RESTful Web Services

Introduction to RESTful Web Services

REST stands for REpresentational State Transfer. It was developed by Roy Thomas Fielding, who also developed HTTP. The primary goal of RESTful web services is to make web services more effective by defining services using concepts already present in HTTP. REST is an architectural approach, not a protocol.

Key Features of REST:

- REST does not define a standard message exchange format.
- Both XML and JSON can be used for REST services, though JSON is more popular.
- The key abstraction in REST is a resource, which can be accessed through a Uniform Resource Identifier (URI).

HTTP Methods:

- **GET**: Reads a resource.
- **PUT**: Updates an existing resource.
- **POST**: Creates a new resource.
- **DELETE**: Deletes a resource.

Example URIs:

- **POST /users**: Creates a user.
- **GET /users/{id}**: Retrieves the details of a user.
- **GET /users**: Retrieves the details of all users.
- **DELETE /users**: Deletes all users.
- **DELETE /users/{id}**: Deletes a specific user.
- **GET /users/{id}/posts/{post_id}**: Retrieves the details of a specific post.
- **POST /users/{id}/posts**: Creates a post for a user.

HTTP Status Codes:

- **200**: Success
- **201**: Created
- **401**: Unauthorized
- **404**: Resource Not Found
- **500**: Server Error

RESTful Service Constraints:

1. There must be a service producer and consumer.
2. The service must be stateless.
3. Results must be cacheable.
4. The interface should uniformly expose resources.
5. The service should assume a layered architecture.

Advantages of RESTful Web Services:

- Platform-independent.
- Can be written in any programming language.
- Supports multiple data formats (e.g., JSON, XML, HTML, text).
- Faster compared to SOAP due to the lack of strict specifications.
- Reusable and language-neutral.

Spring Boot CRUD Operations: A Comprehensive Guide

1. Introduction to CRUD Operations

CRUD represents the four basic operations performed on persistent storage:

- **Create**: Add new records to the database.
- **Read**: Retrieve existing records.
- **Update**: Modify existing records.
- **Delete**: Remove records from the database.

2. Project Setup and Dependencies

2.1. Project Setup

A. Create a New Spring Boot Project

There are multiple ways to create a Spring Boot project:

1. **Using Spring Initializr (Recommended)**
    - Go to https://start.spring.io/
    - Choose options:
        - Project: Maven
        - Language: Java
        - Spring Boot: Latest stable version
        - Project Metadata:
            - Group: com.example
            - Artifact: employee-crud

        - Packaging: Jar
        - Java: 17 or 21
2. **Dependencies to Add** Select these dependencies:
    - Spring Web
    - Spring Data JPA
    - H2 Database
    - Validation (optional but recommended)
3. **Generate and Download**
    - Click "Generate"
    - Download the ZIP file
    - Extract to your preferred workspace

Maven Dependencies

```
<dependencies>
    <!-- Spring Boot Starter Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Spring Boot Starter Web -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- H2 Database (for demonstration) -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

A. application.properties

Open `src/main/resources/application.properties` and add:

```
# Database Configuration

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto=update

 # Enable H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console
```

3. Complete CRUD Implementation

3.1 Entity Class

Create `Employee.java` in `model` package and Add:

```
package com.example.employeecrud.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;

    private String name;
    private String department;
    private Double salary;

    // Constructors
    public Employee() {}

    public Employee(String name, String department, Double salary) {
        this.name = name;
        this.department = department;
        this.salary = salary;
    }

    // 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; }

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

3.2 Repository Interface

Create `EmployeeRepository.java` and Add:

```
package com.example.demo.repository;

import com.example.demo.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> {

    List<Employee> findByDepartment(String department);

    List<Employee> findBySalaryGreaterThan(Double salary);

}
```

3.3 Service Layer

Create `EmployeeService.java` and And:

```
Package com.example.employeecrud.service;
import com.example.demo.model.Employee;
import com.example.demo.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

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

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

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

public Optional<Employee> getEmployeeById(Long id) {
    return employeeRepository.findById(id);
}

public Employee updateEmployee(Long id, Employee employeeDetails) {
    Employee employee = employeeRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("Employee not found"));

    employee.setName(employeeDetails.getName());
    employee.setDepartment(employeeDetails.getDepartment());
    employee.setSalary(employeeDetails.getSalary());

    return employeeRepository.save(employee);
}

public void deleteEmployee(Long id) {
    Employee employee = employeeRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("Employee not found"));
    employeeRepository.delete(employee);
}

public void deleteAllEmployees() {
    employeeRepository.deleteAll();
}

public List<Employee> getEmployeesByDepartment(String department) {
    return employeeRepository.findByDepartment(department);
```

```
    }

    public List<Employee> getEmployeesWithSalaryAbove(Double salary) {
        return employeeRepository.findBySalaryGreaterThan(salary);
    }
}
```

3.4 Controller Layer

Create `EmployeeController.java` and Add:

```
package com.example.employeecrud.controller;
import com.example.demo.model.Employee;
import com.example.demo.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

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

    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @GetMapping("/{id}")
```

```
    public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }

    @PutMapping("/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable Long id,
@RequestBody Employee employeeDetails) {
        Employee updatedEmployee = employeeService.updateEmployee(id,
employeeDetails);
        return ResponseEntity.ok(updatedEmployee);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {
        employeeService.deleteEmployee(id);
        return ResponseEntity.ok().build();
    }

    @GetMapping("/department/{department}")
    public List<Employee> getEmployeesByDepartment(@PathVariable String
department) {
        return employeeService.getEmployeesByDepartment(department);
    }

    @GetMapping("/salary-above/{salary}")
    public List<Employee> getEmployeesAboveSalary(@PathVariable Double salary) {
        return employeeService.getEmployeesWithSalaryAbove(salary);
    }
}
```

7. Running the Application

A. Using IDE

- Open the project in IntelliJ IDEA or Eclipse
- Right-click on the main application class
- Select "Run" or "Run as Spring Boot Application"

8. Testing the API

You can test the API using:

1. **Postman**
2. **cURL**
3. **Browser** (for GET requests)
4. **H2 Console** (for database verification)

Example API Calls

1. Create Employee (POST):

   ```
   curl -X POST http://localhost:8080/api/employees -H "Content-Type: application/json" -d '{"name":"John Doe","department":"IT","salary":50000.0}'
   ```

2. Get All Employees (GET):

   ```
   curl http://localhost:8080/api/employees
   ```

3. Update Employee (PUT):

   ```
   curl -X PUT http://localhost:8080/api/employees/1 -H "Content-Type: application/json" -d '{"name":"John Smith","department":"HR","salary":55000.0}'
   ```

4. Delete Employee (DELETE):

   ```
   curl -X DELETE http://localhost:8080/api/employees/1
   ```

4. Additional Considerations

Validation

- Use `@Valid` annotations for validation constraints in the entity.
- Handle validation errors using `BindingResult`.

Error Handling

- Implement a global exception handler using `@ControllerAdvice`.

Pagination and Sorting

- Use Pageable in repository methods to implement pagination.

Transaction Management

- Use @Transactional for managing complex database operations.

5. Configuration (application.properties)

```
# Database Configuration

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Hibernate ddl auto (create, create-drop, validate, update)

spring.jpa.hibernate.ddl-auto=update

# Enable H2 Console

spring.h2.console.enabled=true
```