SpringBoot-MVC

1. This module is given by SpringBoot to build webapplications.
2. Using this module we can build webapplication with any view as per our choice.
   eg: View can be JSP,Thymleaf,Velocity,FreeMarker,....
3. This module of Spring Supports the user in the following ways.
   a. Maven project creation step.
   b. pom.xml dependencies and plugins.
   c. Writing common code(AppInit,AppConfig)
   d. Handle runtime environment and creating jars/wars

4. The above mentioned steps are handled by SpringBoot through "AutoConfiguration".
5. In SpringMVC we have 3 components
   a. DispatcherServlet[FrontController]
   b. HandlerMapper[Informs about which component to handle]
   c. ViewResolver[Informs about which view to execute for a particula request]

6. In the above 3 components as a programmer we need to handle only 2 things
   a. HanlderMapper -> Provide the information through prefix
   b. ViewResolver -> Provide the information thorugh suffix
   c. DispatcherServlet -> it is autoconfigured in SpringBoot and mapped to the URL called "/".

7. FC,ViewResolver and HandlerMapper configuration are taken care by SpringBoot,Controller and UI files should be defined by the programmer.

8. DataRendering -> Reading the data from the Model and sending it to UI is known as "DataRendering", It is implemented using
   "EL" programming.

9.Programmer should provided inputs like portno, viewresolver details using properties file

application.properties

```
server.port = 9999
spring.mvc.view.prefix = /WEB-INF/views/
spring.mvc.view.suffix = .jsp
```

=> Default port no mapped to 8080 by using server.port
=> we can change the port no using the key as shown, server.port = 9999
=> SpringBoot has provided 3 server internally
   a. tomcat(default server)
   b. jetty
   c. undertow
=> In general tomcat server comes with 2 engine namely
   a. Servlet Engine(servlet)
   b. jasper engine(JSP)
=> In springboot,tomcat comes only with "Servlet Engine", That's whsy it is also called as "LightWeight" that works only for "DispatcherServlet".

=> While writing the webapplication, we need to keep the static resources like html,css,images inside
   src/main/resources folder.

=> To work with JSP in SpringBoot we need to add the following dependancy in pom.xml file

```
      <!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-
jasper -->
      <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope> runtime </scope>
      </dependency>

=> Springstarter file for Spring-mvc is spring-boot-starter-web
      <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>2.2.2.RELEASE</version>
      </dependency>


Note:
 tomcat   :: Apache
 jetty        :: Eclipse
 undertow :: JBoss

++++++++++++++++++++++++++++++
Control flow in SpringBoot-MVC
++++++++++++++++++++++++++++++
 Note: Start the server and keep the application ready

 1. Client will send the request
 2. FrontController(DispatcherServlet) takes the request(URL + HTTPRequestType) and
hands it over to HandlerMapper.
 3. HandlerMapper will identify which component to execute for the given URL +
HTTPRequestType(class+ methodName)
 4. HanlderMapper send the details to DispatcherServlet,Not FC will execute the
method to generate the response.
 5.
      5a.While executing the method the controller may keep the information
required for view in Model.
        5b. After keeping the data inside the model, the controll will go back to
DispatcherServlet.

 6. Now the Dispatcher servlet will recive the viewName, this information it will
give it to "ViewResolver".
 7. ViewResolver will give the following information
       => location + viewName + extension
      location : /WEB-INF/view/
      extension: .jsp
      filename : sample
      eg::  /WEB-INF/view/sample.jsp

 8. Using the ViewResolver information, Dispatcher servlet will pick the suitable
view
 9.
      9a. If need the view component will take the model data from the model.
      9b. After picking the data, now view will send the information to
DispatcherServlet.

 10. DispatcherServlet will make the view ready for execution(sending the response
to the client).
```

```
++++++++++++++++++++++++
FirstProgram control flow
++++++++++++++++++++++++
 a. request : http://localhost:9999/show
 b. /show ----> send to handlermapper
 c. handlermapper -> EmployeeController + showPages()
 d. DispactherServlet -> EmployeeController.showPages() ----> "home".
 e. ViewResolver -> /WEB-INF/view/home.jsp
 f. DispatcherServlet -> /WEB-INF/pages/home.jsp ---> rendering to client

            refer: SpringBootMVC-01

+++++++++++++++++++++++++++
SecondProgram control flow
+++++++++++++++++++++++++++
  a. request : http://localhost:9999/MYAPP/emp/show
  b. /show  ---> send to handlermapper
  c. handlermapper -> EmployeeController + showPages()
  d. DispatchereServlet -> EmployeeController.showPages()
                           |-----> 1st -> keep the data in model with key called
"msg".
                           |-----> 2nd -> send the view name as "home".

  e.ViewResolver -> /WEB-INF/view/home.jsp
  f.DispatcherServlet -> /WEB-INF/pages/home.jsp
                         ---> 1st -> Collect the data from the model using the key
called "msg".
                           ---> 2nd -> rendering to client


application.properties
======================
server.port = 9999
spring.mvc.view.prefix= /WEB-INF/view/
spring.mvc.view.suffix= .jsp
server.servlet.context-path=/MYAPP

EmployeeController.java
=======================
@Controller
@RequestMapping("/emp")
public class EmployeeController {
      @GetMapping("/show")
      public String showPages(Model model) {
            model.addAttribute("msg", "Welcome to PWSKILLS :: " + new
java.util.Date());
            return "home";
      }

}

Note: To retrieve the data from the request object we need to use an annotation
called "@RequestParam".

@PostMapping("/login")
public String validateCredentials(
                  @RequestParam("username") String username,
                  @RequestParam("userpassword") String password,
                  Model model) {
```

```
            ......
}

Model         -> To send the data from Controlle to UI.
@RequestParam -> To collect the data from the UI inside the Controller.
```