

Aug 31st Lecture and Project spring-mvc-demo

Create a Maven MVC project and deploy on an IDE Tomcat server. This document describes creating a MVC project in the SpringToolSuite IDE. Other documents describes Tomcat server deployments within the IDE, on local machine, and on AWS.

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE.

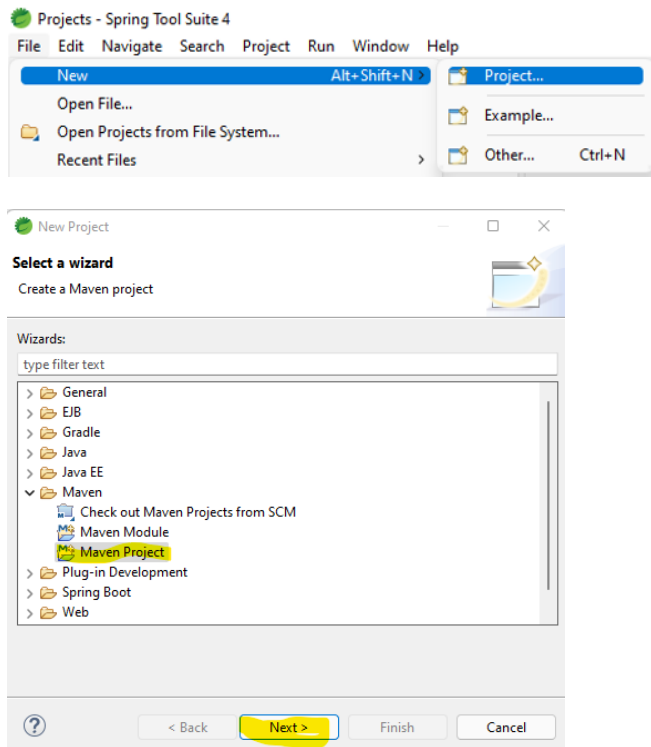
Table of Contents

Create a Maven MVC project and deploy on an IDE Tomcat server	3
Create New Maven Project	3
Update the pom.xml	5
Update Java Version.....	5
Correct Error: Web.xml is missing.....	8
Update web.xml.....	13
Update outdated link	13
Add DispatcherServlet to web.xml	14
Create an applicationContext.xml File	15
Update the applicationContext.xml File	16
Add other project dependencies to pom.xml	18
Complete the MVC demo project	20
Create controller package.....	20
Create controller class	21
Update the class shell TestController	21
Create Data Transfer Object (DTO) package.....	23
Create DTO class	23
Update the class shell LoginDTO.....	24
Updates for Future Projects.....	26
Test the MVC Demo project on the IDE Tomcat Server	28
Start the Tomcat Server	29

Hello World Test	29
Login Test	29
Postman web (cloud) based issue.....	29
Open Postman Desktop Agent.....	30
Spring MVC Project Next Steps for Tomcat Server(s)	32

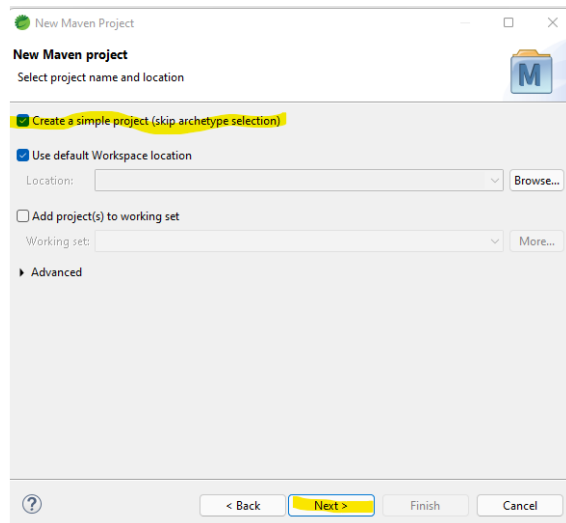
Create a Maven MVC project and deploy on an IDE Tomcat server

Create New Maven Project



Select “Create a simple project ...”

Click “Next >”



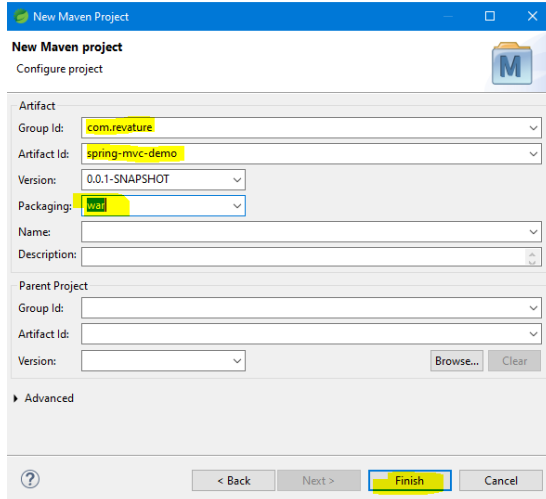
Enter the Group Id: com.revature

Enter the Artifact Id: spring-mvc-demo

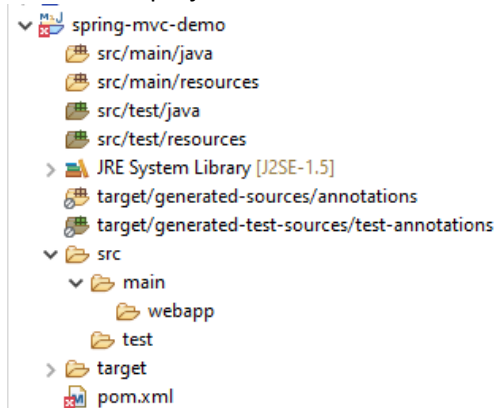
Select Packaging: **war**

Note: packaging must be war to deploy on a Tomcat server.

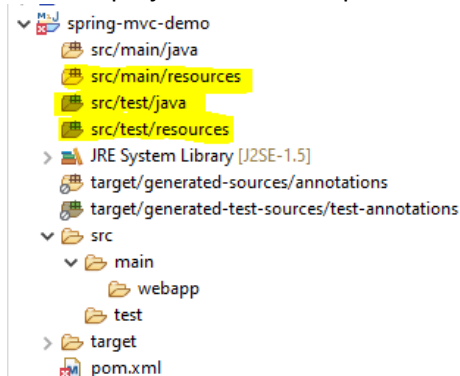
Click Finish



The default project structure should look as follows:



It is important that the project structure does contain the highlighted folders. These will be important for later projects that are copied from this project.



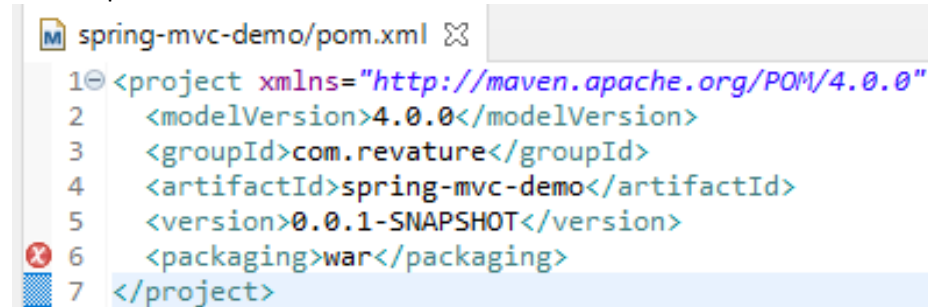
The error indicated will be resolved in future steps.

Update the pom.xml

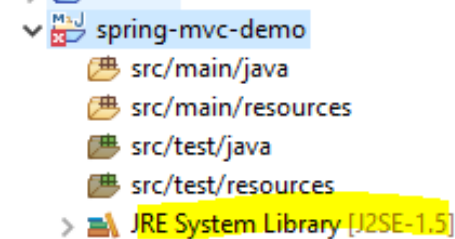
Update Java Version

BEFORE PROCEEDING MAKE SURE THE PROJECT IS ON Java 1.8 in the pom.xml and has been rebuilt.

Default pom.xml:

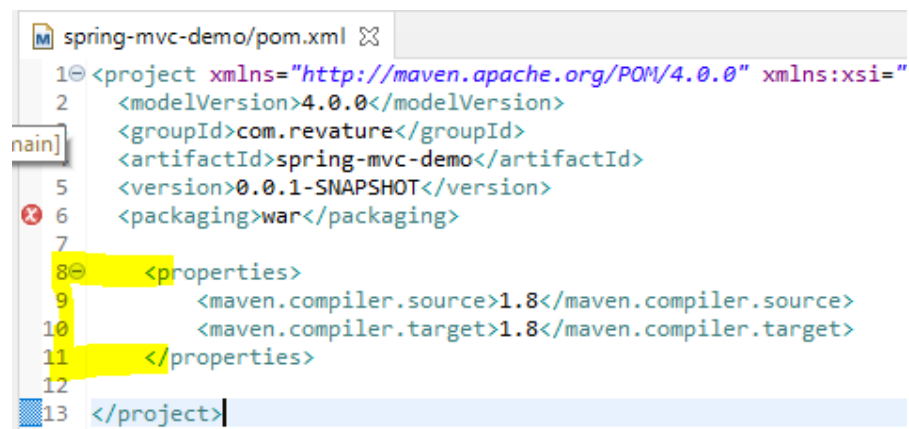


Default project library:



Add the Java 1.8 in the pom.xml

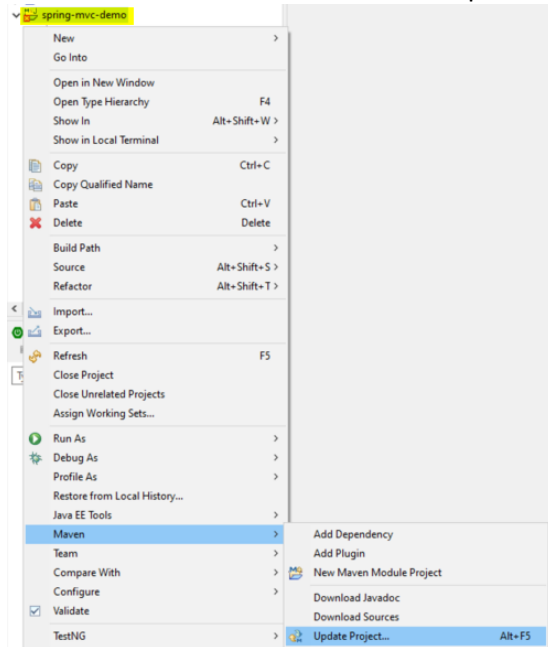
```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```



Rebuild the Maven project

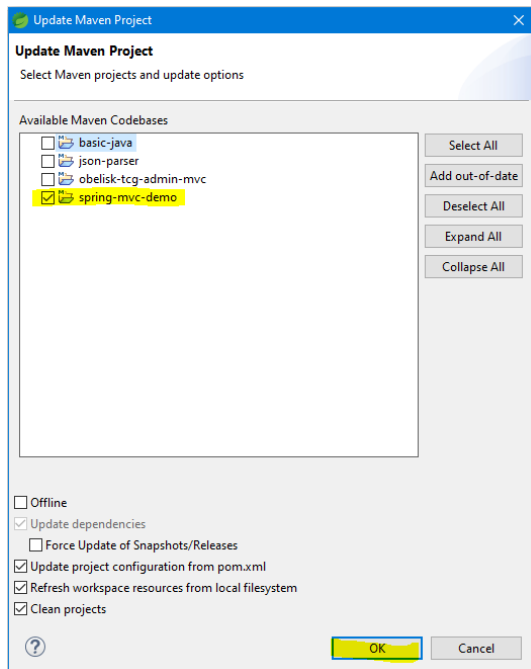
Right click on the Project

Mouse hover on “Maven” → select “Update Project...”

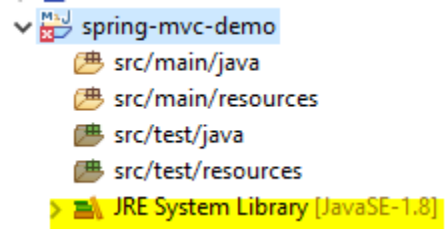


The project should already be selected.

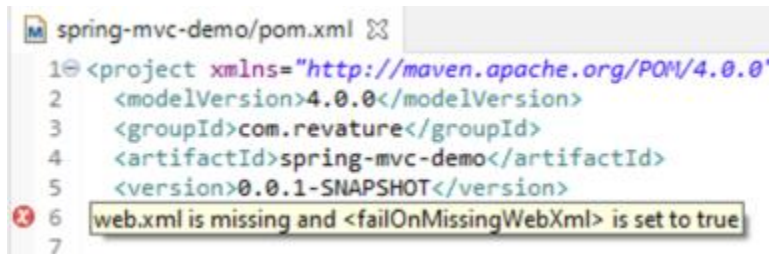
Click “OK”



Project library should now be at the new Java version:



Correct Error: Web.xml is missing



The screenshot shows a code editor with a file named 'spring-mvc-demo/pom.xml'. The XML content is as follows:

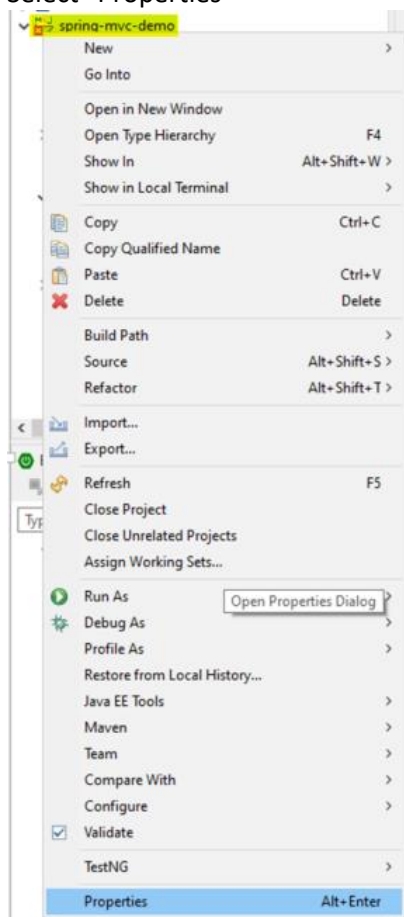
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3       <modelVersion>4.0.0</modelVersion>
4       <groupId>com.revature</groupId>
5       <artifactId>spring-mvc-demo</artifactId>
6       <version>0.0.1-SNAPSHOT</version>
7       <web.xml is missing and <failOnMissingWebXml> is set to true
```

An error icon (a red 'x') is visible on the left margin next to line 6, and the error message is highlighted in a yellow box.

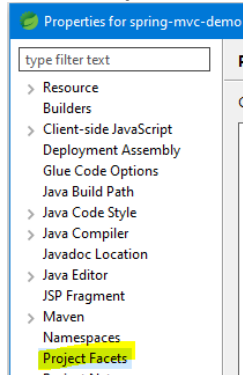
Change Project Properties

Right click on project

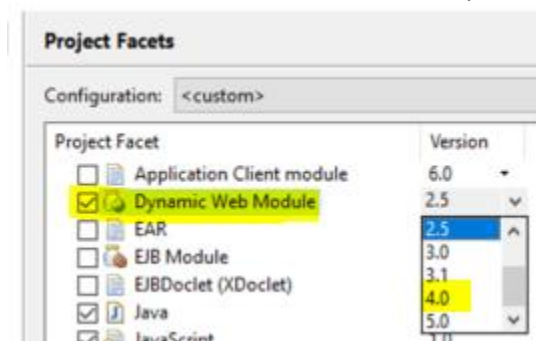
Select "Properties"



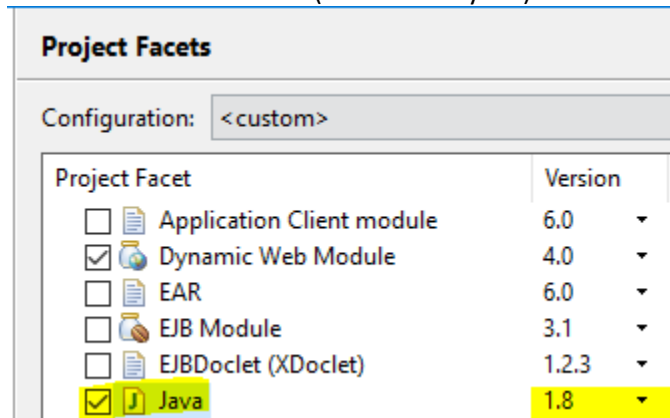
Select Project Facets



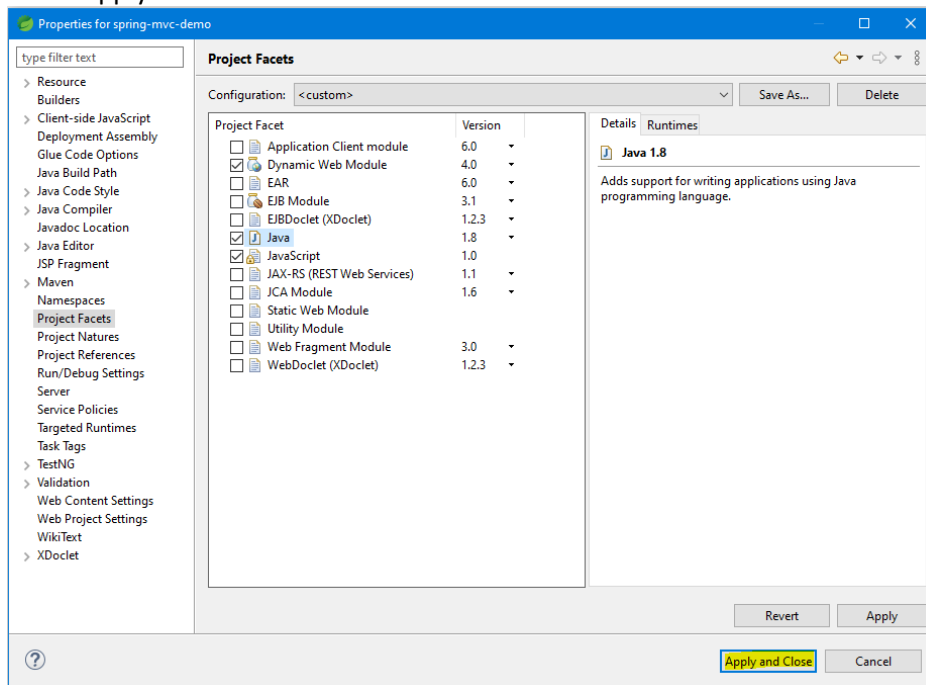
Click on Project Facet: "Dynamic Web Module", check if unchecked
Select "Version" → "4.0" from the dropdown



Click on Project Facet: "Java", check if unchecked
Select "Version" → "1.8" (if not already set)



Click “Apply and Close”

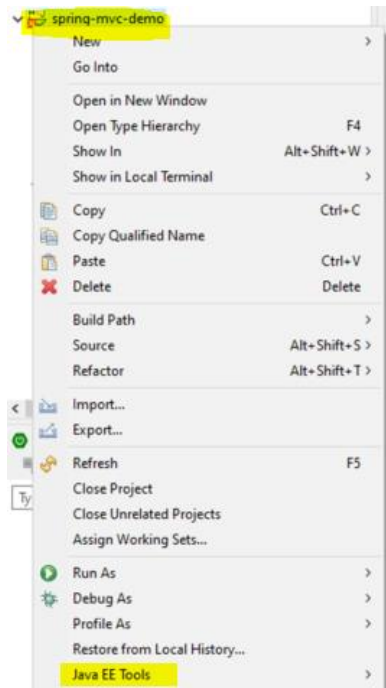


Install Java EE Tool if needed

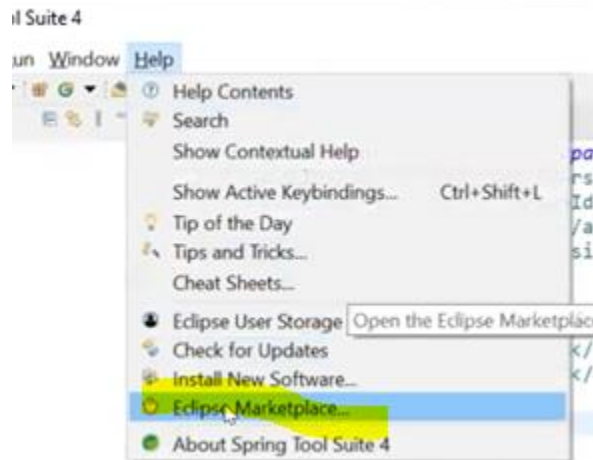
Right click on project

If “Java EE Tools” is in the menu list skip to **“Generate Deployment Descriptor Stub”**

If “Java EE Tools” is not in the menu list follow the next two steps



Step 1: Install via Eclipse Market Place under the Help menu



Step 2: Search tab: enter enterprise

If not already installed, install the component

Once the installation completes **restart** the IDE for the changes to take affect

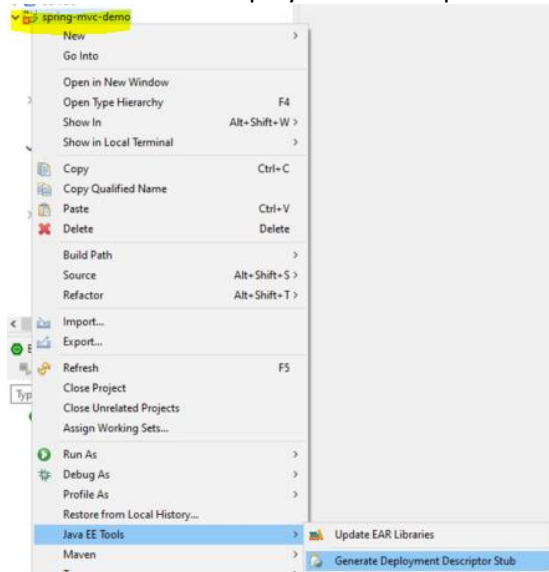


Generate Deployment Descriptor Stub

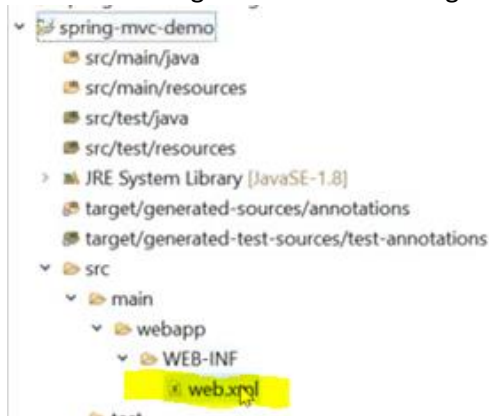
Right click on the project

Highlight “Java EE Tools”

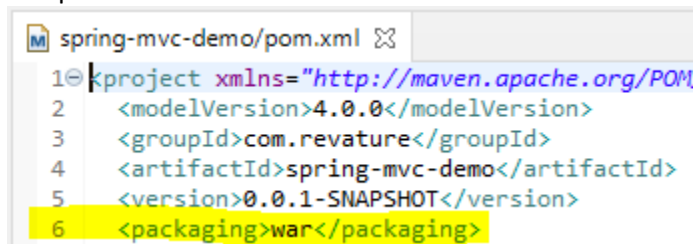
Select “Generate Deployment Descriptor Stub”



The web.xml is generated in following location.



The pom.xml error should now be resolved.



Update web.xml

Any time the web.xml file it can all of sudden have an error. This is some kind of bug with the IDE. To resolve the false error just add a space or add a line and save the file.

Open Web.xml File

Update outdated link

```
web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.
3 <display-name>spring-mvc-demo</display-name>
4 <welcome-file-list>
5   <welcome-file>index.html</welcome-file>
6   <welcome-file>index.htm</welcome-file>
7   <welcome-file>index.jsp</welcome-file>
8   <welcome-file>default.html</welcome-file>
9   <welcome-file>default.htm</welcome-file>
10  <welcome-file>default.jsp</welcome-file>
11 </welcome-file-list>
12 </web-app>
```

From:

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
```

To:

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app 4.0.xsd" version="4.0">
```

You might need to do spacing on the display name as well to get rid of the errors.

```
web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3
4   <display-name>spring-mvc-demo</display-name>
5   <welcome-file-list>
6     <welcome-file>index.html</welcome-file>
7     <welcome-file>index.htm</welcome-file>
8     <welcome-file>index.jsp</welcome-file>
9     <welcome-file>default.html</welcome-file>
10    <welcome-file>default.htm</welcome-file>
11    <welcome-file>default.jsp</welcome-file>
12  </welcome-file-list>
13 </web-app>
```

Add DispatcherServlet to web.xml

The web.xml file is used for Spring Framework and Tomcat server configuration.

Add the following to the web.xml file

```
<!-- This is where we configure our DispatcherServlet (which comes from Spring Web) -->
<!-- The DispatcherServlet is the sole Servlet that receives HTTP requests through our Tomcat
server -->
<!-- It then routes the HTTP requests to the appropriate controller -->
<servlet>
    <servlet-name>DispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>DispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

```
web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/ja
5     version="4.0">
6
7     <display-name>spring-mvc-demo</display-name>
8     <welcome-file-list>
9         <welcome-file>index.html</welcome-file>
10        <welcome-file>index.htm</welcome-file>
11        <welcome-file>index.jsp</welcome-file>
12        <welcome-file>default.html</welcome-file>
13        <welcome-file>default.htm</welcome-file>
14        <welcome-file>default.jsp</welcome-file>
15    </welcome-file-list>
16
17    <!-- This is where we configure our DispatcherServlet (which comes from
18         Spring Web) -->
19    <!-- The DispatcherServlet is the sole Servlet that receives HTTP requests
20         through our Tomcat server -->
21    <!-- It then routes the HTTP requests to the appropriate controller -->
22    <servlet>
23        <servlet-name>DispatcherServlet</servlet-name>
24        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
25
26        <init-param>
27            <param-name>contextConfigLocation</param-name>
28            <param-value>/WEB-INF/applicationContext.xml</param-value>
29        </init-param>
30        <load-on-startup>1</load-on-startup>
31    </servlet>
32
33    <servlet-mapping>
34        <servlet-name>DispatcherServlet</servlet-name>
35        <url-pattern>/</url-pattern>
36    </servlet-mapping>
37
38 </web-app>
```

All http request goes to the DispatcherServlet which in turn sends to the controller(s) endpoints.

“Front Controller Design Pattern”

```
23     <servlet-name>DispatcherServlet</servlet-name>
24     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

Create an applicationContext.xml File

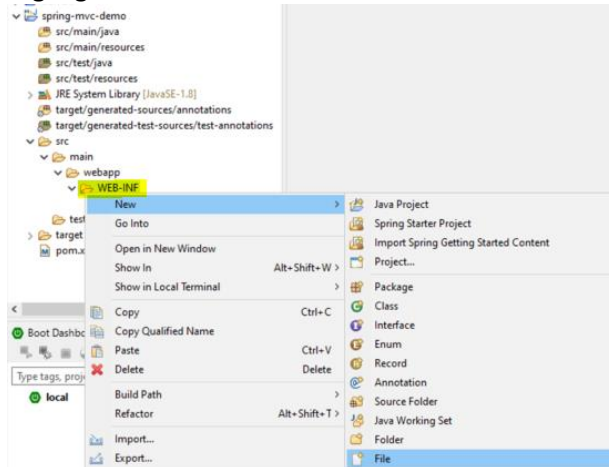
The web.xml file makes reference to an applicationContext.xml file:

```
27 <param-name>contextConfigLocation</param-name>
28 <param-value>/WEB-INF/applicationContext.xml</param-value>
```

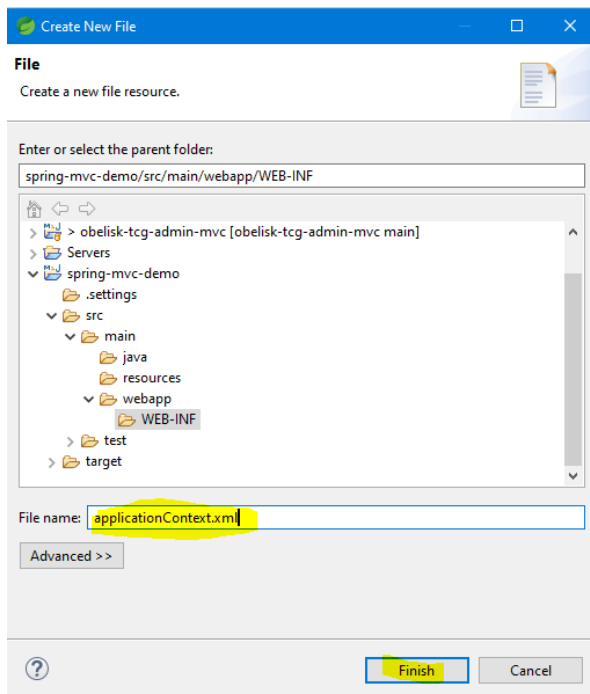
Create the applicationContext.xml file in the directory:
spring-mvc-demo\src\main\webapp\WEB-INF\

Right click on WEB-INF folder

Highlight "New" → select "File"



Enter the "File name:" applicationContext.xml
Click "Finish"



Update the applicationContext.xml File

Add the following to the xml file:

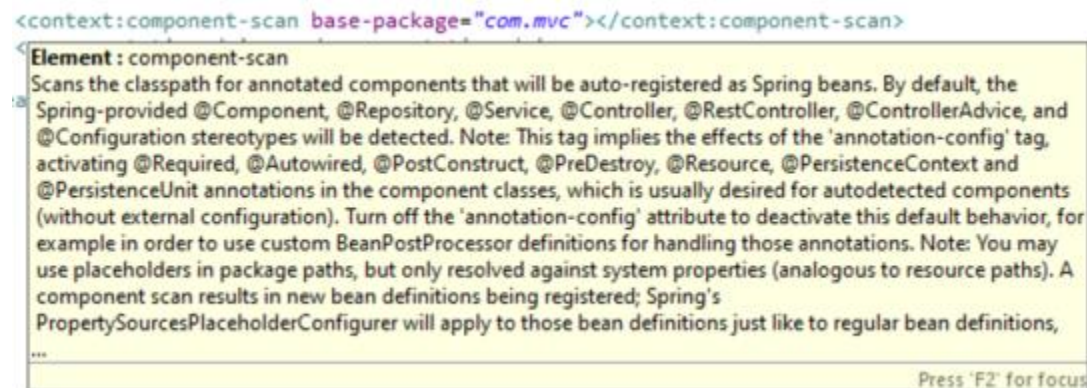
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <context:component-scan base-package="com.revature"></context:component-scan>
    <mvc:annotation-driven></mvc:annotation-driven>

</beans>
```



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:mvc="http://www.springframework.org/schema/mvc"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7 http://www.springframework.org/schema/beans/spring-beans.xsd
8 http://www.springframework.org/schema/context
9 http://www.springframework.org/schema/context/spring-context.xsd
10 http://www.springframework.org/schema/mvc
11 http://www.springframework.org/schema/mvc/spring-mvc.xsd">
12
13 <context:component-scan base-package="com.revature"></context:component-scan>
14
15 </beans>
```



```
<context:component-scan base-package="com.mvc"></context:component-scan>
```

Element: component-scan
Scans the classpath for annotated components that will be auto-registered as Spring beans. By default, the Spring-provided @Component, @Repository, @Service, @Controller, @RestController, @ControllerAdvice, and @Configuration stereotypes will be detected. Note: This tag implies the effects of the 'annotation-config' tag, activating @Required, @Autowired, @PostConstruct, @PreDestroy, @Resource, @PersistenceContext and @PersistenceUnit annotations in the component classes, which is usually desired for autodetected components (without external configuration). Turn off the 'annotation-config' attribute to deactivate this default behavior, for example in order to use custom BeanPostProcessor definitions for handling those annotations. Note: You may use placeholders in package paths, but only resolved against system properties (analogous to resource paths). A component scan results in new bean definitions being registered; Spring's PropertySourcesPlaceholderConfigurer will apply to those bean definitions just like to regular bean definitions, ...

Press 'F2' for focus

Make sure to update the base package: com.revature to the project's definition.

Make sure to add the `<mvc:annotation>` tags

`mvc:annotation` allows for the `@Component`, `@Service` and `@RestController` tags.

```
web.xml applicationContext.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:mvc="http://www.springframework.org/schema/mvc"
6       xsi:schemaLocation="http://www.springframework.org/schema/beans
7       http://www.springframework.org/schema/beans/spring-beans.xsd
8       http://www.springframework.org/schema/context
9       http://www.springframework.org/schema/context/spring-context.xsd
10      http://www.springframework.org/schema/mvc
11      http://www.springframework.org/schema/mvc/spring-mvc.xsd">
12
13     <context:component-scan base-package="com.revature"></context:component-scan>
14     <mvc:annotation-driven></mvc:annotation-driven>
15
16 </beans>
```

`<mvc:annotation-driven></mvc:annotation-driven>`

Element: annotation-driven

Configures the annotation-driven Spring MVC Controller programming model. Note that this tag works in Web MVC only, not in Portlet MVC! See org.springframework.web.servlet.config.annotation.EnableWebMvc javadoc for details on code-based alternatives to enabling annotation-driven Spring MVC support.

Content Model: all(path-matching?, message-converters?, argument-resolvers?, return-value-handlers?, async-support?)?

Press 'F2' for focus

Add other project dependencies to pom.xml

The current pom.xml file should look like the following after we Java 1.8 to the default file earlier in this document:

```
spring-mvc-demo/pom.xml
1<?xml version="1.0" encoding="UTF-8" ?>
2<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>com.revature</groupId>
6  <artifactId>spring-mvc-demo</artifactId>
7  <version>0.0.1-SNAPSHOT</version>
8  <packaging>war</packaging>
9  <properties>
10   <maven.compiler.source>1.8</maven.compiler.source>
11   <maven.compiler.target>1.8</maven.compiler.target>
12 </properties>
13</project>
```

Add dependencies tags

Between the dependencies tags

- Add dependencies for Spring Framework
- Add dependencies for Tomcat
- Add dependencies for Project Lombok
- Other dependencies for other common project items in future projects depending on needs.

The dependencies added are taken from the Maven Repository: <https://mvnrepository.com/>. Future dependencies needed can also be found on this website.

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
  </dependency>
  <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
  <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.5</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.9</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.20</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

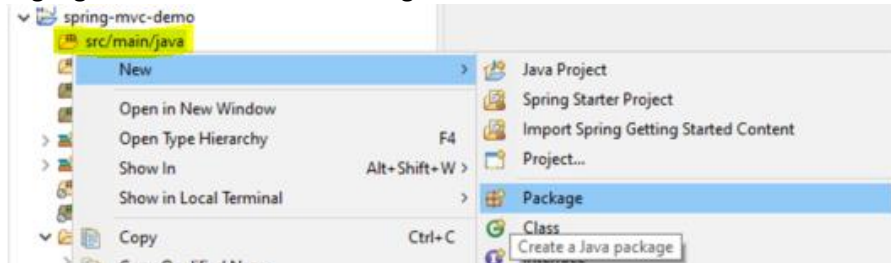
```
spring-mvc-demo/pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ir
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>com.revature</groupId>
4   <artifactId>spring-mvc-demo</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6   <packaging>war</packaging>
7
8   <properties>
9     <maven.compiler.source>1.8</maven.compiler.source>
10    <maven.compiler.target>1.8</maven.compiler.target>
11  </properties>
12
13  <dependencies>
14    <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
15    <dependency>
16      <groupId>javax.servlet</groupId>
17      <artifactId>javax.servlet-api</artifactId>
18      <version>4.0.1</version>
19      <scope>provided</scope>
20    </dependency>
21    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
22    <dependency>
23      <groupId>com.fasterxml.jackson.core</groupId>
24      <artifactId>jackson-databind</artifactId>
25      <version>2.12.5</version>
26    </dependency>
27    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
28    <dependency>
29      <groupId>org.springframework</groupId>
30      <artifactId>spring-webmvc</artifactId>
31      <version>5.3.9</version>
32    </dependency>
33    <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
34    <dependency>
35      <groupId>org.projectlombok</groupId>
36      <artifactId>lombok</artifactId>
37      <version>1.18.20</version>
38      <scope>provided</scope>
39    </dependency>
40  </dependencies>
41
42 </project>
```

Complete the MVC demo project

Create controller package

Right click folder: "src/main/java"

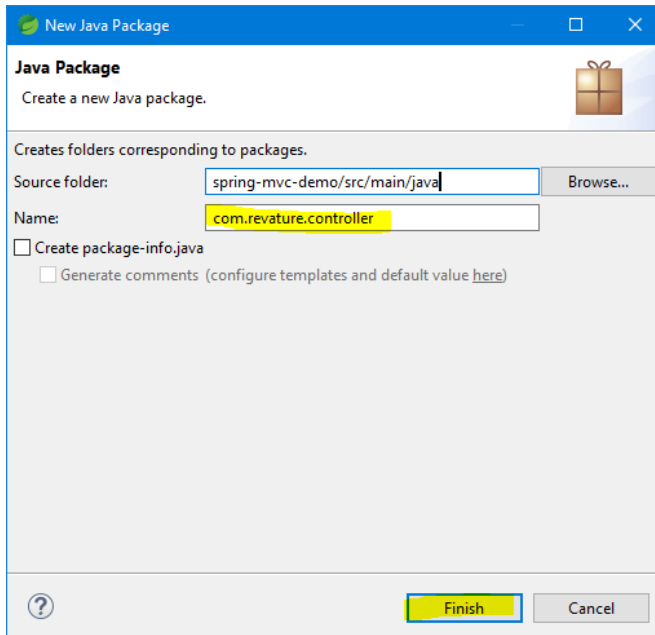
Highlight "New" → select "Package"



Enter "Name:"

com.revature.controller

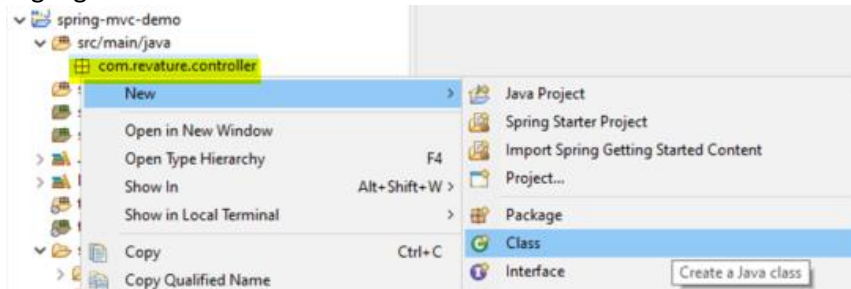
Click "Finish"



Create controller class

Right click package: "com.revature.controller"

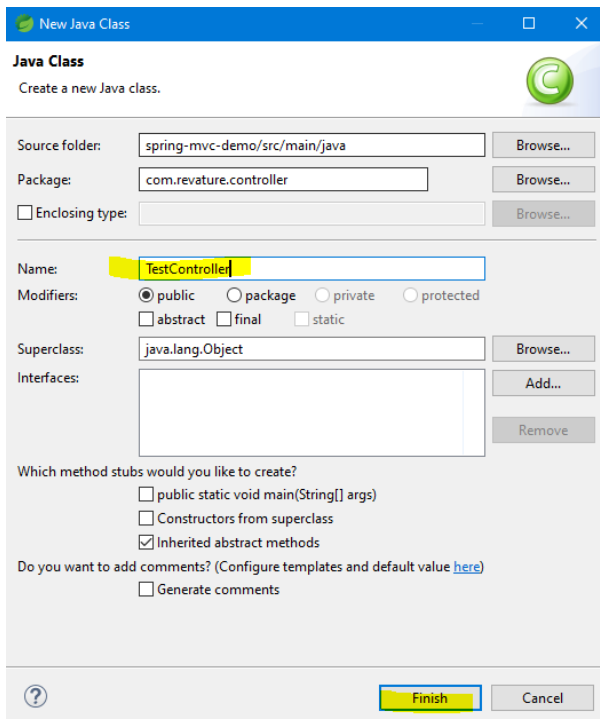
Highlight "New" → select "Class"



Enter "Name:"

TestController

Click "Finish"



Update the class shell TestController

```
TestController.java
1 package com.revature.controller;
2
3 public class TestController {
4
5 }
```

The final class code will contain:

```
package com.revature.controller;

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.RequestBody;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.revature.dto.LoginDTO;

@RestController // I changed the annotation from @Controller to @RestController
// So I do not need to put @ResponseBody on my methods anymore
// @ResponseBody's purpose is to specify that the return type should be serialized into, for
// example, JSON and placed into the
// body of our HTTP response
public class TestController {

    @GetMapping(path = "/hello", produces = "application/json")
    public String hello() {
        return "Hello world!";
    }

    @PostMapping(path = "/login", consumes = "application/json", produces =
"application/json")
    public LoginDTO login(@RequestBody LoginDTO loginDto) {
        return loginDto;
    }
}

```

You can type each line of code starting with `@RestController`. If you take this approach then required imports are generally added automatically and you can take care of other dependencies or errors as you type.

Alternatively just copy and paste code above into the TestController class. Then continue following instruction in this document and by the end all error will be resolved.

TestController class after copy and paste showing errors for a dependent package and class.

```

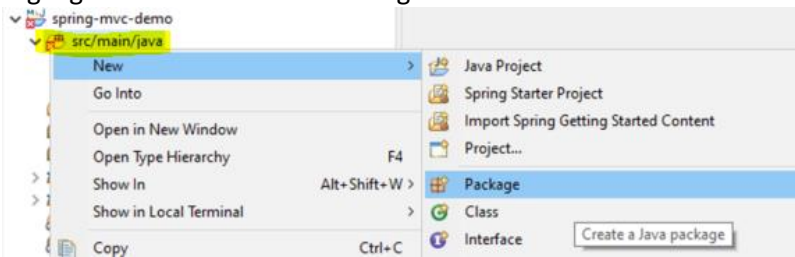
TestController.java
1 package com.revature.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.PostMapping;
6 import org.springframework.web.bind.annotation.RequestBody;
7 import org.springframework.web.bind.annotation.ResponseBody;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import com.revature.dto.LoginDTO;
11
12 @RestController // I changed the annotation from @Controller to @RestController
13 // So I do not need to put @ResponseBody on my methods anymore
14 // @ResponseBody's purpose is to specify that the return type should be serialized into, for example, JSON and placed into the
15 // body of our HTTP response
16 public class TestController {
17
18     @GetMapping(path = "/hello", produces = "application/json")
19     public String hello() {
20         return "Hello world!";
21     }
22
23     @PostMapping(path = "/login", consumes = "application/json", produces = "application/json")
24     public LoginDTO login(@RequestBody LoginDTO loginDto) {
25         return loginDto;
26     }
27
28 }

```

Create Data Transfer Object (DTO) package

Right click folder: "src/main/java"

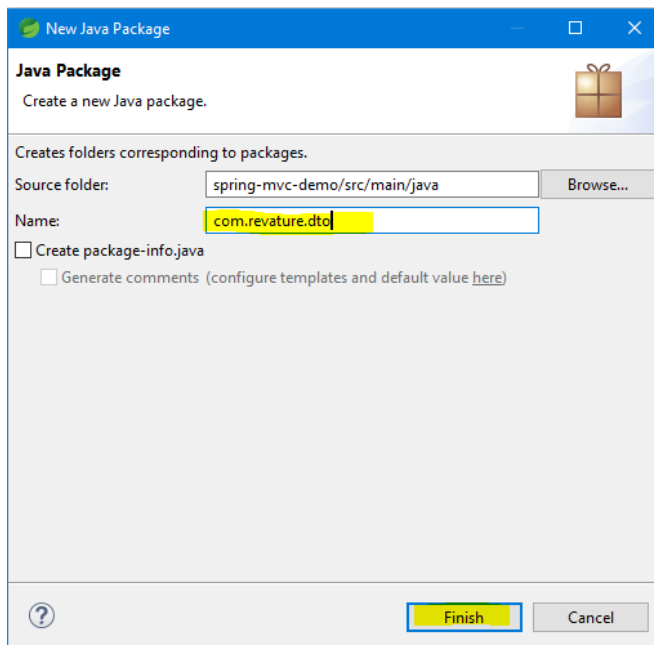
Highlight "New" → select "Package"



Enter "Name:"

com.revature.dto

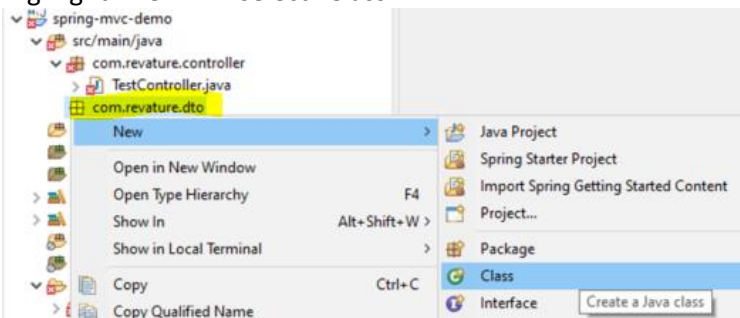
Click "Finish"



Create DTO class

Right click package: "com.revature.dto"

Highlight "New" → select "Class"



Enter "Name:"

LoginDTO

Click "Finish"

New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

Once the LoginDTO class is created the errors in TestController class are resolved.

Update the class shell LoginDTO

```
1 package com.revature.dto;  
2  
3 public class LoginDTO {  
4  
5 }
```

The final class code will contain:

```
package com.revature.dto;  
  
import lombok.EqualsAndHashCode;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import lombok.ToString;  
  
@Getter @Setter @EqualsAndHashCode @NoArgsConstructor @ToString  
public class LoginDTO {  
  
    private String username;  
    private String password;  
  
    public LoginDTO(String username, String password) {  
        this.username = username;  
        this.password = password;  
    }  
}
```



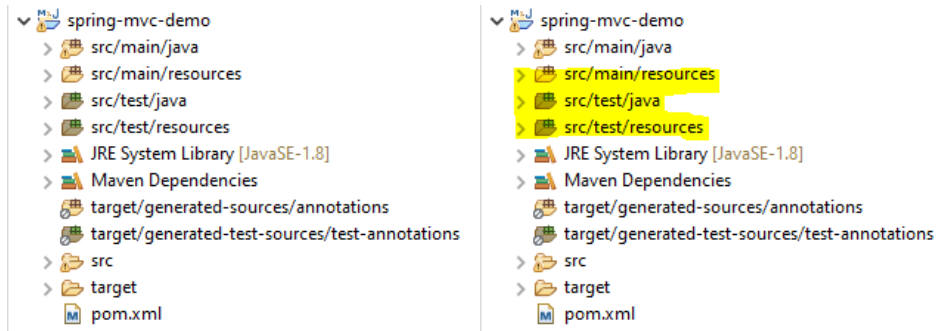
```
}
```

As before you can type each line of code starting with `@Getter`. If you take this approach then required imports are generally added automatically and you can take care of other dependencies or errors as you type.

Alternatively just copy and paste code above into the LoginDTO class. After completing this class there should be no other errors in the project.

Updates for Future Projects

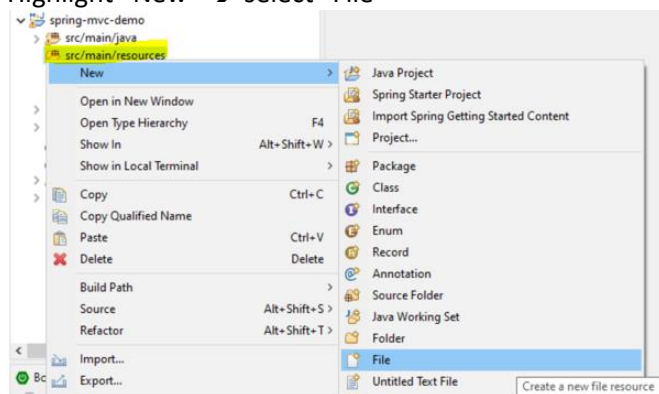
As mentioned when first creating the Maven project for this demo, the following default folder structures were created:



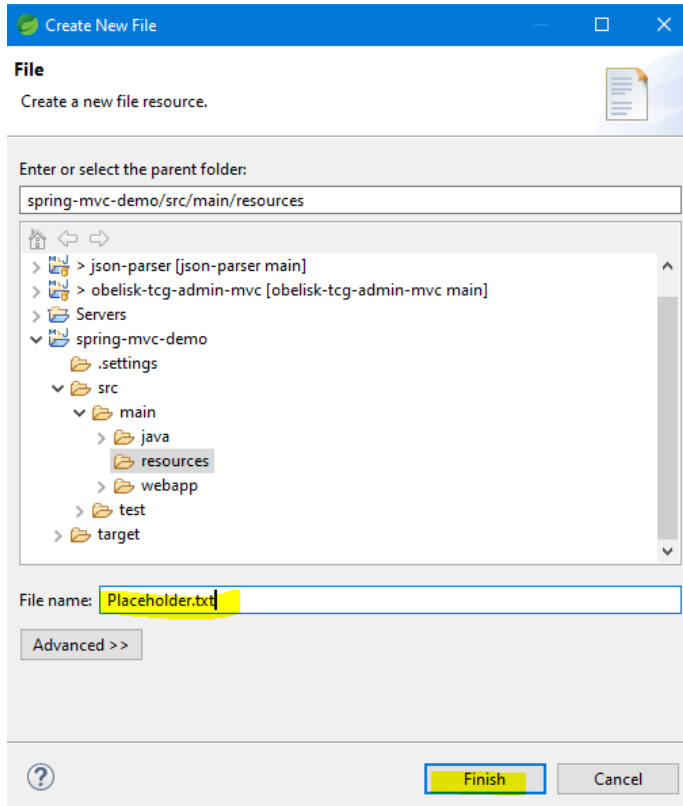
The highlighted folders contains no files as part of this project. An issue of concern is when adding this project to a Git Hub repository. Git Hub will not maintain the folder structure if there are no files in any of the folders. This becomes an issue if this project is deleted from the IDE and from disk and the Git Hub version is then cloned and imported. The highlighted folders will not be in the project. The missing folders become an issue in a demo called spring-mvc-with-orm-testing-demo. The spring-mvc-with-orm-testing-demo will be copied from another demo spring-mvc-with-orm-demo which is copied from this project spring-mvc-demo.

A simple fix to the Git Hub repository issue is to create placeholder files in each of the highlighted directories above.

Right click one of the highlighted directories
Highlight "New" → select "File"

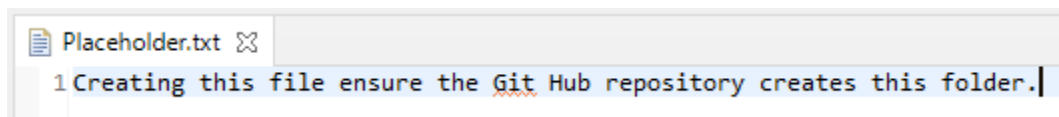


Enter "File name:" Placeholder.txt
Click "Finish"

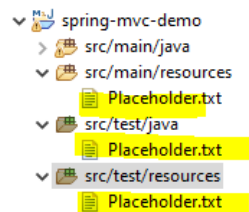


Add some text and save the file

Creating this file ensure the Git Hub repository creates this folder.



You can copy and paste this file to the other two folders or repeat the steps above to create the file again.



These changes will ensure the folder structure is preserved when using Git Hub.

Test the MVC Demo project on the IDE Tomcat Server

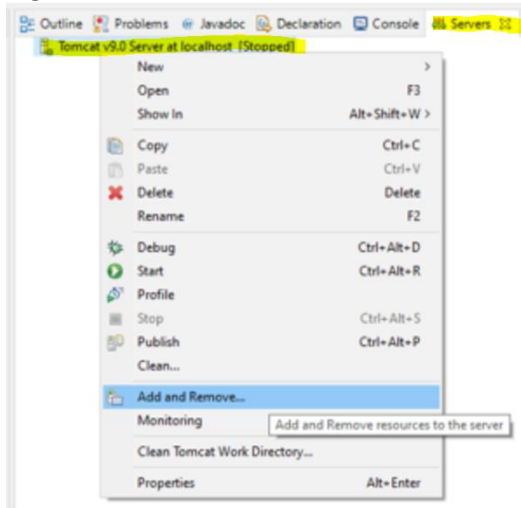
If you **do not** have a Tomcat server installed in the SpringToolSuite IDE proceed to section: [Spring MVC Project Next Steps for Tomcat Server\(s\)](#).

If you **do** have a Tomcat server installed in the SpringToolSuite IDE complete the steps in this section.

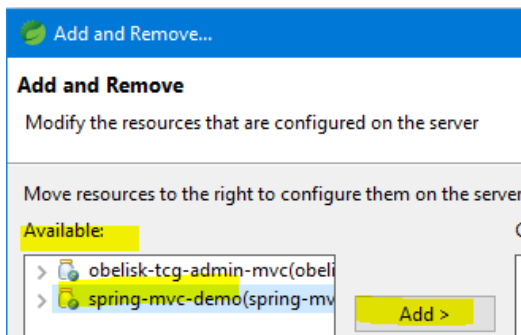
Your location of the Servers tab and window may vary depending on your IDE layout.

In the “Servers” tab

Right click the “Tomcat ...” server → select “Add and Remove...”

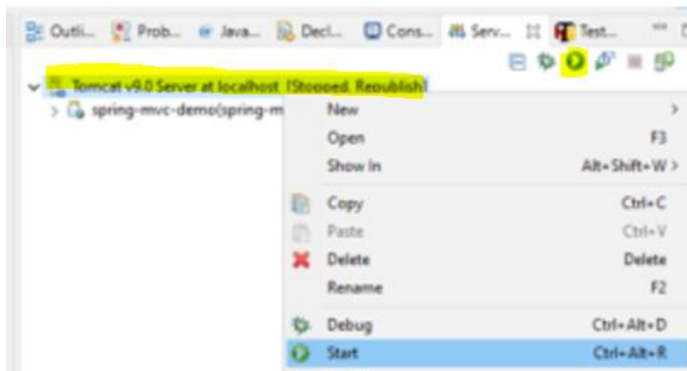


Select our project from “Available:” window
Click “Add >”



Start the Tomcat Server

Right click the server
Select "Start"

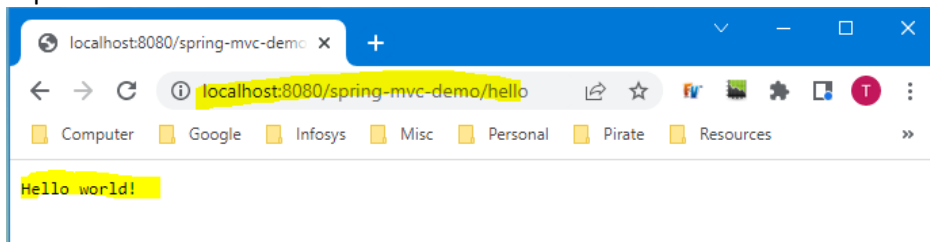


Hello World Test

Enter the following URL in a web browser:

localhost:8080/spring-mvc-demo/hello

Expected results:



Login Test

Postman web (cloud) based issue

At some point the web based version of Postman no longer allows sending a request to localhost. You will need to use the Desktop Agent (**install** it if not already done).

Response



Could not send request

Cloud Agent Error: Can not send requests to localhost. Select a different agent. Use Postman's Desktop Agent

After installing and logging into the desktop version of Postman, you should have access to any Collections in your Workspace created using the web based version.

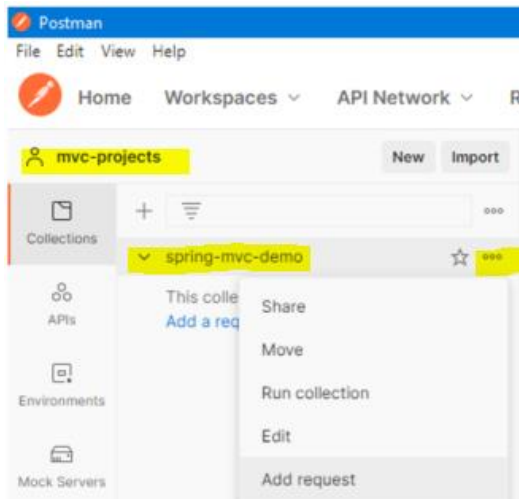
Open Postman Desktop Agent

In your work space you can create a new collection if desired.

Add a Request

In a collection select the menu dots

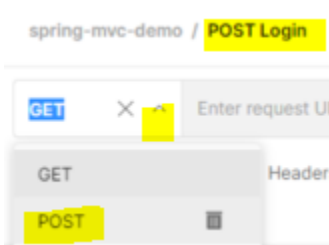
Select "Add request"



Modify the request

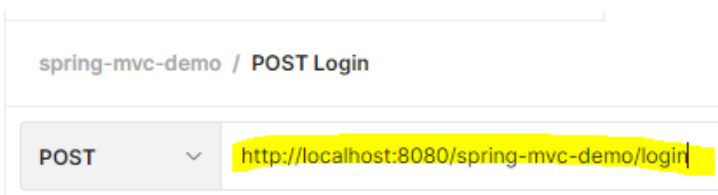
Give the request a name: POST Login

Select "POST" from request type dropdown



Enter the POST URL:

`http://localhost:8080/spring-mvc-demo/login`



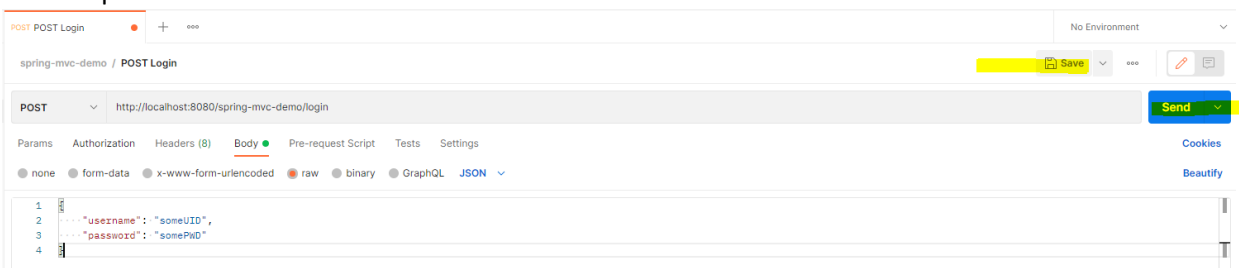
Add login request body
Select tab: "Body"
Click radio button: "raw"
Select "JSON" from type dropdown



Add body definition:

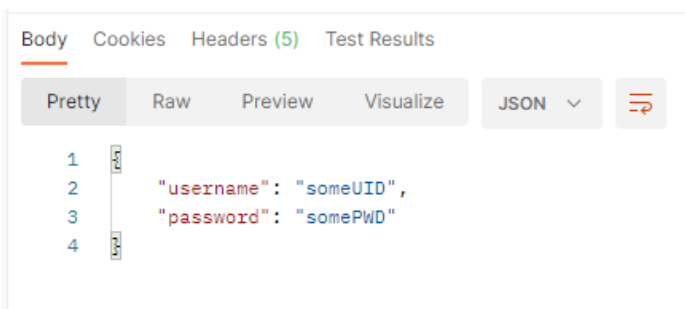
```
{  
  "username": "someUID",  
  "password": "somePWD"  
}
```

Save request
Send request



Expected Response

The spring-mvc-demo is setup to echo the same information it received in JSON format back to the caller.



Spring MVC Project Next Steps for Tomcat Server(s)

After completing the code in this document, follow the steps in the “Tomcat 01 IDE Setup Aug 31st” setup document which walks through the steps to create a Tomcat server in the Spring Tool IDE and run this MVC based applications.

Another Tomcat document “Tomcat 02 localhost Deploy war Sep 8th” detail Tomcat setup on a local computer, deploy and run an MVC application war file and connect to the application through localhost.

Another Tomcat document “Tomcat 03 AWS EC2 Sep 8th” detail Tomcat installation and setup on an AWS EC2 instance, deploy and run an MVC application war file and connect to the application through an AWS URL.