Spring Boot with Database (Spring Data JPA & Hibernate) 🗄

- ✅ What is Spring Data JPA?
- ✅ Setting up MySQL/PostgreSQL in Spring Boot
- ✅ Creating an Entity (@Entity)
- ✅ Using JPA Repository (JpaRepository)
- ✅ Performing CRUD Operations
- ✅ Using @Query for Custom Queries
- ✅ Pagination & Sorting

What is Spring Data JPA? 🤔

Spring Data JPA is an abstraction over JPA (Java Persistence API) that makes working with databases easier.

- Uses Hibernate (default ORM) to manage database operations.
- Allows CRUD operations without writing SQL queries.
- Works with multiple databases (MySQL, PostgreSQL, H2, etc.).

Setting Up MySQL in Spring Boot 🛠

Step 1: Add Dependencies in pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

✅ This will enable JPA and MySQL support in your Spring Boot project.

Step 2: Configure Database in application.properties 📌 For MySQL Configuration

```properties
spring.datasource.url=jdbc:mysql://localhost:3306/mydatabase
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate settings
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

✅ Ensure MySQL is running and create a database named mydatabase.

Creating an Entity (@Entity)

🔹 Entities represent database tables in Java. 🔹 Use @Entity, @Id, and @GeneratedValue for primary keys.

```java
import jakarta.persistence.*;

@Entity
@Table(name = "users") // Table name in DB
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-increment
    private Long id;

    private String name;
    private String email;

    // Constructors
    public User() {}
    public User(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters & Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

✅ This will create a users table in MySQL.

Creating JPA Repository (JpaRepository)

🔹 Spring Data JPA provides built-in repository methods for CRUD operations. 🔹 Extend JpaRepository<User, Long> to enable database operations.

```java
import org.springframework.data.jpa.repository.JpaRepository;
```

```java
// Interface for database operations
public interface UserRepository extends JpaRepository<User, Long> {
}
```

✅ Now, Spring Boot will automatically handle database queries

Performing CRUD Operations (Service Layer) 🎯

Create a UserService to manage business logic

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    // Create User
    public User createUser(User user) {
        return userRepository.save(user);
    }

    // Get All Users
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    // Get User by ID
    public User getUserById(Long id) {
        return userRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("User not 
found!"));
    }

    // Update User
    public User updateUser(Long id, User newUser) {
        User existingUser = getUserById(id);
        existingUser.setName(newUser.getName());
        existingUser.setEmail(newUser.getEmail());
        return userRepository.save(existingUser);
    }

    // Delete User
    public void deleteUser(Long id) {
        userRepository.deleteById(id);
    }
}
```

✅ Now, we have all CRUD methods ready to use!

Creating a REST Controller (@RestController)

🔹 Expose API endpoints to interact with the database.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    // Create User (POST)
    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.createUser(user);
    }

    // Get All Users (GET)
    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    // Get User by ID (GET)
    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) {
        return userService.getUserById(id);
    }

    // Update User (PUT)
    @PutMapping("/{id}")
    public User updateUser(@PathVariable Long id, @RequestBody User user) {
        return userService.updateUser(id, user);
    }

    // Delete User (DELETE)
    @DeleteMapping("/{id}")
    public String deleteUser(@PathVariable Long id) {
        userService.deleteUser(id);
        return "User deleted successfully!";
    }
}
```

Custom Queries using @Query

🔹 Spring JPA allows writing custom SQL queries using @Query.

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

public interface UserRepository extends JpaRepository<User, Long> {

    // Find user by email
    @Query("SELECT u FROM User u WHERE u.email = :email")
    User findByEmail(@Param("email") String email);
}
```

✅ Use this method in the service to fetch users by email.

Pagination & Sorting 📊

🔹 Spring Data JPA makes pagination easy using Pageable.

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;

public interface UserRepository extends JpaRepository<User, Long> {
   Page<User> findAll(Pageable pageable);
}
```

✅ Get Paginated Results:

```
@GetMapping("/page")
public Page<User> getUsers(Pageable pageable) {
   return userRepository.findAll(pageable);
}
```

✅ Call API with pagination:

```
GET http://localhost:8080/users/page?page=0&size=5
```

✅ Summary

- ✔ Spring Boot + JPA simplifies database operations.
- ✔ CRUD operations are automatically handled by JpaRepository.
- ✔ MySQL/PostgreSQL can be integrated with application.properties.
- ✔ Custom queries and pagination are easy to implement.