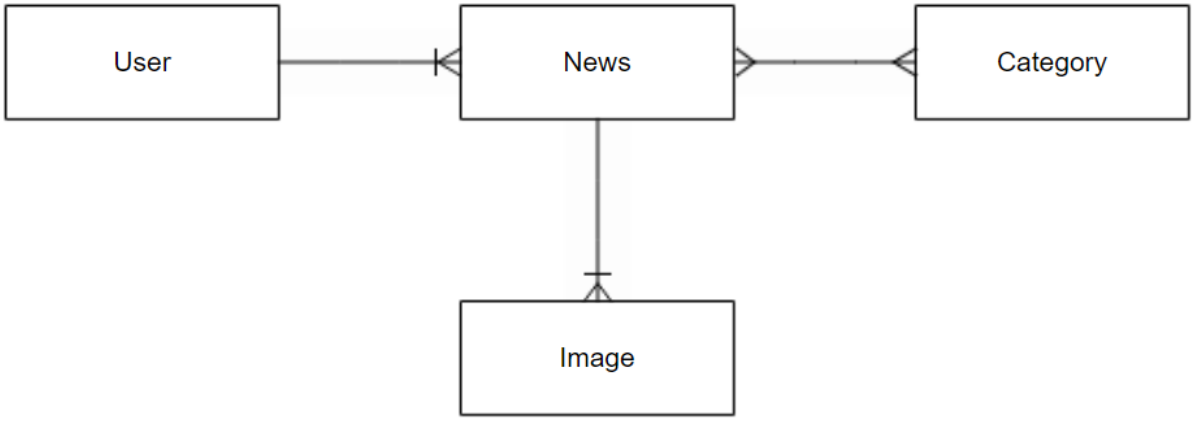
Day-4: Building News Platform with Laravel API – Laravel Passport Authentication

Prepared by: Rana M. Fakeeh

Application overview:

Our goal is to create a News Platform's CRUD APIs with the following entities:

- ***Category***: To categorize news articles.
- ***News***: To store news articles.
- ***Image***: To associate image album with news articles.
- ***User***: To manage user accounts.



Training program:

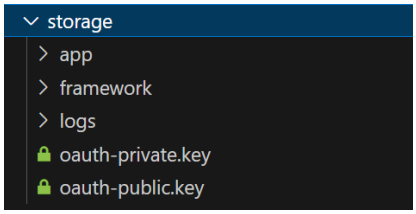
Day-1	Setup. CRUD for Category entity.
Day-2	CRUD for News, and Image entities. Upload image files to server. One-to-Many and Many-to-Many associations.
Day-3	Recap. Filtering, Sorting, and Pagination of list results.
Day-4	Laravel Passport authentication (OAuth2 token-based) Register, Login, Logout, and Profile APIs for User entity. Public vs. Protected endpoints.
Day-5	Integration with a ReactJS Frontend. Localization. Deployment (Docker).

Learning objectives:

- Setup the Laravel Passport authentication package (OAuth2 token-based).
- Update User table.
- Create RESTful API for Register, Login, Logout, and Profile operations.
- Define public and protected (private) endpoints.
- Sending authenticated requests using Postman.
- Hands-on.

Steps:

1- **Environment**
   - Run XAMP control panel.
   - Start Apache server
   - Start MySQL server
   - Open VS Code on previous project.

2- **Setup the Laravel Passport authentication.**
   - Install passport package
   - composer require laravel/passport --with-all-dependencies
   - The ***--with-all-dependencies*** flag allows resolving conflicts in php packages versions.
   - Migrate changes to database schema.
   - php artisan migrate
   - Check database changes in MySQL (phpMyAdmin)
   - Open the ***config/auth.php*** file and add ***api guards*** under the ***web guards***:
   - 'api' => [
   - 'driver' => 'passport',
   - 'provider' => 'users',
   - 'hash' => false,
   -],
   - Open ***app/Models/User.php*** file.
   - At the top use the ***Passport*** namespace instead of ***Sanctum***:
   - //use Laravel\ Sanctum \HasApiTokens;
   - use Laravel\Passport\HasApiTokens;
   - Install Passport encryption keys
   - php artisan passport:install --force
   - After running this command, your Laravel application will be configured to issue and validate OAuth2 access tokens, allowing you to implement secure API authentication and authorization.
   -
```
Encryption keys generated successfully.
Personal access client created successfully.
Client ID: 1
Client secret: jV95V2D1dJaWIMdgWWi31snYlXq88Ow4eziNM4AT
Password grant client created successfully.
Client ID: 2
Client secret: SdizotXy8x4pDLqLrbRkODbDO3n2DeBjXiW7PDTu
```
   - Check encryption keys generated under ***storage*** directory.
   -


3- User Module (Migration -> Model -> Factory -> Seeder -> Resource)

- ***Migration***
- Open ***database/migrations/timestamp_create_users_table.php*** file.
- Add fields ***first_name***, ***last_name***, and ***avatar***.

```
//$table->string('name');
$table->string('first_name');
$table->string('last_name');
$table->string('avatar');
```

- ***Model***
- Open ***app/Models/User.php*** file.
- Add the new attributes in the ***$fillable*** mass assigned array.

```
//'name',
'first_name',
'last_name',
'avatar',
```

- ***Factory***
- Open ***database/factories/UserFactory.php*** file.
- Create fake values for fields ***first_name***, ***last_name***, and ***avatar***.

```
//'name' => fake()->name(),
'first_name' => ucfirst(fake()->word()),
'last_name' => ucfirst(fake()->word()),
'avatar' => fake()->imageUrl(),
```

- ***Seeder***
- Open ***database/seeders/DatabaseSeeder.php*** file.
- Populate database with 5 dummy users.

```
User::factory(5)->create();
```

- Migrate changes to database schema.

```
php artisan migrate
```

- [if migration failed due to an existing table, drop it from the database then re-migrate].
- Refresh migrations to empty the tables.

```
php artisan migrate:refresh
```

- Run seeding

```
php artisan db:seed
```

- Re-install Passport encryption keys

```
php artisan passport:install --force
```

- Check database changes in MySQL (phpMyAdmin)
- ***Resource***
- Create User resource.

```
php artisan make:resource UserResource
```

- Open ***app/Http/Resources/UserResource.php*** file.
- Return only the user's ***id***, ***full_name***, ***avatar***, and ***email***.

```
[
    'id' => $this->resource->id,
```

```
-       'name' => $this->resource->first_name . ' ' . $this->resource->last_name,
-       'avatar' => $this->resource->avatar,
-       'email' => $this->resource->email
-   ];
```

4- Eloquent One-to-Many Relationship (Migration -> Model -> Factory -> Seeder -> Resource)

- A User **has many** News articles.
- A News article **belongs to** one User.
- Therefore, News table (the many side) should have the foreign key from User table.
- **News Module**
- ***Migration***
- Open ***database/migrations/timestamp_create_news_table.php*** file.
- Add the foreign key ***user_id*** from User table.

```
-   $table->unsignedBigInteger('user_id');
-   $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
```

- ***Model***
- Open ***app/Models/News.php*** file.
- Add foreign key field ***user_id***.

```
-   protected $fillable = [
-           'title',
-           'body',
-           'thumbnail',
-           'completed',
-           'visible',
-           'user_id'
-   ];
```

- Create function to retrieve the associated record (singular).

```
-   public function user() {
-       return $this->belongsTo(User::class);
-   }
```

- ***Factory***
- Open ***database/factories/NewsFactory.php*** file.
- Create fake values for foreign key field ***user_id***.

```
-   'user_id' => fake()->numberBetween(1,5)
```

- ***Seeder***
- Open ***database/seeders/DatabaseSeeder.php*** file.
- Check that User fake records are created before News recodes.
- Migrate changes to database schema.

```
-   php artisan migrate
```

- [if migration failed due to an existing table, drop it from the database then re-migrate].
- Refresh migrations to empty the tables.

```
-   php artisan migrate:refresh
```

- Run seeding

```
-   php artisan db:seed
```

- Re-install Passport encryption keys

- php artisan passport:install --force
- Check database changes in MySQL (phpMyAdmin)
- ***Resource***
- Open ***app/Http/Resources/NewsResource.php*** file.
- Return also the associated ***user*** resource.

```
$author = (new UserResource($this->user))->toArray($request);
return [
        ....
        'author_id' => $author['id'],
        'author_avatar' => $author['avatar'],
        'author_name' => $author['name']
];
```

- **User Module**
- ***Model***
- Open ***app/Models/User.php*** file.
- Create function to retrieve the associated records (plural).

```
public function news() {
    return $this->hasMany(News::class);
}
```

- ***Seeder***
- Open ***database/seeders/DatabaseSeeder.php*** file.
- Check that User fake records are created before News recodes.
- ***Resource***
- Open ***app/Http/Resources/UserResource.php*** file.
- Return also the associated ***news_count***.

```
"total_news_count" => $this->resource->news()->count(),
"public_news_count" => $this->resource->news()->where('visible', true)->count(),
"private_news_count" => $this->resource->news()->where('visible', false)->count(),
"completed_news_count" => $this->resource->news()->where('completed', true)->count()
```

5- User Module (Controller -> Route)

- Create controller for ***User*** model

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

- Open ***app/Http/Controllers/UserController.php*** file and add the code snippet in (Appendix-I)

6- News Module (Controller -> Route)

- Open ***app/Http/Controllers/NewsController.php*** file and add the code snippet in (Appendix-II)
- Open ***app/Http/Controllers/Controller.php*** file and add the code snippet in (Appendix-III)

7- Define public vs. protected API routes

- Open ***routes/api.php*** file and add Laravel Passport APIs with ***user/*** prefix (Appendix-IV).

8- Handle unauthorized access and bad route errors.

- Open ***app/Exceptions/Handler.php*** class file and add the code snippet in (Appendix- V)

9- Testing API with Postman

- Run XAMP control panel.
- Start Apache server
- Start MySQL server
- Migrate changes to database schema.
- php artisan migrate
- Test the server running
- php artisan serve
- Open Postman
- ***http://localhost:8000/api/user/****
- After successful login, in the response, you got the token. Copy it and paste it with the next secure requests.
- ***http://localhost:8000/api/category/****
- ***http://localhost:8000/api/news/****
- ***http://localhost:8000/api/album/****
- For create, use ***Body*** -> ***form-data***
- For update, use ***Body*** -> ***x-www-form-urlencoded***

Appendix-I

UserController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;
use App\Models\User;
use App\Http\Resources\UserResource;
use Exception;
use Illuminate\Support\Facades\Hash;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;
use Illuminate\Support\Facades\Auth;

class UserController extends Controller
{
    /**
     * Register User
     *
     * @param Request $request
     * @return void
     */
    public function register(Request $request) { // store

        $validator = Validator::make($request->all(), [
            "first_name" => "required",
            "last_name" => "required",
            "email" => "required|email|unique:users,email",
            "password" => "required|min:8",
            'avatar' => 'image|mimes:jpg,png,jpeg,gif,svg|max:2048',

        ]);

        if($validator->fails()) {
            return response()->json([
                "status" => "fail",
                "error" => true,
                "validation_errors" => $validator->errors()
            ]);
        }

        try {

            $user = User::create([
                'first_name' => $request->first_name,
```

```
                'last_name' => $request->last_name,
                'email' => $request->email,
                'password' => Hash::make($request->password)
            ]);

            // if has avatar image
            if($request->file('avatar')) {
                $image_path = $request->file('avatar')->store('avatars',
'public');
                $user['avatar'] = $image_path;
                $user->save();
            }

            return response()->json([
                "status" => "success",
                "error" => false,
                "message" => "Success! User registered.",
                "data" => new UserResource($user),
            ], 201);

        } catch(Exception $exception) {

            return response()->json([
                "status" => "fail",
                "error" => true,
                "message" => $exception->getMessage(),
            ], 404);
        }

    }

    /**
     * User Login
     *
     * @param Request $request
     * @return void
     */
    public function login(Request $request) {

        $validator = Validator::make($request->all(), [
            "email" => "required|email",
            "password" => "required|min:8"
        ]);

        if($validator->fails()) {
            return response()->json([
                "status" => "fail",
                "error" => true,
```

```
                "validation_errors" => $validator->errors()
            ]);
        }

        try {
            if(Auth::attempt(['email' => $request->email, 'password' => $request->password])) {
                $user = Auth::user();
                $token = $user->createToken('token')->accessToken;
                return response()->json([
                    "status" => "success",
                    "error" => false,
                    "message" => "Success! you are logged in.",
                    "token" => $token
                ], 200);
            }
            return response()->json([
                "status" => "failed",
                "message" => "Failed! invalid credentials."
            ], 404);

        } catch(Exception $exception) {

            return response()->json([
                "status" => "fail",
                "error" => true,
                "message" => $exception->getMessage(),
            ], 404);
        }
    }

    /**
     * Logged User Data Using Auth Token
     *
     * @return void
     */
    public function profile() {
        try {
            $user = Auth::user();
            return response()->json([
                "status" => "success",
                "error" => false,
                "data" => new UserResource($user)
            ], 200);

        } catch(NotFoundHttpException $exception) {
            return response()->json([
                "status" => "failed",
```

```
                "error" => $exception->getMessage(),
            ], 401);
        }
    }

    /**
    * Logout Auth User
    *
    * @param Request $request
    * @return void
    */
    public function logout() {

        if(Auth::check()) {
            Auth::user()->token()->revoke();
            return response()->json([
                "status" => "success",
                "error" => false,
                "message" => "Success! You are logged out."
            ], 200);
        }
        return response()->json([
            "status" => "failed",
            "error" => true,
            "message" => "Failed! You are already logged out."
        ], 403);
    }
}
```

Appendix-II

NewsController.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Category;
use App\Http\Resources\CategoryResource;
use App\Http\Resources\CategoryCollection;
use App\Models\News;
use App\Http\Resources\NewsResource;
use App\Http\Resources\NewsCollection;
use App\Models\Image;
use Illuminate\Support\Facades\Validator;
use Exception;
use Facade\FlareClient\Http\Exceptions\NotFound;
use Illuminate\Support\Facades\Auth;

class NewsController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index(Request $request,  $articles)
    {
        // Filter by attribute
        if($request->exists('visible')) {
            $visible = $request->boolean('visible'); // it parses 0/1,
true/false, on/off into boolean
            $articles = $articles->where('visible','=',$visible);
        }

        // Filter by attribute
        $start_date = $request->input('start_date');
        $end_date = $request->input('end_date');
        if ($start_date && $end_date) {
            $articles->whereBetween('updated_at', [$start_date, $end_date]);
        }

        // Filter by category
        $category_id = $request->input('category');
        if ($category_id) {
```

```
            $articles->whereHas('categories', function($query) use($category_id)
{
                //$query->where('categories.id', $category_id);
                $query->whereIn('categories.id', $category_id);
            });
        }

        // Filter by query string in title or body
        $q = $request->input('q');
        if ($q) {
            $articles->where(function ($query) use ($q) {
                $query->where('title', 'LIKE', '%' . $q . '%')->orWhere('body',
'LIKE', '%' . $q . '%');
            });
        }

        // Sorting
        $articles = $this->sorted($articles,$request);

        // Paginatable collection
        $articles = new NewsCollection($articles->get());

        // Pagination
        $articles = $this->paginated($articles,$request);

        return $articles;
    }

    public function public_index(Request $request)
    {
        $articles = News::where('visible', true);

        // Filter by author
        $author_id = $request->input('author');
        if ($author_id) {
            $articles->whereHas('user', function($query) use($author_id) {
                $query->where('users.id', $author_id);
            });
        }

        return response()->json([
            "status" => "success",
            "error" => false,
            "data" => $this->index($request,$articles),
        ],200);
    }

    public function private_index(Request $request)
```

```
    {
        // Retrieve the authenticated user
        $user = Auth::user();

        // Filter by authenticated user
        $articles = $user->news();

        return response()->json([
            "status" => "success",
            "error" => false,
            "data" => $this->index($request,$articles),
        ],200);
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            "title" => "required|min:3|unique:news,title",
            'thumbnail' => 'required|image|mimes:jpg,png,jpeg,gif,svg|max:2048',
        ]);

        if($validator->fails()) {
            return response()->json([
                "status" => "fail",
                "error" => true,
                "validation_errors" => $validator->errors()
            ]);
        }

        $news_id = NULL;
        try {

            $image_path = $request->file('thumbnail')-
>store('thumbnails','public');
            $article = News::create([
                "title" => $request->title,
                "body" => $request->body,
                "thumbnail" => $image_path,
                "completed" => $request->completed,
                "visible" => $request->visible,
                "user_id" => Auth::user()->id
            ]);
```

```
            $news_id = $article->id;

            if($request->categories) {
                $categories = $request->categories;
                $article->categories()->attach($categories);
            }

            if($request->images) {
                $images = $request->images;
                Image::whereIn('id', $images)->update(['news_id' => $news_id]);
                Image::where('news_id', $news_id)->whereNotIn('id', $images)-
>delete();
            }

            return response()->json([
                "status" => "success",
                "error" => false,
                "message" => "Success! news article created.",
                "data" => new NewsResource($article),
            ], 201);
        }
        catch(Exception $exception) {

            if($news_id) {
                $article = News::find($news_id);
                $article->categories()->detach();
                Image::where('news_id', $id)->delete();
                $article->delete();
            }

            return response()->json([
                "status" => "fail",
                "error" => true,
                "message" => $exception->getMessage(),
            ], 404);
        }
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //$article = News::find($id);
```

```php
        // Retrieve the authenticated user
        $user = Auth::user();

        // Filter by authenticated user
        $article = $user->news()->find($id);

        if($article) {
            return response()->json([
                "status" => "success",
                "error" => false,
                "data" => new NewsResource($article),
            ], 200);
        }

        return response()->json([
            "status" => "failed",
            "error" => true,
            "message" => "Failed! no news article found."
        ], 404);
    }

    public function public_show($id)
    {
        $article = News::where('visible', true)->find($id);

        if($article) {
            return response()->json([
                "status" => "success",
                "error" => false,
                "data" => new NewsResource($article),
            ], 200);
        }

        return response()->json([
            "status" => "failed",
            "error" => true,
            "message" => "Failed! no news article found."
        ], 404);
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
```

```
    {

        //$article = News::find($id);

        // Retrieve the authenticated user
        $user = Auth::user();

        // Filter by authenticated user
        $article = $user->news()->find($id);

        if($article) {

            $validator = Validator::make($request->all(), [
                "title" => "required|min:3",
                'thumbnail' => 'image|mimes:jpg,png,jpeg,gif,svg|max:2048',
            ]);

            if($validator->fails()) {
                return response()->json([
                    "status" => "fail",
                    "error" => true,
                    "validation_errors" => $validator->errors()
                ]);
            }

            $article['title'] = $request->title;

            // if has body
            if($request->body) {
                $article['body'] = $request->body;
            }

            // if has thumbnail image
            if($request->file('thumbnail')) {
                $image_path = $request->file('thumbnail')->store('thumbnails',
'public');
                $article['thumbnail'] = $image_path;
            }

            // if has visible
            if($request->visible) {
                $article['visible'] = $request->visible;
            }

            // if has completed
            if($request->completed) {
                $article['completed'] = $request->completed;
            }
```

```
            // if has categories
            if($request->categories) {
                $categories = $request->categories;
                $article->categories()->sync($categories);
            }

            // if has album images
            if($request->images) {
                $images = $request->images;
                Image::whereIn('id', $images)->update(['news_id' => $id]);
                Image::where('news_id', $id)->whereNotIn('id', $images)-
>delete();
            }

            $article->save();

            return response()->json([
                "status" => "success",
                "error" => false,
                "message" => "Success! news article updated.",
                "data" => new NewsResource($article)
            ], 201);
        }

        return response()->json([
            "status" => "failed",
            "error" => true,
            "message" => "Failed no news article found."
        ], 404);
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //$article = News::find($id);

        // Retrieve the authenticated user
        $user = Auth::user();

        // Filter by authenticated user
        $article = $user->news()->find($id);
```

```
        if($article) {
            $article->categories()->detach();
            Image::where('news_id', $id)->delete();
            $article->delete();

            return response()->json([
                "status" => "success",
                "error" => false,
                "message" => "Success! news article deleted."
            ], 200);
        }

        return response()->json([
            "status" => "failed",
            "error" => true,
            "message" => "Failed no news article found."
        ], 404);
    }
}
```

Appendix-III

Controller.php

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Routing\Controller as BaseController;
use Illuminate\Http\Request;

class Controller extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;

    public static $perPage = 5;

    public function sorted($query, Request $request) {

        $sortField = $request->input('sort_field', 'id');
        $sortDirection = $request->input('sort_direction', 'DESC');
        $query->orderBy($sortField, $sortDirection);

        return $query;
    }

    public function paginated($query, Request $request) {

        $perPage = $request->input('per_page', Controller::$perPage);
        $query = $query->paginate($perPage);
        $query->appends($request->query());

        return $query;
    }
}
```

Appendix-IV

api.php

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\NewsController;
use App\Http\Controllers\ImageController;
use App\Http\Controllers\UserController;

/*
|--------------------------------------------------------------------------
| API Routes
|--------------------------------------------------------------------------
|
| Here is where you can register API routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| is assigned the "api" middleware group. Enjoy building your API!
|
*/

//Route::resource('category', CategoryController::class);
//Route::get('open_document', [CategoryController::class, 'open_document']);

// public api routes with api/ prefix
Route::get('category', [CategoryController::class, 'index']);
Route::get('category/{id}', [CategoryController::class, 'show']);
Route::get('news/', [NewsController::class, 'public_index']);
Route::get('news/{id}', [NewsController::class, 'public_show']);

// grouped api routes with api/user/ prefix
Route::prefix('user')->group(function () {

    Route::post('register', [UserController::class, 'register']);
    Route::post('login', [UserController::class, 'login']);

    // passport auth api routes
    Route::middleware(['auth:api'])->group(function () {

        Route::get('/', [UserController::class, 'profile']);
        Route::get('logout', [UserController::class, 'logout']);

        Route::post('category', [CategoryController::class, 'store']);
        Route::put('category/{id}', [CategoryController::class, 'update']);
        Route::delete('category/{id}', [CategoryController::class, 'destroy']);
```

```
        Route::get('news', [NewsController::class, 'private_index']);
        Route::post('news', [NewsController::class, 'store']);
        Route::get('news/{id}', [NewsController::class, 'show']);
        Route::put('news/{id}', [NewsController::class, 'update']);
        Route::delete('news/{id}', [NewsController::class, 'destroy']);

        Route::post('album', [ImageController::class, 'store']);
        Route::delete('album/{id}', [ImageController::class, 'destroy']);

    });

});

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

Appendix-V

Handler.php

```php
<?php

namespace App\Exceptions;

use Illuminate\Foundation\Exceptions\Handler as ExceptionHandler;
use Symfony\Component\Routing\Exception\RouteNotFoundException;
use Symfony\Component\HttpKernel\Exception\MethodNotAllowedHttpException;

class Handler extends ExceptionHandler
{
    /**
     * A list of exception types with their corresponding custom log levels.
     *
     * @var array<class-string<\Throwable>, \Psr\Log\LogLevel::*>
     */
    protected $levels = [
        //
    ];

    /**
     * A list of the exception types that are not reported.
     *
     * @var array<int, class-string<\Throwable>>
     */
    protected $dontReport = [
        //
    ];

    /**
     * A list of the inputs that are never flashed to the session on validation
exceptions.
     *
     * @var array<int, string>
     */
    protected $dontFlash = [
        'current_password',
        'password',
        'password_confirmation',
    ];

    /**
     * Register the exception handling callbacks for the application.
     *
     * @return void
```

```
     */
    public function register()
    {
        $this->renderable(function (RouteNotFoundException $exception, $request)
{
            return response()->json([
                "status" => "Unauthorized access",
                "error" => true,
                "message" => "You are not allowed to access the API."
            ], 401);
        });

        $this->renderable(function (MethodNotAllowedHttpException $exception,
$request) {
            return response()->json([
                "status" => "Unrecognized route",
                "error" => true,
                "message" => $exception->getMessage()
            ], 401);
        });

    }

}
```