

# Building REST APIs with Spring Boot

## 3.4 Data Layer and Persistence

# Learning objectives

---

- ▶ What Spring Data JPA is and why we use it
- ▶ How to model entities using JPA annotations
- ▶ How to work with H2 in-memory database
- ▶ How to define and use Repository interfaces
- ▶ How to test repository behavior using JUnit + Spring Boot

# Introduction to Spring Data JPA

---

- ▶ Spring Data JPA provides:
  - ▶ A simplified way to work with relational databases
  - ▶ Automatic CRUD methods through repositories
  - ▶ Integration with JPA and Hibernate
  - ▶ Convention-based query generation
  - ▶ Minimal boilerplate code

# Why Use Spring Data JPA?

---

- ▶ Eliminates the need for manual SQL in basic cases
- ▶ Reduces boilerplate data access code
- ▶ Easy integration with Spring Boot
- ▶ Encourages clean Layered Architecture
- ▶ Works seamlessly with H2, PostgreSQL, MySQL, etc.

# How Spring Data JPA Fits in the Architecture

---

- ▶ Controller → Service → Repository → Database
  - ▶ Controllers handle requests
  - ▶ Services contain logic
  - ▶ Repositories handle persistence
  - ▶ Database stores data

# What Is a JPA Entity?

---

- ▶ A JPA Entity is:
  - ▶ A Java class mapped to a database table
  - ▶ Managed by Hibernate
  - ▶ Annotated with @Entity
  - ▶ Must have a primary key
  - ▶ Used by repositories for CRUD operations

# Essential JPA Annotations

---

- ▶ `@Entity` - marks a class as a JPA entity
- ▶ `@Id` - identifies the primary key
- ▶ `@GeneratedValue` - auto-generated IDs
- ▶ `@Column` - customize database column mapping
- ▶ `@Table` - optional table-level configuration

# H2 In-Memory Database

---

- ▶ Lightweight and easy to configure
- ▶ Perfect for demos and training
- ▶ Runs entirely in memory
- ▶ Automatically resets on application restart
- ▶ Works great with Spring Boot auto-configuration



# CRUD with JPA Repositories

---

- ▶ A JPA Repository is:
  - ▶ A data access abstraction
  - ▶ Provided by Spring Data
  - ▶ Automatically implemented at runtime
  - ▶ Used by services to interact with the DB

# What You Get Out of the Box

---

- ▶ Spring Data JPA provides automatic methods like:
  - ▶ findAll()
  - ▶ findById()
  - ▶ save()
  - ▶ deleteById()
- ▶ No implementation needed – Spring generates the code.

# Query Methods by Convention

---

- ▶ Repository interfaces may define methods such as:
  - ▶ `findByTitle(String title)`
  - ▶ `findByCompleted(boolean completed)`
  - ▶ `findByAgeGreaterThan(int age)`
- ▶ Spring parses the method name and creates the SQL automatically.

# Testing Repository Methods

---

- ▶ Why Test Repositories?
  - ▶ Ensure entity mappings are correct
  - ▶ Validate database integration
  - ▶ Confirm that auto-generated queries behave as expected
  - ▶ Catch issues when the persistence model changes

# How Spring Boot Helps Testing Repositories

---

- ▶ Spring Boot provides:
  - ▶ Automatic creation of an in-memory database
  - ▶ Automatic loading of entity classes
  - ▶ Automatic creation of repository beans
  - ▶ `@SpringBootTest` support for full context
  - ▶ Clean test environment on each run

# Testing Strategies for Repositories

---

- ▶ Typical approaches:
  - ▶ Use `@SpringBootTest` to load context
  - ▶ Autowire the repository directly
  - ▶ Insert sample entities
  - ▶ Test CRUD operations
  - ▶ Test custom repository methods

# Demo Overview – Repository & H2

---

- ▶ In the demo, the instructor will show:
  - ▶ Defining a simple entity
  - ▶ Creating a repository
  - ▶ Running the app and using H2 console to inspect data
  - ▶ Executing CRUD operations through the service/controller
  - ▶ Writing repository tests with Spring Boot

# Lab Overview: Adding Persistence

---

- ▶ Use the solution from 3-3 and replace the in-memory TaskService
- ▶ Introduce a JPA repository for Task
- ▶ Use H2 as an in-memory database
- ▶ Load initial Task data using data.sql
- ▶ Modify the controller to use repository
- ▶ Write repository tests for CRUD operations



# Step 1: Add Maven Dependencies

---

- ▶ Add the required dependencies:
  - ▶ Spring Data JPA
  - ▶ H2 Database
  - ▶ Spring Boot Test (already included)
- ▶ Rebuild project after changes
- ▶ (Look at the demo code for help with many of these steps)

# Step 2: Update application.properties

---

- ▶ Configure H2:
  - ▶ Enable H2 console
  - ▶ Set datasource URL
  - ▶ Set JDBC driver and dialect
  - ▶ Use `spring.jpa.hibernate.ddl-auto=update`
  - ▶ Use `spring.jpa.defer-datasource-initialization=true`

# Step 3: Create JPA Entity for Task

---

- ▶ Modify Task class:
  - ▶ Add @Entity
  - ▶ Add @Id and @GeneratedValue
  - ▶ Ensure fields are correct
  - ▶ Ensure a no-args constructor exists
  - ▶ Validate naming and capitalization

# Step 4: Create TaskRepository

---

- ▶ Create repository interface:
  - ▶ Extend JpaRepository<Task, Long>
  - ▶ Provide no additional methods yet
  - ▶ Let Spring generate CRUD operations

# Step 5: Replace Service Layer

---

- ▶ Remove in-memory service logic:
  - ▶ Delete TaskService
  - ▶ Update controller to inject the TaskRepository
  - ▶ Use the TaskRepository for all CRUD logic

# Step 6: Create data.sql

---

- ▶ Provide initial test data:
  - ▶ Insert a few Task rows
  - ▶ Ensure IDs are created by the database

# Step 7: Write Repository Tests

---

- ▶ Remove all tests from the previous lab
- ▶ Write tests that:
  - ▶ Load Spring context
  - ▶ Use H2 database
  - ▶ Verify findAll()
  - ▶ Verify findById()
  - ▶ Test insert, update, delete
  - ▶ Use transactional rollback (@Transactional)

# Step 8: Run and Validate

---

- ▶ Verify everything works:
  - ▶ Start the application
  - ▶ Open H2 console
  - ▶ Test controller endpoints
  - ▶ Ensure repository tests pass



# Key Takeaways

---

- ▶ Spring Data JPA simplifies database access
- ▶ Entities represent tables in your application
- ▶ H2 lets you develop and test quickly
- ▶ Repositories provide built-in CRUD operations
- ▶ Query methods allow flexible querying with no SQL
- ▶ Repository tests validate your persistence layer