RESTful Web Services Integration

Table of Contents

Objectives

Problem Statement

Implementation Details

3.1 Creating Static Employee Data

3.2 Employee REST Service

3.3 Department REST Service

Testing with Postman

Angular Integration

1. Objectives

Demonstrate integration of RESTful Web Services (GET operations) with:

Proper layered architecture (Controller → Service → DAO)

Spring XML configuration for static data

Postman testing procedures

Angular frontend consumption

2. Problem Statement

Display Employee List and Edit Employee Form using RESTful Web Services

Key Requirements:

Migrate hardcoded Angular data to Spring REST services

Three-phase implementation:

Create static employee data in Spring XML

Develop employee REST service

Modify Angular components to consume services

3. Implementation Details

3.1 Creating Static Employee Data (employee.xml)

<beans>

<!-- Departments -->

<bean id="dept1" class="com.example.model.Department">

<property name="id" value="1"/>

<property name="name" value="Engineering"/>

</bean>

<bean id="dept2" class="com.example.model.Department">

<property name="id" value="2"/>

<property name="name" value="Human Resources"/>

</bean>

<!-- Employees -->

<bean id="emp1" class="com.example.model.Employee">

<property name="id" value="101"/>

<property name="name" value="John Doe"/>

<property name="department" ref="dept1"/>

<property name="skills">

<list>

<ref bean="javaSkill"/>

<ref bean="springSkill"/>

</list>

</property>

</bean>

<!-- Additional employees... -->

</beans>

EmployeeDao Implementation:

public class EmployeeDao {

public static ArrayList<Employee> EMPLOYEE\_LIST;

public EmployeeDao() {

// Initialize from XML configuration

ApplicationContext context = new ClassPathXmlApplicationContext("employee.xml");

EMPLOYEE\_LIST = new ArrayList<>(context.getBeansOfType(Employee.class).values());

}

public ArrayList<Employee> getAllEmployees() {

return EMPLOYEE\_LIST;

}

}

3.2 Employee REST Service

Service Layer:

@Service

public class EmployeeService {

@Autowired

private EmployeeDao employeeDao;

@Transactional

public List<Employee> getAllEmployees() {

return employeeDao.getAllEmployees();

}

}

Controller:

@RestController

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping("/employees")

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

}

3.3 Department REST Service

DAO Layer:

public class DepartmentDao {

public static ArrayList<Department> DEPARTMENT\_LIST;

public DepartmentDao() {

ApplicationContext context = new ClassPathXmlApplicationContext("employee.xml");

DEPARTMENT\_LIST = new ArrayList<>(context.getBeansOfType(Department.class).values());

}

public List<Department> getAllDepartments() {

return DEPARTMENT\_LIST;

}

}

Service Layer:

@Service

public class DepartmentService {

@Autowired

private DepartmentDao departmentDao;

public List<Department> getAllDepartments() {

return departmentDao.getAllDepartments();

}

}

Controller:

@RestController

public class DepartmentController {

@Autowired

private DepartmentService departmentService;

@GetMapping("/departments")

public List<Department> getAllDepartments() {

return departmentService.getAllDepartments();

}

}

4. Testing with Postman

Employee Service Test

URL: GET http://localhost:8080/employees

Expected Response:

[

{

"id": 101,

"name": "John Doe",

"department": {

"id": 1,

"name": "Engineering"

},

"skills": [

{"id": 1, "name": "Java"},

{"id": 2, "name": "Spring"}

]

}

]

Department Service Test

URL: GET http://localhost:8080/departments

Expected Response:

[

{"id": 1, "name": "Engineering"},

{"id": 2, "name": "Human Resources"}

]

Verification Steps:

Check response status code (200 OK)

Validate JSON structure

Verify Content-Type header (application/json)

Check server logs for service invocation

5. Angular Integration

Employee Service (Angular):

@Injectable()

export class EmployeeService {

private apiUrl = 'http://localhost:8080/employees';

constructor(private http: HttpClient) {}

getEmployees(): Observable<Employee[]> {

return this.http.get<Employee[]>(this.apiUrl);

}

}

Component Implementation:

export class EmployeeListComponent implements OnInit {

employees: Employee[];

constructor(private employeeService: EmployeeService) {}

ngOnInit() {

this.employeeService.getEmployees().subscribe(

data => this.employees = data,

error => console.error(error)

);

}

}

Appendix: Architecture Diagram

[Angular UI] → [REST Controller] → [Service Layer] → [DAO Layer] → [XML Configuration]

↑

[Postman]

Key Features:

Proper separation of concerns

Transaction management

Case-insensitive search

Comprehensive error handling

RESTful best practices