```
===============
Spring Web MVC
===============

-> It is one module available in Spring Framework

-> Spring Web MVC is used to develop 2 types of applications

                        1) Web Applications  ( C 2 B )

                                Ex: Gmail, facebook, naukri

                        2) Distributed Applications ( B 2 B )

                                Ex:

                                                MakeMyTrip ---------> IRCTC

                                                Passport ----------> AADHAR


-> Web & Distributed applications developement made easy

-> Form Data Binding To Java Objects

-> Flexibility in Form Binding (Type casting will be done)

-> Form Validations (Server Validation)

-> Supports Multiple Presentation Technologies (Ex: Jsp & thymeleaf )

-> Embedded Servers ( Default: Tomcat )

=============================
Spring Web MVC Architecture
==============================

1) Dispatcher Servlet  : It acts as a front controller

2) HandlerMapper : It will identify which request should be processed by which controller and which
method

3) Controller : It will handle request and decides response to send using ModelAndView object.

4) ModelAndView : Model represents data in key-value format. View Represents logical file name to
display.

5) View Resolver : It is used to identify physical location of view files

6) View : It is used to render model data on view file.


===========================================
Building First Spring Web MVC Application
===========================================

1) Create Spring Boot application with below dependencies

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

2) Create Controller class using @Controller annotation
```

```
3) Write required methods in Controller class and bind them to Http Request Methods

4) Create Presentation file ( jsp ) with presentation logic

5) Configure View Resolver in application.properties file

6) Run the application and test it.

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


==================================
Query Paramters / Request Params
===================================

=> To send data to server in URL
=> Key Value format
=> will present At end of the URL
=> Starts with ? symbol
=> Will be seperated by & symbol

        URL : www.ashokitech.com/course?name=sbms&trainer=ashok

=> To Read query params we will use @RequestParam annotation

=================================Query Param Example=============================
@Controller
public class BookController {

        // http://localhost:8080/msg?name=ashok

        @GetMapping("/msg")
        public ModelAndView getMsg(@RequestParam String name) {

                String msgTxt = name + ", Good Evening";

                ModelAndView mav = new ModelAndView();
                mav.addObject("msg", msgTxt);

                mav.setViewName("index");

                return mav;
        }

        // http://localhost:8080/book?name=spring&author=johnson

        @GetMapping("/book")
        public ModelAndView getBookData(@RequestParam String name, @RequestParam String author) {

                System.out.println("Name :: " + name);
                System.out.println("Author ::" + author);

                ModelAndView mav = new ModelAndView();
                mav.addObject("msg", name + " By " + author + " is out of stock...");

                mav.setViewName("index");

                return mav;
        }
}
```

```
==================================
Path Paramters / URI Params
===================================

=> To send data to server in URL
=> It will represent data directley (no keys)
=> Can present anywhere in uRL
=> Starts with / symbol

        URL : www.ashokitech.com/course/sbms/

=> To Read query params we will use @PathVariable annotation

========================== Path Param Example =====================================
@Controller
public class CarController {

        // http://localhost:8080/car/101/hyd
        @GetMapping("/car/{carId}/hyd")
        public ModelAndView getCarColor(@PathVariable Integer carId) {
                ModelAndView mav = new ModelAndView();

                String color = null;

                if (carId >= 100) {
                        color = "Red";
                } else {
                        color = "Black";
                }

                mav.addObject("msg", "Car Color is :" + color);

                mav.setViewName("index");

                return mav;
        }

        // http://localhost:8080/stock/benz/location/hyd

        @GetMapping("/stock/{brand}/location/{loc}")
        public ModelAndView getCarStock(@PathVariable String brand, @PathVariable String loc) {

                ModelAndView mav = new ModelAndView();

                mav.addObject("msg", "In " + loc + " " + brand + " cars Out Of Stock");

                mav.setViewName("index");
                return mav;
        }

}
==============================================================================

We can develop controller methods in 2 ways
=============================================================================

1) By Taking ModelAndView as return type

2) By taking String as return type

====================================2 approches=================================================
@Controller
public class MyController {

        @GetMapping("/welcome")
        public ModelAndView getWelcomeMsg(@RequestParam String name) {

                String msgTxt = name + ", Welcome to Ashok IT..";

                ModelAndView mav = new ModelAndView();
                mav.addObject("msg", msgTxt);

                mav.setViewName("index");

                return mav;
        }


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

                model.addAttribute("msg", name+", Good Evening...!!");

                return "index";
        }
}
```

Note: Method return type is string which represents logical view name. Model is used to send data from controller to UI in key-value format.

```
==================
Forms development
==================
```

=> Forms are essential part in web applications

=> Forms are used to collect data from user to perform business operation

```
=================================== index.jsp ======================================
<%@ 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>


        <font color='green'>${msg}</font>
        <h3>User Form</h3>

        <form action="user" method="POST">

                <table>
                        <tr>
                                <td>Name:</td>
                                <td><input type="text" name="name" /></td>
                        </tr>
                        <tr>
                                <td>Email:</td>
                                <td><input type="email" name="email" /></td>
                        </tr>
                        <tr>
                                <td>Phno:</td>
                                <td><input type="number" name="phno" /></td>
                        </tr>
                        <tr>
                                <td></td>
                                <td><input type="submit" value="Submit" /></td>
                        </tr>

                </table>

        </form>

</body>
</html>
=====================================UserController.java=========================================

package in.ashokit.controller;

import javax.servlet.http.HttpServletRequest;

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

import in.ashokit.binding.User;

@Controller
public class UserController {

        @GetMapping("/")
        public String loadForm(HttpServletRequest req) {

                return "index";
        }

        @PostMapping("/user")
        public String handleSubmitBtn(User user, Model model) {

                System.out.println(user);

                // save in database

                model.addAttribute("msg", "User Saved");

                return "index";
        }
}
================================================================================================

=> Spring Web MVC module provided Form Tag library to simplify forms development

<form:form/>

<form:input/>

<form:password/>

<form:radiobutton/>

<form:select/>

<form:checkbox/>

=> To use web mvc form tag library we have to use below directive

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

```
=================================================================================================

Requirement : Develop student web application with below functionality

1) Student Registration - Store in Database

                                a) Name - textfield
                                b) Email - textfield
                                c) gender - radiobuttons
                                d) course - drop down
                                e) timings - checkboxes

2) View Students - display all registered students details in web page


=================================================================================================

1) Create SpringBoot application with below dependencies

                        a) web-starter
                        b) datajpa-starter
                        c) h2
                        d) devtools
                        e) jstl
                        f) tomcat-embed-jasper

2) Create Entity class & repository interface to store the data

3) Create Controller class to handle request & response

4) Create View Files using JSP

5) Configure below properties in application.properties / yml file

6) Run the application and test it.


Project Code : https://github.com/ashokitschool/Spring_WEB_MVC_FORM_APP

==========
Thymeleaf
==========

=> Thymeleaf we can use as presentation technology in Spring Web MVC based applications.

=> Thymeleaf is an alternate of JSP

=> JSP files can't be executed in browser directley. JSP page should be converted into Servlet for
execution.

=> Thymeleaf is a template engine which can be used in HTML pages directley

=> HTML pages will execute in browser directley.

                        HTML + Thymeleaf = Dynamic Web Pages

=>  Performance wise Thmyleaf pages are faster than jsp pages

=> Thmyleaf introduced to overcome the problems of JSP.


=> To use Thymleaf in boot application we need to add below starter

                        'spring-boot-starter-thymleaf'
```

```
======================================
Application Development with thymeleaf
======================================

1) Create boot app with below dependencies

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

2) Create Spring Controller with Required methods

3) Create Thymeleaf templates under src/main/resources/templates folder
                                (file extension is .html)

4) Run the application and test it.


Git Repo URL : https://github.com/ashokitschool/springboot_thyemleaf_app.git


=============================================
How to configure Jetty as Embedded Server ?
=============================================

1) Exclude starter-tomcat from starter-web dependency

2) Add jetty-starter in pom.xml file

        <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>


===============================================================================
How to send direct response from Spring Controller without using View Pages ?
==============================================================================

=> By using @ResponseBody annotation in Spring Controller class method we can send direct response to
client.

@Controller
public class MessageController {

        @GetMapping("/welcome")
        @ResponseBody
        public String getWelcomeMsg() {
                return "Welcome to Ashok IT..!!";
        }

        @GetMapping("/greet")
        public String getGreetMsg(Model model) {
                model.addAttribute("msg", "Good Evening");
                return "index";
        }
}
```

Note : @Controller + @ResponseBody = @RestController

=> @RestController is used to send direct response to client without any view files.

==
What is Interceptor in Spring Web MVC ?
==

-> We can use Interceptor to perform pre-processing and post-processing of every request

Pre-Processing : Before Request Procesing by Controller method

Post-Processing : After request processed by controller method

-> Using Interceptor we can trap each and every request

Use case for Interceptor
----
1) Calculate Each Request processing time

2) Log Each Request and Response details

3) Request Authentication etc...

==
Exception Handling in Spring Web MVC
==

-> Exception means un-expected and un-wanted situation

-> Exception distrubs normal flow of our application execution

-> When exception occurs then our program will terminate abnormally

-> As a developer we should handle exception to achieve graceful termination of our application.

-> To handle exceptions, Java provided below keywords

1) try
2) catch
3) throw
4) throws
5) finally

=> To handle Exceptions in Spring Web MVC application then we can create a method and we can use below annotation

@ExceptionHandler

=> When exception occurs then we will redirect user to error page like below.

```
@ControllerAdvice
public class GlobalExceptionHandler {

        @ExceptionHandler(value = Exception.class)
        public ModelAndView handleAE(Exception ex) {
                ModelAndView mav = new ModelAndView();
                mav.setViewName("page");
                return mav;
        }
}
```

Note : here error represents our error page which display some message to client to try after sometime.

=============
Requirement:
=============

1) Develop one To-Do task application. Application should contains below functionalities

        a) User Registration (Name, Email, Pwd, Gender & Phno)

        b) User Login (Email & Pwd)

        c) Create Task (Task Name, Date, Timing)

        d) View Tasks

Note: Task Creation & Display Tasks functionality should work based on Logged in user.

e) Logout

===

1) What is Spring Web MVC ?

2) Advantages of Spring Web MVC ?

3) Spring Web MVC Architecture

```
                                - DispatcherServlet
                                - HandlerMapper
                                - Controller
                                - ModelAndView
                                - ViewResolver
                                - View
```

4) What is Embedded Server (Ex : Tomcat & Jetty)

5) Building Web Application using Spring Boot

6) What is @Controller ?

```
7) What is @GetMapping & @PostMapping ?

8) What is Query Params and how to work with them ? ( @RequestParam )

9) What is Path Params and how to work with them ? ( @PathVariable )

10) Form Based Application Development

11) Spring Web MVC form tag library

12) What is Thymeleaf ?

13) Form Validation ( validation-starter )

14) What is @ResponseBody annotation ?

15) How to make jetty as default embedded server.

16) How to configure H2 Database (Embedded Database)

17) Web app development using => Web MVC + Data JPA + H2 DB
```