++++++++++++++
Spring Web MVC
++++++++++++++

-> Spring Web MVC is one module available in the spring framework

-> Using Spring Web MVC module we can develop 2 types of applications

1) Web Applications

2) Distributed Application (Webservices)

-> Web Applications will have user interface (UI)
-> Customers can access web applications directley using internet
-> Web Applications meant for customer to business communication (C 2 B)

Ex: facebook, gmail, linkedin, naukri etc...

-> Distributed Applications are meant for Business to Business Communication (B 2 B)
-> If one application is communicating with another application then we call them as Distributed apps
-> Distributed Applications we can develop in 2 ways

1) SOAP Webservices
2) RESTFul Services

Note: SOAP Webservices & RESTFul Services can be developed using Spring Web MVC

-> Distributed applications we are developing to re-use logic of one application in another application.

Ex:

MakeMyTrip ----------> IRCTC
Passport ----------> AADHAR
Gpay ----------> Banking Apps
Swiggy ----------> Banking Apps

++++++++++++++++++++++++++
Advantages of Spring Web MVC
++++++++++++++++++++++++++

1) Easily we can develop web & distributed applications using Web MVC module

2) It supports Multiple Presentation Technologies (JSP & Thymeleaf)

3) I 18 N Support (Internationalization)

4) Form Bindings (Form Data will be binded to Object and vice versa)

5) Form Tag Library (To simplify forms development with Dynamic behvaiour)

6) Having support for XML to Java object conversion and vice versa

7) Having support for JSON to java object conversion and vice versa

++++++++++++++++++++++++++
Spring Web MVC Architecture
++++++++++++++++++++++++++

1) Dispatcher Servlet (Front Controller)
2) Handler Mapper
3) Controller
4) Model And View
5) ViewResolver
6) View

-> DispatcherServlet is a pre-defined servlet class in Spring Web MVC
-> DispatcherServlet is called as Front Controller / Framework Servlet
-> It is responsible to perform pre-processing and post-processing for every request

-> HandlerMapper is a pre-defined class in spring web mvc
-> HandlerMapper is used to identify Request Handler
-> It will identify which request should be processed by which Controller class
-> HandlerMapper will identify Request Handler based on URL Pattern

-> Controller is a class which contains logic to handle request and response
-> Controller is also called as Request Handler
-> We will create Controller classes using @Controller annotation
-> Controller will return data to Dispatcher in ModelAndView object

-> Model represents Data in Key-Value pair format
-> View represents presentation logical file name
-> To display data in view file we will use ModelAndView object

-> ViewResolver is used to identify where view files available in our project
-> ViewResolver is responsible to identify physical location of view files

-> View component is used to render model data on physical view file to display that to end user

++
Building First Web Application using Spring Web MVC
++

1) Create Spring Starter application with below dependencies

    a) spring-boot-starter-web
    b) tomcat-embed-jasper
    c) devtools

2) Create Controller class and write Required methods

3) Create View Files with Presentation logic

4) Configure ViewResolver in application.properties file with prefix and suffix

5) Run the application and test it.

Note-1 : web-starter will provide the support to build web apps with MVC architecture and it provides Tomcat as default embedded container (we no neeed to setup server manually).

Note-2 : tomcat-embed-jasper will provide the support to work with JSP files in Spring Web MVC

Note-3 : devtools is used to re-start the server when changes happend in the code.

Note-4 : Java class will be represented as a Spring Controller using @Controller annotation

Note-5 : Controller class methods should be binded to HTTP Protocol methods to handle HTTP Requests

tomact-embed-jasper dependency

```
<dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

Controller class

```
@Controller
public class WelcomeController {

        @GetMapping("/welcome")
        public ModelAndView getWelcomeMsg() {

                ModelAndView mav = new ModelAndView();

                mav.addObject("msg", "Welcome to Ashok IT...!!");

                mav.setViewName("index");

                return mav;
        }
}
```

view resolver configuration in application.properties

```
spring.mvc.view.prefix=/views/
spring.mvc.view.suffix=.jsp
```

jsp file

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
        <h1>${msg}</h1>
</body>
</html>
```

++++++++++++++++++++
What is Context-Path ?
++++++++++++++++++++

-> Context-Path represents name of our application

-> In Spring Boot, the default context-path is empty

-> In Spring Boot we can set our own context-path using below property in application.properties file

```
server.servlet.context-path=/webapp
```

-> When we set context we have to access our application using context-path in URL.

```
URL : http://localhost:8080/webapp/welcome
```

Note: Embedded Tomcat Server will run on the port number 8080. This is default behaviour.

-> We can change embedded server port number using below property in application.properties

```
server.port=9090
```

++++++++++++++++++++++++++++++++
Sending Data From Controller To UI
++++++++++++++++++++++++++++++++

-> We can send data from controller to UI in multiple ways

1) ModelAndView

2) Model

3) @ResponseBody

```
################# Approach-1   (ModelAndView) ####################

@Controller
public class WelcomeController {

	@GetMapping("/welcome")
	public ModelAndView welcomeMsg() {

		ModelAndView mav = new ModelAndView();

		mav.addObject("msg", "Welcome to Ashok IT");

		mav.setViewName("welcome");

		return mav;
	}
}


################# Approach-2   (Model) ####################

@Controller
public class GreetController {

	@GetMapping("/greet")
	public String getGreetMsg(Model model) {

		String msgTxt = "Good Morning..";

		model.addAttribute("msg", msgTxt);

		return "greet";
	}
}


################# Approach-3   ( @ResponseBody ) ####################

@Controller
public class WishController {

	@GetMapping("/wish")
	@ResponseBody
	public String getWishMsg() {

		String msg = "All the best...!!!";

		return msg;
	}
}



################# sending object data from Controller to UI
######################
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Book {

	private Integer bookId;
	private String bookName;
	private Double bookPrice;

}
```

```
@Controller
public class BookController {

	@GetMapping("/book")
	public String getBookData(Model model) {

		// setting data to binding obj
		Book bookObj = new Book(101, "Spring", 450.00);

		// adding data to model obj to send to UI
		model.addAttribute("book", bookObj);

		// return view name
		return "book";
	}
}
```

```
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

	<h2>Book Data</h2>

	Book Id : ${book.bookId} <br/>

	Book Name : ${book.bookName} <br/>

	Book Price : ${book.bookPrice} <br/>

</body>
</html>
```

Assignment: Develop Spring Boot web application to display multiple books in a table format.

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

--------------------------

	@GetMapping("/books")
	public String getBooksData(Model model) {
		// setting data to binding obj
		Book b1 = new Book(101, "Spring", 350.00);
		Book b2 = new Book(102, "Python", 450.00);
		Book b3 = new Book(103, "AWS", 550.00);

		List<Book> booksList = Arrays.asList(b1, b2, b3);

		// adding data to model obj to send to UI
		model.addAttribute("books", booksList);

		// return view name
		return "books";
	}
----------------------------------------

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<table border="1">
		<thead>
			<tr>
				<th>Book ID</th>
				<th>Book Name</th>
				<th>Book Price</th>
			</tr>
		</thead>
		<tbody>
			<c:forEach items="${books}" var="book">
				<tr>
					<td>${book.bookId}</td>
					<td>${book.bookName}</td>
					<td>${book.bookPrice}</td>
				</tr>
			</c:forEach>
		</tbody>
	</table>
</body>
</html>
```

++++++++++++++++++++++++++++++++++
Forms Development Using Spring Web MVC
++++++++++++++++++++++++++++++++++

-> Forms are very important in every web application

-> Forms are used to collect data from the user

Ex: Login form, Registration form, Search Forms etc....

-> Spring Web MVC provided form tag library to develop forms easily

-> Spring Form Tag Library contains several Tags

<form:form >
<form:input >
<form:password>
<form:radioButton> & <form:radioButtons>
<form:select>
<form:option> & <form:options>
<form:checkbox> & <form:checkboxes>
<form:hidden>
<form:error>

-> Spring Web MVC support Form Binding that means it can bind form data to object and vice versa.

Note: In servlets we use request.getParameter("key") to capture form data.

-> In Spring Web MVC we no need to use request.getParameter("") to capture form data bcz Web MVC having Form Binding Support.

-> To achieve Form Binding we need to create a binding class
    (class variables will be mapped with form fields)

-> To work with Spring Form Tag library we need to use below taglib directive

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

++++++++++++++++++++++++++++++++
Steps to build first form based application
++++++++++++++++++++++++++++++++

1) Create boot app with below dependencies

    a) web-starter
    b) devtools
    c) lombok
    d) tomcat-embed-jasper

2) Create Form Binding class

3) Create a controller class with required methods

a) method to display empty form (GET Request Method)
b) method to handle form submission (POST Request Method)

4) Create View Page with presentation logic

5) configure view resolver in application.properties file

```
@Data
public class Product {

	private Integer productId;

	private String productName;

	private Double productPrice;

}
```

```
@Controller
public class ProductController {

	@GetMapping("/")
	public String getProductForm(Model model) {
		Product productObj = new Product();
		model.addAttribute("product", productObj);
		return "index";
	}

	@PostMapping("/product")
	public String handleFormSubmit(Product product, Model model) {
		System.out.println(product);
		model.addAttribute("msg", "Product Saved Successfully");
		return "success";
	}
}
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1"%>

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

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
```

```
	<h3>Save Product Data</h3>
	<form:form action="product" modelAttribute="product" method="POST">
		<table>
			<tr>
				<td>Product ID</td>
				<td><form:input path="productId" /></td>
			</tr>
			<tr>
				<td>Product Name</td>
				<td><form:input path="productName" /></td>
			</tr>
			<tr>
				<td>Product Price</td>
				<td><form:input path="productPrice" /></td>
			</tr>
			<tr>
				<td><input type="submit" value="Submit" /></td>
			</tr>
		</table>
	</form:form>
</body>
</html>

--------------------------------------------------------------------------------
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<h2>${msg}</h2>
	<a href="/">Go Home</a>
</body>
</html>

--------------------------------------------------------------------------------
------------


++++++++++++++++++
Form Validations
++++++++++++++++++

-> Forms are used to capture data from the user

-> To make sure users are entering valid data we will form validations on the
data

-> Spring Web MVC having support to perform form validations....


@NotEmpty
```

@NotNull

@Size

@Min

@Max

-> We need to below dependency in pom.xml to perform form validations

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

```
@Data
public class User {

	@NotEmpty(message = "Uname is required")
	@Size(min = 3, max = 8, message = "Uname should be 3 to 8 characters")
	private String uname;

	@NotEmpty(message = "Pwd is required")
	private String pwd;

	@NotEmpty(message = "Email is required")
	@Email(message = "Enter valid email id")
	private String email;

	@NotEmpty(message = "Phno is required")
	@Size(min = 10, message = "Phno should have atleast 10 digits")
	private String phno;

	@NotNull(message = "Age is required")
	@Min(value = 21, message = "Age should be minimum 21 years")
	@Max(value = 60, message = "Age shouldn't cross 60 years")
	private Integer age;

}
```

```
@Controller
public class UserController {

	@GetMapping("/")
	public String getForm(Model model) {
		User userObj = new User();
		model.addAttribute("user", userObj);
		return "index";
```

```
	}

	@PostMapping("/register")
	public String handleRegisterBtn(@Valid User userForm, BindingResult
result, Model model) {
		if(result.hasErrors()) {
			return "index";
		}
		System.out.println(userForm);
		//logic to store form data in db
		model.addAttribute("msg", "Your Registration
Successful...!!");
		return "success";
	}
}
-------------------------------------
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
	pageEncoding="ISO-8859-1"%>

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

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>

<style>
.error {
	color: red
}
</style>

</head>
<body>

	<h3>User Registration Form</h3>
	<form:form action="register" modelAttribute="user" method="POST">
		<table>
			<tr>
				<td>Username</td>
				<td><form:input path="uname" /> <form:errors
path="uname" cssClass="error"/></td>
			</tr>
			<tr>
				<td>Pwd</td>
				<td><form:password path="pwd" /> <form:errors
path="pwd" cssClass="error"/></td>
			</tr>
			<tr>
				<td>Email</td>
				<td><form:input path="email" /> <form:errors
path="email" cssClass="error"/></td>
			</tr>
			<tr>
				<td>Phno</td>
```

```
                                <td><form:input path="phno" /> <form:errors
path="phno" cssClass="error"/> </td>
                        </tr>
                        <tr>
                                <td>Age</td>
                                <td><form:input path="age" /> <form:errors
path="age" cssClass="error"/> </td>
                        </tr>
                        <tr>
                                <td></td>
                                <td><input type="submit" value="Register"
/></td>
                        </tr>
                </table>
        </form:form>
</body>
</html>
-----------------------------------------------------
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
        <h2>${msg}</h2>

        <a href="/">Home</a>

</body>
</html>
-----------------------------------------------------


+++++++++
Thymeleaf
++++++++++

-> We used JSP as a presentation technology in our spring web mvc based
applications

-> JSP can't be executed in browser directley

-> When the request comes to JSP then internally JSP will be converted to
Servlet and that servlet will send response to browser

-> When we use JSP for presentation then burden will be increased on server
because every JSP should be converted into Servlet to produce the response to
browser.

-> To overcome problems of JSP we can use Thymeleaf as a presentation
technology

-> Thymleaf is a template engine that can be used in HTML pages directley
```

-> HTML pages can be executed in browser directley
  (Thymeleaf performance will be fast when compared with jsps)

-> In general, HTML pages are used for static data. If we use thymleaf in HTML then we can add dynamic nature to HTML pages.

-> We can develop spring boot application with thymleaf as a presentation technology

-> To use Thymleaf in spring boot we have below starter

'spring-boot-starter-thymleaf'

Procedure to develop spring boot application with thymeleaf

---

1) Create Spring Starter Project with below dependencies

a) web-starter
b) thymeleaf-starter
c) devtools

2) Create Controller with required methods (@Controller)

3) Create Theymeleaf templates in src/main/resources/templates folder (file extension .html)

4) Run the application and test it

Note: No need to configure view resolver because Spring Boot will detect theymeleaf template files and will process them

---

```
<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-thymeleaf</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
```

```
		</dependency>
	</dependencies>
```

```
@Controller
public class WelcomeController {

	@GetMapping("/welcome")
	public String welcomeMsg(@RequestParam("name") String name, Model
model) {
		String msgTxt = name + ", Welcome to Ashok IT..!!";
		model.addAttribute("msg", msgTxt);
		return "index";
	}
}
```

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

	<p th:text=${msg} />

</body>
</html>
```

Spring Boot + Thymeleaf (Form Based Application)

---

1) create a spring starter project with below dependencies

	a) web-starter
	b) thymeleaf-starter
	c) lombok
	d) devtools

2) Create Form Binding Class

```
@Data
public class Product {

	private Integer pid;
	private String pname;
	private Double price;

}
```

3) Create Controller

```
@Controller
public class ProductController {

	@GetMapping("/product")
	public ModelAndView loadForm() {
		ModelAndView mav = new ModelAndView();
		mav.addObject("product", new Product());
		mav.setViewName("productView");
		return mav;
	}

	@PostMapping("/product")
	public ModelAndView handleSubmitBtn(Product product) {
		ModelAndView mav = new ModelAndView();
		mav.setViewName("successView");
		return mav;
	}
}
```

4) Develop view file display form (productView.html)

```
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

	<form th:action="@{/product}" th:object="${product}" method="POST">
		<table>
			<tr>
				<td>Product Id:</td>
				<td><input type="text" th:field="*{pid}"
/></td>
			</tr>
			<tr>
				<td>Product Name:</td>
				<td><input type="text" th:field="*{pname}"
/></td>
			</tr>
			<tr>
				<td>Product Price:</td>
				<td><input type="text" th:field="*{price}"
/></td>
			</tr>

			<tr>
				<td></td>
				<td><input type="submit" value="Save" /></td>
			</tr>
		</table>
	</form>
</body>
```

```
</html>
```

5) Develop view file to display success message (successView.html)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<h1> Product Record Saved Successfully</h1>
	<a href="product">Go Back</a>
</body>
</html>
```

6) Configure the port number and run the application.

```
================================================================================
=================
Spring Boot + Thymeleaf + Form validations - Example
--------------------------------------------------------------------------------
------------------
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.6.2</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>in.ashokit</groupId>
	<artifactId>25-SB-Web-MVC-Form-Validations</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>25-SB-Web-MVC-Form-Validations</name>
	<description>Demo project for Spring Boot</description>
	<properties>
		<java.version>1.8</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-thymeleaf</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-validation</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
```

```
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-
plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
--------------------------------------------------------------------------------
------------------
package in.ashokit.binding;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class Person {

	@NotNull
	@Size(min = 3, max = 8)
	private String name;

	@NotNull
	@Min(18)
	private Integer age;

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Integer getAge() {
		return age;
	}

	public void setAge(Integer age) {
```

```
		this.age = age;
	}

	@Override
	public String toString() {
		return "Person [name=" + name + ", age=" + age + "]";
	}

}
-------------------------------------------------------------------------------
------------------
package in.ashokit.controller;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

import in.ashokit.binding.Person;

@Controller
public class PersonController {

	@GetMapping("/person")
	public String displayForm(Model model) {
		Person personObj = new Person();
		model.addAttribute("person", personObj);
		return "index";
	}

	@PostMapping("/savePerson")
	public String savePerson(@Valid Person person, BindingResult result,
Model model) {
		System.out.println(person);

		if (result.hasErrors()) {
			return "index";
		}

		model.addAttribute("msg", person.getName() + " record saved
successfully");
		return "data";
	}

}
-------------------------------------------------------------------------------
------------------
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
```

```
        <form th:action="@{/savePerson}" th:object="${person}" method="POST">
                <table>
                        <tr>
                                <td>Name :</td>
                                <td><input type="text" th:field="*{name}"
/></td>

                        </tr>
                        <tr>
                                <td>Age :</td>
                                <td><input type="text" th:field="*{age}"
/></td>

                        </tr>
                        <tr>
                                <td></td>
                                <td><input type="submit" value="Save" /></td>
                        </tr>
                </table>
        </form>

</body>
</html>
-------------------------------------------------------------------------------
------------------
<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
        <p th:text="${msg}" />

        <a href="person">Go Back</a>

</body>
</html>
-------------------------------------------------------------------------------
-----------------------



++++++++++++++++++
Embedded Servers
++++++++++++++++++


-> Spring Boot provided embedded containers to run our web applications

-> When we add "web-starter" by default it is giving 'tomcat' as default
embedded container

-> In spring boot we have multiple embedded containers

                1) tomcat
```

2) jetty
3) netty
4) undertow etc...

Note: we can deploy spring boot application into external servers also as a war file.

Q) How change default container from tomcat to jetty ?

-> To make jetty as embedded container we need make below 3 changes in pom.xml file

1) Remove tomcat-starter dependency

2) Exclude tomcat-starter from web-starter

3) Add jetty dependency

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
	<exclusions>
		<exclusion>

	<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-tomcat</artifactId>
		</exclusion>
	</exclusions>
</dependency>

<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

---

1) What is Spring Web MVC ?
2) What is C2B & B2B ?
3) Spring Web MVC advantages
4) Spring Web MVC architecture
5) Front Controller (DispatcherServlet)
6) HandlerMapper
7) Controller
8) ViewResolver
9) View
10) Web MVC Annotations
- @Controller
- @GetMapping

- @PostMapping

11) ModelAndView
12) Model
13) Sending data from Controller to UI
14) Spring MVC Form Tag Library
15) Forms Development
16) Form Validations
17) Thymeleaf Introduction
18) Form development using Thymleaf

++
50 Interview Questions on Spring Web MVC :
++
https://www.youtube.com/watch?v=1_SsosC4Cs8&list=PLpLBSl8eY8jSMr1hJLB096nq8W0ABQoXH