We'll use a simple scenario involving **Student** and **Course** entities with a many-to-many relationship.

The goal is to perform various operations using Postman.

**Exercise: Student-Course Management**

Entities:

1. **Student** Entity:
   * Attributes: **id**, **name**, **email**
   * Relationship: Many-to-Many with **Course**
2. **Course** Entity:
   * Attributes: **id**, **title**, **description**
   * Relationship: Many-to-Many with **Student**

**Tasks:**

1. **Create a Student:**
   * **Method:** POST
   * **URL:** **http://localhost:8080/students**
   * **Request Body:**

{

"name": "John Doe",

"email": "john.doe@example.com"

}

**Expected Response:** The created student with an assigned ID.

**2. Create a Course:**

* **Method:** POST
* **URL:** **http://localhost:8080/courses**
* **Request Body:**

{

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming"

}

**Expected Response:** The created course with an assigned ID.

**3. Enroll Student in a Course:**

* **Method:** PUT
* **URL:** **http://localhost:8080/students/{studentId}/enroll**
* **Request Body:**

{

"courseId": 1

}

**Expected Response:** The updated student object with the enrolled course.

**3. Enroll Student in a Course:**

* **Method:** PUT
* **URL:** **http://localhost:8080/students/{studentId}/enroll**
* **Request Body:**

{

"courseId": 1

}

**Expected Response:** The updated student object with the enrolled course.

**4. Get All Students:**

* + **Method:** GET
  + **URL:** **http://localhost:8080/students**
  + **Expected Response:** A list of all students with their details.

**5. Get Student by ID:**

* + **Method:** GET
  + **URL:** **http://localhost:8080/students/{studentId}**
  + **Expected Response:** Details of the student with the specified ID.

**6. Get All Courses:**

* + **Method:** GET
  + **URL:** **http://localhost:8080/courses**
  + **Expected Response:** A list of all courses with their details.

**7. Update Student's Details:**

* + **Method:** PUT
  + **URL:** **http://localhost:8080/students/{studentId}**
  + **Request Body:**

{

"name": "Updated Name",

"email": "updated.email@example.com"

}

**Expected Response:** The updated student object.

**7. Delete a Student:**

* + **Method:** DELETE
  + **URL:** **http://localhost:8080/students/{studentId}**
  + **Expected Response:** No content (204 status code).

**8. Delete a Course:**

* + **Method:** DELETE
  + **URL:** **http://localhost:8080/courses/{courseId}**
  + **Expected Response:** No content (204 status code).

**Answers:**

1. **Create a Student:**
   * **Response:**

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"courses": []

}

1. **Create a Course:**
   * **Response:**

{

"id": 1,

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming",

"students": []

}

1. **Enroll Student in a Course:**
   * **Response:**

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"courses": [

{

"id": 1,

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming",

"students": [1]

}

]

}

1. **Get All Students:**
   * **Response:**

[

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com", "courses":

[

{

"id": 1,

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming",

"students": [1]

}

]

}

]

1. **Get Student by ID:**
   * **Response:**

{

"id": 1,

"name": "John Doe",

"email": "john.doe@example.com",

"courses": [

{

"id": 1,

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming",

"students": [1]

}

]

}

1. **Get All Courses:**
   * **Response:**

[

{

"id": 1,

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming",

"students": [1]

}

]

1. **Update Student's Details:**
   * **Response:**

{

"id": 1,

"name": "Updated Name",

"email": "updated.email@example.com",

"courses": [

{

"id": 1,

"title": "Introduction to Programming",

"description": "A beginner-friendly course on programming",

"students": [1]

}

]

}

1. **Delete a Student:**
   * **Response:** No content (204 status code).
2. **Delete a Course:**
   * **Response:** No content (204 status code).

Note: Ensure that you replace **{studentId}** and **{courseId}** in the URLs with the actual IDs obtained from the responses. The exercise demonstrates basic CRUD operations and many-to-many relationships between **Student** and **Course** entities.

**Answer:**

Below is a simple Spring Boot application that includes the entities, repositories, controllers, and configurations necessary for the exercise. The application uses an H2 in-memory database and Spring Data JPA for data persistence. Remember to adapt this example to your specific requirements in a real-world scenario.

src

└── main

├── java

│ └── com

│ └── (spring.data.jpa.demo)

│ ├── controller

│ │ ├── CourseController.java

│ │ └── StudentController.java

│ ├── model

│ │ ├── Course.java

│ │ └── Student.java

│ ├── repository

│ │ ├── CourseRepository.java

│ │ └── StudentRepository.java

│ └── Application.java

└── resources

└── application.properties

**Course.java Entity:**

import com.fasterxml.jackson.annotation.JsonBackReference;

import javax.persistence.\*;

import java.util.List;

@Entity

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String description;

@ManyToMany(mappedBy = "courses")

@JsonBackReference

private List<Student> students;

// getters and setters

}

**Student.java Entity:**

import com.fasterxml.jackson.annotation.JsonManagedReference;

import javax.persistence.\*;

import java.util.List;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

@ManyToMany

@JoinTable(

name = "student\_course",

joinColumns = @JoinColumn(name = "student\_id"),

inverseJoinColumns = @JoinColumn(name = "course\_id"))

@JsonManagedReference

private List<Course> courses;

// getters and setters

}

**CourseRepository.java:**

import org.springframework.data.repository.CrudRepository;

public interface CourseRepository extends CrudRepository<Course, Long> {

}

**StudentRepository.java:**

import org.springframework.data.repository.CrudRepository;

public interface StudentRepository extends CrudRepository<Student, Long> {

}

**CourseController.java:**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/courses")

public class CourseController {

@Autowired

private CourseRepository courseRepository;

@GetMapping

public List<Course> getAllCourses() {

return (List<Course>) courseRepository.findAll();

}

@GetMapping("/{id}")

public Course getCourseById(@PathVariable Long id) {

return courseRepository.findById(id).orElse(null);

}

@PostMapping

public Course addCourse(@RequestBody Course course) {

return courseRepository.save(course);

}

@PutMapping("/{id}")

public Course updateCourse(@PathVariable Long id, @RequestBody Course course) {

course.setId(id);

return courseRepository.save(course);

}

@DeleteMapping("/{id}")

public void deleteCourse(@PathVariable Long id) {

courseRepository.deleteById(id);

}

}

**StudentController.java:**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/students")

public class StudentController {

@Autowired

private StudentRepository studentRepository;

@GetMapping

public List<Student> getAllStudents() {

return (List<Student>) studentRepository.findAll();

}

@GetMapping("/{id}")

public Student getStudentById(@PathVariable Long id) {

return studentRepository.findById(id).orElse(null);

}

@PostMapping

public Student addStudent(@RequestBody Student student) {

return studentRepository.save(student);

}

@PutMapping("/{id}")

public Student updateStudent(@PathVariable Long id, @RequestBody Student student) {

student.setId(id);

return studentRepository.save(student);

}

@DeleteMapping("/{id}")

public void deleteStudent(@PathVariable Long id) {

studentRepository.deleteById(id);

}

@PutMapping("/{studentId}/enroll")

public Student enrollStudentInCourse(@PathVariable Long studentId, @RequestBody EnrollRequest enrollRequest) {

// Implementation to enroll a student in a course

return null;

}

}

**EnrollRequest.java:**

public class EnrollRequest {

private Long courseId;

// getters and setters

}

**Application.java:**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

**application.properties:**

spring.datasource.url=jdbc:h2:mem:testdb (msql-database)

spring.datasource.driverClassName=org.h2.Driver(mysql driver)

spring.datasource.username=root

spring.datasource.password=mysql

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect(mysql)