German University in Cairo
Department of Computer Science
Assoc. Prof. Mervat Abu El-Kheir

Architecture of Massively Scalable Applications, Spring 2025, Spring 2025

Task 2

Deadline is next Saturday 08/03/2025 11:59 PM

1 Introduction

The objective of this task is to design and implement a database schema using PostgreSQL and integrate it with a Spring Boot application. The schema will include three main entities: **Student**, **Course**, and **Instructor**. Students will establish the appropriate relationships between these entities and implement the necessary methods in the service layer. You Should clone the task base code from this link: https://github.com/Scalable2025/Task_2_Base.git

2 Database Schema

The system will include the following tables:

2.1 Tables

2.1.1 Students

- Integer id (Identity Primary Key)
- String name
- String email

2.1.2 Instructor

- Integer id (Identity Primary Key)
- String name
- String email

2.1.3 Course

- Integer id (Identity Primary Key)
- String name
- String code
- Integer credit

2.2 Relationships

- A **Many-to-Many**
  - relationship between **Student** and **Course** (a student can enroll in multiple courses, and a course can have multiple students).
  - Student is the owner of the relation
- A **One-to-Many**
  - relationship between **Instructor** and **Course** (one instructor teaches multiple courses, but each course has only one instructor).
  - Course is the owner of the relation

---

3 Repositories

3.1 Student Repostory

You are required to create a Student Repository that has ***One custom Query*** which is ***findStudents-ByCourseId*** that takes in a course ID and returns a List of students that are enrolled in this course.

Listing 1: StudentRepository

```
public interface StudentRepository extends JpaRepository<Student, Integer> {


}
```

3.2 Instructor Repository

You are required to create an Instructor Repository that has ***one custom ORM (follwing the naming convention)*** which is ***findInstructorByEmail*** that takes in the email and returns an Instructor that has this email.

Listing 2: InstructorRepository

```
public interface InstructorRepository extends JpaRepository<Instructor, Integer> {


}
```

3.3 Course Repository

You are required to create a Course Repository that has ***Two custom Query***:

- ***Enroll Student***:
  It takes CourseID and StudentID and enroll this student in that course.

- ***Unenroll Student***
  It takes CourseID and StudentID and un-enroll the student from the course.

You should also implement ***a custom ORM*** by following the name convention which is ***findByInstructorId*** that takes the Instructor ID as input and returns a list of courses that this Instructor is teaching.

Listing 3: CourseRepository

```
public interface CourseRepository extends JpaRepository<Course, Integer> {


}
```

---

4 Services

4.1 StudentService

Your student service should implement the following methods:

4.1.1 Get All Students:

This method should return all the students present in the student table.

Listing 4: Get All Students

```
public List<Student> getAllStudents() {

}
```

4.1.2 Get Student By ID:

This method should take the ID of the student as a parameter and return the matching student from the student's table.

Listing 5: Get Student By ID

```
public Optional<Student> getStudentById(Integer id) {

}
```

4.1.3 Create a Student:

This method should take a student object and save it to the database.

Listing 6: Create a Student

```
public Student saveStudent(Student student) {

}
```

4.1.4 Delete a Student:

This method should take a Student ID and delete it from the database.

Listing 7: Delete a Student

```java
public void deleteStudent(Integer id) {

}
```

4.1.5 Get Student by Course ID:

This method should take a Course ID and return a list of all the students taking this course.

Listing 8: Get Student by Course ID

```java
public List<Student> getStudentsByCourseId(Integer courseId) {

}
```

4.2 InstructorService

Your Instructor Service should implement the following methods:

4.2.1 Get all Instructors:

This method should return all the instructors present in the instructors table.

Listing 9: Get All Instructors

```java
public List<Instructor> getAllInstructors() {

}
```

4.2.2 Get Instructor by ID

This method should take the ID of the instructor as a parameter and return the matching instructor from the instructor's table.

Listing 10: Get Instructor By ID

```java
public Optional<Instructor> getInstructorByID(Integer id) {

}
```

4.2.3 Get Instructor by Email

This method should take the Email of the instructor as a parameter and return the matching instructor from the instructor's table.

Listing 11: Get Instructor By Email

```java
public Instructor getInstructorByEmail(String email) {

}
```

4.3 CourseService

Your Course Service should implement the following methods:

4.3.1 Get all Courses:

This method should return all the courses present in the courses table.

Listing 12: Get All Courses

```java
public List<Course> getAllCourses() {

}
```

4.3.2 Get Course By ID:

This method should take the ID of the course as a parameter and return the matching course from the course's table.

Listing 13: Get Course By ID

```java
public Optional<Course> getCourseByID(Integer id) {

}
```

4.3.3 Get Courses By Instructor ID:

This method should take the ID of the instructor as a parameter and return a list of the courses that this instructor teaches.

Listing 14: Get Courses By Instructor ID

```java
public List<Course> getCoursesByInstructorId(Integer instructorId) {

}
```

4.3.4 Enroll a Student:

This method should take the ID of the course and the ID of the students as parameters and enroll the student in the course

Listing 15: Enroll a Student

```java
public void enrollStudent(Integer courseId, Integer studentId) {

}
```

4.3.5 Unenroll a Student:

This method should take the ID of the course and the ID of the students as parameters and un-enroll the student from the course

Listing 16: Unenroll a Student

```java
public void unenrollStudent(Integer courseId, Integer studentId) {

}
```

---

5 Controllers

5.1 Student Controller:

Listing 17: Student Controller

```java
@RestController
@RequestMapping("/students")
public class StudentController {


}
```

Your student controller should handle the following Requests:

5.1.1 Get All Students:

GET Request to get all the student in our system. This method should ***return a list of all the students***.

5.1.2 Get Student By ID

GET Request to get a specific student by passing its ID in the URL. This method should ***return the found Student***.

5.1.3 Create a Student:

POST Request to create a new Student by passing a student object in the request Body. This method should ***return the created Student***.

5.1.4 Update a Student:

PUT Request to update a student by passing its ID in the URL and the updated student in the request Body. This method should ***return the updated Student***.

5.1.5 Delete a Student:

DELETE Request to delete a Student by passing ID in the URL. This method should be ***void***.

5.1.6 Get Students by Course ID:

GET Request to get all the students taking a specfic course by passing the course ID in the URL. Your URL should be like this something like this ***/course/{courseID}***This method should ***return a list of the found students***.

5.2 Instructor Controller:

Listing 18: Instructor Controller

```
@RestController
@RequestMapping("/instructors")
public class InstructorController {


}
```

Your Instructor Controller should handle the following Requests:

5.2.1 Get All Instructors:

GET Request to get all the instructors present in our system. This method should ***return a list of instructors***

5.2.2 Get Instructor by Email:

GET Request to get the instructor by passing its email in the URL. This method should ***return an Instructor Object***.

5.3 Course Controller:

Listing 19: Course Controller

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


}
```

Your Course Controller should handle the following requests:

5.3.1 Get all Courses

GET Request to get all the courses in our system. This method should ***return a list of courses***.

5.3.2 Get Courses By Instructor ID:

Get Request to get all the courses taught by a specific instructor by passing his/her ID in the URL. This method ***should return a list of courses***.

5.3.3 Enroll a Student

POST Request to add a Student to the course. This method should take the course ID and the student ID from the URL as follows ***/{courseId}/students/{studentId}***. The method should ***return a string telling the user that the student is enrolled in the course successfully***.

5.3.4 Unenroll a Student

PUT Request to remove a Student from the course. This method should take the course ID and the student ID from the URL as follows ***/{courseId}/students/{studentId}***. The method should ***return a string telling the user that the student is unenrolled from the course successfully***.

---

6 Database Seeding

There is a controller called ***Seeder*** in your code that you can use to seed your database with some data. You may called using ***seed***

---

7 Docker

Dockerize your application by creating the necessary docker-compose and docker files as seen in the Lab.

---

8 Testing

To run the public API tests, uncomment the code in the test file then run mvn test in terminal.

---

9 Submission Guidlines:

Please Follow the following guidelines to avoid receiving a cheating case:

a) Name your database in the following format: FirstName_ID (eg. random_52_1234)

b) the port that you will use to connect to your app that will be specified in your docker compose file should be the part after the dash in your ID ***(e.g if your ID is 52-8078 then your port will be 8078)***

c) You ID and your Name should be environment variable in your application.properties and use it accordingly.

IF YOU DIDN'T FOLLOW THESE GUIDELINES YOU WILL RECEIVE A CHEATING CASE WHICH WILL GET YOU A ZERO IN YOUR TASK GRADE.

Please Zip your Java Project and submit your task using the following Link:

https://forms.gle/GAAU1g74FSXkbyZY9