Práctica UD-7 Sesión-1

|  |  |  |  |
| --- | --- | --- | --- |
| Objetivos | * Aprender a trabajar en Laravel con servicios web. | | |
|
| Instrucciones de entrega | * No hay que entregar nada. | | |

* Abre una línea de comandos desde **VS Code** y ve a la ruta donde está el directorio ***htdocs*** de Xampp (**c:\xampp\htdocs**) después crea un nuevo proyecto con nombre ***PHPPracticaUD7Sesion1***:

|  |
| --- |
| C:\xampp\htdocs>laravel new PHPPracticaUD7Sesion1 |

* O como alternativa usando *composer*:

|  |
| --- |
| C:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel PHPPracticaUD7Sesion1 |

* Una vez creado carga el proyecto desde ***VS Code*** simplemente abriendo la carpeta o bien dentro del directorio con el comando ***code .*** .

## Instalación de Postman

* Programa que nos permite probar los servicios web.
* Vaya a la página oficial e instale la última versión:
  + <https://www.postman.com/downloads/>
* Una vez instalado al iniciar la primera vez nos pedirá que nos loguemos o creemos una cuenta pero podremos saltar este paso:



## Laravel y servicios web

### Ejemplo básico

* Vamos a realizar un ejemplo simple donde tendremos un controlador para un CRUD pero en vez de retornar a las vistas sus métodos devolverán contenido JSON, el ejemplo se base en el tutorial:
  + <https://www.netireki.eus/crear-rest-api-con-laravel-step-1/>
* Crea un nuevo modelo para artículos junto con su fichero de migración:
  + *php artisan make:model Article -m*
* Arranca **Xampp** y crea la base de datos a mano a través de phpMyAdmin:
  + ***phppracticaud7sesion1***
* Ahora abre el fichero del modelo ***Article*** y añade los campos que son editables:

class Article extends Model

{

use HasFactory;

protected $fillable = ['title', 'body'];

}

* A continuación vamos a crear un fichero de siembra (seed) para insertar datos de prueba en la base de datos para el modelo *Article*:
  + *php artisan make:seeder ArticleSeeder*
* Editamos el fichero recién creado ***seeders/ArticleSeeder*** con el siguiente contenido:

class ArticleSeeder extends Seeder

{

/\*\*

\* Run the database seeds.

\*

\* @return void

\*/

public function run()

{

// Truncamos la tabla para comenzar con una vacía.

Article::truncate();

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

// Ahora creamos los nuevos artículos de prueba:

for ($i = 0; $i < 50; $i++) {

Article::create([

'title' => $faker->sentence,

'body' => $faker->paragraph,

]);

}

}

}

* Ahora ejecutamos la migración:
  + *php artisan migrate*
* Después ejecutamos la siembra:
  + *php artisan db:seed --class=ArticleSeeder*
* Comprobamos en base de datos que se han insertado bien los registros de prueba.
* Ahora crearemos el controlador asociado al modelo Article para gestionarlo:
  + *php artisan make:controller ArticleController --resource --model=Article*

* Editaremos el contenido del fichero borrando los métodos:
  + create
  + edit
* En los métodos restantes añadiremos el siguiente código:

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

return Article::all();

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

$article = Article::create($request->all());

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

}

/\*\*

\* Display the specified resource.

\*

\* @param \App\Models\Article $article

\* @return \Illuminate\Http\Response

\*/

public function show(Article $article)

{

return $article;

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param \App\Models\Article $article

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, Article $article)

{

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

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

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param \App\Models\Article $article

\* @return \Illuminate\Http\Response

\*/

public function destroy(Article $article)

{

$article->delete();

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

}

* Por último añadiremos las rutas aunque esta vez en lugar de en fichero ***routes/web.php*** vamos a hacer en el fichero ***routes/api.php***:

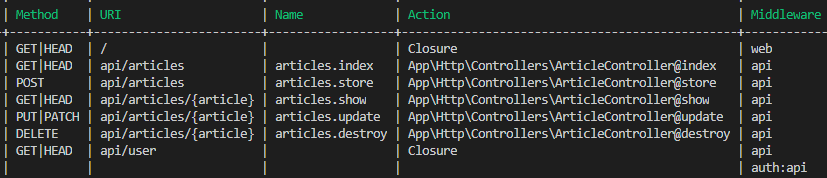
Route::middleware('auth:api')->get('/user', function (Request $request) {

return $request->user();

});

Route::apiResource('articles', ArticleController::class);

* Comprobamos que el registro de las rutas han ido:
  + ***php artisan route:list***
* Nos fijamos que ahora nuestras rutas van precedidas del contexto **/api**.



* Ya sólo nos queda probarlo, arranca el servidor de desarrollo de Laravel y abre en el navegador en la URL:
  + <http://localhost:8000/api/articles/>
* Continúa el tutorial de referencia y prueba también con ***Postman***.

### Ejemplo utilizando Eloquent: API Resources

* Documentación oficial (API):
  + <https://laravel.com/docs/8.x/eloquent-resources>
* Siempre podremos convertir modelos o colecciones de Eloquent a JSON sin embargo al crear una API, es posible que se necesite una capa de transformación entre los modelos Eloquent y las respuestas JSON. Por ejemplo, es posible que queremos mostrar ciertos atributos para un subconjunto de usuarios y no otros, o incluir siempre ciertas relaciones en la representación JSON de los modelos.
* Las clases de recursos de Eloquent son las que permiten transformar de manera fácil y expresiva los modelos y colecciones de modelos en JSON.
* Código de referencia para el ejercicio de ejemplo:
  + <https://github.com/Tony133/laravel-api-rest>
* Crea un nuevo modelo con nombre ***Book*** junto con sus ficheros asociados del ***controlador de tipo api, la migración, la siembra y la factoria*** todo desde una sóla línea de comandos:
  + *php artisan make:model Book -a --api*
* Comenzamos editando el fichero de **migración** situado en ***database/migrations/xxx\_\_create\_books\_table.php*** con el siguiente contenido:

public function up()

{

Schema::create('books', function (Blueprint $table) {

$table->id();

$table->string('title');

$table->string('price');

$table->string('author');

$table->string('editor');

$table->timestamps();

});

}

* Ahora editamos el fichero del **modelo** para señalar los campos que serán rellenables:

/\*\*

\* The attributes that are mass assignable.

\* @var array

\*/

protected $fillable = [

'title','price', 'author', 'editor',

];

* Continuamos editando el fichero de la **factoría** situado en ***database/factories/BookFactory.php***:

public function definition()

{

return [

'title' => $this->faker->sentence,

'price' => $this->faker->randomNumber(2),

'author' => $this->faker->name,

'editor'=> $this->faker->company,

];

}

* Seguidamente editamos el fichero de la **siembra** para el modelo Book situado en ***database/seeders/BookSeeder*** :

public function run()

{

// Truncamos la tabla para comenzar con una vacía.

Book::truncate();

Book::factory()->count(20)->create();

}

* También editamos el fichero de la **siembra** general ***DatabaseSeeder.php*** para no tener que hacer la siembra de forma individual:

public function run()

{

$this->call([

ArticleSeeder::class,

BookSeeder::class

]);

}

* Ejecutamos la migración con la siembra y comprobamos el resultado en la base de datos:
  + *php artisan migrate --seed*
* En este punto es hora de crear la clase de recurso para el modelo Book:
  + *php artisan make:resource BookResource*
* A continuación editamos el contenido de la clase creada en el fichero ***Http/Resources/BookResource.php***:

public function toArray($request)

{

return [

'id' => $this->id,

'title' => $this->title,

'price' => $this->price,

'author' => $this->author,

'editor'=> $this->editor,

];

}

* Seguimos creando un último fichero de la clase de tipo *form request* para el modelo Book:
  + *php artisan make:request BookRequest*
  + <https://laravel.com/docs/8.x/validation#form-request-validation>
* Una vez creado editamos el contenido de la clase situado en el fichero ***Http/Requests/BookRequest.php***:

/\*\*

\* Determine if the user is authorized to make this request.

\*

\* @return bool

\*/

public function authorize()

{

return true;

}

/\*\*

\* Get the validation rules that apply to the request.

\*

\* @return array

\*/

public function rules()

{

return [

'title' => 'required',

'price' => 'required',

'author' => 'required',

'editor' => 'required',

];

}

/\*\*

\* Get the error messages for the defined validation rules.

\*

\* @return array

\*/

public function messages()

{

return [

'title.required' => 'Please fill the title.',

'price.required' => 'Please fill the price.',

'author.required' => 'Please fill the author.',

'editor.required' => 'Please fill the editor.',

];

}

* Ahora toca editar el controlador ***BookController.php***, añadimos el siguiente contenido a los métodos creados automáticamente (asegúrate de modificar el objeto Request por el ***BookRequest*** en la llamada a los métodos):

namespace App\Http\Controllers;

use App\Models\Book;

use App\Http\Resources\BookResource;

use App\Http\Requests\BookRequest;

use Symfony\Component\HttpKernel\Exception\HttpException;

use Illuminate\Http\Request;

class BookController extends Controller

{

/\*\*

\* Display a listing of the resource.

\*

\* @return \Illuminate\Http\Response

\*/

public function index()

{

return BookResource::collection(Book::paginate(25));

}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(BookRequest $request)

{

try {

$book = Book::create($request->validated());

return new BookResource($book);

} catch(\Exception $exception) {

throw new HttpException(400, "Invalid data - {$exception->getMessage}");

}

}

/\*\*

\* Display the specified resource.

\*

\* @param \App\Models\Book $book

\* @return \Illuminate\Http\Response

\*/

public function show(Book $book)

{

return new BookResource($book);

}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param \App\Models\Book $book

\* @return \Illuminate\Http\Response

\*/

public function update(BookRequest $request, Book $book)

{

try {

$book->update($request->validated());

return new BookResource($book);

} catch(\Exception $exception) {

throw new HttpException(400, "Invalid data - {$exception->getMessage}");

}

}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param \App\Models\Book $book

\* @return \Illuminate\Http\Response

\*/

public function destroy(Book $book)

{

$book->delete();

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

}

}

* Por último añadimos las rutas en el fichero ***routes/api.php***:

Route::apiResource('books', BookController::class);

* Para probar el resultado abre en el navegador y ve a la URL:
  + <http://localhost:8000/api/books>
* Prueba con ***Postman*** también el resultado creando, editando y borrando elementos.

## Recursos Extras

* <https://rapidapi.com/blog/how-to-create-an-api-in-php/>