JavaAcademy

Student Management System: JPA vs Spring Data JPA

Author

Pedro Jahir Hinojosa García

Instructor

Miguel Rugerio

Delivery date: September 6, 2024

1 Introduction

This document explains the implementation of a **Student Management System** using Java Spring Boot and MySQL, highlighting the key differences between using standard JPA with `EntityManager` and Spring Data JPA (`JpaRepository`). The project manages student information such as name, scores, and pass/fail status. We will look at how these two approaches impact the implementation of the same features.

2 Project Overview

The Student Management System provides REST APIs for performing CRUD operations on the `Students` table. The system allows:

- Adding new students.
- Retrieving all students.
- Filtering students by pass/fail status.
- Deleting students by ID.

MySQL Database Schema The MySQL table `Students` is defined as follows:

```
CREATE TABLE `Students` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `last_name` varchar(64),
  `first_name` varchar(64),
  `first_period` DECIMAL(10,2),
  `second_period` DECIMAL(10,2),
  `average` DECIMAL(10,2) GENERATED ALWAYS AS
      ((`first_period` + `second_period`) / 2) STORED,
  `pass_status` VARCHAR(10) GENERATED ALWAYS AS(
    CASE
      WHEN `average` >= 70 THEN 'Pass'
      ELSE 'Fail'
    END) STORED,
  PRIMARY KEY (`id`)
);
```

Example API Output Here's an example response from the API when retrieving all students:

```
[
    {
        "id": 1,
        "lastName": "Smith",
        "firstName": "John",
        "firstPeriod": 10.00,
        "secondPeriod": 90.00,
        "average": 50.00,
        "passStatus": "Fail"
    },
    {
        "id": 2,
        "lastName": "Johnson",
        "firstName": "Emily",
        "firstPeriod": 78.25,
        "secondPeriod": 82.50,
        "average": 80.375,
        "passStatus": "Pass"
    }
]
```

3 Using JPA with `EntityManager`

In JPA, we use the `EntityManager` to handle database operations. Here's how a method to retrieve all students looks using JPA:

```
public List<Student> findAllStudents() {
    TypedQuery<Student> query = entityManager
    .createQuery("FROM Student", Student.class);
    return query.getResultList();
}
```

With JPA, we must manually write the query using JPQL (Java Persistence Query Language), and the `EntityManager` handles the database connection and execution of the query.

Differences in Key Areas:

- **Manual Queries**: JPA requires manually writing queries (e.g., `FROM Student`) to fetch data, making it more flexible but also more verbose.
- **Entity Management**: JPA allows more control over the 'EntityManager' lifecycle, transactions, and persistence context, but it requires more manual handling, such as managing entity states and manually merging entities.

4 Using Spring Data JPA

Spring Data JPA simplifies database operations by abstracting common tasks. Here's how the same method to retrieve all students would look with Spring Data JPA:

```
@Autowired
```

```
private StudentRepository studentRepository;

public List<Student> findAllStudents() {
    return studentRepository.findAll();
}
```

Key Differences:

- **Built-in Repository**: With Spring Data JPA, we can extend `JpaRepository`, which provides built-in methods like `findAll()`, `save()`, and `deleteById()`, reducing boilerplate code.
- **Less Manual Code**: Since methods like `findAll()` are provided out-of-the-box, developers can avoid writing custom queries for common operations.

Here's how the repository interface would look:

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

This automatically provides CRUD functionality without the need to define the query.

5 Implementation Comparison

Saving a Student (JPA): Using JPA with `EntityManager`, saving or updating a student requires explicit merging:

```
public Student save(Student theStudent) {
    return entityManager.merge(theStudent);
}
```

Saving a Student (Spring Data JPA): With Spring Data JPA, this is simplified as the repository provides a `save()` method:

```
public Student save(Student theStudent) {
    return studentRepository.save(theStudent);
}
```

Filtering Pass Students (JPA): To filter pass students using JPA, we can use Java Streams to process the query results:

```
public List<Student> findPassStudents() {
    return entityManager.createQuery("FROM Student", Student.class)
        .getResultList()
        .stream()
        .filter(student -> student.getAverage().compareTo(new

        BigDecimal(70)) >= 0)
        .collect(Collectors.toList());
}
```

Filtering Pass Students (Spring Data JPA): With Spring Data JPA, this logic remains similar, but the repository handles most of the database interactions:

```
public List<Student> findPassStudents() {
    return studentRepository.findAll()
        .stream()
        .filter(student -> student.getAverage().compareTo(new

        BigDecimal(70)) >= 0)
        .collect(Collectors.toList());
}
```

6 Advantages of Each Approach

JPA with `EntityManager`:

- Provides more flexibility and control over transactions and entity management.
- Suitable for complex queries that require fine-grained control over the persistence context.
- Ideal when you need custom behavior not provided by Spring Data JPA.

Spring Data JPA:

- Reduces boilerplate code by providing built-in repository methods.
- Makes development faster by eliminating the need to manually manage `EntityManager`.
- Suitable for simple applications where CRUD operations and basic queries are needed.

7 Conclusion

Both JPA with `EntityManager` and Spring Data JPA have their advantages. If you need more control over database operations, JPA with `EntityManager` is the better choice. However, for most typical applications, Spring Data JPA can significantly speed up development and reduce the complexity of the codebase.