Assignment: Spring Boot Data JPA - Entity Relationships

Objective: To understand the various relationship mappings in JPA using Spring Boot Data JPA.
Background: In the world of relational databases, the relations between tables play a vital role in ensuring data integrity and making sense of large volumes of data. JPA provides annotations and tools to map these relations directly onto our Java classes, allowing a table-per-class strategy.
Prerequisites:

- Basic knowledge of Java.
- Familiarity with Spring Boot and JPA.
- An Integrated Development Environment (IDE) such as IntelliJ IDEA or Eclipse.
- A relational database system (e.g., MySQL, PostgreSQL, H2).

Tasks:

1. **Setup**:
    - Initialize a new Spring Boot project using Spring Initializr. Include the following dependencies:
        - Spring Web
        - Spring Data JPA
        - Your preferred database (H2 for simplicity, or MySQL, PostgreSQL, etc.)
2. **One-To-One Mapping**:
    - Create two entities: **Profile** and **User**.
    - A user has one profile, and a profile belongs to one user.
    - Use the **@OneToOne** annotation to map this relationship. Ensure you also utilize the **@JoinColumn** annotation.
    - Implement a CRUD repository for each entity and create a simple RESTful API to manipulate and fetch the data.
3. **One-To-Many Mapping**:
    - Create two entities: **Author** and **Book**.
    - An author can write many books, but each book has only one author.
    - Use the **@OneToMany** and **@ManyToOne** annotations to map this relationship.
    - Implement a CRUD repository for each entity and create a RESTful API to manipulate and fetch the data.
4. **Many-To-Many Mapping**:
    - Create two entities: **Student** and **Course**.
    - A student can enroll in many courses, and a course can have many students.
    - Use the **@ManyToMany** annotation to map this relationship. Ensure that you use a join table.
    - Implement a CRUD repository for each entity and create a RESTful API to manipulate and fetch the data.

Submission:

- Push your code to a public GitHub repository.
- Include the README documentation and any other supplementary materials.

- Submit the link to your repository.

Evaluation Criteria:

- Correctness of entity relationship mappings.
- Functionality of CRUD operations for each entity.
- Completeness and clarity of documentation.
- Proper error handling and validation.
- Test coverage and the correctness of tests.

Sample Database tables

```
-- User & Profile tables (One-To-One)
CREATE TABLE User (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(255) NOT NULL,
   password VARCHAR(255) NOT NULL,
   profile_id BIGINT,
   UNIQUE (profile_id),
   FOREIGN KEY (profile_id) REFERENCES Profile(id)
);

CREATE TABLE Profile (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   firstName VARCHAR(255),
   lastName VARCHAR(255),
   email VARCHAR(255) UNIQUE NOT NULL
);

-- Author & Book tables (One-To-Many)
CREATE TABLE Author (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL
);

CREATE TABLE Book (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   title VARCHAR(255) NOT NULL,
   author_id BIGINT,
   FOREIGN KEY (author_id) REFERENCES Author(id)
);

-- Student & Course tables (Many-To-Many)
CREATE TABLE Student (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL
```

```
);

CREATE TABLE Course (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   courseName VARCHAR(255) NOT NULL
);

CREATE TABLE Student_Course (
   student_id BIGINT,
   course_id BIGINT,
   PRIMARY KEY (student_id, course_id),
   FOREIGN KEY (student_id) REFERENCES Student(id),
   FOREIGN KEY (course_id) REFERENCES Course(id)
);

-- Insert for User & Profile
INSERT INTO Profile(firstName, lastName, email) VALUES ('John', 'Doe',
'john.doe@example.com');
INSERT INTO User(username, password, profile_id) VALUES ('johndoe', 'password123', 1);

INSERT INTO Profile(firstName, lastName, email) VALUES ('Jane', 'Doe',
'jane.doe@example.com');
INSERT INTO User(username, password, profile_id) VALUES ('janedoe', 'password123', 2);

INSERT INTO Profile(firstName, lastName, email) VALUES ('Alice', 'Smith',
'alice.smith@example.com');
INSERT INTO User(username, password, profile_id) VALUES ('alicesmith', 'password123', 3);

-- Insert for Author & Book
INSERT INTO Author(name) VALUES ('George Orwell');
INSERT INTO Book(title, author_id) VALUES ('1984', 1);

INSERT INTO Author(name) VALUES ('J.K. Rowling');
INSERT INTO Book(title, author_id) VALUES ('Harry Potter and the Sorcerer''s Stone', 2);

INSERT INTO Author(name) VALUES ('J.R.R. Tolkien');
INSERT INTO Book(title, author_id) VALUES ('The Hobbit', 3);

-- Insert for Student & Course
INSERT INTO Student(name) VALUES ('Robert');
INSERT INTO Course(courseName) VALUES ('Mathematics');

INSERT INTO Student(name) VALUES ('Julia');
INSERT INTO Course(courseName) VALUES ('Physics');

INSERT INTO Student(name) VALUES ('Steve');
INSERT INTO Course(courseName) VALUES ('Chemistry');
```

-- For Many-To-Many, we need a mapping. Let's assume Robert takes all three courses:
INSERT INTO Student_Course(student_id, course_id) VALUES (1, 1);
INSERT INTO Student_Course(student_id, course_id) VALUES (1, 2);
INSERT INTO Student_Course(student_id, course_id) VALUES (1, 3);

Profile Table:

id	firstName	lastName	email
1	John	Doe	john.doe@example.com
2	Jane	Doe	jane.doe@example.com
3	Alice	Smith	alice.smith@example.com

User Table:

id	username	password	profile_id
1	johndoe	password123	1
2	janedoe	password123	2
3	alicesmith	password123	3

Author Table:

id	name
1	George Orwell
2	J.K. Rowling
3	J.R.R. Tolkien

Book Table:

id	title	author_id
1	1984	1
2	Harry Potter and the Sorcerer's Stone	2
3	The Hobbit	3

Student Table:

id	name
1	Robert
2	Julia
3	Steve

Course Table:

id	courseName
1	Mathematics
2	Physics
3	Chemistry

Student_Course Table:

student_id	course_id
1	1
1	2
1	3