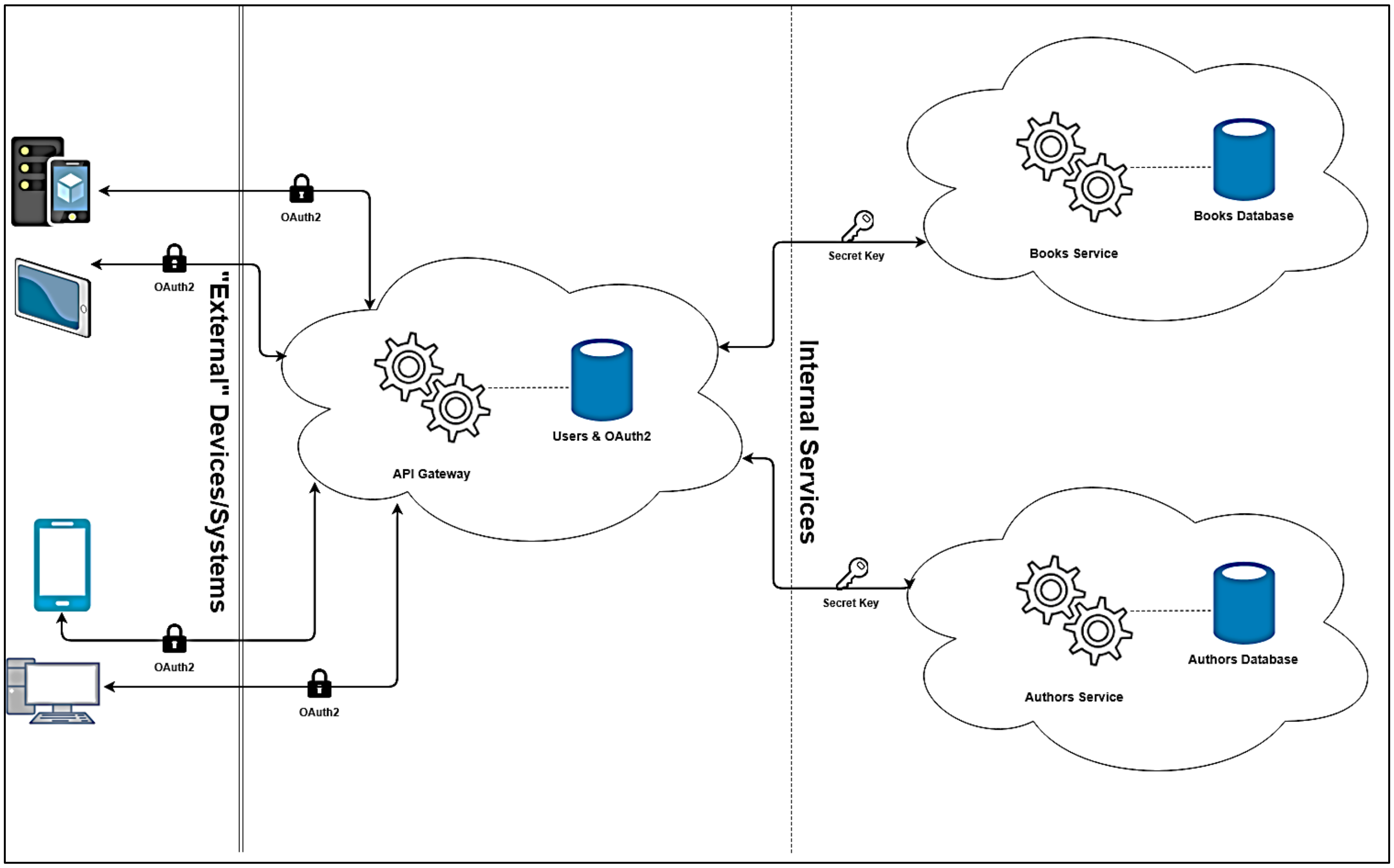
Creating Microservices and API in PHP-Laravel/Lumen

Introduction to Microservices and API



Microservices

Microservices architecture is a method of developing software applications as a suite of small, independent services, each running its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There are a few key aspects to microservices:

- Independence: Each microservice can be developed, deployed, and scaled independently from the others. This allows teams to work on different services simultaneously without impacting the operation of others.
- Specialization: Each service is designed to accomplish a specific task or to manage a certain piece of business logic, leading to a modular and flexible system.
- Technology Diversity: Microservices can be written in different programming languages and use different data storage technologies as long as they can communicate with each other, typically via APIs.

API (Application Programming Interface)

An API is a set of rules and specifications that software programs can follow to communicate with each other. It serves as an interface between different software applications, allowing them to interact without knowing the implementation details of each other. APIs can be used to enable communication between different software components or between microservices in a microservices architecture. Key aspects of APIs include:

- Interface: APIs define a standard way for software components to interact, often through a set of accessible endpoints that perform specific functions.
- Abstraction: APIs abstract the underlying implementation details of a software component, allowing other components or services to use its functionality without understanding how it works internally.
- Integration: APIs are crucial for integrating different software systems or components, whether they are part of the same application or spread across different environments.

Laravel and Lumen
When comparing Laravel and Lumen, it's important to understand that both are PHP frameworks developed by Taylor Otwell and are designed to simplify the development process for web applications. However, they are tailored for different kinds of projects due to their structure, performance, and intended use cases. Here's a detailed comparison:

Laravel is a full-stack PHP framework that's known for its elegant syntax and provides a comprehensive ecosystem for developing web applications. It is one of the most popular PHP frameworks and is designed to make tasks such as authentication, routing, sessions, and caching easier for developers. Laravel comes with a wide range of features that enable rapid application development for complex web applications. Some key features include:
- ✓ Eloquent ORM: An advanced PHP implementation for working with databases using an Object-Relational Mapping (ORM) technique.
- ✓ Artisan Console: A built-in tool for performing repetitive and complex programming tasks that developers avoid doing manually.
- ✓ Blade Templating Engine: A lightweight yet powerful templating engine that allows for the creation of hierarchical layouts and pre-processing of PHP code in views.
- ✓ Middleware Support: Offers a convenient mechanism for filtering HTTP requests entering the application.
- ✓ Comprehensive ecosystem: Includes official packages like Laravel Horizon (for queue management), Laravel Echo (for real-time events), and Laravel Socialite (for OAuth authentication), among others.

Lumen is a micro-framework derived from Laravel, designed for building microservices and high-performance APIs. It's often described as a "lighter" version of Laravel, with some of the functionality stripped out to improve speed and efficiency for tasks that don't require the full Laravel stack. Lumen is particularly suited for services that need to handle thousands of requests per second without incurring significant overhead. Key features include:
- ✓ Lightweight: Strips away many of the features of Laravel that are unnecessary for API development or microservices, resulting in a leaner framework.
- ✓ Fast: Designed for speed, making it one of the fastest PHP frameworks available, ideal for services where response time is critical.
- ✓ Laravel Components: Utilizes many of the same components as Laravel, such as Eloquent ORM and the Blade templating engine, but in a more streamlined form.
- ✓ Easy Upgrade Path to Laravel: Lumen applications can easily be upgraded to the full Laravel framework if the scope of the project expands beyond what Lumen can comfortably handle.

Introduction to MVC Frameworks and PHP Laravel/Lumen

Demo: Laravel MVC

Step 1: Install Laravel
First, you need to install Laravel on your system. You'll need Composer, a PHP dependency manager, to do this. If you don't have Composer installed, download it from getcomposer.org.

Once Composer is installed, run the following command in your terminal to create a new Laravel project named `crud_demo`:

```
composer create-project --prefer-dist laravel/laravel crud_demo
```

Navigate into your project directory:

```
Cd crud_demo
```

Step 2: Configure Environment

Edit the `.env` file in your project root to set up your database connection. This example will use MySQL:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=your_database_name
DB_USERNAME=your_database_username
DB_PASSWORD=your_database_password
```

Step 3: Create a Model and Migration

Run the following command to create a model and a migration file for your CRUD. This example creates a `Post` model:

```
php artisan make:model Post -m
```

Open the created migration file in the `database/migrations` directory, and add some fields to the `posts` table:

```
Schema::create('posts', function (Blueprint $table) {
   $table->id();
   $table->string('title');
   $table->text('body');
   $table->timestamps();
});
```

Run the migration to create the table:

```
php artisan migrate
```

Update the `app\model\post.php`

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
   use HasFactory;
   protected $fillable = ['id', 'title', 'body'];
}
```

Step 4: Create Controller

Generate a controller with resource methods:

```
php artisan make:controller PostController --resource
```

This command creates a controller at `app/Http/Controllers/PostController.php` with methods for handling CRUD operations.

Step 5: Define Routes

Open `routes/web.php` and define routes for your CRUD operations:

```
use App\Http\Controllers\PostController;

Route::resource('posts', PostController::class);
```

Step 6: Implement CRUD Methods in Controller

Edit `PostController.php` to implement methods for CRUD operations. For instance, the `index` method can be implemented as follows:

```
<?php
namespace App\Http\Controllers;
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    public function index()
    {
        $posts = Post::all();
        return view('posts.index', compact('posts'));
    }

    public function create()
    {
        $posts = Post::all();
        return view('posts.create', compact('posts'));
    }

    public function store(Request $request)
    {
        $request->validate([
            'title' => 'required',
            'body' => 'required',
        ]);

        Post::create($request->all());
        return redirect()->route('posts.index')
                        ->with('success','Post created successfully.');
    }

    public function show(string $id)
    {
        $post = Post::findOrFail($id);
        return view('posts.show', compact('post'));
    }

    public function edit(string $id)
    {
        $post = Post::findOrFail($id);
        return view('posts.edit', compact('post'));
    }

    public function update(Request $request, Post $post)
    {
        $request->validate([
            'title' => 'required',
            'body' => 'required',
        ]);

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

        return redirect()->route('posts.index')
                        ->with('success','Post updated successfully');
    }

    public function destroy(string $id)
    {
        $post = Post::findOrFail($id);
        $post->delete();
        return redirect()->route('posts.index')->with('success', 'Post deleted successfully');
    }
}
```

Step 7: Create Views

Create views in the `resources/views/posts` directory for different operations.

`index.blade.php`

```
@foreach ($posts as $post)
    <h2>{{ $post->title }}</h2>
    <p>{{ $post->body }}</p>
    <form action="{{ route('posts.destroy', $post->id) }}" method="POST">
        @csrf
        @method('DELETE')
        <button type="submit">Delete</button>
    </form>
    <a href="{{ route('posts.edit', $post->id) }}" class="btn btn-primary">Edit</a>
@endforeach
```

`edit.blade.php`

```
<form action="{{ route('posts.update', $post->id) }}" method="POST">
    @csrf
    @method('PUT')
    <label for="title">Title:</label>
    <input type="text" name="title" value="{{ $post->title }}" required>
    <label for="body">Body:</label>
    <textarea name="body" required>{{ $post->body }}</textarea>
    <button type="submit">Update Post</button>
</form>
```

`show.blade.php`

```
<h2>{{ $post->title }}</h2>
<p>{{ $post->body }}</p>
```

`create.blade.php`

```
<form action="{{ route('posts.store') }}" method="POST">
   @csrf
   <label for="title">Title:</label>
   <input type="text" name="title" id="title" required>
   <br /><br />
   <label for="body">Body:</label>
   <textarea name="body" id="body" required></textarea>
   <br />
   <button type="submit">Create Post</button>
</form>
```

Step 8: Displaying Success Messages

After creating, updating, or deleting posts, it's good practice to inform the user of their successful action. Modify your views to include session flash messages. For instance, at the top of your `index.blade.php`, you could add:

```
@if ($message = Session::get('success'))
   <div>
      <strong>{{ $message }}</strong>
   </div>
@endif
```

Why Lumen?

Opting for Lumen over Laravel might seem counterintuitive at first, given Laravel's popularity and comprehensive feature set. However, there are specific scenarios where Lumen could be the more suitable choice. Understanding these scenarios involves considering the design philosophy and performance characteristics of both frameworks. Here's a breakdown of reasons why you might opt for Lumen if you already have Laravel:

1. Performance and Speed
   - ✓ High Performance: Lumen is designed to be leaner and faster than Laravel. It's optimized for speed and can handle more requests per second. This makes it an ideal choice for microservices or APIs that need to process a high volume of requests efficiently.
   - ✓ Lightweight: Since Lumen strips away many of the features that make Laravel a full-stack framework, it consumes fewer resources. This can be crucial for applications where performance and response time are critical, such as microservices that are part of a larger distributed system.

2. Microservices Architecture
   - ✓ Suitability for Microservices: Lumen's streamlined nature makes it particularly well-suited for building microservices. If your project involves creating a series of small, loosely coupled services, Lumen's performance benefits and minimalist approach can be advantageous.
   - ✓ Focused API Development: For projects that primarily serve as API backends without the need for the full spectrum of Laravel's features (like views or sessions), Lumen provides just enough functionality to be effective without the overhead.

3. Resource Constraints
   - ✓ Lower Resource Consumption: In environments where computing resources are limited, such as small servers or when optimizing for cost in cloud environments, Lumen's lower resource footprint can make it a more cost-effective option.
   - ✓ Efficiency at Scale: For applications expected to scale out significantly, the incremental resource savings per instance of Lumen can result in substantial overall savings on infrastructure costs.

4. Simplicity and Development Speed
   - ✓ Rapid Development: While both Laravel and Lumen are designed for rapid development, Lumen's simplicity allows developers to quickly set up and deploy microservices or APIs without wading through the functionalities that are unnecessary for backend services.
   - ✓ Ease of Use: Developers familiar with Laravel will find Lumen easy to pick up and use, as it shares many of the same components and principles but in a more streamlined form.

5. Upgrade Path
   - ✓ Easy Upgrades to Laravel: Starting a project with Lumen doesn't lock you into a decision. If your project's requirements grow to necessitate the full range of Laravel's features, you can upgrade from Lumen to Laravel. This provides a path to scale your application complexity without starting from scratch.

Project Making Initial Steps

Note:
We can use ***php artisan serve*** (a small server that comes with Laravel that runs in a local address --127.0.0.1--) along to SQLite that allows putting the entire database in a file called database.sqlite. This approach allows us to stop dealing with multiple dependencies; focus on the main content and keep everything simple.

Note:
To test our API, we will be using Postman: https://www.postman.com/downloads/

Obtaining the Laravel/Lumen structure for the projectsite1 service

[Creating the ***author's service***]

Create the lumen project:

```
composer create-project --prefer-dist laravel/lumen LumenAuthorsApi
composer create-project --prefer-dist laravel/lumen LumenAuthorsApi 5.7.*
```

Navigate inside the project:

```
\> cd  LumenAuthorsApi
```

Open the project in VSCode:

```
\> code .
```

Run the project using a built-in server

```
\> php -S localhost:8000 -t .\public
```

Try accessing the project through the built-in server from the browser

```
localhost:8000
```

*try accessing also using postman (with a GET request)

Preparing the service for its correct operation

1. Enable Facades (for some helper functions) and Eloquent (ORM Database Modeling)

./bootstrap/app.php

```
//uncomment these two lines:
$app->withFacades();
$app->withEloquent();
```

.env

```
# you may use sqlite instead of mysql

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

./database

```
// create a new file called `database.sqlite`
```

.env

```
// http://www.unit-conversion.info/texttools/random-string-generator/
// generate (32 character) )APP_KEY using an online random string generator or Laravel app key generator
// then set it to the APP_KEY property of the .env file
APP_KEY=5wOYleWqkjyxR5EbwPpTtPFU6wFNqPhs
```

.gitignore

```
/vendor
/.idea
Homestead.json
Homestead.yaml
.env
.phpunit.result.cache
database/database.sqlite
```

Building the projectsite1 table with a migration of Laravel/Lumen

Make a database migration

```
\> php artisan make:migration CreateAuthorsTable --create=authors
```

Specify the attributes (columns/fields) of the authors table
./database/migrations/CreateAuthorsTable.php

```
…
   public function up(): void
   {
        Schema::create('authors', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('gender');
            $table->string('country');
            $table->timestamps();
        });
    }
...
```

Rerun all migrations:

```
\> php artisan migrate
```

Creating the projectsite1 model

Go to ./app/model and rename the existing Model class (User.php → Author.php)

```
// make the following changes
<?php
namespace App\Models;
use Illuminate\Auth\Authenticatable;
use Illuminate\Contracts\Auth\Access\Authorizable as AuthorizableContract;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Laravel\Lumen\Auth\Authorizable;

class Author extends Model
{
    use HasFactory;

    protected $fillable = [
        'name',
        'gender',
        'country',
    ];
}
```

Creating a factory for projectsite1 and building test values

./database/factories/AuthorFactory.php

```
<?php
namespace Database\Factories;
use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\Author;

class AuthorFactory extends Factory
{
    protected $model = Author::class;

    public function definition()
    {
        $gender = $this->faker->randomElement(['male', 'female']);

        return [
            'name' => $this->faker->name($gender),
            'gender' => $gender,
            'country' => $this->faker->country,
        ];
    }
}
```

Run the seeder to populate the table using our factory

./database/seeders/DatabaseSeeder.php

```
<?php
namespace Database\Seeders;
use App\Models\Author;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        //factory(Author::class, 30)->create();     //laravel 5.7
        Author::factory()->count(30)->create();     //laravel 8+
    }
}
```

Run the database seeder:

```
\>php artisan db:seed
```

(Optional) Destroy and recreate fresh new migration

```
\>php artisan migrate:fresh --seed
```

Creating the controller for projectsite1

Create a new controller file (or duplicate an existing)
./app/Http/Controllers/AuthorController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Laravel\Lumen\Routing\Controller as BaseController;

class AuthorController extends BaseController
{
    public function index(){
    }

    public function store(Request $request){
    }

    public function show($author){
    }

    public function update(Request $request,$author){
    }

    public function destroy($author){
    }
}
```

Creating the routes for CRUD operations on the projectsite1

./routes/web.php

```
<?php

$router->get('/authors','AuthorController@index');
$router->post('/authors','AuthorController@store');

//using route parameter
$router->get('/authors/{author}','AuthorController@show');
$router->put('/authors/{author}','AuthorController@update');
$router->patch('/authors/{author}','AuthorController@update');
$router->delete('/authors/{author}','AuthorController@destroy');
```

Normalizing the projectsite1' microservice responses

./app/Traits/ApiResponder.php

```
//this is a new file you have to create
<?php
namespace App\Traits;
use Illuminate\Http\Response as HttpResponse;

trait ApiResponder{

    public function successResponse($data,$code=HttpResponse::HTTP_OK){
        return response()->json(['data'=>$data],$code);
    }

    public function errorResponse($message,$code){
        return response()->json(['error'=>$message,'code'=>$code],$code);
    }
}
```

Use the trait in the Controller
./app/Http/Controllers/AuthorController.php

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

use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Laravel\Lumen\Routing\Controller as BaseController;

class AuthorController extends BaseController
{

   use ApiResponder;

…
```

Implementing the functions of the projectsite1 microservice

Showing the list of authors from the Laravel/Lumen controller

./app/Http/Controllers/AuthorController.php

```
   // http://localhost:8000/authors
   // GET
   public function index(){
      $authors = Author::all();
      return $authors;
   }
```

*test using browser and postman

You may then apply our ApiResponder

```
   // http://localhost:8000/authors
   // GET
   public function index(){
      $authors = Author::all();
      return $this->successResponse($authors);
   }
```

Allowing creating author instances from the controller

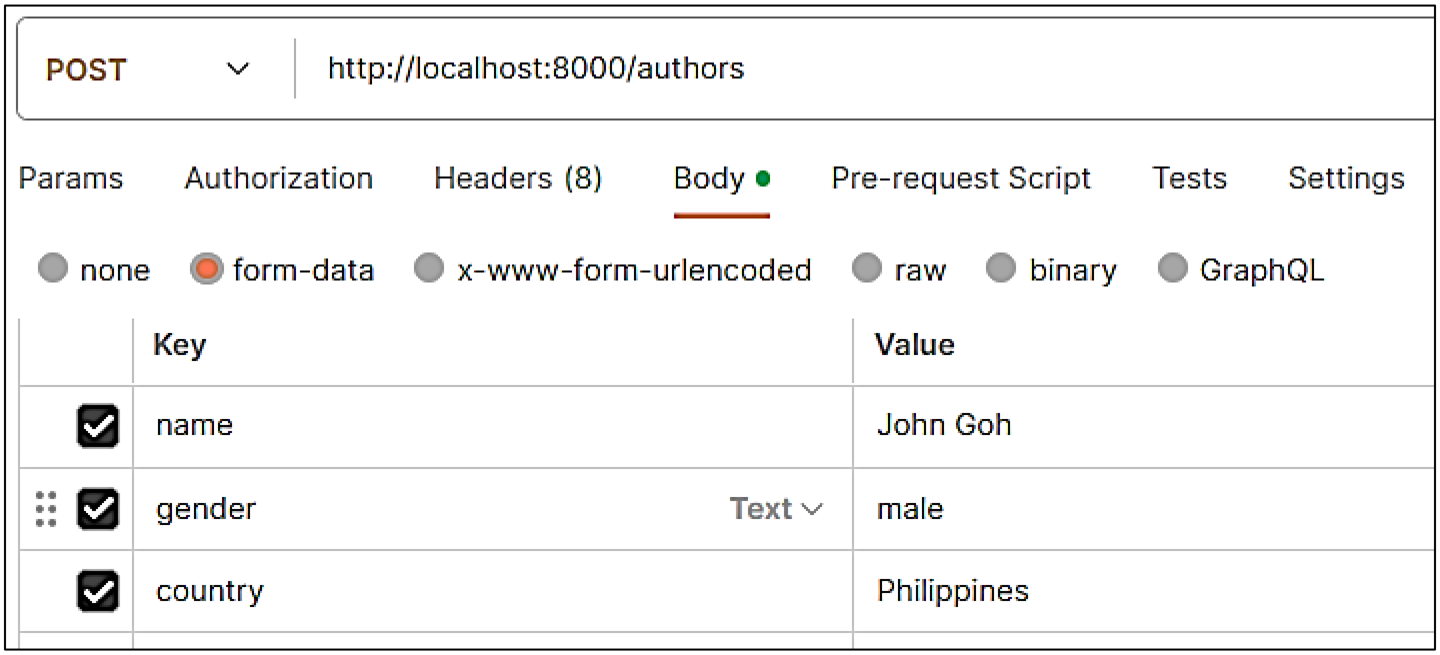
./app/Http/Controllers/AuthorController.php

```
   // http://localhost:8000/authors
   // POST
   public function store(Request $request){
      $rules = [
         'name' => 'required|max:255',
         'gender' => 'required|max:255|in:male,female',
         'country' => 'required|max:255',
      ];
```

```
    $this->validate($request,$rules);
    $author = Author::create($request->all());
    return $this->successResponse($author,Response::HTTP_CREATED);
  }
```

*test using postman with a POST method request without sending any request body values and see that it fails since the values are required.

*now test again using postman but this time with correct values



POST http://localhost:8000/authors

	Key	Value
✓	name	John Goh
✓	gender	male
✓	country	Philippines

<u>Allowing showing an author with a given id with Laravel/Lumen</u>

./app/Http/Controllers/AuthorController.php

```
  // http://localhost:8000/authors/1
  // GET
  public function show($author){
    $author = Author::findOrFail($author);
    return $this->successResponse($author);
  }
```

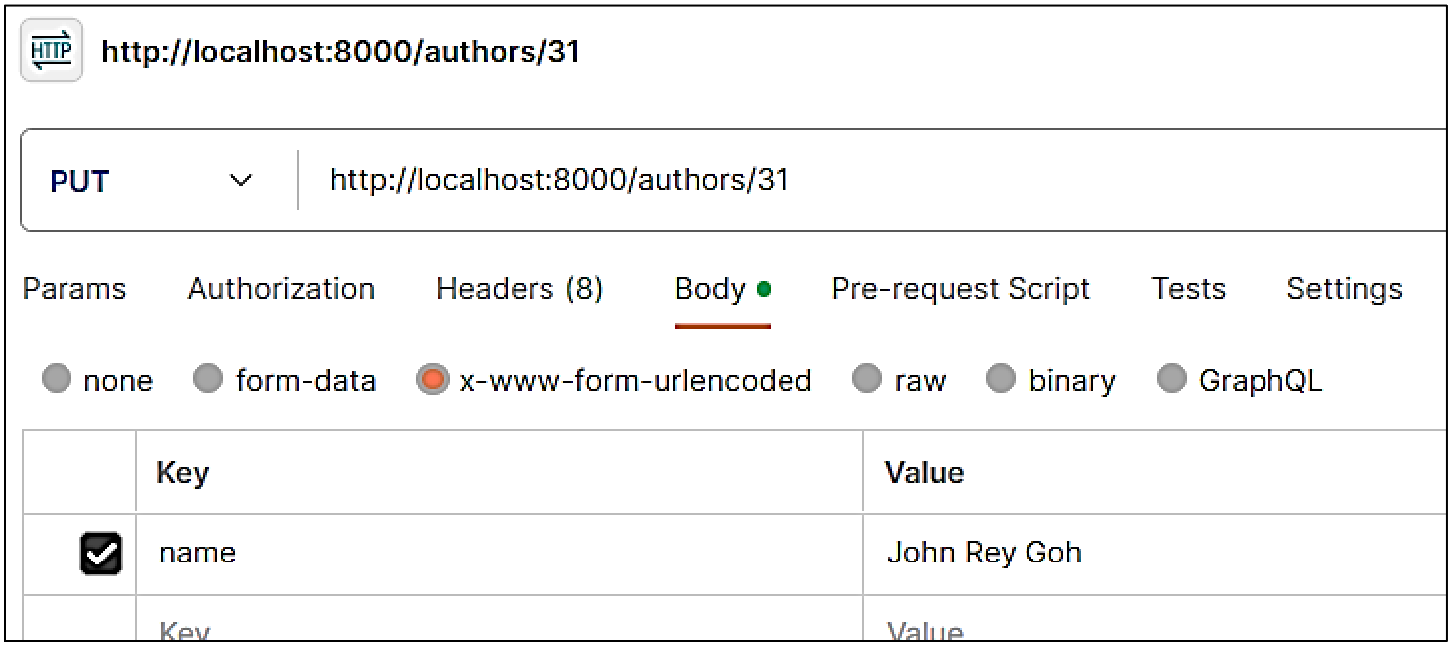
*test using the browser and postman

Allowing editing an existing record

./app/Http/Controllers/AuthorController.php

```
  // http://localhost:8000/authors/1
  // PUT
  public function update(Request $request,$author){
     $rules = [
        'name' => 'max:255',
        'gender' => 'max:255|in:male,female',
        'country' => 'max:255',
     ];

     $this->validate($request,$rules);
     $author = Author::findOrFail($author);
     $author->fill($request->all());

     if($author->isClean()){
        //check if no changes detected
        return $this->errorResponse('No changes detected.',Response::HTTP_UNPROCESSABLE_ENTITY);
     }

     $author->save();
     return $this->successResponse($author,Response::HTTP_CREATED);

  }
```

*you can test this in postman with a PUT/PATCH method request.
*first try sending the request without submitting values to see how the ***isClean()*** method works.
*then try sending another request but this time with values, the values should not be in the body form-data, but rather, it should be in the body **x-www-form-urlencoded**


http://localhost:8000/authors/31

PUT http://localhost:8000/authors/31

Params | Authorization | Headers (8) | Body ● | Pre-request Script | Tests | Settings

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

	Key	Value
✓	name	John Rey Goh
	Key	Value

Allowing deleting an existing record

./app/Http/Controllers/AuthorController.php

```
  // http://localhost:8000/authors/1
  public function destroy($author){
    $author = Author::findOrFail($author);
    $author->delete();
    return $this->successResponse($author);
  }
```

*test in postman using DELETE method request

Handling important errors and exceptions with Laravel/Lumen

./app/Exceptions/Handler.php

```
<?php
namespace App\Exceptions;
use App\Traits\ApiResponder;
use Illuminate\Auth\Access\AuthorizationException;
use Illuminate\Auth\AuthenticationException;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Http\Response;
use Illuminate\Validation\ValidationException;
use Laravel\Lumen\Exceptions\Handler as ExceptionHandler;
use Symfony\Component\HttpKernel\Exception\HttpException;
use Throwable;

class Handler extends ExceptionHandler
{
  use ApiResponder;

  protected $dontReport = [
    AuthorizationException::class,
    HttpException::class,
    ModelNotFoundException::class,
    ValidationException::class,
  ];

  public function report(Throwable $exception)
  {
    parent::report($exception);
  }

  public function render($request, Throwable $exception)
  {
    if($exception instanceof HttpException){
      $code = $exception->getStatusCode();
      $message = Response::$statusTexts[$code];
```

```
        return $this->errorResponse($message,$code);
    }

    if($exception instanceof ModelNotFoundException){
        $model = strtolower(class_basename($exception->getModel()));
        return $this->errorResponse("There are no instances of {$model} with the given ID",
                                                                    Response::HTTP_NOT_FOUND);
    }

    if($exception instanceof AuthorizationException){
        return $this->errorResponse($exception->getMessage(), Response::HTTP_FORBIDDEN);
    }

    if($exception instanceof AuthenticationException){
        return $this->errorResponse($exception->getMessage(), Response::HTTP_UNAUTHORIZED);
    }

    if($exception instanceof ValidationException){
        $errors = $exception->validator->errors()->getMessages();
        return $this->errorResponse($errors,Response::HTTP_UNPROCESSABLE_ENTITY);
    }

    if(env('APP_DEBUG',false)){
        return parent::render($request, $exception);
    }

    return $this->errorResponse('Unexpected Error. Try later.',Response::HTTP_INTERNAL_SERVER_ERROR);

  }
}
```

Extra Section: Create a microservice that can fetch data from another microservice and add it to its own database

To create a Lumen controller function that fetches data from another Lumen API and stores it in its own database, you'll need to use HTTP client for making the API request and Eloquent (or the Query Builder) for database interactions.

Setup HTTP Client

Lumen does not include a default HTTP client. You can use Guzzle, a PHP HTTP client, to make HTTP requests. First, install Guzzle via Composer:

```
composer require guzzlehttp/guzzle
```

Create the Model and Migration

If you haven't already, create a model and corresponding migration for the data you intend to store. For example, if you're storing user information, you might have a `Author` model.

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Author extends Model
{
   use HasFactory;

   protected $fillable = [
      'name',
      'gender',
      'country',
   ];
}
```

Create a migration

```
php artisan make:migration CreateAuthorsTable --create authors
```

In the generated migration file, define the structure of your authors table. Here's an example:

```
      Schema::create('authors', function (Blueprint $table) {
         $table->increments('id');
         $table->string('name');
         $table->string('gender');
         $table->string('country');
         $table->timestamps();
      });
```

Run the migration to update your database schema:

```
php artisan migrate
```

Create the Controller Function

Now, you'll create a controller function that fetches data from another API and stores it. Ensure you have a controller created; if not, you can generate one using:

```
php artisan make:controller DataImportController
```

In your `DataImportController`, add the following function:

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

use Laravel\Lumen\Routing\Controller as BaseController;
use GuzzleHttp\Client;
use App\Models\Author;

class DataImportController extends BaseController
{
    public function ImportData(){

        $client = new Client();
        $response = $client->request('GET','http://localhost:8000/authors');
        $data = json_decode($response->getBody()->getContents(),true);

        foreach($data['data'] as $item){
            Author::updateOrCreate(
                ['name' => $item['name']],
                [
                    'gender' => $item['gender'],
                    'country' => $item['country'],
                ]
            );
        }

        return response()->json(['Message' => 'Data Import OK'],200);

    }
}
```

Define a Route

Finally, define a route in your `routes/web.php` or `routes/api.php` that points to this controller action. For example:

```
$router->get('/import','DataImportController@ImportData');
```

Extra Section:
Combine the fetch result array from multiple microservice function

In the controller, create a function that can consolidate data fetched from multiple apis

```
public function consolidate(){

    $authors1 = Author::all();

    $client = new Client();
    $response = $client->request('GET','http://localhost:8000/authors');
    $authors2 = json_decode($response->getBody()->getContents(),true);

    //Simple Merge
    //If you simply want to append one collection to another,
    //you can use the merge method. This method is particularly useful
    //when you don't need to worry about duplicating keys or overwriting
    //values.
    //$combinedCollection = $authors1->merge($authors2);

    //Merge With Unique Values
    //If you want to ensure that the combined collection contains unique values,
    //you might first merge the collections and then filter out duplicates.
    //$combinedUniqueCollection = $authors1->merge($authors2)->unique('id');

    //Using Laravel Collections' concat Method
    //Another straightforward approach for combining two collections without worrying
    //about keys is to use the concat method. Unlike merge, concat doesn’t change the
    //original collections.
    $combinedCollection = $authors1->concat($authors2);

    return $combinedCollection;
}
```

And the route in ./routes/web.php

```
$router->get('/consolidate','DataImportController@Consolidate');
```

Extra Section:
How to run a microservice function directly

1. Using Artisan Command

If you're looking to trigger controller actions as part of your application's CLI (Command Line Interface) operations or scheduled tasks, you can define a custom Artisan command in Lumen. Then, within this command, you can instantiate the controller and call its methods directly.

1. Define a Custom Command: Create a new command within the `app/Console/Commands` directory. If this directory doesn't exist, you may need to create it.

2. Call Controller Method: In your command's handle method, instantiate the controller and call the function you wish to run.

```
<?php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use App\Http\Controllers\DataImportController;

class Testimport extends Command
{
   protected $signature = 'go:import';

   protected $description = 'Description of what this command does';

   public function handle()
   {
      $controller = new DataImportController();
      $controller->ImportData(); // Call your function here
   }
}
```

3. Register Your Command: Make sure to register your command in `app/Console/Kernel.php` so it's recognized by the Artisan CLI.

```
protected $commands = [
   \App\Console\Commands\Testimport::class,
];
```

4. Run Your Command: Finally, you can run your command using the Artisan CLI.

```
php artisan go:import
```

2. Using Routes for Local Testing

If your goal is to test or run the controller method locally without a front-end, you can simply define a route in your Lumen application that points to the controller action. You can then trigger this route using a tool like Postman or even a browser, if appropriate.

```
<?php
// Within your routes/web.php or routes/api.php
$router->get('/test-controller', 'YourController@yourFunction');
```

This way, you can directly access the function by navigating to `http://your-lumen-app.test/test-controller` or whatever your local development URL is.

3. Scheduled Tasks

If the function needs to be executed periodically, you can use Lumen's task scheduling capabilities. Define a scheduled task in `app/Console/Kernel.php` that invokes the controller method.

Lumen leverages the Cron scheduling service, so you'll need to have Cron running and configured to call the `php artisan schedule:run` command at your desired intervals.

```
<?php
// Inside app/Console/Kernel.php
```

```
protected function schedule(Schedule $schedule)
{
   $schedule->call(function () {
      $controller = new YourController();
      $controller->yourFunction();
   })->everyMinute(); // Adjust the timing as needed
}
```

You may then start the scheduled task by:

```
php artisan schedule:work
php artisan schedule:run
```

Creating the microservice

Getting the structure of Laravel/Lumen for the projectsite2 microservice

[Creating the ***book's service***]

Create the lumen project:

```
composer create-project --prefer-dist laravel/lumen LumenBooksApi
composer create-project --prefer-dist laravel/lumen LumenBooksApi 5.7.*
```

Navigate inside the project:

```
\> cd  LumenBooksApi
```

Open the project in VSCode:

```
\> code .
```

Run the project using a built-in server

```
\> php -S localhost:8001 -t .\public
```

Try accessing the project through the built-in server from the browser

```
localhost:8001
```

*try accessing also using postman (with a GET request)

Preparing the projectsite2 microservice

1. Enable Facades (for some helper functions) and Eloquent (ORM Database Modeling)

./bootstrap/app.php

```
//uncomment these two lines:
$app->withFacades();
$app->withEloquent();
```

.env

```
# you may use sqlite instead of mysql
```

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

./database

```
// create a new file called `database.sqlite`
```

.env

```
// http://www.unit-conversion.info/texttools/random-string-generator/
// generate (32 character) )APP_KEY using an online random string generator or Laravel app key generator
// then set it to the APP_KEY property of the .env file
APP_KEY=5wOYleWqkjyxR5EbwPpTtPFU6wFNqPhs
```

.gitignore

```
/vendor
/.idea
Homestead.json
Homestead.yaml
.env
.phpunit.result.cache
database/database.sqlite
```

__Creating the table for projectsite2 with migrations__

Make a database migration

```
\> php artisan make:migration CreateBooksTable --create=books
```

Specify the attributes (columns/fields) of the authors table

./database/migrations/CreateBooksTable.php

```
…
   public function up(): void
   {
      Schema::create('books', function (Blueprint $table) {
         $table->increments('id');
         $table->string('title');
         $table->string('description');
         $table->integer('price')->unsigned();
         $table->integer('authors_id')->unsigned();
         $table->timestamps();
      });
   }
…
```

Rerun all migrations:

```
\> php artisan migrate
```

Creating the model for projectsite2

Go to ./app/model and rename the existing Model class (User.php → Book.php)

```
// make the following changes
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    use HasFactory;

    protected $fillable = [
        'title',
        'description',
        'price',
        'authors_id',
    ];
}
```

Creating a factory for projectsite2 and generating random values

./database/factories/BookFactory.php

```
<?php
namespace Database\Factories;

use App\Models\Book;
use Illuminate\Database\Eloquent\Factories\Factory;

class BookFactory extends Factory
{

    protected $model = Book::class;

    public function definition()
    {
        return [
            'title' => $this->faker->sentence(3,true),
            'description' => $this->faker->sentence(6,true),
            'price' => $this->faker->numberBetween(25,150),
            'authors_id' => $this->faker->numberBetween(1,30), //how many authors do you have?
        ];
    }
}
```

Run the seeder to populate the table using our factory
./database/seeders/DatabaseSeeder.php

```
<?php
namespace Database\Seeders;
use App\Models\Book;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    public function run()
    {
        //factory(Book::class, 30)->create();    //laravel 5.7
        Book::factory()->count(30)->create();    //laravel 8+
    }
}
```

Run the database seeder:

```
\>php artisan db:seed
```

(Optional) Destroy and recreate fresh new migration

```
\>php artisan migrate:fresh --seed
```

Creating the Laravel/Lumen controller for projectsite2

Create a new controller file (or duplicate an existing)
./app/Http/Controllers/BookController.php

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Laravel\Lumen\Routing\Controller as BaseController;

class BookController extends BaseController
{
    public function index(){
    }

    public function store(Request $request){
    }

    public function show($book){
    }

    public function update(Request $request,$book){
    }

    public function destroy($book){
    }
}
```

Creating the Laravel/Lumen routes for CRUD operations on projectsite2

./routes/web.php

```
<?php

$router->get('/books’,'BookController@index');
$router->post('/books','BookController@store');

//using route parameter
$router->get('/books/{book}','BookController@show');
$router->put('/books/{book}','BookController@update');
$router->patch('/books/{book}','BookController@update');
$router->delete('/books/{book}','BookController@destroy');
```

Standardizing the responses across the services

./app/Traits/ApiResponder.php

```
//this is a new file you have to create
<?php
namespace App\Traits;
use Illuminate\Http\Response as HttpResponse;

trait ApiResponder{

    public function successResponse($data,$code=HttpResponse::HTTP_OK){
        return response()->json(['data'=>$data],$code);
    }

    public function errorResponse($message,$code){
        return response()->json(['error'=>$message,'code'=>$code],$code);
    }
}
```

Use the trait in the Controller
./app/Http/Controllers/BookController.php

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

use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Laravel\Lumen\Routing\Controller as BaseController;

class BookController extends BaseController
{
    use ApiResponder;

...
```

Implementing the functions of the projectsite2s microservice

Showing the complete list of projectsite2

./app/Http/Controllers/BookController.php

```
    // GET
    // http://localhost:8000/books
    public function index(){
       $authors = Book::all();
       return $this->successResponse($authors);
    }
```

Allowing creating new instances of projectsite2

./app/Http/Controllers/BookController.php

```
    // POST
    // http://localhost:8000/books
    public function store(Request $request){
       $rules = [
          'title' => 'required|max:255',
          'description' => 'required|max:255',
          'price' => 'required|min:1',
          'authors_id' => 'required|min:1',
       ];

       $this->validate($request,$rules);
       $book = Book::create($request->all());
       return $this->successResponse($book,Response::HTTP_CREATED);
    }
```

Allowing showing a specific projectsite2

./app/Http/Controllers/BookController.php

```
    // GET
    // http://localhost:8000/books/1
    public function show($book){
       $book = Book::findOrFail($book);
       return $this->successResponse($book);
    }
```

Allowing updating an existing projectsite2

./app/Http/Controllers/BookController.php

```
    // PUT
```

```
  // http://localhost:8000/books/1
  public function update(Request $request,$book){
    $rules = [
      'title' => 'max:255',
      'description' => 'max:255',
      'price' => 'min:1',
      'authors_id' => 'min:1',
    ];

    $this->validate($request,$rules);
    $book = Book::findOrFail($book);
    $book->fill($request->all());

    if($book->isClean()){
      //check if no changes detected
      return $this->errorResponse('No changes detected.',Response::HTTP_UNPROCESSABLE_ENTITY);
    }

    $book->save();
    return $this->successResponse($book,Response::HTTP_CREATED);

  }
```

Allowing removing an existing projectsite2

./app/Http/Controllers/BookController.php

```
  // DELETE
  // http://localhost:8000/books/1
  public function destroy($book){
    $book = Book::findOrFail($book);
    $book->delete();
    return $this->successResponse($book);
  }
```

Handling relevant errors and exceptions with Laravel/Lumen

./app/Exceptions/Handler.php

```
<?php
namespace App\Exceptions;

use App\Traits\ApiResponder;
use Illuminate\Auth\Access\AuthorizationException;
use Illuminate\Auth\AuthenticationException;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Http\Response;
use Illuminate\Validation\ValidationException;
use Laravel\Lumen\Exceptions\Handler as ExceptionHandler;
use Symfony\Component\HttpKernel\Exception\HttpException;
use Throwable;

class Handler extends ExceptionHandler
{
   use ApiResponder;

   protected $dontReport = [
      AuthorizationException::class,
      HttpException::class,
      ModelNotFoundException::class,
      ValidationException::class,
   ];

   public function report(Throwable $exception)
   {
      parent::report($exception);
   }

   public function render($request, Throwable $exception)
   {
      if($exception instanceof HttpException){
         $code = $exception->getStatusCode();
         $message = Response::$statusTexts[$code];
         return $this->errorResponse($message,$code);
      }

      if($exception instanceof ModelNotFoundException){
         $model = strtolower(class_basename($exception->getModel()));
         return $this->errorResponse("There are no instances of {$model} with the given ID",
                                                                        Response::HTTP_NOT_FOUND);
      }

      if($exception instanceof AuthorizationException){
         return $this->errorResponse($exception->getMessage(), Response::HTTP_FORBIDDEN);
      }

      if($exception instanceof AuthenticationException){
         return $this->errorResponse($exception->getMessage(), Response::HTTP_UNAUTHORIZED);
      }

      if($exception instanceof ValidationException){
         $errors = $exception->validator->errors()->getMessages();
         return $this->errorResponse($errors,Response::HTTP_UNPROCESSABLE_ENTITY);
      }

      if(env('APP_DEBUG',false)){
         return parent::render($request, $exception);
      }

      return $this->errorResponse('Unexpected Error. Try later.',Response::HTTP_INTERNAL_SERVER_ERROR);
```

```
    }
}
```

Creating and preparing the API Gateway with Laravel/Lumen

Creating the Laravel/Lumen project for the API Gateway using Composer

[Creating the ***API Gateway***]

Create the lumen project:

```
composer create-project --prefer-dist laravel/lumen LumenApiGateway
composer create-project --prefer-dist laravel/lumen LumenApiGateway 5.7.*
```

Navigate inside the project:

```
\> cd  LumenApiGateway
```

Open the project in VSCode:

```
\> code .
```

Run the project using a built-in server

```
\> php -S localhost:8002 -t .\public
```

Try accessing the project through the built-in server from the browser

```
localhost:8002
```

*try accessing also using postman (with a GET request)

Preparing Laravel/Lumen for the API Gateway

1. Enable Facades (for some helper functions) and Eloquent (ORM Database Modeling)

./bootstrap/app.php

```
//uncomment these two lines:
$app->withFacades();
$app->withEloquent();
```

.env

```
# you may use sqlite instead of mysql

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

./database

```
// create a new file called `database.sqlite`
```

.env

```
// http://www.unit-conversion.info/texttools/random-string-generator/
// generate (32 character) )APP_KEY using an online random string generator or Laravel app key generator
// then set it to the APP_KEY property of the .env file
APP_KEY=5wOYleWqkjyxR5EbwPpTtPFU6wFNqPhs
```

.gitignore

```
/vendor
/.idea
Homestead.json
Homestead.yaml
.env
.phpunit.result.cache
database/database.sqlite
```

Creating the controllers for the projectsite1 and projectsite2 Laravel/Lumen microservices

We will create two(2) controllers (for the Authors Microservice and the Books Microservice) in the API Gateway:

./app/Http/Controllers/AuthorController.php

```
//we will be using the ExampleController.php as the primary template
<?php

namespace App\Http\Controllers;

use App\Traits\ApiResponder;
use Illuminate\Http\Request;

class AuthorController extends Controller
{

   use ApiResponder;

   public function __construct(){
   }

   public function index(){
   }

   public function store(Request $request){
   }

   public function show($author){
   }

   public function update(Request $request,$author){
   }

   public function destroy($author){
    }
}
```

./app/Http/Controllers/BookController.php

```
//we will be using the ExampleController.php as the primary template
<?php

namespace App\Http\Controllers;

use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Illuminate\Http\Response;

class BookController extends Controller
{
    use ApiResponder;

    public function __construct()
    {
        //
    }

    public function index(){
    }

    public function store(Request $request){
    }

    public function show($book){
    }

    public function update(Request $request,$book){
    }

    public function destroy($book){
    }
}
```

Unifying Laravel/Lumen responses for the API Gateway

./app/Traits/ApiResponder.php

```
//this is a new file you have to create
<?php
namespace App\Traits;
use Illuminate\Http\Response as HttpResponse;

trait ApiResponder{

  public function successResponse($data,$code=HttpResponse::HTTP_OK){
     return response($data,$code)->header('Content-Type','application/json');
  }

  public function errorResponse($message,$code){
     return response()->json(['error'=>$message,'code'=>$code],$code);
  }

  public function errorMessage($message,$code){
     return response($message,$code)->header('Content-Type','application/json');
  }

}
```

Use the trait in the Controllers present in the API Gateway
./app/Http/Controllers/AuthorController.php

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

use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Laravel\Lumen\Routing\Controller as BaseController;

class AuthorController extends BaseController
{

  use ApiResponder;

…
```

./app/Http/Controllers/BookController.php

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

use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Laravel\Lumen\Routing\Controller as BaseController;

class BookController extends BaseController
{

  use ApiResponder;
…
```

Registering routes for microservices in Laravel/Lumen from the Gateway

./routes/web.php

```
<?php
```

```
$router->get('/authors','AuthorController@index');
$router->post('/authors','AuthorController@store');
$router->get('/authors/{author}','AuthorController@show');
$router->put('/authors/{author}','AuthorController@update');
$router->patch('/authors/{author}','AuthorController@update');
$router->delete('/authors/{author}','AuthorController@destroy');

$router->get('/books','BookController@index');
$router->post('/books','BookController@store');
$router->get('/books/{book}','BookController@show');
$router->put('/books/{book}','BookController@update');
$router->patch('/books/{book}','BookController@update');
$router->delete('/books/{book}','BookController@destroy');
```

Preparing the Gateway in Laravel/Lumen to consume services

Use composer to download a package that allows our API to make HttpRequests similar to an Http Client

```
composer require guzzlehttp/guzzle
```

create a configuration folder and file for services to identify an array of components:
./config/services.php

```
<?php

return  [

];
```

Make sure lumen will load the configuration file
./bootstrap/app.php

```
…
$app->withFacades();
$app->withEloquent();
$app->configure('services');
…
```

Create a trait that allows the use of guzzle to consume other web services
./app/Traits/ConsumesExternalService.php

```
<?php

namespace App\Traits;

use GuzzleHttp\Client;

trait ConsumesExternalService{

   public function performRequest($method,$requestUrl,$formParams=[],$headers=[]){
      $client = new Client([
         'base_uri' => $this->baseUri,
      ]);
```

```
        $response = $client->request($method, $requestUrl, ['form_params'=>$formParams,'headers'=>$headers]);
        return $response->getBody()->getContents();
    }

}
```

Preparing the Laravel/Lumen components to consume the internal services

Create the file:
./config/services.php

```
<?php

return  [
    'authors' => [
        'base_uri' => env('AUTHORS_SERVICE_BASE_URL'),
    ],

    'books' => [
        'base_uri' => env('BOOKS_SERVICE_BASE_URL'),
    ],
];
```

.env

```
…
AUTHORS_SERVICE_BASE_URL=localhost:8000
BOOKS_SERVICE_BASE_URL=localhost:8001
```

Create the ff files:
./app/Services/AuthorService.php

```
<?php
namespace App\Services;

use App\Traits\ConsumesExternalService;

class AuthorService{
    use ConsumesExternalService;
    public $baseUri;

    public function __construct()
    {
        $this->baseUri =  config('services.authors.base_uri');
    }
}
```

./app/Services/BookService.php

```
<?php
namespace App\Services;

use App\Traits\ConsumesExternalService;
```

```
class BookService{
   use ConsumesExternalService;
   public $baseUri;

   public function __construct()
   {
      $this->baseUri =  config('services.books.base_uri');
   }
}
```

Let our controllers use the services:

./app/Http/Controllers/AuthorController.php

```
…
class AuthorController extends Controller
{

   use ApiResponder;

   public $authorService;

   public function __construct(AuthorService $authorService)
   {
      $this->authorService = $authorService;
   }
…
```

./app/Http/Controllers/BookController.php

```
…
class BookController extends Controller
{
   use ApiResponder;

   public $bookService;

   public function __construct(BookService $bookService)
   {
      $this->bookService = $bookService;
   }
…
```

Implementing the functions of the Gateway with Laravel/Lumen

Obtaining the list of records from the projectsite1 Laravel/Lumen microservice

./app/Services/AuthorService.php

```
public function obtainAuthors(){
   return $this->performRequest('GET','/authors');
}
```

./app/Http/Controllers/AuthorController.php

```
// GET
// http://localhost:8002/authors
public function index(){
   return $this->successResponse($this->authorService->obtainAuthors());

}
```

Creating a record instance with the projectsite1 service

./app/Services/AuthorService.php

```
public function createAuthor($data)
{
   return $this->performRequest('POST', '/authors', $data);
}
```

./app/Http/Controllers/AuthorController.php

```
// POST
// http://localhost:8002/authors
public function store(Request $request)
{
   return $this->successResponse($this->authorService->createAuthor($request->all(), Response::HTTP_CREATED));
}
```

Showing a record instance using the projectsite1 Laravel/Lumen microservice

./app/Services/AuthorService.php

```
public function obtainAuthor($author)
{
   return $this->performRequest('GET', "/authors/{$author}");
}
```

./app/Http/Controllers/AuthorController.php

```
// GET
// http://localhost:8002/authors/1
public function show($author)
{
   return $this->successResponse($this->authorService->obtainAuthor($author));
}
```

Editing record instances using the projectsite1 service

./app/Services/AuthorService.php

```
    public function editAuthor($data, $author)
    {
        return $this->performRequest('PUT', "/authors/{$author}", $data);
    }
```

./app/Http/Controllers/AuthorController.php

```
    // PUT
    // http://localhost:8002/authors/1
    public function update(Request $request, $author)
    {
        return $this->successResponse($this->authorService->editAuthor($request->all(), $author));
    }
```

Deleting record instances using the projectsite1 Laravel/Lumen service

./app/Services/AuthorService.php

```
    public function deleteAuthor($author)
    {
        return $this->performRequest('DELETE', "/authors/{$author}");
    }
```

./app/Http/Controllers/AuthorController.php

```
    // DELETE
    // http://localhost:8002/authors/1
    public function destroy($author)
    {
        return $this->successResponse($this->authorService->deleteAuthor($author));
    }
```

Implementing operations for projectsite2s based on the projectsite1

./app/Services/BookService.php

```
<?php

namespace App\Services;

use App\Traits\ConsumesExternalService;

class BookService{

    use ConsumesExternalService;

    public $baseUri;

    public function __construct()
    {
        $this->baseUri =  config('services.books.base_uri');
    }

    public function obtainBooks(){
        return $this->performRequest('GET','/books');
    }

    public function createBook($data)
    {
        return $this->performRequest('POST', '/books', $data);
    }

    public function obtainBook($book)
    {
        return $this->performRequest('GET', "/books/{$book}");
    }

    public function editBook($data, $book)
    {
        return $this->performRequest('PUT', "/books/{$book}", $data);
    }

    public function deleteBook($book)
    {
        return $this->performRequest('DELETE', "/books/{$book}");
    }

}
```

./app/Http/Controllers/BookController.php

```
<?php

namespace App\Http\Controllers;

use App\Services\BookService;
use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Illuminate\Http\Response;


class BookController extends Controller
{

    use ApiResponder;

    public $bookService;

    public function __construct(BookService $bookService)
    {
        $this->bookService = $bookService;
    }

    // GET
    // http://localhost:8002/books
    public function index(){
        return $this->successResponse($this->bookService->obtainBooks());
    }

    // POST
    // http://localhost:8002/books
    public function store(Request $request)
    {
        return $this->successResponse($this->bookService->createBook($request->all(), Response::HTTP_CREATED));
    }


    // GET
    // http://localhost:8002/books/1
    public function show($book)
    {
        return $this->successResponse($this->bookService->obtainBook($book));
    }

    // PUT
    // http://localhost:8002/books/1
    public function update(Request $request, $book)
    {
        return $this->successResponse($this->bookService->editBook($request->all(), $book));
    }

    // DELETE
    // http://localhost:8002/books/1
    public function destroy($book)
    {
        return $this->successResponse($this->bookService->deleteBook($book));
    }
}
```

Checking the existence of the record before creating a projectsite2

./app/Http/Controllers/BookController.php

```
//modify the BookController.php of your ApiGateway
<?php

namespace App\Http\Controllers;

use App\Services\BookService;
use App\Services\AuthorService;
use App\Traits\ApiResponder;
use Illuminate\Http\Request;
use Illuminate\Http\Response;

class BookController extends Controller
{

    use ApiResponder;

    public $bookService;
    public $authorService;

    public function __construct(BookService $bookService,AuthorService $authorService)
    {
        $this->bookService = $bookService;
        $this->authorService = $authorService;
    }

    // GET
    // http://localhost:8002/books
    public function index(){
        return $this->successResponse($this->bookService->obtainBooks());
    }

    // POST
    // http://localhost:8002/books
    public function store(Request $request)
    {
        $this->authorService->obtainAuthor($request->author_id);
        return $this->successResponse($this->bookService->createBook($request->all(), Response::HTTP_CREATED));
    }


    // GET
    // http://localhost:8002/books/1
    public function show($book)
    {
        return $this->successResponse($this->bookService->obtainBook($book));
    }

    // PUT
    // http://localhost:8002/books/1
    public function update(Request $request, $book)
    {
        return $this->successResponse($this->bookService->editBook($request->all(), $book));
    }

    // DELETE
    // http://localhost:8002/books/1
    public function destroy($book)
    {
        return $this->successResponse($this->bookService->deleteBook($book));
    }
}
```

Controlling errors obtained from services

./app/Exceptions/Handler.php

```
<?php

namespace App\Exceptions;

use App\Traits\ApiResponder;
use GuzzleHttp\Exception\ClientException;
use Illuminate\Auth\Access\AuthorizationException;
use Illuminate\Auth\AuthenticationException;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Validation\ValidationException;
use Laravel\Lumen\Exceptions\Handler as ExceptionHandler;
use Symfony\Component\HttpKernel\Exception\HttpException;
use Illuminate\Http\Response;
use Throwable;

class Handler extends ExceptionHandler
{
    use ApiResponder;

    protected $dontReport = [
        AuthorizationException::class,
        HttpException::class,
        ModelNotFoundException::class,
        ValidationException::class,
    ];

    public function report(Throwable $exception)
    {
        parent::report($exception);
    }

    public function render($request, Throwable $exception)
    {
        if ($exception instanceof HttpException) {
            $code = $exception->getStatusCode();
            $message = Response::$statusTexts[$code];

            return $this->errorResponse($message, $code);
        }

        if ($exception instanceof ModelNotFoundException) {
            $model = strtolower(class_basename($exception->getModel()));
            return $this->errorResponse("Does not exist any instance of {$model} with the given id",
Response::HTTP_NOT_FOUND);
        }

        if ($exception instanceof AuthorizationException) {
            return $this->errorResponse($exception->getMessage(), Response::HTTP_FORBIDDEN);
        }

        if ($exception instanceof AuthenticationException) {
            return $this->errorResponse($exception->getMessage(), Response::HTTP_UNAUTHORIZED);
        }

        if ($exception instanceof ValidationException) {
            $errors = $exception->validator->errors()->getMessages();

            return $this->errorResponse($errors, Response::HTTP_UNPROCESSABLE_ENTITY);
        }

        if ($exception instanceof ClientException) {
            $message = $exception->getResponse()->getBody();
            $code = $exception->getCode();

            return $this->errorMessage($message, $code);
        }

        if (env('APP_DEBUG', false)) {
            return parent::render($request, $exception);
        }

        return $this->errorResponse('Unexpected error. Try later', Response::HTTP_INTERNAL_SERVER_ERROR);

    }
}
```

Implementing the security layer of the microservices architecture with Laravel/Lumen

Installing and enabling Laravel/Lumen Passport components

*Lumen doesn't have a built-in passport package. We will use:
https://github.com/dusterio/lumen-passport

```
composer require dusterio/lumen-passport
```

*lumen passport may not work well with Lumen 9 or 10 as of March 2024, here is my composer.json require:

```
    "require": {
        "php": ">=7.3",
        "dusterio/lumen-passport": "^0.3.8",
        "guzzlehttp/guzzle": "^7",
        "laravel/lumen-framework": "9.0.*",
        "vlucas/phpdotenv": "^5.4.1",
    "laravel/passport": "^10.4"
  },
  "require-dev": {
    "fakerphp/faker": "^1.9.1",
    "mockery/mockery": "^1.4.4",
    "phpunit/phpunit": "^8.0"
  },
```

*note: you may resort to downloading your lumen version for some reason, here are the steps:

1) Backup your project
2) Change the value of Laravel/Lumen version inside the composer.json
3) Delete vendor folder
4) Run composer install

./bootstrap/app.php

```
…
$app->routeMiddleware([
  'client.credentials' => Laravel\Passport\Http\Middleware\CheckClientCredentials::class,
]);
$app->register(Laravel\Passport\PassportServiceProvider::class);
$app->register(Dusterio\LumenPassport\PassportServiceProvider::class);
$app->configure('auth');
…
```

./app/User.php

```
<?php

namespace App;

use Illuminate\Auth\Authenticatable;
use Illuminate\Contracts\Auth\Access\Authorizable as AuthorizableContract;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Database\Eloquent\Model;
use Laravel\Lumen\Auth\Authorizable;
use Laravel\Passport\HasApiTokens;

class User extends Model implements AuthenticatableContract, AuthorizableContract
{
  use Authenticatable, Authorizable, HasApiTokens;

  protected $fillable = [
    'name', 'email', 'password',
  ];

  protected $hidden = [
    'password',
  ];
}
```

./config/auth.php

```
<?php

return [

   'defaults' => [
      'guard' => env('AUTH_GUARD', 'api'),
   ],

   'guards' => [
      'api' => [
         'driver' => 'passport',
         'provider' => 'users',
      ],
   ],

   'providers' => [
      'users' => [
         'driver' => 'eloquent',
         'model' => App\User::class,
      ],
   ],

   'passwords' => [
      //
   ],

];
```

Preparing and configuring Laravel/Lumen to use Passport

Installing the passport adds oauth migration so we need to run:

```
\> php artisan migrate
```

Install the passport and create some clients and keys:

```
\> php artisan passport:install
```

Sample output:

```
C:\Users\core360\Desktop\lumen\LumenApiGateway>php artisan passport:install
Encryption keys already exist. Use the --force option to overwrite them.
Personal access client created successfully.
Client ID: 1
Client secret: gkg8H157eggi4XrYDtM9In2aztGwKxs3Ps2aFpWC
Password grant client created successfully.
Client ID: 2
Client secret: lqvrHSrL6XBfTwOTjyYbPKsGopawtFf2TSKZYhlV
```

If having problems running passport:install, try running:

```
\> php artisan migrate:fresh
```

*note: You should see oauth public and private keys in ./storage

./app/Providers/AuthServiceProvider.php

```
<?php

namespace App\Providers;

use Dusterio\LumenPassport\LumenPassport;
use Illuminate\Support\ServiceProvider;

class AuthServiceProvider extends ServiceProvider
{

    public function register()
    {
        //
    }

    public function boot()
    {
        LumenPassport::routes($this->app->router);
    }
}
```

<u>Protecting the Gateway routes with Laravel/Lumen Passport</u>

Enable and modify middleware to use authentication
./bootstrap/app.php

```
…
$app->routeMiddleware([
    'auth' => App\Http\Middleware\Authenticate::class,
    'client.credentials' => Laravel\Passport\Http\Middleware\CheckClientCredentials::class,
]);
…
```

./routes/web.php

```
<?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.
```

```
|
*/

$router->group(['middleware' => 'client.credentials'], function () use ($router) {
   /
    * Routes for authors
    */
   $router->get('/authors', 'AuthorController@index');
   $router->post('/authors', 'AuthorController@store');
   $router->get('/authors/{author}', 'AuthorController@show');
   $router->put('/authors/{author}', 'AuthorController@update');
   $router->patch('/authors/{author}', 'AuthorController@update');
   $router->delete('/authors/{author}', 'AuthorController@destroy');

   /
    * Routes for books
    */
   $router->get('/books', 'BookController@index');
   $router->post('/books', 'BookController@store');
   $router->get('/books/{book}', 'BookController@show');
   $router->put('/books/{book}', 'BookController@update');
   $router->patch('/books/{book}', 'BookController@update');
   $router->delete('/books/{book}', 'BookController@destroy');

});
```

*try accessing the routes and see that you are not authenticated to do so.

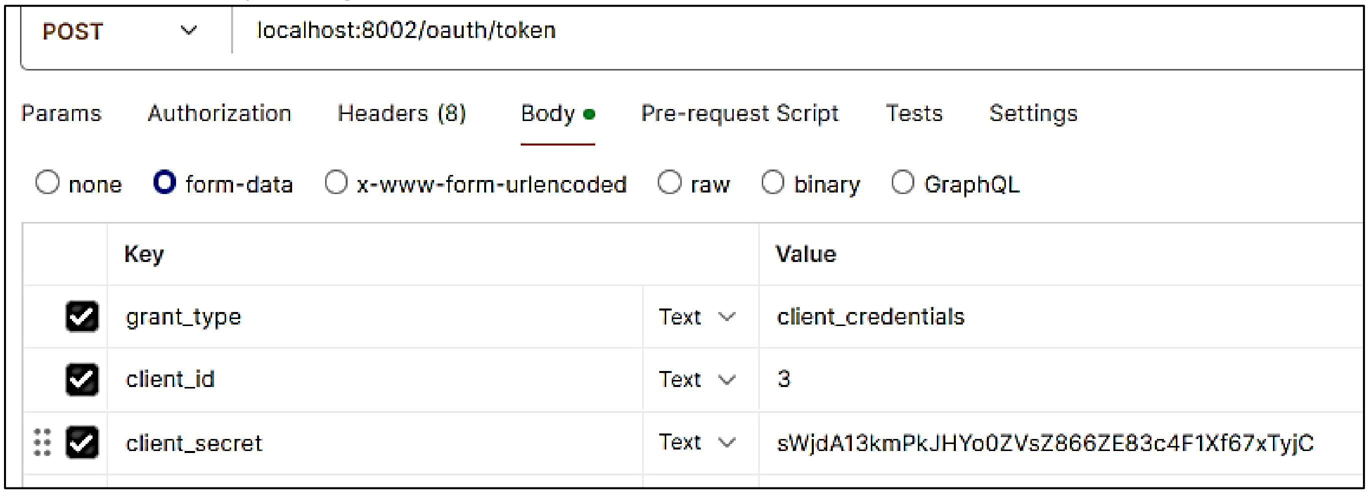
Obtaining and using access tokens for the Laravel/Lumen API Gateway

You can get access token from client credentials
*you already have 2 client credentials when you used ***php artisan passport:install***, but you can create more clients:

```
\> php artisan passport:client
```

*you can leave the id and the redirection blank

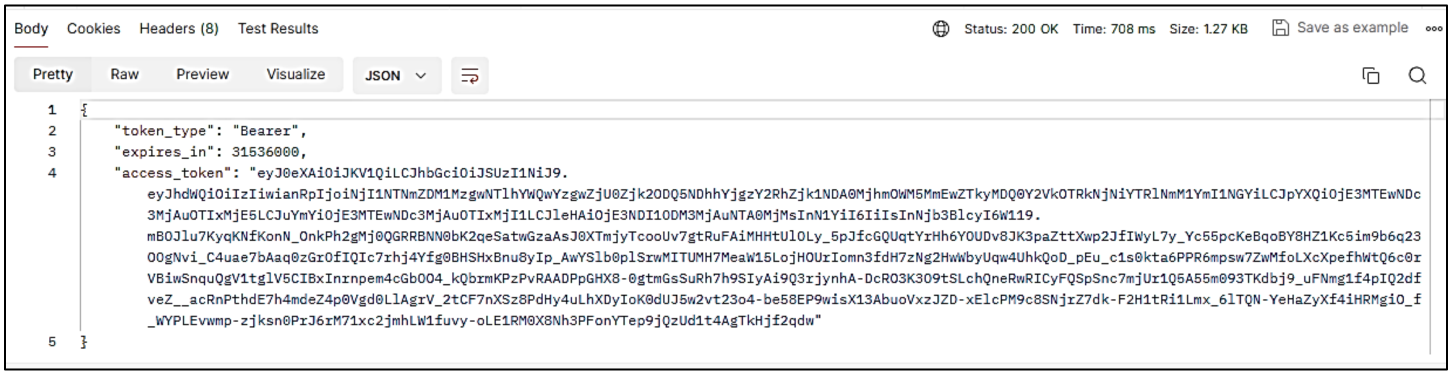
You can then send a request to get an access token


POST localhost:8002/oauth/token

Params | Authorization | Headers (8) | Body ● | Pre-request Script | Tests | Settings

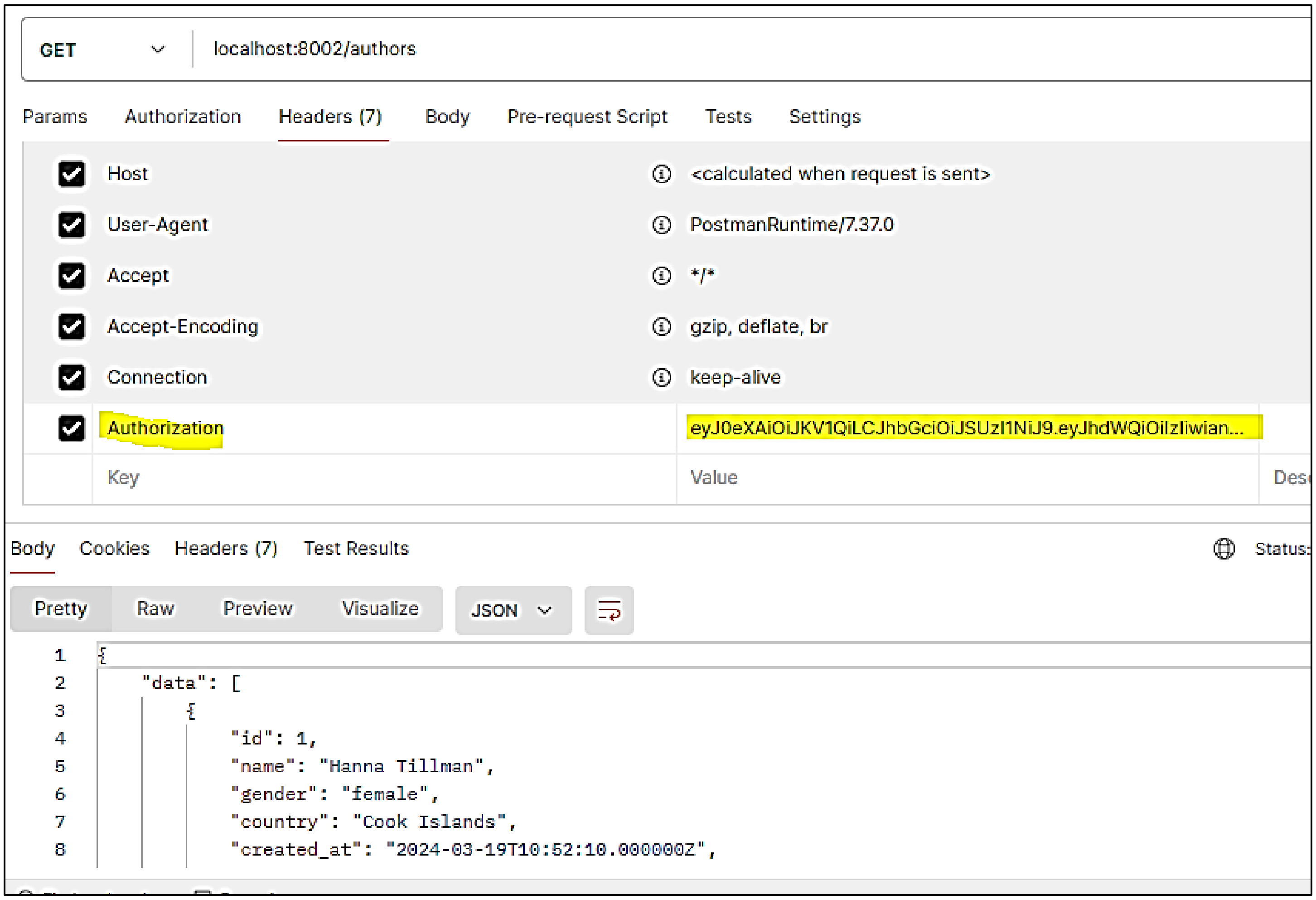
none [✓ form-data] x-www-form-urlencoded raw binary GraphQL

	Key		Value
✓	grant_type	Text	client_credentials
✓	client_id	Text	3
✓	client_secret	Text	sWjdA13kmPkJHYo0ZVsZ866ZE83c4F1Xf67xTyjC



```
{
    "token_type": "Bearer",
    "expires_in": 31536000,
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
        eyJhdWQiOiIzIiwianRpIjoiNjI1NTNmZDM1MzgwNTlhYWQwYzgwZjU0Zjk2ODQ5NDhhYjgzY2RhZjk1NDA0MjhmOWM5MmEwZTkyMDQ0Y2VkOTRkNjNiYTRlNmM1YmI1NGYiLCJpYXQiOjE3MTEwNDc
        3MjAuOTIxMjE5LCJuYmYiOjE3MTEwNDc3MjAuOTIxMjI1LCJleHAiOjE3NDI1ODM3MjAuNTA0MjMsInN1YiI6IiIsInNjb3BlcyI6W119.
        mBOJlu7KyqKNfKonN_OnkPh2gMj0QGRRBNN0bK2qeSatwGzaAsJ0XTmjyTcooUv7gtRuFAiMHHtUlOLy_5pJfcGQUqtYrHh6YOUDv8JK3paZttXwp2JfIWyL7y_Yc55pcKeBqoBY8HZ1Kc5im9b6q23
        OOgNvi_C4uae7bAaq0zGrOfIQIc7rhj4Yfg0BHSHxBnu8yIp_AwYSlb0plSrwMITUMH7MeaW15LojHOUrIomn3fdH7zNg2HwWbyUqw4UhkQoD_pEu_c1s0kta6PPR6mpsw7ZwMfoLXcXpefhWtQ6c0r
        VBiwSnquQgV1tglV5CIBxInrnpem4cGbOO4_kQbrmKPzPvRAADPpGHX8-0gtmGsSuRh7h9SIyAi9Q3rjynhA-DcRO3K3O9tSLchQneRwRICyFQSpSnc7mjUr1Q5A55m093TKdbj9_uFNmg1f4pIQ2df
        veZ__acRnPthdE7h4mdeZ4p0Vgd0LlAgrV_2tCF7nXSz8PdHy4uLhXDyIoK0dUJ5w2vt23o4-be58EP9wisX13AbuoVxzJZD-xElcPM9c8SNjrZ7dk-F2H1tRi1Lmx_6lTQN-YeHaZyXf4iHRMgiO_f
        _WYPLEvwmp-zjksn0PrJ6rM71xc2jmhLW1fuvy-oLE1RM0X8Nh3PFonYTep9jQzUd1t4AgTkHjf2qdw"
}
```

You can now access other routes by adding header('Authorization','<Access Token>')



localhost:8002/authors

✓	Host	<calculated when request is sent>
✓	User-Agent	PostmanRuntime/7.37.0
✓	Accept	*/*
✓	Accept-Encoding	gzip, deflate, br
✓	Connection	keep-alive
✓	Authorization	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJhdWQiOiIzIiwian...

```
{
    "data": [
        {
            "id": 1,
            "name": "Hanna Tillman",
            "gender": "female",
            "country": "Cook Islands",
            "created_at": "2024-03-19T10:52:10.000000Z",
```

Preparing the API Gateway to authenticate its requests

Protect access to the internal microservices (authors and books)

[ApiGateway]
./config/services.php

```
<?php

return [
   'authors' => [
      'base_uri' => env('AUTHORS_SERVICE_BASE_URL'),
      'secret' => env('AUTHORS_SERVICE_SECRET'),
   ],

   'books' => [
      'base_uri' => env('BOOKS_SERVICE_BASE_URL'),
      'secret' => env('BOOKS_SERVICE_SECRET'),
   ],
];
```

.env

```
…
AUTHORS_SERVICE_BASE_URL=localhost:8000
AUTHORS_SERVICE_SECRET=cA13CzTODXkNdNHKlHPJjCFeFAiZcqBc
BOOKS_SERVICE_BASE_URL=localhost:8001
BOOKS_SERVICE_SECRET=NdeIUbizDZr1n9URIDQH5lmLFHUMHITi
…
```

./Services/AuthorService.php

```
…
   public $baseUri;
   public $secret;

   public function __construct()
   {
      $this->baseUri =  config('services.authors.base_uri');
      $this->secret = config('services.authors.secret');
   }
…
```

./Services/BookService.php

```
…
   public $baseUri;
   public $secret;

   public function __construct()
   {
      $this->baseUri =  config('services.books.base_uri');
      $this->secret = config('services.books.secret');
   }
…
```

If the secret exists, then we can add it to the authorization header
./app/Traits/ConsumesExternalService.php

```
…
   public function performRequest($method,$requestUrl,$formParams=[],$headers=[]){
      $client = new Client([
         'base_uri' => $this->baseUri,
      ]);

      if(isset($this->secret)){
         $headers['Authorization'] = $this->secret;
      }

      $response = $client->request($method, $requestUrl, ['form_params'=>$formParams,'headers'=>$headers]);
      return $response->getBody()->getContents();
   }
…
```

Authenticating direct access to the Laravel/Lumen projectsite1 microservice

Allow access to Authors Microservice only from API Gateway
[Authors Microservice]

.env

```
//add all accepted secrets,
//the last one in this example is the secret from the APIGateway .env (AUTHORS_SERVICE_SECRET)
…
ACCEPTED_SECRETS=dsfsvhtrh,vdfgtyrttry,cA13CzTODXkNdNHKlHPJjCFeFAiZcqBc
…
```

Now, only the API Gateway can access Authors Microservice directly

Authenticating direct access to the microservice of projectsite2s

Allow access to Books Microservice only from API Gateway
[Books Microservice]

.env

```
//add all accepted secrets,
//the last one in this example is the secret from the APIGateway .env (BOOKS_SERVICE_SECRET)
…
ACCEPTED_SECRETS=dsfsvhtrh,vdfgtyrttry, NdeIUbizDZr1n9URIDQH5lmLFHUMHITi
…
```

Now, only the API Gateway can access Books Microservice directly

Authenticating user access with access tokens

Creating the migration for the users table in Laravel/Lumen

We can copy the migration file of Users from the Laravel Github repository
https://github.com/laravel/laravel/blob/11.x/database/migrations/0001_01_01_000000_create_users_table.php

you can add that migration file to your migrations
./database/migrations/0001_01_01_000000_create_users_table.php

```
<?php

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

class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Run the migration

```
php artisan migrate
php artisan migrate:fresh          //recreates all tables with 0 records
```

if you recreated the oauth tables also run this again

```
php artisan passport:install
```

Creating the controller and the routes to manage users in Laravel/Lumen

./app/Http/Controllers/UserController.php

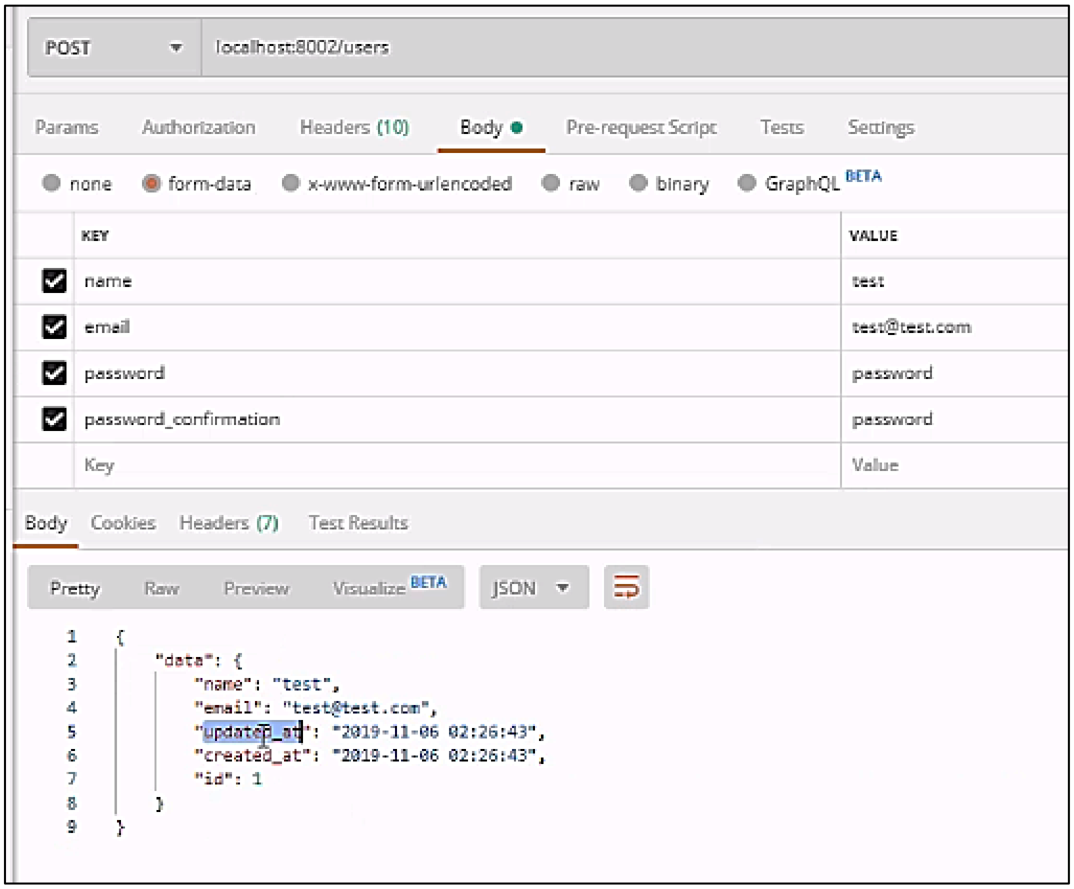
```
<?php

namespace App\Http\Controllers;

use App\Traits\ApiResponder;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Http\Response;
use Illuminate\Support\Facades\Hash;

class UserController extends Controller
{
    use ApiResponder;

    public function __construct()
    {
        //
    }

    public function index()
    {
        $users = User::all();
        return $this->successResponse($users);
    }

    public function store(Request $request)
    {
        $rules = [
            'name' => 'required|max:255',
            'email' => 'required|email|unique:users,email',
            'password' => 'required|min:8|confirmed',
        ];

        $this->validate($request, $rules);

        $fields = $request->all();
        $fields['password'] = Hash::make($request->password);

        $user = User::create($fields);
        return $this->successResponse($user, Response::HTTP_CREATED);
    }

    public function show($user)
    {
        $user = User::findOrFail($user);
        return $this->successResponse($user);
    }
    public function update(Request $request, $user)
    {
        $rules = [
            'name' => 'max:255',
            'email' => 'email|unique:users,email,' . $user,
            'password' => 'min:8|confirmed',
        ];

        $this->validate($request, $rules);
        $user = User::findOrFail($user);
        $user->fill($request->all());

        if ($request->has('password')) {
            $user->password = Hash::make($request->password);
        }

        if ($user->isClean()) {
            return $this->errorResponse('At least one value must change', Response::HTTP_UNPROCESSABLE_ENTITY);
        }

        $user->save();

        return $this->successResponse($user);
    }

    public function destroy($user)
    {
        $user = User::findOrFail($user);
        $user->delete();
        return $this->successResponse($user);
    }

}
```

./routes/web.php

```
<?php

$router->group(['middleware' => 'client.credentials'], function () use ($router) {
    $router->get('/authors', 'AuthorController@index');
    $router->post('/authors', 'AuthorController@store');
    $router->get('/authors/{author}', 'AuthorController@show');
    $router->put('/authors/{author}', 'AuthorController@update');
    $router->patch('/authors/{author}', 'AuthorController@update');
    $router->delete('/authors/{author}', 'AuthorController@destroy');

    $router->get('/books', 'BookController@index');
    $router->post('/books', 'BookController@store');
    $router->get('/books/{book}', 'BookController@show');
    $router->put('/books/{book}', 'BookController@update');
    $router->patch('/books/{book}', 'BookController@update');
    $router->delete('/books/{book}', 'BookController@destroy');

    $router->get('/users', 'UserController@index');
    $router->post('/users', 'UserController@store');
    $router->get('/users/{user}', 'UserController@show');
    $router->put('/users/{user}', 'UserController@update');
    $router->patch('/users/{user}', 'UserController@update');
    $router->delete('/users/{user}', 'UserController@destroy');
});
```

Creating users and creating access tokens associated with users

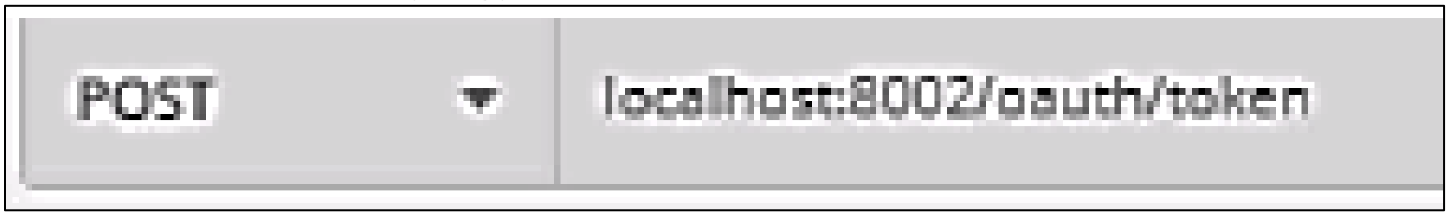
Try adding users thru postman



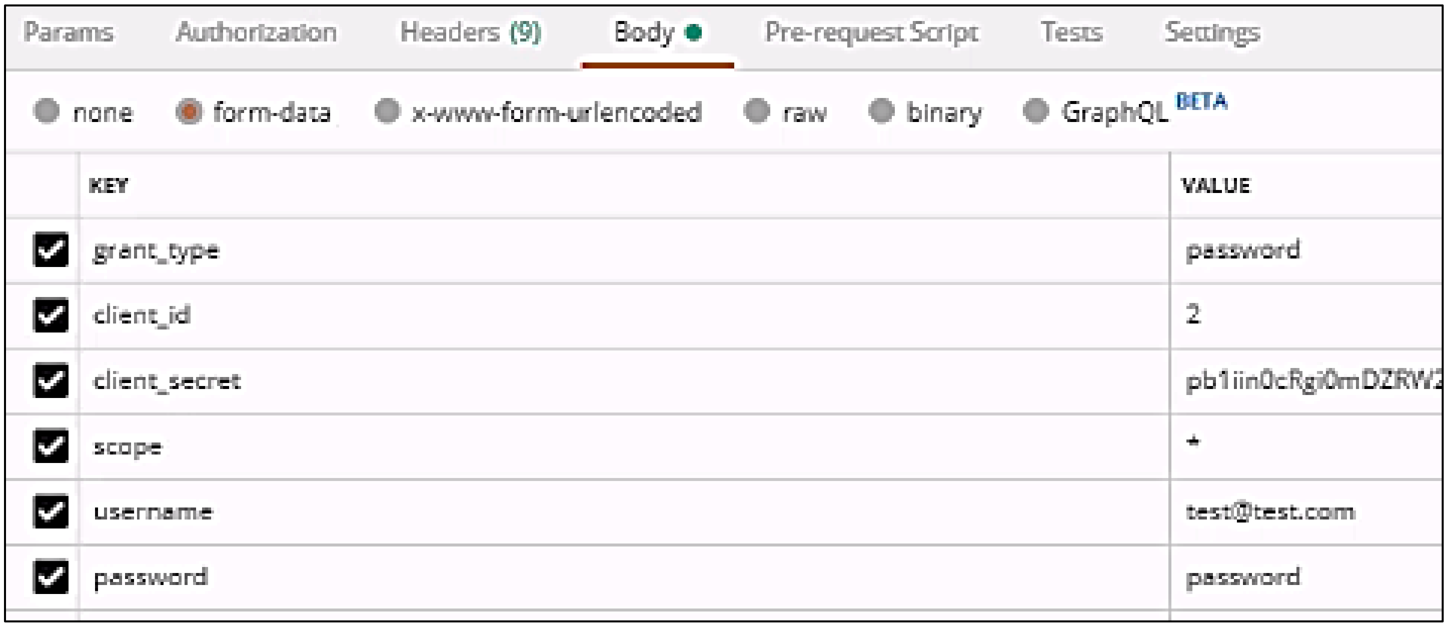
KEY	VALUE
name	test
email	test@test.com
password	password
password_confirmation	password
Key	Value

```
{
    "data": {
        "name": "test",
        "email": "test@test.com",
        "updated_at": "2019-11-06 02:26:43",
        "created_at": "2019-11-06 02:26:43",
        "id": 1
    }
}
```

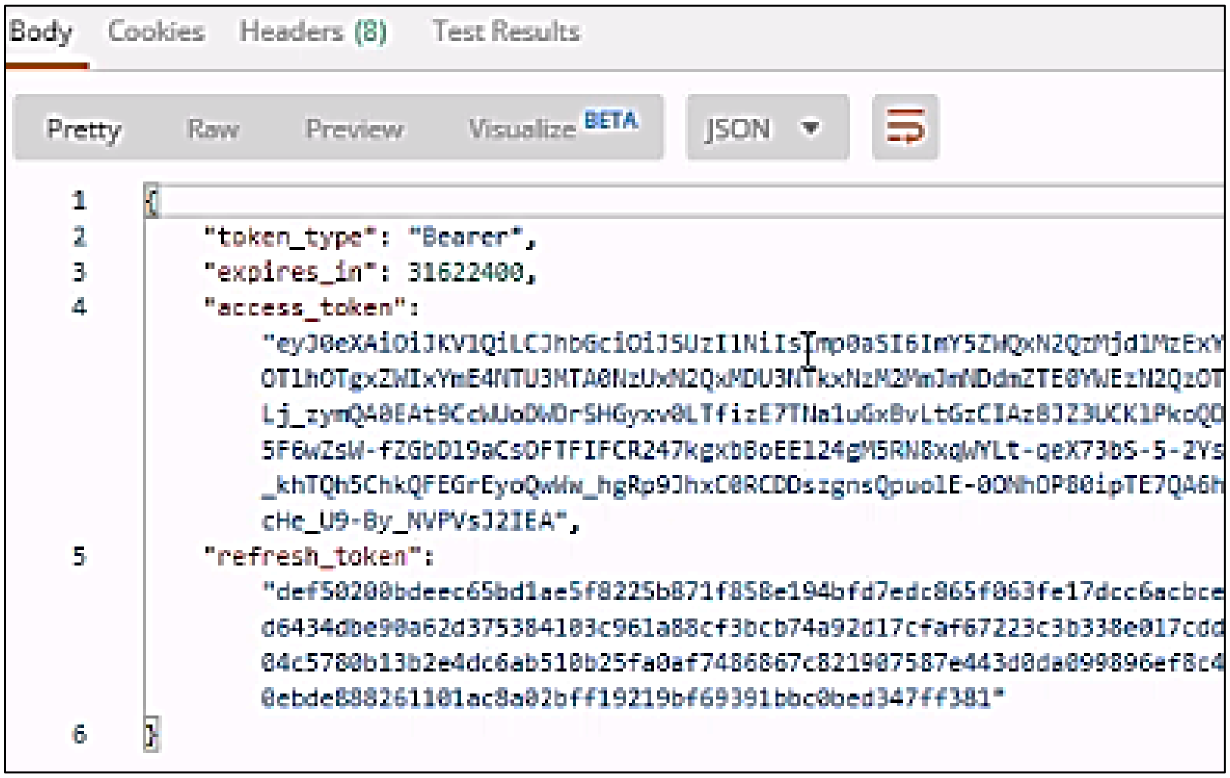
We can also obtain access token by using password



POST localhost:8002/oauth/token



	KEY	VALUE
✓	grant_type	password
✓	client_id	2
✓	client_secret	pb1iin0cRgi0mDZRW2
✓	scope	*
✓	username	test@test.com
✓	password	password



```
{
    "token_type": "Bearer",
    "expires_in": 31622400,
    "access_token":
        "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6ImY5ZWQxN2QzMjd1MzExY
        OTlhOTgxZWIxYmE4NTU3MTA0NzUxN2QxMDU3NTkxNzM2MmJmNDdmZTE0YWEzN2QzOT
        Lj_zymQA0EAt9CcWJoDWDr5HGyxv0LTfizE7TNa1uGxBvLtGzCIAz8JZ3UCK1PkoQD
        5F6wZsW-fZGbD19aCsOFTFIFCR247kgxbBoEE124gM5RN8xqWYLt-qeX73bS-5-2Ys
        _khTQh5ChkQFEGrEyoQwWw_hgRp9JhxC0RCDDszgnsQpuolE-0ONhOP80ipTE7QA6h
        cHe_U9-By_NVPVsJ2IEA",
    "refresh_token":
        "def50200bdeec65bd1ae5f8225b871f858e194bfd7edc865f063fe17dcc6acbce
        d6434dbe90a62d375384103c961a88cf3bcb74a92d17cfaf67223c3b338e017cdd
        04c5780b13b2e4dc6ab510b25fa0af7486867c821907587e443d0da099896ef8c4
        0ebde888261101ac8a02bff19219bf69391bbc0bed347ff381"
}
```

Identifying an authenticated user through access token

./app/Http/Controllers/UserController.php

```
//add a function inside the class
…
   public function me(Request $request)
   {
      return $this->successResponse($request->user());
   }
…
```

./routes/web.php

```
//protect the route by checking the user
<?php

$router->group(['middleware' => 'client.credentials'], function () use ($router) {
   $router->get('/authors', 'AuthorController@index');
   $router->post('/authors', 'AuthorController@store');
   $router->get('/authors/{author}', 'AuthorController@show');
   $router->put('/authors/{author}', 'AuthorController@update');
   $router->patch('/authors/{author}', 'AuthorController@update');
   $router->delete('/authors/{author}', 'AuthorController@destroy');

   $router->get('/books', 'BookController@index');
   $router->post('/books', 'BookController@store');
   $router->get('/books/{book}', 'BookController@show');
   $router->put('/books/{book}', 'BookController@update');
   $router->patch('/books/{book}', 'BookController@update');
   $router->delete('/books/{book}', 'BookController@destroy');

   $router->get('/users', 'UserController@index');
   $router->post('/users', 'UserController@store');
   $router->get('/users/{user}', 'UserController@show');
   $router->put('/users/{user}', 'UserController@update');
   $router->patch('/users/{user}', 'UserController@update');
   $router->delete('/users/{user}', 'UserController@destroy');
});

/
 * User credentials protected routes
 */
$router->group(['middleware' => 'auth:api'], function () use ($router) {
   $router->get('/users/me', 'UserController@me');
});
```

Authentication Steps:

1. Get the user password
2. Use the password to obtain access token

```
[POST] localhost:8002/oauth/token
[BODY FORM-DATA]
grant_type          password
client_id           3
client_secret       sdgdhtrhfngukykuildth
scope               *
username            test@test.com
password            password

*copy the access token generated from this step
```

3. Authenticate

```
[GET] localhost:8002/users/me
[SET IN HEADERS]
Authorization       <Access Token>

*you should be able to get the name of the current user
```

4. Use that name as a confirmation that that user has gained authentication for the resources.