**Cognizant Deep Skilling Mandatory Hands-On Questions**

**TOPIC- Spring Data JPA with Spring Boot, Hibernate**

**Exercise 1: Spring Data JPA – Quick Example**

Ans)

# Why Spring Data JPA?

# Traditional Hibernate vs Spring Data JPA - **Traditional Hibernate:** - Required verbose XML or manual annotations - Manual query writing (HQL/SQL) - Configuration heavy - No built-in pagination, sorting - **Spring Data JPA:** - Uses repository interfaces for automatic CRUD - Query generation from method names - Bootstraps automatically with Spring Boot - Has built-in paging, sorting, and custom queries

# Spring Data JPA Example with H2

# **Project Structure**

spring-data-jpa-demo  
├── src/main/java  
│ └── com/example/demo  
│ ├── DemoApplication.java  
│ ├── model  
│ │ └── User.java  
│ ├── repository  
│ │ └── UserRepository.java  
│ └── controller  
│ └── UserController.java  
├── src/main/resources  
│ └── application.properties

## **User.java**

@Entity  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String name;  
 private String email;  
 // Getters and Setters  
}

## **UserRepository.java**

public interface UserRepository extends JpaRepository<User, Long> {  
 // Spring Data generates all basic CRUD  
}

## **UserController.java**

@RestController  
@RequestMapping("/users")  
public class UserController {  
 private final UserRepository userRepo;  
  
 public UserController(UserRepository userRepo) {  
 this.userRepo = userRepo;  
 }  
  
 @PostMapping  
 public User createUser(@RequestBody User user) {  
 return userRepo.save(user);  
 }  
  
 @GetMapping  
 public List<User> getAllUsers() {  
 return userRepo.findAll();  
 }  
}

## **application.properties (H2)**

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.h2.console.enabled=true  
spring.jpa.hibernate.ddl-auto=update

## **DemoApplication.java**

@SpringBootApplication  
public class DemoApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(DemoApplication.class, args);  
 }  
}

## **Test the API**

POST /users  
{  
 "name": "Anseet Panda",  
 "email": "anseet@example.com"  
}  
  
GET /users

# **Benefits of Spring Data JPA**

# Auto CRUD generation

# No XML or boilerplate DAO

# Integrates with H2, MySQL, PostgreSQL, etc.

# Cleaner, more maintainable

# Built-in pagination/sorting

**Exercise 2: Difference between JPA, Hibernate and Spring Data JPA**

Ans)

**(a) JPA (Java Persistence API) :-**

It is a **Java specification (JSR 338)** that defines how Java objects should be persisted to relational databases.

* JPA is **just a specification** — it **does not have any implementation**.
* You cannot use JPA alone. You need a provider (like **Hibernate**) to actually persist the data.
* Manual Code Required

(b) **Hibernate :-**

It is a **popular implementation of the JPA specification**.

* It is an **Object Relational Mapping (ORM)** tool.
* It implements the rules defined by JPA and **executes the actual SQL** to communicate with the database.
* You can use Hibernate **with or without** JPA.
* Manual Code Required

(c) **Spring Data JPA :-**

It is an **abstraction layer** built **on top of JPA and Hibernate**.

* It **removes boilerplate code** (like writing basic CRUD methods).
* You just create an interface like BookRepository, extend JpaRepository, and Spring Data JPA does the rest.
* It internally uses **Hibernate** (or any JPA provider) to perform database operations.
* Auto generates CRUD logic

**Example :**

// Book.java - Common in all 3

@Entity

public class Book {

@Id

private Long id;

private String title;

}

// Spring Data JPA - Only this is needed

public interface BookRepository extends JpaRepository<Book, Long> {}

* With Spring Data JPA, you don’t write SQL or even JPQL.
* Hibernate will still run the SQL under the hood.
* JPA defines the annotations like @Entity, @Id.