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_05_Opt1
TYPE:	LONG
LOC:	200
DURATION:	180 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:

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

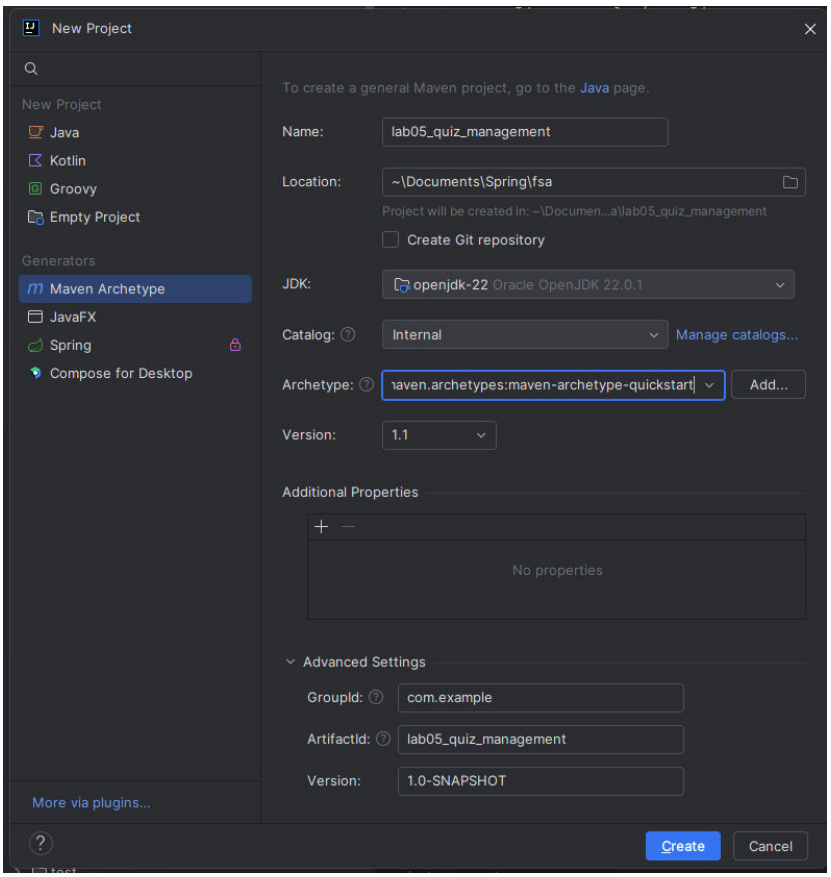
Prerequisites:

- 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 lab05_quiz_managment
- Set the groupId to com.example and artifactId to lab05_ quiz_managment
- Click **Create**.



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

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

And put these inside **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/quiz_management
spring.datasource.username=postgres
spring.datasource.password=1234567890
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.properties.hibernate.default_schema=public

spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.mode=HTML
```

Step 4: Prepare Data:

```
CREATE TABLE Role (
    role_id SERIAL PRIMARY KEY,
    role_name VARCHAR(255) NOT NULL UNIQUE
);
CREATE TABLE Users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    full_name VARCHAR(255),
    role_id INT,
    FOREIGN KEY (role_id) REFERENCES Role(role_id)
);
```

```
-- Role table
INSERT INTO Role (role_name) VALUES ('Admin');
INSERT INTO Role (role_name) VALUES ('Teacher');
INSERT INTO Role (role_name) VALUES ('Student');
-- Users table
INSERT INTO Users (username, password, email, full_name, role_id)
VALUES ('admin', 'adminpass', 'admin@example.com', 'Admin User', 1);


INSERT INTO Users (username, password, email, full_name, role_id)
```

```
VALUES ('teacher1', 'teacherpass', 'teacher1@example.com', 'Teacher One', 2);


INSERT INTO Users (username, password, email, full_name, role_id)
VALUES ('student1', 'studentpass', 'student1@example.com', 'Student One', 3);
```

Step 5: Create entity classes:

1. Create **Role** class in **model** package:

```
package com.example.model;

import org.springframework.stereotype.Component;

@Component
public class Role {
    private Integer roleId;
    private String name;

    // Getters and setters
    public Integer getRoleId() {
        return roleId;
    }

    public void setRoleId(Integer roleId) {
        this.roleId = roleId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

2. Create **Role** class in **model** package:

```
package com.example.model;

import org.springframework.stereotype.Component;

@Component
public class User {
    private Integer userId;
    private String username;
    private String password;
    private String email;
    private String fullName;
    private Role role; // assuming role id as Integer to map it with database

    // Getters and Setters
    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }
```

```
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public Role getRole() {
        return role;
    }

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

Step 6: Create DAO classes in dao package.

These classes will handle the database operations using JdbcTemplate.

1. Create RoleDAO interface:

```
package com.example.dao;

import java.util.List;
import com.example.model.Role;

public interface RoleDAO {
    void save(Role role);
    void update(Role role);
    boolean delete(Integer roleId);
    Role findById(Integer roleId);
    List<Role> findAll();
}
```

2. Create **RoleDAOImpl** class:

```java
package com.example.dao;

import com.example.model.Role;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

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

@Component
public class RoleDAOImpl implements RoleDAO {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public void save(Role role) {
        String sql = "INSERT INTO Role (role_name) VALUES (?)";
        jdbcTemplate.update(sql, role.getName());
    }

    @Override
    public void update(Role role) {
        String sql = "UPDATE Role SET role_name = ? WHERE role_id = ?";
        jdbcTemplate.update(sql, role.getName(), role.getRoleId());
    }

    @Override
    public boolean delete(Integer roleId) {
        String sql = "DELETE FROM roles WHERE role_id = ?";
        int rowsAffected = jdbcTemplate.update(sql, roleId);
        return rowsAffected > 0;
    }

    @Override
    public Role findById(Integer roleId) {
        String sql = "SELECT * FROM Role WHERE role_id = ?";
        return jdbcTemplate.queryForObject(sql, this::mapRowToRole, roleId);
    }

    @Override
    public List<Role> findAll() {
        String sql = "SELECT * FROM Role";
        return jdbcTemplate.query(sql, this::mapRowToRole);
    }

    private Role mapRowToRole(ResultSet rs, int rowNum) throws SQLException {
        Role role = new Role();
        role.setRoleId(rs.getInt("role_id"));
        role.setName(rs.getString("role_name"));
        return role;
    }
}
```

3. Create RoleDAO interface:

```java
package com.example.dao;

import com.example.model.User;

import java.util.List;
```

```
public interface UserDAO {
    void createUser(User user);
    User getUserById(Integer userId);
    List<User> getAllUsers();
    void updateUser(User user);
    boolean deleteUser(Integer userId);
}
```

4. Create **UserDAOImpl** class:

```
package com.example.dao;

import com.example.model.Role;
import com.example.model.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

import javax.sql.DataSource;
import java.util.List;

@Component
public class UserDAOImpl implements UserDAO {
    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public UserDAOImpl(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    @Override
    public void createUser(User user) {
        String sql = "INSERT INTO Users (username, password, email, full_name,
role_id) VALUES (?, ?, ?, ?, ?)";
        jdbcTemplate.update(sql, user.getUsername(), user.getPassword(),
user.getEmail(), user.getFullName(), user.getRole().getRoleId());
    }


    @Override
    public User getUserById(Integer userId) {
        String sql = "SELECT * FROM Users WHERE user_id = ?";
        return jdbcTemplate.queryForObject(sql, new Object[]{userId}, (rs,
rowNum) -> {
            Role role = new Role();
            role.setRoleId(rs.getInt("role_id"));

            User user = new User();
            user.setUserId(rs.getInt("user_id"));
            user.setUsername(rs.getString("username"));
            user.setPassword(rs.getString("password"));
            user.setEmail(rs.getString("email"));
            user.setFullName(rs.getString("full_name"));
            user.setRole(role);

            return user;
        });
    }

    @Override
    public List<User> getAllUsers() {
        String sql = "SELECT u.user_id, u.username, u.password, u.email,
u.full_name, " +
                "r.role_id, r.role_name " +
```

```
                "FROM Users u " +
                "LEFT JOIN Role r ON u.role_id = r.role_id";

        return jdbcTemplate.query(sql, (rs, rowNum) -> {
            User user = new User();
            Role role = new Role();

            // Populate Role
            role.setRoleId(rs.getInt("role_id"));
            role.setName(rs.getString("role_name"));

            // Populate User
            user.setUserId(rs.getInt("user_id"));
            user.setUsername(rs.getString("username"));
            user.setPassword(rs.getString("password"));
            user.setEmail(rs.getString("email"));
            user.setFullName(rs.getString("full_name"));
            user.setRole(role);  // Assign the fully populated Role object

            return user;
        });
    }

    @Override
    public void updateUser(User user) {
        String sql = "UPDATE Users SET username = ?, password = ?, email = ?,
full_name = ?, role_id = ? WHERE user_id = ?";
        jdbcTemplate.update(sql, user.getUsername(), user.getPassword(),
user.getEmail(), user.getFullName(), user.getRole().getRoleId(),
user.getUserId());
    }
    @Override
    public boolean deleteUser(Integer userId) {
        String sql = "DELETE FROM Users WHERE user_id = ?";
        int rowsAffected = jdbcTemplate.update(sql, userId);

        return rowsAffected > 0;
    }
}
```

Step7: Create Views

1. For **Role** function:

a. roleList.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Roles</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">
    <a href="/" class="btn btn-secondary mb-3">Back to Homepage</a>
    <h1>Roles</h1>
    <a th:href="@{/role/create}" class="btn btn-primary mb-3">Create New
Role</a>
    <table class="table">
        <thead>
        <tr>
```

```
            <th>ID</th>
            <th>Name</th>
            <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="role : ${roles}">
            <td th:text="${role.roleId}"></td>
            <td th:text="${role.name}"></td>
            <td>
                <a th:href="@{/role/edit/{id}(id=${role.roleId})}" class="btn
btn-warning">Edit</a>
                <a th:href="@{/role/delete/{id}(id=${role.roleId})}" class="btn
btn-danger">Delete</a>
                <a th:href="@{/role/{id}(id=${role.roleId})}" class="btn btn-
info">View</a>
            </td>
        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

2. roleDetail.html

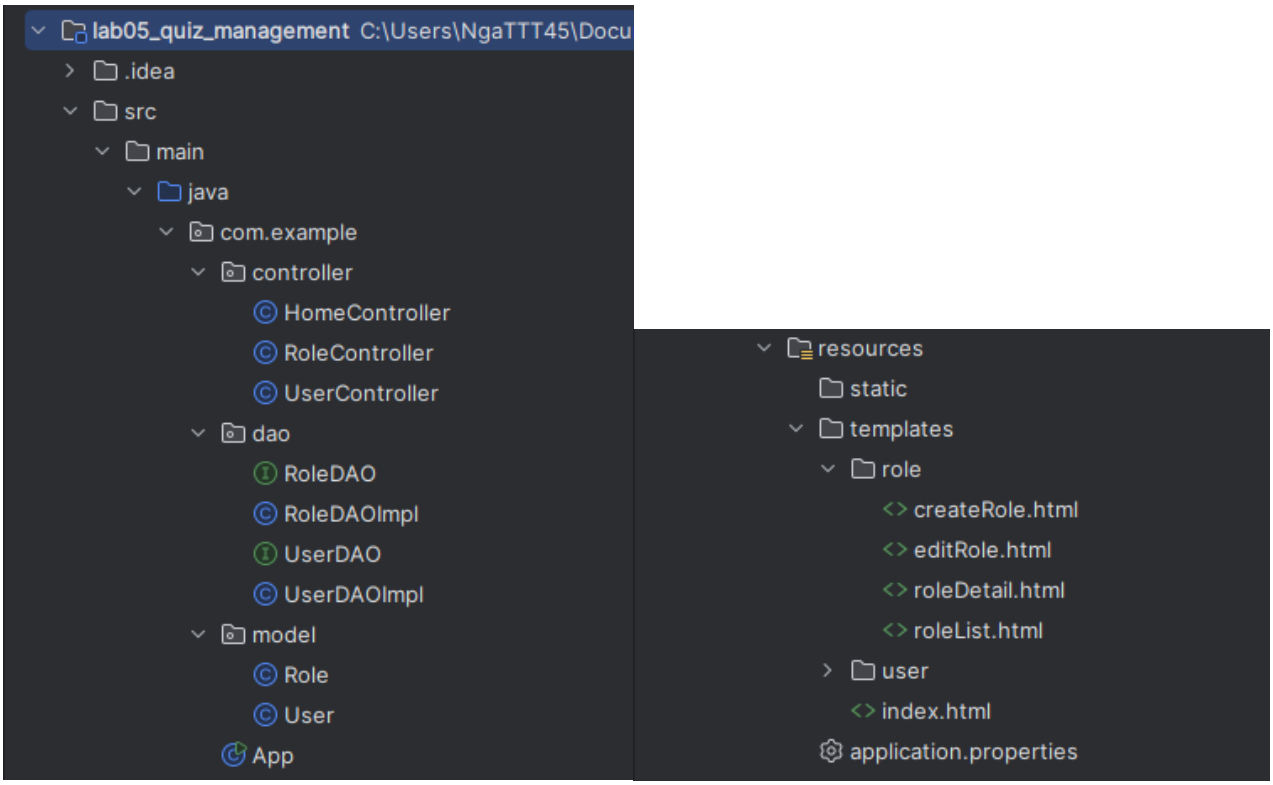
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Role Detail</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <h1>Role Detail</h1>
    <div class="card">
        <div class="card-body">
            <h5 class="card-title" th:text="${role.name}"></h5>
            <p class="card-text"><strong>ID:</strong> <span
th:text="${role.roleId}"></span></p>
            <a th:href="@{/roles}" class="btn btn-secondary">Back to List</a>
            <a th:href="@{/role/edit/{id}(id=${role.roleId})}" class="btn btn-
warning">Edit</a>
            <a th:href="@{/role/delete/{id}(id=${role.roleId})}" class="btn btn-
danger">Delete</a>
        </div>
    </div>
</div>
</body>
</html>
```

3. editRole.html:

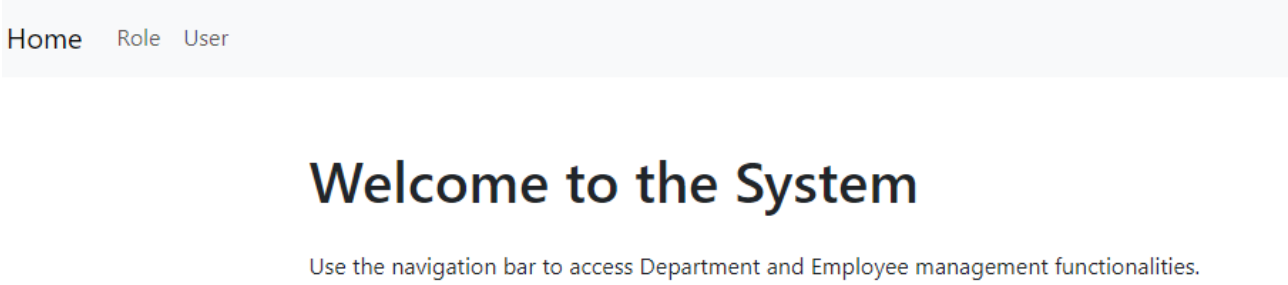
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Edit Role</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
```

```
    <h1>Edit Role</h1>
    <form th:action="@{/role/update/{id}(id=${role.roleId})}" method="post">
        <div class="mb-3">
            <label for="name" class="form-label">Name</label>
            <input type="text" class="form-control" id="name" name="name"
th:value="${role.name}" required>
        </div>
        <button type="submit" class="btn btn-primary">Update</button>
        <a th:href="@{/roles}" class="btn btn-secondary">Cancel</a>
    </form>
</div>
</body>
</html>
```

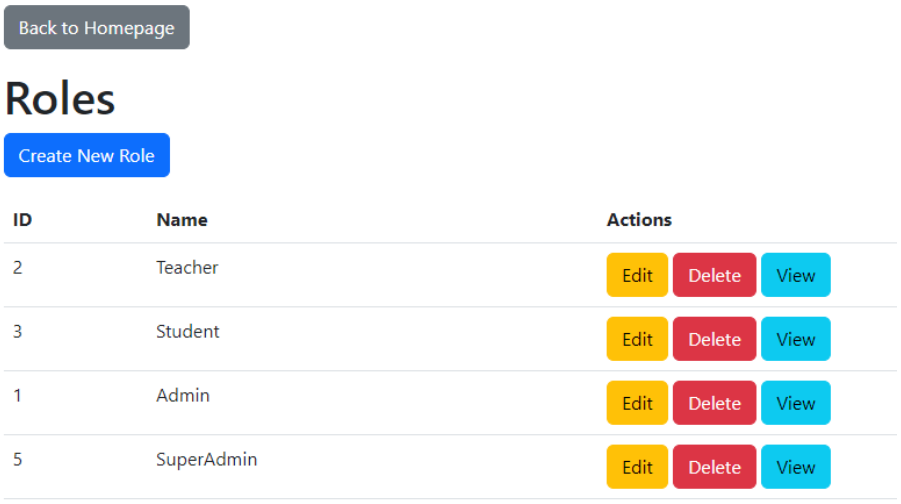
Here is the structure of the program:



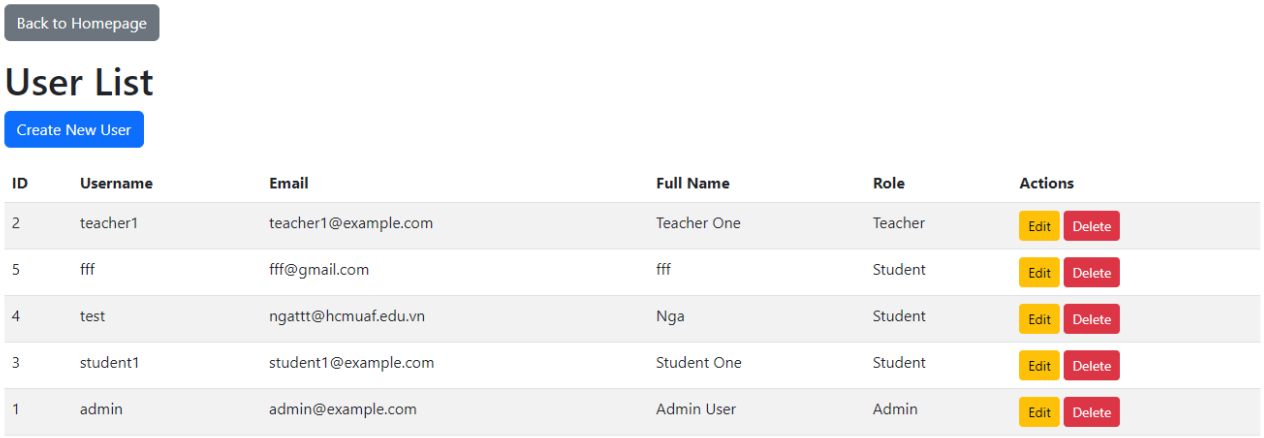
Step 7: Run the application:



When you click [Role]:



When you click [User]:



By the end of this exam, you should be able to:

- Develop a basic role and user management system using Spring MVC.
- Manage user sessions and attributes effectively.
- Use SpEL to create dynamic, context-aware views.
- Implement redirects and flash messages to enhance user experience.

----oOo-----

THE END