Spring MVC

To use **Spring MVC** without **Spring Boot**, you need to manually configure the Spring framework, as Spring Boot typically automates much of the setup. Below is a step-by-step guide to create a simple Spring MVC web application using XML-based configuration. This example will set up a basic application with a single controller that displays a "Welcome"message on a JSP page.

Step-by-Step Guide

1. Set Up the Project Structure

Create a standard web application structure. You can use an IDE like IntelliJ or Eclipse, or manually create the folders. The structure should look like this:

```
SimpleSpringMVCApp/
  src/
     main/
         java/
             com/
                 example/
                     controllers/
                         HomeController.java
         webapp/
             WEB-INF/
                 web.xml
                 spring-mvc-config.xml
                 views/
                     home.jsp
  pom.xml  (if using Maven)
```

- `src/main/java`: Contains your Java source code (e.g., controllers).
- `src/main/webapp/WEB-INF`: Contains configuration files (`web.xml`, `spring-mvc-config.xml`) and JSP views.

2. Add Dependencies

If you're using Maven, include the required dependencies in your `pom.xml`. For a simple Spring MVC application, you need the Spring Web MVC library and the Servlet API (provided by the container).

Create or edit `pom.xml` with the following:

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>SimpleSpringMVCApp</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <dependencies>
        <!-- Spring Web MVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.10</version>
        </dependency>
        <!-- Servlet API (provided by the container) -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>4.0.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.1</version>
            </plugin>
        </plugins>
    </build>
</project>
```

- **Notes**:
  - `<packaging>war</packaging>`: Ensures the project is packaged as a WAR file for deployment to a servlet container.
  - If you're not using Maven, manually download the Spring MVC JARs and Servlet API JARs and add

them to your project's classpath.

3. Configure the DispatcherServlet in `web.xml`

The `web.xml` file is the deployment descriptor for your web application. It configures the `DispatcherServlet`, Spring MVC's front controller, to handle incoming requests.

Create `src/main/webapp/WEB-INF/web.xml` with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0
         version="4.0">

    <!-- Define the DispatcherServlet -->
    <servlet>
        <servlet-name>spring-mvc</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring-mvc-config.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Map the DispatcherServlet to handle all requests -->
    <servlet-mapping>
        <servlet-name>spring-mvc</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

- **Explanation**:
  - `<servlet-class>`: Specifies the `DispatcherServlet`, which routes requests to controllers.
  - `<init-param>`: Points to the Spring configuration file (`spring-mvc-config.xml`).
  - `<url-pattern>/</url-pattern>`: Maps the servlet to handle all requests to the application.

4. Create the Spring Configuration File

Create `src/main/webapp/WEB-INF/spring-mvc-config.xml` to define Spring MVC beans, such as controllers and view resolvers.

```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-bean
           http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
           http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">

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

    <!-- Enable annotation-driven MVC -->
    <mvc:annotation-driven />

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

- **Explanation**:
  - `<context:component-scan>`: Scans the `com.example.controllers` package for annotated components (e.g., `@Controller`).
  - `<mvc:annotation-driven>`: Enables annotation-based MVC features (e.g., `@GetMapping`).
  - `InternalResourceViewResolver`: Maps view names to JSP files in `/WEB-INF/views/` with a `.jsp` suffix.

5. Create a Simple Controller

Create a controller to handle HTTP requests. Add `HomeController.java` in `src/main/java/com/example/controllers/`:

```java
package com.example.controllers;

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

@Controller
public class HomeController {
```

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

- **Explanation**:
  - `@Controller`: Marks this class as a Spring MVC controller.
  - `@GetMapping("/")`: Maps GET requests to the root URL (`/`) to the `home()` method.
  - `return "home"`: Returns the view name `"home"`, which resolves to `/WEB-INF/views/home.jsp`.

6. Create a JSP View

Create a simple JSP file to display the output. Add `home.jsp` in `src/main/webapp/WEB-INF/views/`:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Home</title>
</head>
<body>
    <h1>Welcome to Spring MVC without Spring Boot</h1>
</body>
</html>
```

7. Build and Package the Application

If using Maven, run the following command from the project root to build the WAR file:

```
mvn clean package
```

This generates `SimpleSpringMVCApp-1.0-SNAPSHOT.war` in the `target` directory.

- **Note**: If not using Maven, compile the Java files and package the project into a WAR file manually, maintaining the folder structure.

8. Deploy the Application

Deploy the WAR file to a servlet container like Apache Tomcat: - Copy `SimpleSpringMVCApp-1.0-SNAPSHOT.war` to the `webapps` directory of your Tomcat installation. - Start Tomcat (e.g., run `startup.sh` or `startup.bat` from the Tomcat `bin` directory).

9. Access the Application

Open a browser and navigate to:

```
http://localhost:8080/SimpleSpringMVCApp-1.0-SNAPSHOT/
```

You should see:

```
Welcome to Spring MVC without Spring Boot
```

- **Note**: The context path (`/SimpleSpringMVCApp-1.0-SNAPSHOT`) depends on the WAR file name. If you rename the WAR to `myapp.war`, the URL becomes `http://localhost:8080/myapp/`.

Summary

To use **Spring MVC without Spring Boot**: 1. **Set up a web project** with a standard structure. 2. **Add dependencies** for Spring MVC and the Servlet API. 3. **Configure the DispatcherServlet** in `web.xml`. 4. **Define Spring configuration** in an XML file (e.g., `spring-mvc-config.xml`). 5. **Create a controller** to handle requests. 6. **Add a JSP view** for the response. 7. **Build and deploy** the WAR file to a servlet container.

This approach gives you full control over the configuration, making it a great way to learn Spring MVC fundamentals!