BACK-END 18 MINUTE READ

Laravel API Tutorial: How to Build and Test a RESTful API

ANDRÉ CASTELO

André Castelo is a web developer focused on PHP and JavaScript. He also developed Laravel apps and APIs, as well as AngularJS apps.

SHARE

With the rise of mobile development and JavaScript frameworks, using a RESTful API is the best option to build a single interface between your data and your client.

Laravel is a PHP framework developed with PHP developer productivity in mind. Written and maintained by Taylor Otwell, the framework is very opinionated and strives to save developer time by favoring convention over configuration. The framework also aims to evolve with the web and has already incorporated several new features and ideas in the web development world—such as job queues, API authentication out of the box, real-time communication, and much more.

In this tutorial, we'll explore the ways you can build—and test—a robust API using Laravel with authentication. We'll be using Laravel 5.4, and all of the code is available for reference on GitHub.

RESTful APIs

First, we need to understand what exactly is considered a RESTful API. REST stands for *REpresentational State Transfer* and is an architectural style for network communication between applications, which relies on a stateless protocol (usually HTTP) for interaction.

HTTP Verbs Represent Actions

In RESTful APIs, we use the HTTP verbs as actions, and the endpoints are the resources acted upon. We'll be using the HTTP verbs for their semantic meaning:

- `GET` : retrieve resources
- `POST` : create resources
- `PUT` : update resources
- `DELETE` : delete resources

Update Action: PUT vs. POST

RESTful APIs are a matter of much debate and there are plenty of opinions out there on whether is best to update with `POST` , `PATCH` , or `PUT` , or if the create action is best left to the `PUT` verb. In this article we'll be using `PUT` for the update action, as according to the HTTP RFC, `PUT` means to create/update a resource at a specific location. Another requirement for the `PUT` verb is idempotence, which in this case basically means you can send that request 1, 2 or 1000 times and the result will be the same: one updated resource in the database.

Resources

Resources will be the targets of the actions, in our case Articles and Users, and they have their own endpoints:

- `/articles`
- `/users`

In this laravel api tutorial, the resources will have a 1:1 representation on our data models, but that is not a requirement. You can have resources represented in more than one data model (or not represented at all in the database) and models completely off limits for the user. In the end, you get to decide how to architect resources and models in a way that is fitting to your application.

A Note on Consistency

The greatest advantage of using a set of conventions such as REST is that your API will be much easier to consume and develop around. Some endpoints are pretty straightforward and, as a result, your API will be much more easier to use and maintain as opposed to having endpoints such as `GET /get_article?id_article=12` and `POST /delete_article?number=40`. I've built terrible APIs like that in the past and I still hate myself for it.

However, there will be cases where it will be hard to map to a Create/Retrieve/Update/Delete schema. Remember that the URLs should not contain verbs and that resources are not necessarily rows in a table. Another thing to keep in mind is that you don't have to implement every action for every resource.

Setting Up a Laravel Web Service Project

As with all modern PHP frameworks, we'll need Composer to install and handle our dependencies. After you follow the download instructions (and add to your path environment variable), install Laravel using the command:

```
$ composer global require laravel/installer
```

After the installation finishes, you can scaffold a new application like this:

```
$ laravel new myapp
```

For the above command, you need to have `~/composer/vendor/bin` in your `$PATH`. If you don't want to deal with that, you can also create a new project using Composer:

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

With Laravel installed, you should be able to start the server and test if everything is working:

```
$ php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
```

When you open `localhost:8000` on your browser, you should see this sample page.

Migrations and Models

Before actually writing your first migration, make sure you have a database created for this app and add its credentials to the `.env` file located in the root of the project.

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

> You can also use Homestead, a Vagrant box specially crafted for Laravel, but that is a bit out of the scope of this article. If you'd like to know more, refer to the Homestead documentation.

Let's get started with our first model and migration—the Article. The article should have a title and a body field, as well as a creation date. Laravel provides several commands through Artisan—Laravel's command line tool—that help us by generating files and putting them in the correct folders. To create the Article model, we can run:

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

The `-m` option is short for `--migration` and it tells Artisan to create one for our model. Here's the generated migration:

```php
<?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->timestamps();
        });
    }

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

Let’s dissect this for a second:

- The `up()` and `down()` methods will be run when we migrate and rollback respectively;
- `$table->increments('id')` sets up an auto incrementing integer with the name `id`;
- `$table->timestamps()` will set up the timestamps for us— `created_at` and `updated_at`, but don't worry about setting a default, Laravel takes care of updating these fields when needed.
- And finally, `Schema::dropIfExists()` will, of course, drop the table if it exists.

With that out of the way, let’s add two lines to our `up()` method:

```php
public function up()
{
    Schema::create('articles', function (Blueprint $table) {
        $table->increments('id');
        $table->string('title');
        $table->text('body');
        $table->timestamps();
    });
}
```

The `string()` method creates a `VARCHAR` equivalent column while `text()` creates a `TEXT` equivalent. With that done, let’s go ahead and migrate:

```
$ php artisan migrate
```

“

> You can also use the `--step` option here, and it will separate each migration into its own batch so that you can roll them back individually if needed.

Laravel out of the box comes with two migrations, `create_users_table` and `create_password_resets_table`. We won't be using the `password_resets` table, but having the `users` table ready for us will be helpful.

Now let's go back to our model and add those attributes to the `$fillable` field so that we can use them in our `Article::create` and `Article::update` models:

```
class Article extends Model
{
    protected $fillable = ['title', 'body'];
}
```

> Fields inside the `$fillable` property can be mass assigned using Eloquent's `create()` and `update()` methods. You can also use the `$guarded` property, to allow all but a few properties.

Database Seeding

Database seeding is the process of filling up our database with dummy data that we can use to test it. Laravel comes with Faker, a great library for generating just the correct format of dummy data for us. So let's create our first seeder:

```
$ php artisan make:seeder ArticlesTableSeeder
```

The seeders will be located in the `/database/seeds` directory. Here's how it looks like after we set it up to create a few articles:

```
class ArticlesTableSeeder extends Seeder
{
    public function run()
    {
        // Let's truncate our existing records to start from scratch.
        Article::truncate();

        $faker = \Faker\Factory::create();

        // And now, let's create a few articles in our database:
        for ($i = 0; $i < 50; $i++) {
            Article::create([
                'title' => $faker->sentence,
                'body' => $faker->paragraph,
            ]);
        }
    }
}
```

So let's run the seed command:

```
$ php artisan db:seed --class=ArticlesTableSeeder
```

Let’s repeat the process to create a Users seeder:

```
class UsersTableSeeder extends Seeder
{
    public function run()
    {
        // Let's clear the users table first
        User::truncate();

        $faker = \Faker\Factory::create();

        // Let's make sure everyone has the same password and
        // let's hash it before the loop, or else our seeder
        // will be too slow.
        $password = Hash::make('toptal');

        User::create([
            'name' => 'Administrator',
            'email' => 'admin@test.com',
            'password' => $password,
        ]);

        // And now let's generate a few dozen users for our app:
        for ($i = 0; $i < 10; $i++) {
            User::create([
                'name' => $faker->name,
                'email' => $faker->email,
                'password' => $password,
            ]);
        }
    }
}
```

We can make it easier by adding our seeders to the main `DatabaseSeeder` class inside the `database/seeds` folder:

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(ArticlesTableSeeder::class);
        $this->call(UsersTableSeeder::class);
    }
}
```

This way, we can simply run `$ php artisan db:seed` and it will run all the called classes in the `run()` method.

Routes and Controllers

Let’s create the basic endpoints for our application: create, retrieve the list, retrieve a single one, update, and delete. On the `routes/api.php` file, we can simply do this:

```
Use App\Article;

Route::get('articles', function() {
    // If the Content-Type and Accept headers are set to 'application/json',
    // this will return a JSON structure. This will be cleaned up later.
    return Article::all();
});

Route::get('articles/{id}', function($id) {
    return Article::find($id);
});

Route::post('articles', function(Request $request) {
    return Article::create($request->all);
});

Route::put('articles/{id}', function(Request $request, $id) {
    $article = Article::findOrFail($id);
    $article->update($request->all());

    return $article;
});

Route::delete('articles/{id}', function($id) {
    Article::find($id)->delete();

    return 204;
})
```

The routes inside `api.php` will be prefixed with `/api/` and the API throttling middleware will be automatically applied to these routes (if you want to remove the prefix you can edit the `RouteServiceProvider` class on `/app/Providers/RouteServiceProvider.php`).

Now let’s move this code to its own Controller:

```
$ php artisan make:controller ArticleController
```

ArticleController.php:

```
use App\Article;

class ArticleController extends Controller
{
    public function index()
    {
        return Article::all();
    }

    public function show($id)
    {
        return Article::find($id);
    }

    public function store(Request $request)
    {
        return Article::create($request->all());
    }

    public function update(Request $request, $id)
    {
        $article = Article::findOrFail($id);
        $article->update($request->all());

        return $article;
    }

    public function delete(Request $request, $id)
    {
        $article = Article::findOrFail($id);
        $article->delete();

        return 204;
    }
}
```

The `routes/api.php` file:

```
Route::get('articles', 'ArticleController@index');
Route::get('articles/{id}', 'ArticleController@show');
Route::post('articles', 'ArticleController@store');
Route::put('articles/{id}', 'ArticleController@update');
Route::delete('articles/{id}', 'ArticleController@delete');
```

We can improve the endpoints by using implicit route model binding. This way, Laravel will inject the `Article` instance in our methods and automatically return a 404 if it isn't found. We'll have to make changes on the routes file and on the controller:

```
Route::get('articles', 'ArticleController@index');
Route::get('articles/{article}', 'ArticleController@show');
Route::post('articles', 'ArticleController@store');
Route::put('articles/{article}', 'ArticleController@update');
Route::delete('articles/{article}', 'ArticleController@delete');
```

```
class ArticleController extends Controller
{
    public function index()
    {
        return Article::all();
    }

    public function show(Article $article)
    {
        return $article;
    }

    public function store(Request $request)
    {
        $article = Article::create($request->all());

        return response()->json($article, 201);
    }

    public function update(Request $request, Article $article)
    {
        $article->update($request->all());

        return response()->json($article, 200);
    }

    public function delete(Article $article)
    {
        $article->delete();

        return response()->json(null, 204);
    }
}
```

A Note on HTTP Status Codes and the Response Format

We’ve also added the `response()->json()` call to our endpoints. This lets us explicitly return JSON data as well as send an HTTP code that can be parsed by the client. The most common codes you’ll be returning will be:

- `200` : OK. The standard success code and default option.
- `201` : Object created. Useful for the `store` actions.
- `204` : No content. When an action was executed successfully, but there is no content to return.
- `206` : Partial content. Useful when you have to return a paginated list of resources.
- `400` : Bad request. The standard option for requests that fail to pass validation.
- `401` : Unauthorized. The user needs to be authenticated.
- `403` : Forbidden. The user is authenticated, but does not have the permissions to perform an action.
- `404` : Not found. This will be returned automatically by Laravel when the resource is not found.
- `500` : Internal server error. Ideally you're not going to be explicitly returning this, but if something unexpected breaks, this is what your user is going to receive.

- `503` : Service unavailable. Pretty self explanatory, but also another code that is not going to be returned explicitly by the application.

Sending a Correct 404 Response

If you tried to fetch a non-existent resource, you'll be thrown an exception and you'll receive the whole stacktrace, like this:

We can fix that by editing our exception handler class, located in `app/Exceptions/Handler.php`, to return a JSON response:

```
public function render($request, Exception $exception)
{
    // This will replace our 404 response with
    // a JSON response.
    if ($exception instanceof ModelNotFoundException) {
        return response()->json([
            'error' => 'Resource not found'
        ], 404);
    }

    return parent::render($request, $exception);
}
```

Here's an example of the return:

```
{
    data: "Resource not found"
}
```

If you're using Laravel to serve other pages, you have to edit the code to work with the `Accept` header, otherwise 404 errors from regular requests will return a JSON as well.

```
public function render($request, Exception $exception)
{
    // This will replace our 404 response with
    // a JSON response.
    if ($exception instanceof ModelNotFoundException &&
        $request->wantsJson())
    {
        return response()->json([
            'data' => 'Resource not found'
        ], 404);
    }

    return parent::render($request, $exception);
}
```

In this case, the API requests will need the header `Accept: application/json`.

Authentication

There are many ways to implement API Authentication in Laravel (one of them being Passport, a great way to implement OAuth2), but in this article, we'll take a very simplified approach.

To get started, we'll need to add an `api_token` field to the `users` table:

```
$ php artisan make:migration --table=users adds_api_token_to_users_table
```

And then implement the migration:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('api_token', 60)->unique()->nullable();
    });
}

public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn(['api_token']);
    });
}
```

After that, just run the migration using:

```
$ php artisan migrate
```

Creating the Register Endpoint

We'll make use of the `RegisterController` (in the `Auth` folder) to return the correct response upon registration. Laravel comes with authentication out of the box, but we still need to tweak it a bit to return the response we want.

The controller makes use of the trait `RegistersUsers` to implement the registration. Here's how it works:

```
public function register(Request $request)
{
    // Here the request is validated. The validator method is located
    // inside the RegisterController, and makes sure the name, email
    // password and password_confirmation fields are required.
    $this->validator($request->all())->validate();

    // A Registered event is created and will trigger any relevant
    // observers, such as sending a confirmation email or any
    // code that needs to be run as soon as the user is created.
    event(new Registered($user = $this->create($request->all())));

    // After the user is created, he's logged in.
    $this->guard()->login($user);

    // And finally this is the hook that we want. If there is no
    // registered() method or it returns null, redirect him to
    // some other URL. In our case, we just need to implement
    // that method to return the correct response.
    return $this->registered($request, $user)
                    ?: redirect($this->redirectPath());
}
```

We just need to implement the `registered()` method in our `RegisterController`. The method receives the `$request` and the `$user`, so that's really all we want. Here's how the method should look like inside the controller:

```
protected function registered(Request $request, $user)
{
    $user->generateToken();

    return response()->json(['data' => $user->toArray()], 201);
}
```

And we can link it on the routes file:

```
Route::post(register, 'Auth\RegisterController@register);
```

In the section above, we used a method on the User model to generate the token. This is useful so that we only have a single way of generating the tokens. Add the following method to your User model:

```
class User extends Authenticatable
{
    ...
    public function generateToken()
    {
        $this->api_token = str_random(60);
        $this->save();

        return $this->api_token;
    }
}
```

And that's it. The user is now registered and thanks to Laravel's validation and out of the box authentication, the `name`, `email`, `password`, and `password_confirmation` fields are required, and the feedback is handled automatically. Checkout the `validator()` method inside the `RegisterController` to see how the rules are implemented.

Here's what we get when we hit that endpoint:

```
$ curl -X POST http://localhost:8000/api/register \
 -H "Accept: application/json" \
 -H "Content-Type: application/json" \
 -d '{"name": "John", "email": "john.doe@toptal.com", "password": "toptal123", "password_confirmation": "toptal123"}'
```

```
{
    "data": {
        "api_token":"0syHnl0Y9jOIfszq11EC2CBQwCfObmvscrZYo5o2ilZPnohvndH797nDNyAT",
        "created_at": "2017-06-20 21:17:15",
        "email": "john.doe@toptal.com",
        "id": 51,
        "name": "John",
        "updated_at": "2017-06-20 21:17:15"
    }
}
```

Creating a Login Endpoint

Just like the registration endpoint, we can edit the `LoginController` (in the `Auth` folder) to support our API authentication. The `login` method of the `AuthenticatesUsers` trait can be overridden to support our API:

```
public function login(Request $request)
{
    $this->validateLogin($request);

    if ($this->attemptLogin($request)) {
        $user = $this->guard()->user();
        $user->generateToken();

        return response()->json([
            'data' => $user->toArray(),
        ]);
    }

    return $this->sendFailedLoginResponse($request);
}
```

And we can link it on the routes file:

```
Route::post('login', 'Auth\LoginController@login');
```

Now, assuming the seeders have been run, here’s what we get when we send a `POST` request to that route:

```
$ curl -X POST localhost:8000/api/login \
  -H "Accept: application/json" \
  -H "Content-type: application/json" \
  -d "{\"email\": \"admin@test.com\", \"password\": \"toptal\" }"
```

```
{
    "data": {
        "id":1,
        "name":"Administrator",
        "email":"admin@test.com",
        "created_at":"2017-04-25 01:05:34",
        "updated_at":"2017-04-25 02:50:40",
        "api_token":"Jll7q0BSijLOrzaOSm5Dr5hW9cJRZAJKOzvDlxjKCXepwAeZ7JR6YP5zQqnw"
    }
}
```

To send the token in a request, you can do it by sending an attribute `api_token` in the payload or as a bearer token in the request headers in the form of `Authorization: Bearer Jll7q0BSijLOrzaOSm5Dr5hW9cJRZAJKOzvDlxjKCXepwAeZ7JR6YP5zQqnw`.

Logging Out

With our current strategy, if the token is wrong or missing, the user should receive an unauthenticated response (which we’ll implement in the next section). So for a simple logout endpoint, we’ll send in the token and it will be removed on the database.

`routes/api.php`:

```
Route::post('logout', 'Auth\LoginController@logout');
```

`Auth\LoginController.php`:

```
public function logout(Request $request)
{
    $user = Auth::guard('api')->user();

    if ($user) {
        $user->api_token = null;
        $user->save();
    }

    return response()->json(['data' => 'User logged out.'], 200);
}
```

Using this strategy, whatever token the user has will be invalid, and the API will deny access (using middlewares, as explained in the next section). This needs to be coordinated with the front-end to avoid the user remaining logged without having access to any content.

Using Middlewares to Restrict Access

With the `api_token` created, we can toggle the authentication middleware in the routes file:

```
Route::middleware('auth:api')
    ->get('/user', function (Request $request) {
        return $request->user();
    });
```

We can access the current user using the `$request->user()` method or through the Auth facade

```
Auth::guard('api')->user(); // instance of the logged user
Auth::guard('api')->check(); // if a user is authenticated
Auth::guard('api')->id(); // the id of the authenticated user
```

And we get a result like this:

This is because we need to edit the current `unauthenticated` method on our Handler class. The current version returns a JSON only if the request has the `Accept: application/json` header, so let's change it:

```
protected function unauthenticated($request, AuthenticationException $exception)
{
    return response()->json(['error' => 'Unauthenticated'], 401);
}
```

With that fixed, we can go back to the article endpoints to wrap them in the `auth:api` middleware. We can do that by using route groups:

```
Route::group(['middleware' => 'auth:api'], function() {
    Route::get('articles', 'ArticleController@index');
    Route::get('articles/{article}', 'ArticleController@show');
    Route::post('articles', 'ArticleController@store');
    Route::put('articles/{article}', 'ArticleController@update');
    Route::delete('articles/{article}', 'ArticleController@delete');
});
```

This way we don't have to set the middleware for each of the routes. It doesn't save a lot of time right now, but as the project grows it helps to keep the routes DRY.

Testing Our Endpoints

Laravel includes integration with PHPUnit out of the box with a `phpunit.xml` already set up. The framework also provides us with several helpers and extra assertions that makes our lives much easier, especially for testing APIs.

There are a number of external tools you can use to test your API; however, testing inside Laravel is a much better alternative—we can have all the benefits of testing an API structure and results while retaining full control of the database. For the list endpoint, for example, we could run a couple of factories and assert the response contains those resources.

To get started, we'll need to tweak a few settings to use an in-memory SQLite database. Using that will make our tests run lightning fast, but the trade-off is that some migration commands (constraints, for example) will not work properly in that particular setup. I advise moving away from SQLite in testing when you start getting migration errors or if you prefer a stronger set of tests instead of performant runs.

We'll also run the migrations before each test. This setup will allow us to build the database for each test and then destroy it, avoiding any type of dependency between tests.

In our `config/database.php` file, we'll need to set up the `database` field in the `sqlite` configuration to `:memory:`:

```
...
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' => ':memory:',
        'prefix' => '',
    ],

    ...
]
```

Then enable SQLite in `phpunit.xml` by adding the environment variable `DB_CONNECTION`:

```
    <php>
        <env name="APP_ENV" value="testing"/>
        <env name="CACHE_DRIVER" value="array"/>
        <env name="SESSION_DRIVER" value="array"/>
        <env name="QUEUE_DRIVER" value="sync"/>
        <env name="DB_CONNECTION" value="sqlite"/>
    </php>
```

With that out of the way, all that's left is configuring our base `TestCase` class to use migrations and seed the database before each test. To do so, we need to add the `DatabaseMigrations` trait, and then add an `Artisan` call on our `setUp()` method. Here's the class after the changes:

```
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\TestCase as BaseTestCase;
use Illuminate\Support\Facades\Artisan;

abstract class TestCase extends BaseTestCase
{
    use CreatesApplication, DatabaseMigrations;

    public function setUp()
    {
        parent::setUp();
        Artisan::call('db:seed');
    }
}
```

One last thing that I like to do is to add the test command to `composer.json`:

```
    "scripts": {
        "test" : [
            "vendor/bin/phpunit"
        ],
    ...
    },
```

The test command will be available like this:

```
$ composer test
```

Setting Up Factories for Our Tests

Factories will allow us to quickly create objects with the right data for testing. They're located in the `database/factories` folder. Laravel comes out of the box with a factory for the `User` class, so let's add one for the `Article` class:

```
$factory->define(App\Article::class, function (Faker\Generator $faker) {
    return [
        'title' => $faker->sentence,
        'body' => $faker->paragraph,
    ];
});
```

The Faker library is already injected to help us create the correct format of random data for our models.

Our First Tests

We can use Laravel’s assert methods to easily hit an endpoint and evaluate its response. Let’s create our first test, the login test, using the following command:

```
$ php artisan make:test Feature/LoginTest
```

And here is our test:

```
class LoginTest extends TestCase
{
    public function testRequiresEmailAndLogin()
    {
        $this->json('POST', 'api/login')
            ->assertStatus(422)
            ->assertJson([
                'email' => ['The email field is required.'],
                'password' => ['The password field is required.'],
            ]);
    }


    public function testUserLoginsSuccessfully()
    {
        $user = factory(User::class)->create([
            'email' => 'testlogin@user.com',
            'password' => bcrypt('toptal123'),
        ]);

        $payload = ['email' => 'testlogin@user.com', 'password' => 'toptal123'];

        $this->json('POST', 'api/login', $payload)
            ->assertStatus(200)
            ->assertJsonStructure([
                'data' => [
                    'id',
                    'name',
                    'email',
                    'created_at',
                    'updated_at',
                    'api_token',
                ],
            ]);

    }
}
```

These methods test a couple of simple cases. The `json()` method hits the endpoint and the other asserts are pretty self explanatory. One detail about `assertJson()`: this method converts the response into an array searches for the argument, so the order is important. You can chain multiple `assertJson()` calls in that case.

Now, let’s create the register endpoint test and write a couple for that endpoint:

```
$ php artisan make:test RegisterTest
```

```
class RegisterTest extends TestCase
{
    public function testsRegistersSuccessfully()
    {
        $payload = [
            'name' => 'John',
            'email' => 'john@toptal.com',
            'password' => 'toptal123',
            'password_confirmation' => 'toptal123',
        ];

        $this->json('post', '/api/register', $payload)
            ->assertStatus(201)
            ->assertJsonStructure([
                'data' => [
                    'id',
                    'name',
                    'email',
                    'created_at',
                    'updated_at',
                    'api_token',
                ],
            ]);;
    }

    public function testsRequiresPasswordEmailAndName()
    {
        $this->json('post', '/api/register')
            ->assertStatus(422)
            ->assertJson([
                'name' => ['The name field is required.'],
                'email' => ['The email field is required.'],
                'password' => ['The password field is required.'],
            ]);
    }

    public function testsRequirePasswordConfirmation()
    {
        $payload = [
            'name' => 'John',
            'email' => 'john@toptal.com',
            'password' => 'toptal123',
        ];

        $this->json('post', '/api/register', $payload)
            ->assertStatus(422)
            ->assertJson([
                'password' => ['The password confirmation does not match.'],
            ]);
    }
}
```

And lastly, the logout endpoint:

```
$ php artisan make:test LogoutTest
```

```
class LogoutTest extends TestCase
{
    public function testUserIsLoggedOutProperly()
    {
        $user = factory(User::class)->create(['email' => 'user@test.com']);
        $token = $user->generateToken();
        $headers = ['Authorization' => "Bearer $token"];

        $this->json('get', '/api/articles', [], $headers)->assertStatus(200);
        $this->json('post', '/api/logout', [], $headers)->assertStatus(200);

        $user = User::find($user->id);

        $this->assertEquals(null, $user->api_token);
    }

    public function testUserWithNullToken()
    {
        // Simulating login
        $user = factory(User::class)->create(['email' => 'user@test.com']);
        $token = $user->generateToken();
        $headers = ['Authorization' => "Bearer $token"];

        // Simulating logout
        $user->api_token = null;
        $user->save();

        $this->json('get', '/api/articles', [], $headers)->assertStatus(401);
    }
}
```

“

It’s important to note that, during testing, the Laravel application is not instantiated again on a new request. Which means that when we hit the authentication middleware, it saves the current user inside the `TokenGuard` instance to avoid hitting the database again. A wise choice, however—in this case, it means we have to split the logout test into two, to avoid any issues with the previously cached user.

Testing the Article endpoints is straightforward as well:

```
class ArticleTest extends TestCase
{
    public function testsArticlesAreCreatedCorrectly()
    {
        $user = factory(User::class)->create();
        $token = $user->generateToken();
        $headers = ['Authorization' => "Bearer $token"];
        $payload = [
            'title' => 'Lorem',
            'body' => 'Ipsum',
        ];

        $this->json('POST', '/api/articles', $payload, $headers)
            ->assertStatus(200)
            ->assertJson(['id' => 1, 'title' => 'Lorem', 'body' => 'Ipsum']);
    }

    public function testsArticlesAreUpdatedCorrectly()
    {
        $user = factory(User::class)->create();
        $token = $user->generateToken();
        $headers = ['Authorization' => "Bearer $token"];
        $article = factory(Article::class)->create([
            'title' => 'First Article',
            'body' => 'First Body',
        ]);

        $payload = [
            'title' => 'Lorem',
            'body' => 'Ipsum',
        ];

        $response = $this->json('PUT', '/api/articles/' . $article->id, $payload, $headers)
            ->assertStatus(200)
            ->assertJson([
                'id' => 1,
                'title' => 'Lorem',
                'body' => 'Ipsum'
            ]);
    }

    public function testsArtilcesAreDeletedCorrectly()
    {
        $user = factory(User::class)->create();
        $token = $user->generateToken();
        $headers = ['Authorization' => "Bearer $token"];
        $article = factory(Article::class)->create([
            'title' => 'First Article',
            'body' => 'First Body',
        ]);

        $this->json('DELETE', '/api/articles/' . $article->id, [], $headers)
            ->assertStatus(204);
    }

    public function testArticlesAreListedCorrectly()
    {
        factory(Article::class)->create([
            'title' => 'First Article',
            'body' => 'First Body'
        ]);

        factory(Article::class)->create([
            'title' => 'Second Article',
            'body' => 'Second Body'
        ]);

        $user = factory(User::class)->create();
        $token = $user->generateToken();
        $headers = ['Authorization' => "Bearer $token"];

        $response = $this->json('GET', '/api/articles', [], $headers)
            ->assertStatus(200)
            ->assertJson([
                [ 'title' => 'First Article', 'body' => 'First Body' ],
                [ 'title' => 'Second Article', 'body' => 'Second Body' ]
            ])
            ->assertJsonStructure([
                '*' => ['id', 'body', 'title', 'created_at', 'updated_at'],
            ]);
    }

}
```

Next Steps

That's all there is to it. There's definitely room for improvement—you can implement OAuth2 with the Passport package, integrate a pagination and transformation layer (I recommend Fractal), the list goes on—but I wanted to go through the basics of creating and testing an API in Laravel with no external packages.

Laravel development has certainly improved my experience with PHP and the ease of testing with it has solidified my interest in the framework. It's not perfect, but it's flexible enough to let you work around its issues.

If you're designing a public API, check out 5 Golden Rules for Great Web API Design.

UNDERSTANDING THE BASICS

What is Laravel?

Laravel is an opinionated PHP framework. It abstracts away the minutiae of building a web application to facilitate productivity, maintenance, and forward compatibility.

What is REST?

REpresentational State Transfer (REST) and RESTful web services represent a style of network communication between applications to transfer application states through a stateless protocol (such as HTTP).

What is the difference between JSON and XML?

JSON and XML are textual data formats. JSON (JavaScript Object Notation) uses JavaScript syntax to represent data and make it parseable while XML (eXtensible Markup Language) uses markup tagging and nesting to achieve the same thing.

What is Composer?

Composer is a package manager for PHP that manages software dependencies on an application level.

TAGS

André Castelo
Developer

ABOUT THE AUTHOR

André Castelo is a software engineer focusing in front-end development, with years of experience building Javascript applications using React or Vue. He has also worked in several full-stack projects using PHP and Python, leveraging tools such as Laravel and Django to deliver quality products.

Hire André

Comments
Ярослав
Thank you for your good article. I think better way use soft delete instead of usual delete method. For that put to model: class OurModeName extends Model { use SoftDeletes; protected $dates = ['deleted_at']; } This is more secure, and put data to "Trash"
Yaroslav
Thank you for your good article. I think better way use soft delete instead of usual delete method. For that put to model: class OurModeName extends Model { use SoftDeletes; protected $dates = ['deleted_at']; } This is more secure, and put data to "Trash"
Alessandro Fuda
To return "Resource not found" message on 404, in App\Exceptions\Handler class, needs to add "use Illuminate\Database\Eloquent\ModelNotFoundException;" on top.
Alessandro Fuda
to fix API register process, add "use Illuminate\Http\Request;" in RegisterController class
Alessandro Fuda
ehi man, it works fine! I tested with Postman including Bearer Token Authorization
Karen
I done what author wrote in this topic after some chnges my application works perfectly, thanks for intresting topic!!
Karlhans
the logout part I did with email and name. I did this way because with guard didn't work. Someone knows a solution ? $this->attemptLogin($request); $user = $this->guard()->user(); if($user) { $user->api_token = null; $user->save();}
Nicolas Villalba
Can you point a good resource?
Karim Sakhibgareev
Good article. You can also use `Route::resource('articles', 'ArticleController');` instead of manual list of routes for each resource action (https://laravel.com/docs/5.4/controllers#resource-controllers).
Goran Trlin
Very nice article. Thank you!
Jocelio Lima
Ótimo artigo, parabéns.
Ufere Peace
Great article. Thank you
Pedro Werneck
Unfortunately this article just contributes to the widespread confusion and misconceptions about REST. This might be useful for someone implementing quick and dirty HTTP APIs, but there's nothing RESTful about it.
Henrique Schreiner
In case you have any doubts (which clearly have) what RESTful is about: https://en.wikipedia.org/wiki/Representational_state_transfer
Keyner TYC
Where is function generateToken() ???, fix please.
igorsantos07
Certainly the biggest misconception here is about what comments are for. They're open to amplify the discussion, not to throw meaningless rants into the author. You're not making a point here, just offending. If you see issues with the way the article implements RESTful, why don't you point out what's so ~dirty~ that you've seen?
André Castelo
Thanks for pointing that out, I'll make the changes. The function is inside the User model - you can check the source code for it: https://github.com/andrecastelo/example-api/blob/master/app/User.php
Henrique Schreiner
Totally agree!
Pedro Werneck
If you can't distinguish between criticism about a piece of writing and criticism about the author, you shouldn't be lecturing others on what comments sections are for. <blockquote>If you see issues with the way the article implements RESTful, why don't you point out what's so ~dirty~ that you've seen?</blockquote> I don't have to. Roy Fielding already did. http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
Pedro Werneck
You're patronizing people on the internet with Wikipedia? Seriously? https://stackoverflow.com/a/19884975/1202421
Usama Alshihabi
Thank you
igorsantos07
Good to see you found some solid argument :) Bad to see it's just a link, and you don't really want to open up the discussion. I've heard that point before, but it feels too strong saying this type of REST is for "quick and dirty HTTP APIs". Take a look on the Maturity Model for REST APIs by Leonard Richardson and Martin Fowler[1], you'll see you're just complaining about them not being the upmost level of perfection. You don't need to be hypertext driven for most of the usages we see nowadays, like communicating between custom-made systems, transfering data between your system and the company's mobile app, and so forth. Remember: this tutorial is not supposed to be a hard, advanced explanation on RESTful APIs. It does show how to create a nice API through HTTP that can be expanded and used for many purposes. And does it very well. If you are looking for advanced REST techniques, look for advanced tutorials at least; or a real course, or take a look at your local university/academic youtube channel. Hypertext on APIs is certainly not the topic to be covered on a small tutorial, on my point of view. [1]: https://martinfowler.com/articles/richardsonMaturityModel.html
igorsantos07
Well, at least that's a source. Better than not giving any at all, or giving yourself as one. At the very least, both are equaly verifiable fonts. I really can't understand why people have such bad feelings against common subjects on Wikipedia, like if they were not peer-verified at all.
Pedro Werneck
I'm perfectly aware of Richardson's Maturity Model, and if you read the article you linked you'll see it is about providing a path for non-RESTful HTTP APIs to evolve into actual RESTful APIs. RMM's top level is the <strong>minimum necessary</strong> for an API to be RESTful, so your comment is absurd, unless you think asking for the minimum necessary is the same as asking for the utmost level of perfection. <blockquote>You don't need to be hypertext driven for most of the usages we see nowadays, like communicating between custom-made systems, transfering data between your system and the company's mobile app, and so forth.</blockquote> Sure, but if your API is not hypertext driven, it's not REST. What you're really saying is that you don't need REST for most of the cases we see nowadays, and you're right. We don't. We see people doing something they think it's REST and calling it REST, when in reality it has nothing to do with REST, and they wouldn't even need REST in the first place, like the article. It's design-by-buzzword, as Fielding himself prophetically pointed out in his dissertation. In fact, most of the so-called REST APIs I've seen would be much easier to implement and maintain as RPC services. They're so tightly coupled that they are practically RPC already, but someone insists on calling it REST, for some mysterious reason. <blockquote>Remember: this tutorial is not supposed to be a hard, advanced explanation on RESTful APIs. </blockquote> Remember: I said this tutorial might be useful for someone implementing quick and dirty HTTP APIs. It might be a great tutorial, but not only there's nothing RESTful about the examples given, there are several anti-patterns being suggested as RESTful: the naive mapping of HTTP methods to CRUD operations, URI semantics, generic media-types, etc. <blockquote>It does show how to create a nice API through HTTP that can be expanded and used for many purposes. And does it very well.</blockquote> A nice HTTP API, but not a RESTful API. <blockquote>If you are looking for advanced REST techniques, look for advanced tutorials at least; or a real course, or take a look at your local university/academic youtube channel</blockquote> The patronizing tone of your comment makes it even more absurd. Are you saying the minimum necessary is an "advanced technique"? <blockquote>Hypertext on APIs is certainly not the topic to be covered on a small tutorial, on my point of view.</blockquote> Yes, it is, if you're claiming your tutorial is about building and testing a RESTful API. If your API is not hypertext driven, it's not RESTful. Saying it is doesn't make it so.
Pedro Werneck
http://nordicapis.com/wp-content/uploads/API-Design-on-the-scale-of-Decades.pdf
Pablo George
Excelente artigo, muito top, obrigado André.
Matias Azar
Great Article! Thanks... just a question.. How can i change the Guard used on login (for example) becuase my app still checking "web" guard. thanks
Arya Maharta
Great!
Esraa Saber
Great Article. Thanks
Tomasz Felczyk
When it comes to Laravel and REST it is worth mentioning the Resource Controller. Also writing $article->update($request->all()); is little unsecure. Also nice tip will be use of FormRequest instead of Request or simple validation.
angamba meetei
how to do both in web as well as in mobile, dealing with rest api, actually i want to view the data in blade as well as via api in json, would you tell me the procedure, i will be very thankful. thanks
Chris
Hi Andre, nice article, however in the error handling section, I cannot get it to return the JSON error response, it keeps returning the default Laravel error page. After doing some digging, I added the following to the top of the Handler.php: use Illuminate\Database\Eloquent\ModelNotFoundException as

ModelNotFoundException; Then the exceptions were handled correctly. Not sure if this is the correct thing to do?
oliver smith
http://www.office-setup-install.us
nnduy
I got the same error. I have no fix this error.
Farooq Ahmad
When I run this command "php artisan db:seed --class=ArticlesTableSeeder", I got an error that "[Symfony\Component\Debug\Exception\FatalThrowableErro] Class 'Article' not found" please help me to fix it.
Tucker
I think you might need to do `composer dump-autoload`
Farooq Ahmad
Yeh that's working :)
Dhanushka Jayasekara
When I run "php artisan db:seed --class=ArticlesTableSeeder", I got an error "[Symfony\Component\Debug\Exception\FatalThrowableErro] Class 'Article' not found". And I run "composer dump-autoload" as below said. But still i get this error. Please help.
Panhaseth Heang
I think you forgot to include "use App/Article;" inside ArticlesTableSeeder.
Dhanushka Jayasekara
yep.. thanks :)
Jamie Tyree
Question, coming from an MVC point of view, is it odd that the controller here is essentially a "model." Or would we not consider the code calling "::create()" the model because it instead is simply calling the Eloquent model?
Adam Krell
The generateToken() method is on the User model: https://github.com/andrecastelo/example-api/blob/master/app/User.php
Cris John Rey Tarpin
thank you so much for the reply :)
Akila Thiwanka
Add this "use Illuminate\Database\Eloquent\ModelNotFoundException; " to the top of Handler.php
Cris John Rey Tarpin
Hi, good tutorial, but i was stucked as $user->generateToken() method. it says error, i search the API reference of Authenticatable but i did not found the method. can please help me out? thanks in advance
gvk
php artisan db:seed --class=ArticlesTableSeeder Got error like this :: [Illuminate\Database\QueryException] SQLSTATE[42S02]: Base table or view not found: 1146 Table 'myapp.articles' doesn't exist (SQL: truncate `articles`) [PDOException] SQLSTATE[42S02]: Base table or view not found: 1146 Table 'myapp.articles' doesn't exist
Charles
Its probably a problem emanating from your migration file. Please post the contents of your migration file so that so that we may assist you. or rather check if the migration file is similar to the one provided with this project.
Jarl Gjessing
Looks nice, but I get class ArticlesTableSeeder does not exist when trying to run: php artisan db:seed --class=ArticlesTableSeeder I tried several times by copy/paste etc to ensure there were no spelling errors. I also verified that the file is actually the right place with the right name.. The above commands worked just fine
Kevin Andries
Try running "composer dump-autoload" in your command line before running the seed command.
Jason Rego
Thanks! this article is extremely helpful.
Trần Quốc Việt
Awesome documents. Thanks pro
Noitidart
This is very cool. My consuming app is in javascript, and in javascript we set the id as key in objects. Is it possible make the id's be returned as strings in json? Also when I do the first php artisan migrate I get this error: Migration table created successfully. [Illuminate\Database\QueryException] SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length i s 767 bytes (SQL: alter table `users` add unique `users_email_unique`(`email`)) [PDOException] SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length i s 767 bytes The fix recommended here sounds like not a fix - https://stackoverflow.com/a/42245921/1828637 May you please advise?
Noitidart
I was also stuck on this, thank you Criss and Adam for this. Why did he skip this in the tutorial? I also had to do a php artisan migrate before this worked, this added the api_token column.
Rogerio Pereira Araujo
Very good tutorial, congrats man!
Noitidart
Do you have anywhere that teaches us how to setup the forgot password stuff? Route::post('password/email', 'Auth\ForgotPasswordController@sendResetLinkEmail')->name('password.email'); Route::post('password/reset', 'Auth\ResetPasswordController@reset');
Muhammad Syarif
I try get user : http://localhost:8000/api/user error : { "error": "Unauthenticated." }
Artur Poniedzialek
Under which version of php I can run Laravel?
Ahmed Sliman
Please change this on the tut. this will be good for the visitors :)
Ayman
Great article, thank you! Just a small comment: for the "not found" error handling to work, you need to add the following line to the handler.php use Illuminate\Database\Eloquent\ModelNotFoundException;
Daizy Shah
Great article for api but what if I want to login with api_token without user email id and password. please let me know.
Nimesh Parekh
it seems logout feature is not working correctly. Navigate to app/Http/Controllers/Auth/LoginController and go to logout() method. in this method there is first line is stated as below Auth::guard('api')->user() above line does not give logged in user data. thats why it doesnt clear the api_token. Could you please let me know how we should fix this ? Everything else is awesome !!! Looking forward for immediate reply.
Ibrahim Samad
Very helpful thanks.
Winnie A Damayo
you're fucking awesome. Thank you.
Raikumar Khangembam
Very nice article n very helpful. Posting article did not work got some error then i modified and worked In Route/api.php Route::post('articles', function(Request $request) { $data = $request->all(); return Article::create(['title' => $data['title'], 'body' => $data['body'],]); // return Article::create($request->all); }); And in ArticleController function public function store(Request $request) { $article = Article::save(); return response()->json($article, 201); }
ufu media
here is error when i entry any url . plz help me whats wrong
https://uploads.disquscdn.com/images/43ab96d6483746f830bbbfc8aaa4d053735ecbfd767e84de17e9fd50d1931b05.png
ufu media
when i write http://localhost/API/public/api/register then error disply MethodNotAllowedHttpException in RouteCollection.php (line 251) :(any one can help me whats wrong with me
Allen Ford
This is very nice... however i got stuck on the last steps... seems when i try an migrate the tests i get the following error: [Illuminate\Database\QueryException] SQLSTATE[42000]: Syntax error or access violation: 1103 Incorrect table name '' (SQL: create table `` (`id` int unsigned not null auto_increment primary key, `migration` varchar(255) not null, `batch` int not null) default character set utf8mb4 collate utf8mb4_unicode_ci) --- Seems like the table is returning NULL or ''.. how do i fix this?
Allen Ford
You know if would same me and probably alot other NUBS if the use * was or full pages was saved in this tuturial.. also files names for every block of code.. this kind of got me in on a roller coaster
ufu media
how i can run any one can tell me plz curl -X POST http://localhost:8000/api/register \ -H "Accept: application/json" \ -H "Content-Type: application/json" \ -d '{"name": "John", "email": "john.doe@toptal.com", "password": "toptal123", "password_confirmation": "toptal123"}'
Dave Chambers

Hi Andre, Brilliant tutorial.
Amin Shah Gilani
You're going to have to paste this into your command line. `curl` is a command line utility: https://curl.haxx.se/
Said Bakr
Unfortunately, I expected to see how to implement the picture of SERVER talking with token, specially for listing articles. However, it is not bad as an introduction.
Chung Dev
Thank
Thomas Clifford
The first part of the tutorial is giving me some grief. I downloaded Composer (v1.5.2), and Laravel (Installer v1.4.1), then created the project, calling it 'restapi01'. I upgraded my PHP to v7.1.11. I then ran it using > php artisan serve and received: Warning: require(C:\dl\coding\laravel\0projects\restapi\toptal\restapi01/vendor/autoload.php): failed to open stream: No such file or directory in C:\dl\coding\laravel\0projects\restapi\toptal\restapi01\artisan on line 18 Fatal error: require(): Failed opening required 'C:\dl\coding\laravel\0projects\restapi\toptal\restapi01/vendor/autoload.php' (inc lude_path='.;C:\php\pear') in C:\dl\coding\laravel\0projects\restapi\toptal\restapi01\artisan on line 18 My PHP is installed in "C:\bin\php". There is no 'c:\bin\php\pear' directory. Thanks.....
Anton Antonov
Same problem. Have you found a solution?
Ali Yisa
You should send a POST request instead of GET. Register is meant for only POST. I suggest you use postman for more flexibility.
Ali Yisa
I'm going to guess you're trying to use a browser, and it sends a GET request instead of POST. I suggest to use postman for more flexibility.
Eduardo França
Great article! I'm not sure if anyone has pointed that out already, but you forgot to mention we have to include the Model in the seeder. (like including "use App\Article;" in ArticlesTableSeeder.php)
Dien
hi nice to meet all
Dien
Im beginer work with laravel... Any one help me to share about Laravel API..thank alot
Amit Singh
Can I use this for multi Auth..?
Sone Inthavong
Hi, how would you create this and retain the normal login and register behavior of Laravel? I would like an API + normal Laravel application.
Stacy Thompson
Hi , where do I find api.php file in laravel 5.0, There is no folder named routes
John Shofstall
I'm getting the following error upon posting to the register url, however, the data is being saved to the database. { "message": "Type error: Argument 1 passed to App\\Http\\Controllers\\Auth\\RegisterController::registered() must be an instance of App\\Http\\Controllers\\Auth\\Request, instance of Illuminate\\Http\\Request given, called in /Users/john/Documents/development/toptut/vendor/laravel/framework/src/Illuminate/Foundation/Auth/RegistersUsers.php on line 37", "exception": "Symfony\\Component\\Debug\\Exception\\FatalThrowableError", "file": "/Users/john/Documents/development/toptut/app/Http/Controllers/Auth/RegisterController.php", "line": 72, "trace": [{ "file": "/Users/john/Documents/development/toptut/vendor/laravel/framework/src/Illuminate/Foundation/Auth/RegistersUsers.php", "line": 37, "function": "registered", "class": "App\\Http\\Controllers\\Auth\\RegisterController", "type": "->" }, 300 more lines of trace properties,] } UPDATE: before I hit post I did one more search, and found the answer! You need to have 'use Illuminate\Http\Request;' in the RegisterController.
Saket Mayank
$ php artisan db:seed Seeding: ArticlesTableSeeder [Illuminate\Database\QueryException] SQLSTATE[42S02]: Base table or view not found: 1146 Table 'practice.articles' doesn't exist (SQL: truncate `articles`) [Doctrine\DBAL\Driver\PDOException] SQLSTATE[42S02]: Base table or view not found: 1146 Table 'practice.articles' doesn't exist I am unable to understand why i am getting this error ,please help me
Aron Lilland
thank you for the tutorial, I come from the ruby world and have been a long time enthusiast, and love making API's -- Im unfortunately not very familiar with PHP, i'm dipping my toes in it with your tutorial, unfortunately "php artisan db:seed --class=ArticlesTableSeeder" returns an error message "In ArticlesTableSeeder.php line 10: Class 'Article' not found"
Oscar D. Palencia
Great Work. I loved it!
Ивин Сергей
A very good overall article covering all you need to build RESTful API. Thank you
Luka Sikic
use "NotFoundHttpException" instead of "ModelNotFoundException" and add "use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;" Hope this will help someone! :)
notalentgeek
Most parts of this tutorial does not work. This is sad, because the author seems to put efforts to make the tutorial. * There are a lot of imports need to be done manually, but this tutorial does not tell. * Additionally, I am stuck because I cannot login through `curl`. I received this error: `"Type error: Argument 1 passed to App\\Http\\Controllers\\Auth\\RegisterController::registered() must be an instance of App\\Http\\Controllers\\Auth\\Request, instance of Illuminate\\Http\\Request given, called in /home/notalentgeek/notalentgeek/Projects/testing-restful-api/vendor/laravel/framework/src/Illuminate/Foundation/Auth/RegistersUsers.php on line 50"` * I have looked at SO but there is no solution.
Lambert Lum
Article is shorthand for \App\Article. Add the following to the top of your page, so Article will be recognized. use App\Article; You might want to read up on PHP namespaces.
MichaelOrokola
Great article.
sh.py
Great article,, thanks for sharing. This very helpful
Leandro
Vlw pelo Artigo André, sigo-o lendo como referência.
guy ambar
Very nice article. But I would suggest introducing the database factories earlier, so instead of looping you can send the amount of items you want as a second argument. For example. factory(Article::class, 50)->create();
C P
Hey! Thanks for these very good Article... I am getting error when I am trying to run register or login url.. Symfony \ Component \ HttpKernel \ Exception \ MethodNotAllowedHttpException I followed document I got this error. Can you please tell me why this error?
Donald kagunila
here is the solution @nota@mikaelpratamakristyawicakson:disqus in the LoginController and RegisterController add: use Illuminate\Http\Request; in the LoginContoller add: use Illuminate\Foundation\Auth\AuthenticatesUsers; in the ArticleController add: use App\Article; in the ArticleSeeder add : use App\Article;
Donald kagunila
run $ composer install
C P
Hi André Castelo from Germany, Thanks for this great tutorials.. Above issue solved.. I need to use Postman to test the API and proper parameter.. And things are set.. Thanks again.... :)
RealParanoidAndroid
Thank you. I would never have found that...
Vignesh Vaidyanathan
Thank you. this solved my issue.
gabrieldesousah
obs: In handler.php is necessary to show 404 error: use Illuminate\Database\Eloquent\ModelNotFoundException;
Nancy Rai
Awesome! Great article and easy to understand. It helped! I recently found another very informative tutorial about SSIS REST API. Check out the below link : https://zappysys.com/products/ssis-powerpack/ssis-web-api-integration-pack/ Hope it help others..
Paweł Dziurzyński
use Illuminate\Support\Facades\Auth; in App\Http\Controllers\Auth\LoginController.php solves the issue.
██████ █████
Great article... i loved reading this and found everything smoothly work on my PC.
firoj khan
I am confusing that what we have to write in api.php ,web.php and ArticleController because you have changed route api.php and ArticleController that

confusing me. How can I do because I am new to api.
firoj khan
bro can you help me for getting this tutorial because its difficult to me understand what I have to write in api.php and ArticleController and also in web.php it very confusing.
firoj khan
getting error $ curl -X POST http://localhost:8000/api/register -H "Accept: application/json" ssword_confirmation":"firoj123"}' 0curl: (7) Failed to connect to localhost port 8000: Connection refused how to solve this error.
Noitidart
Bump please. @AndreCastelo may you please show us how to to password reset, password change, email change stuff. In last 6 months I cannot find anyoone that does a REST api tutorial like you. So simple for new comers.
Bien Hoang
I have error with CURL $ curl -X POST http://localhost:8000/api/register \ -H "Accept: application/json" \ -H "Content-Type: application/json" \ -d '{"name": "Administrator", "email": "bhoang@inofthing.com", "password": "123ABC", "password_confirmation": "123ABC"}' These error: --_curl_--"name": "Administrator" --_curl_-- "email": "bhoang@inofthing.com" --_curl_-- "password": "bienhv" --_curl_-- "password_confirmation": "bienhv" curl: (6) Couldn't resolve host ' -H' curl: (3) Illegal port number curl: (6) Couldn't resolve host ' -H' curl: (3) Illegal port number curl: (6) Couldn't resolve host ' -d' [1/4]: "name": "Administrator" --> <stdout> curl: (3) Illegal port number How to fix it? Thank you so much,
Said Bakr
What about authorization in the API?
Stefano Caponi
could you write a guide on how to use phpunit with laravel passport?
ARTURO ATENCIO (Ax3)
it didnt work for me :/ any other idea? I mean, stills give the same result: api_token in database not deleted, bc Auth::guard('api') gives null. So I went to laravel´s docs and it says to edit/modify/alter the auth.php inside config folder. But when I do, it gives an error saying "Auth guard driver [api] is not defined". So if I can not edit the auth.php, and *Auth::guard('api')* stills give null, it will never clear the api_token field in database. (of course, I already tried using the auth facade).
ARTURO ATENCIO (Ax3)
hi, the logout part didnt work for me :/ It gives this result: api_token in database not deleted, bc Auth::guard('api') gives null. So I went to laravel´s docs and it says to edit/modify/alter the auth.php inside config folder. But when I do, it gives an error saying "Auth guard driver [api] is not defined". So if I can not edit the auth.php, and *Auth::guard('api')* stills give null, it will never clear the api_token field in database. Any help? suggestions? hints? .
Ramesh Navi
I need to use username instead of email, I am getting `{"email":["The email field is required."]}`, any workaround ?
Supriyadin Yth
great article. I love it. Thank You
Arthur Heckmann
I had the same issue but it turned out, i was missing the Authorization header in my logout post. only had to add the Authorization header /bearer token with the api_key .
Erick Moises Racancoj Amperez
Thank you Adam Krell. I was stuck on RegisterController because I was missing a using to Illuminate\Http\Request. Thank you so much.
martins
However old this blogpost it keeps helping me everytime. You're awesome
Karlhans
ok works all fine, but few little changes in the test, more exactly the response codes
Gareth G
i'm having issues with the tests specifically the articles. Because the Articles table is already populated when i run the tests it bugs out : The first two tests because there is already a item in the first row (id:1) - obviously i can change this ID in the test to be 51 and then it passes, but the last test - testArticlesAreListedCorrectly() - fails because it's trying to check that row 1 and 2 have the new articles. this is my error message: Failed asserting that an array has the subset Array &0 (0 => Array &1 ('title' => 'First Article' 'body' => 'First Body') 1 => Array &2 ('title' => 'Second Article' 'body' => 'Second Body')). --- Expected +++ Actual @@ @@ [0] => Array ([id] => 1 - [title] => First Article - [body] => First Body + [title] => Et fugiat sapiente beatae quod. + [body] => Odit quo nihil voluptas similique corrupti natus quaerat. Amet dolores sed ex praesentium architecto tenetur. Sunt unde voluptatum vero pariatur qui. Est error sit itaque dolorum neque voluptatem. [created_at] => 2018-08-24 08:16:20 [updated_at] => 2018-08-24 08:16:20) @@ @@ [1] => Array ([id] => 2 - [title] => Second Article - [body] => Second Body + [title] => Nostrum itaque et aspernatur est adipisci consequatur sunt. + [body] => Quae at delectus placeat voluptate accusantium. Accusamus recusandae aut facere. Ipsa omnis nihil ducimus. [created_at] => 2018-08-24 08:16:20
Gareth G
so the problem with this was that in the TestCase.php I was running - Artisan::call('db:seed') when i removed this all tests passed. My opinion is that seeding your db when creating tests is wrong - but i didn't spot this initially
Abdul Haadi
Good!This post is creative,you'll find a lot of new idea,it gives me inspiration. I believe I will also inspired by you and feel about extra new ideas.thanks. (https://www.vedigitize.com)
Juan D Batun
Ausgezeichnete Arbeit, danke, ich helfe, einige Zweifel über die Laravel-API zu lösen. Ich hoffe, Sie haben ein Repository mit einem Beispiel von Oauth2
CNN2
Hey Andre, I have a question regarding API security and need your help I see the authentication with api_token is for authenticated items like logout, details of own profile to edit, dashboard, purchase items, order history..... However, I have lot of information/content which is needed for non-logged in users like static pages, contacts us page, products listing, product detail page and many more pages like these. Is this important to cover APIs for these items with extra security so that no one can steal data from other source or can not use in his frontend app. There are two other authentication Passport and JWT. however I am not aware weather these provide such security facility.
khanzada
I learned alot from this Article...
asifinet1
I was getting error when the article instructed me to do create the UserTableSeeder in the database directory and run the following command, php artisan db:seed --class=UsersTableSeeder I got an error ReflectionException : Class UsersTableSeeder does not exist. To resolve this, I google it to find the solution and found that, I need to run the composer dump-autoload command at cmd prompt. then I run the same command as above to rectify it.
asifinet1
You can use PostMan and Post Request, I tried to use the CURL as well but endup seeing unsual.
asifinet1
Hi Saket, I don't know, If you would have found the solution. but this when you run the command use "up" at the end to create the table. Since you are truncating table, please check, If you are connected to the right environment. in the ".env" file at the root of laravel app. I hope It will help.
asifinet1
Check your AppService provider inside the folder app\Providers\AppServiceProvider.php and put the following line Schema:: public function boot() { Schema::defaultStringLength(191); } and some more in order to create article successfully. you need to do put the following code to ensure article and userTableSeeder runs without any error, Please drop all the table from your database and try running migrate the table again to see it get corrected. Schema::create('articles', function (Blueprint $table) { $table->increments('id'); $table->string('title',1000); $table->string('body',1000); $table->timestamps(); });
asifinet1
Hi All, I am getting the following { "message": "Class 'App\\Http\\Controllers\\Auth\\Registered' not found", "exception": "Symfony\\Component\\Debug\\Exception\\FatalThrowableError", "file": "I:\\laravel\\myapp\\app\\Http\\Controllers\\Auth\\RegisterController.php", "line": 90, "trace": [{ "function": "register", "class": "App\\Http\\Controllers\\Auth\\RegisterController", "type": "->" }, { "file": "I:\\laravel\\myapp\\vendor\\laravel\\framework\\src\\Illuminate\\Routing\\Controller.php", "line": 54, "function": "call_user_func_array" }, { "file": "I:\\laravel\\myapp\\vendor\\laravel\\framework\\src\\Illuminate\\Routing\\ControllerDispatcher.php", "line": 45, "function": "callAction", "class": "Illuminate\\Routing\\Controller", "type": "->" },
asifinet1
I resolved my own error, by searching on internet of above problem implementing the use Illuminate\Auth\Events\Registered at the "I:\\laravel\\myapp\\app\\Http\\Controllers\\Auth\\RegisterController.php",
Gregorio Boada
Hello André Castelo, before everything very well contribution. I have a problem and I have my Api created and I need to consume it from several different domains, example dominio.com and dominio1.com and I get the following error: Access to XMLHttpRequest at 'http: //api.grebo.test: 9000 / oauth / token' from origin 'http: //grebo.test: 90' has been blocked by CORS policy: No 'Access-Control-Allow-Origin 'header is present on the requested resource. I hope you can help me. P.S: Excuse my spelling, is that I do not speak English :)
Renzchler Oxiño
Amazing! <3
Cymburyo Ox
Wow!
Sorin Secan
If you pass the API token gathered from the login function will logout the user using the original code.
pgwebdev

Hello, I am getting this error when user tries to login : "message": "Session store not set on request.", "exception": "RuntimeException", "file": "/code/rest-api/vendor/laravel/framework/src/Illuminate/Http/Request.php", "line": 467 Please advice if anyone knows what's causing this issue and how this can be fixed?
Thao Le
I guess it's because you forgot to add the -H "Authorization: Bearer usertoken" in the curl. Logout actually works.
Anonymouse703
Nice tutorial sir... Hope you make a tutorial regarding Laravel Notification of github webhook.
pgwebdev
Hello C P, how did you resolve above issue. I am getting same exception when try to register.
Ipc Bansur
Access to XMLHttpRequest at 'http://localhost/myapp/oauth/token' from origin 'http://localhost:4200' has been blocked by CORS policy: Request header field key is not allowed by Access-Control-Allow-Headers in preflight response. how can i solve this issue using laravel passport
Moussi Foued
I solved the problem relative to Registered class by use Illuminate\Auth\Events\Registered;
Naveen Roy
Thanks sir for create such a easy tutorial for Rest API for Laravel I used it 2 times.
Rostislav Emanrus
How to implement sorting|search for that API ?? Thanks
FlyOnTheWall
Unfortunately Andre doesn't paste all necessary info, so some things will not work as described, and you have to figure out what you're missing. For instance, I got this error trying run `php artisan db:seed --class=ArticlesTableSeeder`: `Symfony\Component\Debug\Exception\FatalThrowableError : Class 'Article' not found` If you trace the error, ArticlesTableSeeder.php is trying to run `Article::truncate()`, but it can't find the 'Article' class, because it's not automatically loaded for whatever reason. If you go to database/seeds/ArticlesTableSeeder.php and add `use App\Article;` above the class name, it works.
Arafat Hossain
https://uploads.disquscdn.com/images/20d31a9d3926544013f986e8e113d8af72019dc658cf43f64e601ca0b65960ba.jpg
Amin Shah Gilani
Why? :/
Guto Rocha
Add 'use Illuminate\Support\Facades\Auth;' to the LoginController. Add Route::get('logout', 'Auth\LoginController@logout'); in the config/api.php into the group where the articles are and add the token to the header.
asifinet1
https://goo.gl/fzeXFi
asifinet1
https://goo.gl/YjqEmM
asifinet1
https://goo.gl/YgFZnv
asifinet1
https://goo.gl/JNGDww
asifinet1
https://goo.gl/y93x72
Enea Paja
I would like to create a new API request (Like getting all the Used (id)) after the authentication.
asifinet1
https://goo.gl/nPQh4G
asifinet1
https://goo.gl/qSkPHT
asifinet1
https://goo.gl/qrWDM3
asifinet1
https://goo.gl/WKt75T
asifinet1
https://goo.gl/TriRuc
asifinet1
https://goo.gl/dec1zo
asifinet1
https://goo.gl/9YBbBp
asifinet1
https://goo.gl/oP9Kpw
asifinet1
https://goo.gl/nvcjfi
asifinet1
https://goo.gl/gbLSgo
asifinet1
https://goo.gl/sUWHFG
asifinet1
https://goo.gl/fRQMjh
asifinet1
https://goo.gl/AGPRzT
asifinet1
https://goo.gl/TEbK7k
asifinet1
https://goo.gl/RFLXS7
asifinet1
https://goo.gl/vHdrE8
asifinet1
https://goo.gl/dBdwZ9
asifinet1
https://goo.gl/tRGsR9
asifinet1
https://goo.gl/bUx7hX
asifinet1
https://goo.gl/Nf3CKQ
asifinet1
https://goo.gl/HFPMMD
asifinet1
https://goo.gl/BkMWHT
asifinet1
https://goo.gl/1rBtQs
asifinet1
https://goo.gl/edX8e5
asifinet1
https://goo.gl/brjScz
asifinet1
https://goo.gl/PybsDz
asifinet1
https://goo.gl/xswUow
asifinet1
https://goo.gl/uAYqWF
asifinet1
https://goo.gl/nvJJaV
asifinet1
https://goo.gl/NnA17i
asifinet1
https://goo.gl/Q4pQCi

asifinet1
https://goo.gl/PZcCez
asifinet1
https://goo.gl/sbMmdq
asifinet1
https://goo.gl/9ya9NZ
asifinet1
https://goo.gl/BuPZSH
asifinet1
https://goo.gl/JNC9Vr
asifinet1
https://goo.gl/AVZoic
asifinet1
https://goo.gl/yc3dhX
asifinet1
https://goo.gl/vueMEp
asifinet1
https://goo.gl/v63Gxw
asifinet1
https://goo.gl/AqLFVP
asifinet1
https://goo.gl/SsMr5k
asifinet1
https://goo.gl/LURA2R
asifinet1
https://goo.gl/XtqATX
asifinet1
https://goo.gl/pkY5Wm
asifinet1
https://goo.gl/2Bm4HX
asifinet1
https://goo.gl/6hEjDi
asifinet1
https://goo.gl/icgWFZ
asifinet1
https://goo.gl/L6VWY3
asifinet1
https://goo.gl/CSbAKX
asifinet1
https://goo.gl/QeER4E
asifinet1
https://goo.gl/DJWvUJ
Kevin Bloch
Could you be more specific?
MuslimsWithModi
My request is asking for CSRF token.
comments powered by Disqus

TRENDING ARTICLES

ENGINEERING › WEB FRONT-END

WebVR Part 4: Canvas Data Visualizations

ENGINEERING › WEB FRONT-END

WebVR Part 3: Unlocking the Potential of WebAssembly and AssemblyScript

ENGINEERING › TECHNOLOGY

Haxe Review: Haxe 4 Features and Strengths

ENGINEERING › WEB FRONT-END

Optimizing Website Performance and Critical Rendering Path

SEE OUR RELATED TALENTS

Toptal Developers

Android Developers

AngularJS Developers

Back-End Developers

C++ Developers

Data Analysts

Data Engineers

Data Scientists

DevOps Engineers

Ember.js Developers

Freelance Developers

Front-End Developers

Full-Stack Developers

HTML5 Developers

iOS Developers

Java Developers

Machine Learning Engineers

Magento Developers

Mixed Reality Developers

Mobile App Developers

.NET Developers

Node.js Developers

PHP Developers

Python Developers

React.js Developers

Ruby Developers

Ruby on Rails Developers

Salesforce Developers

Scala Developers

Software Architects

Software Developers

Unity or Unity3D Developers

Virtual Reality Developers

Web Developers

WordPress Developers

View more

Join the Toptal® community.

Hire a Developer OR Apply as a Developer

MOST IN-DEMAND TALENT

iOS Developers
Front-End Developers
UX Designers
UI Designers
Financial Modeling Consultants
Interim CFOs
Digital Project Managers

ABOUT

Top 3%
Clients
Freelance Developers
Freelance Designers
Freelance Finance Experts
Freelance Project Managers
Freelance Product Managers
Specialized Services
About Us

CONTACT

Contact Us
Press Center
Careers
FAQ

Hire the top 3% of freelance talent™