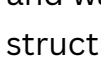
 Case Study Title: Online Course Enrollment System

 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. 2. 3. Display a list of available courses. Allow students to register by filling out an enrollment form. Confirm enrollment and store student details.

 System Requirements:

• Java 17 or later

• Spring MVC framework

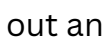
• Apache Tomcat or embedded server

• 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 Model Responsibility Represents the data (Course, Student, Enrollment info) View Displays the HTML pages for course listing and form input Controller Manages user requests and application logic

 Application Flow:

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

@Controller Description Handles web requests (e.g., show courses, process enrollment) @RequestMapping Model object Maps URLs to specific controller methods 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 Config

📄 Example Use Cases: 1. CourseController Defines Spring beans, view resolvers, and component scanning setup

- /courses → Displays list of courses
- /enroll → Shows enrollment form
- /submitEnrollment → Processes submitted data

2. Views (JSP)

- courses.jsp → Displays all courses
- enroll.jsp → Input form for registration
- success.jsp → Confirmation message

pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.enrollment</groupId>
  <artifactId>course-enrollment-system</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <properties>
    <java.version>17</java.version>
  </properties>

  <dependencies>
    <!-- Spring MVC -->
    <dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>6.1.4</version>
    </dependency>

    <!-- Servlet API -->
    <dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
        <version>6.0.0</version>
        <scope>provided</scope>
    </dependency>

    <!-- JSP & JSTL -->
    <dependency>
        <groupId>jakarta.servlet.jsp.jstl</groupId>
        <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
        <version>3.0.0</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.web</groupId>
        <artifactId>jakarta.servlet.jsp.jstl</artifactId>
        <version>3.0.1</version>
    </dependency>
</dependencies>
```

```
  <build>
    <plugins>
      <!-- Compiler Plugin -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.11.0</version>
        <configuration>
          <source>17</source>
          <target>17</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

web.xml:

```
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee" version="6.0">
  <display-name>Online Course Enrollment</display-name>


  <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>
</web-app>
```

Course.java:

```
@Controller
public class CourseController {

    @GetMapping("/courses")
    public String showCourses(Model model) {
        List<Course> courses = List.of(
            new Course(1, "Spring Boot"),
            new Course(2, "Java Fundamentals"));
        model.addAttribute("courses", courses);
        return "courses";
    }

    @GetMapping("/enroll")
    public String showEnrollForm(Model model) {
        model.addAttribute("student", new Student());
```

```
        return "enroll";
    }

    @PostMapping("/submitEnrollment")
    public String processEnrollment(@ModelAttribute Student student, Model model) {
        model.addAttribute("student", student);
        return "success";
    }
}
```

courseController.java:

```
package com.enrollment.controller;

import com.enrollment.model.Course;
import com.enrollment.model.Student;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.Arrays;
import java.util.List;

@Controller
public class CourseController {

    // Show list of courses
```

```
    @GetMapping("/courses")
    public String showCourses(Model model) {
        List<Course> courses = Arrays.asList(
            new Course(1, "Spring MVC", "Learn Spring MVC fundamentals"),
            new Course(2, "Java 17", "Master Java 17 features"),
            new Course(3, "Hibernate ORM", "Persist data with Hibernate")
        );
        model.addAttribute("courses", courses);
        return "courses";  // JSP: /WEB-INF/views/courses.jsp
    }

    // Show enrollment form
    @GetMapping("/enroll")
    public String showEnrollmentForm(@RequestParam("courseId") int courseId, Model model)
{
        Student student = new Student();
        student.setCourseId(courseId);  // Pre-fill selected course
        model.addAttribute("student", student);
        return "enroll";  // JSP: /WEB-INF/views/enroll.jsp
    }

    // Handle form submission
    @PostMapping("/submitEnrollment")
    public String submitEnrollment(@ModelAttribute("student") Student student, Model
model) {
        // Normally, you'd save student info to the database here
        model.addAttribute("student", student);
```

```
        return "success";  // JSP: /WEB-INF/views/success.jsp
    }
}
```

Student.java:

```
package com.enrollment.model;


public class Student {


    private String name;
    private String email;
    private int courseId;


    // Constructors
    public Student() {
    }


    public Student(String name, String email, int courseId) {
        this.name = name;
        this.email = email;
        this.courseId = courseId;
    }


    // Getters and Setters
    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 int getCourseId() {
        return courseId;
    }

    public void setCourseId(int courseId) {
        this.courseId = courseId;
    }
}
```

WebConfig.java:

```
package com.enrollment.config;
```

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.enrollment")
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void configureViewResolvers(ViewResolverRegistry registry) {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/"); // location of JSP files
        resolver.setSuffix(".jsp");           // file extension
        registry.viewResolver(resolver);
    }
}
```

course.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="com.enrollment.model.Course" %>
```

```
<html>
<head>
  <title>Available Courses</title>
</head>
<body>
  <h2>Available Courses</h2>

  <%
    List<Course> courses = (List<Course>) request.getAttribute("courses");

    if (courses == null || courses.isEmpty()) {
  %>
    <p>No courses available at the moment.</p>
  <%
    } else {
      for (Course course : courses) {
  %>
    <div style="border: 1px solid #aaa; padding: 10px; margin: 10px;">
      <h3><%= course.getName() %></h3>
      <p><strong>Description:</strong> <%= course.getDescription() %></p>

      <form action="enroll" method="get">
        <input type="hidden" name="courseId" value="<%= course.getId() %>">
        <input type="submit" value="Enroll">
      </form>
    </div>
```

```
    <%
            }
        }
    %>
</body>
</html>
```

enroll.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="com.enrollment.model.Course" %>

<html>
<head>
    <title>Course Enrollment</title>
</head>
<body>
    <h2>Enroll in Course</h2>

    <%
        Course course = (Course) request.getAttribute("course");

        if (course == null) {
    %>
        <p>Invalid course selected.</p>
    <%
        } else {
```

```
    %>
        <h3>Course: <%= course.getName() %></h3>
        <p><strong>Description:</strong> <%= course.getDescription() %></p>

        <form action="submitEnrollment" method="post">
            <input type="hidden" name="courseId" value="<%= course.getId() %>">

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

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

            <input type="submit" value="Submit Enrollment">
        </form>
    <%
        }
    %>
</body>
</html>
```

success.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="com.enrollment.model.Student" %>
<%@ page import="com.enrollment.model.Course" %>
```

```
<html>
<head>
  <title>Enrollment Successful</title>
</head>
<body>
  <h2>Enrollment Confirmation</h2>


  <%
    Student student = (Student) request.getAttribute("student");
    Course course = (Course) request.getAttribute("course");


    if (student != null && course != null) {
  %>
    <p>Thank you, <strong><%= student.getName() %></strong>, for enrolling in <strong><%= course.getName() %></strong>.</p>
    <p>We've sent a confirmation to your email: <strong><%= student.getEmail() %></strong>.</p>
  <%
    } else {
  %>
    <p>Enrollment information is missing or incomplete.</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.

OrderService.java:

```
package com.shopping.service;


public class OrderService {


    public void addToCart(String product) {

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

    }


    public void placeOrder(String orderId) {

        if (orderId.equals("INVALID_ID")) {

            throw new RuntimeException("OrderNotFoundException");

        }

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

```
    }

    public void cancelOrder(String orderId) {
        System.out.println("Cancelling order: " + orderId);
    }
}
```

OrderLoggingAspect.java:

```
package com.shopping.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class OrderLoggingAspect {

    @Before("execution(* com.shopping.service.OrderService.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("Starting method: " + joinPoint.getSignature().getName()
                + " with arguments: " + java.util.Arrays.toString(joinPoint.getArgs()));
    }

    @AfterReturning(pointcut = "execution(* com.shopping.service.OrderService.*(..))",
returning = "result")
    public void logAfterReturning(JoinPoint joinPoint, Object result) {
```

```
        System.out.println("Method " + joinPoint.getSignature().getName() + " completed successfully.");
    }

    @AfterThrowing(pointcut = "execution(* com.shopping.service.OrderService.*(..))", throwing = "ex")
    public void logAfterThrowing(JoinPoint joinPoint, Throwable ex) {
        System.out.println("Exception in method: " + joinPoint.getSignature().getName()
                + ". Exception: " + ex.getMessage());
    }

    @After("execution(* com.shopping.service.OrderService.*(..))")
    public void logAfter(JoinPoint joinPoint) {
        System.out.println("Method " + joinPoint.getSignature().getName() + " execution finished.");
    }
}
```

AppConfig.java:

```
package com.shopping.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration
@ComponentScan(basePackages = "com.shopping")
@EnableAspectJAutoProxy
```

```
public class AppConfig {

}
```

MainApp.java:

```
package com.shopping;


import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;


import com.shopping.config.AppConfig;

import com.shopping.service.OrderService;


public class MainApp {

    public static void main(String[] args) {

        ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);


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


        orderService.addToCart("Laptop");

        orderService.placeOrder("ORD123");


        try {

            orderService.placeOrder("INVALID_ID");

        } catch (Exception e) {

            // Expected exception

        }
```

```
        orderService.cancelOrder("ORD123");
    }
}
```