**Case Study Title: Online Course Enrollment System**

**Scenario:**

An educational startup wants to build a basic web application for students to view available courses and enroll online. The company has a small IT team familiar with Java and wants to use Spring MVC to ensure the application follows a clean, maintainable structure based on MVC architecture

**Objectives:**

1. Display a list of available courses.

2. Allow students to register by filling out an enrollment form.

3. Confirm enrollment and store student details.

**System Requirements:**

• Java 17 or later

• Spring MVC framework

• Apache Tomcat or embedded server

• Maven for dependency management

• JSP for frontend

• Eclipse or Spring Tool Suite (STS) IDE

**How Spring MVC Helps:**

Spring MVC allows the application to be divided into three main components:

**Layer Responsibility**

Model -Represents the data (Course, Student, Enrollment info)

View -Represents the data (Course, Student, Enrollment info)

Controller -Displays the HTML pages for course listing and form input

**Application Flow:**

Manages user requests and application logic

1. User accesses the homepage

→ A controller handles this request and returns a list of available courses via the view.

2. User selects a course and proceeds to enroll

→ A new view (HTML form) is presented to collect user data (name, email, etc.).

3. Form is submitted

→ The controller receives the form data, validates it, and passes it to the service layer or

model to be processed.

4. Success page is shown

→ A confirmation view is displayed with enrollment details.

Components in Spring MVC:

Component Description

@Controller Handles web requests (e.g., show courses, process enrollment)

@RequestMapping Maps URLs to specific controller methods

Model object Holds the data to be passed to the view

@ComponentScan Auto-detects components (controllers, services, etc.)

ViewResolver Resolves the view name to an actual view (e.g., JSP page)

Beans.xml or Java Defines Spring beans, view resolvers, and component scanning

Config setup

**Example Use Cases:**

1. CourseController

◦/ courses → Displays list of courses

◦ /enroll → Shows enrollment form

◦ /submitEnrollment → Processes submitted data

**//pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>online-course-enrollment</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

<spring.version>5.3.30</spring.version>

</properties>

<dependencies>

<!-- Spring MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- JSTL for JSP -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- Servlet API -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

</project>

**//web.xml**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<display-name>Online Course Enrollment</display-name>

<!-- Spring Dispatcher Servlet -->

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>redirect.jsp</welcome-file>

</welcome-file-list>

</web-app>

**//Dispatcher-servlet.xml**

@Controller

public class CourseController {

@Autowired

private CourseService courseService;

@Autowired

private EnrollmentService enrollmentService;

// Show list of courses

@GetMapping("/courses")

public String listCourses(Model model) {

model.addAttribute("courses", courseService.getAllCourses());

return "courses";

}

// Show enrollment form

@GetMapping("/enroll")

public String showEnrollmentForm(@RequestParam("courseId") int courseId, Model model) {

Course course = courseService.getCourseById(courseId);

model.addAttribute("course", course);

model.addAttribute("student", new Student());

return "enroll";

}

// Process enrollment form

@PostMapping("/submitEnrollment")

public String submitEnrollment(@ModelAttribute("student") Student student, Model model) {

enrollmentService.saveEnrollment(student);

model.addAttribute("student", student);

return "success";

}

}

**//Course.java**

package com.example.model;

public class Course {

private int id;

private String name;

private String description;

public Course() {}

public Course(int id, String name, String description) {

this.id = id;

this.name = name;

this.description = description;

}

public int getId() { return id; }

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

public String getName() { return name; }

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

public String getDescription() { return description; }

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

}

**//Student.java**

package com.example.model;

public class Student {

private String name;

private String email;

private String selectedCourse;

public Student() {}

public Student(String name, String email, String selectedCourse) {

this.name = name;

this.email = email;

this.selectedCourse = selectedCourse;

}

public String getName() { return name; }

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

public String getEmail() { return email; }

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

public String getSelectedCourse() { return selectedCourse; }

public void setSelectedCourse(String selectedCourse) { this.selectedCourse = selectedCourse; }

}

**//CourseServive.java**

package com.example.service;

import com.example.model.Course;

import java.util.List;

public interface CourseService {

List<Course> getAllCourses();

Course getCourseById(int id);

}

**//CourseServiceImpl.java**

package com.example.service;

import com.example.model.Course;

import org.springframework.stereotype.Service;

import java.util.Arrays;

import java.util.List;

@Service

public class CourseServiceImpl implements CourseService {

private List<Course> courses = Arrays.asList(

new Course(1, "Java", "Learn Java"),

new Course(2, "Spring MVC", "Build web apps"),

new Course(3, "Database", "Learn SQL and database")

);

@Override

public List<Course> getAllCourses() {

return courses;

}

@Override

public Course getCourseById(int id) {

return courses.stream().filter(c -> c.getId() == id).findFirst().orElse(null);

}

}

**//EnrollmentService.java**

package com.example.service;

import com.example.model.Student;

public interface EnrollmentService {

void saveEnrollment(Student student);

}

**//EnrollmentServiceImpl.java**

package com.example.service;

import com.example.model.Student;

import org.springframework.stereotype.Service;

@Service

public class EnrollmentServiceImpl implements EnrollmentService {

@Override

public void saveEnrollment(Student student) {

System.out.println("Enrolled Student: " + student.getName() ", Email: " + student.getEmail() + ",

Course: " + student.getSelectedCourse());

}

}

**//CourseController.java**

package com.example.controller;

import com.example.model.Course;

import com.example.model.Student;

import com.example.service.CourseService;

import com.example.service.EnrollmentService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;

@Controller

public class CourseController {

@Autowired

private CourseService courseService;

@Autowired

private EnrollmentService enrollmentService;

@GetMapping("/courses")

public String listCourses(Model model) {

model.addAttribute("courses", courseService.getAllCourses());

return "courses";

}

@GetMapping("/enroll")

public String showEnrollmentForm(@RequestParam("courseId") int courseId, Model model) {

Course course = courseService.getCourseById(courseId);

model.addAttribute("course", course);

model.addAttribute("student", new Student());

return "enroll";

}

@PostMapping("/submitEnrollment")

public String submitEnrollment(@ModelAttribute("student") Student student, Model model) {

enrollmentService.saveEnrollment(student);

model.addAttribute("student", student);

return "success";

}

}

**2. Views (JSP)**

**courses.jsp** → Displays all courses

<%@ page contentType="text/html;charset=UTF-8" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>

<head><title>Available Courses</title></head>

<body>

<h2>Available Courses</h2>

<table border="1">

<tr><th>Course</th><th>Description</th><th>Action</th></tr>

<c:forEach var="course" items="${courses}">

<tr>

<td>${course.name}</td>

<td>${course.description}</td>

<td><a href="enroll?courseId=${course.id}">Enroll</a></td>

</tr>

</c:forEach>

</table>

</body>

</html>

**enroll.jsp** → Input form for registration

<%@ page contentType="text/html;charset=UTF-8" %>

<html>

<head><title>Enroll</title></head>

<body>

<h2>Enroll in ${course.name}</h2>

<form action="submitEnrollment" method="post">

<input type="hidden" name="selectedCourse" value="${course.name}" />

Name: <input type="text" name="name" required /><br/><br/>

Email: <input type="email" name="email" required /><br/><br/>

<button type="submit">Submit</button>

</form>

</body>

</html>

**success.jsp** → Confirmation message

<%@ page contentType="text/html;charset=UTF-8" %>

<html>

<head><title>Enrollment Successful</title></head>

<body>

<h2>Enrollment Successful!</h2>

<p>Thank you, ${student.name}. You have successfully enrolled in ${student.selectedCourse}.</p>

</body>

</html>

**Case Study Title: Online Shopping Portal – Order**

**Processing Monitoring**

**Scenario Description**

An online shopping portal provides a service class OrderService that has three key methods:

1. addToCart(String product)

2. placeOrder(String orderId)

3. cancelOrder(String orderId)

As a developer, you want to add cross-cutting concerns like:

• Logging when methods start (@Before)

• Logging after successful method execution (@AfterReturning)

• Logging errors when a method fails (@AfterThrowing)

• Performing cleanup or logging after any method execution, success or failure (@After)

**Spring AOP Setup Components**

1. Business Logic Class

OrderService — contains methods like addToCart, placeOrder, cancelOrder.

2. Aspect Class: OrderLoggingAspect

This class uses four annotations:

**Annotation Purpose**

@Before Logs method entry

@AfterReturning Logs method success result

@AfterThrowing Logs if any exception occurs

@After Logs method exit regardless of outcome

**Flow with Annotations**

Let’s walk through what happens when a user places an order.

**Method: placeOrder("ORD123")**

Step Annotation What Happens

1 @Before Log: “Starting method: placeOrder with order ID: ORD123”

2 — Business Logic — The order is placed successfully

3 @AfterReturning Log: “Order placed successfully: ORD123”

4 @After Log: “Method placeOrder execution finished”

**Method: placeOrder("INVALID\_ID")**

Step Annotation What Happen

1 @Before Log: “Starting method: placeOrder with order ID:ORD123”

2 — Business Logic — The order is placed successfully

3 @AfterReturning Log: “Order placed successfully: ORD123”

4 @After Log: “Method placeOrder execution finished”

**Aspect Class Summary**

Advice Type Trigger Condition Example Log Message

@Before Just before the method execution "Calling method: addToCart"

@AfterReturning When method returns successfully “addToCart completed successfully for product: X"

@AfterThrowing When method throws an exception "Error occurred during addToCart: ProductNotFound"

@After After method finishes (success or error) "addToCart method execution ended

**//pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>spring-aop-shopping</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

<spring.version>5.3.30</spring.version>

</properties>

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- AspectJ -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.22</version>

</dependency>

</dependencies>

</project>

**//OrderService.java**

package com.example.service;

import org.springframework.stereotype.Service;

@Service

public class OrderService {

public void addToCart(String product) {

System.out.println("Adding product to cart: " + product);

}

public void placeOrder(String orderId) {

if ("INVALID\_ID".equals(orderId)) {

throw new RuntimeException("OrderNotFoundException");

}

System.out.println("Placing order with ID: " + orderId);

}

public void cancelOrder(String orderId) {

if ("INVALID\_CANCEL".equals(orderId)) {

throw new RuntimeException("CancelFailedException");

}

System.out.println("Cancelling order with ID: " + orderId);

}

}

**//OrderLoggingAspect.java**

package com.example.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.\*;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class OrderLoggingAspect {

@Before("execution(\* com.example.service.OrderService.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("[BEFORE] Starting method: " + joinPoint.getSignature().getName()

+ " with arguments: " + java.util.Arrays.toString(joinPoint.getArgs()));

}

@AfterReturning(pointcut = "execution(\* com.example.service.OrderService.\*(..))", returning =

"result")

public void logAfterReturning(JoinPoint joinPoint, Object result) {

System.out.println("[AFTER RETURNING] Method " + joinPoint.getSignature().getName()

+ " executed successfully.");

}

@AfterThrowing(pointcut = "execution(\* com.example.service.OrderService.\*(..))", throwing =

"error")

public void logAfterThrowing(JoinPoint joinPoint, Throwable error) {

System.out.println("[AFTER THROWING] Exception in method: " +

joinPoint.getSignature().getName()

+ " - " + error.getMessage());

}

// After method execution (success or failure)

@After("execution(\* com.example.service.OrderService.\*(..))")

public void logAfter(JoinPoint joinPoint) {

System.out.println("[AFTER] Method " + joinPoint.getSignature().getName() + " execution

finished.");

}

}

**//spring-aop-config.xml**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- Scan for @Component, @Service, @Aspect -->

<context:component-scan base-package="com.example" />

<!-- Enable @AspectJ style annotations -->

<aop:aspectj-autoproxy />

</beans>

**//AppMain.java**

package com.example.main;

import com.example.service.OrderService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AppMain {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("spring-aop-config.xml");

OrderService orderService = context.getBean(OrderService.class);

System.out.println("=== Valid Order ===");

orderService.addToCart("Laptop");

orderService.placeOrder("ORD123");

System.out.println("\n=== Invalid Order ===");

try {

orderService.placeOrder("INVALID\_ID");

} catch (Exception e) {

// Exception handled

}

System.out.println("\n=== Cancel Order ===");

orderService.cancelOrder("ORD123");

System.out.println("\n=== Invalid Cancel ===");

try {

orderService.cancelOrder("INVALID\_CANCEL");

} catch (Exception e) {

// Exception handled

}

}

}