EXPERIMENT NO. 08: SPRING FRAMEWORK APPLICATION

Objective

To create a Java application using the Spring Framework, demonstrating the development of an industry-oriented application.

Theory

The Spring Framework is a powerful, lightweight framework used to build scalable Java applications. It provides essential features like dependency injection, aspect-oriented programming, and simplified data access. Spring Boot, an extension of Spring, helps streamline application setup by reducing boilerplate code.

Use Case Overview

We will develop a Spring Boot application to manage employee records using:
• Spring Web for REST APIs
• Spring Data JPA for database access
• H2 for in-memory database

Step 1: Define the Employee Entity

```
@Entity

public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String name;
   private String department;

   // Getters and setters
   public Long getId() { return id; }
   public void setId(Long id) { this.id = id; }
   public String getName() { return name; }
   public void setName(String name) { this.name = name; }
   public String getDepartment() { return department; }
   public void setDepartment(String department) { this.department = department; }
}
```

Step 2: Create the Repository Interface

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```

Step 3: Implement the Service Layer

```
@Service
public class EmployeeService {
    private final EmployeeRepository employeeRepository;

    @Autowired
    public EmployeeService(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    public Employee getEmployeeById(Long id) {
        return employeeRepository.findById(id).orElse(null);
    }

    public Employee createEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    public void deleteEmployee(Long id) {
        employeeRepository.deleteById(id);
    }
}
```

Step 4: Develop the REST Controller

```
@RestController

@RequestMapping("/api/employees")
public class EmployeeController {

    private final EmployeeService employeeService;
    @Autowired
    public EmployeeController(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    @GetMapping
    public List<Employee> getAllEmployees() {
```

```
        return employeeService.getAllEmployees();
    }

    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id);
    }

    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.createEmployee(employee);
    }

    @DeleteMapping("/{id}")
    public void deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployee(id);
    }
}
```

How to Run the Application

1. Create a new Spring Boot project using Spring Initializr.
2. Add dependencies: Spring Web, Spring Data JPA, H2 Database.
3. Replace or create the above classes in your source folder.
4. Run the application and test endpoints using Postman or a browser.

Sample API Endpoints

• GET `/api/employees` – Fetch all employees
• GET `/api/employees/{id}` – Fetch employee by ID
• POST `/api/employees` – Create a new employee
• DELETE `/api/employees/{id}` – Delete an employee

Conclusion

This experiment demonstrates how to build a layered Java application using the Spring Framework and Spring Boot. It includes entity creation, repository integration, business logic in the service layer, and REST API exposure via controller.