
UD10.10 – Laravel

API REST

2º CFGS
Desarrollo de Aplicaciones Web
2022-23

1.- Introducción

Antes de comenzar con la implementación de una API en un proyecto Laravel, se revisarán algunos conceptos vistos previamente en la unidad 9.

- API
- REST
- JSON

2.- API

Una **API** (Application Programming interface) es un **conjunto de funcionalidades** que los desarrolladores de una aplicación ponen a disposición de quienes las quieran utilizar.

En muchas ocasiones los propios desarrolladores utilizan sus propias API's en los proyectos que desarrollan.

Por ejemplo, en aplicaciones web se suelen realizar **peticiones AJAX a la propia API** para acceder a las funcionalidades de una manera sencilla.

También es habitual **poner a disposición del público la API** para así que otros desarrolladores puedan incorporar funcionalidades de tu aplicación en su aplicación.

Por ejemplo, incrustar publicaciones de Twitter, Instagram... en otras aplicaciones web.

3.- REST

REST → **RE**presentational **State Transfer**

Es una arquitectura que se basa en **peticiones HTTP** para trabajar con los datos de la aplicación web (almacenados en la base de datos).

Fundamentos:

- Protocolo cliente/servidor sin estado.
- Operaciones bien definidas: **GET, POST, PUT, PATCH, DELETE**
- Sintaxis universal.

4.- JSON

JSON: JavaScript Object Notation

Permite describir objetos con notación de texto.

Es mucho más ligero y sencillo que XML.

Se ha convertido en la alternativa a XML por el uso de AJAX.

```
{
  "libro": [
    {
      "id": "01",
      "lenguaje": "Java",
      "edición": "tercera",
      "autor": "Herbert Schildt"
    },
    {
      "id": "07",
      "lenguaje": "C++",
      "edición": "segunda",
      "autor": "E.Balagurusamy"
    }
  ]
}
```

5.- Controladores para la API

Aunque se pueden usar los controladores normales (usados hasta ahora) para tratar las peticiones de a la API, es recomendable tener el proyecto organizado y crear controladores específicos para estas funciones.

Para crear un controlador para la API se usa el siguiente comando:

```
# php artisan make:controller Api/nombreControladorApiController --api --model=nombreModelo
```

Por ejemplo:

```
# php artisan make:controller Api/BookApiController --api --model=Book
```

La opción **--api** indica que es un controlador que se usará en la API.

La opción **--model** indica que el controlador irá ligado a ese modelo.

5.- Controladores para la API

Por ejemplo:

```
# php artisan make:controller Api/BookApiController --api --model=Book
```

El comando anterior creará si no existe la carpeta **Api** dentro de la carpeta de controladores y además creará el archivo **BookApiController.php**:

```
app/Http/Controllers/Api/BookApiController.php
```

5.- Controladores para la API

Los controladores tipo API contienen los siguientes métodos:

- **index**
- **store**
- **show**
- **update**
- **destroy**

Los métodos **edit** y **create** no tienen sentido En una API ya que sirven para mostrar formularios y la API solo devuelve datos en formato JSON.

Si se necesitan se pueden crear métodos extra como por ejemplo para obtener un subconjunto de registros según alguna condición.

6.- Rutas para la API

Las rutas para la API se deben introducir en el archivo **routes/api.php** de la misma manera y con las mismas reglas con las que se añaden al archivo de rutas web.php.

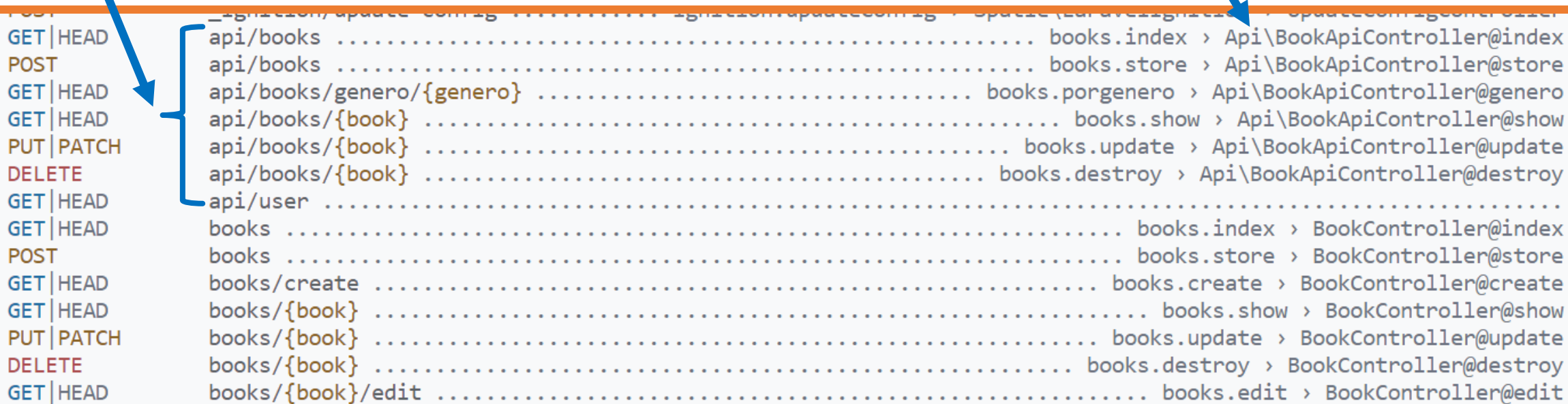
```
Route::get('books/genero/{genero}', [BookApiController::class, 'genero'])  
    ->name('books.porgenero');
```

Si se tiene que añadir una ruta con todos los métodos típicos de la API:

```
Route::apiResource('books', BookApiController::class);
```

6.- Rutas para la API

Al crear las rutas en el archivo **routes/api.php** automáticamente todas las esas rutas tendrán el prefijo **api**.



The screenshot shows the contents of the `routes/api.php` file. A blue bracket on the left groups the routes for the `api/books` resource, which include `index`, `store`, `show`, `update`, and `destroy`. A blue arrow points from the text above to the `api/books` routes. Another blue arrow points from the text above to the `api/books/{book}` routes. The routes are listed as follows:

```
GET|HEAD api/books ..... books.index > Api\BookApiController@index
POST api/books ..... books.store > Api\BookApiController@store
GET|HEAD api/books/genero/{genero} ..... books.porgenero > Api\BookApiController@genero
GET|HEAD api/books/{book} ..... books.show > Api\BookApiController@show
PUT|PATCH api/books/{book} ..... books.update > Api\BookApiController@update
DELETE api/books/{book} ..... books.destroy > Api\BookApiController@destroy
GET|HEAD api/user .....
GET|HEAD books ..... books.index > BookController@index
POST books ..... books.store > BookController@store
GET|HEAD books/create ..... books.create > BookController@create
GET|HEAD books/{book} ..... books.show > BookController@show
PUT|PATCH books/{book} ..... books.update > BookController@update
DELETE books/{book} ..... books.destroy > BookController@destroy
GET|HEAD books/{book}/edit ..... books.edit > BookController@edit
```

7.- Enviando respuestas

Las respuestas de las **peticiones a la API** deben devolver datos en **formato JSON**.

Cuando se devuelven **colecciones** u **objetos** en las consultas a la base de datos mediante Eloquent **se consigue directamente código JSON**.

```
return Post::all();
```

```
return Post::where('visibility', 1)->get();
```

```
return $post
```

```
[
  {
    "id": 1,
    "nombre": "Entrada cine",
    "empresa": "Kinepolis",
    "precio": 9,
    "descuento": 10,
    "fechalimite": "2022-01-28",
    "created_at": "2022-01-12T09:49:52.000000Z",
    "updated_at": "2022-01-12T09:49:52.000000Z"
  },
  {
    "id": 2,
    "nombre": "Lego Sonic",
    "empresa": "Lego",
    "precio": 69.99,
    "descuento": 5,
    "fechalimite": "2022-01-26",
    "created_at": "2022-01-12T09:49:52.000000Z",
    "updated_at": "2022-01-12T09:49:52.000000Z"
  },
  {
    "id": 3,
    "nombre": "Teclado K70",
    "empresa": "Corsair",
    "precio": 140,
    "descuento": 12.5,
    "fechalimite": "2022-01-31",
    "created_at": "2022-01-12T09:49:52.000000Z",
    "updated_at": "2022-01-12T09:49:52.000000Z"
  }
]
```

7.- Enviando respuestas

Se pueden devolver los datos que se quiera pero siempre se deben devolver en formato JSON.

Si se devuelve un array automáticamente se convertirá en JSON.

Por ejemplo, si se quiere devolver un post junto con su autor se podría hacer de la siguiente manera:

```
public function show(Post $post)
{
    return ['post' => $post, 'writer' => $post->writer];
}
```

7.- Enviando respuestas

Si no se indica lo contrario una petición a la API siempre va a devolver el código de respuesta **200**.

Para un correcto funcionamiento de la API se debe gestionar los códigos para que dependiendo de la operación realizada se devuelva el código correcto:

200 → todo correcto.

201 → se añade un elemento a la base de datos.

204 → cuando no se devuelven datos (null) pero se ha realizado una operación.

404 → ha habido un error.

Para devolver los datos junto con el código de respuesta se usa el método **response**.

```
return response()->json($post, 200);
```

7.- Enviando respuestas

Ejemplos:

```
public function show(Post $post)
{
    return response()->json($post, 200);
}
```

```
public function index()
{
    $posts = Post::all();
    return response()->json($posts, 200);
}
```

7.- Enviando respuestas

Ejemplos:

Al guardar/actualizar un registro nuevo es habitual devolver el propio registro además del código. Si se actualiza el código de respuesta es 201.

```
public function store(Request $request)
{
    $post = new Post();
    $post->title = $request->get('titulo');
    $post->slug = Str::slug($post->title);
    $post->content = $request->get('contenido');
    $post->visibility = $request->has('visibilidad') ? 1 : 0;
    $post->writer()->associate(Writer::findOrFail($request->get('autor')));
    $post->save();

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

7.- Enviando respuestas

Ejemplos:

Al eliminar un registro se puede devolver null y el código de respuesta:

```
public function destroy(Post $post)
{
    $post->delete();
    return response()->json(null, 204);
}
```

O se puede devolver el propio registro (por si se quisiera deshacer la operación) y el código de respuesta:

```
public function destroy(Post $post)
{
    $post->delete();
    return response()->json($post, 204);
}
```


8.- Respuestas de error

Cuando se produce un error en alguna petición de la API la respuesta también debe ser en formato JSON y además se debe mostrar el código de error.

Se debe añadir al método **register** del archivo **app/Exceptions/Handler.php** el código:

```
$this->renderable(function (Throwable $exception) {  
    if (request()->is('api')) {  
        if($exception instanceof ModelNotFoundException)  
            return response()->json(['error' => 'Recurso no encontrado'], 404);  
        else if($exception instanceof ValidationException)  
            return response()->json(['error' => 'Datos no válidos'], 400);  
        else if (isset($exception)) {  
            return response()->json(['error' => 'Error: '. $exception->getMessage()], 500);  
        }  
    }  
});
```

De esta manera los errores en las peticiones a la api se devolverán en formato JSON indicando un mensaje y el código de error.