To create an effective 1-hour presentation on \*\*Spring MVC\*\*, we'll break it down into sections that introduce core concepts, explain key components, and provide a hands-on demo with a sample application. The presentation will aim to give attendees a foundational understanding of Spring MVC, its architecture, and how to create a basic web application using it.

### Presentation Plan: Spring MVC Overview (1 Hour)

#### \*\*0 - 5 minutes: Introduction to Spring MVC\*\*

- \*\*Objective\*\*: Introduce the audience to Spring MVC and its relevance.

- What is Spring MVC?

- Key features of Spring MVC (Model-View-Controller architecture, lightweight framework, flexible view resolution).

- Difference between Spring MVC and other web frameworks.

- Overview of the Spring Web Module.

#### \*\*5 - 15 minutes: Core Components of Spring MVC\*\*

- \*\*Objective\*\*: Explain the core components of Spring MVC and their roles.

- \*\*DispatcherServlet\*\*: The front controller of the MVC pattern, responsible for request handling.

- \*\*Controller\*\*: Role and use of annotated controllers (`@Controller`, `@RestController`).

- \*\*ViewResolver\*\*: How Spring resolves views using different view resolvers (Thymeleaf, JSP, etc.).

- \*\*Model and View\*\*: The `Model` interface and its usage in data binding between the controller and view.

#### \*\*15 - 25 minutes: Understanding Request Handling in Spring MVC\*\*

- \*\*Objective\*\*: Dive into how Spring MVC processes requests and maps them to controllers.

- \*\*RequestMapping\*\*: Usage of `@RequestMapping`, `@GetMapping`, `@PostMapping`, etc., to handle different types of HTTP requests.

- \*\*Handler Method Parameters\*\*: Commonly used parameters like `@RequestParam`, `@PathVariable`, `Model`, `HttpServletRequest`.

- \*\*Handler Method Return Types\*\*: Different return types such as `String`, `ModelAndView`, `ResponseEntity`.

#### \*\*25 - 35 minutes: View Resolution and Templates\*\*

- \*\*Objective\*\*: Explain how Spring MVC resolves views and demonstrates template engines.

- \*\*Thymeleaf and JSP View Resolvers\*\*: Configuring view resolvers and using different template engines.

- \*\*Thymeleaf vs. JSP\*\*: Brief comparison and use case discussion.

- \*\*Example Code Snippet\*\*: Show a sample Thymeleaf template and corresponding controller method.

#### \*\*35 - 55 minutes: Live Demo - Building a Simple Spring MVC Application\*\*

- \*\*Objective\*\*: Walk through the creation of a basic Spring MVC application.

- \*\*Step 1\*\*: Set up a new Spring Boot project with Spring Web dependency.

- \*\*Step 2\*\*: Create a simple controller (`HomeController`) that handles basic HTTP GET requests.

- \*\*Step 3\*\*: Configure Thymeleaf as a view resolver and create a simple `home.html` page.

- \*\*Step 4\*\*: Demonstrate form handling with `@RequestParam` for user input.

- \*\*Step 5\*\*: Show an example of using `@PathVariable` to handle dynamic URLs.

- \*\*Step 6\*\*: Explain and demonstrate error handling with `@ControllerAdvice` and `@ExceptionHandler`.

#### \*\*55 - 60 minutes: Q&A and Recap\*\*

- \*\*Objective\*\*: Address any questions from the audience and summarize the key takeaways.

- Recap key concepts covered (DispatcherServlet, controllers, request mapping, view resolution).

- Encourage attendees to explore further by building more complex applications with Spring MVC.

### Example Application for Demo: Simple Product Management System

#### 1. \*\*Project Setup\*\*

- \*\*Tool\*\*: Spring Boot Initializr (or any IDE with Spring Boot support).

- \*\*Dependencies\*\*: `Spring Web`, `Thymeleaf`, `Spring Boot DevTools`.

#### 2. \*\*Create a Basic Spring MVC Application\*\*

- \*\*Application Structure\*\*:

```

└── src

└── main

├── java

│ └── com.example.springmvc

│ ├── SpringMvcDemoApplication.java

│ └── controller

│ └── ProductController.java

└── resources

├── templates

│ └── home.html

└── application.properties

```

#### 3. \*\*Code Implementation\*\*

- \*\*Main Application Class\*\* (`SpringMvcDemoApplication.java`):

```java

package com.example.springmvc;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringMvcDemoApplication {

public static void main(String[] args) {

SpringApplication.run(SpringMvcDemoApplication.class, args);

}

}

```

- \*\*Controller Class\*\* (`ProductController.java`):

```java

package com.example.springmvc.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestParam;

@Controller

public class ProductController {

@GetMapping("/")

public String home() {

return "home"; // maps to home.html in the templates folder

}

@GetMapping("/product/{id}")

public String getProduct(@PathVariable("id") int id, Model model) {

// Simulate a product retrieval

model.addAttribute("product", "Product " + id);

return "product";

}

@GetMapping("/search")

public String searchProduct(@RequestParam("query") String query, Model model) {

// Simulate search

model.addAttribute("result", "Search result for: " + query);

return "search";

}

}

```

- \*\*Thymeleaf Templates\*\*:

- \*\*`home.html`\*\*:

```html

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Home</title>

</head>

<body>

<h1>Welcome to Spring MVC Demo</h1>

<a href="/product/1">View Product 1</a><br>

<form action="/search" method="get">

<input type="text" name="query" placeholder="Search product">

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

</form>

</body>

</html>

```

- \*\*Additional Templates\*\* for `product.html` and `search.html` would similarly display dynamic content based on the controller logic.

#### 4. \*\*Run and Test the Application\*\*

- Start the Spring Boot application and demonstrate:

- Home page rendering.

- Viewing a product using dynamic URLs.

- Searching for products using form input.

This plan covers foundational concepts and provides a practical, hands-on demo to solidify understanding. Let me know if you'd like more details or specific content for any part of this presentation!