ACCESO A DATOS

UD06: PRÁCTICA 1 - SPRING BOOT

Realiza un proyecto Java denominado **adt6_practica1** que contenga todo lo que se pide a continuación. Para ello crea un base de datos con el mismo nombre en MySQL.

Gestión de vuelos

Crear una API Rest que ofrezca servicios web para la gestión de vuelos. La aplicación tendrá una base de datos de vuelos donde almacenará:

- origen, destino, precio, número de escalas y compañía.

Deberá ofrecer las siguientes operaciones:

1. Búsqueda de todos los vuelos.
2. Registro de un nuevo vuelo.
3. Modificar un vuelo.
4. Dar de baja un vuelo.

Métodos específicos:

5. Dar de baja todos los vuelos a un destino determinado.
6. Búsqueda de vuelos, pudiendo filtrar por origen, destino y número de escalas.

Para conocer mejor las opciones que podemos hacer en los métodos específicos:

- https://docs.spring.io/spring-data/jpa/docs/current-SNAPSHOT/reference/html/#reference
    - 3.1. JPA Repositories
        - 3.1.3. Query Methods
            - Query Creation
            - Using @Query
            - Modifying Queries

Datos de ejemplo:

```
INSERT INTO vuelo (origen, destino, precio, num_escalas, compania)
VALUES('Valencia', 'Paris', 56.5, 0, 'Iberia');
INSERT INTO vuelo (origen, destino, precio, num_escalas, compania)
VALUES('Valencia', 'Londres', 99, 0, 'Ryanair');
INSERT INTO vuelo (origen, destino, precio, num_escalas, compania)
VALUES('Valencia', 'Washington', 560.5, 2, 'Iberia');
INSERT INTO vuelo (origen, destino, precio, num_escalas, compania)
VALUES('Madrid', 'Roma', 30, 0, 'AlItalia');
INSERT INTO vuelo (origen, destino, precio, num_escalas, compania)
VALUES('Barcelona', 'Paris', 10, 0, 'AirFrance');
```

AYUDA PARA EL MÉTODO DE DAR DE BAJA TODOS LOS VUELOS A UN DETERMINADO DESTINO

En el repository:

- Crear un método específico adicional a los que nos proporciona CrudRepository o JpaRepository. Para ello utilizaremos la nomenclatura de Spring Data JPA.
- Las anotaciones que necesitamos:
  - @Query para especificar la consulta JPQL (Java Persistence Query Language) que se ejecutará en la base de datos.
  - @Modifying indica que el método realiza una operación de escritura en la base de datos. Es decir, si la operación es de tipo INSERT, UPDATE o DELETE tendremos que añadir esta anotación.

En la implementación del service:

- Cuando sobreescribimos el método tendremos que anotarlo con @Transactional, esta anotación se utiliza para marcar un método o clase como transaccional, lo que significa que todas las operaciones realizadas dentro de ese método o clase se realizarán dentro de una transacción. Si una operación falla, todas las operaciones realizadas dentro de la transacción se deshacen.

En el controller:

- Usar la anotación @DeleteMapping para mapear la solicitud DELETE del método.
- Pasarle por parámetro el destino que queremos eliminar.
- Podríamos validar si existe algún vuelo o vuelos con ese destino para eliminarlo:
  - Si existen vuelos, devolver una respuesta 204 (No Content) como respuesta a que se ha eliminado con éxito.
  - Si no hay vuelos, devolver una respuesta 404 (Not Found).

AYUDA PARA EL MÉTODO DE BÚSQUEDA DE VUELOS CON FILTROS

El problema a abordar se puede diseñar de diferentes formas:

1. Podemos tener un método específico para cada filtro, para ello debemos:
   a. En el repository: crear un método específico para cada consulta.
      i. Spring Data JPA: realiza automáticamente la generación de consultas SQL a partir de los nombres de los métodos en el repositorio (sin escribir la JPQL):
         - *findByOrigenAndDestinoAndNumEscalas(...)*
   b. En el service:
      i. Añadir la cabecera de cada uno a la interfaz.
      ii. Implementar cada uno de los métodos en la implementación del servicio.
   c. En el controller:
      i. Crear un endpoint utilizando los métodos específicos.

2. No crear ningún método específico ni en repository ni en service. En el controlador creamos un endpoint utilizando el método de obtener todos los vuelos:

    a. En este caso, tenemos que filtrar los datos y crear un nuevo listado con las condiciones que deseamos.

```
@RestController
@RequestMapping("/vuelos")
public class VueloController {

    @GetMapping("/search")
    public ResponseEntity<List<Object>> search(
              @RequestParam(value = "param1", required = false) String param1,
              @RequestParam(value = "param2", required = false) String param2,
              @RequestParam(value = "param3", required = false) String param3) {

        // Listado de resultados finales
        List<Object> results = new ArrayList<>();

        // Obtenga todos los datos
        List<Object> allData = getAllData();

        // Recorrer listado con todos y añadir a resultados según condiciones
        for (Object data : allData) {
            if (param1 != null && data.getParam1().equals(param1)) {
                results.add(data);
            } else if (param2 != null && data.getParam2().equals(param2)) {
                results.add(data);
            } else if (param3 != null && data.getParam3().equals(param3)) {
                results.add(data);
            }
            ...
        }

        return ResponseEntity<>(results, HttpStatus.OK);
    }
}
```

En este ejemplo, se itera a través de una lista de todos los datos disponibles y se agrega un elemento a la lista de resultados si cualquiera de los parámetros proporcionados coincide con los valores en ese elemento. La lógica puede ser personalizada según sus requisitos específicos.