Main Elements of Spring MVC

Presentation Layer Components

1. JSP/EL/JSTL
2. Spring Form Tag Library
3. Message Bundles
4. Commands
5. Command validetors
6. View Resolvers

Controller Layer Components:

1. DispatcherServlet
2. HandlerMappings
3. Handler Interceptors
4. Controllers

Presentation Layer Components

1. JSP/EL/JSTL

You have already studied in JSP Technology.

2. Spring Form Tag Library

<form:form/>	<form:input/>	<form:label/>	<form:password/>
<form:hidden/>	<form:checkbox/>	<form:checkboxes/>	<form:errors/>
<form:option/>	<form:options/>	<form:radiobutton/>	<form:radiobuttons/>
<form:select/>	<form:textarea/>		

3. Message Bundles

- You can centralize Labels and Error Messages in message bundle.
- You can write one or more message bundles in your web application

  Syntax:

  basename_LanguageISOCode_CountryISOCode.properties
  basename_LanguageISOCode.properties

- You need to register **ReloadableResourceBundleMessageSource** with basename.

```
@Bean
public MessageSource messageSource() {
    ReloadableResourceBundleMessageSource messageSource = null;
    messageSource = new ReloadableResourceBundleMessageSource();
    messageSource.setBasename("classpath:messages");
    messageSource.setDefaultEncoding("UTF-8");
    return messageSource;
}
```

Using Labels:

A) Define the Labels in message bundle:

Ex:

messages.properties

```
jlc.title=JLC(EN)
```

messages_hi.properties

```
jlc.title=JLC(HI)
```

B) Use the required Label with key in JSP using Spring Tag Library tags as follows.

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

<spring:message code="jlc.title" text="Java Learning Center" />
```

Using Errors:

A) Define the Errors in message bundle:

messages.properties

```
errors.username.required=Username is Required.
errors.required={0} is Required.
errors.length={0} length must be between {1} and {2}
```

B) Use the key In validate() method

```
errors.rejectValue("username","errors.username.required",null,"Username is Mandatory");
```

Displays - Username is Required

```
errors.rejectValue("username","uname.errors.required",null,"Username is Mandatory");
```

Displays - Username is Mandatory

```
errors.rejectValue("username","errors.required",Object []{"UName"},"Username is Mandatory");
```

Displays - UName is Required

```
errors.rejectValue("username","errors.length",new Object[]{"Username","5","9"},"Defalut value");
```

Displays - Username length must be between 5 and 9

```
errors.rejectValue("password","errors.length",new Object[]{"Password","4","8"},"Defalut value");
```

Displays - Password length must be between 4 and 8

C) Display erros In JSP

```
<form:errors path="username"/>
<form:errors path="password"/>
```

4. Commands

- Command is mainly used to store the client submitted data.
- Command class is simple Java Bean Class with private variables and public getter and setter methods.
- Client submitted data will be stored in command object by calling setter methods.
- Data in the command object will be collected by calling getter methods and will be populated into JSP form elements.

5. Command Validators

- Write the validations required for your Command data in a separate Validator class with the following Steps:
  - Write your validator class by implementing Validator interface which is available in org.springframewok.validation package.
  - Mark the custome validator class with @Component, So that it can be injected in controller class.
  - Override the following two methods
    a) boolean supports(Class cls)
    b) void validate(Object command,Errors errors)
  - Write the code inside the supports() method to check whether correct command is used or not
  - Write the code inside the validate() method to validate input data.
  - When any input data is voiliting rules then add the errors messages using the following methods:
    a) void rejectValue(String,Streing,Object[],String)
    b) void rejectValue(String,Streing)

6. View Resolvers

- Spring MVC provides the ViewResolver which resolve view logical name to actual views.
- Following are the list of View Resolver provided in Spring MVC:
  A) InternalResourceViewResolver;
  B) ResourceBundleViewResolver;
  C) XmlViewResolver;
  D) CommonsMultipartResolver
  E) CookieLocaleResolver
  F) ContentNegotiatingViewResolver;
  G) BeanNameViewResolver;
  H) UrlBasedViewResolver;
  I) FreeMarkerViewResolver;
  J) VelocityViewResolver;
  K) JasperReportsViewResolver;
  L) XsltViewResolver;
  Etc

A) InternalResourceViewResolver

```
@Bean
public InternalResourceViewResolver viewResolver() {
InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
viewResolver.setViewClass(JstlView.class);
viewResolver.setPrefix("/WEB-INF/myjsps/");
viewResolver.setSuffix(".jsp");
return viewResolver;
}
```

B) ResourceBundleViewResolver

```
@Bean
public ViewResolver resourceBundleViewResolver() {
   ResourceBundleViewResolver bean = new ResourceBundleViewResolver();
   bean.setBasename("views");
   return bean;
}
```

views.properties

```
hello.(class)=org.springframework.web.servlet.view.JstlView

hello=/WEB-INF/myjsps/hello.jsp
show=/WEB-INF/myjsps/show.jsp
```

C) XmlViewResolver

```
@Bean
public ViewResolver xmlViewResolver() {
   XmlViewResolver bean = new XmlViewResolver();
   bean.setLocation(new ClassPathResource("views.xml"));
   return bean;
}

<bean id="xmlConfig" class="org.springframework.web.servlet.view.JstlView">
   <property name="hello" value="/WEB-INF/view/hello.jsp" />
   <property name="show" value="/WEB-INF/view/show.jsp" />
</bean>
```

Chaining ViewResolvers and Define an Order Priority:

- Spring MVC also supports multiple view resolvers.
- We can simply chain view resolvers by adding more than one resolver to the configuration.
- One you define multiple view resolvers , then you need to define an order for these resolvers.
- The order property is used to define which the order of invocations in the chain.
- The higher the order property (largest order number), the later the view resolver is positioned in the chain.
- To define the order we can add the follow line of code to the configuration of the our view resolvers:

  bean.setOrder(0);

Controller Layer Components

1. **Dispatcher Servlet**
   - DispatcherServlet is the Front Controller Component which is responsible for the following:
     a) Receives the incoming request
     b) Process the request completely
     c) Deliver the response.

2. **Handler Mappings**
   - Handler Mappings are responsible for identifying corresponding controller for incoming request URI.
   - All the Handler Mapping class are the subclass of HandlerMapping interface.
   - Following are the various Handler Mappings provided:
     1) BeanNameUrlHandlerMapping
     2) RequestMappingHandlerMapping
     3) SimpleUrlHandlerMapping
     4) ControllerBeanNameHandlerMapping
     5) ControllerClassNameHandlerMapping

A) BeanNameUrlHandlerMapping

- This is the default Handler Mappings which will be registered by the Spring Container automatically.
- This Handler Mapping checks whether any bean available whose name is same as incoming request URI.

B) RequestMappingHandlerMapping

- RequestMappingHandlerMapping scans all @RequestMapping annotations in all controller classes.
- It provides a method setInterceptors() to add interceptors.

3. Handler Interceptors

- Handler Interceptors will be invoked before and after the controller invocation for performing pre-propcessing and post-propcessing tasks.
- You can have one or more Handler Interceptors in the application.

Refer Page No: 265

4. Controllers

- Controller is the last component in the Controller Layer from where Business Layer starts.
- Controller is mainly responsible for the following:
    1) Collects the client submitted data from the Command object
    2) Calls the Business Layer Components
    3) If Exception is occurred then Propagate to DS directly
    4) If data is returned successfully without any error then store that in any scope and return View logical name.
- Only one instance will be created per Controller for all the users.
- When you are writing Controller class with XML Configuration then you need to do the following steps:
    1) Write you controller class by extending one of the following Built-In Controllers:
        a) AbstractController
        b) SimpleFormController (Removed in Spring 4)
        c) MultiActionController
        d) AbstractWizardFormController (Removed in Spring 4)

        etc
    2) Override the required methods in your Controller class:
    3) Configure the controller with the bean name in Spring Configuration file.

- When you are writing Controller class with Annotation Configuration then you need to do the following steps:

1) Write you controller class by without extending Built-In Controllers.
2) Mark the controller class with @Controller annotation.

```
@Controller
public class LoginController {
        …
}
```

3) Implement the required method in the controller class for processing the incoming request.
4) Mark the controller methods with @RequestMapping annotation or with Annotations based on Incoming Request Http Method like @GetMapping, @PostMapping etc by specifying the required URI

```
@RequestMapping(value="/login", method = RequestMethod.GET)
@GetMapping("/login",)
public String showLoginForm(Model model) throws ServletException {

return …;
}
@RequestMapping(value = "/verifyUser", method = RequestMethod.POST)
@PostMapping("/verifyUser")

public String verifyUser(@ModelAttribute("user") User user…….) {
        return …;
}
```

5) No need to configure the Controller as Bean in Spring Configuration Class.
6) You need to specify the @ComponentScan annotation with the List of Packages to Scan

```
@ComponentScan({ "com.coursecube.spring " })
```

Various Method Signatures of Controller class

a) When you are requesting for the resource then the controller method which you are invoking can have the following method signature:

String showLoginPage ()

String showLoginPage (Map)

String showLoginPage (Model)

String showLoginPage (HttpServletRequest)

String showLoginPage (HttpServletResponse)

String showLoginPage (HttpSession)

String showLoginPage (Model,HttpServletRequest)

String showLoginPage (Map, HttpServletRequest)

String showLoginPage (Model,HttpServletResponse)

String showLoginPage (Map,HttpServletResponse)

String showLoginPage (Model,HttpSession)

String showLoginPage (Map,HttpSession)

String showLoginPage (HttpServletRequest, HttpServletResponse)

String showLoginPage (HttpServletRequest, HttpServletResponse, HttpSession)

String showLoginPage (Map,HttpServletRequest, HttpServletResponse)

String showLoginPage (Model,HttpServletRequest, HttpServletResponse)

b) When you are submitting a JSP Form then the controller method which you are invoking can have the following method signature

String verifyUser (User , Errors,)

String verifyUser (User , Errors, Model)

String verifyUser (User , Errors, Map)

String verifyUser (User , Errors, HttpServletRequest)

String verifyUser (User , Errors, HttpServletResponse)

String verifyUser (User , Errors, Map,HttpServletRequest)

String verifyUser (User , Errors, Model,HttpServletRequest)

String verifyUser (User , Errors, HttpServletRequest, HttpServletResponse)

String verifyUser (User , Errors, HttpServletRequest, HttpServletResponse)

String verifyUser (User , Errors, Map ,HttpServletRequest, HttpServletResponse)

String verifyUser (User , Errors, Model ,HttpServletRequest, HttpServletResponse)

File Upload

1) Copy the following jars to WEB-INF/lib directory.
   a) commons-fileupload-1.3.1.jar
   b) commons-io-2.4.jar
2) Configure the bean with the class CommonsMultipartResolver by specifying the information about the file that will be uploaded.

```
@Bean
public CommonsMultipartResolver multipartResolver() {
        CommonsMultipartResolver vr = new CommonsMultipartResolver();
        vr.setMaxUploadSize(500000);
        return vr;
}
```

3) Use the form tag with enctype="" in HTML/JSP.

```
<form method="POST" action="uploadfile.jlc" enctype="multipart/form-data">
```

4) Use the <input type='file' .../> for file selection.

```
<input type="file" name="file"/>
```

5) Implement the required method in the controller class and mark that with following

```
@PostMapping("/uploadfile.jlc")
```

6) Define the MultipartFile as parameter for the method to collect the uploaded file data.
   a) For Single file

```
MultipartFile file
```

   b) For Multiple Files

```
MultipartFile[] files
```

7) Define the RequestParam annotation with MultipartFile.
   a) For Single file

```
@RequestParam("file") MultipartFile file
```

   b) For Multiple File

```
@RequestParam("file") MultipartFile[] files
```

8) Controller class will looks like

```
@Controller
public class FileUploadController {
    @PostMapping("/uploadfile.jlc")
    public String uploadFile(@RequestParam("file") MultipartFile file,HttpServletRequest req) {

        ...
        return "success";
    }
}
```

Lab74: Files required

1. index.jsp	2. FileUploadController.java
3. JLCWebConfig.java	4. JLCWebAppInitializer.java

1. index.jsp

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

<html>
<body>
<c:if test="${Upload  eq 'TRUE'}">
<h2> File uploaded successfully to ${Path} </h2>
</c:if>
<c:if test="${Upload  eq 'FALSE'}">
<h2>Error in file upload -- ${ErrMsg} </h2>
</c:if>
<form  action="uploadFile" method="POST" enctype="multipart/form-data">
<table>
<tr>
<td>Student Name</td>
<td> <input type="text" name="sname"/></td>
</tr>
<tr>
<td>Select File</td>
<td><input type="file" name="myfile"/></td>
</tr>
<tr>
<td><input type="submit" value="Upload"/> </td>
</tr>
</table>
</form>
</body>
</html>
```

2. FileUploadController.java

```
package com.coursecube.spring;

import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
/*
* @Author : Srinivas Dande
```

```
* @company : Java Learning Center
* */
@Controller
public class FileUploadController {

@GetMapping("/")
public String showIndexPage() {
System.out.println("---------showIndexPage()---------");
return "index";
}

@PostMapping("/uploadFile")
public String uploadFile(@RequestParam("sname") String sname, @RequestParam("myfile")
MultipartFile myfile,HttpServletRequest req) {
System.out.println("---------uploadFile()--------by--"+sname);

if (myfile.isEmpty()) {
req.setAttribute("ErrMsg", myfile.getOriginalFilename() + " is empty");
req.setAttribute("Upload", "FALSE");
return "index";
} else {
try {
File dir = new File("E:/myUploadedFiles");
if (!dir.exists())
dir.mkdirs();

String fileName = myfile.getOriginalFilename();
File fileToWrite = new File(dir, fileName);
FileOutputStream fos=new FileOutputStream(fileToWrite);
BufferedOutputStream bos = new BufferedOutputStream(fos);
byte data[] = myfile.getBytes();
bos.write(data);
bos.close();

req.setAttribute("Path", fileToWrite.getAbsolutePath());
req.setAttribute("Upload", "TRUE");
System.out.println("uploaded successfully");

return "index";
} catch (Exception ex) {
ex.printStackTrace();
req.setAttribute("ErrMsg", myfile.getOriginalFilename() + "--.-- " + ex.getMessage());
req.setAttribute("Upload", "FALSE");
return "index";
}
}
}
}
```

3. JLCWebConfig.java

```
package com.coursecube.spring;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.multipart.commons.CommonsMultipartResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

/*
* @Author : Srinivas Dande
* @company : Java Learning Center
* */
@EnableWebMvc
@Configuration
@ComponentScan({ "com.coursecube.spring" })
public class JLCWebConfig   {
        @Bean
        public InternalResourceViewResolver viewResolver() {
                InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
                viewResolver.setViewClass(JstlView.class);
                viewResolver.setPrefix("/WEB-INF/myjsps/");
                viewResolver.setSuffix(".jsp");
                return viewResolver;
        }
        @Bean
        public CommonsMultipartResolver multipartResolver() {
                CommonsMultipartResolver viewResolver = new CommonsMultipartResolver();
                viewResolver.setMaxUploadSize(600000);
                return viewResolver;
        }
}
```

4. JLCWebAppInitializer.java

```
package com.coursecube.spring;

import org.springframework.web.servlet.support.*;
/*
* @Author : Srinivas Dande
* @company : Java Learning Center
* */
public class JLCWebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
@Override
protected Class<?>[] getRootConfigClasses() {
return new Class[] { JLCWebConfig.class };
}
```

```
@Override
protected Class<?>[] getServletConfigClasses() {
return new Class[] { JLCWebConfig.class };
}
@Override
protected String[] getServletMappings() {
return new String[] { "/" };
}
}
```

Lab75: Files required

1. index.jsp	**Updated in Lab75**
2. FileUploadController.java	**Updated in Lab75**
3. JLCWebConfig.java	Same as Lab74
4. JLCWebAppInitializer.java	Same as Lab74

1. index.jsp

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

<html>   <body>
<c:if test="${Upload  eq 'TRUE'}">
<h2> File uploaded successfully to ${Path} </h2>
</c:if>
<c:if test="${Upload  eq 'FALSE'}">
<h2>Error in file upload -- ${ErrMsg} </h2>
</c:if>
<form  action="uploadFile" method="POST" enctype="multipart/form-data">
<table>
<tr>  <td>Student Name</td>
<td> <input type="text" name="sname"/></td>
</tr>
<tr>  <td>Select File 1</td>
<td><input type="file" name="myfile"/></td>
</tr>
<tr>  <td>Select File 2</td>
<td><input type="file" name="myfile"/></td>
</tr>
<tr>  <td>Select File 3</td>
<td><input type="file" name="myfile"/></td>
</tr>
<tr>
<td><input type="submit" value="Upload"/> </td>
</tr>
</table>
</form>  </body>   </html>
```

2. FileUploadController.java

```
package com.coursecube.spring;

import java.io.*;
import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;

@Controller
public class FileUploadController {

@GetMapping("/")
public String showIndexPage() {
System.out.println("---------showIndexPage()---------");
return "index";
}

@PostMapping("/uploadFile")
public String uploadFile(@RequestParam("sname") String sname, @RequestParam("myfile")
MultipartFile myfiles[],HttpServletRequest req) {

System.out.println("---------uploadFile()--------by--"+sname);
for (MultipartFile myfile : myfiles) {

System.out.println(myfile.getOriginalFilename());

if (myfile.isEmpty()) {
req.setAttribute("ErrMsg", myfile.getOriginalFilename() + " is empty");
req.setAttribute("Upload", "FALSE");

return "index";
} else {
try {
File dir = new File("E:/myUploadedFiles");
if (!dir.exists())
dir.mkdirs();

String fileName = myfile.getOriginalFilename();
File fileToWrite = new File(dir, fileName);
FileOutputStream fos=new FileOutputStream(fileToWrite);
BufferedOutputStream bos = new BufferedOutputStream(fos);
byte data[] = myfile.getBytes();
bos.write(data);
bos.close();
```

```
} catch (Exception ex) {
ex.printStackTrace();
req.setAttribute("ErrMsg", myfile.getOriginalFilename() + "--.-- " + ex.getMessage());
req.setAttribute("Upload", "FALSE");
return "index";
}
}
}//End of for loop

req.setAttribute("Path", "E:/myUploadedFiles");
req.setAttribute("Upload", "TRUE");
System.out.println("uploaded successfully");

return "index";

} //End of method
} //end of class
```

I18N

1) Create the required resource bundle file for the required language.
   messages.properties
   messages_hi.properties

2) Configure the bean with the class **ReloadableResourceBundleMessageSource** by specifying the basename of resource bundle.

```
@Bean
public MessageSource messageSource() {
        ReloadableResourceBundleMessageSource messageSource = null;
        messageSource = new ReloadableResourceBundleMessageSource();
        messageSource.setBasename("classpath:messages");
        messageSource.setDefaultEncoding("UTF-8");
        return messageSource;
}
```

3) Configure the bean with the class CookieLocaleResolver by specifying the defaultLocale.

```
@Bean
public LocaleResolver localeResolver() {
        CookieLocaleResolver localeResolver = new CookieLocaleResolver();
        localeResolver.setDefaultLocale(new Locale("en"));
        return localeResolver;
}
```

4) Add the LocaleChangeInterceptor by specifying the parameter name for the language code.

```
@Override
public void addInterceptors(InterceptorRegistry registry) {
        LocaleChangeInterceptor myLocaleInterceptor = null;
        myLocaleInterceptor =new LocaleChangeInterceptor();
        myLocaleInterceptor.setParamName("language");
        registry.addInterceptor(myLocaleInterceptor);
}
```

5) Provide the link in JSP to select the Language as follows:

```
<a href="?language=en">English</a> | <a href="?language=hi">Hindi</a>
```

6) Use the page directive by specifying the character set as UTF-8

```
<%@ page contentType="text/html;charset=UTF-8" %>
```

7) Use the taglib directive to use the spring tags.

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

8) Use the <spring:message/> tag to display the message from resource bundle.

```
<spring:message code="jlc.header" text="Java Learning Center" />
```

Lab76: Files required

1. index.jsp	2. home.jsp
3. messages.properties	4. messages_hi.properties
5. messages_kn.properties	6. messages_te.properties
7. HomeController.java	8. JLCWebConfig.java
9. JLCWebAppInitializer.java	

1. index.jsp

```
<html>
<body>
 <br><h1>Java Learning Center<br/>
<a href="home">Home</a><br/><br/>
</h1>
</body>
</html>
```

2. home.jsp

```
<%@ page contentType="text/html;charset=UTF-8" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<html>
<body>
<h2> <spring:message code="jlc.header" text="Java Learning Center" /> </h2>
<h2> <spring:message code="jlc.body" text="Welcome to JLC" /> </h2>
<br/><br/>
<h3>Language :
<a href="?mylanguage=en">English</a> |
 <a href="?mylanguage=hi">Hindi</a> |
<a href="?mylanguage=kn">Kannada</a> |
 <a href="?mylanguage=te">Telugu</a>
</h3>
</body>
 </html>
```

3. messages.properties

```
jlc.header=Java Learning Center [EN]
jlc.body=Welcome to Java Learning Center [EN]
```

4. messages_hi.properties

```
jlc.header=\u091C\u093E\u0935\u093E \u0932\u0930\u094D\u0928\u093F\u0902\u0917
\u0938\u0947\u0902\u091F\u0930

jlc.body=\u091C\u093E\u0935\u093E \u0932\u0930\u094D\u0928\u093F\u0902\u0917
\u0938\u0947\u0902\u091F\u0930 \u092E\u0947 \u0906\u092A\u0915\u093E
\u0938\u094D\u0935\u093E\u0917\u0924 \u0939\u0948
```

5. messages_kn.properties

```
jlc.header=Java Learning Center [KN]
jlc.body=Welcome to Java Learning Center [KN]
```

6. messages_te.properties

```
jlc.header=Java Learning Center [TE]
jlc.body=Welcome to Java Learning Center [TE]
```

7. HomeController.java

```
package com.coursecube.spring;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
@Controller
public class HomeController {
@GetMapping("/")
public String showIndexPage() {
System.out.println("---------showIndexPage()---------");
return "index";
}
@GetMapping("/home")
public String showHome() {
System.out.println("---------showHome()---------");
return "home";
}
}
```

8. JLCWebConfig.java

```
package com.coursecube.spring;

import java.util.Locale;
import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.support.ReloadableResourceBundleMessageSource;
import org.springframework.web.servlet.LocaleResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.CookieLocaleResolver;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;
/*
* @Author : Srinivas Dande
* @company : Java Learning Center
* */
@EnableWebMvc
@Configuration
@ComponentScan({ "com.coursecube.spring" })
public class JLCWebConfig   implements WebMvcConfigurer   {

@Bean
public InternalResourceViewResolver viewResolver() {
System.out.println("viewResolver");

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
viewResolver.setViewClass(JstlView.class);
viewResolver.setPrefix("/WEB-INF/myjsps/");
viewResolver.setSuffix(".jsp");
return viewResolver;
}

@Bean
public MessageSource messageSource() {
System.out.println("messageSource");

ReloadableResourceBundleMessageSource messageSource = new
ReloadableResourceBundleMessageSource();
messageSource.setBasename("classpath:messages");
messageSource.setDefaultEncoding("UTF-8");
return messageSource;
}
```

```
@Bean
public LocaleResolver localeResolver() {
System.out.println("localeResolver");

CookieLocaleResolver localeResolver = new CookieLocaleResolver();
localeResolver.setDefaultLocale(new Locale("en"));
return localeResolver;
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
System.out.println("addInterceptors");

LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();
localeChangeInterceptor.setParamName("mylanguage");
registry.addInterceptor(localeChangeInterceptor);
}
}
```

Spring Mail API

1) Create the Java Project with the Lab77.
2) Add All the Spring5 Jars to Application Build Path.
3) Add the following mail jars to Application Build Path.

   mail-1.4.7.jar

   activation-1.1.1.jar

4) Configure the bean with the class JavaMailSender by specifying the mail server details.

```
@Bean
public JavaMailSender javaMailSenderImpl() {
        JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
        mailSender.setHost("smtp.gmail.com");
        mailSender.setPort(465);

        // Set gmail email id
        mailSender.setUsername("xxxxx@gmail.com");

        // Set gmail email password
        mailSender.setPassword("xxxxx");

        Properties prop = mailSender.getJavaMailProperties();

        prop.put("mail.smtp.host", "smtp.gmail.com");
        prop.put("mail.smtp.socketFactory.port", "465");
        prop.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
        prop.put("mail.smtp.auth", "true");
        prop.put("mail.smtp.startssl.enable", "true");
        //prop.put("mail.debug", "true");

        return mailSender;
}
```

5) Write the MailService.java which has two methods.

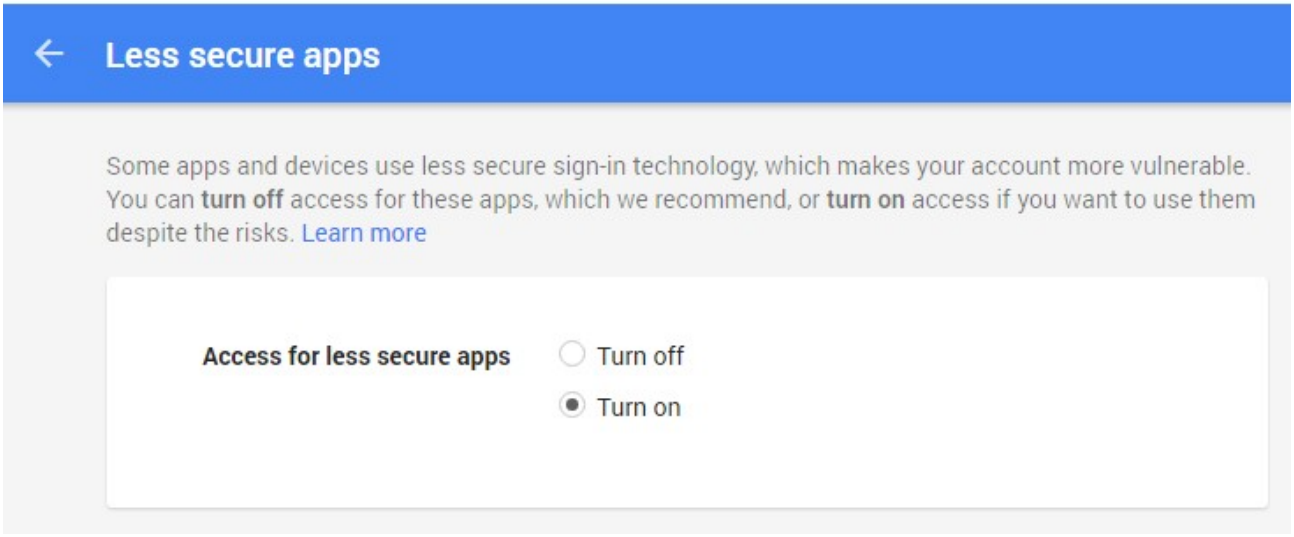
   public void sendMail(String from,String to,String subject,String body)
   public void sendMail(String from,String to,String subject,String body,File file)

6) Write the MailTest1.java which send the mail without Attachments
7) Write the MailTest2.java which send the mail with Attachments

Note:

We need to set up less secure with our Gmail account as follows:

A) Login to Gmail.
B) Access the URL https://www.google.com/settings/security/lesssecureapps
C) Select "Turn on"



Lab 77: Spring Mail Example.

Lab76: Files required

1. MailTest1.java	2. MailTest2.java
3. MailService.java	4. JLCAppConfig.java

1. MailTest1.java

```
package com.coursecube.spring;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class MailTest1 {
public static void main(String[] args) {
AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(JLCConfig.class);
MailService ms=ctx.getBean(MailService.class);

String from="sri@gmail.com"; //You set Valid Email ID
String to="sri@coursecube.com";  //You set Valid Email ID
String subject="Srinivas !!! Test mail from Spring Application";
String body="<font color='red' size='7'>Hello Guys<br/> This is test mail from spring
application</font>";

ms.sendMail(from, to, subject, body);

System.out.println("---Done---");
}
}
```

2. MailTest2.java

```
package com.coursecube.spring;

import java.io.File;
import org.springframework.context.annotation.*;
/*
* @Author : Srinivas Dande
* @company : Java Learning Center
* */
public class MailTest2 {
public static void main(String[] args) {
AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(JLCConfig.class);
MailService ms=ctx.getBean(MailService.class);

String from="sri@gmail.com"; //You set Valid Email ID
String to="sri@coursecube.com";  //You set Valid Email ID
String subject="Srinivas !!! Test mail from Spring Application";
String body="<font color='red' size='7'>Hello Guys<br/> This is test mail from spring
application</font>";
File file=new File("E:/hello/hello.txt");

ms.sendMail(from, to, subject, body,file);
System.out.println("---Done---");
}
}
```

3. MailService.java

```
package com.coursecube.spring;

import java.io.File;
import javax.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.FileSystemResource;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;
/*
* @Author : Srinivas Dande
* @company : Java Learning Center
* */

@Service
public class MailService {

@Autowired
JavaMailSender mailSender;
```

```
public void sendMail(String from,String to,String subject,String body) {
try {
MimeMessage mimeMessage = mailSender.createMimeMessage();
MimeMessageHelper mailMsg = new MimeMessageHelper(mimeMessage);
mailMsg.setFrom(from);
mailMsg.setTo(to);
mailMsg.setSubject(subject);
mailMsg.setText(body,true);
mailSender.send(mimeMessage);
} catch (Exception ex) {
ex.printStackTrace();
}
}
public void sendMail(String from,String to,String subject,String body,File file) {
try {
MimeMessage mimeMessage = mailSender.createMimeMessage();
MimeMessageHelper mailMsg = new MimeMessageHelper(mimeMessage, true);
mailMsg.setFrom(from);
mailMsg.setTo(to);
mailMsg.setSubject(subject);
mailMsg.setText(body,true);
FileSystemResource fileRes = new FileSystemResource(file);
mailMsg.addAttachment(file.getName(), fileRes);
mailSender.send(mimeMessage);
} catch (Exception ex) {
ex.printStackTrace();
}
}
}
```

4. JLCAppConfig.java

```
package com.coursecube.spring;

import java.util.Properties;
import org.springframework.context.annotation.*;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;
/*
* @Author : Srinivas Dande
* @company : Java Learning Center
* */

@Configuration
@ComponentScan({ "com.coursecube.spring" })
public class JLCConfig   {
```

```
@Bean
public JavaMailSender javaMailSenderImpl() {
JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
mailSender.setHost("smtp.gmail.com");
mailSender.setPort(465);

mailSender.setUsername("<username>@gmail.com");
mailSender.setPassword("<password>");

Properties prop = mailSender.getJavaMailProperties();
prop.put("mail.smtp.host", "smtp.gmail.com");
prop.put("mail.smtp.socketFactory.port", "465");
prop.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
prop.put("mail.smtp.auth", "true");
prop.put("mail.smtp.startssl.enable", "true");
//prop.put("mail.debug", "true");
return mailSender;
}

}
```