*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

| | | |
|---|---|---|
| FRESHER ACADEMY | CODE: | JSFW_Lab_03_Opt2 |
| | TYPE: | SHORT |
| | LOC: | 200 |
| | DURATION: | 120 MINUTES |

# Java Spring Framework Introduction

## Objectives:

- Understand how to manage application data using model and controller classes in Spring MVC.
- Learn to configure and use Spring MVC to handle web requests and interact with in-memory data storage.

## Lab Specifications:

- In this lab exercise, you will work with a University Management System where you will manage Course and Subject entities using Spring MVC controller. The focus will be on implementing CRUD operations and handling data through Spring MVC.

## Problem Description:

- Trainees will define model classes for Course and Subject. These classes represent the entities in your application and hold the data attributes.
- Trainees will create controllers to handle HTTP requests and manage the interaction between the model and the views. This includes viewing all courses, adding new courses, and managing subjects within courses.

## Prerequisites:

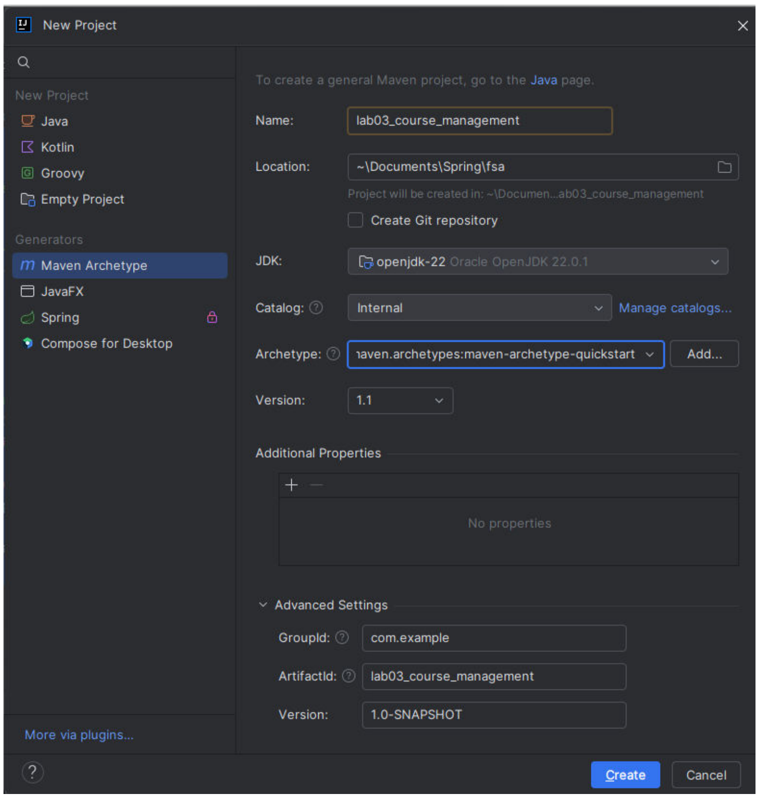
- Using Java SDK version 8.0 at least.
- Using Maven.
- Using Spring Framework 5.0 or higher version.

## 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, and then choose maven-archetype-quickstart as the archetype.
- Click Next and set the project name to **lab03_course_management**
- Set the groupId to com.example and artifactId to **lab03_ course_management**

- Click **Create**.



New Project

New Project: Java, Kotlin, Groovy, Empty Project

Generators: Maven Archetype, JavaFX, Spring, Compose for Desktop

More via plugins...

To create a general Maven project, go to the Java page.

Name: lab03_course_management

Location: ~\Documents\Spring\fsa

Project will be created in: ~\Documen...ab03_course_management

Create Git repository

JDK: openjdk-22 Oracle OpenJDK 22.0.1

Catalog: Internal Manage catalogs...

Archetype: ıaven.archetypes:maven-archetype-quickstart Add...

Version: 1.1

Additional Properties

No properties

Advanced Settings

GroupId: com.example

ArtifactId: lab03_course_management

Version: 1.0-SNAPSHOT

Create Cancel

**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>2.5.3</version>   <!—1-->
   <relativePath />
 </parent>

 <dependencies>
  <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>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>org.junit.vintage</groupId>
        <artifactId>junit-vintage-engine</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <!-- tag::lombokDependency[] -->
```

Spring Boot version

Starter dependencies

```
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
  <!-- end::lombokDependency[] -->
  <!-- tag::validationStarter[] -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
  </dependency>
  <!-- end::validationStarter[] -->
</dependencies>
```

**Step 3: Write a main class:**

```java
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CourseApplication {
    public static void main(String[] args) {
        SpringApplication.run(CourseApplication.class, args);
    }
}
```

**Step 4: Establishing the domain:**

An application's domain is the subject area that it addresses—the ideas and concepts that influence the understanding of the application. In the Course application, the domain includes such objects as **Subject**, **Course**, and **CourseRegister**. These domain objects represent the key entities and relationships within the application.

- **Subject**: A subject represents a topic or area of study, identified by a unique ID and a name.
- **Course**: A course is a collection of subjects grouped together under a common name.
- **CourseRegister**: The course register manages the list of courses, allowing new courses to be added and existing courses to be retrieved.

To get started, we'll focus on the **Subject** and **Course**. These classes define the core domain objects you need to build and understand the application.

1. **Subject** class:

```java
package com.example.model;

public class Subject {
    private String id;
    private String name;

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

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

**2. Course** class:

```
package com.example.model;

import java.util.ArrayList;
import java.util.List;

public class Course {
    private String name;
    private List<Subject> subjects = new ArrayList<>();

    public Course() {}

    public Course(String name, List<Subject> subjects) {
        this.name = name;
        this.subjects = subjects;
    }

    public String getName() {
        return name;
    }

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

    public List<Subject> getSubjects() {
        return subjects;
    }

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

    public void addSubject(Subject subject) {
        this.subjects.add(subject);
    }
}
```

**3. CourseRegister** class:

- **The CourseRegister** class maintains a collection of courses and provides methods to add new courses, retrieve the list of courses, and find a course by its name.

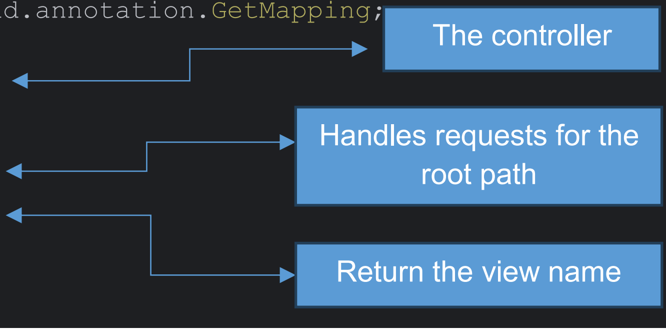
```java
package com.example.model;

import java.util.ArrayList;
import java.util.List;

public class CourseRegister {
    private List<Course> courses;

    public CourseRegister() {
        this.courses = new ArrayList<>();
        initializeSampleData();
    }

    public List<Course> getCourses() {
        return courses;
    }

    public void addCourse(Course course) {
        this.courses.add(course);
    }

    public Course findCourseByName(String name) {
        return courses.stream()
                .filter(course -> course.getName().equalsIgnoreCase(name))
                .findFirst()
                .orElse(null);
    }

    private void initializeSampleData() {
        // Sample Subjects
        Subject subject1 = new Subject("S1", "Mathematics 101");
        Subject subject2 = new Subject("S2", "Physics 101");
        Subject subject3 = new Subject("S3", "Chemistry 101");
        Subject subject4 = new Subject("S4", "English Literature");

        // Sample Courses
        Course course1 = new Course("Engineering", new ArrayList<>());
        course1.addSubject(subject1);
        course1.addSubject(subject2);

        Course course2 = new Course("Science", new ArrayList<>());
        course2.addSubject(subject3);
        course2.addSubject(subject4);

        Course course3 = new Course("Arts", new ArrayList<>());
        course3.addSubject(new Subject("S5", "History"));
        course3.addSubject(new Subject("S6", "Sociology"));

        // Adding Courses to CourseRegister
        courses.add(course1);
        courses.add(course2);
        courses.add(course3);
    }
}
```

**Step 5: Write controllers:**

1. **HomeController** class:

Write a simple controller class that handles requests for the root path (for example, /) and forwards those requests to the homepage view without populating any model data. The following listing shows the simple controller class.

```java
package com.example;


import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller             // <1>
public class HomeController {

    @GetMapping("/")      // <2>
    public String home() {
        return "home";      // <3>
    }

}
```



**2. SubjectController** class:

Write a controller class that handles requests related to subjects. This controller manages the creation, viewing, and association of subjects within courses. The following listing shows the SubjectController class:

```java
package com.example.controller;

import com.example.model.Course;
import com.example.model.Subject;
import com.example.model.CourseRegister;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.atomic.AtomicLong;

@Controller
public class SubjectController {

    private final CourseRegister courseRegister = new CourseRegister();
    private final AtomicLong idCounter = new AtomicLong();

    @GetMapping("/subjects")
    public String viewAllSubjects(Model model) {
        List<Subject> allSubjects = new ArrayList<>();
        for (Course course : courseRegister.getCourses()) {
            allSubjects.addAll(course.getSubjects());
        }
        model.addAttribute("subjects", allSubjects);
        return "viewAllSubjects";
    }

    @GetMapping("/courses/{courseName}/subjects")
    public String viewCourseSubjects(@PathVariable("courseName") String
courseName, Model model) {
        Course course = courseRegister.findCourseByName(courseName);
        if (course == null) {
            return "error/404"; // Redirect to an error page if the course is
not found
        }
        model.addAttribute("course", course);
        return "viewCourseSubjects";
    }

    @GetMapping("/courses/{courseName}/subjects/add")
    public String addSubjectForm(@PathVariable String courseName, Model model) {
        model.addAttribute("subject", new Subject());
        model.addAttribute("courseName", courseName);
        return "addSubject";
    }

    @PostMapping("/courses/{courseName}/subjects/add")
    public String addSubjectToCourse(@PathVariable String courseName,
@ModelAttribute Subject subject) {
        Course course = courseRegister.findCourseByName(courseName);
        if (course != null) {
            if (subject.getId() == null || subject.getId().isEmpty()) {
                subject.setId(generateUniqueSubjectId(course));
            }
            course.addSubject(subject);
        }
        return "redirect:/courses/" + courseName + "/subjects";
    }

    @GetMapping("/courses")
    public String viewCourses(Model model) {
        model.addAttribute("courses", courseRegister.getCourses());
        return "viewCourses";
    }

    @GetMapping("/courses/add")
    public String addCourseForm(Model model) {
        model.addAttribute("course", new Course());
        return "addCourse";
    }

    @PostMapping("/courses/add")
    public String addCourse(@ModelAttribute Course course) {
        courseRegister.addCourse(course);
        return "redirect:/courses";
    }

    // Method to generate a unique ID based on the current number of subjects in
the course
    private String generateUniqueSubjectId(Course course) {
        int newId = course.getSubjects().size() + 1;
        return "S" + newId; // Simple example: S1, S2, S3, etc.
    }
}
```

**Step 6: Defining views:**

The next listing shows the basic Thymeleaf templates:

1. index.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Course Management</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 class="text-center">Course Management</h1>
    <div class="text-center mt-4">
        <a href="/courses" class="btn btn-primary btn-lg">Manage Courses</a>
        <a href="/subjects" class="btn btn-secondary btn-lg">View All
Subjects</a>
    </div>
</div>
</body>
</html>
```

**2. viewCourses.html:**

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>View Courses</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>Courses</h1>
    <a href="/" class="btn btn-primary">Back to Home</a>
    <a href="/courses/add" class="btn btn-success">Add New Course</a>
    <table class="table table-bordered mt-4">
        <thead>
        <tr>
            <th>Name</th>
            <th>Actions</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="course : ${courses}">
            <td th:text="${course.name}">Course Name</td>
            <td>
                <a th:href="@{'/courses/' + ${course.name} + '/subjects'}"
class="btn btn-info">View Subjects</a>
            </td>
        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

**3. addCourse.html:**

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Course</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>Add Course</h1>
    <a href="/courses" class="btn btn-primary">Back to Courses</a>
    <form action="#" th:action="@{/courses/add}" th:object="${course}"
method="post" class="mt-3">
        <div class="form-group">
            <label for="name">Course Name:</label>
            <input type="text" id="name" class="form-control" th:field="*{name}"
placeholder="Enter course name" required/>
        </div>
        <button type="submit" class="btn btn-success">Add Course</button>
    </form>
</div>
</body>
</html>
```

**4. viewCourseSubjects.html:**

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Subjects of Course</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>Subjects of Course: <span th:text="${course.name}">Course
Name</span></h1>
    <a href="/courses" class="btn btn-primary">Back to Courses</a>
    <a th:href="@{'/courses/' + ${course.name} + '/subjects/add'}" class="btn
btn-success">Add New Subject</a>
    <table class="table table-bordered mt-4">
        <thead>
        <tr>
            <th>Subject ID</th>
            <th>Subject Name</th>
        </tr>
        </thead>
        <tbody>
        <tr th:each="subject : ${course.subjects}">
            <td th:text="${subject.id}">Subject ID</td>
            <td th:text="${subject.name}">Subject Name</td>
        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

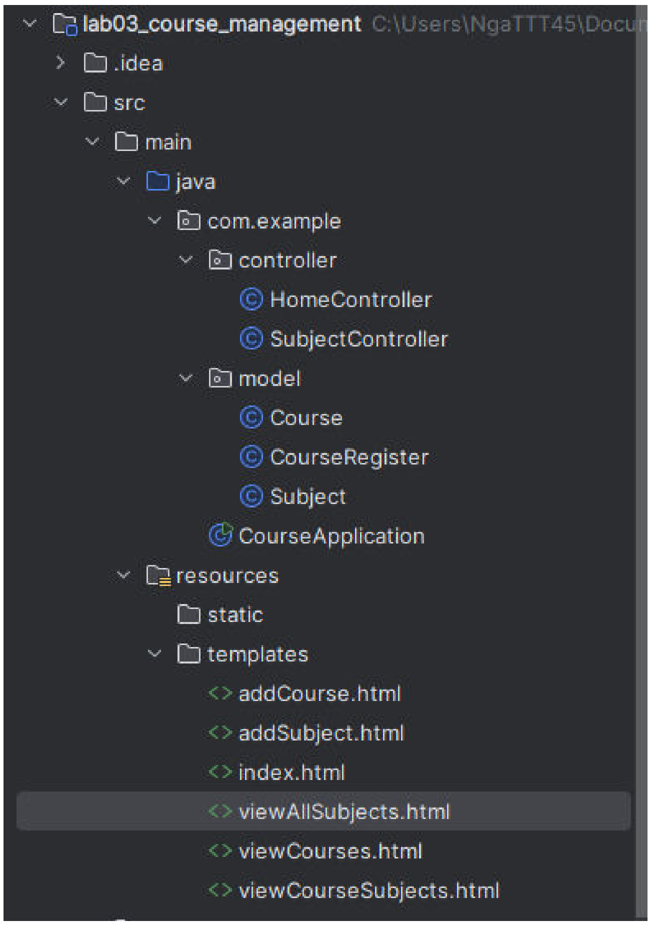
**5. addSubject.html:**

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Add Subject</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>Add Subject to Course: <span th:text="${courseName}">Course
Name</span></h1>
    <a th:href="@{'/courses/' + ${courseName} + '/subjects'}" class="btn btn-
primary">Back to Subjects</a>
    <form action="#" th:action="@{'/courses/' + ${courseName} +
'/subjects/add'}" th:object="${subject}" method="post" class="mt-3">
        <div class="form-group">
            <label for="id">Subject ID:</label>
            <input type="text" id="id" class="form-control" th:field="*{id}"
placeholder="Enter ID or leave blank for auto-generation" />
        </div>
        <div class="form-group">
            <label for="name">Subject Name:</label>
            <input type="text" id="name" class="form-control" th:field="*{name}"
placeholder="Enter subject name" required/>
        </div>
        <button type="submit" class="btn btn-success">Add Subject</button>
    </form>
</div>
</body>
</html>
```

**6. viewAllSubjects.html:**

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
    <title>All Subjects</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>All Subjects</h1>
    <a href="/" class="btn btn-primary mb-3">Back to Home</a>
    <table class="table table-bordered">
        <thead>
        <tr>
            <th>Subject ID</th>
            <th>Subject Name</th>

        </tr>
        </thead>
        <tbody>
        <tr th:each="subject : ${subjects}">
            <td th:text="${subject.id}">Subject ID</td>
            <td th:text="${subject.name}">Subject Name</td>

        </tr>
        </tbody>
    </table>
</div>
</body>
</html>
```

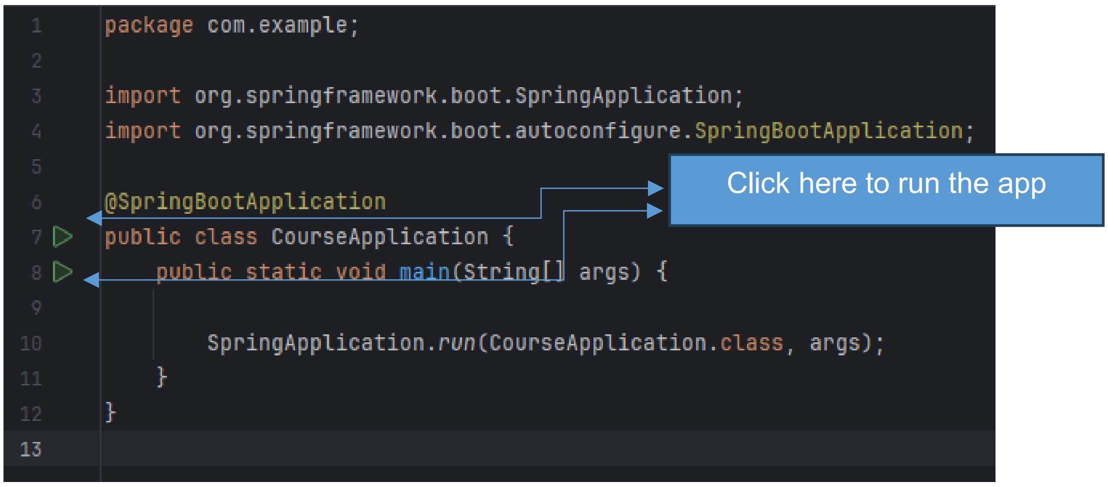
**Here is the structure of the application:**



```
lab03_course_management C:\Users\NgaTTT45\Docu
  .idea
  src
    main
      java
        com.example
          controller
            HomeController
            SubjectController
          model
            Course
            CourseRegister
            Subject
          CourseApplication
      resources
        static
        templates
          addCourse.html
          addSubject.html
          index.html
          viewAllSubjects.html
          viewCourses.html
          viewCourseSubjects.html
```

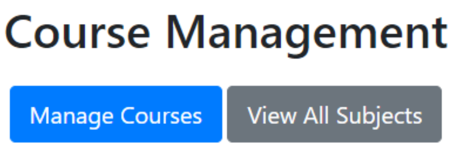
**Step 6: Run the application:**

In the CourseApplication class, click to the icon in the left (in green color) before [public class…] or [main] method to run the application.

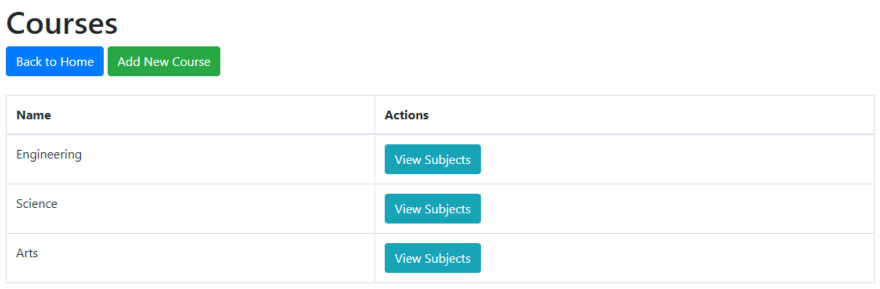


```java
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CourseApplication {
    public static void main(String[] args) {

        SpringApplication.run(CourseApplication.class, args);
    }
}
```

Now that the application has started, point your web browser to http://localhost:8080, you should see something like this:



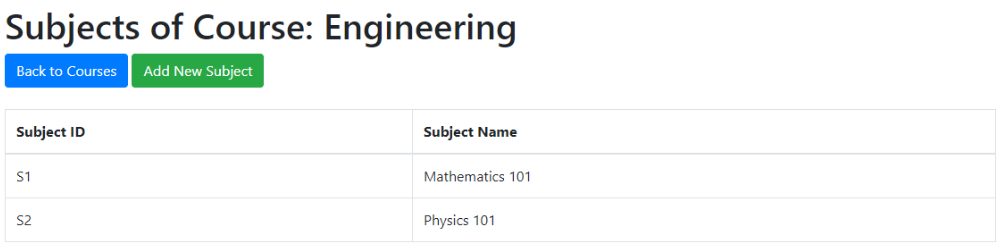
When you click to [Manage Courses]:


Courses

Back to Home Add New Course

| Name | Actions |
|---|---|
| Engineering | View Subjects |
| Science | View Subjects |
| Arts | View Subjects |

When you click to [View Subjects] of “Engineering” course:


Subjects of Course: Engineering

Back to Courses Add New Subject

| Subject ID | Subject Name |
|---|---|
| S1 | Mathematics 101 |
| S2 | Physics 101 |

**Step 6: Testing the controllers:**

Add dependencies to the pom.xml file:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
  <exclusions>
    <exclusion>
      <groupId>org.junit.vintage</groupId>
      <artifactId>junit-vintage-engine</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```

Here is the HomeControllerTest:

```
package com.example;


import static org.hamcrest.Matchers.containsString;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static
```

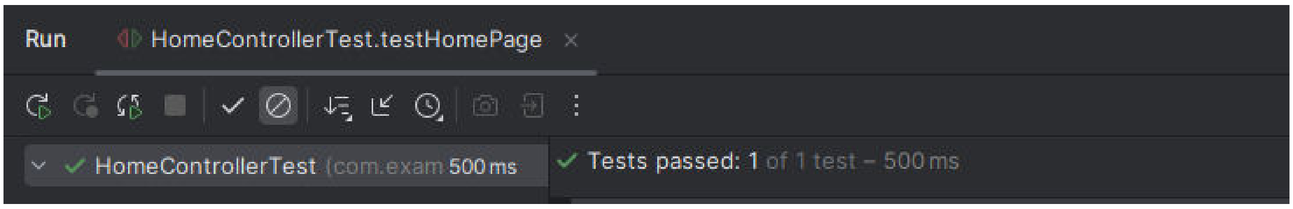
```
org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;

import com.example.controller.HomeController;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;

@WebMvcTest(HomeController.class)
public class HomeControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testHomePage() throws Exception {
        mockMvc.perform(get("/"))
                .andExpect(status().isOk())
                .andExpect(view().name("index"))
                .andExpect(content().string(containsString("Course
Management")));
    }
}
```

2. **SubjectControllerTest** class:

```
package com.example.controller;

import static org.hamcrest.Matchers.containsString;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.view;

import com.example.model.Course;
import com.example.model.CourseRegister;
import com.example.model.Subject;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.test.web.servlet.MockMvc;

import java.util.Collections;

@WebMvcTest(SubjectController.class)
public class SubjectControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private SubjectController subjectController;

    @BeforeEach
    public void setup() {
        // Create a mock course and subject
        // Create mock data
        CourseRegister courseRegister = new CourseRegister();
        Subject testSubject = new Subject("S1", "Test Subject Description");
        Course testCourse = new Course("testCourse",
Collections.singletonList(testSubject));

        // Add the course to the course register
        courseRegister.addCourse(testCourse);

    }

    @Test
    public void testAddSubjectForm() throws Exception {
        mockMvc.perform(get("/courses/testCourse/subjects/add"))
                .andExpect(status().isOk())
                .andExpect(view().name("addSubject"))
                .andExpect(content().string(containsString("Add Subject to
Course")));
    }

    @Test
    public void testViewAllSubjects() throws Exception {
        mockMvc.perform(get("/subjects"))
                .andExpect(status().isOk())
                .andExpect(view().name("viewAllSubjects"))
                .andExpect(content().string(containsString("All Subjects")));
    }


    @Test
    public void testAddSubjectToCourse() throws Exception {
        mockMvc.perform(post("/courses/testCourse/subjects/add")
                        .param("name", "Test Subject"))
                .andExpect(status().is3xxRedirection())

.andExpect(view().name("redirect:/courses/testCourse/subjects"));
    }

    @Test
    public void testViewCourses() throws Exception {
        mockMvc.perform(get("/courses"))
                .andExpect(status().isOk())
                .andExpect(view().name("viewCourses"))
                .andExpect(content().string(containsString("Courses")));
    }

    @Test
    public void testAddCourseForm() throws Exception {
        mockMvc.perform(get("/courses/add"))
                .andExpect(status().isOk())
                .andExpect(view().name("addCourse"))
                .andExpect(content().string(containsString("Course Name")));
    }

    @Test
    public void testAddCourse() throws Exception {
        mockMvc.perform(post("/courses/add")
                        .param("name", "Test Course"))
                .andExpect(status().is3xxRedirection())
                .andExpect(view().name("redirect:/courses"));
    }
}
```

Click the icon on the left side of the files to run test cases:

Here is the result after running test:



```
Run    HomeControllerTest.testHomePage ×

HomeControllerTest (com.exam 500 ms    Tests passed: 1 of 1 test – 500 ms
```

This application provides a comprehensive way to manage courses and subjects, enabling users to view, add, and organize course-related information effectively. The use of Spring MVC allows for clear separation of concerns, with controllers handling user requests and model data being managed independently.

----oOo-----

**THE END**