = Validator::make($request->all(),
$rules);

        if ($validator-> fails()){

            return $this->respondValidationError('Fields
Validation Failed.', $validator->errors());

        }

        else{

            $user = User::create([

                'name' => $request['name'],
                'email' => $request['email'],
                'password' =>
\Hash::make($request['password']),

            ]);

            return $this->_login($request['email'],
$request['password']);

        }

    }



    /**
     * @description: Api user logout method
     * @author: Adelekan David Aderemi
     * @param: null
     * @return: Json String response
     */
    public function logout($api_token)
    {
```

```
        try{

            $user = JWTAuth::toUser($api_token);

            $user->api_token = NULL;

            $user->save();

            JWTAuth::setToken($api_token)->invalidate();

            $this->setStatusCode(Res::HTTP_OK);

            return $this->respond([

                'status' => 'success',
                'status_code' => $this->getStatusCode(),
                'message' => 'Logout successful!',

            ]);


        }catch(JWTException $e){

            return $this->respondInternalError("An error
occurred while performing an action!");

        }

    }

}
```

Step 7

Then inside the api.php route file add then following

```
Route::group(['middleware' => 'cors', 'prefix' =>
'/v1'], function () {

    Route::post('/login',
'UserController@authenticate');

    Route::post('/register', 'UserController@register');
```

```
    Route::get('/logout/{api_token}',
'UserController@logout');

});
```

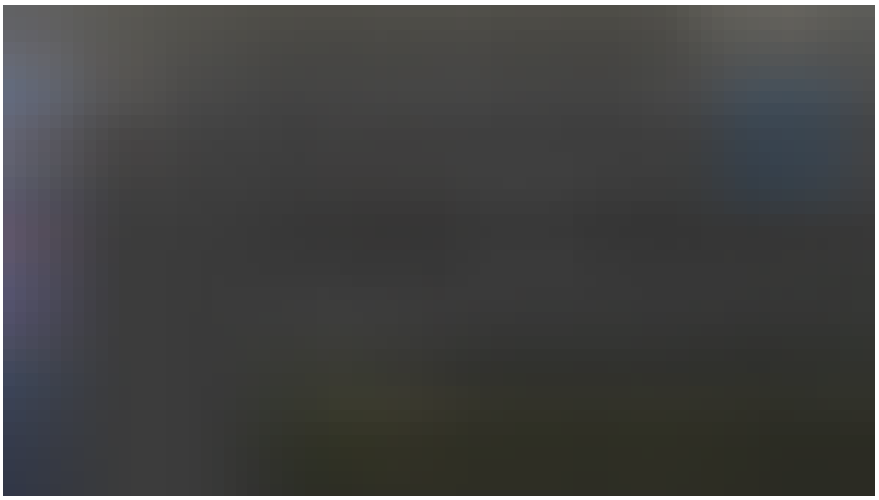
Step 8

Start the laravel local development server

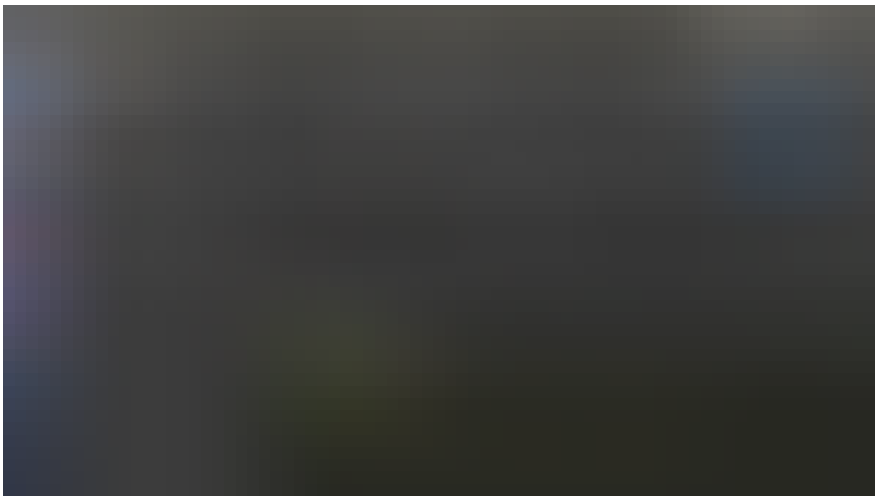
```
php artisan serve
```

Using an API testing tool called POSTMAN to test the registration, login and logout functionality. See screenshots below.

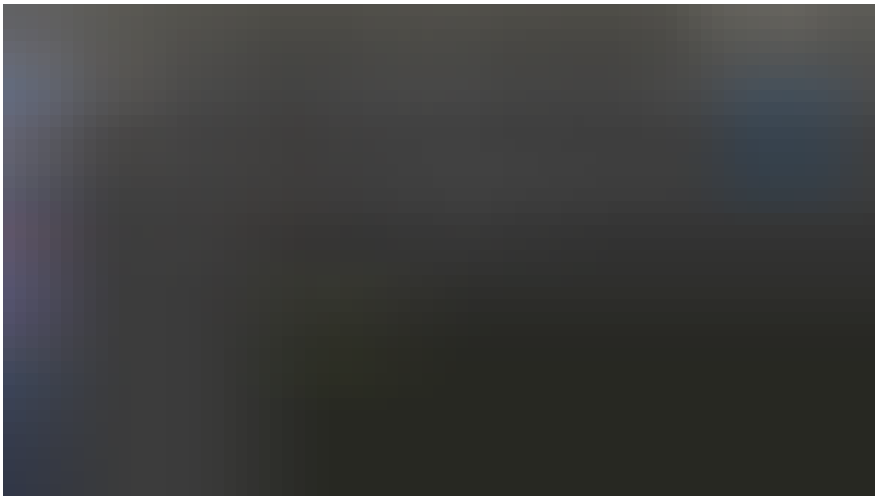
User Registration



User Login



User Logout



Conclusion

Hope you enjoyed this post and learnt much from it. The second part of this series would be coming up shortly. You can download the source code of this first part here.

If you have any comments, questions or observations, please feel free to drop them in the comment section below.

You could check our my blog for more information at https://adelekand.wordpress.com/