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

=> One of the most famous & important module in spring framework

=> Using Spring Web MVC we can develop 2 types of applications.

1) Web Applications (C2B)

2) Distributed Applications (B2B)

=================
Web Application
=================

=> Web Applications are used for Customer to Business Communication.

Ex: amazon, flipkart, naukri, ashokit

Note: In Web application we will have 3 components

1) Presentation Components (UI)

2) Business Components (Controllers + Services)

3) Data Access Components (Repositories)

Note: To develop presentation (UI) components in Spring Web MVC application we can use JSP and Thymeleaf.

=================================
What is Distributed application
=================================

=> Distributed applications are called as Webservices / Rest APIs.

=> Webservices are used to communicate from one application to another application.

```
ex: passport ----------> aadhar
    gpay ----------> sbi bank
    makemytrip ----> irctc
```

Note: In distributed appliations UI will not be available (pure backend apis).

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

1) Dispatcher Servlet

2) Handler Mapper

3) Controller / Request Handler

4) ModelAndView

5) ViewResolver

6) View

===================
DispatcherServlet

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

=> It is predefined class in spring web mvc

=> It acts as front controller (main gate of the house)

=> It is responsible to recieve request and send the response to client.

Note: It is also called as framework servlet class.

================
Handler Mapper
================

=> It is predefined class in spring web mvc

=> It is responsible to identify controller class to handle the request based on url-pattern and give controller class details to dispatcher servlet.

===========
Controller
===========

=> Controllers are java classes which are used to handle the request (request processing).

=> DispatcherServlet will call controller class methods.

=> After processing request, controller method will return ModelAndView object to dispatcher servlet.

Model -> It is a map to represent data in key-value format

View -> It represents view page name

===============
View Resolver
===============

=> It is used to identify view files location.

=> Dispatcher Servlet will give view name to View Resolver then it will identify the view file location and give it to Dispatcher Servlet.

========
View
========

=> It is responsible to render model data on the view page and give it to dispatcher servlet.

Note: DispatcherServlet will send final response to client.

===========================================
Developing First Spring Web MVC Based App
===========================================

Step-1 : Create boot app with below dependencies

a) web-starter

b) Thymeleaf

c) devtools

Step-2 : Create Controller class with required methods

```
Step-3 : Create View Page and access Model data in view page

        Views Location : src/main/resources/templates/

Step-4 : Run the application and test it using browser

--------------------------------------------------------------------
@Controller
@RequestMapping("/msg")
public class MsgController {

        // URL : http://localhost:8080/msg/greet

        @GetMapping("/greet")
        public ModelAndView greetMsg() {

                ModelAndView mav = new ModelAndView();
                mav.addObject("msg", "Good Morning..!!");
                mav.setViewName("index");

                return mav;
        }

        // URL : http://localhost:8080/msg/welcome

        @GetMapping("/welcome")
        public String welcomeMsg(Model model) {

                model.addAttribute("msg", "Welcome to Ashok IT..!!");

                return "index";
        }

}
--------------------------------------------------------------------
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Ashok IT</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-4Q6Gf2aSP4eDXB8Miphtr37CMZZQ5oXLH2yaXMJ2w8e2ZtHTl7GptT4jmndRuHDT"
crossorigin="anonymous">
  </head>
  <body>
    <h1 th:text="${msg}"></h1>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.6/dist/js/bootstrap.bundle.min.js"
integrity="sha384-j1CDi7MgGQ12Z7Qab0qlWQ/Qqz24Gc6BM0thvEMVjHnfYGF0rmFCozFSxQBxwHKO"
crossorigin="anonymous"></script>
  </body>
</html>
---------------------------------------------------------------------------


===================================================================================================
Assignment : Develop spring boot application to retreive users_table data from database and display
users data in html page in the table format. Develop this project using layered architecture.
=================================================================================================



@Controller + @ResponseBody = RestController

============================
What is Request Parameter ?
============================

=> Request Parameters also called as Query Parameters.

=> These are used to send data from client to server in URL.

=> Request Parameters will represent data in key-value format.

                Ex : https://www.youtube.com/watch?v=McmckGLzZ4Q&t=11700s

=> Request Parameters will start with ? and will be seperate by &.

=> To read Request Parameters from the URL we will use @RequestParam annotation.

----------------------------------------------------------------------------------
@Controller
public class MsgController {

        // URL : http://localhost:8080/greet?name=raj
        @GetMapping("/greet")
        @ResponseBody
        public String greetMsg(@RequestParam("name") String name) {

                String msg = name + ", Good Morning..!!";

                return msg;
        }

        // URL : http://localhost:8080/course?c=sbms&t=ashok
        @GetMapping("/course")
        @ResponseBody
        public String getCourse(@RequestParam("c") String course, @RequestParam("t") String trainer)
{

                String msg = course + " By " + trainer + " will start soon...";

                return msg;
        }
}
---------------------------------------------------------------------------------

==========================
What is Path Variable ?
==========================

=> Path Variables also called as URI variables and Path Parameters.

=> These are used to send data from client to server in URL.

                Ex : https://www.instagram.com/reel/{DJtyr-igaPD}/

Note: Path Variables will represent data directley without any key.

Note: Path Variables position we need to represent in URL template.

          Ex: @GetMapping("/greet/{name}")

=> To read path variables from URL we will use @PathVariable annotation.

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

// URL : http://localhost:8080/welcome/raj
        @GetMapping("/welcome/{name}")
        @ResponseBody
```

```
	public String getWelcomeMsg(@PathVariable("name") String name) {

		String msg = name + ", Welcome to Ashok IT";

		return msg;
	}
```

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

Path Variable : Used to uniquely identify a resource

Request Parameter : Used to filter, sort, or provide optional input.

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

=> We have below limitations with URL data

1) Data is exposing in browser URL (others can read it who are sitting beside us)

2) URL length limitation

3) Sensitive data we can't send in URL

4) Will not support for binary data (ex: images, videos, audios, files etc..)

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

=======================
What is Request Body ?
=======================

=> Using Request Body we can send data from client to server without exposing in URL.

=> Using Request Body we can send binary data also (ex: images, videos, audios, files etc..) to the server.

Note: When we are submitting forms then we willuse Request Body to send form data to Controller.

========================
What is Form Binding ?
========================

=> In Servlets, programmer is responsible to capture form data and store that in object manually.

```
// capture form data
String name = request.getParam("name");
String email = request.getParam("email");
String phno = request.getParam("phno");

// store in object
User user = new User();
user.setName(name);
user.setEmail(email);
user.setPhn(Long.parseLong(phno));
```

Note: The above is logic common for every form and for every project.

=> To avoid above problem Spring Web MVC introduced, Form Binding concept.

=> The process of binding Java object to form fields is called as Form Binding.

=> With the help of form binding we can map java object to form fields and form fields data to java object.

```
Note: If we use form binding concept then DispatcherServlet will take care of capturing form data and
mapping form data to java object.

=> To do form biding to java object we will use below properties in thymeleaf

                th:object ====> To represent which object mapping to form

                th:field =====> To represent which field is mapped to which variable in obj

=====================
Form Validations
=====================

=> Validations are used to verify users are entering correct data in the form or not before
submitting the form.

-> Form validations we can implement in 2 ways

                1) Client side validations

                2) server side validations

-> Client side validations will execute at browser level.

 # Advantage : we can stop invalid requests at browser only (no need to send to server)

 # Dis-Advantage: We can disable client side validations at brower using inspect element.

-> Server side validations will execute at code level. Nobody can disable it.


## Step-1 : Add validation starter pom.xml

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

## Step-2 : Use annotations at binding class to validate form data

                                        a) @NotEmpty
                                        b) @NotNull
                                        c) @Email

## Step-3 : Validate form data using @Valid annotation at controller method     and verify validation
errors using BindingResult.

-------------------------------
@PostMapping("/user-submit")
        public String handleSubmit(@Valid @ModelAttribute("user") UserDto user, BindingResult result,
Model model) {

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

                System.out.println(user);

                // TODO: logic to save user in db

                model.addAttribute("msg", "User Form Submitted");

                return "index";
        }
```

```
--------------------------------------
## Step-4 : Display validation errors in view page.

<p th:if="${#fields.hasErrors('email')}" th:errors="*{email}" class="text-danger" />



=============
Http Session
=============

=> Session is used to store logged in user data in the application to access that in multiple places.

=> When user logged in,  then session obj will be created and we will store user data in the session.

Note: Session data we can access across the application.

=> For every user one session object will be created.

=> When user logout from the application then we will remove session object from the application.

Usecase : Session is used in the applications to display data based on logged in user.

Ex : user dashboard, user-personal-details, user-education-details, user-enrolled-courses etc.

--------------------------------------------------------------------------------
@Controller
public class UserController {

        @GetMapping("/")
        public String index(Model model) {

                UserDto userDto = new UserDto();
                model.addAttribute("user", userDto);

                return "index";
        }

        @PostMapping("/login")
        public String login(@ModelAttribute("user") UserDto user, HttpServletRequest req, Model
model) {

                String email = user.getEmail();
                String pwd = user.getPwd();

                if (email.equals("admin@gmail.com") && pwd.equals("admin@123")) {

                        // create new session and store obj
                        HttpSession session = req.getSession(true);
                        session.setAttribute("email", email);

                        return "redirect:dashboard";
                } else {
                        model.addAttribute("msg", "Invalid Credentials");
                        return "index";
                }
        }

        @GetMapping("/dashboard")
        public String buildDashboard(HttpServletRequest req, Model model) {

                // get existing session and retrieve obj
                HttpSession session = req.getSession(false);
                String email = (String) session.getAttribute("email");
                model.addAttribute("email", email);

                // TODO : Get Courses purchased by user based on email

                return "dashboard";
        }

        @GetMapping("/edu-details")
        public String ed(HttpServletRequest req, Model model) {

                // get existing session and retrieve obj
                HttpSession session = req.getSession(false);
                String email = (String) session.getAttribute("email");

                model.addAttribute("email", email);

                // TODO : Get education details based on email

                return "dashboard";
        }

        @GetMapping("/logout")
        public String logout(HttpServletRequest req, Model model) {

                // get existing session
                HttpSession session = req.getSession(false);

                // remove session
                session.invalidate();

                return "redirect:/";
        }
}
-------------------------------------------------------------------------
```

```
======================================================
How to configure jetty as default embedded container?
======================================================
```

### Step-1 : Exclude tomcat from 'web-starter' in pom.xml

```
<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>
```

### Step-2 : Configure jetty starter in pom.xml

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

Note-1 : Choose Tomcat if you're building standard Java EE apps with JSP, and want stability with strong community support.

```
Note-2 : Choose Jetty if you need a lightweight, embeddable, fast-starting server for microservices
or modern cloud-native apps.


================================
Email sending with Spring Boot
================================

=> To send emails we need SMTP properties

                SMTP = Simple mail transfer protocol

Note: For practice purpose we can use gmail smtp properties.

Note: We need to generate gmail "app password" for SMTP authentication purpose.

                        URL : https://myaccount.google.com/apppasswords

################### App pwd : yvrn epas jjnk ksoc ###################


## Step-1 : Add "mail-starter" in pom.xml

## Step-2 : Configure smtp properties in application.properties file

                        spring.mail.host=smtp.gmail.com
                        spring.mail.port=587
                        spring.mail.username=ashokit.classes@gmail.com
                        spring.mail.password=yvrn epas jjnk ksoc
                        spring.mail.properties.mail.smtp.auth=true
                        spring.mail.properties.mail.smtp.starttls.enable=true

## Step-3 : Use JavaMailSender to send emails.

                                javaMailSender.send(Message)

Note: We have 2 types of msgs to send emails

                        1) SimpleMailMessage (plain text)

                        2) MimeMessage  (html body, attachments)


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


@Service
public class EmailService {

        @Autowired
        private JavaMailSender mailSender;

        public boolean sendEmail(String to, String subject, String body) {
                boolean isSent = false;
                try {
                        /*
                        SimpleMailMessage msg = new SimpleMailMessage();
                        msg.setTo(to);
                        msg.setSubject(subject);
                        msg.setText(body);*/

                        MimeMessage msg = mailSender.createMimeMessage();

                        MimeMessageHelper helper = new MimeMessageHelper(msg);
                        helper.setTo(to);
```

```
				helper.setSubject(subject);
				helper.setText(body, true);

				//helper.addAttachment("Report", new File("path-of-file"));

				mailSender.send(msg);
				isSent = true;
			} catch (Exception e) {
				e.printStackTrace();
			}
			return isSent;
		}
}
```

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

1) @Controller

2) @GetMapping

3) @PostMapping

4) @RequestParam

5) @PathVariable

6) @ResponseBody

7) @ModelAttribute

8) @Valid

9) @NotEmpty

10) @NotNull

11) @Email

12) @Size

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