API RestFul

Symfony

Fernando Díaz-Alonso Dorado y Lluís Alandete

Florida Universitària

API Symfony

API Básica

Vamos a crear una API básica a partir de un proyecto de Symfony básico en el que vamos a ir añadiendo los complementos que necesitamos para crear la API.

Los principales pasos son:

- Crear proyecto básico de Symfony
- Configurar el proyecto.
- Crear las entidades y los repositorios necesarios para nuestra API.
- Crear los controladores para recoger las peticiones de nuestra API.
- Modificar cada controlador para definir las rutas y métodos y poder ejecutar un CRUD con nuestra API.
- Usar la API con una API REST CLIENT

API Symfony

Crear Proyecto básico

```
symfony new api --version="7.4.x" --webapp --no-git
```

Notar que usamos **--webapp** aunque es un API. Porque instala más paquetes y algunos los podemos necesitar, por ejemplo “MakerBundle” que facilita la generación automática de controladores y entidades.

Podríamos usar **--minimal** pero quizás luego habría que instalar dependencias manualmente.

API Symfony

Configuración

Una vez instalado configuraremos la base de datos en el archivo .env y la crearemos:

```
DATABASE_URL="mysql://userbd:passbd@rutabd:3306/namebd?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
```

userbd →usuario BB.DD. passbd → contraseña BB.DD.

Rutabd → ruta de la BB.DD. namebd: → nombre BB.DD.

serverVersion y charset debemos tenerlos en cuenta.

```
DATABASE_URL="mysql://root:root@127.0.0.1:3307/api?serverVersion=10.11.2-MariaDB&charset=utf8mb4"
```

Creamos la base de datos:

```
php bin/console doctrine:database:create
```

API Symfony

Crear las entidades y los repositorios

Una vez que ya tenemos configurado lo mínimo para poder trabajar vamos a crear las entidades y los repositorios asociados a ellas.

```
php bin/console make:entity Student
```

Actualizamos el schema:

```
php bin/console doctrine:schema:update --dump-sql --force
```

API Symfony

Crear el controlador para las peticiones

Tras esto vamos a empezar con los CRUD para la API. Pero no vamos a utilizar el comando ***make:crud*** porque no vamos a necesitar el formulario.

Por tanto, para ello vamos a crear cada controlador que necesitemos con el comando ***make:controller*** y modificando después el controlador para usarlo mediante JSON.

php bin/console make:controller ApiStudentController

```php
final class ApiStudentController extends AbstractController
{
    #[Route('/api/student', name: 'app_api_student')]
    public function index(): Response
    {
        return $this->render( view: 'api_student/index.html.twig', [
            'controller_name' => 'ApiStudentController',
        ]);
    }
}
```

API Symfony

Definir las rutas y métodos del API REST I

Posteriormente vamos ha establecer como ruta general del controlador la que tenga que ver con la Entidad y posteriormente sub-enrutar a cada acción.

```php
#[Route('/api/students', name: 'api_students_')]
class ApiStudentController extends AbstractController
{
```

Los diferentes Métodos HTTP que usaremos en la API son:

- Crear / Insertar datos en la BB.DD. → POST
- Leer datos de la BB.DD. → GET
- Actualizar / Modificar datos de la BB.DD. → PUT / PATCH
- Eliminar / Borrar datos de la BB.DD. → DELETE

API Symfony

Definir las rutas y métodos del API REST II

Ahora modificamos el Controller para poder recibir las diferentes rutas de nuestra API con los métodos HTTP.

```php
#[Route('/api/students', name: 'api_students_')]
class ApiStudentController extends AbstractController
{
    #[Route('', methods: ['GET'], name: 'list')]
    public function list(EntityManagerInterface $em): JsonResponse{...}

    #[Route('/{id}', methods: ['GET'], name: 'show')]
    public function show(Student $student): JsonResponse{...}

    #[Route('', methods: ['POST'], name: 'create')]
    public function create(Request $request, EntityManagerInterface $em): JsonResponse{...}

    #[Route('/{id}', methods: ['PUT', 'PATCH'], name: 'update')]
    public function update(Request $request, Student $student, EntityManagerInterface $em): JsonResponse{...}

    #[Route('/{id}', methods: ['DELETE'], name: 'delete')]
    public function delete(Student $student, EntityManagerInterface $em): JsonResponse{...}
}
```

Ruta base de acceso al API

Devuelve la lista de alumnos - GET

Devuelve un alumno por id - GET

Crea un alumno – POST

Editar un alumno – PUT/PATCH

Elimina un alumno - DELETE

API Symfony

Leer datos (GET) I

Crearemos un nuevo método ***list*** para obtener todos los resultados:
y como nombre de la ruta: ***api_name_entity_list.***

En este caso montamos un array con todos los resultados y devolvemos un JSON.

```php
#[Route('', methods: ['GET'], name: 'list')]
public function list(EntityManagerInterface $em): JsonResponse
{
    $students = $em->getRepository(Student::class)->findAll();
    $data = [];

    foreach ($students as $student) {
        $data[] = [
            'id' => $student->getId(),
            'name' => $student->getName(),
            'age' => $student->getAge(),
            'course' => $student->getCourse(),
            'registration_date' => $student->getRegistrationDate()->format( format: 'Y-m-d H:i:s'),
        ];
    }

    return new JsonResponse($data);
}
```

API Symfony

Leer datos (GET) II

Crearemos un nuevo método ***show*** para obtener un resultado a partir de la id que se passara como subruta y como nombre de la ruta:
api_name_entity_show.

En este caso vamos a recibir por Autowiring un objeto entidad que formatearemos en JSON para mostrar y devolverlo.

```php
#[Route('/{id}', methods: ['GET'], name: 'show')]
public function show(Student $student): JsonResponse
{
    $data = [
        'id' => $student->getId(),
        'name' => $student->getName(),
        'age' => $student->getAge(),
        'course' => $student->getCourse(),
        'registration_date' => $student->getRegistrationDate()->format( format: 'Y-m-d H:i:s'),
    ];

    return new JsonResponse($data);
}
```

API Symfony

Crear / Insertar datos (POST)

Crearemos el nuevo método ***create*** que tendrá como subruta: ***/create*** y como nombre de la ruta: ***api_name_entity_create. Notar que la ruta no cambia, cambia el método.***

```php
#[Route('', methods: ['POST'], name: 'create')]
public function create(Request $request, EntityManagerInterface $em): JsonResponse
{
    $data = json_decode($request->getContent(), associative: true);

    $student = new Student();
    $student->setName($data['name']);
    $student->setAge($data['age']);
    $student->setCourse($data['course']);
    $student->setRegistrationDate(new \DateTime());

    $em->persist($student);
    $em->flush();

    return new JsonResponse(['status' => 'Student created'], status: 201);
}
```

Devolveremos la respuesta de creado para obtener un código 2XX y el mensaje en JSON.

API Symfony

Actualizar / Modificar datos (PUT / PATCH)

Vamos a crear un método muy parecido al de crear, pero en este caso lo llamamos ***update*** que tendrá como subruta: ***/update/{id}*** y como nombre de la ruta: ***api_name_entity_update.***

IMPORTANTE: Usaremos PUT cuando modificamos todo el registro y PATCH cuando únicamente modificamos una parte.

```php
#[Route('/{id}', methods: ['PUT', 'PATCH'], name: 'update')]
public function update(Request $request, Student $student, EntityManagerInterface $em): JsonResponse
{
    $data = json_decode($request->getContent(), associative: true);

    if (isset($data['name'])) {
        $student->setName($data['name']);
    }
    if (isset($data['age'])) {
        $student->setAge($data['age']);
    }
    if (isset($data['course'])) {
        $student->setCourse($data['course']);
    }

    $em->flush();

    return new JsonResponse(['status' => 'Student updated']);
}
```

API Symfony

Eliminar / borras datos (DELETE)

Muy parecido al de leer datos, porque una parte es igual. Crearemos el nuevo método ***remove*** que tendrá como subruta: ***/delete/{id}*** y como nombre de la ruta: ***api_name_entity_delete.***

En este caso vamos a recibir por Autowiring un objeto entidad que formatearemos en JSON para mostrar y devolveremos el código 2XX.

```php
#[Route('/{id}', methods: ['DELETE'], name: 'delete')]
public function delete(Student $student, EntityManagerInterface $em): JsonResponse
{
    $em->remove($student);
    $em->flush();

    return new JsonResponse(['status' => 'Student deleted']);
}
```

API Symfony

API CLIENT

Existen muchas herramientas para poder probar nuestras APIs, por ejemplo: Postman,...

Nosotros utilizaremos **Bruno** que es open-source y lo tenemos en nuestro ordenador.

Bruno

Reinventing the API Client

Instalar: https://www.usebruno.com/downloads

Crear colecciones para comprobar cada llamada

Realizar llamadas y así poder hacer pruebas