**WEEK – 3**

Module 6 - Spring Data JPA with Spring Boot, Hibernate

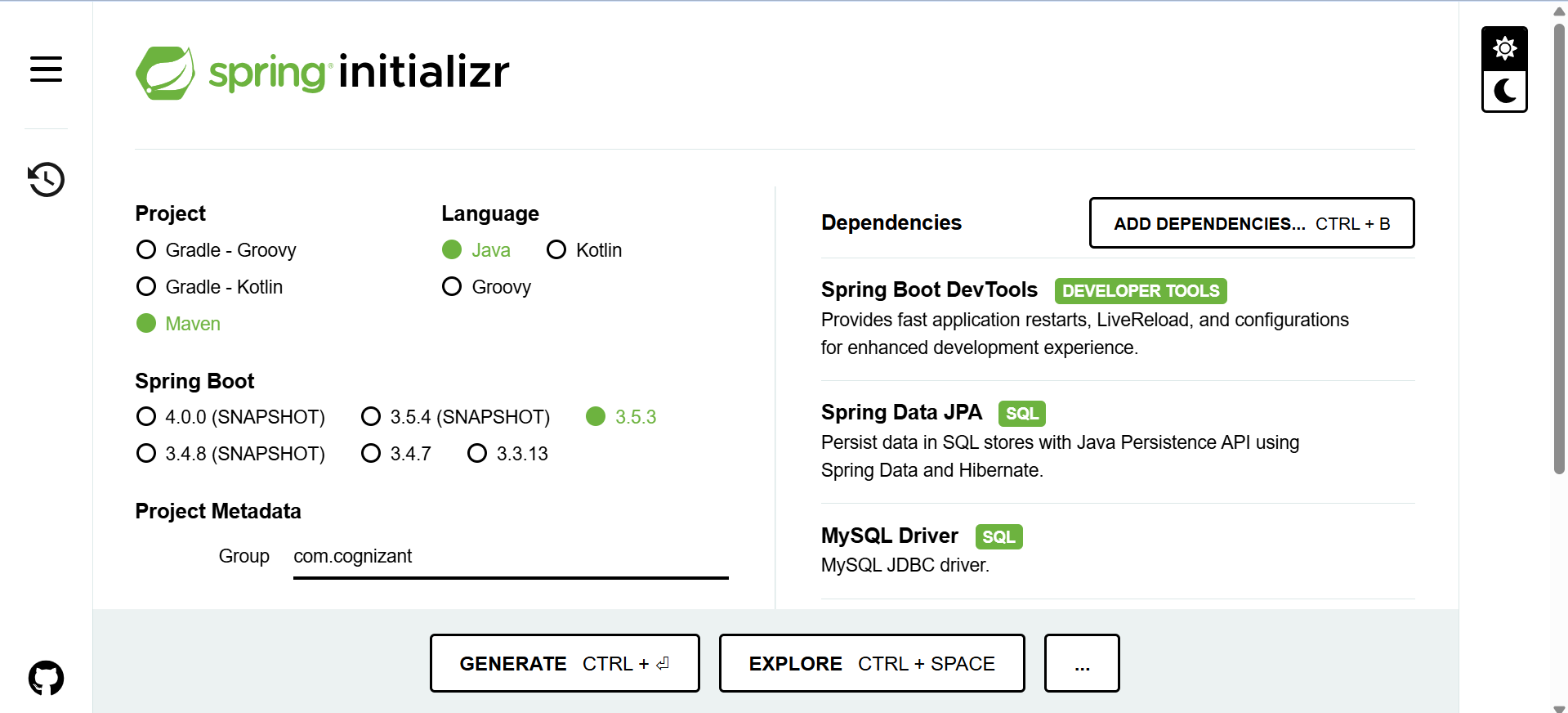
Hands on 1

Spring Data JPA - Quick Example   
  
**Software Pre-requisites**

* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2

**Create a Eclipse Project using Spring Initializr**

* Go to <https://start.spring.io/>
* Change Group as “com.cognizant”
* Change Artifact Id as “orm-learn”
* In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
* Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
* Click Generate and download the project as zip
* Extract the zip in root folder to Eclipse Workspace
* Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
* Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.



Code:

application.properties

# Logging  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
  
# Hibernate SQL logs  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
  
# Log pattern  
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n  
  
# Database Configuration  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=root  
  
# Hibernate  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect  
spring.application.name=orm-learn

OrmLearnApplication.java

package com.cognizant.orm\_learn;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class OrmLearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(OrmLearnApplication.class, args);  
 *LOGGER*.info("Inside main");  
 }  
  
}

MySQL Commands:

CREATE SCHEMA ormlearn;  
use ormlearn;

Output:

06-07-25 22:45:22.123 main INFO c.c.ormlearn.OrmLearnApplication main 13 Inside main

HandsOn - 2

**Difference Between JPA, Hibernate, and Spring Data JPA**

* JPA (Java Persistence API) is just a specification. It defines a set of rules and interfaces like EntityManager, Query, etc., that must be implemented by a persistence provider. It doesn't do anything by itself.
* Hibernate is an implementation of the JPA specification. It follows JPA's rules but also includes extra features such as its own query language (HQL), caching mechanisms, and more powerful options.
* Spring Data JPA sits on top of JPA and Hibernate. It simplifies the entire data access layer by automatically generating boilerplate code for common database operations (like CRUD), letting you focus only on writing business logic.
* JPA is like a contract.
* Hibernate is like the worker who fulfills the contract.
* Spring Data JPA is like a smart manager who automates most of the tasks

**Hierarchy Visualization**

Spring Data JPA

↓

JPA (Java Persistence API)

↓

Hibernate (or any JPA Provider like EclipseLink)

Analogy Example (Library System)

* Think of JPA as a set of rules every library must follow. For example, every book must have a title, author, and ID.
* Hibernate would be the staff in the library who manage the books and follow those rules. They handle how books are stored, updated, or retrieved.
* Spring Data JPA is like a super-efficient librarian who already knows what you need, finds books instantly, registers new ones automatically, and creates reports—saving you from doing anything manually.

**Code:**

1. JPA with Hibernate

Book.java

import javax.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue

private Long id;

private String title;

private String author;

}

**BookDAO.java**

public class BookDAO {

private EntityManager em;

public BookDAO(EntityManager em) {

this.em = em;

}

public void save(Book book) {

em.getTransaction().begin();

em.persist(book);

em.getTransaction().commit();

}

public Book find(Long id) {

return em.find(Book.class, id);

}

}

**2. Hibernate Directly**

**Main.java**

Session session = sessionFactory.openSession();

session.beginTransaction();

Book book = new Book();

book.setTitle("Hibernate Basics");

book.setAuthor("John Doe");

session.save(book);

session.getTransaction().commit();

session.close();

**3. Spring Data JPA**

**Book.java**

import javax.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue

private Long id;

private String title;

private String author;

}

**BookRepository.java**

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

import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByAuthor(String author);

}

**CommandLineRunner.java**

@Autowired

private BookRepository bookRepository;

@Override

public void run(String... args) {

bookRepository.save(new Book("Spring JPA", "Jane"));

List<Book> books = bookRepository.findByAuthor("Jane");

books.forEach(b -> System.out.println(b.getTitle()));

}

**Key Points**

* JPA is an API – it defines the structure and rules.
* Hibernate is an implementation – it takes the JPA rules and builds working code around it.
* Spring Data JPA is a higher-level tool – it builds on JPA and Hibernate and removes most of the repetitive code.
* If you're building something from scratch and want control: use JPA with Hibernate.
* If you want things done faster with less code: use Spring Data JPA.
* Spring Data JPA works best for CRUD-based applications where you want to avoid writing DAO layers manually.