LUMEN

Developing RESTful APIs with Lumen (A PHP Micro-framework)

Lumen is a PHP micro-framework built to deliver micro-services and blazing fast APIs. Learn how to build and secure RESTful APIs with Lumen

Prosper Otemuyiwa

December 26, 2017

TL;DR: In this tutorial, I'll show you how easy it is to build and secure an API with Lumen. Check out the repo to get the code.

Lumen is an open-source PHP based micro-framework created by Taylor Otwell in 2015. Lumen is designed for building lightning fast micro-services and APIs. And it opts for maximum speed rather than flexibility in the bootstrapping process. The PHP micro-framework was born out of the need to have light Laravel installations that could be faster than existing PHP micro-frameworks such as Slim and Silex.

> "Lumen is designed for building lightning fast micro-services and APIs."

Tweet This

Lumen Features And Architecture

Lumen utilizes the Illuminate components that power the Laravel framework. One amazing thing about the way Lumen was built is the fact that you can painlessly upgrade right into Laravel. One of such scenarios where an upgrade process is

applicable is when you discover that you need more features out of the box that Lumen doesn't offer.

> "One amazing thing about the way Lumen was built is the fact that you can painlessly upgrade right into Laravel."

Tweet This

- **Routing**: Lumen provides routing out of the box via Fast Route. *Fast Route* is a library that provides a fast implementation of a regular expression based router.
- **Authentication**: Lumen does not support session state. However, incoming requests are authenticated via a stateless mechanism such as tokens.
- **Caching**: Lumen supports caching just like Laravel. In fact, there are no differences between using the cache in Lumen and Laravel. Cache drivers such as *Database*, *Memcached*, and *Redis* are supported. You will need to install the `illuminate/redis` package via Composer before using a Redis cache with Lumen.
- **Errors and Logging**: Lumen ships with the Monolog library, which provides support for various log handlers.
- **Queuing**: Lumen provides a queuing service that is similar to Laravel's. It provides a unified API across a variety of different queue back-ends.
- **Events**: Lumen's events provide a simple observer implementation, allowing you to subscribe and listen for events in your application.

The entire bootstrap process is located in a single file.

Lumen Key Requirements

In order to use Lumen, you need to have the following tools installed on your machine.

- **PHP**: Make sure PHP is installed on your machine. `PHP >= 7.0`. Furthermore, ensure that the following PHP extensions are installed. `OpenSSL`, `PDO` and `Mbstring`.
- **Composer**: Navigate to the composer website and install it on your machine. Composer is needed to install Lumen's dependencies.
- You'll also need familiarity with database concepts, and working knowledge of PHP.

> **Note: You'll need MySQL for this tutorial. Navigate to the mysql website and install the community server edition. If you are using a Mac, I'll recommend following these instructions. To avoid micromanaging from the terminal, I'll also recommend installing a MySQL GUI, Sequel Pro.**

Building a Fast Authors API Rapidly With Lumen

At Auth0, we have a number of technical writers, otherwise known as authors. A directive has been given to developing an app to manage Auth0 authors. The frontend app will be built with ReactJS. However, it needs to pull data from a source and also push to it. Yes, we need an API!

This is what we need the API to do:

- Get all authors.
- Get one author.
- Add a new author.
- Edit an author.
- Delete an author.

Let's flesh out the possible endpoints for this API. Given some *authors* resource, we'll have the following endpoints:

- Get all authors - `GET /api/authors`
- Get one author - `GET /api/authors/23`
- Create an author - `POST /api/authors`
- Edit an author - `PUT /api/authors/23`
- Delete an author - `DELETE /api/authors/23`

What will be the author attributes? Let's flesh it out like we did the endpoints.

- Author: `name`, `email`, `twitter`, `github`, `location`, and `latest_article_published`.

Install Lumen

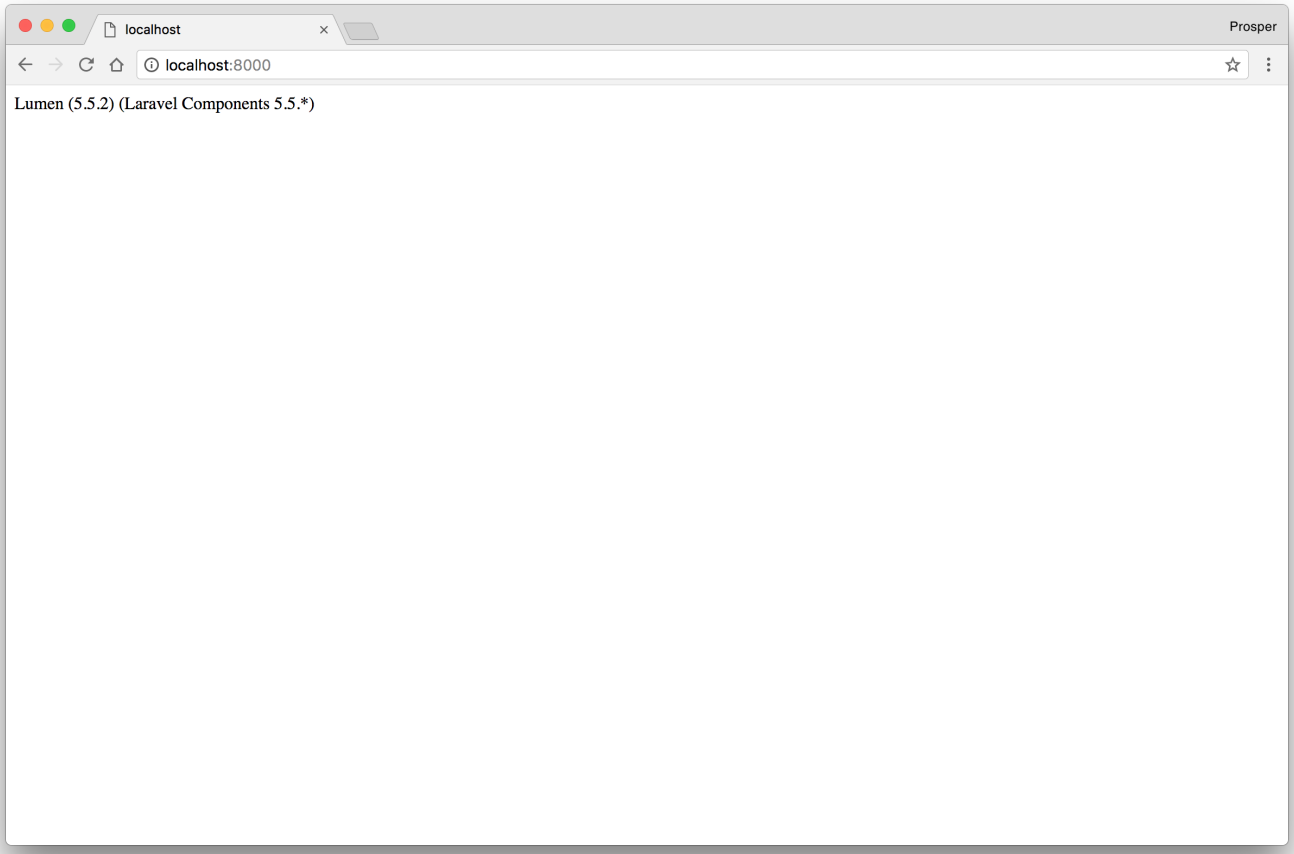
Run the following command in your terminal to create a new project with Lumen:

```
composer create-project --prefer-dist laravel/lumen authors
```

`cd` into the newly created project.

```
cd authors
```

Now, run `php -S localhost:8000 -t public` to serve the project. Head over to your browser. You should see the index page like so:



Authors Index

Activate Eloquent and Facades

As I mentioned earlier, the entire bootstrap process is located in a single file. Open up the `bootstrap/app.php` and uncomment this line, `// app->withEloquent`. Once uncommented, Lumen hooks the Eloquent ORM with your database as configured in the `.env` file.

> **Make sure you set the right details for your database in the .env file.**

In addition uncomment this line `//$app->withFacades();` . Once uncommented, we can make use of Facades in our project.

Setup Database, Models and Migrations

At the time of this writing, Lumen supports four database systems: MySQL, Postgres, SQLite, and SQL Server. We are making use of MySQL in this tutorial. First, we'll create a migration for the Authors table.

> **Migrations are like version control for your database, allowing your team to easily modify and share the application's database schema.**

Run the command below in the terminal to create the `Authors` table migration:

```
php artisan make:migration create_authors_table
```

The new migration will be placed in your `database/migrations` directory. Each migration file name contains a timestamp which allows Lumen to determine the order of the migrations. Next, we'll modify the recently created migration to accommodate the `Authors` attributes.

Open up the migration file and modify it like so:

```
<?php

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

class CreateAuthorsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('authors', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email');
            $table->string('github');
            $table->string('twitter');
            $table->string('location');
            $table->string('latest_article_published');
            $table->timestamps();
        });
    }

    /**
```

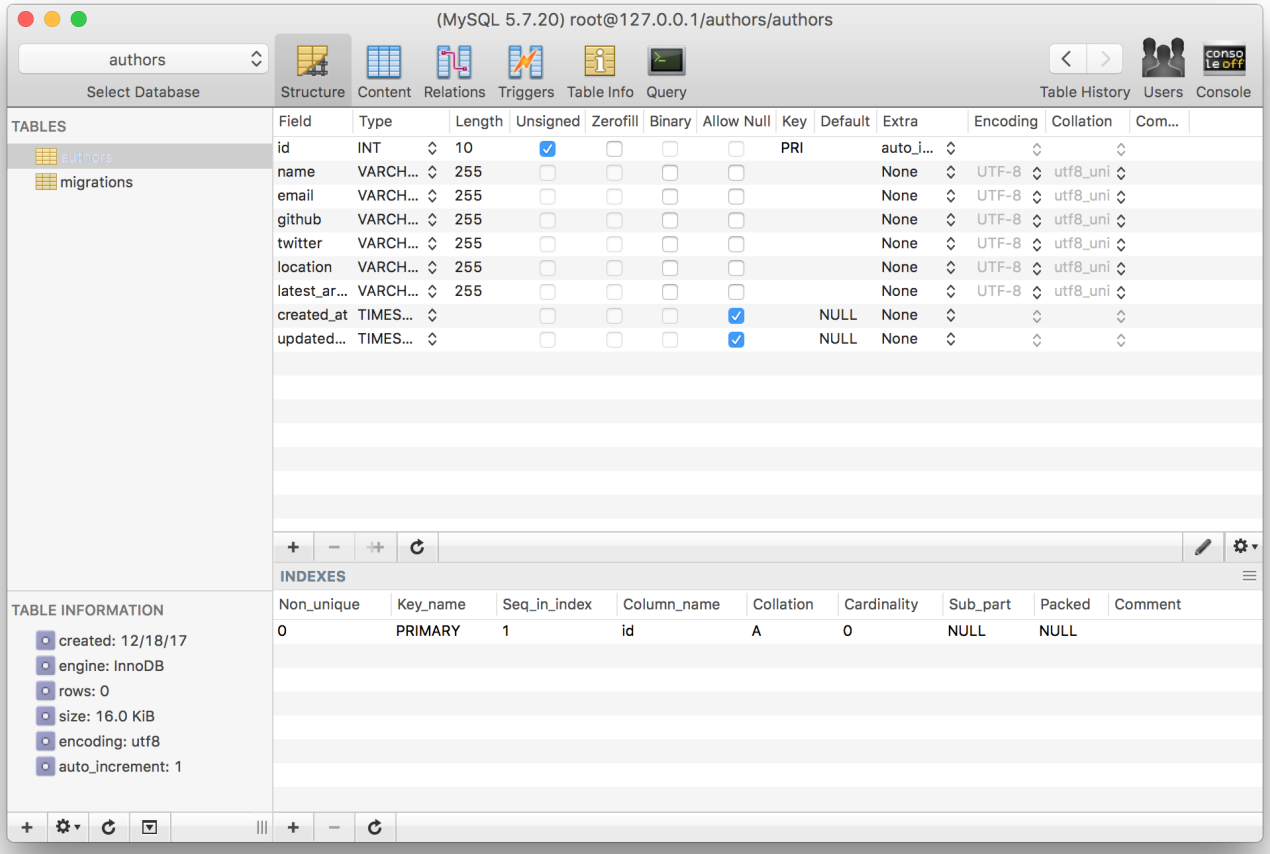
```
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('authors');
    }
}
```

In the code above, we added the columns to the `authors` table.

Now, go ahead and run the migration like so:

```
php artisan migrate
```

Check your database. You should have the `authors` and `migrations` table present.



Let's create the `Author` model. Create an `app/Author.php` file and add the code below to it:

app/Author.php

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Author extends Model
{

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
```

```
    protected $fillable = [
        'name', 'email', 'github', 'twitter', 'location', 'latest_article_published'
    ];

    /**
     * The attributes excluded from the model's JSON form.
     *
     * @var array
     */
    protected $hidden = [];
}
```

In the code above, we made the author attributes mass assignable.

Set up Routes

Routing is straight-forward. Open up `routes/web.php` and modify it like so:

```
<?php

/*
|--------------------------------------------------------------------------
| Application Routes
|--------------------------------------------------------------------------
|
| Here is where you can register all of the routes for an application.
| It is a breeze. Simply tell Lumen the URIs it should respond to
| and give it the Closure to call when that URI is requested.
|
*/

$router->get('/', function () use ($router) {
    return $router->app->version();
});

$router->group(['prefix' => 'api'], function () use ($router) {
  $router->get('authors',  ['uses' => 'AuthorController@showAllAuthors']);

  $router->get('authors/{id}', ['uses' => 'AuthorController@showOneAuthor']);
```

```
    $router->post('authors', ['uses' => 'AuthorController@create']);

    $router->delete('authors/{id}', ['uses' => 'AuthorController@delete']);

    $router->put('authors/{id}', ['uses' => 'AuthorController@update']);
});
```

In the code above, we have abstracted the functionality for each route into a controller, `AuthorController`. Route groups allow you to share route attributes, such as middleware or namespaces, across a large number of routes without needing to define those attributes on each individual route. Therefore, every route will have a prefix of `/api`. Next, let's create the Author Controller.

Set up Author Controller

Create a new file, `AuthorController.php` in `app/Http/Controllers` directory and add the following code to it like so:

```
<?php

namespace App\Http\Controllers;

use App\Author;
use Illuminate\Http\Request;

class AuthorController extends Controller
{

    public function showAllAuthors()
    {
        return response()->json(Author::all());
    }

    public function showOneAuthor($id)
    {
        return response()->json(Author::find($id));
    }

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

```
        $author = Author::create($request->all());

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

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

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

    public function delete($id)
    {
        Author::findOrFail($id)->delete();
        return response('Deleted Successfully', 200);
    }
}
```

Let's analyze the code above. First, we required the Author model, `use App\Author`. Moving forward, we invoked the necessary methods from the Author model for each controller method. We have five methods here. `showAllAuthors`, `showOneAuthor`, `create`, `update` and `delete`.

- `showAllAuthors` - /GET
- `showOneAuthor` - /GET
- `create` - /POST
- `update` - /PUT
- `delete` - /DELETE

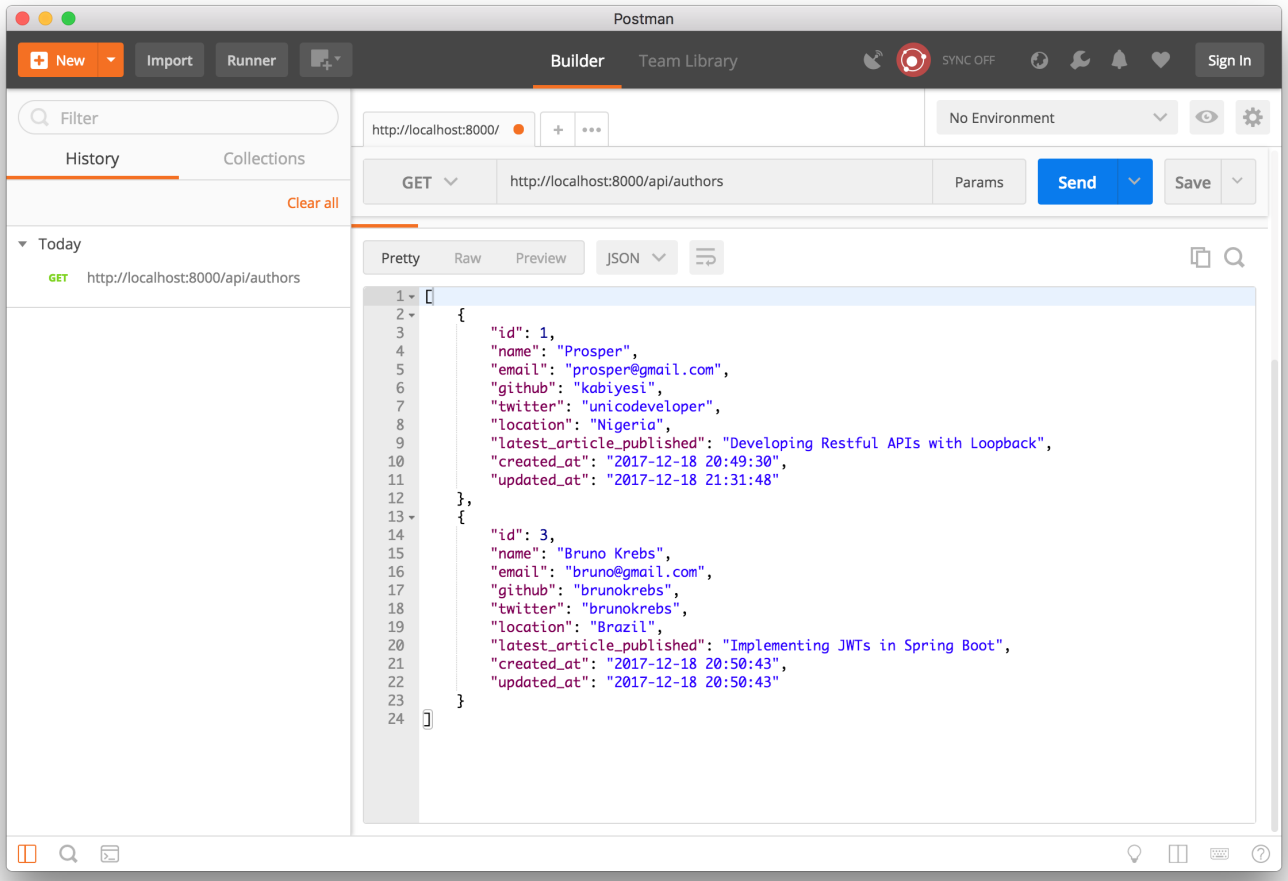
For example, if you make a POST request to `/api/authors` API endpoint, the `create` function will be invoked.

- The `showAllAuthors` method checks for all the author resources.
- The `create` method creates a new author resource.
- The `showOneAuthor` method checks for a single author resource.
- The `update` method checks if an author resource exists and allows the resource to be updated.
- The `delete` method checks if an author resource exists and deletes it.

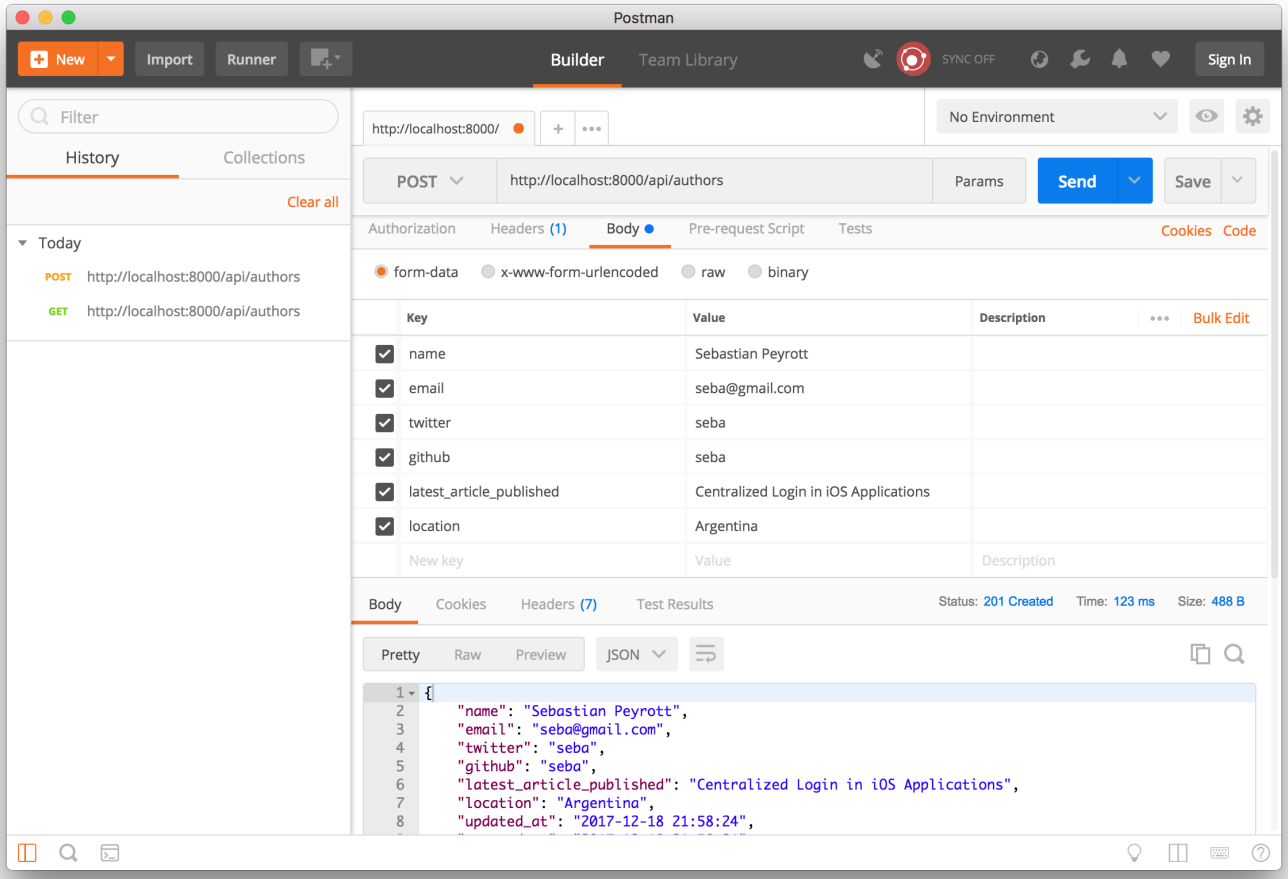
- `response()` is a global helper function that obtains an instance of the response factory. `response()->json()` simply returns the response in JSON format.

- `200` is an HTTP status code that indicates the request was successful.
- `201` is an HTTP status code that indicates a new resource has just been created.
- `findOrFail` method throws a `ModelNotFoundException` if no result is not found.

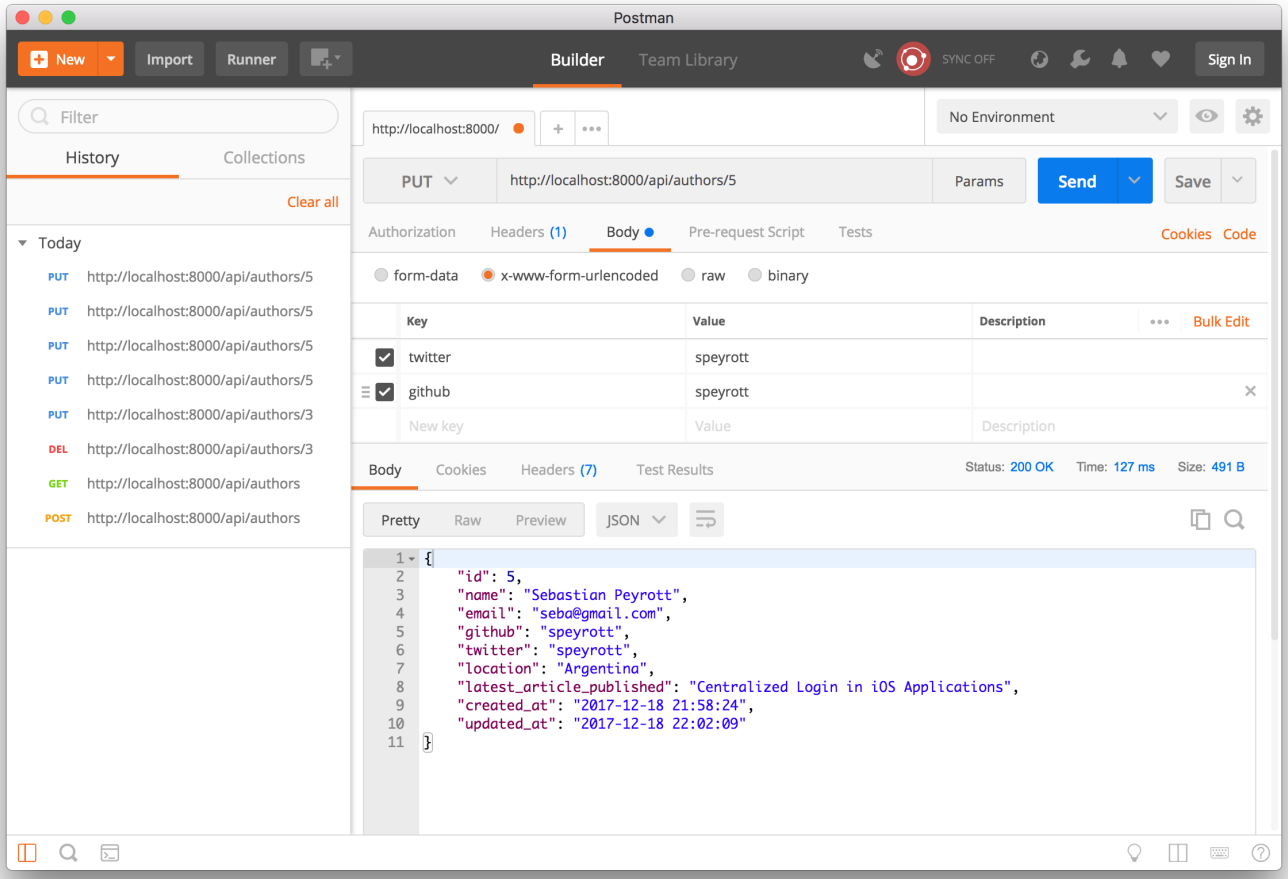
Finally, test the API routes with Postman.



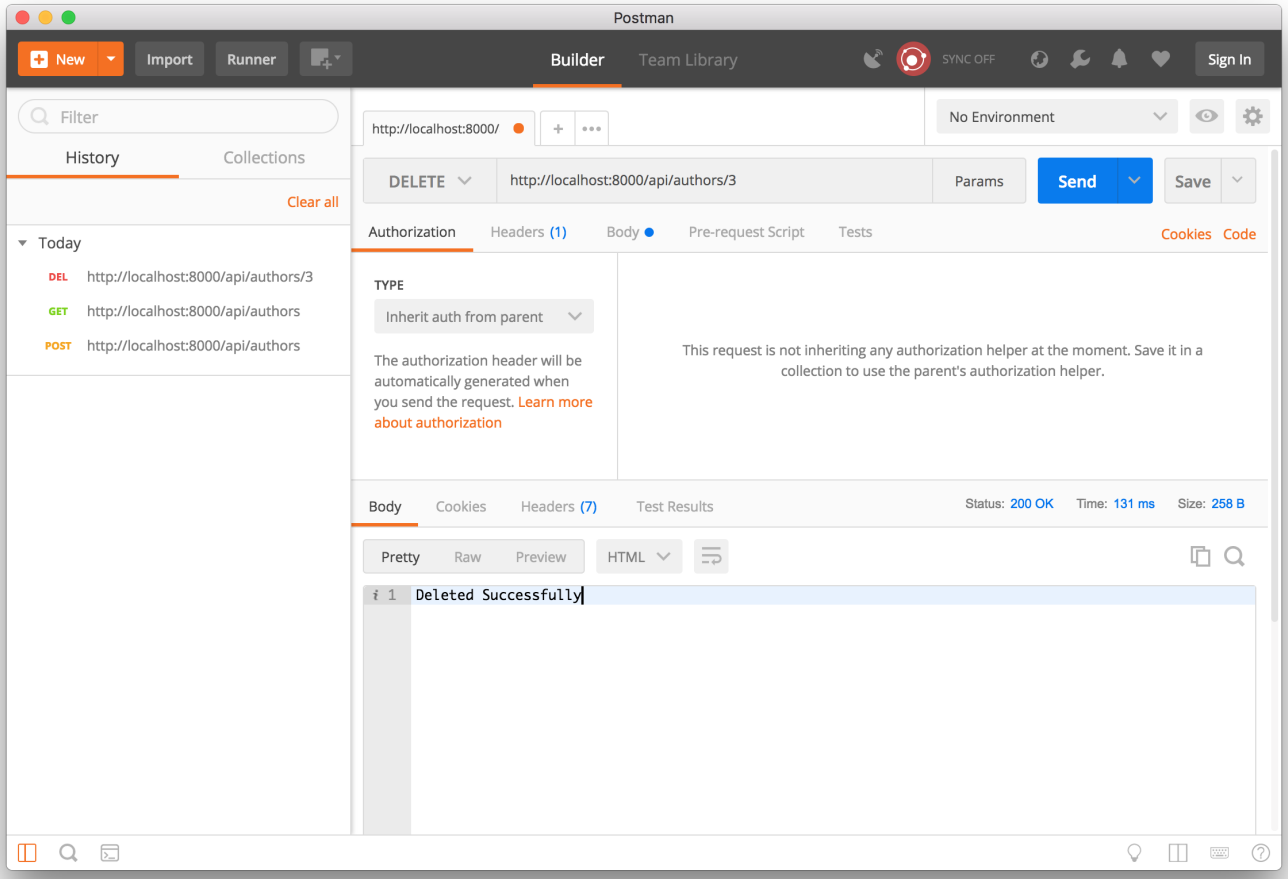
Author GET operation



Author POST operation



Author PUT operation



Author DELETE operation

Our API works. Awesome!

Lumen API Validation

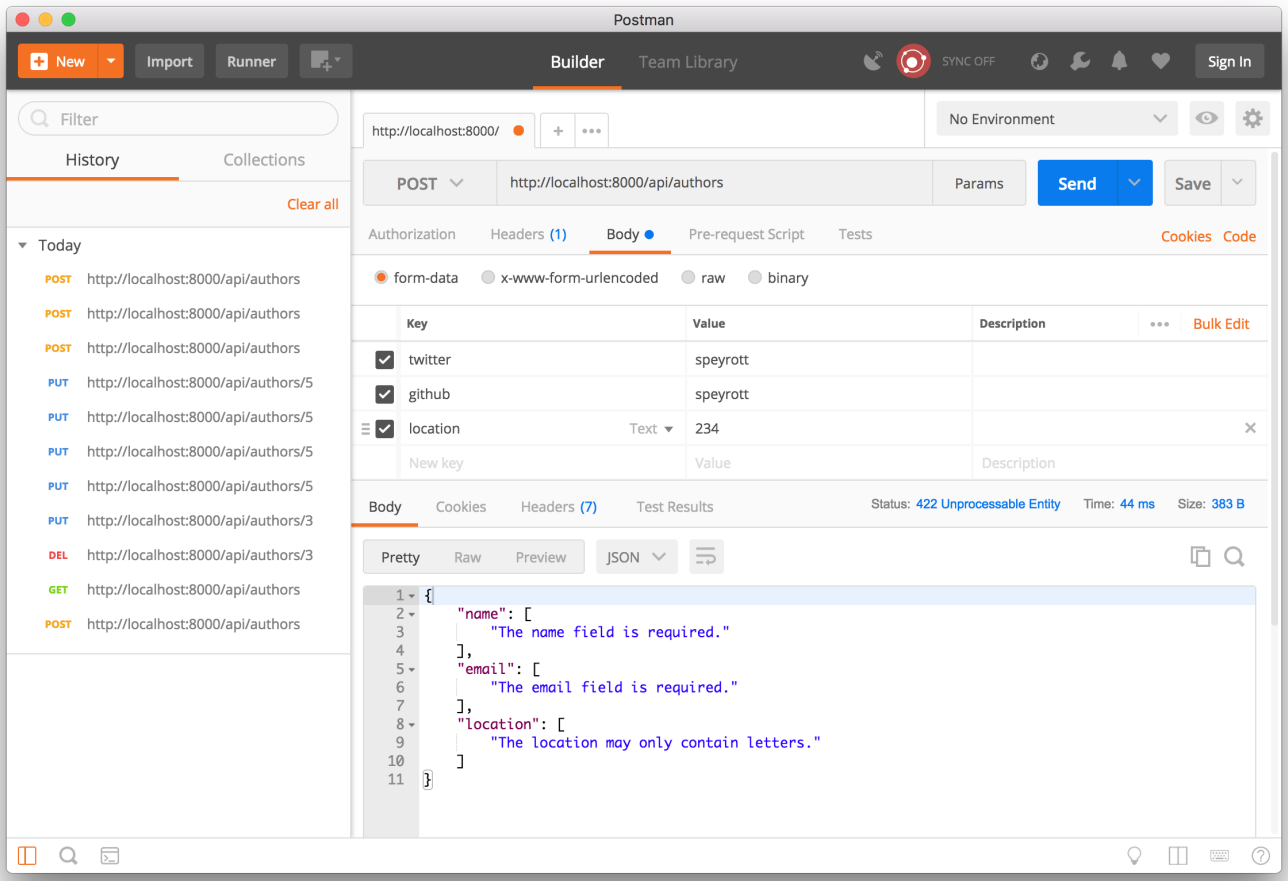
When developing applications, never trust the user. Always validate incoming data. In Lumen, it's very easy to validate your application's incoming data. Lumen provides access to the `$this->validate` helper method from within Route closures.

Open up the `AuthorController` file and add modify the `create` method like so:

```
...
 public function create(Request $request)
    {
        $this->validate($request, [
            'name' => 'required',
            'email' => 'required|email|unique:users',
            'location' => 'required|alpha'
        ]);
```

```
        $author = Author::create($request->all());

        return response()->json($author, 201);
    }
...
```

Test the API POST route with Postman.



It validated the incoming requests and returned the appropriate error message.

- *name*, *email*, and *location* were required. In testing the API, *name* and *email* were not provided.
- *email* was required to be in email format.
- *location* was required to be entirely alphabetic characters, `alpha`. Nothing more. Numbers were provided as the value for *location*.

> **Note: Always validate incoming data. Never trust your users!**

Check out a plethora of validation rules that you can use with Lumen.

Securing the Authors API with Auth0

Right now, anyone can make `GET` and `POST` requests to all of the endpoints present in our API. In a real-world scenario, we should restrict `POST`, `DELETE` and `PUT` requests to certain registered and authorized users.

We'll go ahead and secure some of these API endpoints with JSON Web Tokens.

JSON Web Tokens, commonly known as JWTs, are tokens that are used to authenticate users on applications. This technology has gained popularity over the past few years because it enables backends to accept requests simply by validating the contents of these JWTs. That is, applications that use JWTs no longer have to hold cookies or other session data about their users. This characteristic facilitates scalability while keeping applications secure.

Whenever the user wants to access a protected route or resource (an endpoint), the user agent must send the JWT, usually in the *Authorization* header using the Bearer schema, along with the request.

When the API receives a request with a JWT, the first thing it does is to validate the token. This consists of a series of steps, and if any of these fails then, the request must be rejected. The following list shows the validation steps needed:

- Check that the JWT is well formed.
- Check the signature.
- Validate the standard claims.
- Check the Client permissions (scopes).

We will make use of Auth0 to issue our JSON Web Tokens. With Auth0, we have to write just a few lines of code to get a solid identity management solution, including single sign-on, user management, support for social identity providers (like Facebook, GitHub, Twitter, etc.), enterprise (Active Directory, LDAP, SAML, etc.), and your own database of users.

For starters, if you haven't done so yet, this is a good time to sign up for a free Auth0 account. Having an Auth0 account, the first thing that we must do is to create a new API on the dashboard. An API is an entity that represents an external resource, capable of accepting and responding to protected resource requests made by clients.

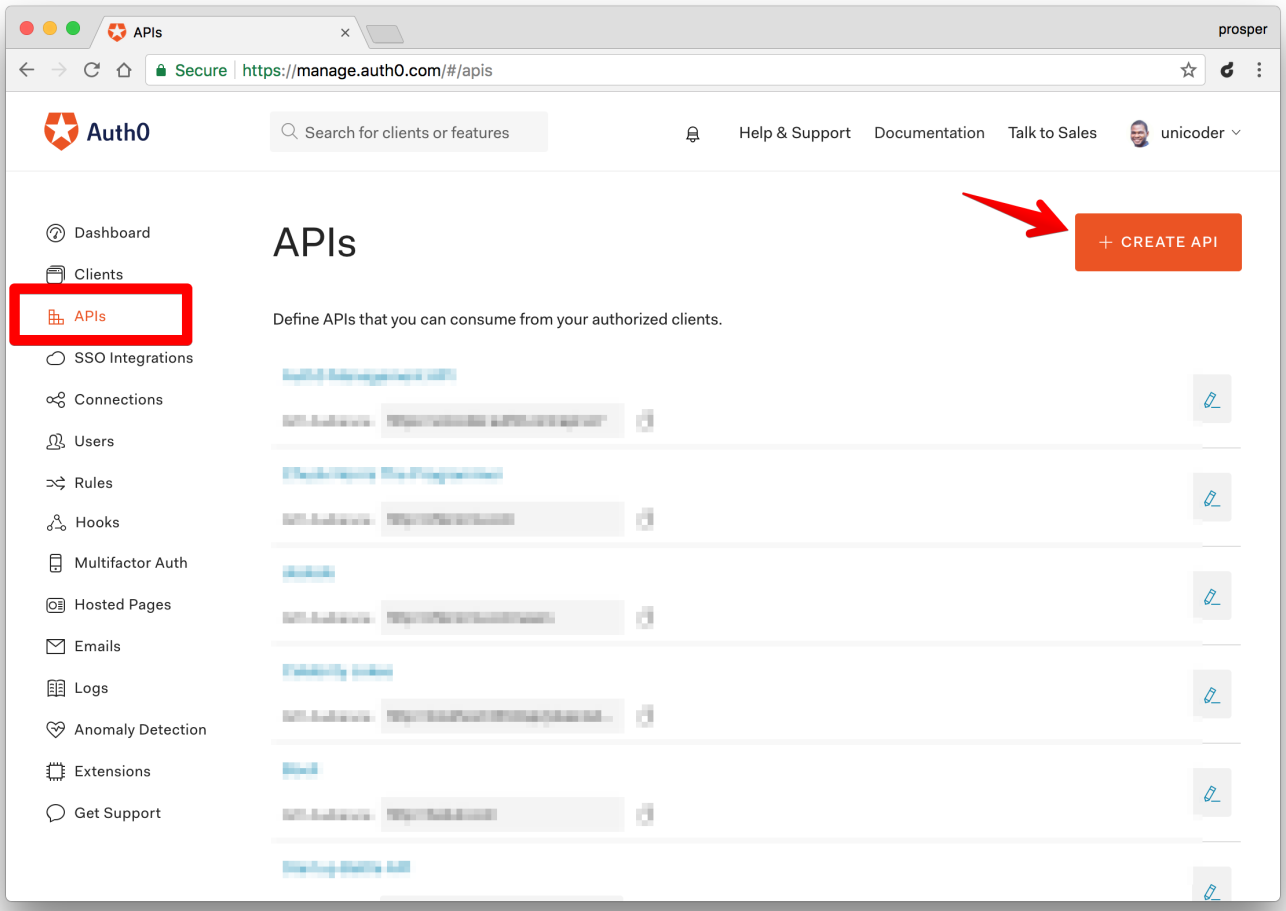
> **Auth0 offers a generous free tier** to get started with modern authentication.

Login to your Auth0 management dashboard and create a new API client.

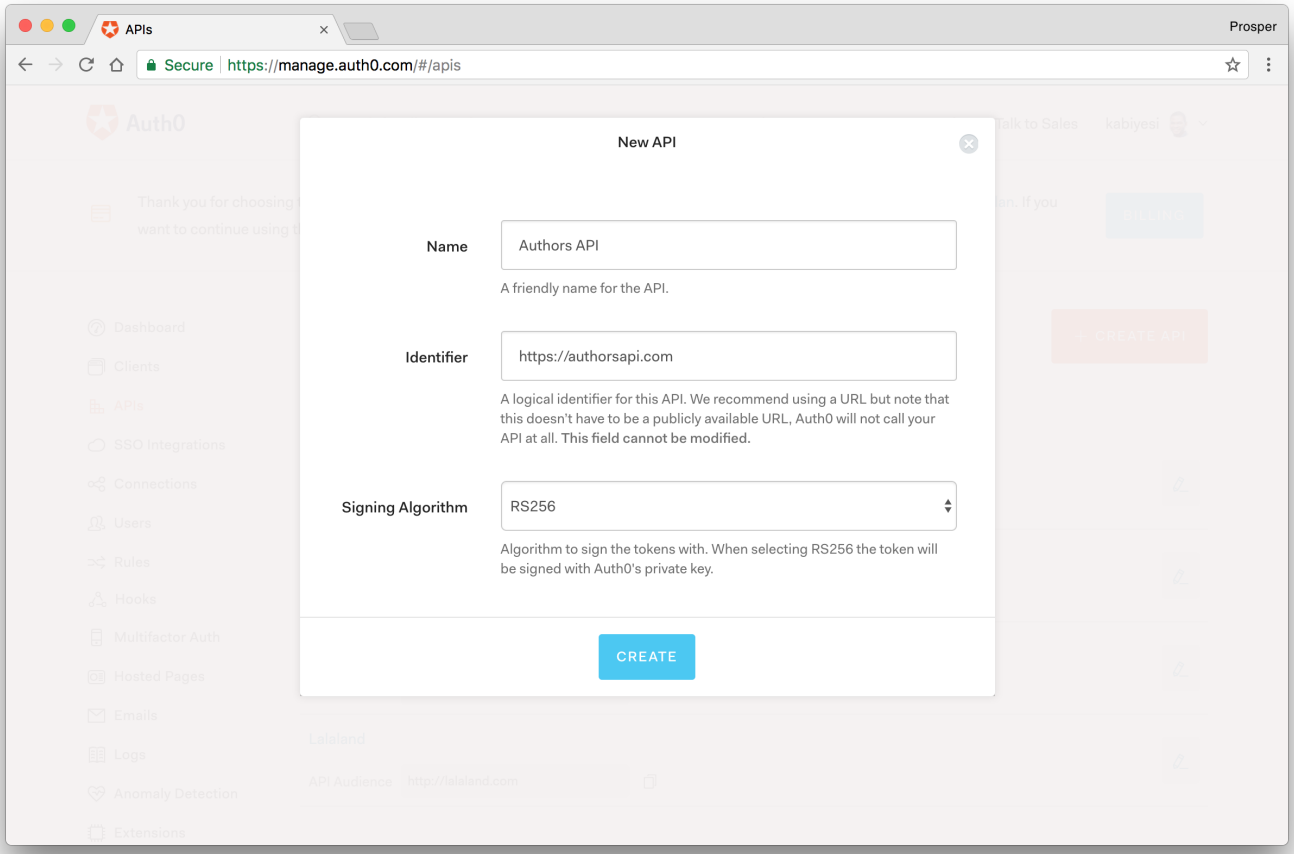
Click on the APIs menu item and then the **Create API** button. You will need to give your API a name and an identifier. The name can be anything you choose, so make it as descriptive as you want.

The identifier will be used to identify your API, this field cannot be changed once set. For our example, I'll name the API, **Authors API**, and for the identifier, I'll set it as **https://authorsapi.com**. We'll leave the signing algorithm as **RS256** and click

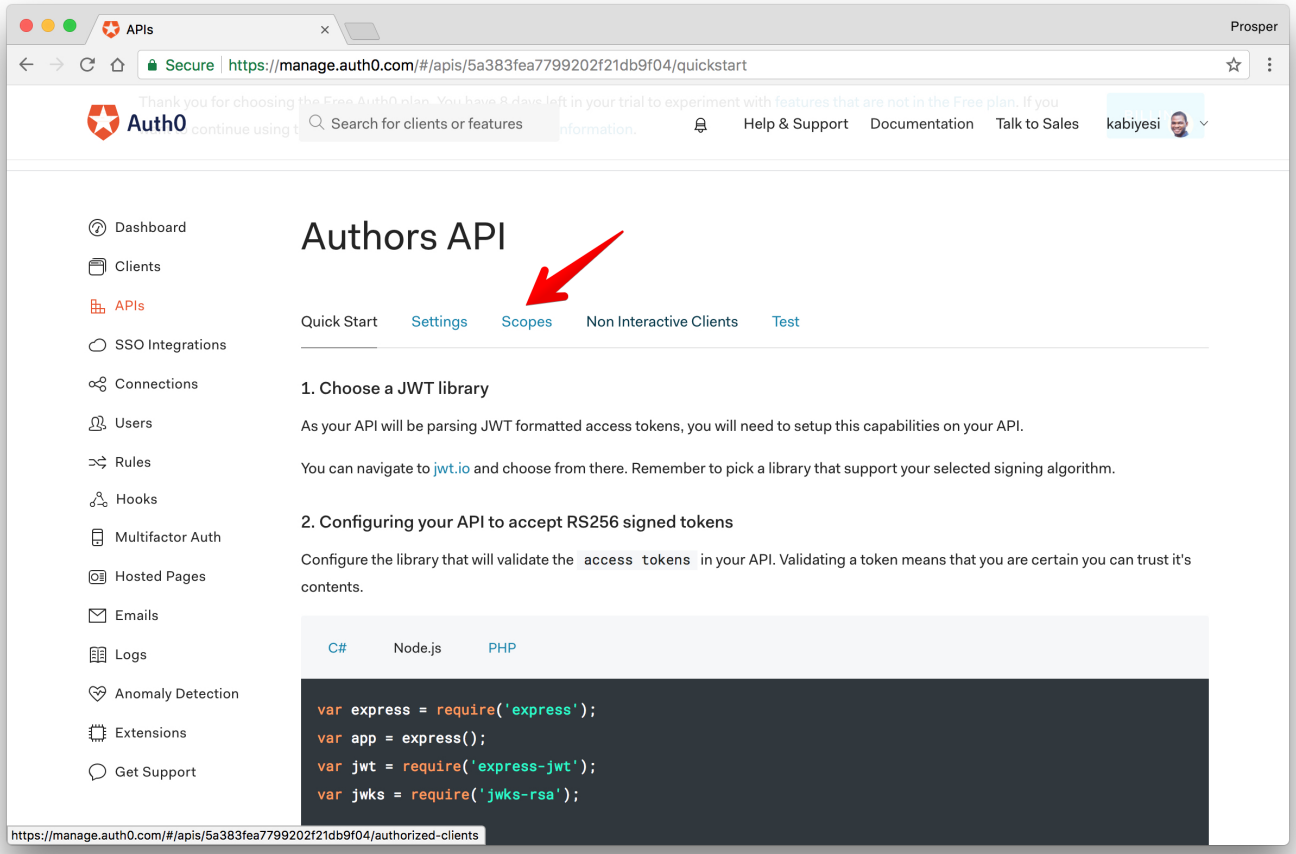
on the **Create API** button.



Create a New API



Creating the Authors API



You can define scopes in this section

Head over to your terminal and install Auth0 PHP SDK:

```
composer require auth0/auth0-php:~5.0
```

Create a new middleware file, `Auth0Middleware.php` in the `app/Http/Middleware` directory. Add the following code to it like so:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Auth0\SDK\JWTVerifier;

class Auth0Middleware
{
```

```
/**
 * Run the request filter.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \Closure  $next
 * @return mixed
 */
public function handle($request, Closure $next)
{
    if(!$request->hasHeader('Authorization')) {
      return response()->json('Authorization Header not found', 401);
    }

    $token = $request->bearerToken();

    if($request->header('Authorization') == null || $token == null) {
      return response()->json('No token provided', 401);
    }

    $this->retrieveAndValidateToken($token);

    return $next($request);
}

public function retrieveAndValidateToken($token)
{
    try {
        $verifier = new JWTVerifier([
          'supported_algs' => ['RS256'],
          'valid_audiences' => ['AUTH0_API_AUDIENCE'],
          'authorized_iss' => ['AUTH0_DOMAIN']
        ]);

        $decoded = $verifier->verifyAndDecode($token);
    }
    catch(\Auth0\SDK\Exception\CoreException $e) {
        throw $e;
    };
}
```

```
}
```

In the `retrieveAndValidateToken` method, we created an instance of `JWTVerifier` to verify the token coming from the Authorization header. It checks the algorithm, the API audience, and the issuer to ensure the token is a valid one issued by Auth0.

Note: Replace the `AUTH0_API_AUDIENCE` and `AUTH0_DOMAIN` placeholders with the API audience and Auth0 domain values from your Auth0 dashboard.

Now, we want to assign the newly created middleware to our routes. The first step is to assign the middleware a short-hand key in `bootstrap/app.php` file's call to the `$app->routeMiddleware()` method.

Go ahead and open up `bootstrap/app.php` and uncomment this line of code:

```
...
// $app->routeMiddleware([
//     'auth' => App\Http\Middleware\Authenticate::class,
// ]);
...
```

Once uncommented, replace the `Authenticate::class` with `Auth0Middleware::class` like so:

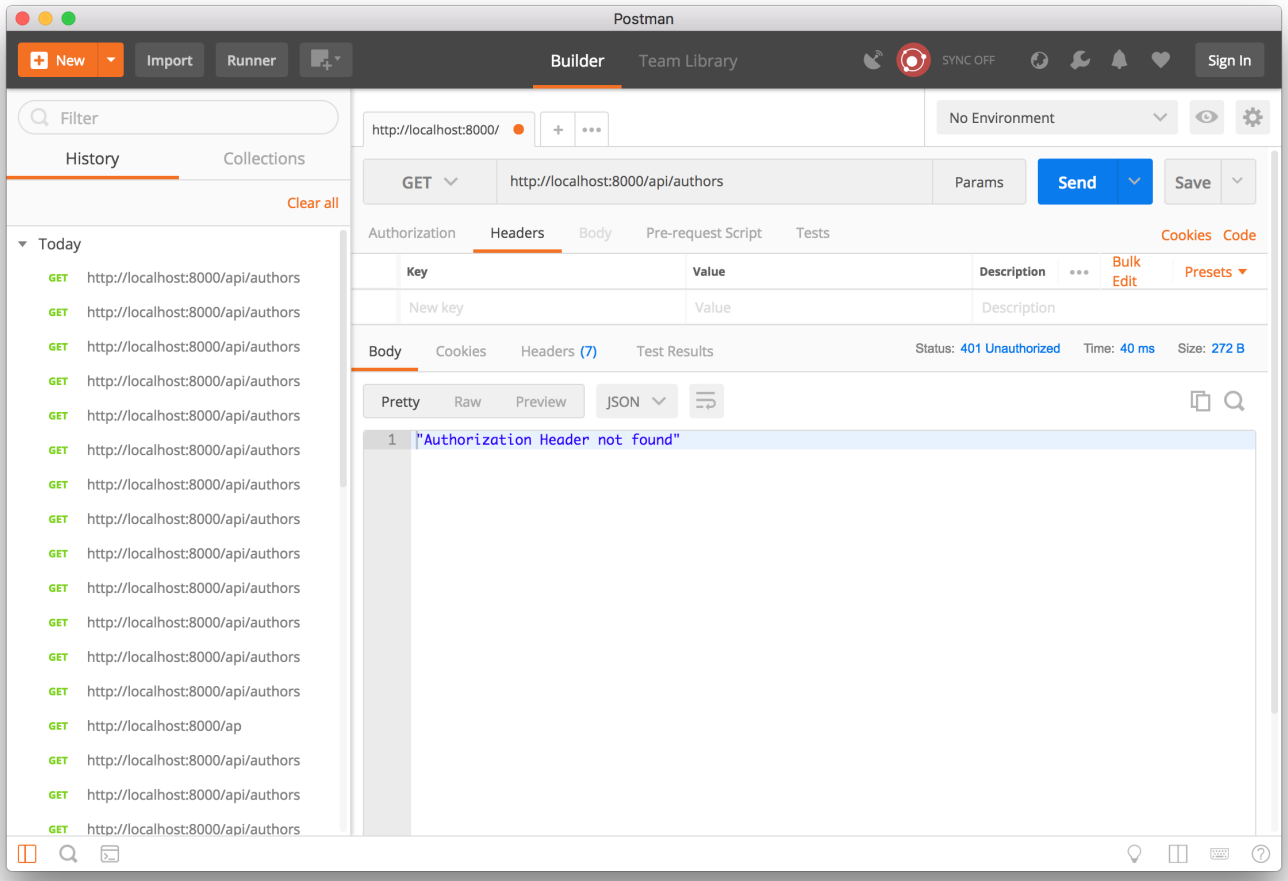
```
$app->routeMiddleware([
    'auth' => App\Http\Middleware\Auth0Middleware::class,
]);
```

Once the middleware has been defined in the HTTP kernel, as we have done above. We can now use the middleware key in the route options array in the `routes/web.php` file like so:

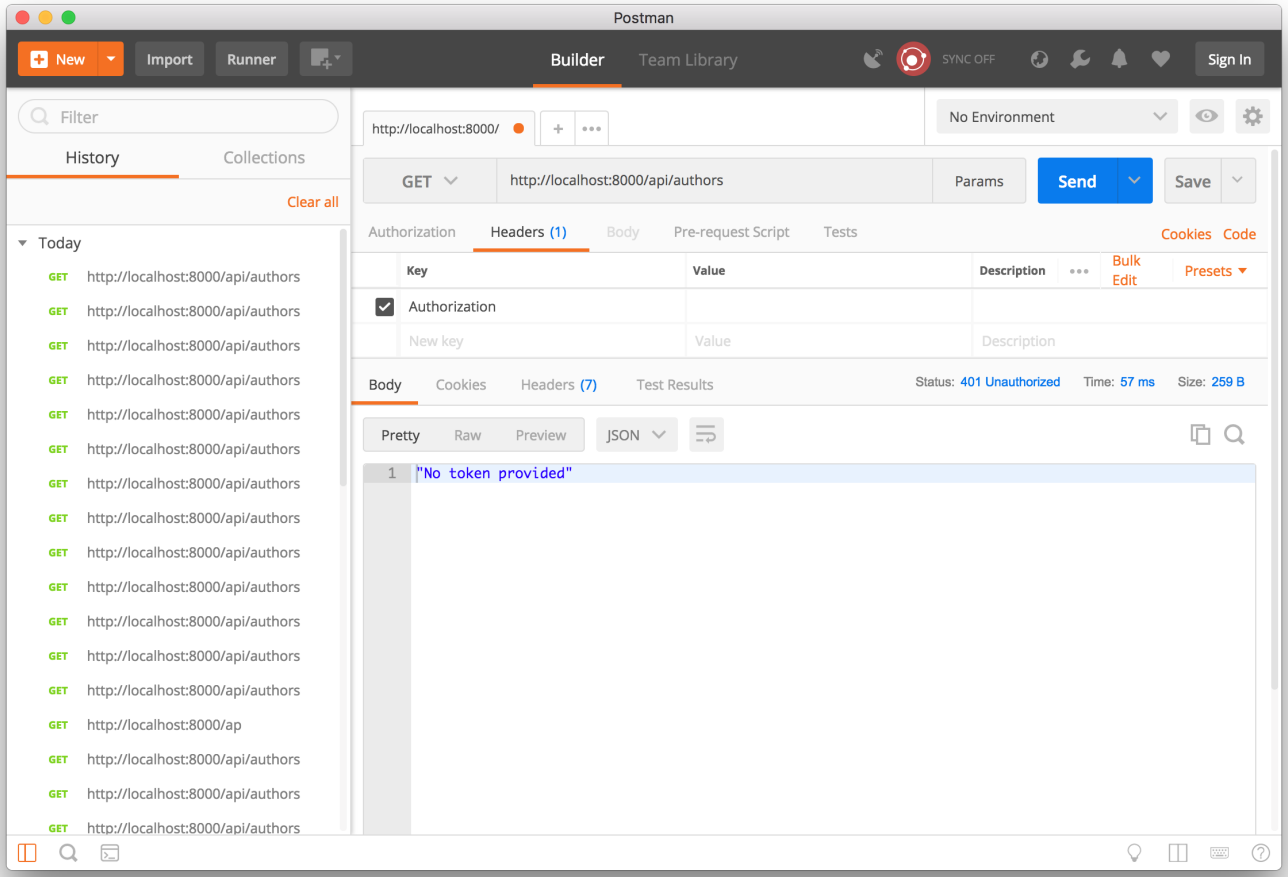
```
...
$router->group(['prefix' => 'api', 'middleware' => 'auth'], function () use ($router) {
  $router->get('authors',  ['uses' => 'AuthorController@showAllAuthors']);

  $router->get('authors/{id}', ['uses' => 'AuthorController@showOneAuthor']);
```

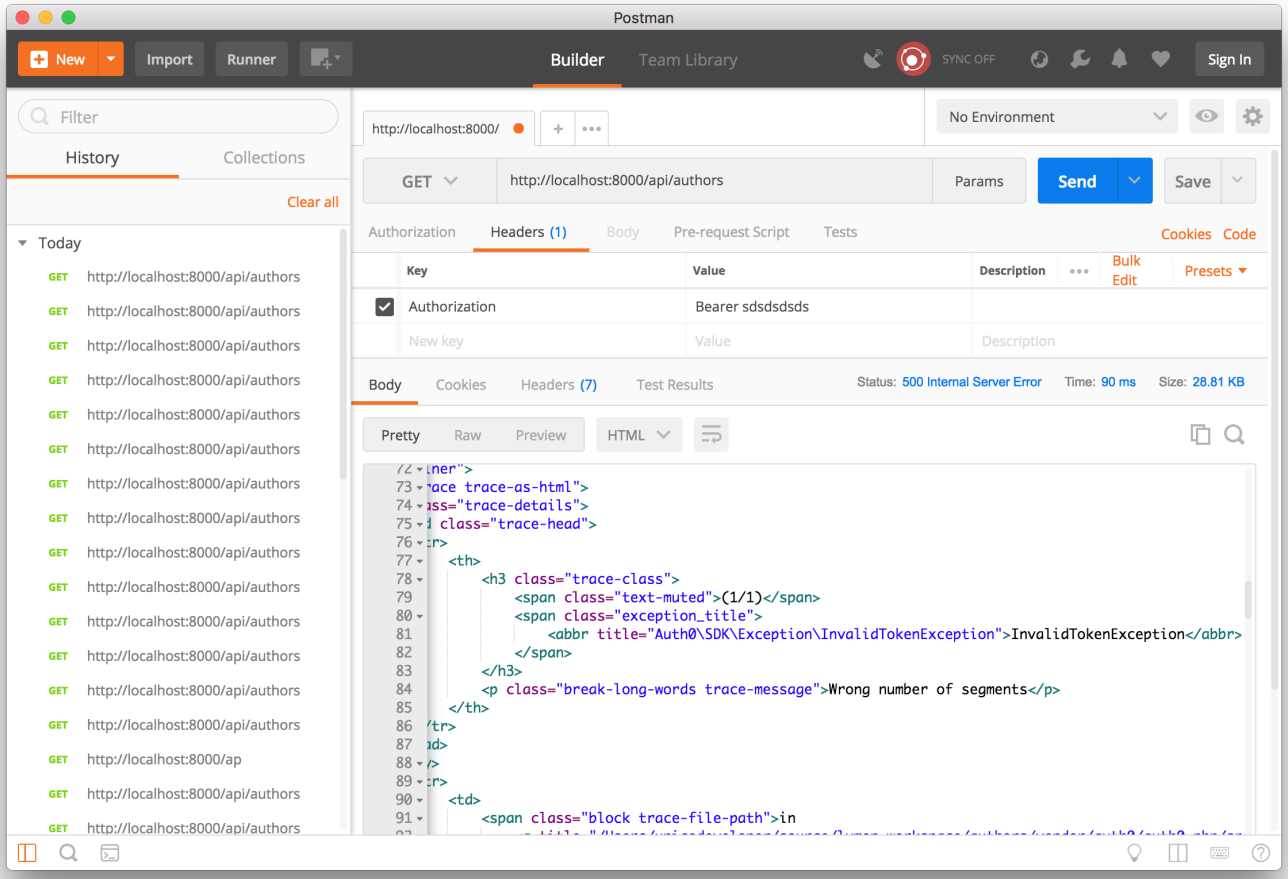
```
    $router->post('authors', ['uses' => 'AuthorController@create']);

    $router->delete('authors/{id}', ['uses' => 'AuthorController@delete']);

    $router->put('authors/{id}', ['uses' => 'AuthorController@update']);
});
```

We just secured all the API endpoints with JWT. If a user accesses these API endpoint/route without a valid access token or no token at all, it returns an error. Try it out.



Accessing any endpoint without an authorization header

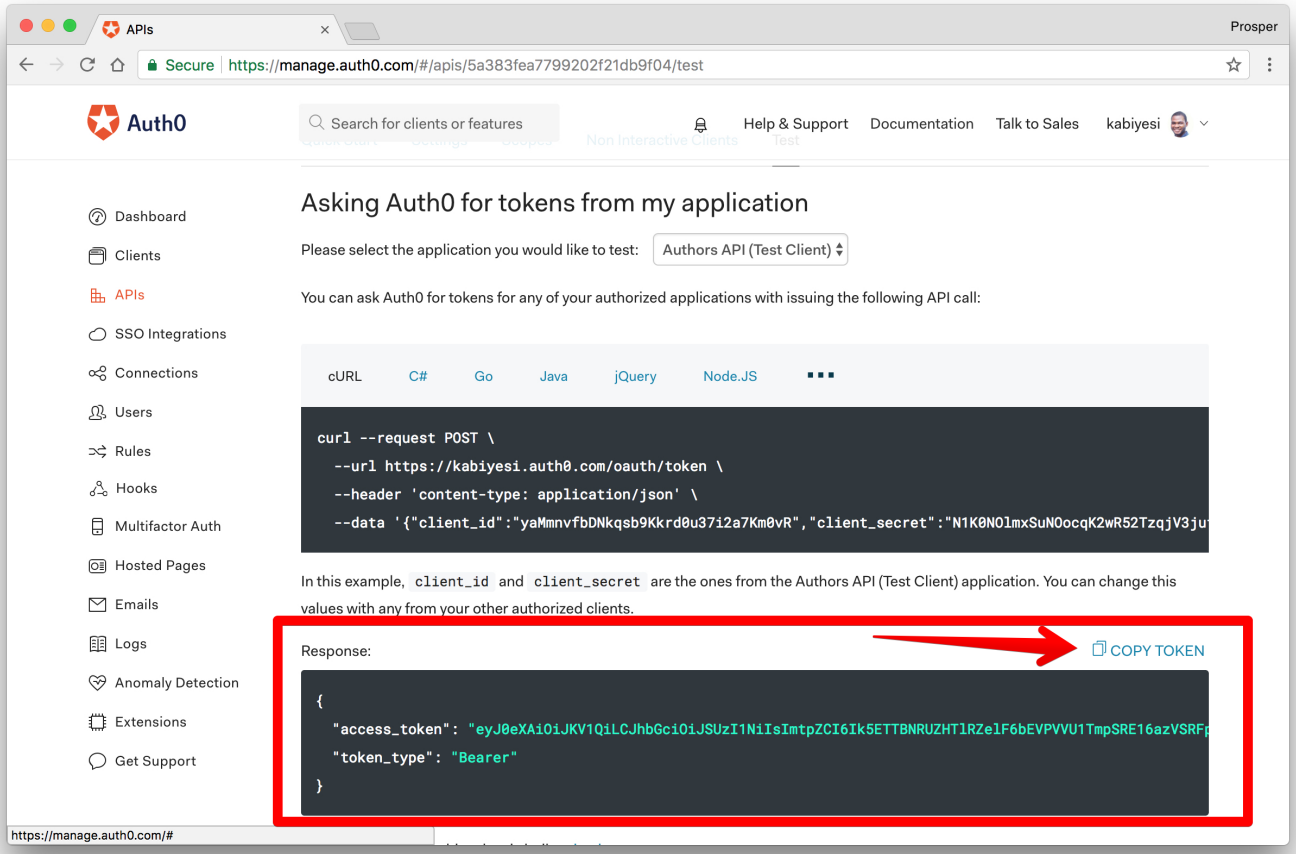

Postman
New
Import
Runner
Builder
Team Library
SYNC OFF
Sign In
Filter
History
Collections
Clear all
Today
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/ap
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
GET http://localhost:8000/api/authors
http://localhost:8000/
No Environment
GET
http://localhost:8000/api/authors
Params
Send
Save
Authorization
Headers (1)
Body
Pre-request Script
Tests
Cookies
Code
Key
Value
Description
Bulk Edit
Presets
Authorization
New key
Value
Description
Body
Cookies
Headers (7)
Test Results
Status: 401 Unauthorized
Time: 57 ms
Size: 259 B
Pretty
Raw
Preview
JSON
1
"No token provided"



Accessing any endpoint without a valid access token

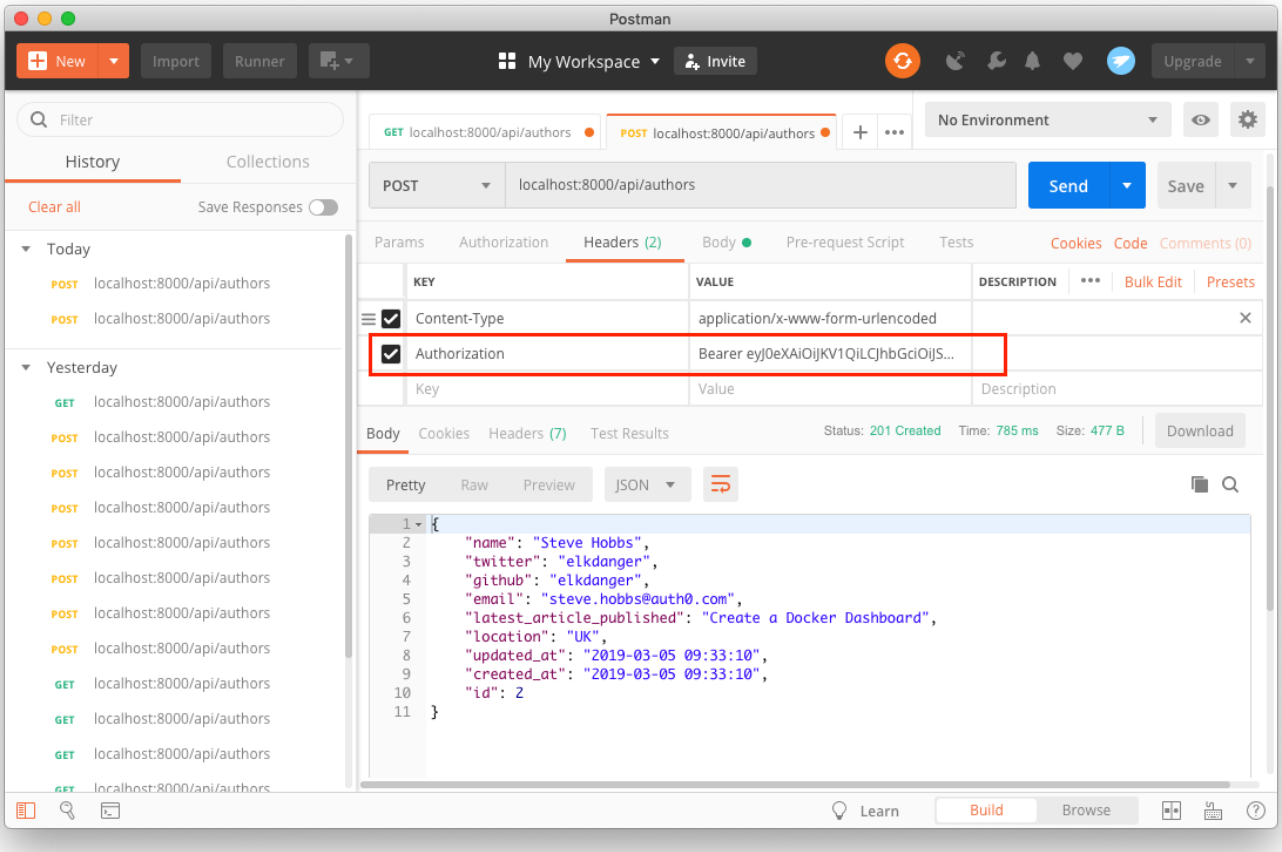
Now, let's test it with a valid access token. Head over to the `test` tab of your newly created API on your Auth0 dashboard.

Grab the Access token from the *Test* tab



Grab the Access Token

Now use this `access token` in Postman by sending it as an Authorization header to make a POST request to `api/authors` endpoint.



Accessing the endpoint securely

It validates the access token and successfully makes the POST request.

Wondering how to integrate the secure API with a frontend? Check out our amazing React and Vue.js authentication tutorials.

Conclusion

Well done! You have learned how to build and secure a rest API with the powerful PHP micro-framework, Lumen, and JWT. Need to use PHP to build your API or micro-service? I'd bet on Lumen as the tool of choice for speed and ease of use.

In addition, Auth0 can help secure your **API** easily. Auth0 provides more than just username-password authentication. It provides features like multifactor auth, breached password detection, anomaly detection, enterprise federation, single sign on (SSO), and more. Sign up today so you can focus on building features unique to your app.

Please, let me know if you have any questions or observations in the comment section. 😊

> **AUTH0 DOCS**
> Implement Authentication in Minutes
>
> **AUTH0 COMMUNITY**

Join the Conversation

Prosper Otemuyiwa

Prosper is a great speaker, community leader, open source hacker, technical consultant, and a fervent Developer Advocate. He is a full-stack software engineer who has worked on biometric, health and developer tools. Prosper recently co-founded a startup called Eden that's focused on improving the quality of lives in Nigeria and currently leads the engineering team. He also co-founded forloop, the largest developer community in Africa.

VIEW PROFILE

More like this

KEYSTONE

Developing Web Apps and RESTful APIs with KeystoneJS

LOOPBACK

Developing RESTful APIs with Loopback

SPRING BOOT

Implementing JWT Authentication on Spring Boot APIs

Follow the conversation

Comments Community 1 Login

Recommend 6 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Adagio • 6 months ago

How does one now access the Userdata of whom made the API-Request i.e. I want to keep track of who changed data via PUT.

1 • Reply • Share ›

Bruno S. Krebs Mod Adagio • 6 months ago

Well, for that you will need to create an Auth0 Application and let users sign in. Then, you can issue a request to the /userinfo endpoint with the access token retrieved while authenticating to see who made the request.

Or, better yet, if your API is part of your web application (or SPA), you can use id tokens instead of access tokens.

• Reply • Share ›

pararang • a year ago

nice explanation, thank you

1 • Reply • Share ›

Francis Sunday • a year ago

Hey Prosper, nice guide on Lumen, this is super helpful

1 • Reply • Share ›

Tomáš Teicher • a day ago

Hi, could anybody help to explain, how the connection was made, between Author class and "authors" schema? How the Author class knows, that it should work with "authors" database table?
Many thanks :)

• Reply • Share ›

Andrew Beak • 2 months ago

What if there is more than one person using your API? There is no concept of identity in this tutorial. How do you determine the "sub" of the cookie and use that in your controller?

Reply • Share ›

pinoyCoder • 3 months ago

Hi,
Thanks for this, but on client side how can you secure the tokens,example on my client side they will use an ajax request and i notice that using the google chrome developer tools i can see the event which the request that occurred and event the token in the header, so the information can still be steal and use buy someone to abuse the my application. Can you please advise how can i prevent or mitigate that ?

Reply • Share ›

Bruno S. Krebs Mod ➚ pinoyCoder • 2 months ago

By abuse your application you mean by injecting some malicious JavaScript in _your_ application to intercept the AJAX request? If that is the case, then I don't think you have much to do in relation to the tokens itself. You would have to be more careful about your app as a whole because having a weak app like that will cause you a lot of other problems (besides losing a token).

Anyway, I just thought a little bit more about it and you could use HTTP-only cookies to help you there.

Reply • Share ›

Monali Patel • 3 months ago

Hi Is it Micro service API?

Reply • Share ›

Bruno S. Krebs Mod ➚ Monali Patel • 2 months ago

Sorry, what?

Reply • Share ›

Marcus Zamora • 4 months ago

I can't get the result on the lsat part. :(Why dude? I get the result on all, but why this last image I cant get it. Is it really "http://localhost:8000/api/people/" not "http://localhost:8000/api/author/"?

Reply • Share ›

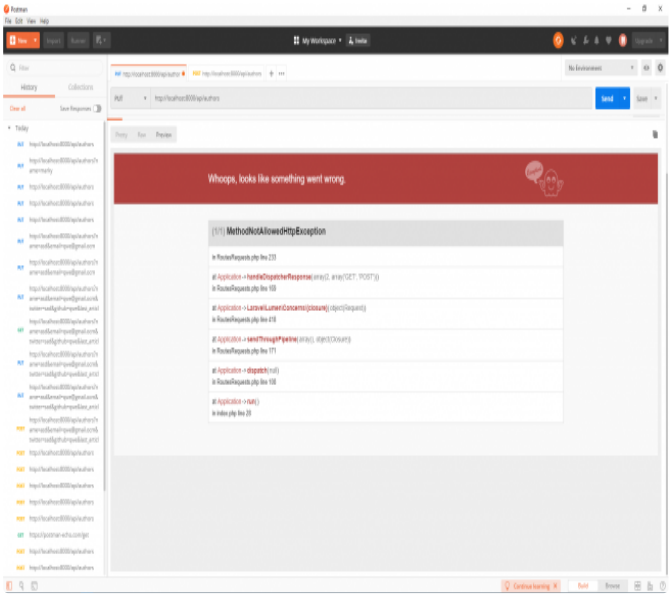
Steve Hobbs ➚ Marcus Zamora • 3 months ago

Hey Marcus,

I think you're right, it should be `http://localhost:8000/api/authors` - does that work for you?

In the meantime I'll see about getting this part fixed - thanks for lettting us know about the problems you're having!

Reply • Share ›



help me please

Reply • Share ›

Sivaramadurai nadar ➚ Mark Louise Fernandez
• 3 months ago

Hi, Instead of PUT, use GET to fetch the authors.

Reply • Share ›

Bruno S. Krebs Mod ➚ Mark Louise Fernandez
• 4 months ago

Hi, have you compared your project with the one built by the author? https://github.com/auth0-bl...

Reply • Share ›

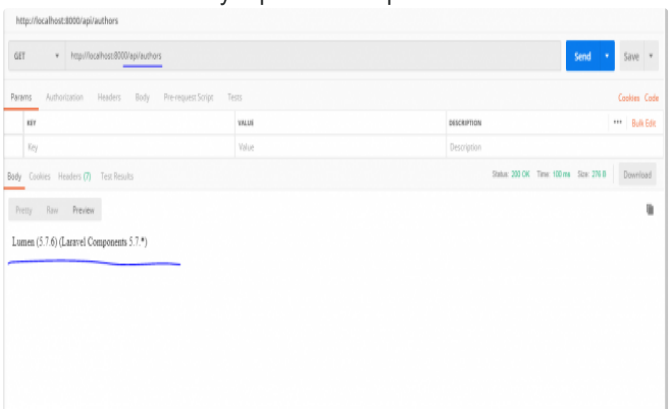
muzanaka • 5 months ago

Мужики, спасибо! Еще бы показали как swagger'ор все описать, и было бы вообще готовая инструкция к действию.

Reply • Share ›

Regita Drajat • 6 months ago

i cant get any reponse
i try to hit http://localhost:8000/api/authors but its return "Lumen (5.7.6) (Laravel Components 5.7.*)"
->here model and my reponse from postman



see more

Reply • Share ›

Ali Ahmed • 8 months ago

I don't know why Lumen hasn't been able to find my controller classes lately, I had to pass a namespace argument to the group with App\Http\Controllers as value to make it work properly.

Reply • Share ›

Aishwarya Zipare • 10 months ago

getting error undefined variable:router

Reply • Share ›

Bruno S. Krebs Mod ➚ Aishwarya Zipare • 10 months ago

Hey there, perhaps this might help you?

Reply • Share ›

Luis Carlos Silva Magallanes • a year ago

Hey prosper. Greate tutorial but I have a problem.
When I try to make a POST (create) I have this message from de Postman: SQLSTATE[42S02]: Base table or view not found: 1146 Table 'authors.users' doesn't exist (SQL: select count(*) as aggregate from `users` where `email` = jp@mail.com)
Any idea about this? Thanks.

Reply • Share ›

b1kjsh ➚ Luis Carlos Silva Magallanes • 8 months ago

Just change the following validation from
'email' => 'required|email|unique:users'
To:
'email' => 'required|email|unique:authors'

Or you can create a table for users using the migrate commands in the earlier part of the tutorial. However, using authors instead of users makes more sense in this tutorial. They should probably just change that in the tutorial.

Reply • Share ›

Luk Berezowski ➚ Luis Carlos Silva Magallanes
• 10 months ago

This is due the validation rule set for the `email` field (in `create` method inside `AuthorController`) which states:
'email' => 'required|email|unique:users',

Just remove `unique:users` so it reads:
'email' => 'required|email',

You have most likely sorted that issue already, since it's from 2 month back, but maybe it will help someone else.

Cheers

• Reply • Share ›

Bruno S. Krebs Mod ➚ Luk Berezowski
• 10 months ago

Thanks for contributing.

• Reply • Share ›

Michael Holland • a year ago

Really useful quick-start guide to Lumen as well as Auth0. However, I think it might be more 'real-world' if you described how to selectively apply the authentication. For example, anyone could list authors but only authenticated users could add or edit.

• Reply • Share ›

Bruno Krebs ➚ Michael Holland • a year ago

Thanks for the feedback. I will take a note and we will try to improve it in our next articles.

• Reply • Share ›

Somcutean Doru • a year ago

Great tutorial! Thanks for explaining this so nicely and in an easy way.

• Reply • Share ›

NandoBas • a year ago

tanks, its work!
now i am integrate with React.

• Reply • Share ›

Dwi Yudi Rayi Anugrah • a year ago

Error message "We can't trust on a token issued by" why ?

• Reply • Share ›

Juan Manuel Heredia Gallardo • a year ago

hey prosper, thanks for the guide, i followed it but i got this error "cURL error 60: SSL certificate problem: self signed certificate in certificate chain", i update php.ini with this "curl.cainfo = "C:\wamp64\bin\php\php7.0.10\extras\ssl\cacert.pem" but nothing change ... any idea ??

Prosper ➚ Juan Manuel Heredia Gallardo • a year ago

Hello **@Juan Manuel Heredia Gallardo** please update the cURL on your server to the latest version. Do you also have SSL activated on your server?

Reply • Share ›

Juan Manuel Heredia Gallardo ➚ Prosper • a year ago

finally i got it working ...

Reply • Share ›

kunamistu • a year ago

Managed to configure everything... but how do I access the User information from my backend after it is authorized?

Reply • Share ›

Rohani Suhadi • a year ago

Reply • Share ›

Sinan ➚ Rohani Suhadi • a year ago

Hi **@Rohani Suhadi**, I had the same error before and I solved this by adding a trailing slash in authorized_iss on Auth0Middleware. See the image:

Reply • Share ›

Mohd Agung Melati ➚ Sinan • 4 months ago
thanks
Reply • Share ›

Show more replies

Rohani Suhadi • a year ago
Show error.
We can't trust on a token issued by:
Reply • Share ›

Sinan • a year ago
Where could I find these two? AUTH0_API_AUDIENCE
AUTH0_DOMAIN
It is not visible on my API Dashboard
Reply • Share ›

Bruno S. Krebs Mod ➚ Sinan • a year ago
Hi there, the AUTH0_API_AUDIENCE is the identifier of your API. The AUTH0_DOMAIN is the domain that you chose when you registered for your account. For example: bkrebs.auth0.com

You can also find domain at the right top corner of the dashboard. Right next to your picture.
Reply • Share ›

Sinan ➚ Bruno S. Krebs • a year ago
Thanks, but it gives me now an error of:
"We can't trust on a token issued by: URL of my API"
Reply • Share ›

Show more replies

jean-luc andrei • a year ago
thanks for this guide.
I have a problem with db connection
_php artisan migrate works well with .env file
_when i tried the API with postman. it seem's it don't use .env file and i have this error