Finny Sabu **Division B** **Roll No 48**

Practical 6

Aim: Develop Spring Boot Web Application and Spring Boot RESTful web services.

Theory:

SpringBoot:

Spring Boot is a popular Java-based framework used for building production-ready applications quickly. It provides pre-configured settings for creating standalone, production-grade applications, making it easier for developers to get started with their projects. With Spring Boot, developers can take advantage of the vast collection of libraries and modules in the Spring framework, and create applications with minimal configuration. This makes it a great choice for developing microservices, web applications, and other types of software. Additionally, Spring Boot provides tools for monitoring, deploying, and testing applications, so developers can focus on writing code, rather than on setting up infrastructure.

Features of SpringBoot:

1. Auto-configuration: Spring Boot provides auto-configuration features that automatically configure the application based on the presence of certain classes and libraries on the classpath. This eliminates the need for manual configuration and makes it easier to get started with a new project.

2. Standalone applications: With Spring Boot, you can create standalone applications that can be run directly from the command line, without the need for an application server. This makes it easier to deploy applications, and also provides more control over the application environment.

3. Embeddable servers: Spring Boot includes embedded servers such as Tomcat, Jetty, and Undertow, so you don't need to install and configure a separate application server. This simplifies the deployment process and allows you to focus on writing code.

4. Actuator: Spring Boot provides an Actuator module that provides production-ready features for monitoring and managing the application. This includes metrics, health checks, and other information about the application that can be used for troubleshooting and performance optimization.

5. Command Line Interface (CLI): The Spring Boot CLI provides a command line interface for developing, testing, and deploying Spring Boot applications. With the CLI, you can quickly create and run Spring Boot applications without the need for an IDE or other tools.

6. Starter POMs: Spring Boot provides a number of "starter" POMs that make it easy to add the necessary dependencies for a given type of application. For example, if you're building a web application, you can use the spring-boot-starter-web POM to get everything you need to get started.

Overall, Spring Boot is a comprehensive framework that provides a lot of features and tools to help developers build production-ready applications quickly and easily.

Code:-

SpringBootDemoApplication.java

```
package com.springbootTest.springBootDemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class SpringBootDemoApplication {

public static void main(String[] args) {
ApplicationContext ctx = SpringApplication.run(SpringBootDemoApplication.class, args);

}

}
```

Employee.java

```
package com.springbootTest.springBootDemo.model;

import jakarta.persistence.*;

@Entity
@Table(name="employee")
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   @Column(name = "id", nullable = false)
   private Long id;
   private String name;
   private Double salary;

   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 Double getSalary() {
        return salary;
    }

    public void setSalary(Double salary) {
        this.salary = salary;
    }

    public Employee() {
    }

    public Employee(Long id, String name, Double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
}
```

EmployeeRepository.java

```
package com.springbootTest.springBootDemo.repository;

import com.springbootTest.springBootDemo.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

EmployeeService.java

```
package com.springbootTest.springBootDemo.service;

import com.springbootTest.springBootDemo.model.Employee;

import java.util.List;
public interface EmployeeService {
    Employee createEmployee(Employee e);
    Employee getEmployeeById(Long id);
    List<Employee> getAllEmployees();
    Employee updateEmployee(Employee e);
    void deleteEmployee(Long id);
}
```

EmployeeServiceImpl.java

```
package com.springbootTest.springBootDemo.service;
```

```
import com.springbootTest.springBootDemo.model.Employee;
import com.springbootTest.springBootDemo.repository.EmployeeRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
@Service
public class EmployeeServiceImpl implements EmployeeService{
   private EmployeeRepository employeeRepository;


   public void setEmployeeRepository(EmployeeRepository employeeRepository) {
      this.employeeRepository = employeeRepository;
   }

   public EmployeeServiceImpl(EmployeeRepository employeeRepository) {
      this.employeeRepository = employeeRepository;
   }

   @Override
   public Employee createEmployee(Employee e) {
      return employeeRepository.save(e);
   }

   @Override
   public Employee getEmployeeById(Long id) {
      Optional<Employee> optionalEmployee = employeeRepository.findById(id);
      return optionalEmployee.get();
   }

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

   @Override
   public Employee updateEmployee(Employee e) {
      Employee existingEmployee = employeeRepository.findById(e.getId()).get();
      existingEmployee.setName(e.getName());
      existingEmployee.setSalary(e.getSalary());
      return employeeRepository.save(existingEmployee);
   }

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

EmployeeAPI.java

```
package com.springbootTest.springBootDemo.api;
```

```
import com.springbootTest.springBootDemo.model.Employee;
import com.springbootTest.springBootDemo.service.EmployeeService;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("api/v1/employee")
public class EmployeeAPI {
    public EmployeeService getEmployeeService() {
        return employeeService;
    }

    public void setEmployeeService(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    public EmployeeAPI(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    private EmployeeService employeeService;
    @GetMapping("all") //Get All
    ResponseEntity<List<Employee>> getAllEmployees(){
        return new ResponseEntity<>(employeeService.getAllEmployees(),HttpStatus.OK);
    }
    @GetMapping("{id}") //Get One
    ResponseEntity<Employee> getEmployee(@PathVariable("id")Long id){
        return new ResponseEntity<>(employeeService.getEmployeeById(id), HttpStatus.OK);
    }

    @PostMapping() //Create One
    ResponseEntity<Employee> createEmployee(@RequestBody Employee employee){
        return new ResponseEntity<>(employeeService.createEmployee(employee), HttpStatus.CREATED);
    }
    @PutMapping("{id}")  //Update One
    ResponseEntity<Employee> updateEmployee(@PathVariable("id") Long id, @RequestBody Employee
e) {
        e.setId(id);
        return new ResponseEntity<>(employeeService.updateEmployee(e), HttpStatus.OK);
    }

    @DeleteMapping("{id}") //Delete One
    ResponseEntity<String> deleteEmployee(@PathVariable("id") Long id) {
        employeeService.deleteEmployee(id);
        return new ResponseEntity<>("User successfully deleted!", HttpStatus.OK);
    }
}
```

application.properties

```
server.port=9090
spring.datasource.url=jdbc:mysql://localhost:3306/java_app
spring.datasource.username=java_app
spring.datasource.password=java_app_pass
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.0.2</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.springbootTest</groupId>
<artifactId>springBootDemo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springBootDemo</name>
<description>Demo Maven Project created by Paxy and Nero</description>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

```
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

Output :-

Conclusion:

I have successfully understood how to use SpringBoot to create a REST API