EXPERIMENT NO. 03

Title: Spring Boot with Database and Data JPA.

Outcome: Students will be able to develop Spring Boot with Database and Data JPA.

Theory:

Spring Data JPA is a part of the Spring Data project, which makes it easier to implement JPA-based repositories.
It abstracts the database layer, allowing developers to focus on business logic instead of SQL queries.

Key Annotations:

- @Entity – Marks a class as a JPA entity.
- @Id – Defines the primary key.
- @GeneratedValue – Automatically generates primary key values.
- @Repository – Indicates that the class provides CRUD operations for the entity.
- @Service – Used to write business logic.
- @RestController – Exposes REST endpoints for client interaction.

PROCEDURE / STEPS:

1. **Create Spring Boot Project**
   - Open Eclipse → File → New → Spring Starter Project.
   - Add dependencies: **Spring Web**, **Spring Data JPA**, **H2 Database**.
2. **Configure application.properties**
3. spring.datasource.url=jdbc:h2:mem:testdb
4. spring.datasource.driverClassName=org.h2.Driver
5. spring.datasource.username=sa
6. spring.datasource.password=
7. spring.jpa.hibernate.ddl-auto=update
8. spring.h2.console.enabled=true
9. **Create an Entity Class**
10. @Entity
11. public class Student {
12. @Id
13. @GeneratedValue(strategy = GenerationType.IDENTITY)
14. private Long id;
15. private String name;
16. private String department;
17. private double marks;
18.
19. // Getters, setters, constructors

20. }
21. **Create Repository Interface**
22. @Repository
23. public interface StudentRepository extends JpaRepository<Student, Long> {
24. }
25. **Create Service Layer (optional)**
26. @Service
27. public class StudentService {
28. @Autowired
29. private StudentRepository repo;
30.
31. public List<Student> getAll() { return repo.findAll(); }
32. public Student save(Student s) { return repo.save(s); }
33. public void delete(Long id) { repo.deleteById(id); }
34. }
35. **Create REST Controller**
36. @RestController
37. @RequestMapping("/students")
38. public class StudentController {
39. @Autowired
40. private StudentService service;
41.
42. @GetMapping
43. public List<Student> getAll() { return service.getAll(); }
44.
45. @PostMapping
46. public Student create(@RequestBody Student student) {
47. return service.save(student);
48. }
49.
50. @DeleteMapping("/{id}")
51. public void delete(@PathVariable Long id) {
52. service.delete(id);
53. }
54. }
55. **Run the Application**
    - Right-click → **Run As → Spring Boot App**.
    - Visit http://localhost:8080/h2-console to view the H2 database.
56. **Test API Endpoints (via Postman):**
    - **POST** /students → add a new student
    - **GET** /students → retrieve all students

- **DELETE** /students/{id} → delete student by ID

OUTPUT:

- Successfully connected Spring Boot application with H2 Database.
- CRUD operations executed successfully through REST API endpoints.
- Verified results using Postman and H2 console.

RESULT:

The Spring Boot REST API was successfully integrated with a database using **Spring Data JPA** to perform CRUD operations.

Questions:

1. What is the role of **Spring Data JPA** in a Spring Boot application?
2. Explain the purpose of the following annotations: @Entity, @Repository, and @GeneratedValue.

Conclusion : Thus, I have implemented Spring Boot with Database and Data JPA.