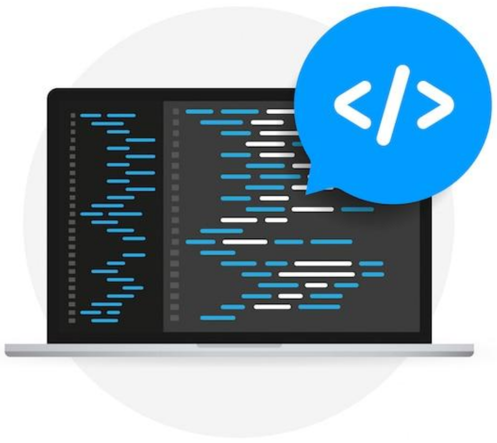
UEF
UNIVERSITY OF ECONOMICS & FINANCE

JAVA TECHNOLOGY



Abstract

This practical manual is designed to guide students in building dynamic web applications using Java based on the Spring MVC architecture, with JSP for the user interface and JDBC for connecting to a Microsoft SQL Server database. Through a series of hands-on labs, students will practice managing data flow across the MVC layers (Controller – Model – View), performing CRUD operations (Create, Read, Update, Delete) on real-world databases, and organizing project code logically following the MVC pattern.

The material also includes instructions on: Setting up the SQL Server JDBC Driver; Creating a sample database schema; Common troubleshooting techniques when working with SQL

Hoang Van Hieu, Msc.

FACULTY OF INFORMATION TECHNOLOGY

Internal circulation, 2025

TABLE OF CONTENTS

LAB 1. INTRODUCTION TO SPRING MVC

1. OBJECTIVE

After the lesson, students will be able to:

– Understand the Spring MVC architecture.

– Create a Maven project with Spring MVC.

– Configure DispatcherServlet and ViewResolver.

– Write a Controller to handle data via URL (GET) and Form (POST).

– Display user data on a JSP interface.

2. CREATE A SPRING MVC PROJECT

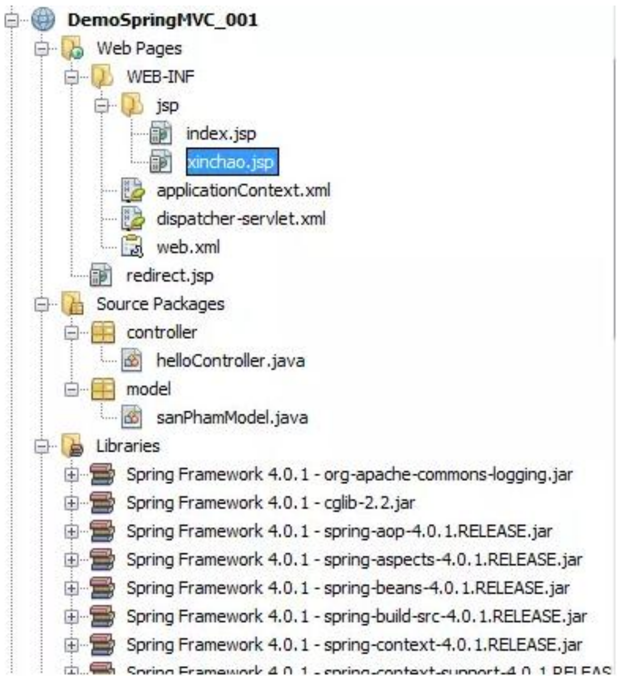
2.1. Create a Maven Web Project

– Open NetBeans

– Navigate to: **File → New Project → Maven → Web Application**

– Set project name: **DemoSpringMVC_001**

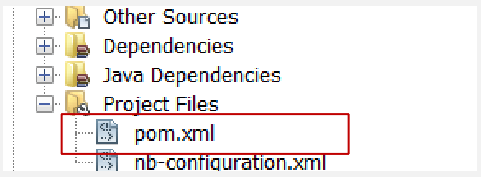
– The generated structure should look like:



```
DemoSpringMVC_001
  Web Pages
    WEB-INF
      jsp
        index.jsp
        xinchao.jsp
      applicationContext.xml
      dispatcher-servlet.xml
      web.xml
    redirect.jsp
  Source Packages
    controller
      helloController.java
    model
      sanPhamModel.java
  Libraries
    Spring Framework 4.0.1 - org-apache-commons-logging.jar
    Spring Framework 4.0.1 - cglib-2.2.jar
    Spring Framework 4.0.1 - spring-aop-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-aspects-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-beans-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-build-src-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-context-4.0.1.RELEASE.jar
    Spring Framework 4.0.1 - spring-context-support-4.0.1.RELEAS
```

2.2. Configure pom.xml

Open the file **pom.xml** and insert the following into the <dependencies> section:



```
<dependencies>
        <!-- Spring MVC -->
        <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-webmvc</artifactId>
           <version>5.3.9</version>
        </dependency>
        <!-- Servlet API (provided by the container, such as Tomcat) -->
        <dependency>
           <groupId>javax.servlet</groupId>
           <artifactId>javax.servlet-api</artifactId>
           <version>4.0.1</version>
           <scope>provided</scope>
        </dependency>
        <!-- JSTL -->
        <dependency>
           <groupId>javax.servlet</groupId>
           <artifactId>jstl</artifactId>
           <version>1.2</version>
        </dependency>
</dependencies>
```

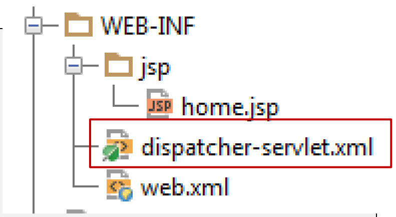
Save the file and wait for the IDE to download the libraries. If not, **right-click** → **Reload Maven Project.**

3. CONFIGURE SPRING MVC

3.1. Create dispatcher-servlet.xml

Create a new file: **WEB-INF/dispatcher-servlet.xml**. You can use a different name instead of using the name **"dispatcher",** follow the naming conventions.

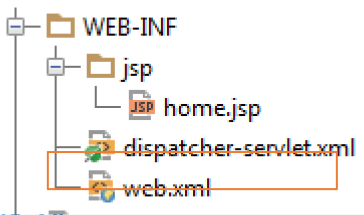
After creating the file, replace all current content with the following content:



```
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:context="http://www.springframework.org/schema/context"
     xmlns:mvc="http://www.springframework.org/schema/mvc"
     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/mvc
        http://www.springframework.org/schema/mvc/spring-mvc.xsd">

  <!-- Scan Java classes with annotations such as @Controller -->
```

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

<!--Enable the use of annotations such as @RequestMapping -->
<mvc:annotation-driven/>

<!-- Configure the view resolver -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/views/"/>
  <property name="suffix" value=".jsp"/>
</bean>

</beans>
```

3.2. Configure web.xml



- Open file: **WEB-INF/web.xml**
- Replace all current content with the following content:

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">

        <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>
        <!-- DispatcherServlet handles all requests at “/” -->
        <servlet-mapping>
            <servlet-name>dispatcher</servlet-name>
            <url-pattern>/</url-pattern>
        </servlet-mapping>

</web-app>
```

- It uses the dispatcher-servlet.xml file to configure controllers, views, beans, etc."

4. CREATE THE CONTROLLER

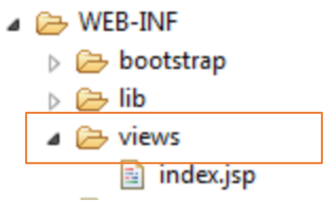
- Create a package: **com.example.controller** (when creating a package containing a Controller, you can use a different prefix; it doesn't have to be "com.example")
- Create class: **HelloController.java**.
- After creating the file, replace all current content with the following content:

```
package com.example.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

@Controller  // @Controller: Marks this as a controller
public class HelloController {

   @RequestMapping("/hello") //Handles requests to /hello.
   public String sayHello(Model model) {
      model.addAttribute("message", "Hi! Spring MVC!"); // Passes data to the view.
      return "hello"; // Will look for the hello.jsp file
   }
   // Display the form
   @RequestMapping(value = "/showForm", method = RequestMethod.GET)
   public String showForm() {
      return "input-form";
   }

   // Handle POST submission
   @RequestMapping(value = "/processForm", method = RequestMethod.POST)
   public String processForm(@RequestParam("name") String name, Model model) {
      String message = "Hello " + name + "!";
      model.addAttribute("message", message);
      return "greet";
   }

   // (Optional) Handle GET with query param: /greet?name=An
   @RequestMapping(value = "/greet", method = RequestMethod.GET)
   public String greetUser(@RequestParam("name") String name, Model model) {
      model.addAttribute("message", "Hello " + name + "!");
      return "greet";
   }
}
```

5. CREATE JSP VIEWS

5.1. Create views directory

– Create folder: **/WEB-INF/views**



5.2. Create View

❖ **Create a new hello.jsp file - View**

After creating the file, replace all current content with the following content:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en-US">
        <head>
            <title>Welcome</title>
        </head>
        <body>
            <h2>${message}</h2>
        </body>
</html>
```

❖ **Create a new input-form.jsp file - View**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en-US">
        <head>
            <title>Enter your name</title>
        </head>
        <body>
            <h2>Please enter your name:</h2>
            <form action="processForm" method="post">
                <input type="text" name="name" required>
                <button type="submit">Submit</button>
            </form>
        </body>
</html>
```

❖ **Create a new greet.jsp file – View**

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html lang="en-US">
        <head>
            <title>Result</title>
        </head>
        <body>
            <h2>${message}</h2>
        </body>
</html>
```

6. RUN & TEST

– Successfully build the project.
– Deploy it to Apache Tomcat.
– Open browser → test GET and POST requests (e.g., /showForm).

6.1. Add Apache Tomcat to NetBeans

- Go to **Tools → Servers**
- Click Add Server
- Choose **Apache Tomcat** → click **Next**
- Browse and select your Tomcat installation folder **(e.g., C:\apache-tomcat-9.x)**
- Set admin username/password if prompted.
- Click **Finish**

If you haven't installed Tomcat yet, download it here: https://tomcat.apache.org/download-90.cgi

6.2. Configure Maven to build WAR file

- Open pom.xml
- Add the following section if not present:

```
<build>
   <finalName>DemoSpringMVC_001</finalName>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-war-plugin</artifactId>
         <version>3.3.2</version>
      </plugin>
   </plugins>
</build>
```

This ensures the deployed WAR file is named DemoSpringMVC_001, so your URL will be: **http://localhost:8080/DemoSpringMVC_001/**

6.3. Clean and Build, Run the Project

❖ **Clean and Build**

- Right-click the project → select **Clean and Build**
- Check the **Output** window → make sure there are **no build errors**

❖ **Run the Project on Tomcat**

- Right-click your project → select **Run**
- NetBeans will start Apache Tomcat (it may take a few seconds)
- Your default browser should open: http://localhost:8080/**DemoSpringMVC_001/**

– Then manually go to: http://localhost:8080/SpringHelloApp/showForm or http://localhost:8080/SpringHelloApp/hello

❖ **Test the Application**

– A form appears asking for a name.

– You enter "An" and click Submit.

– The next page shows: "Hello An!"

❖ **You continue to implement the additional requirements:**

– Add another field for email and display both name and email.

– Create a form with name and age, and output: ***"Hello An, you are 20 years old!"***

6.4. Common Errors & Fixes

Error	Cause & Solution
HTTP 404 Not Found	Incorrect URL (check /showForm), servlet not mapped properly
HTTP 500 Internal Server Error	Syntax error in JSP or missing JSTL library
Tomcat not showing up	Tomcat server not added/configured in NetBeans
Missing web.xml	Ensure web.xml is located at: /src/main/webapp/WEB-INF/web.xml

7. ADVANCED PRACTICE EXERCISES

Exercise 1. Validate empty name and redirect back to form

If the name input is empty:

– Do not display greeting

– Show error message: "Name cannot be empty"

– Reload form with message.

❖ **Update HelloController**

```
@Controller
public class HelloController {

//……
    @RequestMapping(value = "/processForm", method = RequestMethod.POST)
    public String processForm(@RequestParam("name") String name, Model model) {
        if (name == null || name.trim().isEmpty()) {
            model.addAttribute("error", "Name cannot be empty.");
            return "input-form"; // stay on the form page
        }
        model.addAttribute("message", "Hello " + name + "!");
        return "greet";
    }
//……
}
```

@RequestParam("name") is an **annotation** in Spring MVC used to retrieve the value of a parameter from the request URL or from an HTML form.

❖ **Update input-form.jsp**

– Add JSTL dependency (if not yet), in pom.xml:

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

– Also add this to the top of your input-form.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <head>
        <title>Enter your name</title>
    </head>
    <body>
        <h2>Please enter your name:</h2>

        <!-- Show error if exists -->
        <c:if test="${not empty error}">
            <p style="color: red;">${error}</p>
        </c:if>

        <form action="processForm" method="post">
            <input type="text" name="name" required>
            <button type="submit">Submit</button>
        </form>
    </body>
</html>
```

Exercise 2. Greeting with Name and Age

❖ Requirements:

– Create a form with two fields: name (text), age (number)

– After submission via POST, the application must:

– Check if the name is not empty.

– Display the result: “Hello [Name], you are [Age] years old!”
– If the name is empty:
  - Stay on the form page.
  - Display the error message: “Name cannot be empty.”

Exercise 3. Greeting with Name and Email

❖ Requirements:

– Create a form with the following fields: name (text), email (text)
– After form submission via **POST**, your app must display a message like: **“Hello [Name]! We’ve sent a confirmation to: [Email].”**
– Add basic validation, if name or email is empty:
  - Stay on the form page.
  - Show: **“Please fill in all required fields.”**

LAB 2. WORKING WITH MODEL IN SPRING MVC

1. INTRODUCTION

In Spring MVC, the Model represents the data passed from the Controller to the View for display. Working with the Model is essential to implement features such as lists, object details, and form data submission.

In this Lab, students will become familiar with 08 typical use cases of using Model, presented step by step with complete source code, configurations, and detailed explanations.

2. OBJECTIVE

After the lesson, students will be able to:

- Understand the role of the Model in the Spring MVC architecture.
- Use the Model, ModelMap, and ModelAndView objects to pass data from the controller to the view.
- Work with different data types in the model:
  - Primitive values (String, int, double)
  - Single object (POJO)
  - List of objects
  - Map of key-value pairs
  - Nested object (object with list or another object)
- Receive data from the view via form submission and pass it through the model.
- Apply best practices when organizing model classes with clear structure and relationships.

3. PROJECT SETUP

- Create a Maven Spring MVC project.
- Add dependencies to **pom.xml**

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.20</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>
</dependencies>
```

- Configure Dispatcher Servlet in **web.xml**

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

- Configure View Resolver in **dispatcher-servlet.xml**

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

  <!--Enable the use of annotations such as @RequestMapping -->
  <mvc:annotation-driven/>

  <!-- Configure the view resolver -->
  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
  </bean>
```

4. COMPLETE THE FOLLOWING REQUIREMENTS

Passing Primitive Variables from Controller to View

4.1. Display a welcome message and the current year.

- **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {
  @RequestMapping("/hello")
  public String showHello(Model model) {
    model.addAttribute("message", "Welcome to Spring MVC!");
    model.addAttribute("year", 2025);
    return "hello";
  }
}
```

❖ **View (hello.jsp):**

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello Page</title>
</head>
<body>
  <h2>${message}</h2>
  <p>Current Year: ${year}</p>
</body>
</html>
```

❖ **Explanation:**

- The controller passes two simple variables via `model.addAttribute()`.
- The view accesses them with `${}` using Expression Language.

4.2. Passing an Object

❖ **Model:**

```
public class Student {
  private String name;
  private int age;
  private String email;

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

  public String getName() { return name; }
  public int getAge() { return age; }
  public String getEmail() { return email; }
}
```

❖ **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class StudentController {
  @RequestMapping("/student")
  public String studentDetail(Model model) {
    Student st = new Student("John Doe", 21, "john@example.com");
    model.addAttribute("student", st);
    return "student-detail";
  }
}
```

❖ **View (student-detail.jsp):**

```
<!DOCTYPE html>
<html>
<head>
  <title>Student Detail</title>
</head>
<body>
  <h2>Student Information</h2>
  <p>Name: ${student.name}</p>
  <p>Age: ${student.age}</p>
  <p>Email: ${student.email}</p>
</body>
</html>
```

❖ **Explanation:**

- The controller creates an instance of `Student` and passes it to the view.
- The view accesses object properties using `${object.property}`.

4.3. Passing a List of Objects

❖ **Model:**

```
public class Product {
  private int id;
  private String name;
  private double price;

  public Product(int id, String name, double price) {
    this.id = id;
    this.name = name;
    this.price = price;
  }

  public int getId() { return id; }
  public String getName() { return name; }
  public double getPrice() { return price; }
}
```

❖ **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

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

@Controller
```

```
public class ProductController {
  @RequestMapping("/products")
  public String listProducts(Model model) {
    List<Product> list = new ArrayList<>();
    list.add(new Product(1, "Laptop", 1500));
    list.add(new Product(2, "Mouse", 300));
    model.addAttribute("products", list);
    return "product-list";
  }
}
```

❖ **View (product-list.jsp):**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
  <title>Product List</title>
</head>
<body>
  <h2>Products</h2>
  <table border="1">
    <tr><th>ID</th><th>Name</th><th>Price</th></tr>
    <c:forEach var="p" items="${products}">
      <tr>
        <td>${p.id}</td>
        <td>${p.name}</td>
        <td>${p.price}</td>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

❖ **Explanation:**

- A list of Product objects is passed from Controller.
- JSTL `<c:forEach>` is used in the JSP to iterate and display each item.

4.4. Passing a Map to the View

Display province codes and names passed from Controller.

❖ **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import java.util.LinkedHashMap;
import java.util.Map;

@Controller
public class ProvinceController {
  @RequestMapping("/provinces")
  public String showProvinces(Model model) {
    Map<String, String> provinces = new LinkedHashMap<>();
    provinces.put("HCM", "Ho Chi Minh City");
    provinces.put("HN", "Ha Noi");
    provinces.put("DN", "Da Nang");
    model.addAttribute("provinces", provinces);
    return "province";
  }
}
```

❖ **View (province.jsp):**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
  <title>Province List</title>
</head>
<body>
  <h2>List of Provinces</h2>
  <c:forEach var="entry" items="${provinces}">
    <p>${entry.key} - ${entry.value}</p>
  </c:forEach>
</body>
</html>
```

❖ **Explanation:**

The controller sends a Map as a model attribute. The JSP uses JSTL to iterate through the entries.

4.5. Object with List Property

❖ **Models:**

```
public class Order {
  private String item;
  private double price;

  public Order(String item, double price) {
    this.item = item;
    this.price = price;
  }

  public String getItem() { return item; }
  public double getPrice() { return price; }
}

public class Customer {
  private String name;
  private List<Order> orders;

  public Customer(String name, List<Order> orders) {
    this.name = name;
    this.orders = orders;
  }

  public String getName() { return name; }
  public List<Order> getOrders() { return orders; }
}
```

- **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import java.util.List;

@Controller
public class CustomerController {
  @RequestMapping("/customer")
  public String customerDetail(Model model) {
    Customer c = new Customer("Tran B", List.of(
      new Order("iPhone", 1000),
      new Order("AirPods", 200)
    ));
    model.addAttribute("customer", c);
    return "customer";
  }
}
```

- **View (customer.jsp):**

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
  <title>Customer Details</title>
</head>
<body>
  <h2>Customer: ${customer.name}</h2>
  <ul>
    <c:forEach var="o" items="${customer.orders}">
      <li>${o.item} - ${o.price}</li>
    </c:forEach>
  </ul>
</body>
</html>
```

- **Explanation**

Demonstrates an object with a list property. The View accesses nested data using dot notation.

4.6. Receive Data from Form

- **View (form.jsp):**

```
<!DOCTYPE html>
<html>
<head>
  <title>Input Name</title>
</head>
<body>
  <form action="submit" method="post">
    Enter your name: <input type="text" name="name" />
    <input type="submit" value="Submit" />
  </form>
```

```
</body>
</html>
```

❖ **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class FormController {
  @PostMapping("/submit")
  public String handleForm(@RequestParam String name, Model model) {
    model.addAttribute("name", name);
    return "result";
  }
}
```

❖ **View (result.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Result</title></head>
<body>
  <p>Hello, ${name}!</p>
</body>
</html>
```

❖ **Explanation**

Demonstrates using `@RequestParam` to receive user input and return it to the View.

4.7. Pass Data through Redirect

❖ **Controller:**

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
public class RedirectController {
  @RequestMapping("/login")
  public String loginSuccess(RedirectAttributes ra) {
    ra.addFlashAttribute("msg", "Login successful!");
    return "redirect:/dashboard";
  }

  @RequestMapping("/dashboard")
  public String showDashboard() {
    return "dashboard";
  }
}
```

❖ **View (dashboard.jsp):**

```
<!DOCTYPE html>
<html>
  <head><title>Dashboard</title></head>
  <body>
    <h2>${msg}</h2>
  </body>
</html>
```

❖ **Explanation**

`addFlashAttribute()` allows passing temporary data after redirecting.

5. SUMMARY AND EXERCISES

All examples follow the standard Spring MVC structure: Controller → Model → View. Each scenario reflects real-world use cases in web application development.

Students are encouraged to:

– Create additional model classes (e.g., Book, Category)

– Practice binding data from forms using @ModelAttribute

– Combine with view layout and conditional rendering in tables

❖ **Suggested Exercises:**

– Create a Book class (id, title, author, price). Write a controller to send a list of books.

– Display the list in a table view. If price > 100, highlight in red.

– Create a form for users to input a new Book and display it immediately.

LAB 3. WORKING WITH CONTROLLER IN SPRING MVC

1. INTRODUCTION

In this session, students will learn how to define, configure, and work with Controllers in Spring MVC. Controllers are the central components that receive requests from the user, process the business logic (or coordinate with the service/model layers), and determine which view to return.

Key objectives:

- Understand the role and structure of a Controller in Spring MVC
- Learn to map requests using `@RequestMapping`, `@GetMapping`, and `@PostMapping`
- Handle form submissions and path variables
- Return responses to views using `Model`, `ModelAndView`

2. CONTROLLER BASICS

2.1. Creating a Simple Controller

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {
  @RequestMapping("/")
  public String home() {
    return "index";
  }
}
```

When the user accesses the root URL (`/`), Spring will render the `index.jsp` view.

2.2. Mapping Multiple URLs

```
@Controller
public class PageController {
  @RequestMapping("/about")
  public String aboutPage() {
    return "about";
  }

  @RequestMapping("/contact")
  public String contactPage() {
    return "contact";
  }
}
```

You can map different methods to different paths.

3. HANDLING PARAMETERS AND PATH VARIABLES

3.1. Using @RequestParam

❖ **Controller**

```
@Controller
public class GreetingController {
  @RequestMapping("/greet")
  public String greetUser(@RequestParam String name, Model model) {
    model.addAttribute("name", name);
    return "greeting";
  }
}
```

Example URL: `/greet?name=Alice`

❖ **View (greeting.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Greeting</title></head>
<body>
  <p>Hello, ${name}!</p>
</body>
</html>
```

3.2. Using `@PathVariable`

```
@Controller
public class ProfileController {
  @RequestMapping("/profile/{username}")
  public String userProfile(@PathVariable String username, Model model) {
    model.addAttribute("username", username);
    return "profile";
  }
}
```

Example URL: `/profile/johndoe`

4. SUBMITTING FORM DATA

4.1. Show Form

```
<!DOCTYPE html>
<html>
<head><title>Form</title></head>
<body>
  <form action="submit" method="post">
    Name: <input type="text" name="name" />
    <input type="submit" value="Submit" />
  </form>
</body>
```

```
</html>
```

4.2. Handle Submission

❖ **Controller**

```
@Controller
public class FormSubmitController {
  @PostMapping("/submit")
  public String processForm(@RequestParam String name, Model model) {
    model.addAttribute("name", name);
    return "result";
  }
}
```

❖ **View (result.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Result</title></head>
<body>
  <p>Welcome, ${name}!</p>
</body>
</html>
```

4.3. Using `ModelAndView`

```
@Controller
public class InfoController {
  @RequestMapping("/info")
  public ModelAndView showInfo() {
    ModelAndView mv = new ModelAndView("info");
    mv.addObject("course", "Spring MVC Practical Class");
    mv.addObject("semester", "Spring 2025");
    return mv;
  }
}
```

❖ **View (info.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Info</title></head>
<body>
  <h2>Course: ${course}</h2>
  <p>Semester: ${semester}</p>
</body>
</html>
```

5. ADVANCED CONTROLLER USE CASES

5.1. Example of related models

❖ **Model: Category.java**

```
public class Category {
  private int id;
  private String name;

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

  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; }
}
```

❖ **Model: Product.java**

```
public class Product {
  private int id;
  private String name;
  private double price;
  private Category category;

  public Product() {}
  public Product(int id, String name, double price, Category category) {
    this.id = id;
    this.name = name;
    this.price = price;
    this.category = category;
  }

  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 double getPrice() { return price; }
  public void setPrice(double price) { this.price = price; }
  public Category getCategory() { return category; }
  public void setCategory(Category category) { this.category = category; }
}
```

❖ **Controller:**

```
@Controller
public class ProductCategoryController {
  @RequestMapping("/product-detail")
  public String showProductDetail(Model model) {
    Category c = new Category(1, "Electronics");
    Product p = new Product(101, "Smartphone", 899.99, c);
    model.addAttribute("product", p);
    return "product-detail";
  }
}
```

❖ **View (product-detail.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Product Detail</title></head>
<body>
  <h2>Product Information</h2>
  <p>Name: ${product.name}</p>
  <p>Price: ${product.price}</p>
  <p>Category: ${product.category.name}</p>
</body>
</html>
```

5.2. Handling Optional Parameters and Default Values

```
@Controller
public class OptionalParamController {
  @RequestMapping("/search")
  public String search(@RequestParam(defaultValue = "all") String category, Model model)
{
    model.addAttribute("category", category);
    return "search-result";
  }
}
```

The controller can handle optional parameters and apply default values when no value is supplied.

5.3. Handling Form Submission with Object Binding

❖ **Model:**

```
public class User {
  private String name;
  private String email;
  // Getters and Setters
}
```

❖ **Controller:**

```
@Controller
public class UserController {
  @GetMapping("/register")
  public String showForm(Model model) {
    model.addAttribute("user", new User());
    return "register";
  }

  @PostMapping("/register")
  public String submitForm(@ModelAttribute("user") User user, Model model) {
    model.addAttribute("message", "User registered successfully!");
    return "register-result";
  }
}
```

❖ **View (register.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>User Registration</title></head>
<body>
  <form action="register" method="post">
    Name: <input type="text" name="name" /><br>
    Email: <input type="email" name="email" /><br>
    <input type="submit" value="Register" />
  </form>
</body>
</html>
```

❖ **View (register-result.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Result</title></head>
<body>
  <p>${message}</p>
  <p>Name: ${user.name}</p>
  <p>Email: ${user.email}</p>
</body>
</html>
```

5.4. Redirect after post

❖ **Controller**

```
@Controller
public class RedirectExampleController {
  @PostMapping("/process")
  public  String  handlePost(@RequestParam  String  data,  RedirectAttributes
redirectAttributes) {
    redirectAttributes.addFlashAttribute("info", "Submitted: " + data);
    return "redirect:/confirmation";
  }

  @GetMapping("/confirmation")
  public String confirmation() {
    return "confirm";
  }
}
```

- **View (confirm.jsp):**

```
<!DOCTYPE html>
<html>
<head><title>Confirmation</title></head>
<body>
  <h3>${info}</h3>
</body>
</html>
```

The end

LAB 4. CONTROLLER AND MODEL IN SPRING MVC – ADVANCED

1. LEARNING OBJECTIVES

– Understand the role of Models and Controllers in Spring MVC.
– Create data models with 1-n, n-n relationships as they actually (Customer, Order, Product).
– Data processing with @ModelAttribute, @Valid, BindingResult.
– Present the data entry form and validate.

2. PROJECT CONFIGURATION

2.1. pom.xml

```
<dependencies>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>6.0.12</version>
   </dependency>

   <!-- Jakarta Servlet API (Tomcat 10) -->
   <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
   </dependency>

   <!-- Jakarta Validation API -->
   <dependency>
      <groupId>jakarta.validation</groupId>
      <artifactId>jakarta.validation-api</artifactId>
      <version>3.0.2</version>
   </dependency>

   <!-- JSTL cho Jakarta -->
   <dependency>
      <groupId>jakarta.servlet.jsp.jstl</groupId>
      <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
      <version>3.0.0</version>
   </dependency>
</dependencies>
```

2.2. web.xml (/WEB-INF/web.xml)

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

2.3. dispatcher-servlet.xml (/WEB-INF/dispatcher-servlet.xml)

```
<context:component-scan base-package="com.uef"/>
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

The Spring configuration will automatically scan for annotated classes such as @Controller, @Service in the **com.uef** package.

3. BUILDING MODELS (DATA LAYERS)

Relationship: Customer → many Orders, Order → many OrderDetails, OrderDetail → 1 Product.

3.1. Customer.java

```
package com.uef.model;
import jakarta.validation.constraints.*;

public class Customer {
    private int id;

    @NotBlank(message = "Name is required")
    private String name;

    @Email(message = "Invalid email")
```

```
private String email;
private List<Order> orders;

// --- Constructors ---

public Customer() {
}

public Customer(int id, String name, String email) {
    this.id = id;
    this.name = name;
    this.email = email;
}

// --- Getters & Setters ---

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 getEmail() {
    return email;
}

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

public List<Order> getOrders() {
    return orders;
}

public void setOrders(List<Order> orders) {
    this.orders = orders;
}
```

```
    // --- Optional: toString() ---

    @Override
    public String toString() {
        return "Customer{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

3.2. Product.java

```
package com.uef.model;

import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;

public class Product {

    private int id;

    @NotBlank(message = "Product name is required")
    private String name;

    @Min(value = 1000, message = "The product price must be greater than 1000")
    private double price;

    // --- Constructors ---

    public Product() {
    }

    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    // --- Getters & Setters ---

    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 double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    // --- Optional: toString() ---

    @Override
    public String toString() {
        return "Product{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", price=" + price +
                '}';
    }
}
```

3.3. Order.java

```
package com.uef.model;

import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;

import java.time.LocalDate;
import java.util.List;

public class Order {

    private int id;

    @NotNull(message = "Must choose a customer")
    private Customer customer;

    @NotNull(message = "The order date must not be empty")
    private LocalDate orderDate;

    @Size(min = 1, message = "The order must have at least 1 product")
    private List<OrderDetail> details;

    // --- Constructors ---

    public Order() {
    }

    public Order(int id, Customer customer, LocalDate orderDate, List<OrderDetail> details)
{
        this.id = id;
        this.customer = customer;
        this.orderDate = orderDate;
        this.details = details;
    }

    // --- Getters & Setters ---

    public int getId() {
        return id;
    }

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

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public LocalDate getOrderDate() {
        return orderDate;
    }

    public void setOrderDate(LocalDate orderDate) {
        this.orderDate = orderDate;
    }

    public List<OrderDetail> getDetails() {
        return details;
    }
    public void setDetails(List<OrderDetail> details) {
        this.details = details;
    }

    --- Business Logic: Calculate the total ---

    public double getTotalAmount() {
        if (details == null) return 0;
        return details.stream()
                .mapToDouble(d -> d.getProduct().getPrice() * d.getQuantity())
                .sum();
    }

    // --- Optional: toString() ---

    @Override
    public String toString() {
        return "Order{" +
                "id=" + id +
                ", customer=" + customer +
                ", orderDate=" + orderDate +
                ", details=" + details +
                '}';
    }
}
```

3.4. OrderDetail.java

An intermediate layer that represents the n–n relationship between Order and Product, including the quantity of each product in the order.

```
package com.uef.model;

import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotNull;

public class OrderDetail {

    private Order order; // Liên kết ngược (tùy chọn)

    @NotNull(message = "Sản phẩm không được để trống")
    private Product product;

    @Min(value = 1, message = "Số lượng phải từ 1 trở lên")
    private int quantity;

    // --- Constructors ---
    public OrderDetail() {}

    public OrderDetail(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }

    // --- Getters & Setters ---

    public Order getOrder() {
        return order;
    }

    public void setOrder(Order order) {
        this.order = order;
    }

    public Product getProduct() {
        return product;
    }

    public void setProduct(Product product) {
        this.product = product;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return "OrderDetail{" +
                "product=" + product +
                ", quantity=" + quantity +
                '}';
    }
}
```

4. SERVICE – LOGICAL PROCESSING

4.1. CustomerService.java

```
package com.uef.service;

import com.uef.model.Customer;
import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Collectors;

@Service
public class CustomerService {
   private final List<Customer> customerList = new ArrayList<>();
   private final AtomicInteger idGenerator = new AtomicInteger(1);

   Initializing Sample Data
   @PostConstruct
   public void init() {
      add(new Customer(0, "Nguyễn Văn A", "a@example.com"));
      add(new Customer(0, "Trần Thị B", "b@example.com"));
      add(new Customer(0, "Lê Minh C", "c@example.com"));
      add(new Customer(0, "Phạm Văn D", "d@example.com"));
   }

   Get All Clients
   public List<Customer> findAll() {
      return customerList;
   }

   Search by ID
   public Customer findById(int id) {
      return customerList.stream()
            .filter(c -> c.getId() == id)
            .findFirst()
            .orElse(null);
   }

   Add new customers
   public void add(Customer customer) {
      customer.setId(idGenerator.getAndIncrement());
      customerList.add(customer);
   }

   Update customer information
   public void update(Customer customer) {
      for (int i = 0; i < customerList.size(); i++) {
         if (customerList.get(i).getId() == customer.getId()) {
                customerList.set(i, customer);
                return;
            }
        }
    }

    // Xoá theo ID
    public void deleteById(int id) {
        customerList.removeIf(c -> c.getId() == id);
    }

    Search by name or email
    public List<Customer> search(String keyword) {
        if (keyword == null || keyword.isBlank()) return customerList;

        return customerList.stream()
                .filter(c -> c.getName().toLowerCase().contains(keyword.toLowerCase()) ||
                        c.getEmail().toLowerCase().contains(keyword.toLowerCase()))
                .collect(Collectors.toList());
    }

    Get the list by pagination
    public List<Customer> getPage(List<Customer> list, int page, int size) {
        int from = (page - 1) * size;
        int to = Math.min(from + size, list.size());
        if (from >= list.size()) return new ArrayList<>();
        return list.subList(from, to);
    }

    Calculate the number of pages needed
    public int countPages(List<Customer> list, int size) {
        return (int) Math.ceil((double) list.size() / size);
    }
}
```

4.2. ProductService.java

```
package com.example.service;

import com.example.model.Product;
import jakarta.annotation.PostConstruct;
import org.springframework.stereotype.Service;

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

@Service
```

```
public class ProductService {

    private final List<Product> productList = new ArrayList<>();

    private final AtomicInteger idGenerator = new AtomicInteger(1);

    Initialize sample data after starting the app
    @PostConstruct
    public void init() {
        add(new Product(0, "Laptop", 20000000));
        add(new Product(0, "Chuột không dây", 500000));
        add(new Product(0, "Bàn phím cơ", 800000));
    }

    /**
     * Return a list of all products
     */
    public List<Product> findAll() {
        return productList;
    }

    /**
     * Find products by ID
     * Product ID @param
     * Product @return or null if not found
     */
    public Product findById(int id) {
        return productList.stream()
                .filter(p -> p.getId() == id)
                .findFirst()
                .orElse(null);
    }

    /**
     * Add new products (automatic ID assignment)
     * @param products that need more
     */
    public void add(Product product) {
        product.setId(idGenerator.getAndIncrement());
        productList.add(product);
    }

    /**
     * Update product information by ID
     * @param product products need to be updated
     */
    public void update(Product product) {
        for (int i = 0; i < productList.size(); i++) {
            if (productList.get(i).getId() == product.getId()) {
                productList.set(i, product);
                return;
            }
        }
    }

    /**
     * Remove products from the list by ID
     * @param ID ID to be deleted
     */
    public void deleteById(int id) {
        productList.removeIf(p -> p.getId() == id);
    }

    /**
     * Search for products by keyword (name or 1 part of the name)
     * @param keywords
     * @return suitable product list
     */
    public List<Product> search(String keyword) {
        if (keyword == null || keyword.isBlank()) return productList;

        return productList.stream()
                .filter(p -> p.getName().toLowerCase().contains(keyword.toLowerCase()))
                .toList();
    }

    /**
     * Returns paginated listings
     * @param filtered product list list
     * @param current page
     * @param size of the number of lines per page
     */
    public List<Product> getPage(List<Product> list, int page, int size) {
        int from = (page - 1) * size;
        int to = Math.min(from + size, list.size());
        if (from >= list.size()) return new ArrayList<>();
        return list.subList(from, to);
    }

    /**
     * Calculate the total number of pages from the list and page size
     */
    public int countPages(List<Product> list, int size) {
        return (int) Math.ceil((double) list.size() / size);
    }
}
```

4.3. OrderService.java

```
package com.uef.service;

import com.uef.model.Customer;
import com.uef.model.Order;
import com.uef.model.OrderDetail;
import com. uef.model.Product;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.stream.Collectors;

@Service
public class OrderService {

   private final List<Order> orderList = new ArrayList<>();
   private final AtomicInteger idGenerator = new AtomicInteger(1);

   @Autowired
   private CustomerService customerService;

   @Autowired
   private ProductService productService;

   /**
    * Save orders: add new or update
    */
   public void save(Order order) {
      if (order.getId() == 0) {
         order.setId(idGenerator.getAndIncrement());
      } else {
         deleteById(order.getId());
      }

      //Full customer reassignment
      Customer fullCustomer = customerService.findById(order.getCustomer().getId());
      order.setCustomer(fullCustomer);

     // Reassign product information in each OrderDetail
      for (OrderDetail d : order.getDetails()) {
         Product p = productService.findById(d.getProduct().getId());
         d.setProduct(p);
         d.setOrder(order); // nếu OrderDetail có thuộc tính order
      }
```

```
        orderList.add(order);
    }

    /**
     * Trả về tất cả đơn hàng
     */
    public List<Order> findAll() {
        return orderList;
    }

    /**
     * Find orders by ID
     */
    public Order findById(int id) {
        return orderList.stream()
                .filter(o -> o.getId() == id)
                .findFirst()
                .orElse(null);
    }

    /**
     * Delete orders by ID
     */
    public void deleteById(int id) {
        orderList.removeIf(o -> o.getId() == id);
    }
    /**
     * Search for orders by customer name
     */
    public List<Order> search(String keyword) {
        if (keyword == null || keyword.isBlank()) return orderList;

        return orderList.stream()
                .filter(o                                                    ->
o.getCustomer().getName().toLowerCase().contains(keyword.toLowerCase()))
                .collect(Collectors.toList());
    }

    /**
     * Returns an order list for a specific page
     */
    public List<Order> getPage(List<Order> list, int page, int size) {
        int from = (page - 1) * size;
        int to = Math.min(from + size, list.size());
        if (from >= list.size()) return new ArrayList<>();
        return list.subList(from, to);
    }
    /**
     * Calculate the total number of pages
     */
    public int countPages(List<Order> list, int size) {
        return (int) Math.ceil((double) list.size() / size);
    }
}
```

5. CONTROLLER

5.1. CustomerController.java

```
@Controller
@RequestMapping("/customers")
public class CustomerController {

    @Autowired
    private CustomerService customerService;

    @GetMapping
    public String listCustomers(
            @RequestParam(name = "page", defaultValue = "1") int page,
            @RequestParam(name = "keyword", required = false) String keyword,
            Model model) {

        List<Customer> filtered = customerService.search(keyword);
        int size = 5;

        model.addAttribute("customers", customerService.getPage(filtered, page, size));
        model.addAttribute("totalPages", customerService.countPages(filtered, size));
        model.addAttribute("currentPage", page);
        model.addAttribute("keyword", keyword);

        return "customer-list";
    }

    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("customer", new Customer());
        return "customer-form";
    }

    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable int id, Model model) {
        Customer customer = customerService.findById(id);
        if (customer == null) return "redirect:/customers";
        model.addAttribute("customer", customer);
        return "customer-form";
    }

    @PostMapping("/save")
    public String saveCustomer(@ModelAttribute("customer") @Valid Customer customer,
BindingResult result) {
        if (result.hasErrors()) return "customer-form";

        if (customer.getId() == 0)
            customerService.add(customer);
        else
            customerService.update(customer);

        return "redirect:/customers";
    }

    @GetMapping("/delete/{id}")
    public String deleteCustomer(@PathVariable int id) {
        customerService.deleteById(id);
        return "redirect:/customers";
    }
}
```

5.2. ProductController.java

```
@Controller
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductService productService;

    @GetMapping
    public String listProducts(Model model) {
        model.addAttribute("products", productService.findAll());
        return "product-list";
    }

    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("product", new Product());
        return "product-form";
    }

    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable int id, Model model) {
        Product product = productService.findById(id);
        if (product == null) return "redirect:/products";
        model.addAttribute("product", product);
        return "product-form";
    }

    @PostMapping("/save")
    public String saveProduct(@ModelAttribute("product") @Valid Product product,
BindingResult result) {
        if (result.hasErrors()) return "product-form";

        if (product.getId() == 0)
            productService.add(product);
        else
            productService.update(product);

        return "redirect:/products";
    }

    @GetMapping("/delete/{id}")
    public String deleteProduct(@PathVariable int id) {
        productService.deleteById(id);
        return "redirect:/products";
    }
}
```

5.3. OrderController.java

```
@Controller
@RequestMapping("/orders")
public class OrderController {

    @Autowired
    private OrderService orderService;

    @Autowired
    private CustomerService customerService;

    @Autowired
    private ProductService productService;

    @GetMapping
    public String listOrders(
            @RequestParam(name = "page", defaultValue = "1") int page,
            @RequestParam(name = "keyword", required = false) String keyword,
            Model model) {

        List<Order> filtered = orderService.search(keyword);
        int size = 5;
        model.addAttribute("orders", orderService.getPage(filtered, page, size));
        model.addAttribute("totalPages", orderService.countPages(filtered, size));
        model.addAttribute("currentPage", page);
        model.addAttribute("keyword", keyword);

        return "order-list";
    }

    @GetMapping("/create")
    public String showCreateForm(Model model) {
        Order order = new Order();
        order.setOrderDate(LocalDate.now());

        model.addAttribute("order", order);
        model.addAttribute("customers", customerService.findAll());
        model.addAttribute("products", productService.findAll());

        return "order-form";
    }

    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable int id, Model model) {
        Order order = orderService.findById(id);
        if (order == null) return "redirect:/orders";

        model.addAttribute("order", order);
        model.addAttribute("customers", customerService.findAll());
        model.addAttribute("products", productService.findAll());

        return "order-form";
    }

    @PostMapping("/save")
    public String saveOrder(@ModelAttribute("order") @Valid Order order, BindingResult
result, Model model) {
        if (result.hasErrors()) {
            model.addAttribute("customers", customerService.findAll());
            model.addAttribute("products", productService.findAll());
            return "order-form";
        }

        orderService.save(order);
        return "redirect:/orders";
    }

    @GetMapping("/delete/{id}")
    public String deleteOrder(@PathVariable int id) {
        orderService.deleteById(id);
        return "redirect:/orders";
    }
}
```

6. VIEW

6.1. Customer

- Customer-form.jsp: Add or edit customer forms

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="c" uri="http://jakarta.tags.core" %>

<html>
<head><title>Khách hàng</title></head>
<body>
<h2><c:choose>
    <c:when test="${customer.id == 0}">Thêm khách hàng</c:when>
    <c:otherwise>Chỉnh sửa khách hàng</c:otherwise>
</c:choose></h2>

<form:form method="post" modelAttribute="customer"
action="${pageContext.request.contextPath}/customers/save">
    <form:hidden path="id"/>

    <p>
        <label>Tên:</label><br/>
        <form:input path="name"/>
        <form:errors path="name" cssClass="error"/>
    </p>

    <p>
        <label>Email:</label><br/>
        <form:input path="email"/>
        <form:errors path="email" cssClass="error"/>
    </p>

    <p><button type="submit">Lưu</button></p>
</form:form>
</body>
</html>
```

- customer-list.jsp

```
<%@ taglib prefix="c" uri="http://jakarta.tags.core" %>

<html>
```

```
<head><title>Danh sách khách hàng</title></head>
<body>
<h2>Danh sách khách hàng</h2>

<form method="get" action="${pageContext.request.contextPath}/customers">
   <input type="text" name="keyword" value="${keyword}" placeholder="Tìm kiếm..."/>
   <button type="submit">Tìm</button>
</form>

<a href="${pageContext.request.contextPath}/customers/add">Thêm khách hàng</a>

<table border="1" cellpadding="5">
   <tr><th>ID</th><th>Tên</th><th>Email</th><th>Thao tác</th></tr>
   <c:forEach var="c" items="${customers}">
      <tr>
         <td>${c.id}</td>
         <td>${c.name}</td>
         <td>${c.email}</td>
         <td>
            <a href="${pageContext.request.contextPath}/customers/edit/${c.id}">Sửa</a> |
            <a href="${pageContext.request.contextPath}/customers/delete/${c.id}"
              onclick="return confirm('Xoá khách hàng này?')">Xoá</a>
         </td>
      </tr>
   </c:forEach>
</table>

<c:if test="${totalPages > 1}">
   <p>Trang:
      <c:forEach begin="1" end="${totalPages}" var="p">
         <a href="?page=${p}&keyword=${keyword}">
            <c:choose>
               <c:when test="${p == currentPage}"><b>${p}</b></c:when>
               <c:otherwise>${p}</c:otherwise>
            </c:choose>
         </a>
      </c:forEach>
   </p>
</c:if>
</body>
</html>
```

6.2. Product

- product-form.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

```
<html>
<head><title>Sản phẩm</title></head>
<body>

<h2><c:choose>
   <c:when test="${product.id == 0}">Thêm sản phẩm</c:when>
   <c:otherwise>Chỉnh sửa sản phẩm</c:otherwise>
</c:choose></h2>

<form:form method="post" modelAttribute="product"
action="${pageContext.request.contextPath}/products/save">
   <form:hidden path="id"/>

   <p>
      <label>Tên sản phẩm:</label><br/>
      <form:input path="name"/>
      <form:errors path="name" cssClass="error"/>
   </p>

   <p>
      <label>Giá:</label><br/>
      <form:input path="price"/>
      <form:errors path="price" cssClass="error"/>
   </p>

   <p><button type="submit">Lưu</button></p>
</form:form>
</body>
</html>
```

- ❖ product-list.jsp

```
<%@ taglib prefix="c" uri="http://jakarta.tags.core" %>

<html>
<head><title>Danh sách sản phẩm</title></head>
<body>
<h2>Danh sách sản phẩm</h2>

<a href="${pageContext.request.contextPath}/products/add">Thêm sản phẩm</a>

<table border="1" cellpadding="5">
   <tr><th>ID</th><th>Tên</th><th>Giá</th><th>Thao tác</th></tr>
   <c:forEach var="p" items="${products}">
      <tr>
         <td>${p.id}</td>
         <td>${p.name}</td>
         <td>${p.price}</td>
```

```
            <td>
                <a href="${pageContext.request.contextPath}/products/edit/${p.id}">Sửa</a> |
                <a href="${pageContext.request.contextPath}/products/delete/${p.id}"
                   onclick="return confirm('Xoá sản phẩm này?')">Xoá</a>
            </td>
        </tr>
    </c:forEach>
</table>
</body>
</html>
```

6.3. Order

- order-form.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="c" uri="http://jakarta.tags.core" %>

<html>
<head><title>Đơn hàng</title></head>
<body>

<h2><c:choose>
    <c:when test="${order.id == 0}">Tạo đơn hàng</c:when>
    <c:otherwise>Chỉnh sửa đơn hàng</c:otherwise>
</c:choose></h2>

<form:form method="post" modelAttribute="order"
action="${pageContext.request.contextPath}/orders/save">
    <form:hidden path="id"/>

    <p>
        <label>Ngày đặt:</label><br/>
        <form:input path="orderDate" type="date"/>
        <form:errors path="orderDate" cssClass="error"/>
    </p>

    <p>
        <label>Khách hàng:</label><br/>
        <form:select path="customer.id">
            <form:options items="${customers}" itemValue="id" itemLabel="name"/>
        </form:select>
    </p>

    <h3>Danh sách sản phẩm:</h3>
    <table border="1">
        <tr><th>Sản phẩm</th><th>Số lượng</th></tr>
        <c:forEach var="product" items="${products}" varStatus="i">
```

```
         <tr>
            <td>
               ${product.name} - ${product.price}
               <input type="hidden" name="details[${i.index}].product.id"
value="${product.id}"/>
            </td>
            <td>
               <input type="number" name="details[${i.index}].quantity" value="0"
min="0"/>
            </td>
         </tr>
      </c:forEach>
   </table>

   <p><button type="submit">Lưu đơn hàng</button></p>
</form:form>

</body>
</html>
```

❖ order-list.jsp

```
<%@ taglib prefix="c" uri="http://jakarta.tags.core" %>

<html>
<head><title>Danh sách đơn hàng</title></head>
<body>
<h2>Danh sách đơn hàng</h2>

<form method="get" action="${pageContext.request.contextPath}/orders">
   <input type="text" name="keyword" value="${keyword}" placeholder="Tìm theo tên
khách hàng"/>
   <button type="submit">Tìm kiếm</button>
</form>

<a href="${pageContext.request.contextPath}/orders/create">Tạo đơn hàng mới</a>

<table border="1" cellpadding="5">
   <tr>
      <th>ID</th><th>Khách hàng</th><th>Ngày</th><th>Tổng tiền</th><th>Thao
tác</th>
   </tr>
   <c:forEach var="o" items="${orders}">
      <tr>
         <td>${o.id}</td>
         <td>${o.customer.name}</td>
         <td>${o.orderDate}</td>
         <td>${o.totalAmount}</td>
```

```
            <td>
                <a href="${pageContext.request.contextPath}/orders/edit/${o.id}">Sửa</a> |
                <a href="${pageContext.request.contextPath}/orders/delete/${o.id}"
                   onclick="return confirm('Bạn có chắc muốn xoá không?')">Xoá</a>
            </td>
        </tr>
    </c:forEach>
</table>

<c:if test="${totalPages > 1}">
    <p>Trang:
        <c:forEach begin="1" end="${totalPages}" var="p">
            <a href="?page=${p}&keyword=${keyword}">
                <c:choose>
                    <c:when test="${p == currentPage}"><b>${p}</b></c:when>
                    <c:otherwise>${p}</c:otherwise>
                </c:choose>
            </a>
        </c:forEach>
    </p>
</c:if>

</body>
</html>
```

7. PROJECT STRUCTURE

7.1. Functional structure

Ingredient	**Main Description**
Customer	Client management (add, edit, delete, search, paginate)
Product	Product Management (Full CRUD)
Order	Order management (product + customer selection, total payment, pagination)

7.2. Components of the app

- ❖ Model:

  Customer.java, Product.java, Order.java, OrderDetail.java

- Service:

  CustomerService.java, ProductService.java, OrderService.java

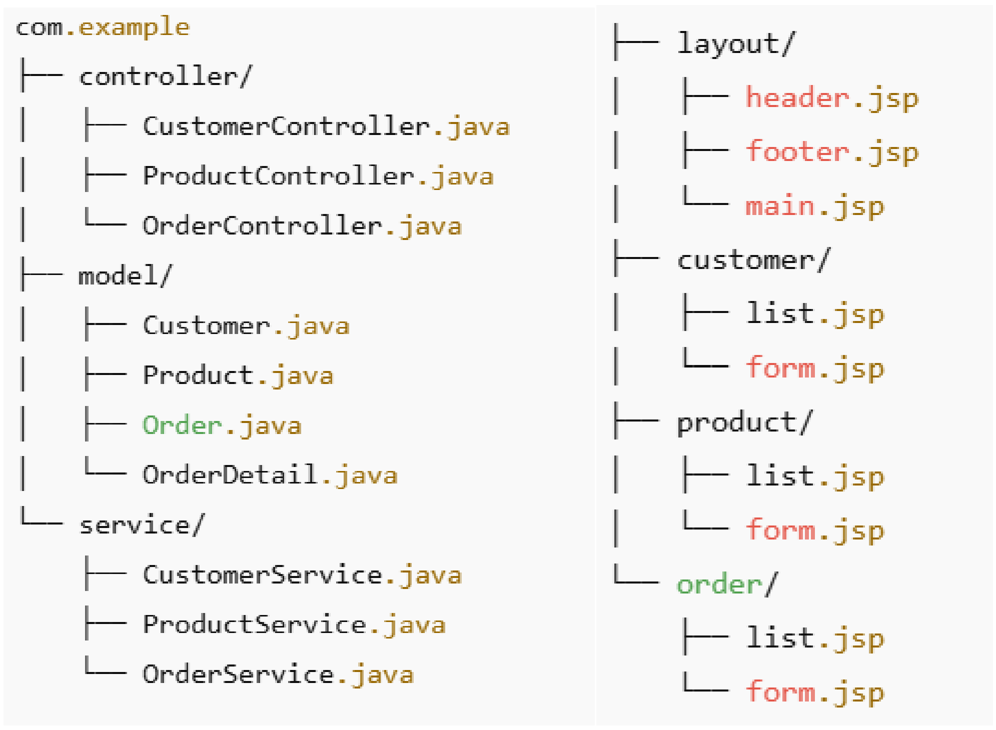
- Controller: CustomerController.java, ProductController.java, OrderController.java
- JSP View: customer-form.jsp, customer-list.jsp; product-form.jsp, product-list.jsp; order-form.jsp, order-list.jsp

LAB 5. WORKING WITH VIEW IN SPRING MVC

1. LEARNING OBJECTIVES

- Understand the interaction between Model, Controller, and View in Spring MVC
- Build a complete Order Management System with Customer, Product, and Order modules
- Separate layout using reusable JSP includes: header.jsp, footer.jsp, main.jsp
- Implement full CRUD, pagination, and search features

2. PROJECT STRUCTURE



```
com.example
├── controller/
│   ├── CustomerController.java
│   ├── ProductController.java
│   └── OrderController.java
├── model/
│   ├── Customer.java
│   ├── Product.java
│   ├── Order.java
│   └── OrderDetail.java
└── service/
    ├── CustomerService.java
    ├── ProductService.java
    └── OrderService.java
```

```
├── layout/
│   ├── header.jsp
│   ├── footer.jsp
│   └── main.jsp
├── customer/
│   ├── list.jsp
│   └── form.jsp
├── product/
│   ├── list.jsp
│   └── form.jsp
└── order/
    ├── list.jsp
    └── form.jsp
```

3. MODEL

Students inherit the Model component in Lab 4.

4. CONTROLLER

Students inherit the Controller component in Lab 4 and updates it as follows:

4.1. CustomerController.java

```
@Controller
@RequestMapping("/customers")
public class CustomerController {

   @Autowired private CustomerService customerService;

   @GetMapping
   public String list(Model model) {
      model.addAttribute("customers", customerService.findAll());
      model.addAttribute("body", "/WEB-INF/views/customer/list.jsp");
      return "layout/main";
   }

   @GetMapping("/add")
   public String add(Model model) {
      model.addAttribute("customer", new Customer());
      model.addAttribute("body", "/WEB-INF/views/customer/form.jsp");
      return "layout/main";
   }

   @PostMapping("/save")
   public String save(@ModelAttribute Customer customer) {
      if (customer.getId() == 0)
         customerService.add(customer);
      else
         customerService.update(customer);
      return "redirect:/customers";
   }

   @GetMapping("/edit/{id}")
   public String edit(@PathVariable int id, Model model) {
      model.addAttribute("customer", customerService.findById(id));
      model.addAttribute("body", "/WEB-INF/views/customer/form.jsp");
      return "layout/main";
   }

   @GetMapping("/delete/{id}")
   public String delete(@PathVariable int id) {
      customerService.deleteById(id);
      return "redirect:/customers";
   }
}
```

The student updates the methods of the remaining Controllers.

5. VIEW

❖ **header.jsp in the layout directory.**

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
   <meta charset="UTF-8">
   <title>Spring MVC Order App</title>
   <style>
      body { font-family: sans-serif; margin: 20px; }
      nav a { margin-right: 10px; }
   </style>
</head>
<body>
<nav>
   <a href="${pageContext.request.contextPath}/customers">Customers</a>
   <a href="${pageContext.request.contextPath}/products">Products</a>
   <a href="${pageContext.request.contextPath}/orders">Orders</a>
</nav>
<hr/>
```

- **taglib c:** declares JSTL core tag library.
- **${pageContext.request.contextPath}:** dynamically sets the correct context path (useful when deployed under a subfolder).
- **<nav>:** creates a simple navigation bar.
- This file is included in every page as the common header layout.

❖ **footer.jsp in the layout directory.**

```
<hr/>
<footer>
   <p>&copy; 2025 - Powered by Spring MVC + JSP</p>
</footer>
</body>
</html>
```

- Marks the end of the content section.
- Provides a consistent footer for all pages.
- Included at the end of every layout through **main.jsp.**

❖ **main.jsp in the layout directory.**

```
<jsp:include page="/WEB-INF/views/layout/header.jsp"/>
<jsp:include page="${body}"/>
<jsp:include page="/WEB-INF/views/layout/footer.jsp"/>
```

– Acts as the main layout wrapper for the entire application.
– ${body} is dynamically set in the controller to include content pages like list.jsp or form.jsp.
– This pattern enables modular layout reuse (header + footer + dynamic content).

❖ **list.jsp in the customer directory.**

```
<h2>Customer List</h2>
<a href="${pageContext.request.contextPath}/customers/add">Add</a>

<table border="1">
   <tr><th>ID</th><th>Name</th><th>Email</th><th>Actions</th></tr>
   <c:forEach var="c" items="${customers}">
      <tr>
         <td>${c.id}</td>
         <td>${c.name}</td>
         <td>${c.email}</td>
         <td>
            <a href="${pageContext.request.contextPath}/customers/edit/${c.id}">Edit</a> |
            <a
href="${pageContext.request.contextPath}/customers/delete/${c.id}">Delete</a>
         </td>
      </tr>
   </c:forEach>
</table>
```

– **c:forEach:** iterates over the customers list provided by the controller.
– **${c.id}, ${c.name}:** output data for each customer.
– Action buttons link to the edit and delete endpoints.
– The list is embedded as content into the layout.

❖ **form**.jsp in the customer directory.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<h2><c:choose>
   <c:when test="${customer.id == 0}">Add Customer</c:when>
   <c:otherwise>Edit Customer</c:otherwise>
</c:choose></h2>
```

```
<form:form method="post" modelAttribute="customer"
action="${pageContext.request.contextPath}/customers/save">
  <form:hidden path="id"/>

  <p>
    <label>Name:</label><br/>
    <form:input path="name"/>
    <form:errors path="name" cssClass="error"/>
  </p>

  <p>
    <label>Email:</label><br/>
    <form:input path="email"/>
    <form:errors path="email" cssClass="error"/>
  </p>

  <p><button type="submit">Save</button></p>
</form:form>
```

- Uses Spring’s form:form tag to bind form data to a model object (customer).
- form:input maps to individual fields like name, email.
- form:errors displays validation errors for each field.
- form:hidden path="id" keeps the ID during updates.

❖ **Summary**

File	Purpose
`layout/header.jsp`	Shared navigation and page header
`layout/footer.jsp`	Common page footer
`layout/main.jsp`	Main wrapper that includes header + body + footer
`*.list.jsp`	Display data in a table format
`*.form.jsp`	Display form inputs using Spring’s form taglib
`${body}`	Dynamic content that is included per request

The student performs similar updates for Product and Order. Then, run the program to view the results.

LAB 6. WORKING WITH DATABASE IN SPRING MVC

1. LEARNING OBJECTIVES

- Configure the Dispatcher Servlet using web.xml and Java-based configuration, and set up a connection to MS SQL Server using JdbcTemplate and DriverManagerDataSource.
- Organize a real-world project structure with clear separation of concerns across controller, model, service, repository, and config layers.
- Perform CRUD operations on the Apartments table and implement business logic through service classes, ensuring proper separation between Controller, Service, and Repository.
- Design web views using JSP with dynamic content rendered via JSTL tags such as <c:forEach> and <c:if>.
- Apply Spring MVC to simulate a practical apartment management system and extend it with modules for household management, complaints, parking, and reporting.

2. PROJECT CONFIGURATION

2.1. pom.xml

```
<dependencies>
   <dependency>
      <groupId>jakarta.platform</groupId>
      <artifactId>jakarta.jakartaee-api</artifactId>
      <version>10.0.0</version>
      <scope>provided</scope>
   </dependency>
   <!-- Spring Web MVC -->
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>6.0.11</version>
   </dependency>
   <dependency>
      <groupId>org.hibernate.validator</groupId>
      <artifactId>hibernate-validator</artifactId>
      <version>8.0.0.Final</version>
   </dependency>
```

```
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>4.0.1</version>
                <scope>provided</scope>
         </dependency>
        <dependency>
           <groupId>jakarta.servlet.jsp.jstl</groupId>
           <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
           <version>3.0.1</version>
        </dependency>
        <dependency>
           <groupId>org.glassfish.web</groupId>
           <artifactId>jakarta.servlet.jsp.jstl</artifactId>
           <version>3.0.0</version>
        </dependency>
        <!-- JDBC driver for SQL Server -->
        <dependency>
           <groupId>com.microsoft.sqlserver</groupId>
           <artifactId>mssql-jdbc</artifactId>
           <version>12.2.0.jre11</version> <!-- Hoặc jre8 tuỳ vào JDK -->
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
        <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-jdbc</artifactId>
           <version>6.2.5</version>
        </dependency>
</dependencies>
```

2.2. Servlet Dispatcher Configuration - web.xml

```
<web-app xmlns="http://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://jakarta.ee/xml/ns/jakartaee
                     http://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
         version="6.0">
    <display-name>Apartment Management</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>
```

2.3. Spring MVC Configuration - dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       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/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">

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

    <!-- Enable @Controller, @RequestMapping, etc. -->
    <mvc:annotation-driven/>

    <!-- View Resolver -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- SQL Server DataSource -->
        <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"
value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
        <property name="url"
value="jdbc:sqlserver://localhost:1433;databaseName=YourDB;encrypt=false " />
        <property name="username" value="sa" />
        <property name="password" value="your_password" />
    </bean>

    <!-- JdbcTemplate Bean -->
    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
```

```
        <property name="dataSource" ref="dataSource"/>
    </bean>
</beans>
```

3. MODEL

3.1. Apartment.java

```
package com.uef.model;
import jakarta.validation.constraints.*;

public class Apartment {

    private int apartmentID;

    @NotBlank(message = "Apartment number is required")
    @Size(max = 10, message = "Max 10 characters allowed")
    private String apartmentNumber;

    @Min(value = 1, message = "Floor must be greater than 0")
    private int floor;

    @DecimalMin(value = "10.0", message = "Area must be at least 10 m²")
    private double area;

    @NotBlank(message = "Status is required")
    private String status;

    // Constructors
    public Apartment() {}
    public Apartment(int apartmentID, String apartmentNumber, int floor, double area, String
status) {
        this.apartmentID = apartmentID;
        this.apartmentNumber = apartmentNumber;
        this.floor = floor;
        this.area = area;
        this.status = status;
    }
    // Getters and Setters - Students continue to practice

}
```

3.2. Household.java

```
package com.uef.model;
import jakarta.validation.constraints.*;

public class Household {
    private int householdID;

    @Min(value = 1, message = "Apartment ID must be selected")
    private int apartmentID;

    @NotBlank(message = "Head of household name is required")
    @Size(max = 100)
    private String headOfHousehold;

    @Pattern(regexp = "\\d{10,15}", message = "Phone number must be 10–15 digits")
    private String contactNumber;

    @Email(message = "Invalid email format")
    private String email;

    // Constructors
    public Household() {}

    public Household(int householdID, int apartmentID, String headOfHousehold, String
contactNumber, String email) {
        this.householdID = householdID;
        this.apartmentID = apartmentID;
        this.headOfHousehold = headOfHousehold;
        this.contactNumber = contactNumber;
        this.email = email;
    }

    // Getters and Setters - Students continue to practice
}
```

3.3. Resident.java

```
package com.uef.model;
import jakarta.validation.constraints.*;
import org.springframework.format.annotation.DateTimeFormat;
import java.time.LocalDate;

public class Resident {
    private int residentID;

    @Min(value = 1, message = "Household ID must be valid")
    private int householdID;

    @NotBlank(message = "Full name is required")
    private String fullName;

    @NotNull(message = "Date of birth is required")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate dateOfBirth;

    @NotBlank(message = "Gender is required")
    private String gender;

    @NotBlank(message = "Relationship is required")
    private String relationship;

    // Constructors
    public Resident() {}

    public Resident(int residentID, int householdID, String fullName, LocalDate dateOfBirth,
String gender, String relationship) {
        this.residentID = residentID;
        this.householdID = householdID;
        this.fullName = fullName;
        this.dateOfBirth = dateOfBirth;
        this.gender = gender;
        this.relationship = relationship;
    }

    // Getters and Setters - Students continue to practice
}
```

3.4. Fee.java

```
package com.uef.model;
import jakarta.validation.constraints.*;
import org.springframework.format.annotation.DateTimeFormat;
import java.time.LocalDate;

public class Fee {
    private int feeID;

    @Min(value = 1, message = "Household ID is required")
    private int householdID;

    @NotBlank(message = "Fee type is required")
    private String feeType;

    @DecimalMin(value = "0.01", message = "Amount must be greater than 0")
    private double amount;

    @NotNull(message = "Due date is required")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate dueDate;

    @NotBlank(message = "Status is required")
    @Pattern(regexp = "Paid|Unpaid", message = "Status must be Paid or Unpaid")
    private String status;

    // Constructors
    public Fee() {}

    public Fee(int feeID, int householdID, String feeType, double amount, LocalDate dueDate,
String status) {
        this.feeID = feeID;
        this.householdID = householdID;
        this.feeType = feeType;
        this.amount = amount;
        this.dueDate = dueDate;
        this.status = status;
    }
    // Getters and Setters - Students continue to practice
}
```

3.5. Parking.java

```
package com.uef.model;

import jakarta.validation.constraints.*;

public class Parking {
    private int parkingID;

    private int householdID;

    @NotBlank(message = "Parking number is required")
    @Size(max = 10)
    private String parkingNumber;

    @NotBlank(message = "Vehicle type is required")
    @Pattern(regexp = "Car|Motorbike", message = "Must be Car or Motorbike")
    private String vehicleType;

    @NotBlank(message = "Status is required")
    @Pattern(regexp = "Occupied|Vacant", message = "Status must be Occupied or Vacant")
    private String status;

    // Constructors
    public Parking() {}

    public Parking(int parkingID, Integer householdID, String parkingNumber, String
vehicleType, String status) {
        this.parkingID = parkingID;
        this.householdID = householdID;
        this.parkingNumber = parkingNumber;
        this.vehicleType = vehicleType;
        this.status = status;
    }
    // Getters and Setters - Students continue to practice
}
```

3.6. Complaint.java

```
package com.uef.model;

import jakarta.validation.constraints.*;
import org.springframework.format.annotation.DateTimeFormat;
import java.time.LocalDate;

public class Complaint {

    private int complaintID;

    @Min(value = 1, message = "Household ID is required")
    private int householdID;

    @NotBlank(message = "Description is required")
    @Size(min = 10, max = 1000, message = "Description must be 10–1000 characters")
    private String description;

    @NotNull(message = "Submission date is required")
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private LocalDate submissionDate;

    @NotBlank(message = "Status is required")
    private String status;

    // Constructors
    public Complaint() {
    }

    public Complaint(int complaintID, int householdID, String description, LocalDate
submissionDate, String status) {
        this.complaintID = complaintID;
        this.householdID = householdID;
        this.description = description;
        this.submissionDate = submissionDate;
        this.status = status;
    }

    // Getters and Setters - Students continue to practice
}
```

4. REPOSITORY

4.1. ApartmentRepository.java

```
package com.uef.repository;
import com.uef.model.Apartment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class ApartmentRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    // Mapping ResultSet to Apartment object
    private Apartment mapRow(ResultSet rs, int rowNum) throws SQLException {
        Apartment apartment = new Apartment();
        apartment.setApartmentID(rs.getInt("ApartmentID"));
        apartment.setApartmentNumber(rs.getString("ApartmentNumber"));
        apartment.setFloor(rs.getInt("Floor"));
        apartment.setArea(rs.getDouble("Area"));
        apartment.setStatus(rs.getString("Status"));
        return apartment;
    }

    // SELECT * FROM Apartments
    public List<Apartment> findAll() {
        String sql = "SELECT * FROM Apartments";
        return jdbcTemplate.query(sql, this::mapRow);
    }

    // SELECT * FROM Apartments WHERE ApartmentID = ?
    public Apartment findById(int id) {
        String sql = "SELECT * FROM Apartments WHERE ApartmentID = ?";
        return jdbcTemplate.queryForObject(sql, this::mapRow, id);
    }

    // INSERT INTO Apartments (...)
    public void save(Apartment apartment) {
        String sql = "INSERT INTO Apartments (ApartmentNumber, Floor, Area, Status)
VALUES (?, ?, ?, ?)";
```

```
        jdbcTemplate.update(sql,
                apartment.getApartmentNumber(),
                apartment.getFloor(),
                apartment.getArea(),
                apartment.getStatus());
    }
    // UPDATE Apartments SET ... WHERE ApartmentID = ?
    public void update(Apartment apartment) {
        String sql = "UPDATE Apartments SET ApartmentNumber = ?, Floor = ?, Area = ?, 
Status = ? WHERE ApartmentID = ?";
        jdbcTemplate.update(sql,
                apartment.getApartmentNumber(),
                apartment.getFloor(),
                apartment.getArea(),
                apartment.getStatus(),
                apartment.getApartmentID());
    }

    // DELETE FROM Apartments WHERE ApartmentID = ?
    public void delete(int id) {
        String sql = "DELETE FROM Apartments WHERE ApartmentID = ?";
        jdbcTemplate.update(sql, id);
    }
}
```

4.2. HouseholdRepository.java

```
package com.uef.repository;

import com.uef.model.Resident;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class ResidentRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    private Resident mapRow(ResultSet rs, int rowNum) throws SQLException {
        Resident r = new Resident();
        r.setResidentID(rs.getInt("ResidentID"));
        r.setHouseholdID(rs.getInt("HouseholdID"));
        r.setFullName(rs.getString("FullName"));
        r.setDateOfBirth(rs.getDate("DateOfBirth").toLocalDate());
        r.setGender(rs.getString("Gender"));
        r.setRelationship(rs.getString("Relationship"));
        return r;
    }

    public List<Resident> findAll() {
        return jdbcTemplate.query("SELECT * FROM Residents", this::mapRow);
    }

    public Resident findById(int id) {
        return jdbcTemplate.queryForObject("SELECT * FROM Residents WHERE
ResidentID=?", this::mapRow, id);
    }

    public void save(Resident r) {
        String sql = "INSERT INTO Residents (HouseholdID, FullName, DateOfBirth, Gender,
Relationship) VALUES (?, ?, ?, ?, ?)";
        jdbcTemplate.update(sql, r.getHouseholdID(), r.getFullName(), r.getDateOfBirth(),
r.getGender(), r.getRelationship());
    }

    public void update(Resident r) {
        String sql = "UPDATE Residents SET HouseholdID=?, FullName=?, DateOfBirth=?,
Gender=?, Relationship=? WHERE ResidentID=?";
        jdbcTemplate.update(sql, r.getHouseholdID(), r.getFullName(), r.getDateOfBirth(),
r.getGender(), r.getRelationship(), r.getResidentID());
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE FROM Residents WHERE ResidentID=?", id);
    }
}
```

4.3. ResidentRepository.java

```
package com.uef.repository;

import com.uef.model.Resident;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
```

```
@Repository
public class ResidentRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    private Resident mapRow(ResultSet rs, int rowNum) throws SQLException {
        Resident r = new Resident();
        r.setResidentID(rs.getInt("ResidentID"));
        r.setHouseholdID(rs.getInt("HouseholdID"));
        r.setFullName(rs.getString("FullName"));
        r.setDateOfBirth(rs.getDate("DateOfBirth").toLocalDate());
        r.setGender(rs.getString("Gender"));
        r.setRelationship(rs.getString("Relationship"));
        return r;
    }

    public List<Resident> findAll() {
        return jdbcTemplate.query("SELECT * FROM Residents", this::mapRow);
    }

    public Resident findById(int id) {
        return jdbcTemplate.queryForObject("SELECT * FROM Residents WHERE
ResidentID=?", this::mapRow, id);
    }

    public void save(Resident r) {
        String sql = "INSERT INTO Residents (HouseholdID, FullName, DateOfBirth, Gender,
Relationship) VALUES (?, ?, ?, ?, ?)";
        jdbcTemplate.update(sql, r.getHouseholdID(), r.getFullName(), r.getDateOfBirth(),
r.getGender(), r.getRelationship());
    }

    public void update(Resident r) {
        String sql = "UPDATE Residents SET HouseholdID=?, FullName=?, DateOfBirth=?,
Gender=?, Relationship=? WHERE ResidentID=?";
        jdbcTemplate.update(sql, r.getHouseholdID(), r.getFullName(), r.getDateOfBirth(),
r.getGender(), r.getRelationship(), r.getResidentID());
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE FROM Residents WHERE ResidentID=?", id);
    }
}
```

4.4. FeeRepository.java

```
package com.uef.repository;
```

```
import com.uef.model.Fee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class FeeRepository {

   @Autowired
   private JdbcTemplate jdbcTemplate;

   private Fee mapRow(ResultSet rs, int rowNum) throws SQLException {
      Fee f = new Fee();
      f.setFeeID(rs.getInt("FeeID"));
      f.setHouseholdID(rs.getInt("HouseholdID"));
      f.setFeeType(rs.getString("FeeType"));
      f.setAmount(rs.getDouble("Amount"));
      f.setDueDate(rs.getDate("DueDate").toLocalDate());
      f.setStatus(rs.getString("Status"));
      return f;
   }


   public List<Fee> findAll() {
      return jdbcTemplate.query("SELECT * FROM Fees", this::mapRow);
   }

   public Fee findById(int id) {
      return jdbcTemplate.queryForObject("SELECT * FROM Fees WHERE FeeID=?",
this::mapRow, id);
   }

   public void save(Fee f) {
      String sql = "INSERT INTO Fees (HouseholdID, FeeType, Amount, DueDate, Status)
VALUES (?, ?, ?, ?, ?)";
      jdbcTemplate.update(sql, f.getHouseholdID(), f.getFeeType(), f.getAmount(),
f.getDueDate(), f.getStatus());
   }

   public void update(Fee f) {
      String sql = "UPDATE Fees SET HouseholdID=?, FeeType=?, Amount=?, DueDate=?,
Status=? WHERE FeeID=?";
      jdbcTemplate.update(sql, f.getHouseholdID(), f.getFeeType(), f.getAmount(),
f.getDueDate(), f.getStatus(), f.getFeeID());
```

```
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE FROM Fees WHERE FeeID=?", id);
    }
}
```

4.5. ParkingRepository.java

```
package com.uef.repository;

import com.uef.model.Parking;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class ParkingRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;


    private Parking mapRow(ResultSet rs, int rowNum) throws SQLException {
        Parking p = new Parking();
        p.setParkingID(rs.getInt("ParkingID"));
        p.setHouseholdID((Integer) rs.getObject("HouseholdID")); // Nullable
        p.setParkingNumber(rs.getString("ParkingNumber"));
        p.setVehicleType(rs.getString("VehicleType"));
        p.setStatus(rs.getString("Status"));
        return p;
    }

    public List<Parking> findAll() {
        return jdbcTemplate.query("SELECT * FROM Parking", this::mapRow);
    }

    public Parking findById(int id) {
        return jdbcTemplate.queryForObject("SELECT * FROM Parking WHERE
ParkingID=?", this::mapRow, id);
    }

    public void save(Parking p) {
        String sql = "INSERT INTO Parking (HouseholdID, ParkingNumber, VehicleType, Status) VALUES (?, ?, ?, ?)";
        jdbcTemplate.update(sql, p.getHouseholdID(), p.getParkingNumber(), p.getVehicleType(), p.getStatus());
    }

    public void update(Parking p) {
        String sql = "UPDATE Parking SET HouseholdID=?, ParkingNumber=?, VehicleType=?, Status=? WHERE ParkingID=?";
        jdbcTemplate.update(sql, p.getHouseholdID(), p.getParkingNumber(), p.getVehicleType(), p.getStatus(), p.getParkingID());
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE FROM Parking WHERE ParkingID=?", id);
    }
}
```

4.6. ComplaintRepository.java

```
package com.uef.repository;

import com.uef.model.Complaint;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class ComplaintRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    private Complaint mapRow(ResultSet rs, int rowNum) throws SQLException {
        Complaint c = new Complaint();
        c.setComplaintID(rs.getInt("ComplaintID"));
        c.setHouseholdID(rs.getInt("HouseholdID"));
        c.setDescription(rs.getString("Description"));
        c.setSubmissionDate(rs.getDate("SubmissionDate").toLocalDate());
        c.setStatus(rs.getString("Status"));
        return c;
    }

    public List<Complaint> findAll() {
        return jdbcTemplate.query("SELECT * FROM Complaints", this::mapRow);
    }

    public Complaint findById(int id) {
        return jdbcTemplate.queryForObject("SELECT * FROM Complaints WHERE ComplaintID=?", this::mapRow, id);
    }

    public void save(Complaint c) {
        String sql = "INSERT INTO Complaints (HouseholdID, Description, SubmissionDate, Status) VALUES (?, ?, ?, ?)";
        jdbcTemplate.update(sql, c.getHouseholdID(), c.getDescription(), c.getSubmissionDate(), c.getStatus());
    }

    public void update(Complaint c) {
        String sql = "UPDATE Complaints SET HouseholdID=?, Description=?, SubmissionDate=?, Status=? WHERE ComplaintID=?";
        jdbcTemplate.update(sql, c.getHouseholdID(), c.getDescription(), c.getSubmissionDate(), c.getStatus(), c.getComplaintID());
    }

    public void delete(int id) {
        jdbcTemplate.update("DELETE FROM Complaints WHERE ComplaintID=?", id);
    }
}
```

5. SERVICE

5.1. ApartmentService.java

```
package com.uef.service;

import com.uef.model.Apartment;
import com.uef.repository.ApartmentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ApartmentService {

    @Autowired
    private ApartmentRepository apartmentRepository;

    public List<Apartment> getAll() {
        return apartmentRepository.findAll();
    }

    public Apartment getById(int id) {
        return apartmentRepository.findById(id);
    }

    public void add(Apartment apartment) {
        apartmentRepository.save(apartment);
    }

    public void update(Apartment apartment) {
        apartmentRepository.update(apartment);
    }

    public void delete(int id) {
        apartmentRepository.delete(id);
    }
}
```

5.2. HouseholdService.java

```
package com.uef.service;

import com.uef.model.Household;
import com.example.repository.HouseholdRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class HouseholdService {

    @Autowired
    private HouseholdRepository householdRepository;

    public List<Household> getAll() {
        return householdRepository.findAll();
    }

    public Household getById(int id) {
        return householdRepository.findById(id);
    }

    public void add(Household h) {
        householdRepository.save(h);
    }

    public void update(Household h) {
        householdRepository.update(h);
    }

    public void delete(int id) {
        householdRepository.delete(id);
    }
}
```

5.3. ResidentService.java

```
package com.example.service;

import com.uef.model.Resident;
import com.example.repository.ResidentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ResidentService {

    @Autowired
    private ResidentRepository residentRepository;

    public List<Resident> getAll() {
        return residentRepository.findAll();
    }

    public Resident getById(int id) {
        return residentRepository.findById(id);
    }

    public void add(Resident r) {
        residentRepository.save(r);
    }

    public void update(Resident r) {
        residentRepository.update(r);
    }

    public void delete(int id) {
        residentRepository.delete(id);
    }
}
```

5.4. FeeService.java

```
package com.uef.service;

import com.uef.model.Fee;
import com.uef.repository.FeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class FeeService {

    @Autowired
    private FeeRepository feeRepository;

    public List<Fee> getAll() {
        return feeRepository.findAll();
    }

    public Fee getById(int id) {
        return feeRepository.findById(id);
    }

    public void add(Fee f) {
        feeRepository.save(f);
    }

    public void update(Fee f) {
        feeRepository.update(f);
    }

    public void delete(int id) {
        feeRepository.delete(id);
    }
}
```

5.5. ParkingService.java

```
package com.uef.service;

import com.uef.model.Parking;
import com.uef.repository.ParkingRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ParkingService {

    @Autowired
    private ParkingRepository parkingRepository;

    public List<Parking> getAll() {
        return parkingRepository.findAll();
    }

    public Parking getById(int id) {
        return parkingRepository.findById(id);
    }

    public void add(Parking p) {
        parkingRepository.save(p);
    }

    public void update(Parking p) {
        parkingRepository.update(p);
    }

    public void delete(int id) {
        parkingRepository.delete(id);
    }
}
```

5.6. ComplaintService.java

```
package com.uef.service;
import com.uef.model.Complaint;
import com.uef.repository.ComplaintRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ComplaintService {

    @Autowired
    private ComplaintRepository complaintRepository;

    public List<Complaint> getAll() {
        return complaintRepository.findAll();
    }
    public Complaint getById(int id) {
        return complaintRepository.findById(id);
    }

    public void add(Complaint c) {
        complaintRepository.save(c);
    }

    public void update(Complaint c) {
        complaintRepository.update(c);
    }

    public void delete(int id) {
        complaintRepository.delete(id);
    }
}
```

6. CONTROLLER

6.1. ApartmentController.java

```
package com.uef.controller;
import com.uef.model.Apartment;
import com.uef.service.ApartmentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@Controller
@RequestMapping("/apartments")
public class ApartmentController {

    @Autowired
    private ApartmentService apartmentService;
    private final String path = "/WEB-INF/views/";


    // Hiển thị danh sách căn hộ
    @GetMapping
    public String listApartments(Model model) {
        List<Apartment> apartments = apartmentService.getAll();
        model.addAttribute("apartments", apartments);
        model.addAttribute("body", path + "apartment/list.jsp");
        return "layout/main";
    }
    // Hiển thị form thêm mới
    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("apartment", new Apartment());
        model.addAttribute("body", path + "apartment/form.jsp");
        return "layout/main";
    }

    // Xử lý thêm mới
    @PostMapping("/add")
    public String addApartment(@ModelAttribute Apartment apartment) {
        apartmentService.add(apartment);
        return "redirect:/apartments";
    }

    // Hiển thị form cập nhật
    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable("id") int id, Model model) {
        Apartment apartment = apartmentService.getById(id);
        model.addAttribute("apartment", apartment);
        model.addAttribute("body", path + "apartment/form.jsp");
        return "layout/main";
    }

    // Xử lý cập nhật
    @PostMapping("/edit")
    public String updateApartment(@ModelAttribute Apartment apartment) {
        apartmentService.update(apartment);
        return "redirect:/apartments";
    }

    // Xử lý xóa
    @GetMapping("/delete/{id}")
    public String deleteApartment(@PathVariable("id") int id) {
        apartmentService.delete(id);
        return "redirect:/apartments";
    }
}
```

6.2. HouseholdController.java

```
package com.uef.controller;
import com.uef.model.Household;
import com.uef.model.Apartment;
import com.uef.service.ApartmentService;
import com.uef.service.HouseholdService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@Controller
@RequestMapping("/households")
public class HouseholdController {

   @Autowired
   private HouseholdService householdService;

   @Autowired
   private ApartmentService apartmentService;

   private final String path = "/WEB-INF/views/";

   // Hiển thị danh sách hộ dân
   @GetMapping
   public String listHouseholds(Model model) {
      List<Household> households = householdService.getAll();
      model.addAttribute("households", households);
      model.addAttribute("body", path + "household/list.jsp");
      return "layout/main";
   }

   // Hiển thị form thêm mới
   @GetMapping("/add")
   public String showAddForm(Model model) {
      model.addAttribute("household", new Household());
      model.addAttribute("apartments", apartmentService.getAll());
      model.addAttribute("body", path + "household/form.jsp");
      return "layout/main";
   }

   // Xử lý thêm mới
   @PostMapping("/add")
   public String addHousehold(@ModelAttribute Household household) {
      householdService.add(household);
      return "redirect:/households";
   }

   // Hiển thị form cập nhật
   @GetMapping("/edit/{id}")
   public String showEditForm(@PathVariable("id") int id, Model model) {
      Household household = householdService.getById(id);
      model.addAttribute("household", household);
      model.addAttribute("apartments", apartmentService.getAll());
```

```
        model.addAttribute("body", path + "household/form.jsp");
        return "layout/main";
    }

    // Xử lý cập nhật
    @PostMapping("/edit")
    public String updateHousehold(@ModelAttribute Household household) {
        householdService.update(household);
        return "redirect:/households";
    }

    // Xử lý xóa
    @GetMapping("/delete/{id}")
    public String deleteHousehold(@PathVariable("id") int id) {
        householdService.delete(id);
        return "redirect:/households";
    }
}
```

6.3. ResidentController.java

```
package com.uef.controller;
import com.uef.model.Resident;
import com.uef.model.Household;
import com.uef.service.ResidentService;
import com.uef.service.HouseholdService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@Controller
@RequestMapping("/residents")
public class ResidentController {

    @Autowired
    private ResidentService residentService;

    @Autowired
    private HouseholdService householdService;

    private final String path = "/WEB-INF/views/";

    // Hiển thị danh sách cư dân
    @GetMapping
    public String listResidents(Model model) {
        List<Resident> residents = residentService.getAll();
        model.addAttribute("residents", residents);
        model.addAttribute("body", path + "resident/list.jsp");
        return "layout/main";
    }

    // Hiển thị form thêm cư dân
    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("resident", new Resident());
        model.addAttribute("households", householdService.getAll());
        model.addAttribute("body", path + "resident/form.jsp");
        return "layout/main";
    }

    // Xử lý thêm mới cư dân
    @PostMapping("/add")
    public String addResident(@ModelAttribute Resident resident) {
        residentService.add(resident);
        return "redirect:/residents";
    }

    // Hiển thị form cập nhật cư dân
    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable("id") int id, Model model) {
        Resident resident = residentService.getById(id);
        model.addAttribute("resident", resident);
        model.addAttribute("households", householdService.getAll());
        model.addAttribute("body", path + "resident/form.jsp");
        return "layout/main";
    }

    // Xử lý cập nhật
    @PostMapping("/edit")
    public String updateResident(@ModelAttribute Resident resident) {
        residentService.update(resident);
        return "redirect:/residents";
    }

    // Xử lý xóa cư dân
    @GetMapping("/delete/{id}")
    public String deleteResident(@PathVariable("id") int id) {
        residentService.delete(id);
        return "redirect:/residents";
    }
}
```

6.4. FeeController.java

```
package com.uef.controller;

import com.uef.model.Fee;
import com.uef.service.FeeService;
import com.uef.service.HouseholdService;
import com.uef.model.Household;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping("/fees")
public class FeeController {

    @Autowired
    private FeeService feeService;

    @Autowired
    private HouseholdService householdService;

    private final String path = "/WEB-INF/views/";

    // Hiển thị danh sách phí
    @GetMapping
    public String listFees(Model model) {
        List<Fee> fees = feeService.getAll();
        model.addAttribute("fees", fees);
        model.addAttribute("body", path + "fee/list.jsp");
        return "layout/main";
    }

    // Hiển thị form thêm phí
    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("fee", new Fee());
        model.addAttribute("households", householdService.getAll());
        model.addAttribute("body", path + "fee/form.jsp");
        return "layout/main";
    }

    // Xử lý thêm mới
    @PostMapping("/add")
    public String addFee(@ModelAttribute Fee fee) {
        feeService.add(fee);
        return "redirect:/fees";
    }

    // Hiển thị form chỉnh sửa
    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable("id") int id, Model model) {
        Fee fee = feeService.getById(id);
        model.addAttribute("fee", fee);
        model.addAttribute("households", householdService.getAll());
        model.addAttribute("body", path + "fee/form.jsp");
        return "layout/main";
    }

    // Xử lý cập nhật
    @PostMapping("/edit")
    public String updateFee(@ModelAttribute Fee fee) {
        feeService.update(fee);
        return "redirect:/fees";
    }

    // Xử lý xóa
    @GetMapping("/delete/{id}")
    public String deleteFee(@PathVariable("id") int id) {
        feeService.delete(id);
        return "redirect:/fees";
    }
}
```

6.5. ParkingController.java

```
package com.uef.controller;
import com.uef.model.Parking;
import com.uef.model.Household;
import com.uef.service.ParkingService;
import com.uef.service.HouseholdService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping("/parking")
public class ParkingController {
```

```
@Autowired
private ParkingService parkingService;

@Autowired
private HouseholdService householdService;

private final String path = "/WEB-INF/views/";

// Hiển thị danh sách chỗ đỗ xe
@GetMapping
public String listParkingSpots(Model model) {
    List<Parking> parkings = parkingService.getAll();
    model.addAttribute("parkings", parkings);
    model.addAttribute("body", path + "parking/list.jsp");
    return "layout/main";
}

// Hiển thị form thêm chỗ đỗ
@GetMapping("/add")
public String showAddForm(Model model) {
    model.addAttribute("parking", new Parking());
    model.addAttribute("households", householdService.getAll());
    model.addAttribute("body", path + "parking/form.jsp");
    return "layout/main";
}

// Xử lý thêm mới
@PostMapping("/add")
public String addParking(@ModelAttribute Parking parking) {
    parkingService.add(parking);
    return "redirect:/parking";
}

// Hiển thị form cập nhật
@GetMapping("/edit/{id}")
public String showEditForm(@PathVariable("id") int id, Model model) {
    Parking parking = parkingService.getById(id);
    model.addAttribute("parking", parking);
    model.addAttribute("households", householdService.getAll());
    model.addAttribute("body", path + "parking/form.jsp");
    return "layout/main";
}

// Xử lý cập nhật
@PostMapping("/edit")
public String updateParking(@ModelAttribute Parking parking) {
    parkingService.update(parking);
    return "redirect:/parking";
    }

    // Xử lý xóa
    @GetMapping("/delete/{id}")
    public String deleteParking(@PathVariable("id") int id) {
        parkingService.delete(id);
        return "redirect:/parking";
    }
}
```

6.6. ComplaintController.java

```
package com.uef.controller;
import com.uef.model.Complaint;
import com.uef.service.ComplaintService;
import com.uef.service.HouseholdService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller
@RequestMapping("/complaints")
public class ComplaintController {

    @Autowired
    private ComplaintService complaintService;

    @Autowired
    private HouseholdService householdService;

    private final String path = "/WEB-INF/views/";

    // Hiển thị danh sách khiếu nại
    @GetMapping
    public String listComplaints(Model model) {
        List<Complaint> complaints = complaintService.getAll();
        model.addAttribute("complaints", complaints);
        model.addAttribute("body", path + "complaint/list.jsp");
        return "layout/main";
    }

    // Hiển thị form thêm mới khiếu nại
    @GetMapping("/add")
    public String showAddForm(Model model) {
        model.addAttribute("complaint", new Complaint());
        model.addAttribute("households", householdService.getAll());
        model.addAttribute("body", path + "complaint/form.jsp");
        return "layout/main";
    }

    // Xử lý thêm mới
    @PostMapping("/add")
    public String addComplaint(@ModelAttribute Complaint complaint) {
        complaintService.add(complaint);
        return "redirect:/complaints";
    }

    // Hiển thị form chỉnh sửa
    @GetMapping("/edit/{id}")
    public String showEditForm(@PathVariable("id") int id, Model model) {
        Complaint complaint = complaintService.getById(id);
        model.addAttribute("complaint", complaint);
        model.addAttribute("households", householdService.getAll());
        model.addAttribute("body", path + "complaint/form.jsp");
        return "layout/main";
    }

    // Xử lý cập nhật
    @PostMapping("/edit")
    public String updateComplaint(@ModelAttribute Complaint complaint) {
        complaintService.update(complaint);
        return "redirect:/complaints";
    }

    // Xử lý xóa
    @GetMapping("/delete/{id}")
    public String deleteComplaint(@PathVariable("id") int id) {
        complaintService.delete(id);
        return "redirect:/complaints";
    }
}
```

7. VIEW

7.1. JSP folder structure

```
/WEB-INF/views/
├── shared/
│   ├── header.jsp
│   ├── footer.jsp
│   └── main.jsp

├── apartment/
│   ├── list.jsp
│   ├── listContent.jsp
│   ├── form.jsp
│   └── formContent.jsp
```

```
├── household/
│   ├── list.jsp
│   ├── listContent.jsp
│   ├── form.jsp
│   └── formContent.jsp

├── resident/
│   ├── list.jsp
│   ├── listContent.jsp
│   ├── form.jsp
│   └── formContent.jsp
```

```
├── parking/
│   ├── list.jsp
│   ├── listContent.jsp
│   ├── form.jsp
│   └── formContent.jsp

├── complaint/
│   ├── list.jsp
│   ├── listContent.jsp
│   ├── form.jsp
│   └── formContent.jsp
```

7.2. Layout

1. **Header (header.jsp)**

```
<%@ page language="java" contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <title>Apartment Management</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-primary mb-3">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">Chung cư</a>
        <div class="collapse navbar-collapse">
            <ul class="navbar-nav me-auto">
                <li class="nav-item"><a class="nav-link"
href="${pageContext.request.contextPath}/apartments">Căn hộ</a></li>
                <li class="nav-item"><a class="nav-link"
href="${pageContext.request.contextPath}/households">Hộ dân</a></li>
                <li class="nav-item"><a class="nav-link"
href="${pageContext.request.contextPath}/residents">Thành viên</a></li>
                <li class="nav-item"><a class="nav-link"
href="${pageContext.request.contextPath}/parking">Bãi đỗ</a></li>
                <li class="nav-item"><a class="nav-link"
href="${pageContext.request.contextPath}/complaints">Khiếu nại</a></li>
```

```
                <li class="nav-item"><a class="nav-link"
href="${pageContext.request.contextPath}/fees">Phí</a></li>
            </ul>
        </div>
    </div>
</nav>
<div class="container">
```

2. **Footer (footer.jsp)**

```
<%@page language="java" contentType="text/html" pageEncoding="UTF-8"%>

</div> <!-- Đóng container -->
<footer class="bg-light text-center mt-4 py-3 border-top">
    <p class="mb-0">&copy; 2025 Hệ thống quản lý chung cư</p>
</footer>

<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

3. **Main (main.jsp)**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<jsp:include page="/WEB-INF/views/layout/header.jsp"/>
<jsp:include page="${body}" />
<jsp:include page="/WEB-INF/views/layout/footer.jsp"/>
```

7.3. Apartment

1. **List of apartments (list.jsp)**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<div class="d-flex justify-content-between align-items-center mb-3">
    <h2>Danh sách căn hộ</h2>
    <a href="${pageContext.request.contextPath}/apartments/add" class="btn btn-
success">+ Thêm mới</a>
</div>

<table class="table table-bordered table-hover">
    <thead class="table-primary">
        <tr>
            <th>ID</th>
            <th>Số căn</th>
            <th>Tầng</th>
            <th>Diện tích (m²)</th>
            <th>Trạng thái</th>
            <th>Hành động</th>
        </tr>
    </thead>
    <tbody>
        <c:forEach var="a" items="${apartments}">
            <tr>
                <td>${a.apartmentID}</td>
                <td>${a.apartmentNumber}</td>
                <td>${a.floor}</td>
                <td>${a.area}</td>
                <td>
                    <c:choose>
                        <c:when test="${a.status == 'Sold'}"><span class="badge bg-danger">Đã
bán</span></c:when>
                        <c:when test="${a.status == 'Rented'}"><span class="badge bg-warning
text-dark">Đang thuê</span></c:when>
                        <c:otherwise><span class="badge bg-success">Trống</span></c:otherwise>
                    </c:choose>
                </td>
                <td>
                    <a
href="${pageContext.request.contextPath}/apartments/edit/${a.apartmentID}" class="btn
btn-sm btn-primary">Sửa</a>
                    <a
href="${pageContext.request.contextPath}/apartments/delete/${a.apartmentID}" class="btn
btn-sm btn-danger" onclick="return confirm('Bạn chắc chắn muốn xóa?')">Xóa</a>
                </td>
            </tr>
        </c:forEach>
    </tbody>
</table>
```

2. Form Add/Edit (form.jsp)

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<div class="container mt-4">
    <h3 class="mb-4">
        <c:choose>
            <c:when test="${apartment.apartmentID != null}">
                Edit Apartment
            </c:when>
            <c:otherwise>
                Add New Apartment
            </c:otherwise>
        </c:choose>
    </h3>

    <form:form modelAttribute="apartment" method="post" cssClass="needs-validation">

        <form:hidden path="apartmentID" />

        <div class="mb-3">
            <form:label path="apartmentNumber" cssClass="form-label">Apartment
Number</form:label>
            <form:input path="apartmentNumber" cssClass="form-control" required="true"/>
        </div>

        <div class="mb-3">
            <form:label path="floor" cssClass="form-label">Floor</form:label>
            <form:input path="floor" cssClass="form-control" type="number" required="true"/>
        </div>

        <div class="mb-3">
            <form:label path="area" cssClass="form-label">Area (m²)</form:label>
            <form:input path="area" cssClass="form-control" type="number" step="0.01"
required="true"/>
        </div>

        <div class="mb-3">
            <form:label path="status" cssClass="form-label">Status</form:label>
            <form:select path="status" cssClass="form-select">
                <form:option value="Sold">Sold</form:option>
                <form:option value="Rented">Rented</form:option>
                <form:option value="Vacant">Vacant</form:option>
            </form:select>
        </div>

        <button type="submit" class="btn btn-primary">
            <c:choose>
                <c:when test="${apartment.apartmentID != null}">
                    Update
                </c:when>
                <c:otherwise>
                    Save
                </c:otherwise>
            </c:choose>
        </button>
        <a href="<c:url value='/apartments' />" class="btn btn-secondary">Cancel</a>
    </form:form>
</div>
```

Students continue to complete the rest of the lab.