Week – 4

Name: Lucky Awasthi

Spring Data JPA –

Quick Example • Demonstrate the need and benefit of Spring Data JPA • Evolution of ORM solutions, Hibernate XML Configuration, Hibernate Annotation Configuration, Spring Data JPA, Hibernate benefits, open source, light weight, database independent query ▪ With H2 in memory database - https://www.mkyong.com/springboot/spring-boot-spring-data-jpa/ ▪ With MySQL - https://www.mkyong.com/spring-boot/spring-bootspring-data-jpa-mysql-example/ ▪ XML Configuration Example - https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm ▪ Hibernate Configuration .

Spring Data JPA is a part of the larger Spring Data family, and it simplifies the implementation of data access layers for JPA-based applications.

**Need of Spring Data JPA**

* To **reduce boilerplate code** in DAO layers.
* To enable **easy database access** using interfaces.
* To allow **query generation from method names**.
* To seamlessly **integrate with Spring Boot** and support features like pagination, sorting, and custom queries.

**Benefits:**

* **Open Source** and maintained by the Spring team.
* **Lightweight** – no heavy configuration required.
* **Declarative queries** using method names (e.g., findByAuthor()).
* **Database independent** – supports H2, MySQL, PostgreSQL, etc.
* **Minimal configuration** – often just a few lines in application.properties.

**Evolution of ORM (Object Relational Mapping)**

1. **JDBC (Java Database Connectivity)**
   * Manual connection handling and SQL queries
   * Verbose and error-prone
2. **Hibernate with XML Configuration**
   * Mapping entities and tables via XML
   * Tedious and error-prone for large projects
   * Example: hibernate.cfg.xml and mapping files
3. **Hibernate with Annotations**
   * Used annotations like @Entity, @Table, @Id
   * Cleaner and easier than XML
4. **Spring + Hibernate Integration**
   * Added Spring’s dependency injection and transaction support
   * Still required DAOs and boilerplate code
5. **Spring Data JPA**
   * Fully abstracts the DAO layer
   * Auto-implements repositories based on method names

@Entity

public class Book {

@Id

@GeneratedValue

private Long id;

private String title;

private String author;

}

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByAuthor(String author);

}

@RestController

public class BookController {

@Autowired

private BookRepository repo;

@GetMapping("/books")

public List<Book> getAllBooks() {

return repo.findAll();

}

}

Explain the difference between Java Persistence API, Hibernate and Spring Data JPA

**Java Persistence API (JPA)**

* JPA is a **specification** (i.e., a set of interfaces and annotations) provided by Java (Java EE/Jakarta EE) for ORM (Object Relational Mapping).
* It defines **how Java objects interact with a relational database**, but it **does not provide any actual implementation**.
* Common JPA annotations include @Entity, @Table, @Id, @OneToMany, etc.
* You must use a JPA implementation like Hibernate or EclipseLink to actually use JPA in a project.

**Hibernate**

* Hibernate is a **framework** and a **popular implementation of JPA**.
* It existed even before JPA was introduced and can be used **with or without JPA**.
* Hibernate offers **additional features** beyond JPA such as:
  + Lazy loading
  + Caching
  + HQL (Hibernate Query Language)
  + Better performance tuning
* Hibernate can use **XML configuration** or **annotations** to map objects to database tables.

**Spring Data JPA**

* Spring Data JPA is a **framework built on top of JPA** to make database operations easier in **Spring applications**.
* It **simplifies the DAO layer** by generating repository code at runtime.
* Developers just define an interface (e.g., BookRepository extends JpaRepository) and Spring generates the implementation automatically.
* It integrates seamlessly with Spring Boot and reduces boilerplate code.
* Spring Data JPA typically uses **Hibernate as the default JPA provider**.