LESSON 23
MySQL & JPA Basics

WEEK 05

What is JPA?

- **Definition**:
  - Java Persistence API (JPA) is a specification for object-relational mapping (ORM).
  - Maps Java objects to database tables.
- **Key Components**:
  - Entity, EntityManager, Persistence Unit.
- **Why Use JPA?**:
  - Simplifies database operations (no manual SQL for basic CRUD).
  - Supports multiple databases (e.g., MySQL, PostgreSQL).

Spring Data JPA Overview

- **What is Spring Data JPA?**:
  - Extension of Spring Data for JPA-based repositories.
  - Provides built-in methods for CRUD operations.
- **Key Features**:
  - Repository interfaces (CrudRepository, JpaRepository).
  - Query methods derived from method names.
  - Custom queries with @Query annotation.
- **Benefits**:
  - Reduces boilerplate code for database access.
  - Integrates seamlessly with Spring Boot.

Setting Up the Development Environment

- **Tools Required**:
  - JDK 17+, IntelliJ IDEA (or Eclipse or VS Code), MySQL, Maven/Gradle.
- **Steps**:
  - Install MySQL and create a database (e.g., school_db).
  - Configure IDE with Spring Boot plugin.
  - Add Spring Boot Starter dependencies.
- **Dependencies**:
  - spring-boot-starter-data-jpa
  - mysql-connector-java
- **Reference**: Spring Initializr

Creating a Spring Boot Project

- **Using Spring Initializr**:
  - Select Java, Gradle, Spring Boot 3.x.
  - Add dependencies: Spring Web, Spring Data JPA, MySQL Driver.
- **Project Structure**:
  - src/main/java: Application code.
  - src/main/resources: Configuration files (e.g., application.properties).
- **Example**:
  - Generate project at start.spring.io.
  - Import into IDE and run.

Configuring MySQL in Spring Boot

- **Configuration File**:
  - Edit application.properties to connect to MySQL.
- **Example Configuration**:

```
spring.datasource.url=jdbc:mysql://localhost:3306/school_db
spring.datasource.username=root
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
```

- **Explanation**:
  - ddl-auto=update: Automatically creates/updates database schema based on entities.
  - Ensure MySQL server is running.

Creating a JPA Entity

- **What is an Entity?**:
  - A Java class mapped to a database table.
- **Annotations**:
  - @Entity: Marks class as an entity.
  - @Id: Defines primary key.
  - @GeneratedValue: Auto-generates ID values.

Creating a JPA Repository

- **Repository Interface**:
  - Extends JpaRepository<EntityClass, IDType>.
  - Provides built-in CRUD methods.
- Example:

```
public interface StudentRepository extends JpaRepository<Student, Long> {
    // Custom query methods
}
```

- **Built-in Methods**:

  save(), findById(), findAll(), deleteById().

Implementing Create Operation

- **Purpose**:
  - Save a new entity to the database.
- **Explanation**:
  - save() persists the entity to the database.
  - Returns the saved entity with generated ID.

```
@Autowired
private StudentRepository repository;

public Student createStudent(Student student) {
    return repository.save(student);
}
```

Implementing Read Operation

- **Purpose**:
  - Retrieve entities from the database.
- **Explanation**:
  - findAll(): Retrieves all records.
  - findById(): Retrieves a single record by ID.

```
public List<Student> getAllStudents() {
    return repository.findAll();
}

public Optional<Student> getStudentById(Long id) {
    return repository.findById(id);
}
```

Implementing Update Operation

- **Purpose**:
  - Modify an existing entity in the database.
- **Explanation**:
  - Fetch entity, update fields, and save.

```
public Student updateStudent(Long id, Student updatedStudent) {
    Student student = repository.findById(id).orElseThrow();
    student.setName(updatedStudent.getName());
    student.setEmail(updatedStudent.getEmail());
    return repository.save(student);
}
```

Implementing Delete Operation

- **Purpose**:
  - Remove an entity from the database.
- **Explanation**:
  - deleteById() removes the entity with the specified ID.
  - Throws exception if ID does not exist.

```
public void deleteStudent(Long id) {
    repository.deleteById(id);
}
```

Creating a REST Controller

- **Purpose**:
  - Expose CRUD operations via RESTful APIs.

```java
@RestController
@RequestMapping("/api/students")
public class StudentController {
    @Autowired
    private StudentService service;

    @PostMapping
    public Student create(@RequestBody Student student) {
        return service.createStudent(student);
    }

    @GetMapping
    public List<Student> getAll() {
        return service.getAllStudents();
    }
}
```

Full CRUD Example

- **Scenario**:
  - Manage student records (create, read, update, delete).

```java
@RestController
@RequestMapping("/api/students")
public class StudentController {
    @Autowired
    private StudentService service;

    @PostMapping
    public Student create(@RequestBody Student student) {
        return service.createStudent(student);
    }

    @GetMapping("/{id}")
    public Student getById(@PathVariable Long id) {
        return service.getStudentById(id).orElseThrow();
    }

    @PutMapping("/{id}")
    public Student update(@PathVariable Long id, @RequestBody Student student) {
        return service.updateStudent(id, student);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        service.deleteStudent(id);
    }
}
```

Testing APIs with Postman

- **Steps**:
  - Start Spring Boot application.
  - Use Postman to send HTTP requests (POST, GET, PUT, DELETE).
- **Example**:
  - POST: http://localhost:8080/api/students with JSON body:

```
{"name": "John Doe", "email": "john@example.com"}
```

Validation with JPA

- **Purpose**:
  - Ensure valid data before saving to database.
- **Explanation**:
  - Use annotations like @NotNull, @Email from javax.validation.

```
public class Student {
    @NotNull
    private String name;

    @Email
    private String email;
}
```

Conclusion and Next Steps

- **Summary**:
  - Learned to build a CRUD application with Spring Boot, JPA, and MySQL.
  - Covered entities, repositories, REST APIs, and basics JPA features.
- **Next Steps**:
  - Build a full-stack application with a front-end (e.g., React).
- **References**:
  - Spring Boot
  - Spring Data JPA
  - JPA Specification