```
Git Repo URL : https://github.com/ashokitschool/SBMS-39.git


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

-> Spring Web MVC is one module in spring framework.

-> Using Spring Web MVC, we can develop below types of applications

                1) Web Application ( C 2 B )

                2) Distributed Application ( B 2 B )

-> We need to use 'springboot-starter-web' depenency to work with Spring Web MVC module.

-> 'web-starter' will provide embedded container by default. We no need to setup server manually.

                                - Apache (Default)
                                - Jetty
                                - Netty

-> Spring Web MVC supports multiple presentation technologies

                        Ex: JSP, Thymeleaf...

-> Spring Web MVC supports Form Binding. form data will be mapped to java object.

Note: When we develop java web app using servlets we need capture form data like below.

                String phno = request.getParameter("phno");
                Long ph = Long.parseLong(phno);

Note: We no need to write this logic in web mvc. It will take care of capturing form data and
convert into corresponding data type and store into java object.

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

=> In spring web mvc, below components will be involved...

1) DispatcherServlet : front controller/framework servlet

2) HandlerMapper : To identify request handler (controller)

3) Controller : Request Handler (spring bean) - we have to develop

4) ModelAndView : Model represents data & view represents UI page.

5) ViewResolver : To identify where view pages available in app

6) View : To render model data on view page.

=========================================
Building First Web App using Spring Boot
==========================================

1) Create Boot app with below dependencies

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

2) Create Controller class with required methods and map methods to HTTP methods with unique url
patterns.
```

3) Create View Page (HTML + Thymeleaf) (under templates folder)

4) Run the application and test it.

```
----------------- Controller -------------------------------

@Controller
public class MsgController {

        @GetMapping("/greet")
        public ModelAndView getMsg2() {

                ModelAndView mav = new ModelAndView();
                mav.addObject("msg2", "Good Morning...!!");
                mav.setViewName("index");

                return mav;
        }

        @GetMapping("/welcome")
        public ModelAndView getMsg1() {

                ModelAndView mav = new ModelAndView();
                mav.addObject("msg1", "Welcome to Ashok IT");
                mav.setViewName("index");
                return mav;
        }
}

------------------------index.html------------------------------
<html>
        <body>
                <p th:text="${msg1}"></p>
                <p th:text="${msg2}"></p>
        </body>
</html>
----------------------------------------------------------------
```

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

1) Develop Spring Boot web app to retrieve products data from db table and display in UI page as a table.

=========================
Form Based Applications
=========================

=> In every web app many forms will be available

- login form
- register form
- search form

=> We need to capture form data and we need to perform operation on that data...

Note: Web MVC supports form binding.

===
Develop Boot web app to save and retrieve products data
==

1) Product.java (form binding + entity)

```
                Integer pid; (PK, Auto_Increment)
                String pname;
                Double price;
                Integer qty;

2) ProductRepo.java (JpaRepository)

3) ProductService.java

                - public boolean saveProduct(Product p);

                - public List<Product> getProducts( );

4) ProductController.java

                public ModelAndView loadForm( ); - GET

                public ModelAndView saveProduct(Product p) - POST

                public ModelAndView getAllProducts( )  - GET

5) View Pages

                index.html - form to enter data
                data.html  - table to display data

=================================
Embedded Database in spring boot
=================================

=> Embedded databases are called temporary databases

=> H2 we can use as embedded database

=> When application starts h2 db will start and when application stopped h2 db also gets stopped.

Note: If application re-started then we will loose old data.

=> H2 db is used only for practice purpose.

====================================
How to use H2 DB in spring boot ?
====================================

### Step-1 : Add h2 dependency in pom.xml file

                <dependency>
                        <groupId>com.h2database</groupId>
                        <artifactId>h2</artifactId>
                        <scope>runtime</scope>
                </dependency>

### Step-2 : Configure H2 datasource properties in application.properties file


spring.datasource.username=ashokit
spring.datasource.password=abc
spring.datasource.url=jdbc:h2:mem:sbms

spring.jpa.show-sql=true

server.port=8081


### Step-3 : Run the application and access h2-console in browser

                URL : http://localhost:port/h2-console
```

```
============================================
How to change default container to  jetty ?
============================================

=> When we add web starter then tomcat will become default embedded container to run boot
application.

=> If we want to change from tomcat to jetty then we need to make below changes in pom.xml

### Step-1 : Exclude tomact from web-starter 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>

### Step-2 : Add jetty starter

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

### Step-3 : Run the application and observe console.


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

Develop  Spring Boot web application with below functionalities

        Registration Page : Name, Email, Pwd and Phno fields

        Login Page : Email & Pwd fields

        Dashboard Page : <msg>

Note: When user registered, application should send an email to the user.

Email Subject : Account Created - Ashok IT

Email Body : Congratuations.. you are onboard..


=================================
Email Sending using spring boot
=================================

=> To send emails using spring boot we have add 'mail-starter' dependency in pom.xml

=> We need to configure SMTP properties in application.properties file

Note: In SMTP props, we need to our gmail account credentials for authentication purpose.

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

        URL To generate app pwd: https://g.co/kgs/f1ic3P9

=> Spring boot provided JavaMailSender to send emails
```

```
                - SimpleMailMessage (plain text)

                - MimeMessage   (formats, attachments)


============================================
Exception Handling in Boot web application
============================================

=> Exception : Unexpected and unwanted situation

=> When exception occurs program will be terminated abnormally

=> We need to handle exceptions for app graceful termination.

=> We have below keywords to handle exceptions in java

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


=> In spring boot we can handle exceptions in 2 ways

                1) Controller/class Based (specific to class)

                2) Global Exception Handling (entire application)

================================================================
@ControllerAdvice
public class AppExceptionHandler {

        @ExceptionHandler(value = Exception.class)
        public String handleAe(Exception e) {
                // logic
                return "exView";
        }
}
==================================================================

1) What is web mvc ?
2) Advantages with Web MVC
3) Web MVC Architecture
4) What is Embedded Container
5) How to develop boot web app
6) How to send data from controller to UI
7) Web MVC Form with Form Binding
8) Embedded Database (h2)
9) How to change default container
10) Email Sending using Spring Boot
11) Exception Handling in Web MVC
12) Login & Registration app
13) Product Store App (CRUD Ops)

===================================
Query Parameters (key-value)
===================================

=> Query Params are used to send data to server in URL

=> Query Params will represent data in key-value format

=> Query params will start with ?

=> Query Params will be seperated by &
=> Query Params will present at end of the URL

        ex: www.youtube.com/watch?v=ljsdf79/

                www.ashokit.in/course?name=sbms

                www.ashokit.in/course?cname=sbms&tname=ashok

Note: We can read query parameters from URL using @RequestParam annotation in the controller.

================
Path Parameters
================

=> Path Params are used to send data to server in URL

=> Path Parameters will represent value directley

        ex : www.youtube.com/c/AshokIT

=> Path Parameters will be seperated by '/'

=> Path Parameters can present anywhere in the URL

        Ex :  www.ashokit.in/course/{java}/info

=> We can read Path Parameters using @PathVariable annotation.


=========================
What is @ResponseBody ?
=========================

=> It is used to send direct response to client without any view page.

=> This can be used at controller class level and method level

Note: If we use at class level then it is applicable for all methods in that class

                @Controller + @ResponseBody = @RestController


==================================================================
@Controller
public class MsgController {

        @GetMapping("/welcome")
        @ResponseBody
        public String welcomeMethod(@RequestParam("name") String name) {
                return name + ", Welcome to Ashok IT";
        }

        @GetMapping("/greet/{name}")
        public String greetMethod(@PathVariable("name") String name, Model model) {
                model.addAttribute("msg", name+", Good Morning");
                return "index";
        }
}


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

=> To restrict users to provide valid information in the form

                        - Client Side Validations

                        - Server side validation
```

=> Client side validations will execute at browser. People can disable client side validations using inspect option in browser.

=> Server side validations will execute in our code. These are highly recommended in application.

=> To implement server side validations we will use below starter in pom.xml file

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

=> We can use below annotations to perform server side validations

```
@Valid
@NotEmpty
@NotNull
@Size
@Email
```

Note: We will use these annotations at binding class.

```
        @PostMapping("/user")
        public String handleSubmit(@Valid User user, BindingResult result, Model model) {

                if (result.hasErrors()) {
                        // validation failed
                        return "index";
                } else {
                        // validation passed
                        System.out.println(user);
                        // logic to save in db
                        model.addAttribute("msg", "User Saved");
                        return "index";
                }
        }
```

==

Requirement : Develop spring boot web application to upload and download files.

===

======================
Spring Boot with JSP
======================

=> JSP stands for Java Server Pages

=> JSP is used to develop presentation layer

Note: JSP will be translated to Servlet for execution..

=> Spring Web MVC supports JSP as presentation technology.

## Step-1 : Add tomcat-embed-jasper dependency in pom.xml file

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

## Step-2 : Create jsp pages in below location

```
                Location : src/main/webapp/pages/index.jsp
```

## Step-3 : Configure view resolver in application.properties file

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

=========================
Actuator in spring Boot
=========================

=> Used to monitor and manage our spring boot applications

=> Production ready features...

=> With the help of actuators we can get below details

- Health of App
- Beans loaded
- Metrics
- Loggers
- URL Mappings
- Config Props
- Thread Dump
- Heap Dump

=> To work with actuators we need to add below dependency

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

Note: With above dependency, By default /health will be exposed.

=> We need to write below property to expose other endpoints

```
        management.endpoints.web.exposure.include=*
```

=> We can exclude actuator endpoint like below

```
        management.endpoints.web.exposure.exclude=beans
```

==============
Endpoint URLS
==============

/health : health of the app (UP or DOWN)

/beans : Spring Beans loaded

/configprops : Properties loaded

/mappings : URL patterns of our application

/threaddump : Threads info

/heapdump : Heap info

/loggers : Logs of our applications

/shutdown : Stop server (HTTP POST Request)

=============================
What is shutdown endpoint ?
=============================

=> It is used to stop the application.

Note: We need to enable shutdown endpiont in our properties file like below

```
                management.endpoint.shutdown.enabled=true
```

Note: Shutdown endpoint is mapped to POST request. We can send post request using POSTMAN software.

===================================
What are Profiles in Spring Boot ?
===================================

=> Environment means a platform which is used to run our application.

=> In Real-time one application contains multiple environments like below

- Local
- Dev
- QA
- UAT
- PILOT
- PROD

-> Local env is used for development purpose

-> DEV env is used by developers for integration testing

-> QA env is used by Testing team for system integration testing

-> UAT env is used by Client side team for testing (GO/No-GO)

-> PILOT env is used to test app with live data (Pre-Prod)

-> PROD env is used for live access.

=> Below properties will be changing from environment to environment.

- datasource properties
- smtp properties
- kafka properties
- redis properties
- payment-gateway properties

=> If we use single application.properties file to maintain config properties then maintanence will become difficult.

Note: to deploy code into env, everytime we have to change config props

=> To avoid this problem we will use Profiles in springboot

=> Using profiles we can maintain environment specific configuration.

```
                        application.properties   ---- main file

                        application-dev.properties

                        application-qa.properties

                        application-uat.properties

                        application-prod.properties
```

=> We need to activate profile in main configuration file

```
                        spring.profiles.active=dev
```

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

1) Develop Java application to convert java object to json and json data to java object.

2) Develop Java application to convert java object to xml and xml data to java object.

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