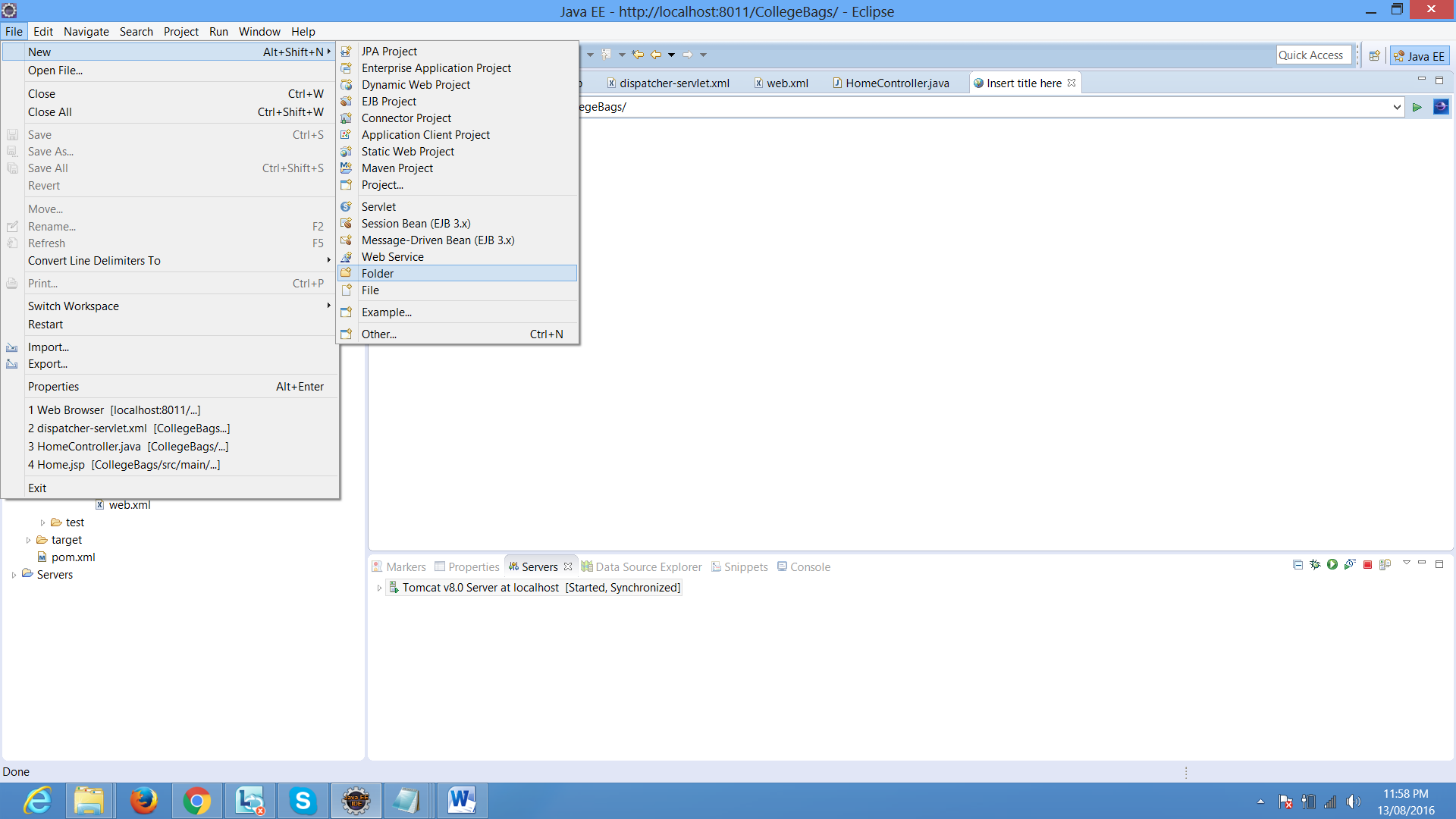
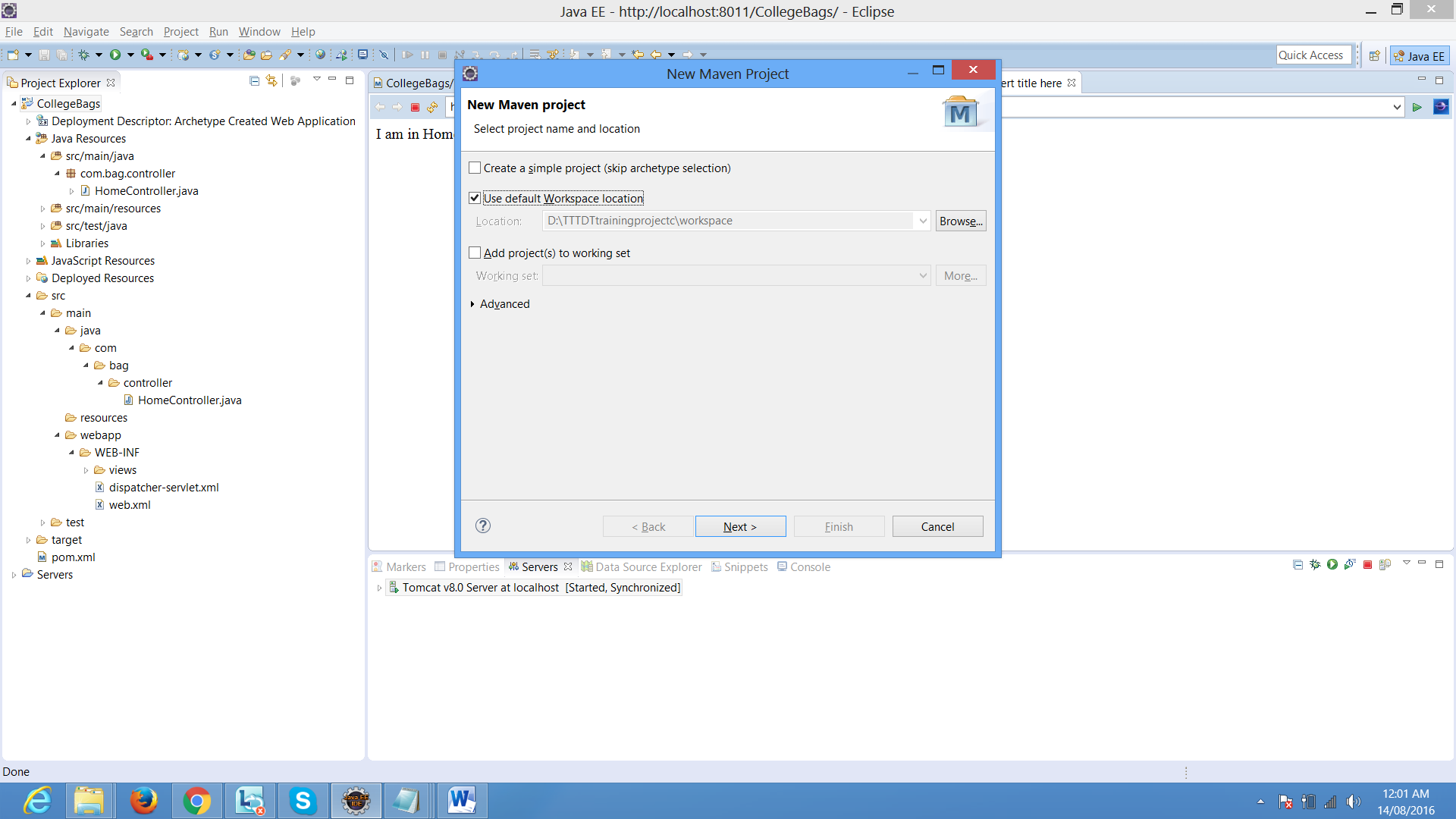
Step By Step Document for First Project :

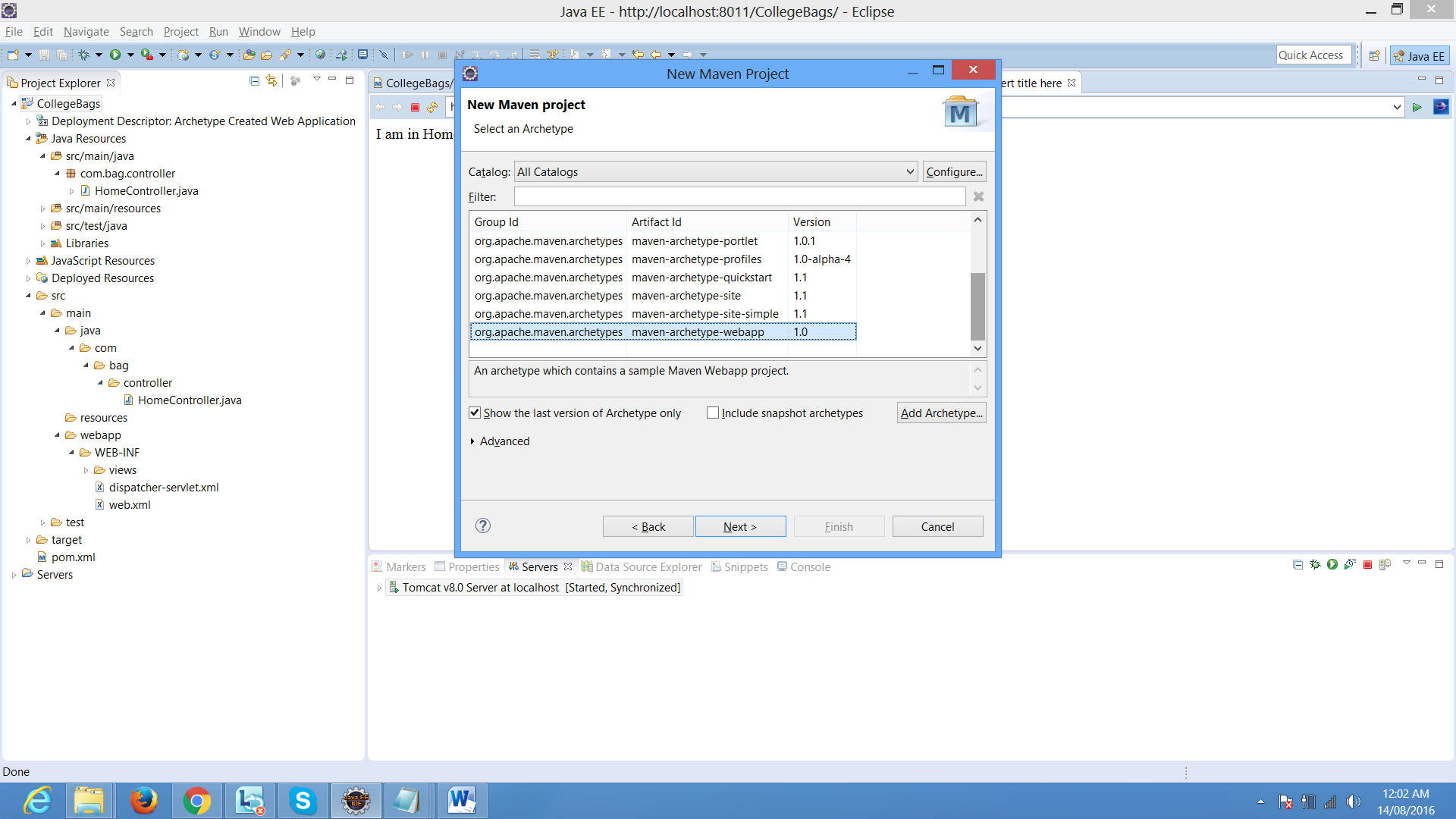
1. Maven Project
2. Spring MVC application
3. Need a concept of Core Java , Jsp , Database , Spring frame work , Hibernate Validation, Spring Security , using XML.
4. To create a Maven Project
5. Go to Eclipse -> New-> Maven Project



1. Once you click maven project we need to give a location where we want to create a our project location:

Selc

1. Click Next
2. Select an Archetype org.apache.maven.archetypes maven-archetype-webapp1.0



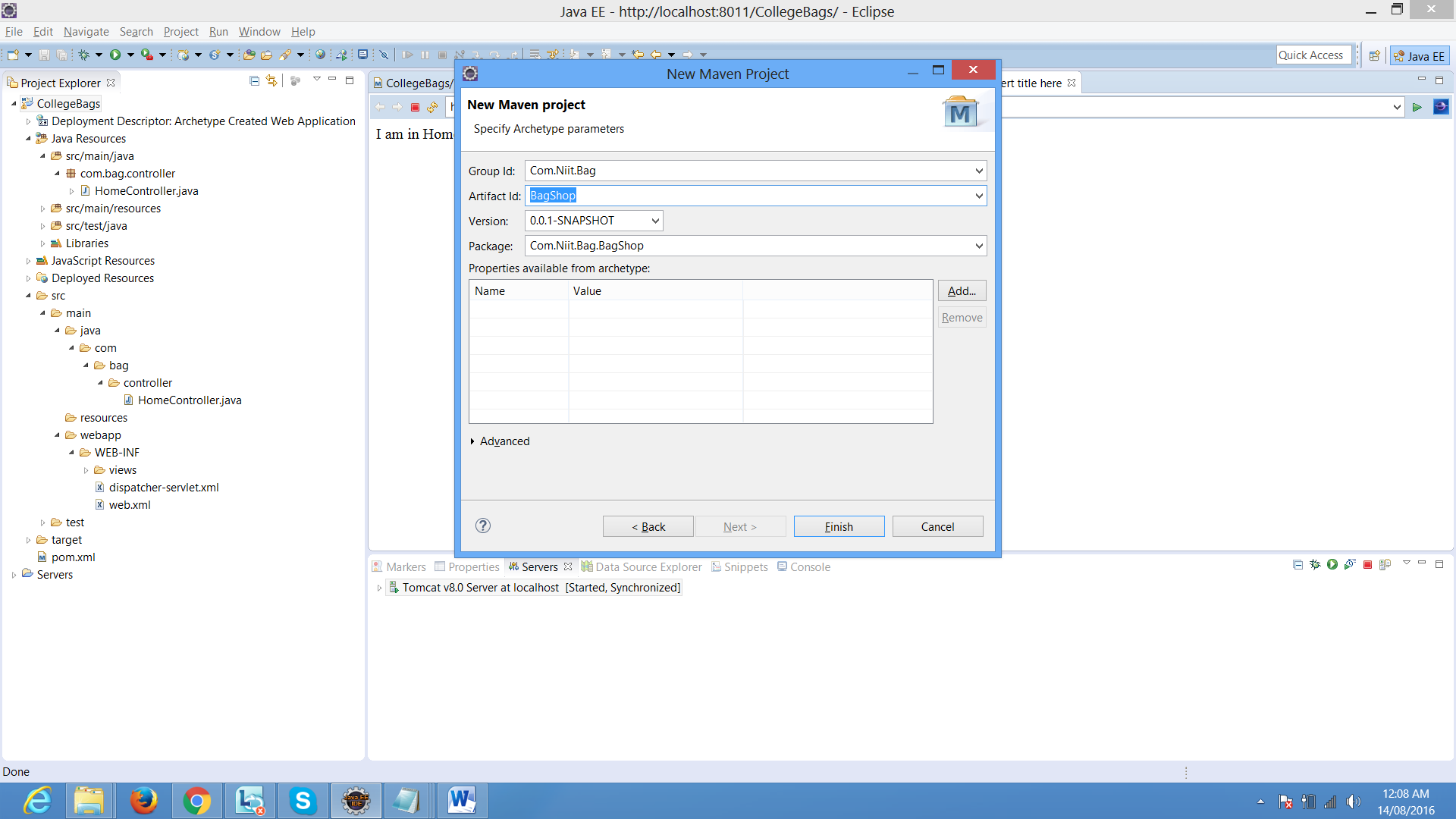
1. Need to give a GroupID, Artiface ID, packagename

When you are giving a groupid : give what type of project your creating a example commercial, organization, government,etc., then followed by your company name and your project name.

Ex: Com.Niit.Bag

Artifact ID: here you have to give a project name

Exa: BagShop

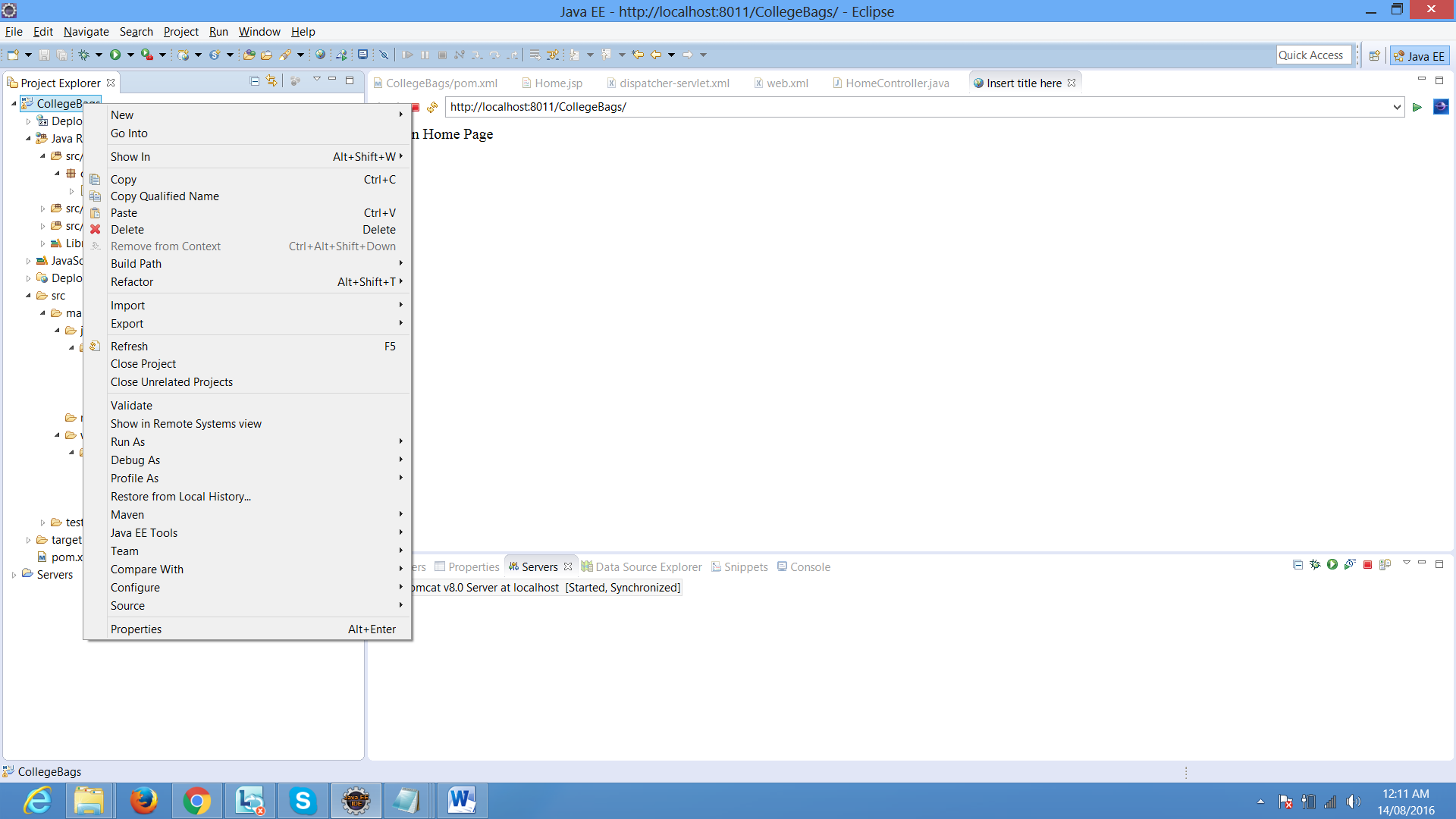


Then click on finish -🡪button

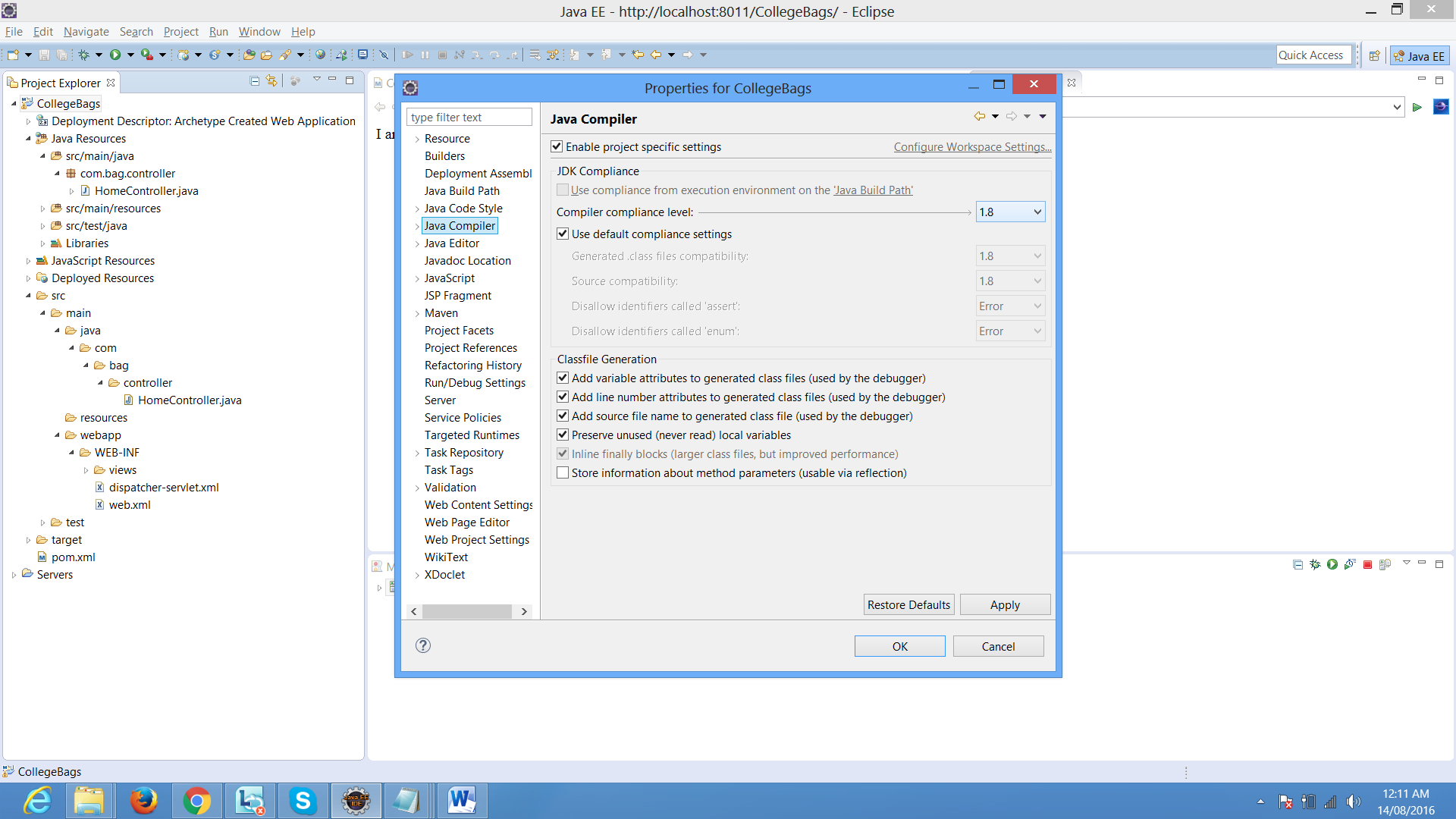
After finish we can see our project in project explorer window.

Before we going to start our project we need to do some set of setups:

1)right ->click on your project -> select properties->go to java compiler

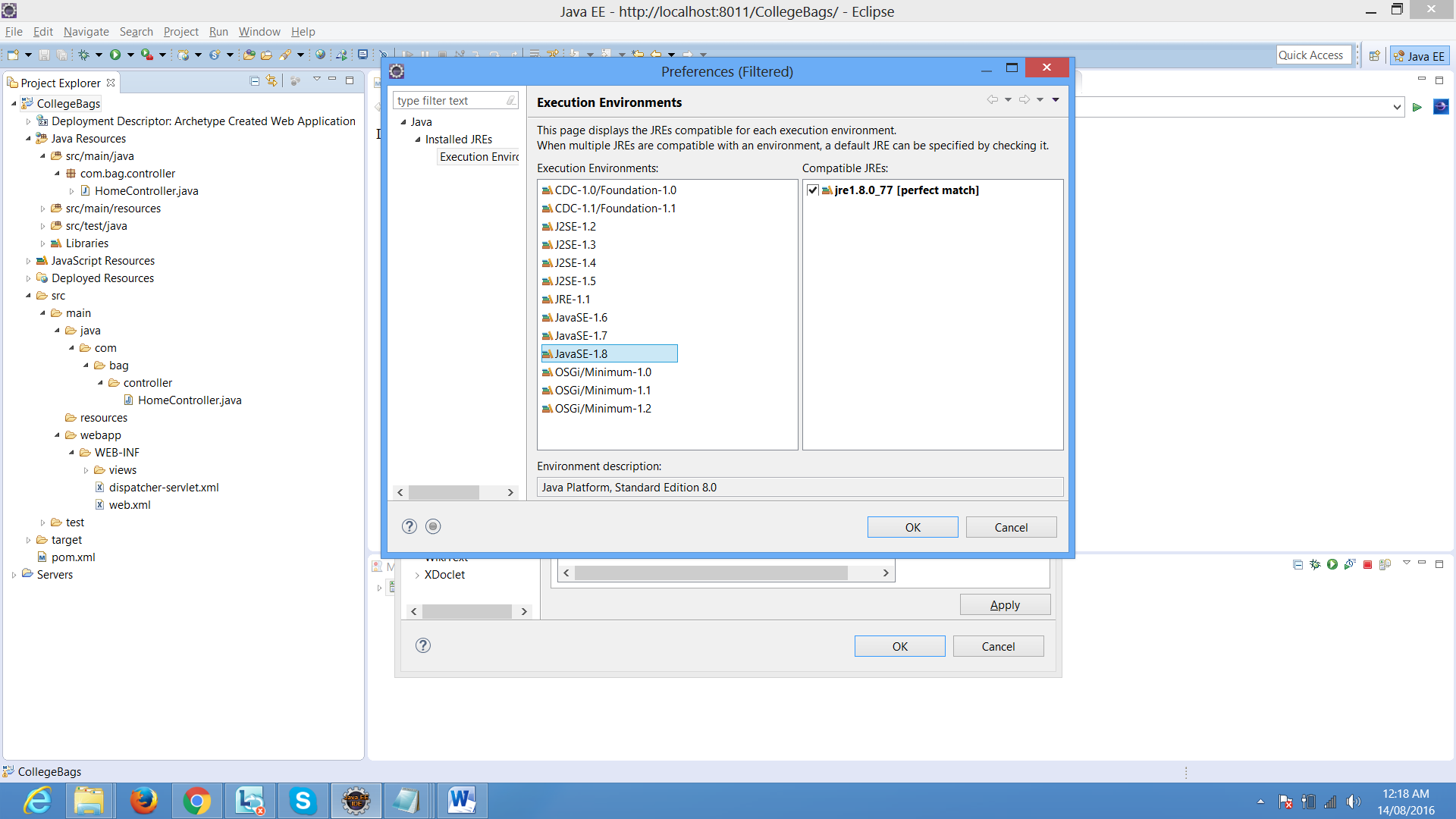


1. Select -> java compiler -> select jdk compliance ->1.8 version->click on apply button



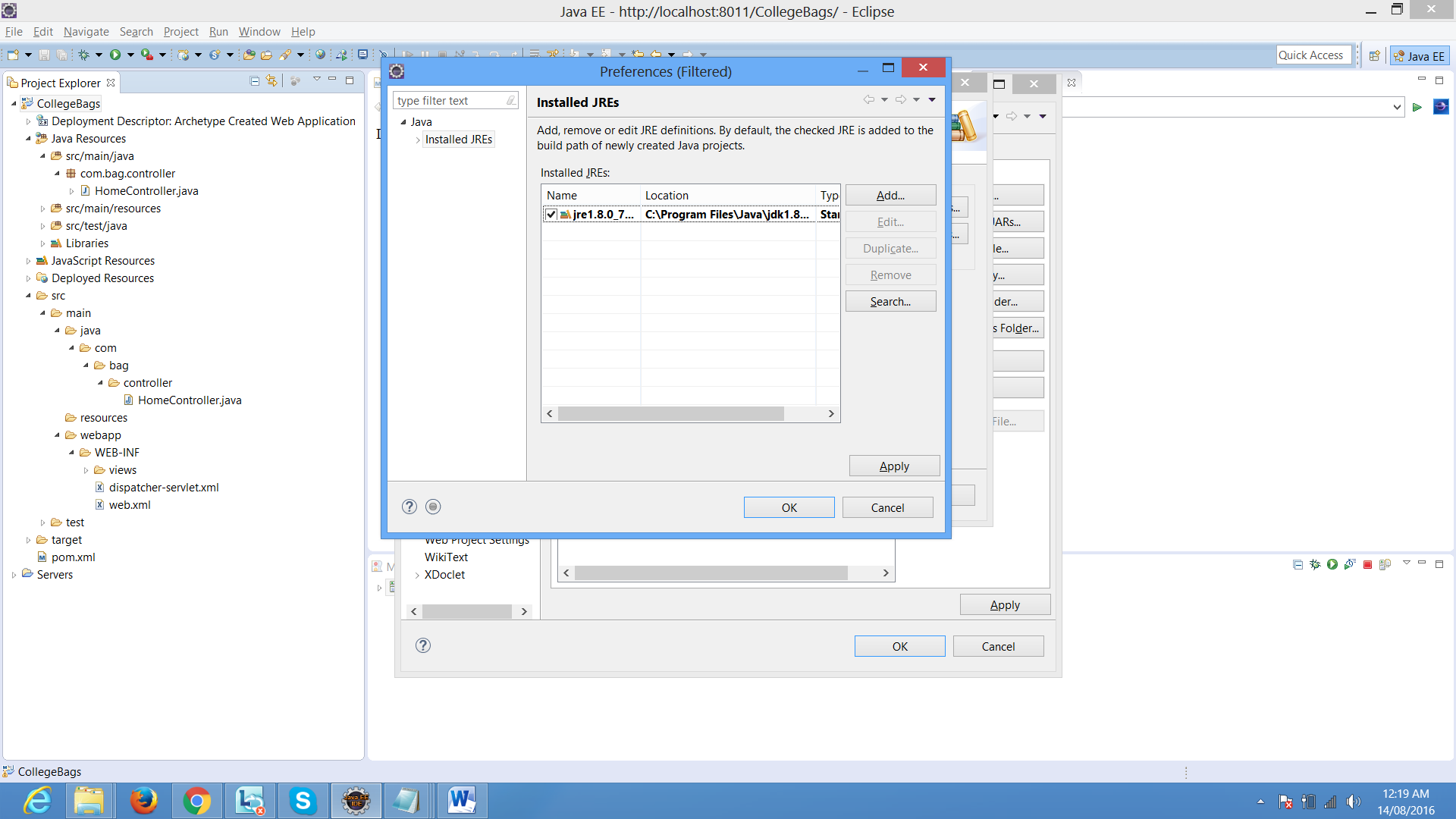
1. Select -> java Build Path->libraries->select jre systemlibrary[j2se1.5] by default jre will be1.5

We need to click on Edit button-> select execution environments jre 1.8 and



Click ok button

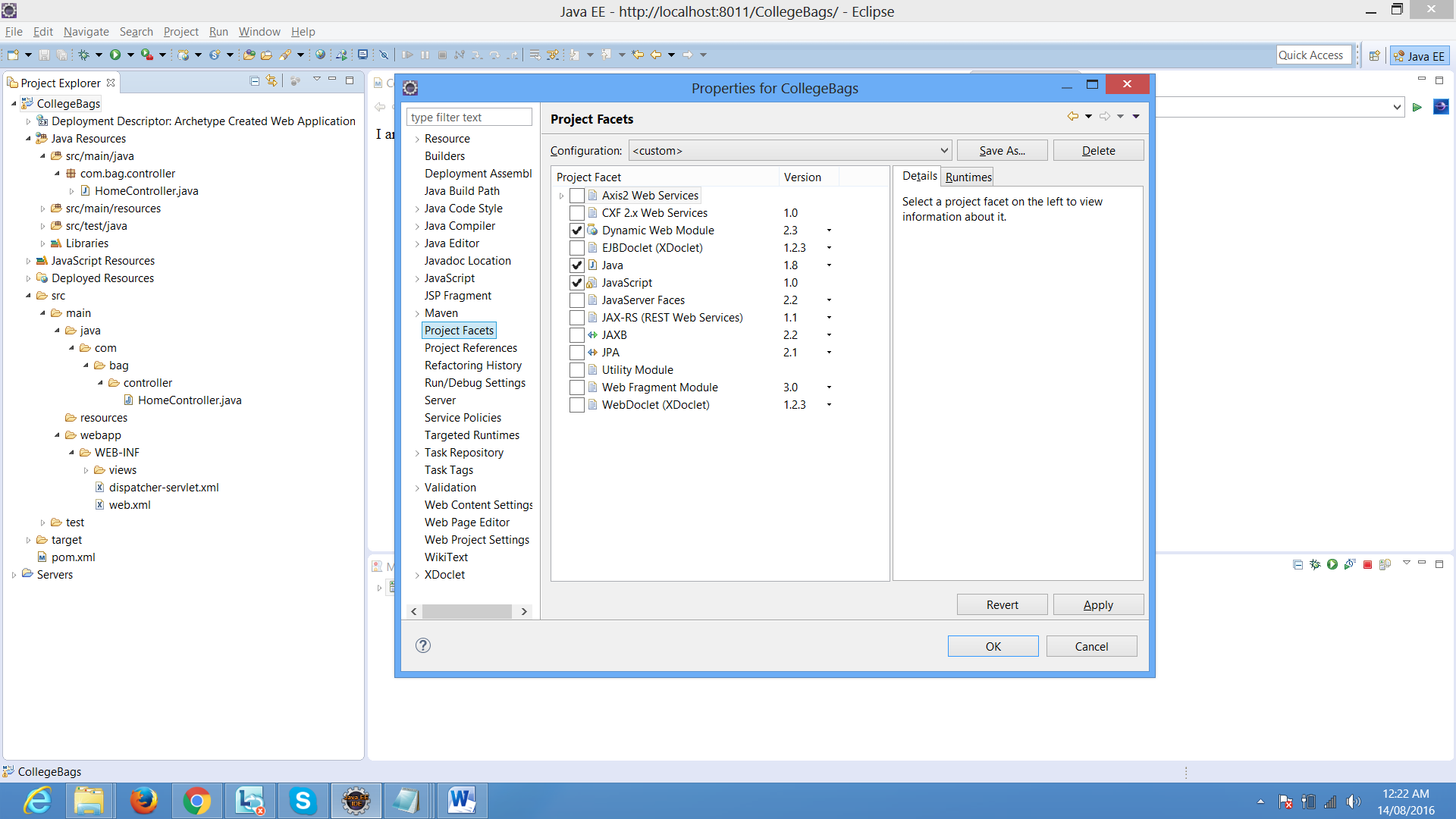
9)select down installed jres-> should be jdk1.8



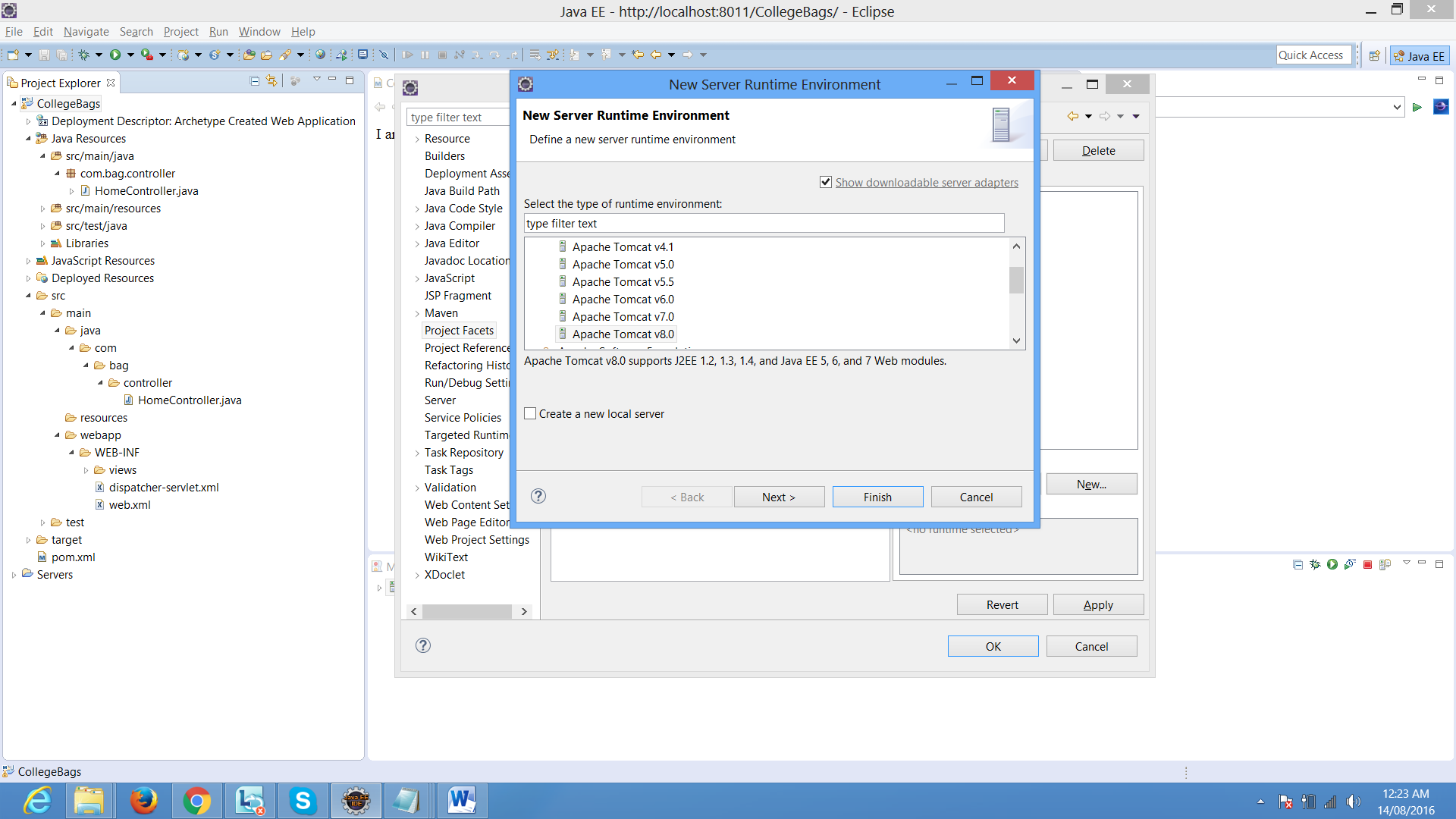
Select jre-> edit -> select a location where you have jdk1.8 location -> then select apply button

And click ok button -> then click on finish button -> then click on apply button

Go to->project facets->> select java 1.8 version and right side runtime -> apache tomcat location



Apache location:



Select apache tomcat where you installed / extracted:

And click on finish button

1. Pom.xml file click
2. Write first dependency for spring mvc frame work
3. After writing each dependency save through internet it get the jar file and put in your project in java-resources->libraries->maven dependency . check which you add dependency those jar files are executed or not.
4. Ex:--

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

After wards in our jsp if we are using jstl tags then we need to add jstl and taglib dependencies

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.1.1</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>1.1.1</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>c</artifactId>

<version>1.1.1</version>

<scope>runtime</scope>

<type>tld</type>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>fmt</artifactId>

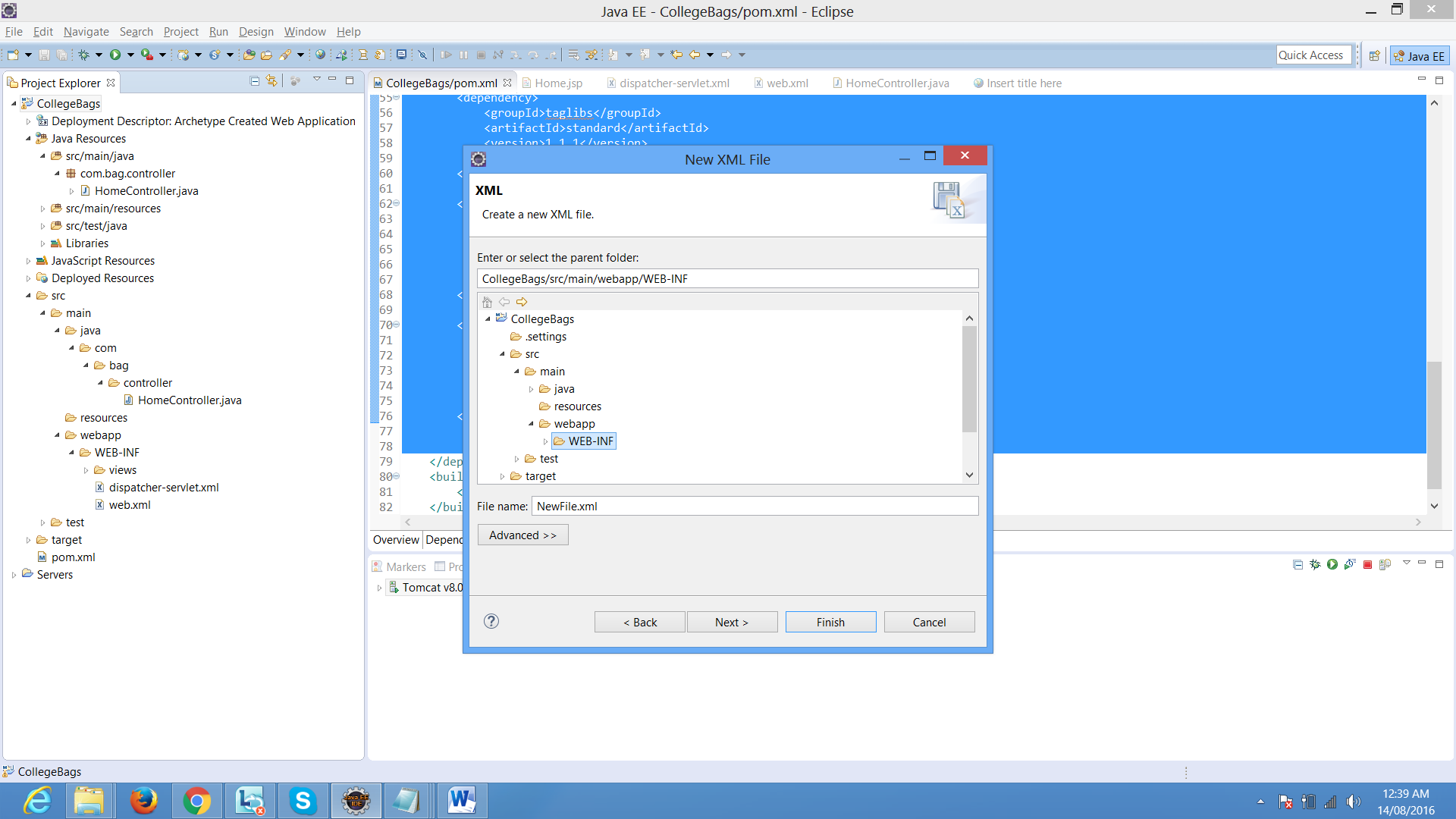
<version>1.1.1</version>

<scope>runtime</scope>

<type>tld</type>

</dependency>

1. Go to Your project in project explorer-> Right click on your project -> create a .xml file in the
2. WEB-INF-> GIVE A name for dispatcher-servlet.xml file



click on finish button

1. Write a dispatcher-servlet.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:p=*"http://www.springframework.org/schema/p"*

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-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc.xsd"*>

<!—this tag for to scan your controller package🡪

<context:component-scan base-package=*"com.bag.controller"* />

<!—this tag for to map your views folder which all jsp is there in web-inf/view folder-->

<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"*>

<value>/WEB-INF/views/</value>

</property>

<property name=*"suffix"*>

<value>.jsp</value>

</property>

</bean>

</beans>

1. Go to web.xml file

<web-app version=*"2.5"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<display-name>Archetype Created Web Application</display-name>

<!—providing information to our container about dispatcher servlet 🡪

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

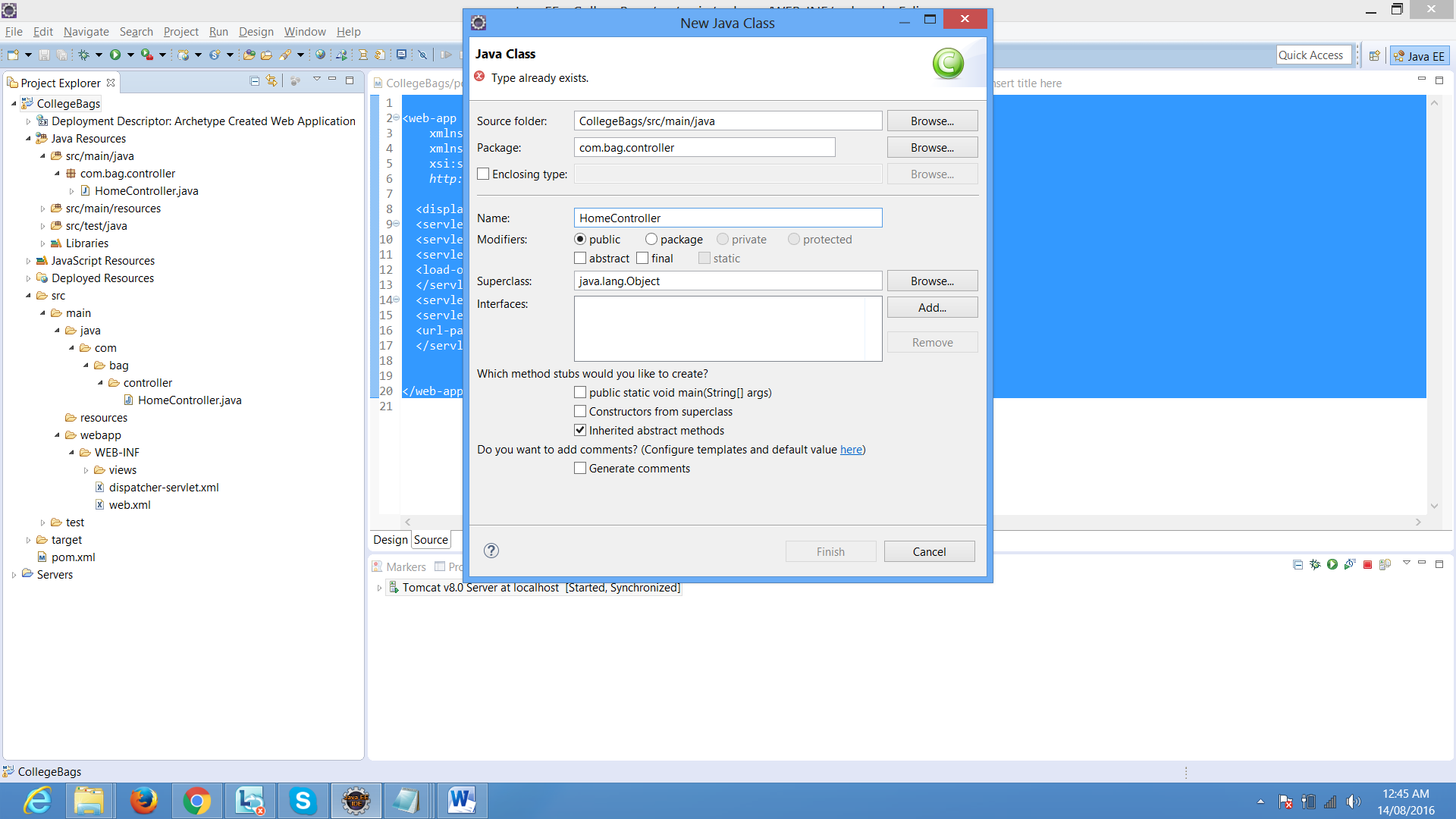
<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

1. Right click on your project create a java class->



Write the class name and package name which your class will be created on that package.

Write the code which your main page should load from controller:

**package** com.bag.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.servlet.ModelAndView;

//@controller is a predefined annotation which we need to specified for our class to be act like as controller

@Controller

**public** **class** HomeController

{

//request mapping is also predefined a annotation for map the address which jsp page //u need to execute .

//here in these example my home page should display as soon as I will run my project

//Without giving a extension of jsp page

@RequestMapping("/")

//user defined function which return a ModelAndView object .

**public** ModelAndView LandingPage()

{

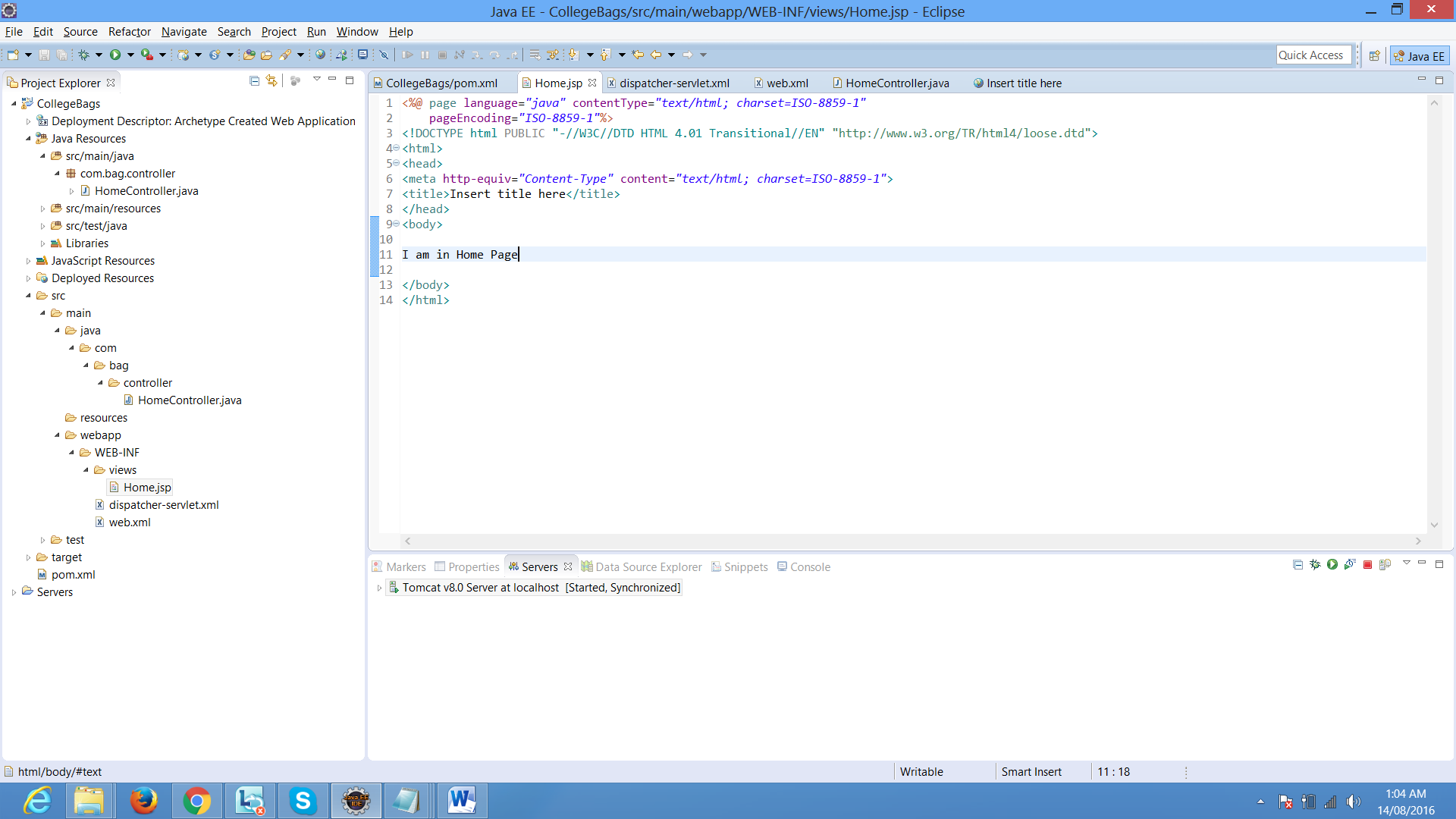
//creating a object for modelandview class and passing the parameter for //jsp page . without extension of jsp page. It will execute your main //page.

**return** **new** ModelAndView("Home");

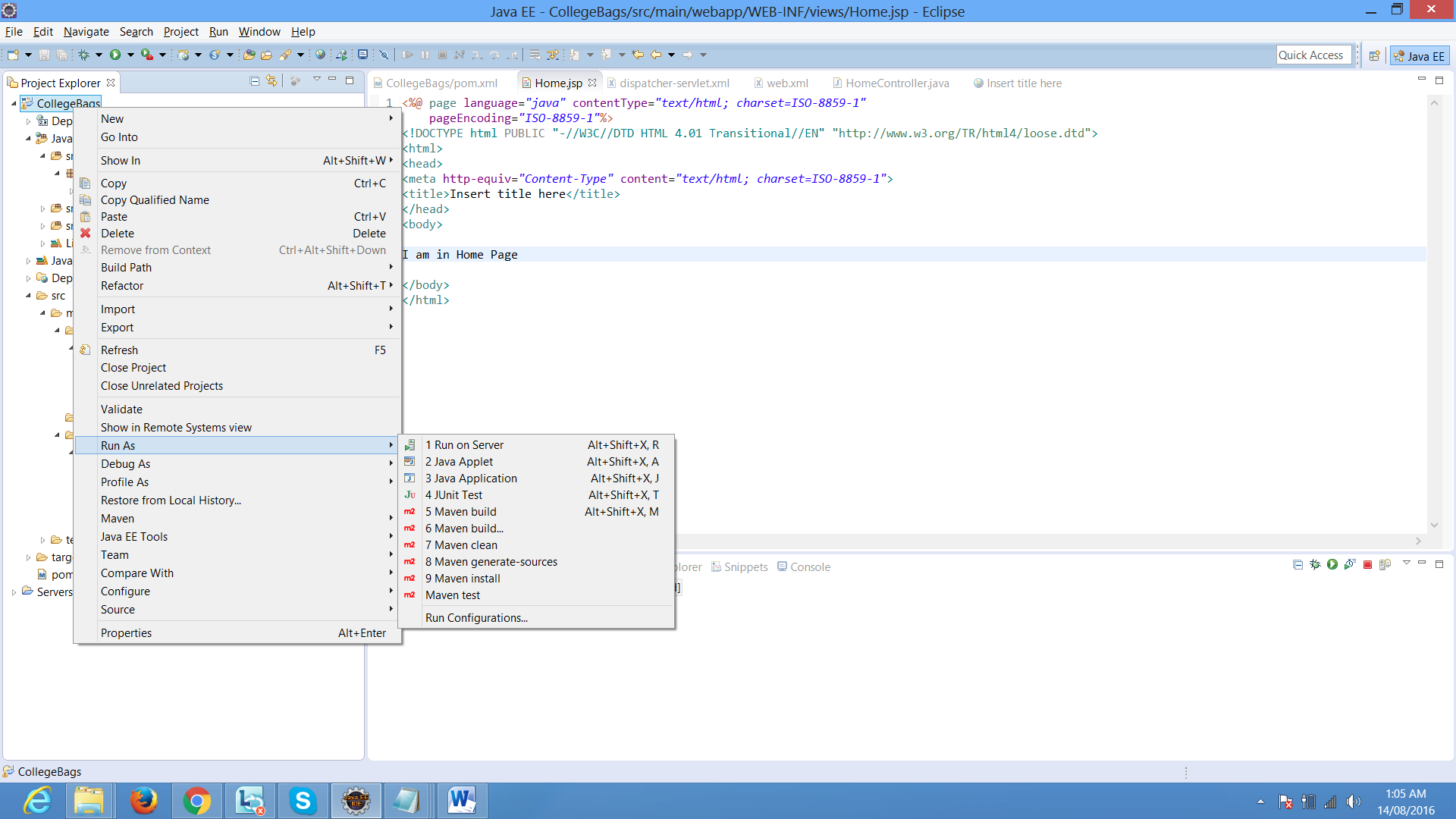
}

}

Just create a jsp page in WEB-INF-views folder one jsp page with name Home.jsp



1. Write a click on your project and select as run as🡪 run on server



Ok write know we are able to get our home page through a controller.

1. We need to create a signup page . So user can register themselves to do online shopping application . So before creating application first we need to create a model class. To set and get the properties values for sign up page. We going to create a hibernate validation and auto generation of table with 0javax.persistance. And in front end spring form.

For this we need to add the dependency that is hibernate validator, javax.persistance, spring orm , hibernate entity manager,h2database, common dbcp

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.1.0.Final</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>5.1.0.Final</version>

</dependency>

<dependency>

<groupId>javax.persistence</groupId>

<artifactId>persistence-api</artifactId>

<version>1.0.2</version>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.191</version>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.2.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-validator -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>5.2.0.Final</version>

</dependency>

To create application-context.xml file in WEB-INF

<?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:aop=*"http://www.springframework.org/schema/aop"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

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/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd"*>

<!-- this bean is for session factory -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"*>

<!-- property for database for source information -->

<property name=*"dataSource"* ref=*"dataSource"*/>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.H2Dialect</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>create</prop>

<prop key=*"hibernate.current\_session\_context\_class"*>thread</prop>

<prop key=*"hibernate.show\_sql"*>true</prop>

</props>

</property>

<!-- to specify for annotated classes -->

<property name=*"annotatedClasses"*>

<list>

<value>com.bag.model.Usermodel</value>

</list>

</property>

</bean>

<!-- scan package -->

<context:component-scan base-package=*"com.bag.controller"*/>

<context:component-scan base-package=*"com.bag.model"*/>

<!-- giving a database driver url address user name and password -->

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>

<property name=*"driverClassName"* value=*"org.h2.Driver"*/>

<property name=*"url"* value=*"jdbc:h2:tcp://localhost/~/newdb"*/>

<property name=*"username"* value=*"sa"*/>

<property name=*"password"* value=*""*/>

</bean>

</beans>

In web.xml file need to add about application-context.xml file and context load listener

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/application-context.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

Ex:

<web-app version=*"2.5"* xmlns=*"http://java.sun.com/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee*

*http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/application-context.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

</web-app>

To create Model class for User Page/ customer data has to register

**package** com.bag.model;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

**import** org.hibernate.validator.constraints.NotEmpty;

@Entity

@Table(name="Usermodel")

**public** **class** Usermodel

{

@Id@GeneratedValue(strategy=GenerationType.***AUTO***)

**private** **int** id;

@NotEmpty(message="should not be empty")

**private** String firstname;

@NotEmpty(message="should not be empty")

**private** String username;

@NotEmpty(message="should not be empty")

**private** String password;

@NotEmpty(message="should not be empty")

**private** String emailid;

@NotEmpty(message="should not be empty")

**private** String mobileno;

@NotEmpty(message="should not be empty")

**private** String address;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getFirstname() {

**return** firstname;

}

**public** **void** setFirstname(String firstname) {

**this**.firstname = firstname;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getEmailid() {

**return** emailid;

}

**public** **void** setEmailid(String emailid) {

**this**.emailid = emailid;

}

**public** String getMobileno() {

**return** mobileno;

}

**public** **void** setMobileno(String mobileno) {

**this**.mobileno = mobileno;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** **boolean** isIsenabled() {

**return** isenabled;

}

**public** **void** setIsenabled(**boolean** isenabled) {

**this**.isenabled = isenabled;

}

**private** **boolean** isenabled;

}

After that execute a the application and check in the database autogenerated table is created.

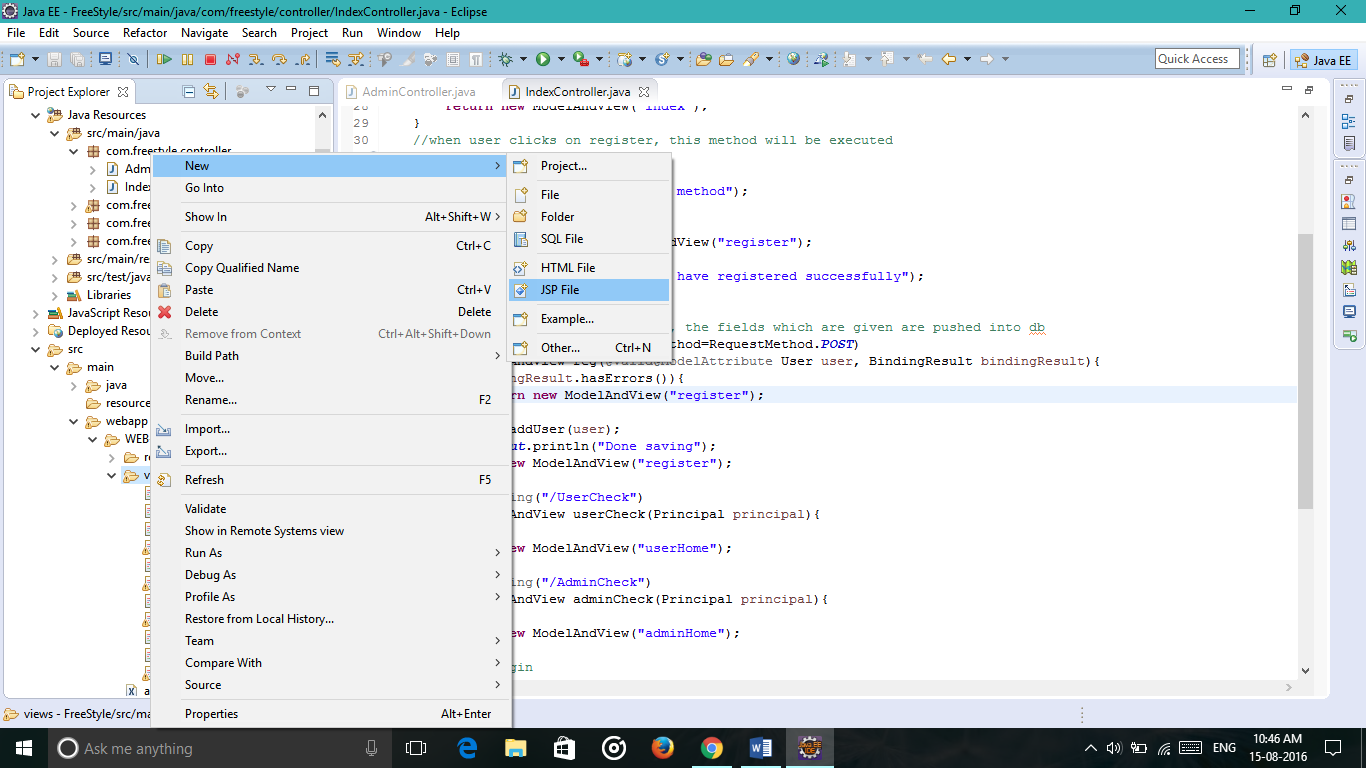
Note : before execute check the model class name given under scan package and annotated classes tag in application-context.xml file and open ur h2 database

Then only u can created.

Design your signup page using spring form

Now create a signup page using spring forms.

Right click on the views folder and create a new jsp page.



Before writing the spring forms, we have to add ‘form’ taglib at the top of the jsp page

[%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %](mailto:%25@taglib%20prefix=%22form%22%20uri=%22http://www.springframework.org/tags/form%22%20%25)

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Register</title>

</head>

<body>

<form:form action=*"reg"* modelAttribute=*"user"* method=*"POST"*>

<table>

<tr>

<td>UserName : </td>

<td><form:input path=*"user\_name"* /><form:errors path=*"user\_name"* style=*"color:red"*></form:errors></td>

</tr>

<tr>

<td>Password : </td>

<td><form:password path=*"password"*/><form:errors path=*"password"* style=*"color:red"*></form:errors></td></tr>

<tr>

<td>EmaiId :</td>

<td><form:input path=*"emailid"*/><form:errors path=*"emailid"* style=*"color:red"*></form:errors></td></tr>

<tr>

<td>Address :</td>

<td><form:input path=*"user\_address"*/><form:errors path=*"user\_address"* style=*"color:red"*></form:errors></td></tr>

<tr colspan=*"2"*>

<td><input type=*"submit"* value=*"Register"*></td>

<td><input type=*"reset"* value=*"Reset"*></td>

</tr>

</table>

</form:form>

</body>

</html>

For this we have to give the path for each form inputs which should match with the variables declared in the model class.

Now for displaying the form, we have to write a method in the home controller.

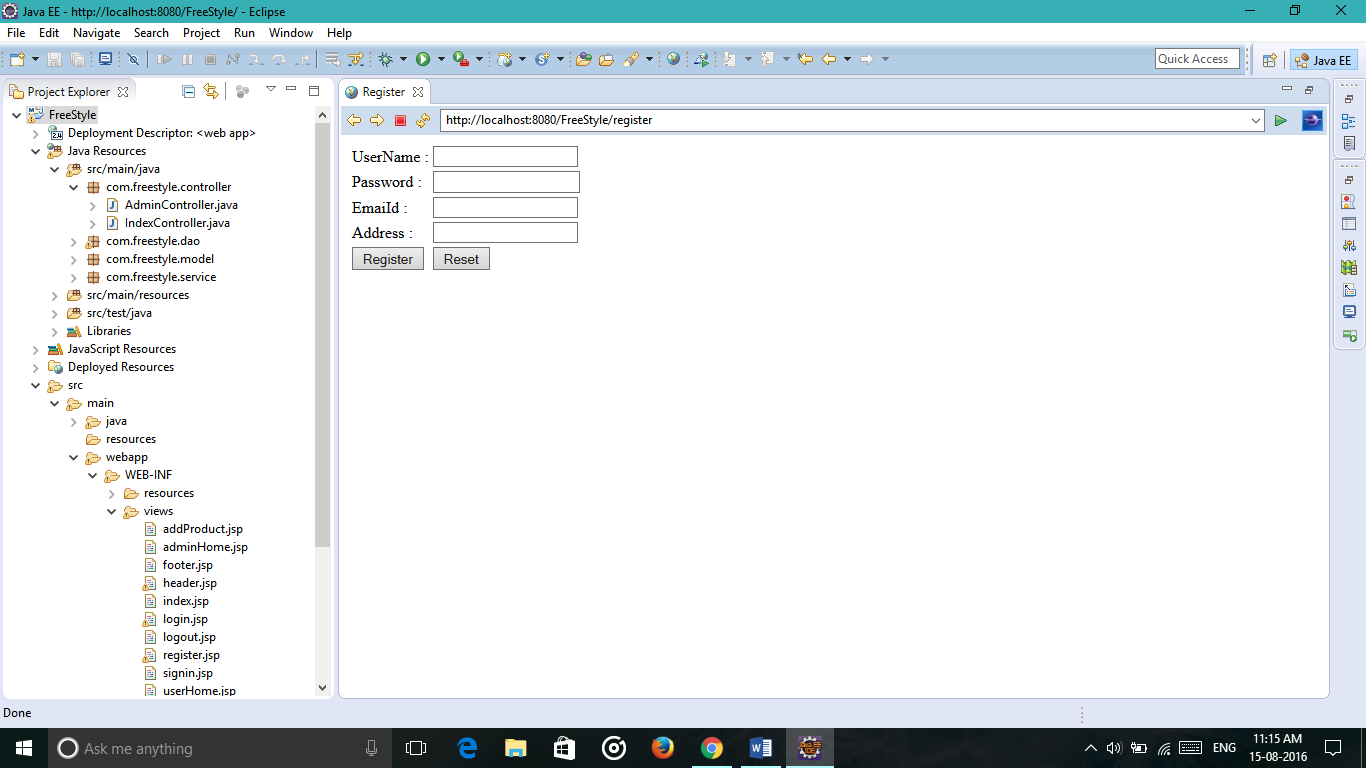
@RequestMapping("/register")

**public** ModelAndView register(){

**return** **new** ModelAndView("register");

}

Now when register is clicked, form is displayed.



Next we have to push the details into the database when register button is clicked, for this we have to write a method in the controller.

We have to create an object of our model class and specify a key value and attribute value. We specify this key value as commandName or modelAttribute in form.

@RequestMapping("/register")

**public** ModelAndView register(){

System.***out***.println("register method");

User user= **new** User();

**return** ModelandView(“register”, “user”, user);

}

Now when user clicks on register, the fields which he has entered has to be entered into the database. For this we have to write a DAO interface it’s implementation and a service it’s implementation.

**DAO:**

package com.freestyle.dao;

import java.util.List;

import com.freestyle.model.User;

public interface UserDAO {

public void addUser(User user);

}

**It’s Implementation:**

package com.freestyle.dao;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import com.freestyle.model.User;

import com.freestyle.model.UserRole;

@Repository

public class UserDAOImpl implements UserDAO {

@Autowired

SessionFactory sessionFactory;

public void addUser(User user) {

System.out.println("UserDAOImpl");

Session session = sessionFactory.getCurrentSession();

Transaction transaction = session.beginTransaction();

session.save(user);

transaction.commit();

}}

**Service:**

package com.freestyle.service;

import java.util.List;

import com.freestyle.model.User;

public interface UserService {

public void addUser(User user);

}

**It`s implementation:**

package com.freestyle.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.freestyle.dao.UserDAO;

import com.freestyle.model.User;

@Service

public class UserServiceImpl implements UserService{

@Autowired

UserDAO userDAO;

public void addUser(User user) {

System.out.println("User service");

userDAO.addUser(user);

}

}

After writing these classes, we have to write a method in the controller which pushes the details into the database.

And for hibernate validations, We have to specify @valid annotation before the model attribute and we have to import a class called BindingResult which includes a method called hasErrors() which checks wheather we have given all the required fields are not.

@RequestMapping(value="/reg", method=RequestMethod.***POST***)

**public** ModelAndView reg(@Valid@ModelAttribute User user, BindingResult bindingResult){

**if**(bindingResult.hasErrors()){

**return** **new** ModelAndView("register");

}

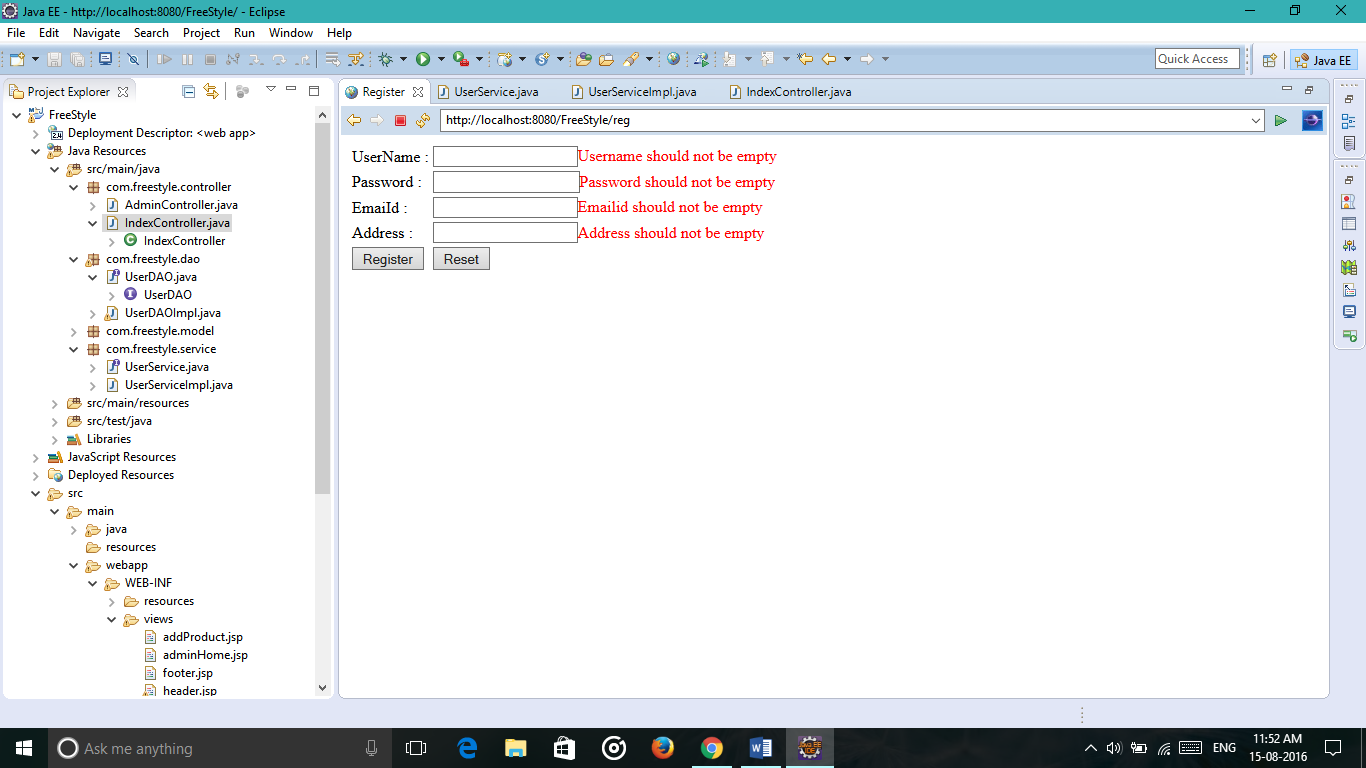
service.addUser(user);

System.***out***.println("Done saving");

**return** **new** ModelAndView("register");

}

When we haven’t given any fields and click register button the validations will be like this



We can give any color to the text b specifying a style attribute in the <form:errors> tag.

**Spring Security**

Main use of spring security is to provide security for the website and to see that user shouldn’t be able to login through admin details and admin shouldn’t be able to login using user details.

For this we create a separate xml file in the WEB-INF folder . Before creating the file we have to add certain dependencies related to spring security

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>3.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>3.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>3.2.5.RELEASE</version>

</dependency>

<beans:beans xmlns=*"http://www.springframework.org/schema/security"*

xmlns:beans=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd"*>

<!-- <http>

<intercept-url pattern="/AdminCheck\*" access="ROLE\_USER"/>

<http-basic/>

</http> -->

<!-- <http>

<intercept-url pattern="/AdminCheck\*" access="ROLE\_ADMIN"/>

<intercept-url pattern="/CustomerCheck\*" access="ROLE\_USER"/>

<form-login/>

<logout logout-success-url="/logout"/>

</http>-->

<http>

<intercept-url pattern=*"/AdminCheck\*"* access=*"ROLE\_ADMIN"*/>

<form-login login-page=*"/login"* default-target-url=*"/AdminCheck"*/>

<intercept-url pattern=*"/UserCheck\*\*"* access=*"ROLE\_USER"*/>

<form-login login-page=*"/login"* default-target-url=*"/UserCheck"*/>

<logout logout-success-url=*"/logout"*/>

</http>

<!-- <http>

<intercept-url pattern="/AdminCheck\*" access="ROLE\_ADMIN"/>

<form-login login-page="/login" default-target-url="/AdminCheck"/>

<logout logout-success-url="/logout"/>

</http> -->

<authentication-manager>

<authentication-provider>

<jdbc-user-service data-source-ref=*"dataSource"*

users-by-username-query=*"select user\_name,password,enabled from user where user\_name=?"*

authorities-by-username-query=*"select u1.user\_name, u2.authority from user u1, userrole u2 where u1.user\_id = u2.user\_id and u1.user\_name =?"* />

</authentication-provider>

</authentication-manager>

</beans:beans>

Add the path of spring-security.xml to the web.xml

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/applicationContext.xml,/WEB-INF/spring-security.xml</param-value>

</context-param>

Also configure the filter by adding spring security filer in the web.xml

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

Now create login and logout pages in views folder and their respective methods in the homeController. For this we have to create another model in the model called UserRole in the model package which has the same ID as the User class and authority which specifies certain person as user or admin.

package com.freestyle.model;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class UserRole {

@Id

private int user\_id;

private String authority;

public int getId() {

return user\_id;

}

public void setId(int user\_id) {

this.user\_id = user\_id;

}

public String getAuthority() {

return authority;

}

public void setAuthority(String authority) {

this.authority = authority;

}

}

Now the table for this will be autocreated in the database because of the annotation @Entity. For setting the same id as the user table we have to add some code to the adduser method in UserDaoImpl.

user.setEnabled(**true**);

session.save(user);

UserRole userrole = **new** UserRole();

userrole.setAuthority("ROLE\_USER");

userrole.setId(user.getUser\_id());

session.save(userrole);

When a person has to be given admin permission , the data must be edited un the database itself.

Now we have to create two jsp pages userHome and adminHome

**UserHome:**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

User Home<a href=*"logout"*>Logout</a>

</body>

</html>

**AdminHome:**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

Admin Home<a href=*"logout"*>Logout</a>

</body>

</html>

Now we will write two methods in the controller called AdminCheck and UserCheck.

@RequestMapping("/UserCheck")

**public** ModelAndView userCheck(Principal principal){

**return** **new** ModelAndView("userHome");

}

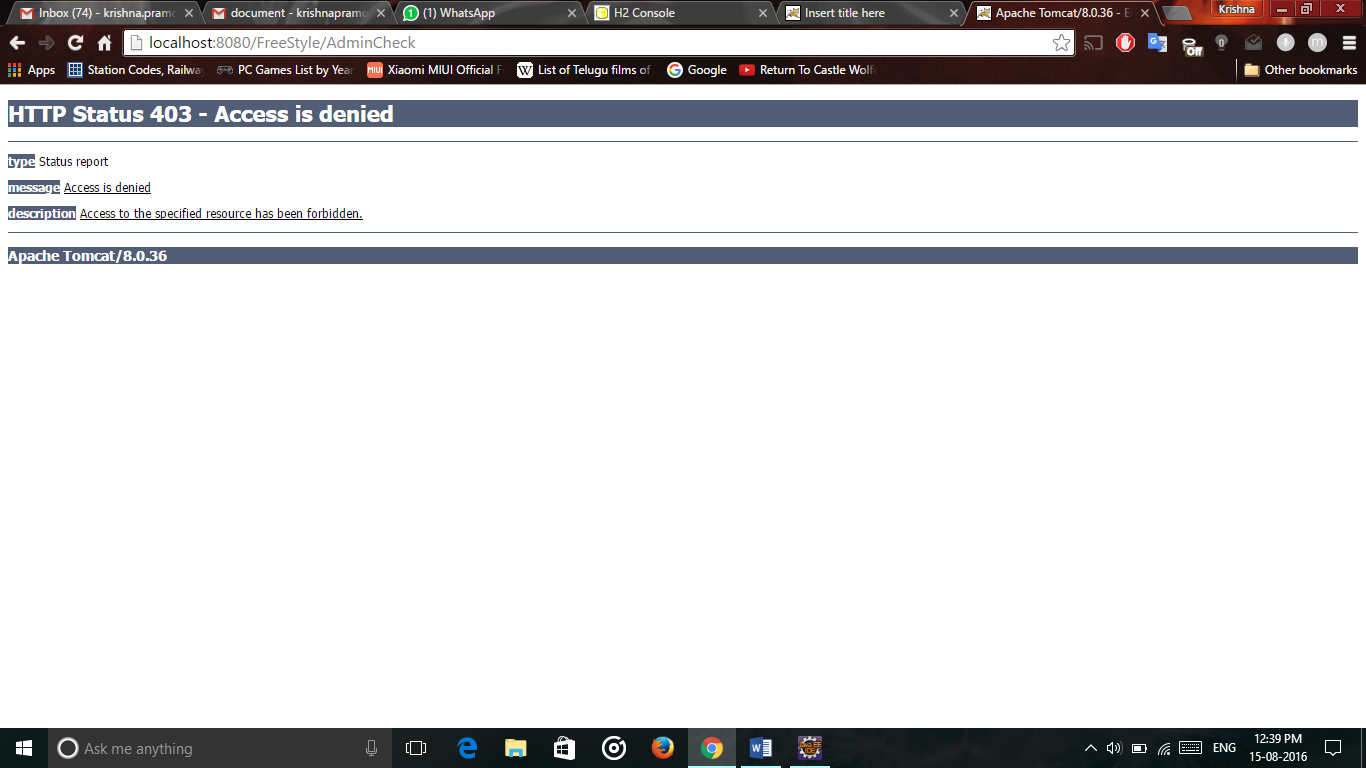
@RequestMapping("/AdminCheck")

**public** ModelAndView adminCheck(Principal principal){

**return** **new** ModelAndView("adminHome");

}

Now change the href attribute of login in the home page to UserCheck to enable security.

Whenever a User tries to use admin credentials when he is logged in, the security stops him and shows Access denied. 

And the same goes for admin.

Admin has certain rights to view user, add a product, delete user etc..

To view the users, we have to add a piece of code to DAO and service.

**DAO:**

public List<User> viewUser();

**It’s Implementation:**

**public** List<User> viewUser() {

Session session= sessionFactory.getCurrentSession();

Transaction transaction=session.beginTransaction();

@SuppressWarnings("unchecked")

List<User>list= session.createCriteria(User.**class**).list();

**return** list;

}

**Service:**

public List<User> viewUser();

**It’s implementation:**

**public** List<User> viewUser() {

List<User>list= userDAO.viewUser();

**return** list;

}

Now create a jsp page called viewUser and using angular js , we can display the table of users who are registered.

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<%

response.setHeader("pragma", "no-cache");

response.setHeader("Cache-control", "no-cache, no-store, must-revalidate");

response.setHeader("Expires", "0");

%>

<title>FreeStyle</title>

<meta charset=*"utf-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1"*>

<head>

<link rel=*"stylesheet"* href=*"http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"*>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"*></script>

<script src=*"http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"*></script>

<script src=*"http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"*></script>

<script>document.write('<base href="' + document.location + '" />');</script>

<link href=*"./resources/css/bootstrap.min.css"* rel=*"stylesheet"* media=*"screen"*>

<link href=*"./resources/css/style.css"* rel=*"stylesheet"* media=*"screen"*>

<script data-require=*"angular.js@1.0.x"* src=*"http://code.angularjs.org/1.0.7/angular.min.js"* data-semver=*"1.0.7"*></script>

<style>

/\* Remove the navbar's default margin-bottom and rounded borders \*/

*.navbar* {

margin-bottom: *0*;

border-radius: *0*;

}

/\* Add a gray background color and some padding to the footer \*/

**footer** {

background-color: *#f2f2f2*;

padding: *25px*;

}

*.carousel-inner* > *.item* > **img,**

*.carousel-inner* > *.item* > **a** > **img** {

width: *70%*;

margin: *auto*;

</style>

</head>

<body>

<div class=*"container"*>

<div ng-app=*"myApp"* ng-controller=*"dataCtrl"*>

Data:${ss}

Enter Name/Email ID: <input type=*"text"* ng-model=*"search"*>&nbsp&nbsp<span class=*"glyphicon glyphicon-search"*></span>

<hr></hr>

<table class=*"table table-striped"*>

<tr>

<th>UserId</th>

<th>UserName</th>

<th>EmailId</th>

<th>Address</th>

</tr>

<tr ng-repeat=*"resource in names | filter:search"*>

<td>{{resource.user\_id}}</td>

<td>{{ resource.user\_name}}</td>

<td>{{ resource.emailid}}</td>

<td>{{ resource.user\_address}}</td>

</tr>

</table>

</div>

<script>

angular.module('myApp',[]).controller('dataCtrl',**function**($scope)

{

$scope.names=${listofusers};

$scope.orderByMe=**function**(x)

{

$scope.myOrderBy=x;

}

});

</script>

</body>

</html>

But angular js cannot understand java. So we have to convert it to json which is a string format. After conversion, we can display the information using angular js.

For converting this to json, we need Jackson api. For this we need to add two more dependencies

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.13</version>

</dependency>

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-core-asl</artifactId>

<version>1.9.13</version>

</dependency>

Create an anchor tag Inside adminHome as viewUser.

Now create a new controller called admin controller inside the controller package and map the viewUser.

package com.freestyle.controller;

import java.io.IOException;

import java.util.List;

import org.codehaus.jackson.JsonGenerationException;

import org.codehaus.jackson.map.JsonMappingException;

import org.codehaus.jackson.map.ObjectMapper;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

import com.freestyle.model.Product;

import com.freestyle.model.User;

import com.freestyle.service.UserService;

@Controller

public class AdminController {

@Autowired

UserService userService;

@RequestMapping("/viewUser")

public ModelAndView viewUser() throws JsonGenerationException, JsonMappingException, IOException {

List<User> list = userService.viewUser();

ObjectMapper object = new ObjectMapper();

String jsonlist = object.writeValueAsString(list);

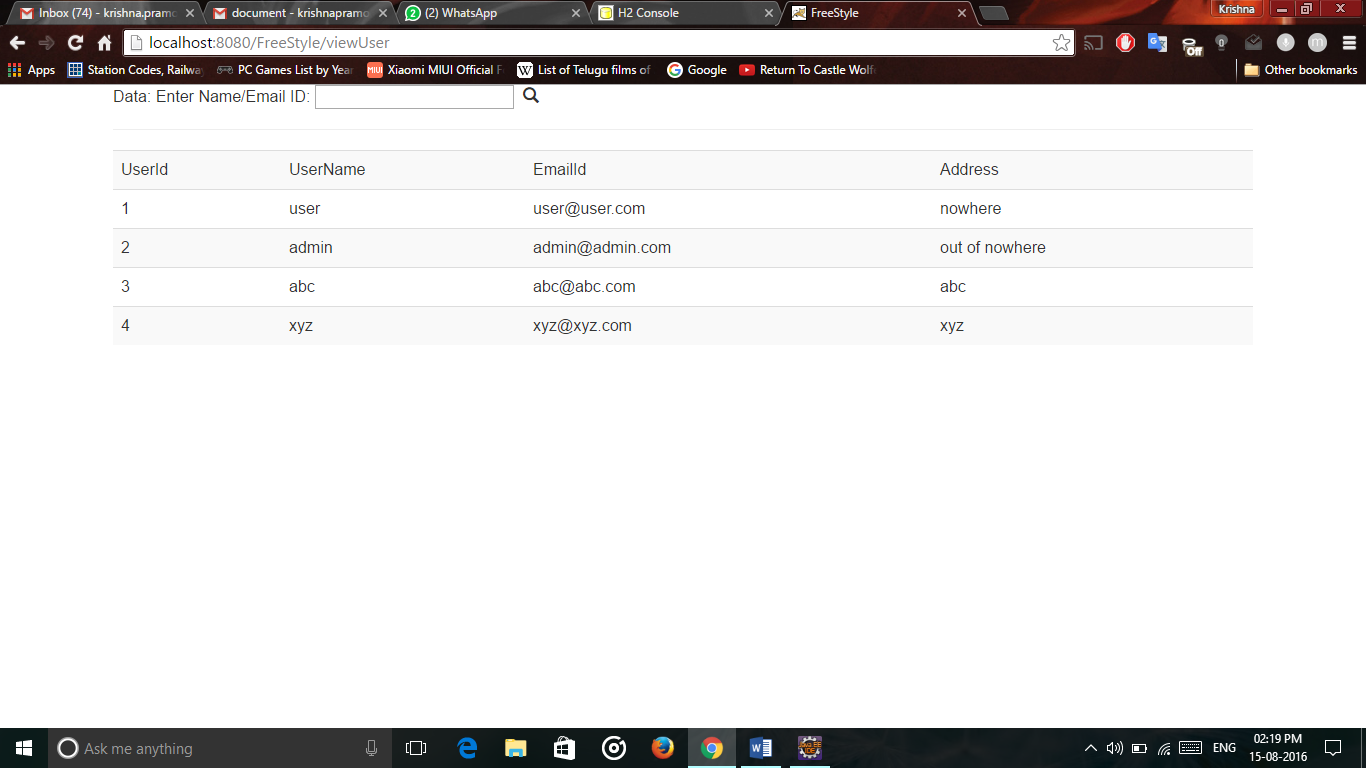
System.out.println(("List is:" + list));

return new ModelAndView("viewUser", "listofusers", jsonlist);

}

}

Now when view user is clicked on admin home page, it displays the list of users.



Now for adding products, we have to create a new model class called products

package com.freestyle.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Transient;

@Entity

public class Product {

@Id@GeneratedValue

private int productid;

private String productname;

private String productdescription;

private double productprice;

@Transient

private String category;

public int getProductid() {

return productid;

}

public void setProductid(int productid) {

this.productid = productid;

}

public String getProductname() {

return productname;

}

public void setProductname(String productname) {

this.productname = productname;

}

public String getProductdescription() {

return productdescription;

}

public void setProductdescription(String productdescription) {

this.productdescription = productdescription;

}

public double getProductprice() {

return productprice;

}

public void setProductprice(double productprice) {

this.productprice = productprice;

}

public String getCategory() {

return category;

}

public void setCategory(String category) {

this.category = category;

}

}

The table will be auto created and id is also auto generated.

Create a jsp page called addProducts which is a spring form for adding products list to the database.

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<table>

<form:form action=*"addProduct"* modelAttribute=*"addProduct"* enctype=*"multipart/form-data"*>

<tr>

<td>Product Name:</td>

<td><form:input path=*"productname"*/></td>

</tr>

<tr>

<td>Product description:</td>

<td><form:input path=*"productdescription"*/></td>

</tr>

<tr>

<td>Product Price:</td>

<td><form:input path=*"productprice"*/></td>

</tr>

<tr>

<td>Category</td>

<td><form:select path=*"category"*>

<form:option value=*"Nike"*></form:option>

<form:option value=*"Puma"*></form:option>

<form:option value=*"Adidas"*></form:option>

</form:select></td></tr>

<tr>

<td><input type=*"file"* name=*"file"* /></td>

</tr>

<tr>

<td><input type=*"submit"* value=*"Add Product"* /></td></tr>

</form:form>

</table>

</body>

</html>

Also provide an anchor tag in admin home page called addproducts.

For mapping the products page, we have to create the method in the controller

@RequestMapping("/addProduct")

**public** ModelAndView addProduct(){

Product product= **new** Product();

System.***out***.println("Products entered");

**return** **new** ModelAndView("addProduct", "addProduct",product);

}

By using this, we can display the form and add the products to database 