

SPRING BOOT EMPLOYEE MANAGEMENT REST API

(Internship Task Submission - Alfido Tech)

1. PROJECT OVERVIEW

This project is a Spring Boot based RESTful web application developed as part of my internship task.

The application provides a complete Employee Management system that supports CRUD (Create, Read, Update, Delete) operations. Employee data is stored in a MySQL database and the APIs are tested and documented using Swagger UI.

2. TECHNOLOGIES USED

- Java 17
 - Spring Boot
 - Spring Data JPA
 - MySQL Database
 - Maven
 - Swagger UI (OpenAPI)
 - VS Code
-

3. FEATURES IMPLEMENTED

- Create Employee using POST API
- Fetch All Employees using GET API
- Update Employee using PUT API
- Delete Employee using DELETE API
- RESTful API design
- Swagger UI integration for API testing and documentation

4. API ENDPOINTS

```
GET  /employees  
POST /employees  
PUT  /employees/{id}  
DELETE /employees/{id}
```

5. PROJECT WORKFLOW

1. Client sends HTTP request using Swagger UI or Thunder Client.
 2. Controller layer receives the request.
 3. Service layer processes business logic.
 4. Repository layer interacts with MySQL database.
 5. Response is returned to the client in JSON format.
-

6. SCREENSHOTS ATTACHED

- Application running in VS Code (Tomcat started)

The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. The terminal window displays the following log output:

```
2025-12-24T17:50:35.732+05:30 INFO 14868 --- [main] c.e.employeeapi.EmployeeapiApplication : Started EmployeeapiApplication in 3.742 seconds (process running for 4.017)
2025-12-24T17:51:51.500+05:30 INFO 14868 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-12-24T17:51:51.500+05:30 INFO 14868 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-12-24T17:51:51.502+05:30 INFO 14868 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
2025-12-24T17:51:51.502+05:30 INFO 14868 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
2025-12-24T17:51:52.115+05:30 INFO 14868 --- [io-8080-exec-10] o.springdoc.api.AbstractOpenApiResource : Init duration for springdoc-openapi is: 254 ms
Hibernate: select e1_.id,e1_.name,e1_.role from employee e1_ where e1_.id=?
Hibernate: update employee set name=?,role=? where id=?
Hibernate: select e1_.id,e1_.name,e1_.role from employee e1_ []

```

Below the terminal, status indicators show 0 errors, 0 warnings, and 0 info messages. The status bar indicates "Ln 60, Col 25 (1757 selected) Tab Size:". To the right of the terminal, a sidebar lists several open terminals: powershell, cmd, and java, with the java terminal currently selected.

Figure 1: Spring Boot application started successfully with embedded Tomcat server and MySQL database integration

- Swagger UI showing all Employee API

The screenshot shows the Swagger UI interface for an API. At the top, the URL is `localhost:8080/swagger-ui/index.html#/employee-controller`. The main title is "OpenAPI definition v0 OAS 3.0". Below it, there's a "Servers" dropdown set to `http://localhost:8080 - Generated server url`. The main content area is titled "employee-controller" and shows a "PUT /employees/{id}" endpoint. This endpoint has a "Parameters" section with a single parameter "id" (required, integer, path) set to the value 3. It also has a "Request body required" section with a dropdown for "application/json". A "Try it out" button is visible in the top right of this section.

This screenshot continues from the previous one, showing the "Request body" and "Responses" sections for the "PUT /employees/{id}" endpoint. In the "Request body" section, the media type is set to "application/json", and the example value is a JSON object with fields "id", "name", and "role", all set to their respective types ("integer", "string", "string"). In the "Responses" section, the 200 OK response is listed with a "Media type" dropdown set to "*/*". The example value for this response is the same JSON object as the request body. There are "Links" and "No links" sections next to the response table.

localhost:8080/swagger-ui/index.html#/employee-controller

DELETE /employees/{id}

Parameters

Name	Description
id <small>* required</small>	integer(\$int32) (path)

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header

Example Value | Schema

string

localhost:8080/swagger-ui/index.html#/employee-controller/createEmployee

GET /employees

Parameters

No parameters

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header

Example Value | Schema

```
[{"id": 0, "name": "string", "role": "string"}]
```

The screenshot shows the Swagger UI interface for a REST API. At the top, the URL is displayed as `localhost:8080/swagger-ui/index.html#/employee-controller/createEmployee`. The main content area is divided into several sections:

- Method and Endpoint:** POST /employees
- Parameters:** No parameters.
- Request body (required):** application/json. A sample JSON object is shown:

```
{
  "id": 0,
  "name": "string",
  "role": "string"
}
```
- Responses:**
 - Code: 200 OK** (Media type: */*)
 - Example Value | Schema
 - A sample JSON object is shown:

```
{
  "id": 0,
  "name": "string",
  "role": "string"
}
```
 - No links**
- Schemas:** Employee

Figure 2: Swagger UI displaying the OpenAPI documentation and available Employee Management REST APIs.

- GET employees API response

The screenshot shows the Swagger UI interface for a GET request to the '/employees' endpoint. The URL is `localhost:8080/swagger-ui/index.html#/employee-controller/createEmployee`. The method is set to GET. Under 'Parameters', it says 'No parameters'. Below that are 'Responses' and 'Server response' sections. The 'Server response' section shows a 200 status code with a JSON response body containing two employee objects:

```
[{"id": 2, "name": "Suresh", "role": "Backend Developer"}, {"id": 3, "name": "Suresh Kumar", "role": "Senior Backend Developer"}]
```

There are 'Execute' and 'Clear' buttons at the top of the responses section.

This screenshot shows the same API endpoint but with more detailed information. It includes the response body, response headers, and a code example for the 200 OK status code. The response body is identical to the one above. The response headers include:

```
connection: keep-alive
content-type: application/json
date: Thu, 25 Dec 2025 05:26:06 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

The 'Responses' section shows a 200 OK status with a media type dropdown set to '*/*'. The 'Links' section indicates 'No links'.

Figure 3: GET /employees API response showing the list of all employees retrieved from the MySQL database in JSON format.

- POST employee API response

The screenshot shows the Swagger UI interface for a POST request to the '/employees' endpoint. The 'Parameters' section indicates 'No parameters'. The 'Request body' section is set to 'application/json' and contains the following JSON payload:

```
{ "name": "Ravi Kumar", "role": "Java Developer" }
```

Below the request body is a 'Responses' section containing a 'Curl' example:

```
curl -X 'POST' \
```

The screenshot shows the detailed response for the POST /employees endpoint. It includes a 'Responses' section with a 'Curl' example:

```
curl -X 'POST' \  
  'http://localhost:8080/employees' \  
  -H 'accept: */*' \  
  -H 'Content-Type: application/json' \  
  -d '{' \  
    "name": "Ravi Kumar", \  
    "role": "Java Developer" \  
  }'
```

Below the responses are sections for 'Request URL' (http://localhost:8080/employees) and 'Server response'.

Server response

Code	Details	Links
200	<p>Response body</p> <pre>({ "id": 4, "name": "Ravi Kumar", "role": "Java Developer" })</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: 12-25-2020 05:33:19 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>	Download
Responses		
Code	Description	Links
200	OK	No links

The screenshot shows the Swagger UI interface for a POST /employees API endpoint. The request URL is `http://localhost:8080/employees`. The server response is a 200 OK status with a JSON body containing an employee record with id 4, name "Ravi Kumar", and role "Java Developer". The response headers include connection, content-type, date, keep-alive, and transfer-encoding. The responses section shows a media type dropdown set to `*/*`, which controls the Accept header. An example value for the schema is provided.

Figure 4: POST /employees API request and response showing successful creation of a new employee record.

- PUT employee API response

The screenshot shows the configuration for a PUT /employees/{id} API endpoint. The method is PUT and the path is `/employees/{id}`. The parameters section shows a required parameter `id` of type integer(\$int32) with a value of 3. The request body is required and has a schema of application/json, containing a JSON object with name and role fields. The execute and clear buttons are at the bottom.

The screenshot shows the Swagger UI interface for a PUT request to the '/employees/{id}' endpoint. The URL is `http://localhost:8080/employees/3`. The response code 200 indicates success, with a response body containing the updated employee details: `{"id": 3, "name": "Ravi Kumar", "role": "Senior Java Developer"}`. The response headers include `connection: keep-alive`, `content-type: application/json`, `date: Thu, 25 Dec 2025 05:40:13 GMT`, `keep-alive: timeout=60`, and `transfer-encoding: chunked`.

Figure 5: PUT /employees/{id} API request and response showing successful update of employee details.

- DELETE employee API response

The screenshot shows the Swagger UI interface for a DELETE request to the '/employees/{id}' endpoint. The URL is `http://localhost:8080/employees/4`. The response code 200 indicates success, with a response body containing the deleted employee details: `{"id": 4, "name": "Ravi Kumar", "role": "Senior Java Developer"}`. The response headers include `connection: keep-alive`, `content-type: application/json`, `date: Thu, 25 Dec 2025 05:40:13 GMT`, `keep-alive: timeout=60`, and `transfer-encoding: chunked`.

The screenshot shows the Swagger UI interface for a DELETE API endpoint. The URL in the address bar is `localhost:8080/swagger-ui/index.html#/employee-controller/createEmployee`. The main content area is titled "Server response". It shows a successful response (HTTP 200) with the following details:

- Code**: 200
- Response body**:
Employee deleted successfully
- Response headers**:
connection: keep-alive
content-length: 29
content-type: text/plain;charset=UTF-8
date: Thu, 25 Dec 2025 05:53:09 GMT
keep-alive: timeout=60
- Download** button

Below this, under "Responses", there is a table:

Code	Description	Links
200	OK	No links

Under the "OK" row, there is a "Media type" section with a dropdown menu set to `*/*`, which is noted to control the Accept header. There are also "Example Value" and "Schema" links.

Figure 6: DELETE /employees/{id} API execution showing successful removal of an employee record from the MySQL database.

- MySQL employee table data

```
Default      + | - 

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE employee_db;
Query OK, 1 row affected (0.01 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| employee_db |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.04 sec)

mysql> USE employee_db;
Database changed
mysql> CREATE TABLE employee (
    ->     id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(100),
    ->     role VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW TABLES;
+-----+
| Tables_in_employee_db |
+-----+
| employee |
+-----+
1 row in set (0.03 sec)

mysql> INSERT INTO employee (name, role)
    -> VALUES ('Ravi', 'Java Developer');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM employee;
+----+----+----+
| id | name | role        |
+----+----+----+
| 1  | Ravi | Java Developer |
+----+----+----+
1 row in set (0.00 sec)

mysql>
```

Figure 7: MySQL database setup showing employee_db, creation of employee table, and inserted employee records used by the Spring Boot application.

7. CODE SNIPPETS

- EmployeeController.java

Java

```
package com.example.employeeapi.controller;

import com.example.employeeapi.entity.Employee;
import com.example.employeeapi.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

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

    @Autowired
    private EmployeeService employeeService;

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

    // POST
}
```

```
@PostMapping  
public Employee createEmployee(@RequestBody Employee employee) {  
    return employeeService.createEmployee(employee);  
}  
  
// PUT (THIS FIXES YOUR ERROR)  
@PutMapping("/{id}")  
public Employee updateEmployee(  
    @PathVariable int id,  
    @RequestBody Employee employee) {  
    return employeeService.updateEmployee(id, employee);  
}  
  
// DELETE  
@DeleteMapping("/{id}")  
public String deleteEmployee(@PathVariable int id) {  
    employeeService.deleteEmployee(id);  
    return "Employee deleted successfully";  
}  
}
```

-EmployeeService.java

Java

```
package com.example.employeeapi.service;  
  
import com.example.employeeapi.entity.Employee;  
import java.util.List;  
  
public interface EmployeeService {
```

```
List<Employee> getAllEmployees();

Employee createEmployee(Employee employee);

Employee updateEmployee(int id, Employee employee);

void deleteEmployee(int id);
}
```

-EmployeeServiceImpl.java

```
Java
package com.example.employeeapi.service;

import com.example.employeeapi.entity.Employee;
import com.example.employeeapi.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class EmployeeServiceImpl implements EmployeeService {

    @Autowired
    private EmployeeRepository employeeRepository;

    // GET ALL
    @Override
```

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

// CREATE
@Override
public Employee createEmployee(Employee employee) {
    return employeeRepository.save(employee);
}

// UPDATE
@Override
public Employee updateEmployee(int id, Employee employee) {
    Employee existingEmployee = employeeRepository.findById(id)
        .orElseThrow(() -> new RuntimeException("Employee
not found"));

    existingEmployee.setName(employee.getName());
    existingEmployee.setRole(employee.getRole());

    return employeeRepository.save(existingEmployee);
}

// DELETE
@Override
public void deleteEmployee(int id) {
    employeeRepository.deleteById(id);
}
```

-Employee.java

```
Java
package com.example.employeeapi.entity;

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

@Entity
@Table(name = "employee")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String role;

    public Employee() {
    }

    public Employee(Long id, String name, String role) {
        this.id = id;
        this.name = name;
        this.role = role;
    }

    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 getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}
```

8.COMMANDS USED

The following commands were used to build and run the Spring Boot application:

1.

Shell

```
mvn clean install
```

This command downloads dependencies and builds the project.

2.

Shell

```
mvn spring-boot:run
```

This command starts the Spring Boot application on the embedded Tomcat server.

3. <http://localhost:8080/employees>

Used to test GET API in browser.

4. <http://localhost:8080/swagger-ui/index.html>

Used to access Swagger UI for API testing.

9. PROJECT LINKS

GitHub Repository:

<https://github.com/Revathi2006-op/Alfido-Tech-Internship>

Swagger UI URL:

<http://localhost:8080/swagger-ui/index.html>

10. CONCLUSION

This project helped me gain hands-on experience in developing RESTful APIs using Spring Boot,
integrating MySQL database using Spring Data JPA, and documenting APIs using Swagger UI.
It improved my understanding of backend development and layered application architecture.

Submitted by:

Name: REVATHI S