Tema 3. Práctica 5.

Spring: API Rest con BBDD.

1. Objetivos

En esta práctica vamos a hacer un API REST partiendo de la implementación web con Spring realizada en Acceso a Datos. Cuando la tengamos realizada haremos un cliente por consola que, con programación reactiva, acceda a dicha API. El objetivo es aprender a crear y consumir APIs sencillas con una base de datos para la persistencia de la información.

2. Creando la API

Partiendo de las ideas de la práctica anterior, vamos a crear la API para crear, actualizar, borrar y mostrar pilotos. La API tendrá los siguientes end points:

GET /api/pilotos (El body de la respuesta contendrá el JSON de todos los pilotos).
POST /api/pilotos (El body de la petición contendrá el JSON del nuevo piloto).
GET /api/pilotos/{id} (El get de la respuesta contendrá el JSON de un piloto).
PUT /api/pilotos/{id} (El body de la petición contendrá el JSON del piloto a modificar).
DELETE /api/pilotos/{id}

Aquí tienes un esqueleto de cómo quedarían los controladores para una API de bicicletas, tendrás que adaptar este ejemplo para que funcione para los pilotos. Te recomiendo empezar por los métodos que usan GET pues son más sencillos.

```
@GetMapping("/bike/{id}")
public ResponseEntity<Bike> getBike(@PathVariable String id) {
   Bike bike = bikeService.findBike(id);
   return ResponseEntity.ok(bike);
}
@DeleteMapping("/bike/{id}")
public ResponseEntity<Bike> removeBike(@PathVariable String id){
   Bike bike = bikeService.deleteBike(id);
   return new ResponseEntity<>(bike, HttpStatus.OK);
}
@PostMapping("/bikes")
public ResponseEntity<Bike> addBike(@Valid @RequestBody Bike bike)
{
   Bike newBike = bikeService.addBike(bike);
   return ResponseEntity.ok(newBike);
}
```

Una vez tenga creada la API comprueba su correcto funcionamiento desde el navegador Firefox y la extensión RESTED, mostrando, creando, borrando y actualizando registros.

3. Crear un cliente por terminal

Crea una aplicación de consola que acceda de manera reactiva a la API que acabamos de crear. Para hacerlo utilizaremos Webflux, la implementación de clientes web de Spring.

La aplicación debe tener 5 opciones:

- Mostrar a todos los pilotos.
- Mostrar un piloto dado un id.
- Crear un piloto con nuevos datos.
- Actualizar un piloto dado un id concreto
- Borrar un piloto dado un id.

El código para recuperar un piloto de la API y mostrarlo por pantalla es el siguiente. Comprueba su funcionamiento y partiendo de él como base implementa toda la funcionalidad de la aplicación.

```
WebClient client = WebClient.create("http://localhost:8080");
Flux<Piloto> pilotoMono = client.get()
       .uri("/pilotos/{id}", "1")
       .retrieve()
       .bodyToMono(Piloto.class);
pilotoMono.subscribe(System.out::println);
```