FPT Software

JAVA SPRING FRAMEWORK

Lab Guides

Document Code	25e-BM/HR/HDCV/FSOFT
Version	1.0
Effective Date	01/09/2024

Hanoi, 08/2024

RECORD OF CHANGES

No	Effective Date	Change Description	Reason	Reviewer	Approver
1	06/08/2024	Create a new Lab	Create new		VinhNV

Contents

CODE:	JSFW_Lab_04_Opt2
TYPE:	LONG
LOC:	200
DURATION:	180 MINUTES

Java Spring Framework Introduction

Objectives:

- Understand DAO Pattern: Learn how to use the Data Access Object (DAO) pattern with Spring MVC to manage entities.
- Configure Spring MVC: Gain experience in configuring and using Spring MVC for managing entities and interacting with a PostgreSQL database.
- Implement CRUD Operations: Implement and test CRUD (Create, Read, Update, Delete) operations using Spring MVC and PostgreSQL for both Employee and Department entities.

Lab Specifications:

In a University Management System, you will manage two entities, Employee and Department, using DAO classes to interact with a PostgreSQL database. You will implement and test CRUD operations for both entities.

Problem Description:

- Trainees must implement and test methods for managing employees and departments using DAO patterns and PostgreSQL for persistence.

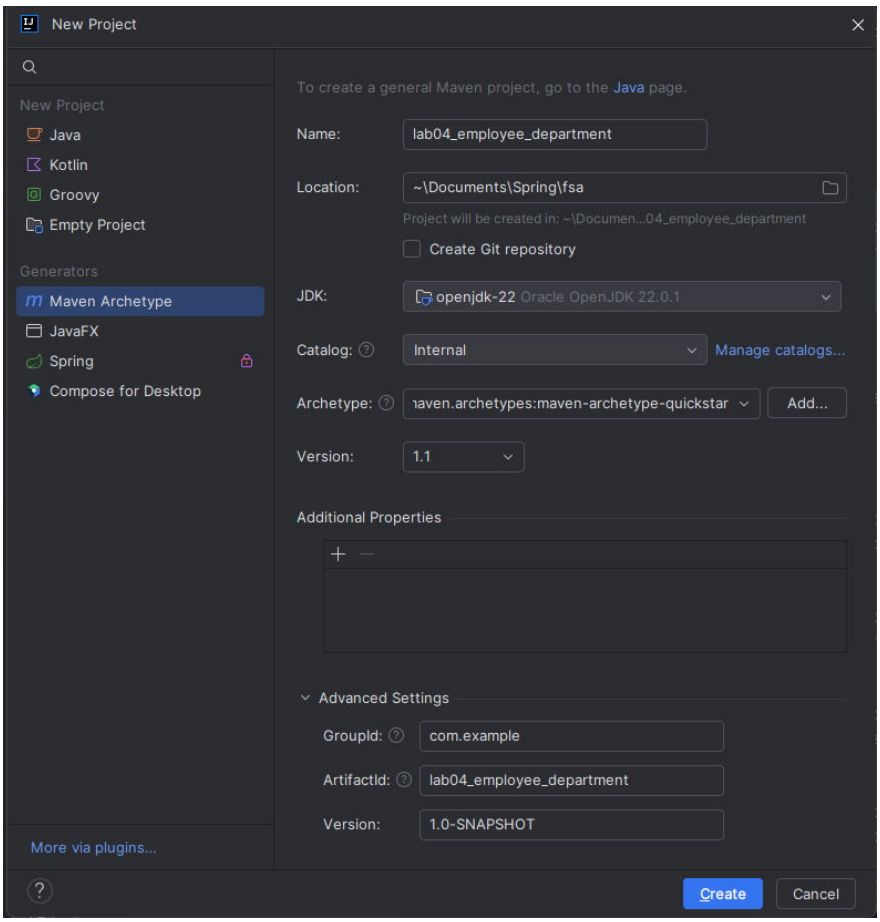
Prerequisites:

- Using Java SDK version 8.0 at least.
- Using Maven.
- Using Spring Framework 5.0 or higher version.

Guidelines:

Step 1: Extend the previous project to include dependency injection:

- Open IntelliJ IDEA.
- Click on File -> New -> Project....
- Select Maven from the project types.
- Click Next and set the project name to **lab04_employee_department**
- Set the groupId to com.example and artifactId to **lab04_employee_department**
- Click **Create**.



Step 2: Add dependencies and configuration into pom.xml file:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.7.5</version>
  <relativePath/>
</parent>
```

Add the Spring Core dependency to your pom.xml file: put these to <dependencies> tag:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
```

```
  <version>42.7.3</version>
</dependency>
```

Step 3: Configure Data Source and JPA:

Create a application.properties file in src/main/resources with PostgreSQL configuration:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
spring.datasource.username=postgres
spring.datasource.password=1234567890
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.default_schema=public
```

Step 4: Prepare Data:

```
CREATE DATABASE dep_emp
CREATE TABLE Department (
    id BIGINT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);
CREATE TABLE Employee (
    id BIGINT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    position VARCHAR(255) NOT NULL,
    department_id BIGINT,
    FOREIGN KEY (department_id) REFERENCES Department(id)
);
```

```
-- Insert data into the Department table
INSERT INTO Department (id, name) VALUES
(1, 'Human Resources'),
(2, 'IT'),
(3, 'Finance'),
(4, 'Marketing');


-- Insert data into the Employee table
INSERT INTO Employee (id, name, position, department_id) VALUES
(1, 'Alice Smith', 'HR Manager', 1),
(2, 'Bob Johnson', 'Software Engineer', 2),
(3, 'Carol White', 'Financial Analyst', 3),
```

```
(4, 'David Brown', 'Marketing Specialist', 4);
```

Step 5: Create entity classes:

- Create **Department** class in **model** package:

```
package com.example.model;

public class Department {
    private Long id;
    private String name;

    public Department() {

    }

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

    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;
    }
}
```

- Create **Employee** class in **model** package:

```
package com.example.model;

public class Employee {
   private Long id;
   private String name;
   private String position;
   private Department department; // Ensure this field exists and is properly set

   public  Employee(){

   }
   public Employee(String name, String position, Department department) {
      this.name = name;
      this.position = position;
      this.department = department;
   }

   public Employee(Long id, String name, String position, Department department) {
```

```
        this.id = id;
        this.name = name;
        this.position = position;
        this.department = department;
    }

    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 getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public Department getDepartment() {
        return department;
    }

    public void setDepartment(Department department) {
        this.department = department;
    }
}
```

Step 6: Create **DBUtil** class in util package

This utility class handles the database connection manually.

```
package com.example.util;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.stereotype.Component;

@Component
public class DBUtil {
```

```
    private final JdbcTemplate jdbcTemplate;

    // Constructor to initialize JdbcTemplate
    public DBUtil(
            @Value("${spring.datasource.url}") String url,
            @Value("${spring.datasource.username}") String username,
            @Value("${spring.datasource.password}") String password,
            @Value("${spring.datasource.driver-class-name}") String
driverClassName) {

        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(driverClassName);
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    public JdbcTemplate getJdbcTemplate() {
        return jdbcTemplate;
    }
}
```

Step 7: Create **DepartmentDAO**, **EmployeeDAO** class in dao package:

These classes will handle the database operations using JdbcTemplate.

```
package com.example.dao;

import com.example.model.Department;
import com.example.util.DBUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Component;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Component
public class DepartmentDAO {

    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public DepartmentDAO(DBUtil dbUtil) {
        this.jdbcTemplate = dbUtil.getJdbcTemplate();
    }

    public List<Department> getAllDepartments() {
        return jdbcTemplate.query("SELECT * FROM departments", new
DepartmentRowMapper());
    }

    public Department getDepartmentById(Long id) {
        return jdbcTemplate.queryForObject("SELECT * FROM departments WHERE id =
?", new DepartmentRowMapper(), id);
    }
```

```
    public void addDepartment(Department department) {
        jdbcTemplate.update("INSERT INTO departments (name) VALUES (?)",
department.getName());
    }

    public void updateDepartment(Department department) {
        jdbcTemplate.update("UPDATE departments SET name = ? WHERE id = ?",
department.getName(), department.getId());
    }

    public void deleteDepartment(Long id) {
        jdbcTemplate.update("DELETE FROM departments WHERE id = ?", id);
    }

    public void createTable() {
        jdbcTemplate.execute("CREATE TABLE IF NOT EXISTS departments (" +
                "id BIGSERIAL PRIMARY KEY, " +
                "name VARCHAR(255) NOT NULL)");
    }

    public void dropTable() {
        jdbcTemplate.execute("DROP TABLE IF EXISTS departments");
    }

    private static class DepartmentRowMapper implements RowMapper<Department> {
        @Override
        public Department mapRow(ResultSet rs, int rowNum) throws SQLException {
            return new Department(rs.getLong("id"), rs.getString("name"));
        }
    }
}
```

Step 7: Create Views

- View for **Department** function

a. department-list.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Department List</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
<div class="container mt-4">
    <a href="/" class="btn btn-secondary mb-3">Back to Homepage</a>
    <h1 class="mb-4">Department List</h1>

    <!-- Add Department Button -->
    <a href="/department/add" class="btn btn-primary mb-3">Add Department</a>

    <!-- Department Table -->
    <table class="table table-striped">
        <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Actions</th>
```

```
        </tr>
        </thead>
        <tbody>
        <tr th:each="department : ${departments}">
            <td th:text="${department.id}"></td>
            <td th:text="${department.name}"></td>
            <td>
                <!-- View Details Button -->
                <a th:href="@{/department/{id}(id=${department.id})}" class="btn
btn-info btn-sm">View</a>
                <!-- Edit Button -->
                <a th:href="@{/department/edit/{id}(id=${department.id})}"
class="btn btn-warning btn-sm">Edit</a>
                <!-- Delete Button -->
                <a th:href="@{/department/delete/{id}(id=${department.id})}"
class="btn btn-danger btn-sm">Delete</a>
            </td>
        </tr>
        </tbody>
    </table>
</div>

<!-- Bootstrap JS and dependencies -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.1/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

b. department-add.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Department</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <h1>Add New Department</h1>
    <form th:action="@{/department/add}" method="post">
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
required>
        </div>
        <button type="submit" class="btn btn-primary">Add Department</button>
        <a href="/departments" class="btn btn-secondary">Cancel</a>
    </form>
</div>
</body>
</html>
```

c. department-detail.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Department Detail</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <h1>Department Detail</h1>
    <div class="card">
        <div class="card-body">
            <h5 class="card-title" th:text="${department.name}"></h5>
            <p class="card-text"><strong>ID:</strong> <span
th:text="${department.id}"></span></p>
            <a href="/departments" class="btn btn-secondary">Back to List</a>
            <a th:href="@{/department/edit/{id}(id=${department.id})}"
class="btn btn-warning">Edit</a>
            <a th:href="@{/department/delete/{id}(id=${department.id})}"
class="btn btn-danger">Delete</a>
        </div>
    </div>
</div>
</body>
</html>
```

d. department-update.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit Department</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
<div class="container mt-5">
    <h1>Edit Department</h1>
    <form th:action="@{/department/update/{id}(id=${department.id})}"
method="post">
        <input type="hidden" name="id" th:value="${department.id}"/>
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
th:value="${department.name}" required>
        </div>
        <button type="submit" class="btn btn-primary">Update Department</button>
        <a href="/departments" class="btn btn-secondary">Cancel</a>
    </form>
</div>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.1/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

e. department-delete.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Delete Employee</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-4">
    <h1 class="mb-4">Delete Employee</h1>
    <p>Are you sure you want to delete this employee?</p>
    <ul>
        <li><strong>Name:</strong> <span th:text="${employee.name}"></span></li>
        <li><strong>Position:</strong> <span
th:text="${employee.position}"></span></li>
        <li><strong>Department:</strong> <span
th:text="${employee.department.name}"></span></li>
    </ul>

    <form action="#" th:action="@{/employee/delete/{id}(id=${employee.id})}"
method="post">
        <button type="submit" class="btn btn-danger">Delete</button>
        <a href="/employees" class="btn btn-secondary">Cancel</a>
    </form>
</div>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

- View for **Employee** function

a. employeeList.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Employee List</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
<div class="container mt-4">
    <a href="/" class="btn btn-secondary mb-3">Back to Homepage</a>
    <h1 class="mb-4">Employee List</h1>

    <!-- Add Employee Button -->
    <a href="/employee/add" class="btn btn-primary mb-3">Add Employee</a>

    <!-- Employee Table -->
```

```
    <table class="table table-striped">
        <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Position</th>
            <th>Department</th>
            <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="employee : ${employees}">
            <td th:text="${employee.id}"></td>
            <td th:text="${employee.name}"></td>
            <td th:text="${employee.position}"></td>
            <td th:text="${employee.department.name}"></td>
            <td>
                <!-- View Details Button -->
                <a th:href="@{/employee/{id}(id=${employee.id})}" class="btn
btn-info btn-sm">View</a>
                <!-- Edit Button -->
                <a th:href="@{/employee/edit/{id}(id=${employee.id})}"
class="btn btn-warning btn-sm">Edit</a>
                <!-- Delete Button -->
                <a th:href="@{/employee/delete/{id}(id=${employee.id})}"
class="btn btn-danger btn-sm">Delete</a>
            </td>
        </tr>
        </tbody>
    </table>
</div>

<!-- Bootstrap JS and dependencies -->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.1/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

- employeeDetail.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Employee Detail</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <h1>Employee Detail</h1>
    <div class="card">
        <div class="card-body">
            <h5 class="card-title" th:text="${employee.name}"></h5>
            <p class="card-text"><strong>Position:</strong> <span
th:text="${employee.position}"></span></p>
            <p class="card-text"><strong>Department:</strong> <span
th:text="${employee.department.name}"></span></p>
            <a href="/employees" class="btn btn-secondary">Back to List</a>
```

```
            <a th:href="@{/employee/edit/{id}(id=${employee.id})}" class="btn
btn-warning">Edit</a>
            <a th:href="@{/employee/delete/{id}(id=${employee.id})}" class="btn
btn-danger">Delete</a>
        </div>
    </div>
</div>
</body>
</html>
```

- employee-add.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Employee</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <h1>Add New Employee</h1>
    <form th:action="@{/employee/add}" method="post">
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
required>
        </div>
        <div class="mb-3">
            <label for="position" class="form-label">Position</label>
            <input type="text" class="form-control" id="position"
name="position" required>
        </div>
        <div class="mb-3">
            <label for="department" class="form-label">Department</label>
            <select class="form-select" id="department" name="departmentId">
                <option th:each="department : ${departments}"
th:value="${department.id}" th:text="${department.name}"></option>
            </select>
        </div>
        <button type="submit" class="btn btn-primary">Add Employee</button>
        <a href="/employees" class="btn btn-secondary">Cancel</a>
    </form>
</div>
</body>
</html>
```

d. employee-update.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit Employee</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
</head>
<body>
<div class="container mt-5">
    <h1>Edit Employee</h1>
    <form th:action="@{/employee/update/{id}(id=${employee.id})}" method="post">
        <input type="hidden" name="id" th:value="${employee.id}"/>
        <div class="mb-3">
```

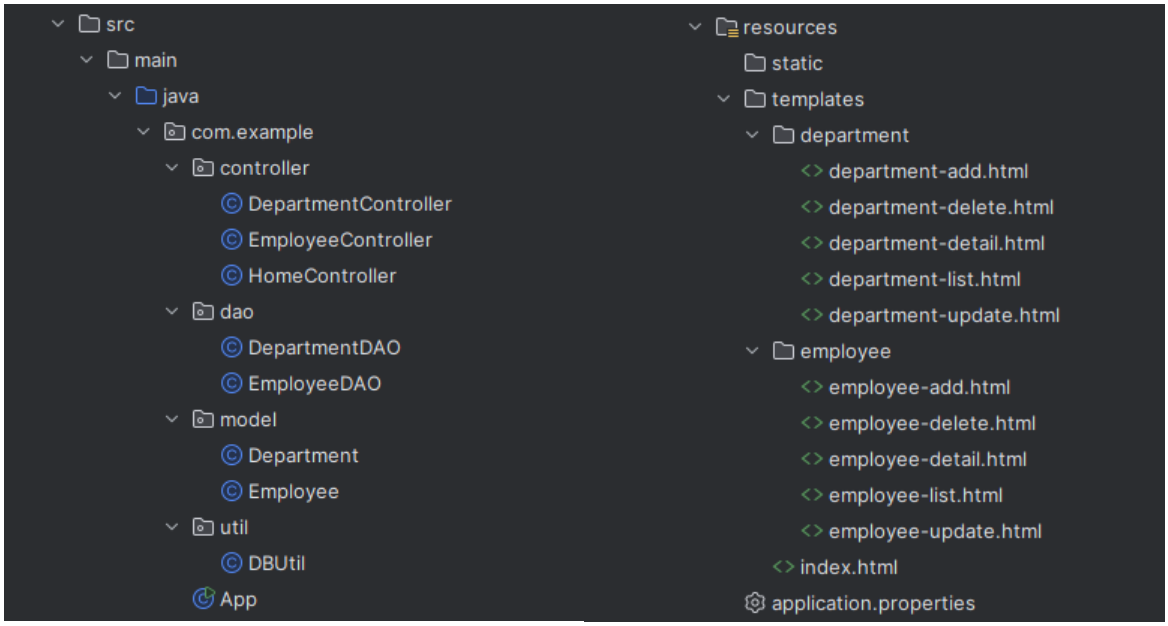
```
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
th:value="${employee.name}" required>
        </div>
        <div class="mb-3">
            <label for="position" class="form-label">Position</label>
            <input type="text" class="form-control" id="position"
name="position" th:value="${employee.position}" required>
        </div>
        <div class="mb-3">
            <label for="department" class="form-label">Department</label>
            <select class="form-select" id="department" name="departmentId"
required>
                <option th:each="department : ${departments}"
                        th:value="${department.id}"
                        th:text="${department.name}"
                        th:selected="${department.id ==
employee.department.id}"></option>
            </select>
        </div>
        <button type="submit" class="btn btn-primary">Update Employee</button>
        <a href="/employees" class="btn btn-secondary">Cancel</a>
    </form>
</div>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.1/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

- employee-delete.html:

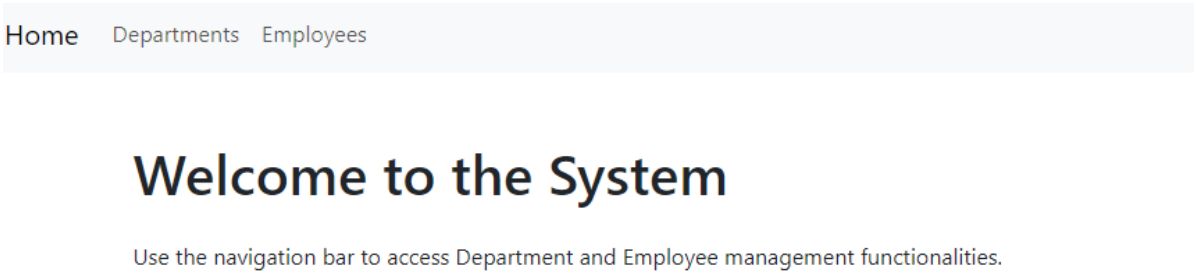
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Delete Employee</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-4">
    <h1 class="mb-4">Delete Employee</h1>
    <p>Are you sure you want to delete this employee?</p>
    <ul>
        <li><strong>Name:</strong> <span th:text="${employee.name}"></span></li>
        <li><strong>Position:</strong> <span
th:text="${employee.position}"></span></li>
        <li><strong>Department:</strong> <span
th:text="${employee.department.name}"></span></li>
    </ul>

    <form action="#" th:action="@{/employee/delete/{id}(id=${employee.id})}"
method="post">
        <button type="submit" class="btn btn-danger">Delete</button>
        <a href="/employees" class="btn btn-secondary">Cancel</a>
    </form>
</div>
```

```
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

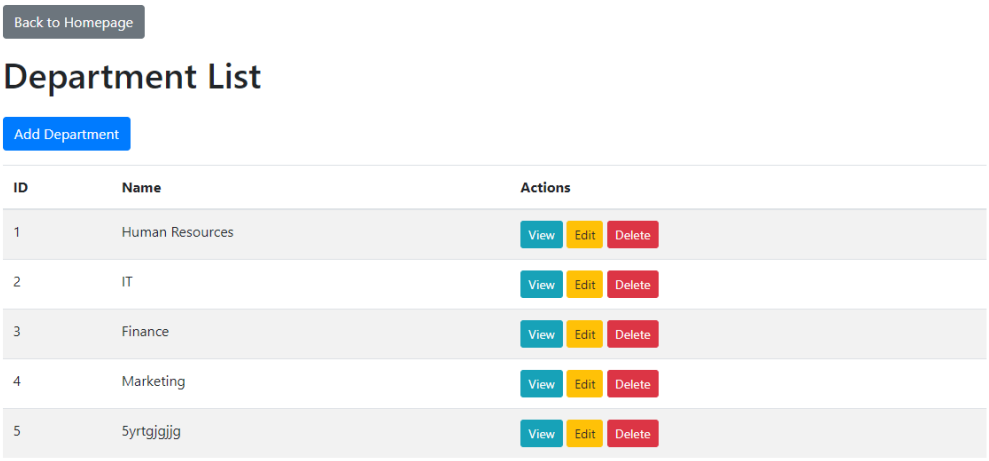
Here is the structure of the system:



Step 7: Run the application:



When you click to the Departments:



In this example:

- **Employee Class**: The Employee class represents an entity with attributes such as id, name, and position, along with a reference to the Department class. It encapsulates employee-related information and serves as the core data structure for employee management within the application.
- **Department Class**: The Department class represents an entity with attributes such as id and name. It encapsulates department-related information and serves as the core data structure for managing different departments within the application.
- **DBUtil Class**: The DBUtil class is responsible for setting up and managing the database connection. It reads configuration values from the application.properties file, such as the database URL, username, and password, and initializes a JdbcTemplate to interact with the database.
- **EmployeeDAO Class**: The EmployeeDAO class handles all data access operations related to the Employee entity. It provides methods to create, read, update, and delete employee records in the database. It uses the JdbcTemplate provided by DBUtil to perform SQL operations, ensuring efficient interaction with the employee-related data.
- **DepartmentDAO Class**: The DepartmentDAO class manages data access operations for the Department entity. It offers methods to create, read, update, and delete department records in the database, using the JdbcTemplate provided by DBUtil. This class facilitates the management of department data within the system.
- **EmployeeController Class**: The EmployeeController class manages HTTP requests related to employees. It uses the EmployeeDAO to retrieve and manipulate data, handling user interactions like listing employees, viewing details, adding, updating, and deleting employee records. It acts as the intermediary between the user interface and the employee-related business logic.
- **DepartmentController Class**: The DepartmentController class manages HTTP requests related to departments. It leverages the DepartmentDAO to retrieve and manipulate department data, facilitating operations like listing departments, viewing details, adding, updating, and deleting department records. It serves as the bridge between the user interface and the department-related business logic.

- **Spring Dependency Injection**: The use of @Autowired ensures that dependencies such as DBUtil, EmployeeDAO, and DepartmentDAO are automatically injected into the respective classes that require them. This simplifies the management of dependencies and promotes loose coupling between components.
- **Spring MVC**: The EmployeeController and DepartmentController classes demonstrate how Spring MVC handles HTTP requests and responses. They map URLs to specific methods that perform business logic and return views, enabling a seamless user experience for managing both employees and departments within the application.

This example provides a clear understanding of how Spring MVC organizes the interaction between controllers, data access layers, and models to create a modular and maintainable web application.

----oOo-----

THE END