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_08_Opt1
TYPE:	MEDIUM
LOC:	200
DURATION:	120 MINUTES

Java Spring Framework Introduction

Objectives:

In this lab, you will learn how to manage training programs and associate them with subjects using Spring Boot. You will set up the necessary entities, repositories, services, and controllers to interact with the TrainingProgram and TrainingProgramSubjects tables.

Lab Specifications:

Trainees are required to:

- Create `TrainingProgram, TrainingProgramSubjects` entities.
- Implement CRUD operations using `TrainingProgramRepository`.
- Create Thymeleaf views to manage training program (create, read, update, and delete)..

Problem Description:

Trainees are required to:

- Implement Training Program functionality and a function to add subject to a training program.
- Use Thymeleaf for the user interface (UI).

Prerequisites:

- Completed **JSFW_Lab_07_Opt2**.

Guidelines:

Step 1: Update pom.xml:

Add Hibernate Validator dependency to pom.xml:

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>8.0.1.Final</version>
</dependency>
```

Step 2: Create TrainingProgram Entity

Define TrainingProgram Class: In com.example.model, create TrainingProgram class:

```
package com.example.model;

import javax.persistence.*;
```

```
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "TrainingProgram")
public class TrainingProgram {

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

    @Column(nullable = false, unique = true)
    private String programName;

    @Column(nullable = false, unique = true)
    private String programCode;

    private String description;

    // Many-to-Many relationship with Subject
    @ManyToMany
    @JoinTable(
            name = "program_subject", // Join table name
            joinColumns = @JoinColumn(name = "program id"), // Column for
TrainingProgram
            inverseJoinColumns = @JoinColumn(name = "subject_id") // Column for
Subject
    )
    private Set<Subject> subjects = new HashSet<>();

    // Getters and Setters
    public Long getProgramId() {
        return programId;
    }

    public void setProgramId(Long programId) {
        this.programId = programId;
    }

    public String getProgramName() {
        return programName;
    }

    public void setProgramName(String programName) {
        this.programName = programName;
    }

    public String getProgramCode() {
        return programCode;
    }

    public void setProgramCode(String programCode) {
        this.programCode = programCode;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Set<Subject> getSubjects() {
```

```
        return subjects;
    }

    public void setSubjects(Set<Subject> subjects) {
        this.subjects = subjects;
    }

    // Method to add a subject to the program
    public void addSubject(Subject subject) {
        this.subjects.add(subject);
    }
}
```

Step 3: Create TrainingProgramRepository:

In com.example.repository, create the TrainingProgramRepository interface:

```
package com.example.repository;

import com.example.model.TrainingProgram;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface TrainingProgramRepository extends
JpaRepository<TrainingProgram, Long> {
}
```

Step 4: Create TrainingProgramService:

In com.example.service, create a TrainingProgramService.java:

```
package com.example.service;

import com.example.model.Subject;
import com.example.model.TrainingProgram;
import com.example.repository.SubjectRepository;
import com.example.repository.TrainingProgramRepository;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
import java.util.Set;

@Service
public class TrainingProgramService {

    private final TrainingProgramRepository trainingProgramRepository;
    private final SubjectRepository subjectRepository;

    public TrainingProgramService(TrainingProgramRepository
trainingProgramRepository, SubjectRepository subjectRepository) {
        this.trainingProgramRepository = trainingProgramRepository;
        this.subjectRepository = subjectRepository;
    }

    public TrainingProgram getProgramById(Long programId) {
        // Find the program by ID and return it (throw an exception if not
found)
        return trainingProgramRepository.findById(programId)
                .orElseThrow(() -> new RuntimeException("Training Program not
found with ID: " + programId));
```

```
    }

    public List<TrainingProgram> findAll() {
        return trainingProgramRepository.findAll();
    }

    public Optional<TrainingProgram> findById(Long id) {
        return trainingProgramRepository.findById(id);
    }

    public TrainingProgram save(TrainingProgram trainingProgram) {
        return trainingProgramRepository.save(trainingProgram);
    }

    public void deleteById(Long id) {
        trainingProgramRepository.deleteById(id);
    }

    /**
     * Add selected subjects to the training program.
     *
     * @param programId   The ID of the TrainingProgram to update
     * @param subjectIds  A set of subject IDs to add to the TrainingProgram
     */
    public void addSubjectsToProgram(Long programId, Set<Long> subjectIds) {
        // Fetch the training program by ID
        Optional<TrainingProgram> programOpt =
trainingProgramRepository.findById(programId);

        if (programOpt.isPresent()) {
            TrainingProgram program = programOpt.get();

            // Fetch each subject by its ID and add it to the training program's
subjects
            for (Long subjectId : subjectIds) {
                Optional<Subject> subjectOpt =
subjectRepository.findById(subjectId);
                subjectOpt.ifPresent(program::addSubject);
            }

            // Save the updated program with the added subjects
            trainingProgramRepository.save(program);
        } else {
            throw new RuntimeException("Training Program not found");
        }
    }
}
```

Step 5: Create TrainingProgramController

In com.example.controller, create a TrainingProgramController.java:

```
package com.example.controller;

import com.example.model.Subject;
import com.example.model.TrainingProgram;
import com.example.service.SubjectService;
import com.example.service.TrainingProgramService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
```

```
import javax.validation.constraints.NotEmpty;
import java.util.List;
import java.util.Set;

@Controller
@RequestMapping("/trainingPrograms")
public class TrainingProgramController {

    @Autowired
    private final TrainingProgramService trainingProgramService;
    private final SubjectService subjectService;

    public TrainingProgramController(TrainingProgramService
trainingProgramService, SubjectService subjectService) {
        this.trainingProgramService = trainingProgramService;
        this.subjectService = subjectService;
    }

    @GetMapping("/{programId}")
    public String showProgramDetails(@PathVariable Long programId, Model model)
{
        TrainingProgram program =
trainingProgramService.getProgramById(programId);
        Set<Subject> subjects = program.getSubjects();
        model.addAttribute("trainingProgram", program);
        model.addAttribute("subjects", subjects);
        return "trainingPrograms/details";
    }

    @GetMapping("/{programId}/add-subjects")
    public String showAddSubjectsForm(@PathVariable Long programId, Model model)
{
        TrainingProgram program =
trainingProgramService.getProgramById(programId);
        List<Subject> subjects = subjectService.getAllSubjects();
        Set<Subject> programSubjects = program.getSubjects();
        model.addAttribute("programId", programId);
        model.addAttribute("subjects", subjects);
        model.addAttribute("programSubjects", programSubjects);
        return "trainingPrograms/add_subjects";
    }

    @PostMapping("/{programId}/add-subjects")
    public String addSubjectsToProgram(@PathVariable Long programId,
@RequestParam @NotEmpty Set<Long> subjectIds, Model model) {
        // Validate the input
        if (subjectIds == null || subjectIds.isEmpty()) {
            model.addAttribute("error", "At least one subject must be
selected.");
            return showAddSubjectsForm(programId, model);
        }

        // Add selected subjects to the program
        trainingProgramService.addSubjectsToProgram(programId, subjectIds);
        return "redirect:/trainingPrograms/" + programId;
    }

    @GetMapping
    public String listTrainingPrograms(Model model) {
        model.addAttribute("trainingPrograms",
trainingProgramService.findAll());
        return "trainingPrograms/list";
    }
```

```
    @GetMapping("/new")
    public String createTrainingProgramForm(Model model) {
        model.addAttribute("trainingProgram", new TrainingProgram());
        return "trainingPrograms/form";
    }

    @PostMapping
    public String saveTrainingProgram(@ModelAttribute TrainingProgram
trainingProgram) {
        trainingProgramService.save(trainingProgram);
        return "redirect:/trainingPrograms";
    }

    @GetMapping("/edit/{id}")
    public String editTrainingProgramForm(@PathVariable Long id, Model model) {
        TrainingProgram trainingProgram =
trainingProgramService.findById(id).orElse(null);
        if (trainingProgram != null) {
            model.addAttribute("trainingProgram", trainingProgram);
            return "trainingPrograms/form";
        }
        return "redirect:/trainingPrograms";
    }

    @PostMapping("/update/{id}")
    public String updateTrainingProgram(@PathVariable Long id, @ModelAttribute
TrainingProgram trainingProgram) {
        trainingProgram.setProgramId(id);
        trainingProgramService.save(trainingProgram);
        return "redirect:/trainingPrograms";
    }

    @GetMapping("/delete/{id}")
    public String deleteTrainingProgram(@PathVariable Long id) {
        trainingProgramService.deleteById(id);
        return "redirect:/trainingPrograms";
    }
}
```

Step 6: Create Thymeleaf Templates

Create the following templates in src/main/resources/templates/trainingPrograms:

a. list.html - List of Training Program:

```
<!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>Training Programs</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <!-- Back to Dashboard button -->
    <a href="/dashboard" class="btn btn-secondary mb-3">Back to Dashboard</a>
    <a href="/trainingPrograms/new" class="btn btn-primary mb-3">Add New
Training Program</a>
```

```
    <!-- Check if there are any subjects -->
    <div th:if="${trainingPrograms.isEmpty()}">
        <p class="text-center text-muted">Don't have any data yet.</p>
    </div>

    <!-- Display table if subjects exist -->
    <table class="table table-striped" th:if="${!trainingPrograms.isEmpty()}">
        <thead>
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Code</th>
            <th>Description</th>
            <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="trainingProgram : ${trainingPrograms}">
            <td th:text="${trainingProgram.programId}">1</td>
            <td th:text="${trainingProgram.programName}">Sample Program</td>
            <td th:text="${trainingProgram.programCode}">PROG101</td>
            <td th:text="${trainingProgram.description}">Description here</td>
            <td>
                <a th:href="@{'/trainingPrograms/' +
${trainingProgram.programId}}" class="btn btn-info">View Details</a>
                <a th:href="@{'/trainingPrograms/edit/' +
${trainingProgram.programId}}" class="btn btn-warning">Edit</a>
                <a th:href="@{'/trainingPrograms/delete/' +
${trainingProgram.programId}}" class="btn btn-danger" onclick="return
confirm('Are you sure?');">Delete</a>
                <!-- Add Subjects Button -->
                <a th:href="@{'/trainingPrograms/' +
${trainingProgram.programId} + '/add-subjects'}" class="btn btn-info">Add
Subjects</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@2.0.7/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. **form.html:** Training Program Form (Create/Update)

```
<!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>Training Program Form</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
```

```
rel="stylesheet">
</head>
<body>
<div class="container mt-5">
    <h2 class="text-center">Training Program Form</h2>
    <form th:action="@{/trainingPrograms}" th:object="${trainingProgram}"
method="post">
        <div class="form-group">
            <label for="programName">Program Name</label>
            <input type="text" class="form-control" id="programName"
th:field="*{programName}" required>
        </div>
        <div class="form-group">
            <label for="programCode">Program Code</label>
            <input type="text" class="form-control" id="programCode"
th:field="*{programCode}" required>
        </div>
        <div class="form-group">
            <label for="description">Description</label>
            <textarea class="form-control" id="description"
th:field="*{description}"></textarea>
        </div>
        <button type="submit" class="btn btn-primary">Save</button>
    </form>
</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@2.0.7/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>
</body>
</html>
```

c. **details.html**: Details of a training program

```
<!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>Training Program Details</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>

<div class="container mt-5">
    <!-- Back to List Button -->
    <a href="/trainingPrograms" class="btn btn-secondary mb-3">Back to List</a>

    <!-- Training Program Details -->
    <div class="card mb-4">
        <div class="card-header">
            <h3 th:text="${trainingProgram.programName}">Training Program
Name</h3>
        </div>
        <div class="card-body">
```

```
            <h5 class="card-title">Program Code: <span
th:text="${trainingProgram.programCode}">PROG101</span></h5>
            <p class="card-text">Description: <span
th:text="${trainingProgram.description}">This is the program
description.</span></p>
        </div>
    </div>

    <!-- Subjects Associated with this Program -->
    <div class="card">
        <div class="card-header">
            <h4>Subjects in this Program</h4>
        </div>
        <ul class="list-group list-group-flush">
            <!-- Loop through the subjects -->
            <li class="list-group-item" th:each="subject : ${subjects}">
                <span th:text="${subject.name}">Subject Name</span>
            </li>
            <!-- If no subjects, show a message -->
            <li class="list-group-item text-center text-muted"
th:if="${subjects.isEmpty()}">
                No subjects have been added to this program yet.
            </li>
        </ul>
    </div>
</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@2.0.7/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>

</body>
</html>
```

d. **add_subjects.html**: add subjects to a program

```
<!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>Add Subjects to Program</title>
    <link
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>

<div class="container mt-5">
    <h1 class="mb-4">Add Subjects to Program</h1>

    <form th:action="@{'/trainingPrograms/' + ${programId} + '/add-subjects'}"
method="post">
        <div class="form-group">
            <h5>Select Subjects:</h5>
            <ul class="list-group">
                <!-- Loop through subjects and create a checkbox for each -->
                <li class="list-group-item" th:each="subject : ${subjects}">
                    <div class="form-check">
```

```
                        <input type="checkbox" class="form-check-input"
th:value="${subject.id}" th:name="subjectIds"
                               th:checked="${programSubjects.contains(subject)}"
/>
                        <label class="form-check-label"
th:text="${subject.name}"></label>
                    </div>
                </li>
            </ul>
        </div>
        <button type="submit" class="btn btn-primary">Add Subjects</button>
        <a href="/trainingPrograms" class="btn btn-secondary ml-2">Cancel</a>
    </form>
</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@2.0.7/dist/umd/popper.min.js"><
/script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></s
cript>

</body>
</html>
```

Setup 6: Update dashboard.html:

Add **Training Program** function to dashboard:

```
<!-- Training Programs Card -->
<div class="col-md-3">
    <div class="card">
        <img src="https://cdn-icons-png.flaticon.com/512/2920/2920265.png"
class="card-img-top" alt="Training Programs">
        <div class="card-body">
            <h5 class="card-title">Training Programs</h5>
            <p class="card-text">Manage and view training programs.</p>
            <a href="/trainingPrograms" class="btn btn-primary">Go to Training
Programs</a>
        </div>
    </div>
</div>
```

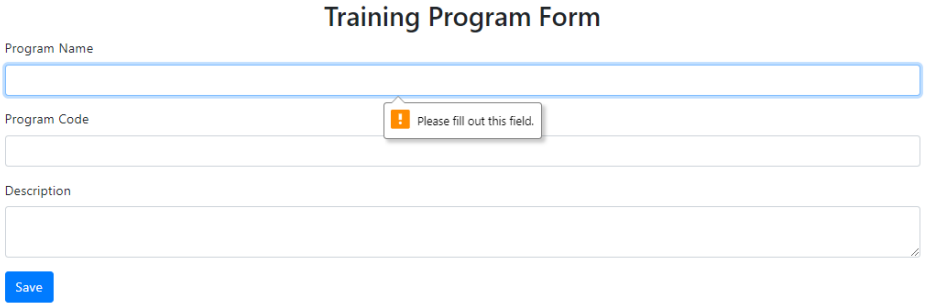
Step 7: Run and Test

Run the Spring Boot application, and here are some screenshots:

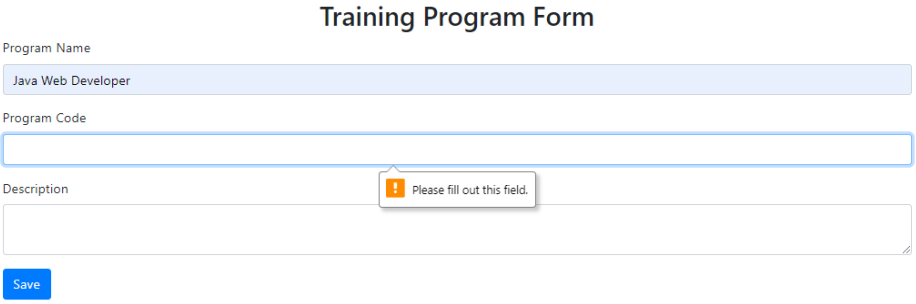
For the [**Training Program**] function: http://localhost:8080/trainingPrograms

Click to [**Add New Training Program**] button then input the information:

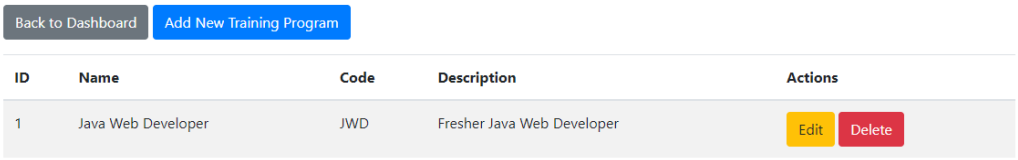
If you don’t enter the Program Name:



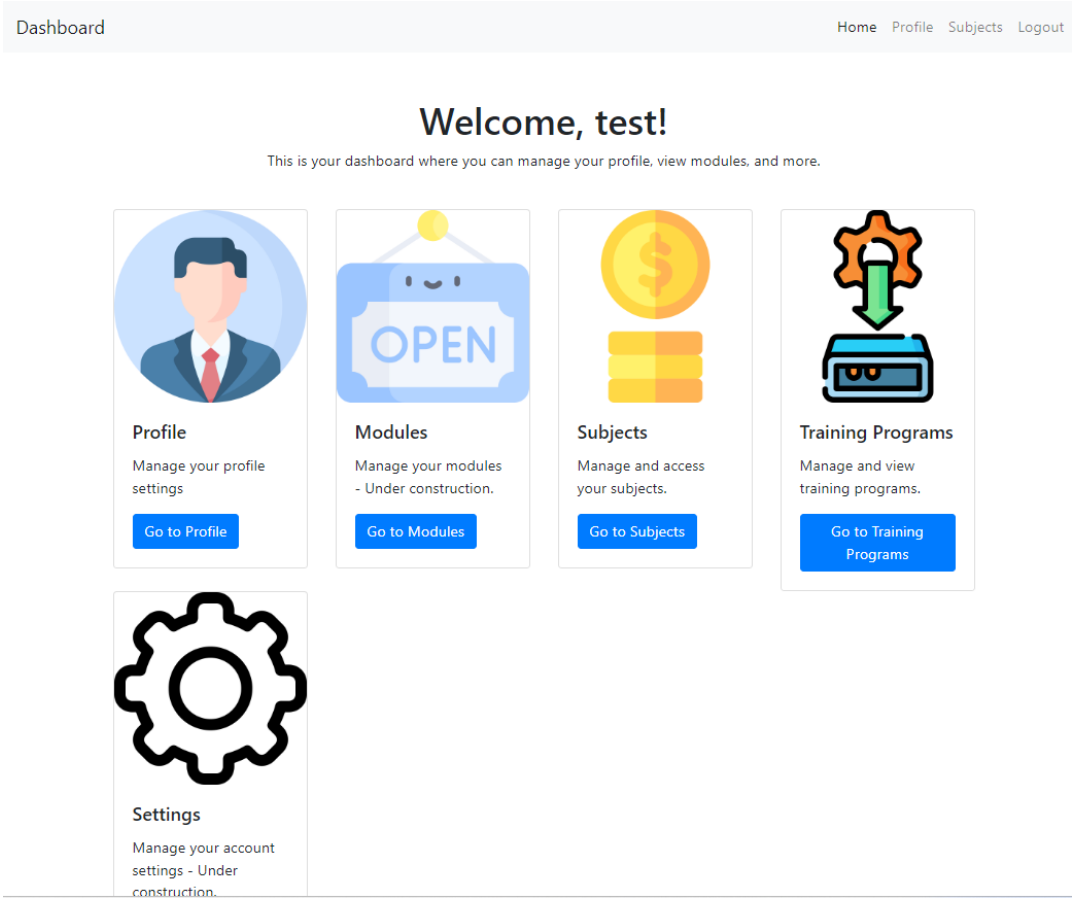
Or if you don’t enter the [Program Code]:



After adding a training program, here the result:



For the [**Dashboard**] function: http://localhost:8080/dashboard


Dashboard
Home Profile Subjects Logout
Welcome, test!
This is your dashboard where you can manage your profile, view modules, and more.
Profile
Manage your profile settings
Go to Profile
OPEN
Modules
Manage your modules - Under construction.
Go to Modules
Subjects
Manage and access your subjects.
Go to Subjects
Training Programs
Manage and view training programs.
Go to Training Programs
Settings
Manage your account settings - Under construction.

----o0o-----

THE END