First REST API with Spring Boot

Introduction

In this exercise, you will create a simple REST API using Spring Boot without a database. You will create a controller that handles HTTP requests and returns JSON responses. You will try out different HTTP methods such as `GET`, `POST`, `PUT`, and `DELETE`, and how to extract **path variables** and **query parameters** in the controller methods.

Step 1: Create a new Spring Boot project

1. Create a new **Spring Boot** project in **IntelliJ** with the following:
    - **Build tool:** `Maven`
    - **Group:** `dk.ek`
    - **Artifact:** `<YOUR_STUDENT_ID>`
    - **Java version:** `25`
    - **Packaging:** `Jar`
2. Add the following dependencies:
    - **`Spring Web`**
3. Add a new package in the `src/main/java/dk/ek/<YOUR_STUDENT_ID>` directory named `courseapi`.
4. Create the following packages inside the `courseapi` package:

```
src.main.java.dk.ek.<YOUR_STUDENT_ID>
    └── courseapi
        ├── controller
        ├── model
        ├── service
        └── repository
```

Step 2: Create a `Course` class

1. Create a new class named `Course` in the `model` package with the following fields:
    - `id`: Long
    - `name`: String
    - `description`: String
    - `active`: Boolean
    - `startDate`: LocalDate
    - `endDate`: LocalDate
    - **Add two constructors** - a no-args constructor and an all-args constructor.
    - **Add getters and setters** for all fields.

Step 3: Create a `CourseController` class

1. Create a new class named `CourseController` in the `controller` package.

```
@RestController
@RequestMapping("/api/courses")
public class CourseController {

    private List<Course> courses = new ArrayList<>();
    private Long nextId = 1L;

    public CourseController() {
        // TODO Add Some sample courses to the list
    }

    @GetMapping
    public ResponseEntity<List<Course>> getAllCourses() {
        return ResponseEntity.ok(courses);
    }
}
```

2. Add some sample courses to the `courses` list in the constructor.
3. Try running the application
4. Test the API using your browser at `http://localhost:8080/api/courses` and try using **curl** from the terminal:

```
curl http://localhost:8080/api/courses
```

Step 4: Extracting Path variables

1. Add a new method to the `CourseController` class to get a course by its ID:

```
@GetMapping("/{id}")
public ResponseEntity<Course> getCourseById(@PathVariable Long id) {
    // TODO Find the course by id and return it

    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
}
```

   - The `@PathVariable` annotation is used to extract the value of the `id` path variable from the URL.
   - `return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);` is used to return a `404 Not Found` response if the course is not found.

2. Test the API using your browser at `http://localhost:8080/api/courses/{id}` and try using **curl** from the terminal:

```
curl http://localhost:8080/api/courses/1
```

   - Try with an ID that does not exist to see the 404 response.

Step 5: Adding a new course: `@PostMapping`

1. Add a new method to the `CourseController` class to add a new course:

```
@PostMapping
public ResponseEntity<Course> addCourse(@RequestBody Course course) {
    // Simple ID generation
    course.setId(nextId++);
    // TODO Add the course to the list and return it
    return ResponseEntity.status(HttpStatus.CREATED).body(course);
}
```

   - The `@RequestBody` annotation is used to bind the HTTP request body to a transfer or domain object, enabling automatic deserialization of the incoming JSON into a Java object.
   - We use `ResponseEntity.status(HttpStatus.CREATED).body(course)` to return the added course with a `201 Created` status.

2. Test the API using **curl** from the terminal:

```
curl -X POST http://localhost:8080/api/courses -H "Content-Type:
application/json" -d '{"name":"Course 3","description":"Description
3","active":true,"startDate":"2023-01-01","endDate":"2023-06-01"}'
```

   - You should see the added course in the response.

Step 6: Updating a course: `@PutMapping`

1. Add a new method to the `CourseController` class to update an existing course:

```
@PutMapping("/{id}")
public ResponseEntity<Course> updateCourse(@PathVariable Long id,
@RequestBody Course course) {
    // TODO Update the course in the list and return it (only if it
exists)

    // If the course is not found, return a 404 response
    return ResponseEntity.status(HttpStatus.NOT_FOUND).body(null);
}
```

- Implement the method to update the course in the `courses` list. If the course is not found, return a `404 Not Found` response.

2. Test the API using **curl** from the terminal:

```
curl -X PUT http://localhost:8080/api/courses/1 -H "Content-Type:
application/json" -d '{"id":1,"name":"Updated Course
1","description":"Updated Description
1","active":false,"startDate":"2023-01-01","endDate":"2023-06-01"}'
```

Step 7: Deleting a course: `@DeleteMapping`

1. Add a new method to the `CourseController` class to delete a course by its ID:

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteCourse(@PathVariable Long id) {
    // TODO Delete the course from the list (if the course exists else
return 404)
    return ResponseEntity.status(HttpStatus.NO_CONTENT).build();
}
```

2. Implement the method to delete the course from the `courses` list. If the course is not found, return a `404 Not Found` response. If the course is deleted successfully, return a `204 No Content` response.

3. Test the API using **curl** from the terminal:

```
curl -X DELETE http://localhost:8080/api/courses/1
```

Step 8: Query parameters: `@RequestParam`

1. To filter the courses based on the course name, refactor the `getAllCourses` method to accept an optional query parameter `name`:

```
@GetMapping
public ResponseEntity<List<Course>>
getAllCourses(@RequestParam(required = false) String name) {
    if (name != null && !name.isEmpty()) {
        List<Course> filteredCourses = new ArrayList<>();
        // TODO Filter courses by name if the name parameter is
provided

        return ResponseEntity.ok(filteredCourses);
    }
```

```
        return ResponseEntity.ok(courses);
    }
```

2. Implement the method to filter the courses by name if the `name` parameter is provided. If the `name` parameter is not provided, return all courses.

3. Test the API using your browser at `http://localhost:8080/api/courses?name=PROG2` and try using **curl** from the terminal:

```
curl "http://localhost:8080/api/courses?name=PROG2"
```

    - You should see the filtered courses in the response.