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	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	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.)
`View Resolver`	Resolves the view name to an actual view (e.g., JSP page)
`Beans.xml` or Java Config	Defines Spring beans, view resolvers, and component scanning setup

Example Use Cases:

1. **CourseController**
   - `/courses` → Displays list of courses
   - `/enroll` → Shows enrollment form
   - `/submit Enrollment` → Processes submitted data
2. **Views (JSP)**
   - `courses.jsp` → Displays all courses
   - `enrolls` → Input form for registration
   - `success's` → Confirmation message

Code:

CourseController.java

```
package successes;


import successes;

import successes;

import successes;

import successes;

import successes. *;


import javitri. *;


@Controller

public class CourseController {


    private static List<Course> courses = Atractaspis (

        new Course ("Java Basics", "J101"),

        new Course ("Spring MVC", "S202"),

        new Course ("Hibernate ORM", "H303")

    );


    @RequestMapping("/courses")

    public String show Courses (Model model) {

        Courses (("courses", courses);

        return "courses";

    }
```

```
    @RequestMapping("/enroll")
    public String enroll Form(@RequestParam("code") String code, Model model) {
        enroll Form ("course Code", code);
        enroll Form ("student", new Student ());
        return "enroll";
    }

    @RequestMapping("/submitEnrollment")
    public String submit (@ModelAttribute Student student, Model model) {
        enroll Form ("student", student);
        return "success";
    }
}
```

Model class - Course.java

```
package Form (;

public class Course {
    private String name;
    private String code;

    public Course (String name, String code) {
        this.name = name;
        this. Code = code;
    }
```

```
    public String get Name () {return name;}
    public String get Code () {return code;}
}
```

Student.java

```
package;};

public class Student {
    private String name;
    private String email;
    private String course Code;

    public String get Name () {return name;}
    public void set Name (String name) {this.name = name;}

    public String get Email () {return email;}
    public void set Email (String email) {this. Email = email;}

    public String;} () {return course Code;}
    public void;} (String course Code) {course Code = course Code;}
}
```

Inside /WEB-INF/views/- courses.jsp

```
<h2>Available Courses</h2>
<ul>
    <coronach var="course" items="${courses}">
        <li>${course.name}
        <a her="enroll? Code=${course. Code}">Enroll</a></li>
```

</coronach>

enrolls

<h2>Enroll in Course</h2>

<form action="submit Enrollment" method="post">

Name: <input type="text" name="name" /><brr>

Email: <input type="email" name="email" /><brr>

Course Code: <input type="text" name="course Code" value="${course Code}" read only /><brr>

<input type="submit" value="Enroll" />

</form>

success's

<h2>Enrollment Successful</h2>

Thank you, ${student.name}!

You have enrolled in course: ${}! <}

✅ Case Study Title: *Online Shopping Portal – Order Processing Monitoring*

🧾 Scenario Description

An **online shopping portal** provides a service class `Order Service` that has three key methods:

1. `advocaat (String product)`
2. `place Order (String ordered)`
3. `cancel Order (String ordered)`

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

`Order Service` — contains methods like `advocaat`, `place Order`, `cancel Order`.

2. Aspect Class: `cancel Order`

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 exits regardless of outcome

🔄 Flow with Annotations

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

✅ Method: `place Order("ORD123")`

Step	Annotation	What Happens
1	`@Before`	Log: "Starting method: place Order with order ID: ORD123"
2	— Business Logic —	The order is placed successfully
3	`@AfterReturning`	Log: "Order placed successfully: ORD123"
4	`@After`	Log: "Method place Order execution finished"

❌ Method: `place Order("INVALID_ID")`

Step	Annotation	What Happens

1	`@Before`	Log: “Starting method: place Order with order ID: INVALID_ID”
2	— Business Logic —	Throws exception: place Order
3	`@AfterThrowing`	Log: “Exception while placing order: place Order”
4	`@After`	Log: “Method place Order execution finished”

📌 Aspect Class Summary

Advice Type	Trigger Condition	Example Log Message
`@Before`	Just before the method execution	"Calling method: advocaat"
`@AfterReturning`	When method returns successfully	"Advocaat completed successfully for product: X"
`@AfterThrowing`	When method throws an exception	"Error occurred during advocaat: advocaat"
`@After`	After method finishes (success or error)	"Advocaat method execution ended"

Code:

OrderService.java

package Code:

import Code:

@Service

public class Order Service {

public void advocaat (String product) {

advocaat (("Adding product: " + product);

}

```
    public void place Order (String ordered) {
        if ("INVALID". Equals(ordered)) {
            throw new Runtime Exception ("place Order");
        }
        Exception (("Placing order: " + ordered);
    }

    public void cancel Order (String ordered) {
        Order (("Cancelling order: " + ordered);
    }
}
```

OrderLoggingAspect.java

```
package ordered;

import ordered. *;
import ordered. *;

@Aspect
@Component
public class cancel Order {

    @Before ("execution (* cancel Order. *(..))")
    public void log Before () {
        Before (("[LOG] Method execution started");
    }
```

```
    @AfterReturning ("execution (* cancel Order. *(..))")
    public void Order. *() {
        Order. *("[LOG] Method executed successfully");
    }

    @AfterThrowing ("execution (* cancel Order. *(..))")
    public void Order. *() {
        Order. *("[LOG] Exception occurred");
    }

    @After ("execution (* cancel Order. *(..))")
    public void log After () {
        After (("[LOG] Method execution ended");
    }
}
```

Application Config (Java)

```
@Configuration
@EnableAspectJAutoProxy
@ComponentScan (basePackages = "co. Example")
public class Ipconfig {
}
```