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_Opt1
TYPE:	MEDIUM
LOC:	200
DURATION:	120 MINUTES

Java Spring Framework Introduction

Objectives:

- Understand how to use DAO (Data Access Object) pattern with Spring MVC.
- Learn to configure and use Spring MVC for managing entities with database interactions.

Lab Specifications:

- In a University Management System, Employee entity will use DAO classes to interact with a PostgreSQL database. Students will learn to implement CRUD operations using Spring MVC and PostgreSQL.

Problem Description:

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

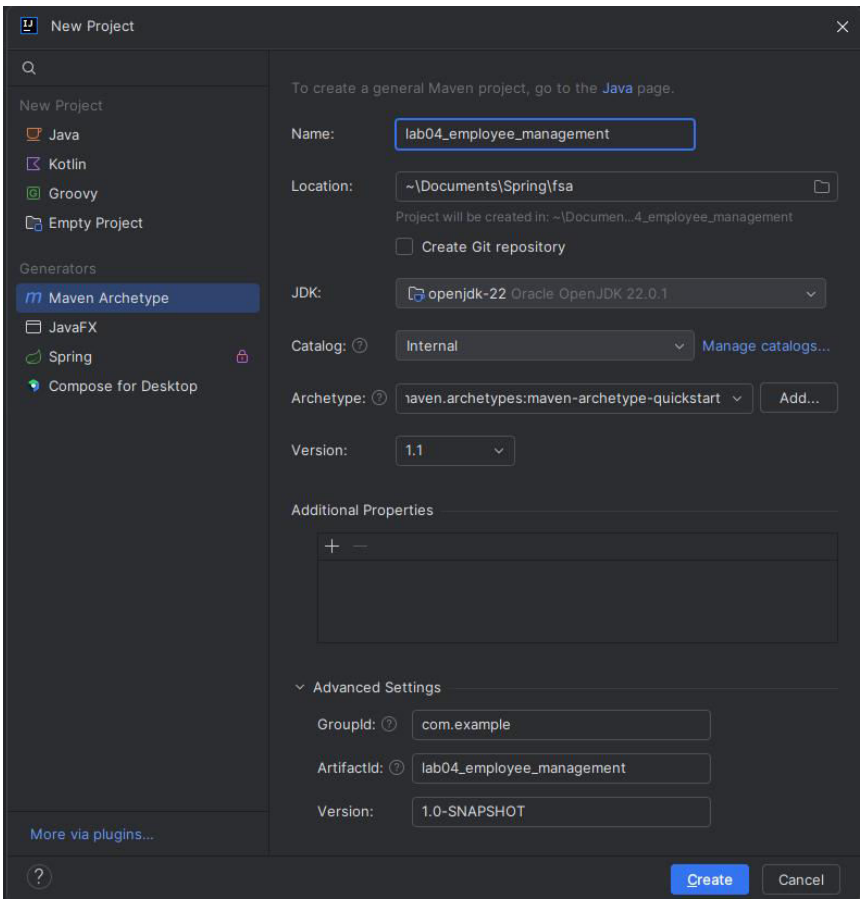
Prerequisites:

- 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_managment**
- Set the groupId to com.example and artifactId to **lab04_employee_managment**
- Click **Create**.



Step 2: Add dependencies and configuration into pom.xml file: Add the Spring Core dependency to your pom.xml file.

```
<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 TABLE IF NOT EXISTS employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    position VARCHAR(255) NOT NULL
);
```

```
INSERT INTO employees (name, position) VALUES ('Alice Johnson', 'Software Engineer');
INSERT INTO employees (name, position) VALUES ('Bob Smith', 'Project Manager');
INSERT INTO employees (name, position) VALUES ('Charlie Brown', 'QA Engineer');
INSERT INTO employees (name, position) VALUES ('Diana Prince', 'Business Analyst');
INSERT INTO employees (name, position) VALUES ('Eve White', 'UX Designer');
```

Step 5: Create entity classes:

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

```
package com.example.model;

public class Employee {
    private Long id;
    private String name;
    private String position;

    public Employee() {
    }

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

    // 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 getPosition() {
        return position;
    }

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

Step 6: Create EmployeeDAO class in dao package.

This class will handle the database operations using JdbcTemplate.

```
package com.example.dao;

import com.example.model.Employee;
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 EmployeeDAO {

    @Autowired
    private DBUtil dbUtil;

    private JdbcTemplate jdbcTemplate;

    @Autowired
    public void initialize() {
        this.jdbcTemplate = dbUtil.getJdbcTemplate();
    }

    public List<Employee> getAllEmployees() {
        return jdbcTemplate.query("SELECT * FROM employees", new
EmployeeRowMapper());
    }

    public Employee getEmployeeById(Long id) {
        return jdbcTemplate.queryForObject("SELECT * FROM employees WHERE id =
?", new EmployeeRowMapper(), id);
    }

    public void addEmployee(Employee employee) {
        jdbcTemplate.update("INSERT INTO employees (name, position) VALUES (?,
?)",
                employee.getName(), employee.getPosition());
    }

    public void updateEmployee(Employee employee) {
        jdbcTemplate.update("UPDATE employees SET name = ?, position = ? WHERE
id = ?",
                employee.getName(), employee.getPosition(), employee.getId());
    }

    public void deleteEmployee(Long id) {
```

```
        jdbcTemplate.update("DELETE FROM employees WHERE id = ?", id);
    }

    public void createTable() {
        jdbcTemplate.execute("CREATE TABLE employees (" +
                "id BIGSERIAL PRIMARY KEY, " +
                "name VARCHAR(255), " +
                "position VARCHAR(255))");
    }

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

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

Step7: 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 8: Create Views

1. **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>
    <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">Employee List</h1>

    <!-- Employee List Table -->
    <table class="table table-striped table-bordered">
        <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Position</th>
            <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="employee : ${employees}">
            <td th:text="${employee.id}"></td>
            <td>
                <a th:href="@{/employee/{id}(id=${employee.id})}"
th:text="${employee.name}"></a>
            </td>
            <td th:text="${employee.position}"></td>
            <td>
                <a th:href="@{/employee/edit/{id}(id=${employee.id})}"
class="btn btn-warning btn-sm">Edit</a>
                <a th:href="@{/employee/delete/{id}(id=${employee.id})}"
class="btn btn-danger btn-sm" onclick="return confirm('Are you
sure?')">Delete</a>
            </td>
        </tr>
        </tbody>
    </table>

    <!-- Add Employee Form -->
    <h2>Add New Employee</h2>
    <form action="/employee/add" method="post" class="form">
        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="name">Name</label>
                <input type="text" id="name" name="name" class="form-control"
placeholder="Name" required />
            </div>
        </div>
        <div class="form-row">
            <div class="form-group col-md-6">
                <label for="position">Position</label>
                <input type="text" id="position" name="position" class="form-
control" placeholder="Position" required />
            </div>
        </div>
        <button type="submit" class="btn btn-primary">Add Employee</button>
    </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>
```

2. **employeeDetail**.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 Detail</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">Employee Detail</h1>
    <div class="card">
        <div class="card-body">
            <h5 class="card-title">Details</h5>
            <p class="card-text"><strong>Name:</strong> <span
th:text="${employee.name}"></span></p>
            <p class="card-text"><strong>Position:</strong> <span
th:text="${employee.position}"></span></p>
        </div>
    </div>
    <a href="/employees" class="btn btn-secondary mt-3">Back to Employee
List</a>
</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>
```

3. **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>Update 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">Update Employee</h1>

    <!-- Update Employee Form -->
```

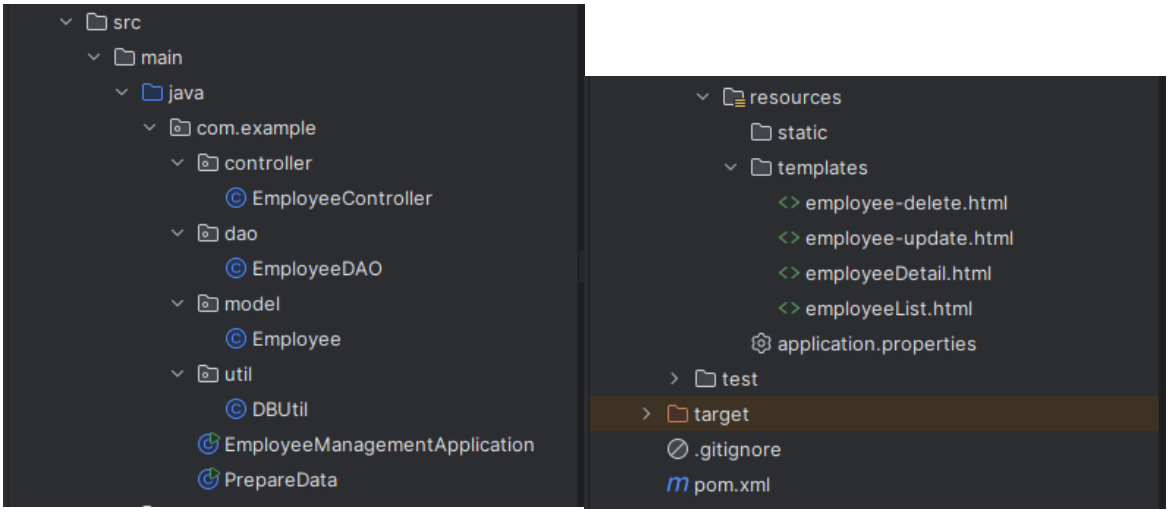
```
    <form action="#" th:action="@{/employee/update/{id}(id=${employee.id})}"
method="post">
        <div class="form-group">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" class="form-control"
th:value="${employee.name}" required />
        </div>
        <div class="form-group">
            <label for="position">Position:</label>
            <input type="text" id="position" name="position" class="form-
control" th:value="${employee.position}" required />
        </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@2.9.3/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

4. **employee-delete**.html:

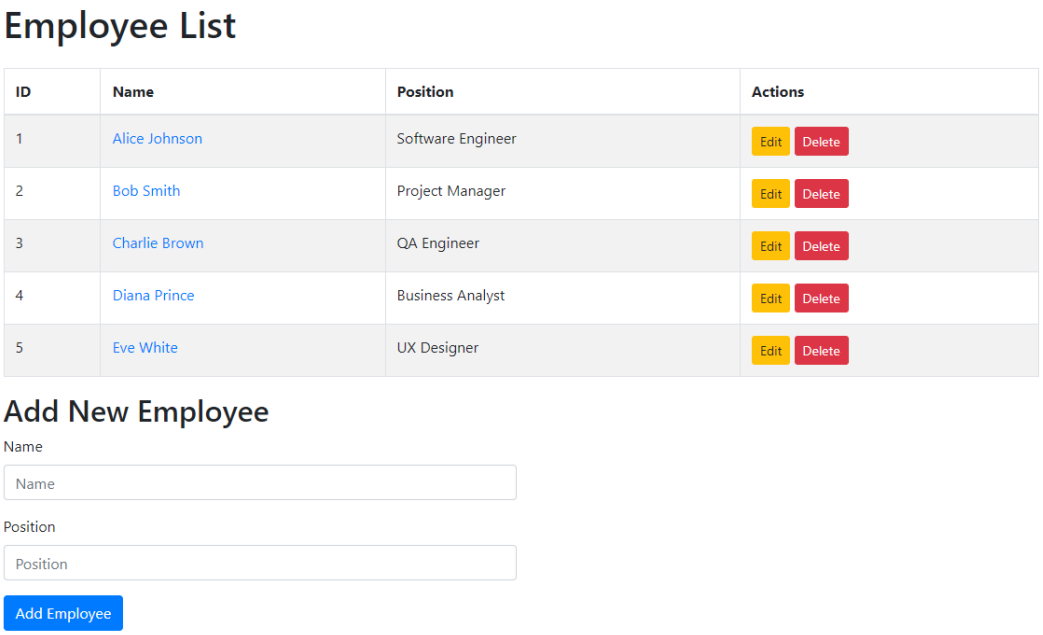
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Delete Employee</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
</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>
    </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 program:



Step 7: Run the application:



In this example:

- **Employee Class**: The Employee class represents an entity with attributes such as id, name, and position. It serves as the core data structure for the application, encapsulating employee-related information.
- **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.
- **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.
- **Spring Dependency Injection**: The use of @Autowired ensures that dependencies such as DBUtil and EmployeeDAO are automatically injected into the classes that require them. This simplifies the management of dependencies and promotes loose coupling between components.
- **Spring MVC**: The EmployeeController class demonstrates how Spring MVC handles HTTP requests and responses, mapping URLs to specific methods that perform business logic and return views.

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