**A**

**Project On**

**To DO List**

**Submitted towards Digital Transformation**

***DT JAVA BATCH-III***

**Done By**

**Name : K.SivaKrishna**

***Student Id :* s170060100075**

***Center : Basheerbagh***

# Introduction

The project titled “To Do List” is to be developed in a rapid pace for any tasks to be performed. The project To Do List is a web project, enables a learner to implement the various tasks and how to create add task, edit task , update task.

# Objective

The project learning, implementation and methodology is centrered on Project Based Learning will encompass the various aspects of enterprise app development and covers the following objectives:

* Understand and implement object-oriented concepts using Java technology programs
* Write SQL queries to retrieve, manage, and manipulate data
* Design responsive Web/enterprise apps using HTML5, CSS3, and various JavaScript frameworks such as Bootstrap & AngularJS
* Build persistent and loosely coupled Web apps using Hibernate and Spring technologies
* Build RESTFull Web Services.

Software Requirements:

* Operating System Windows 7, 8, 10 or above
* Eclipse Mars IDE
* Apache tomcat 8 server
* Github Desktop
* Java Development Kit version 8
* H2 Database
* Bootstrap Scripts
* Angular JS Scripts

# Creating Maven Project on Eclipse

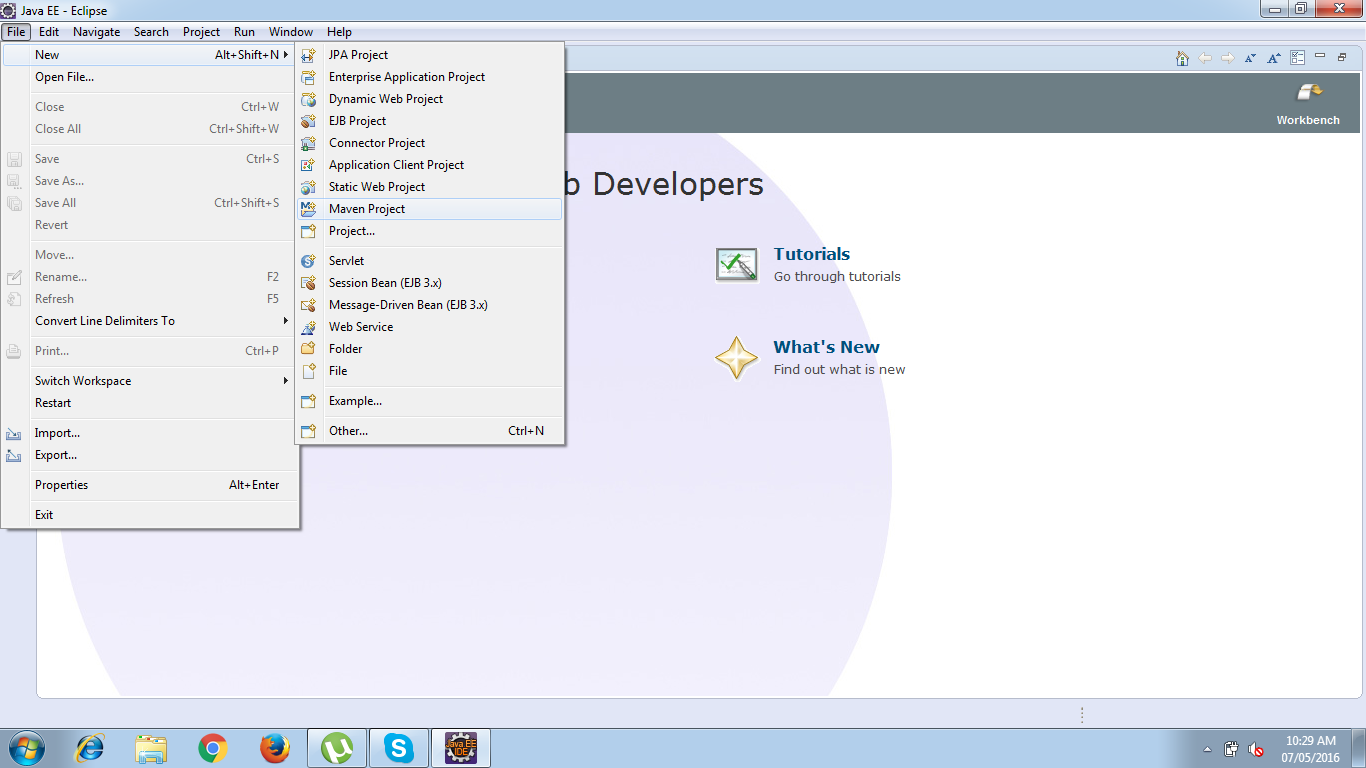
**Prerequisite**

You will need to install the Eclipse IDE Maven plugin found at the Eclipse Marketplace.

Video Url : https://www.youtube.com/watch?v=YeC7XQho-O0

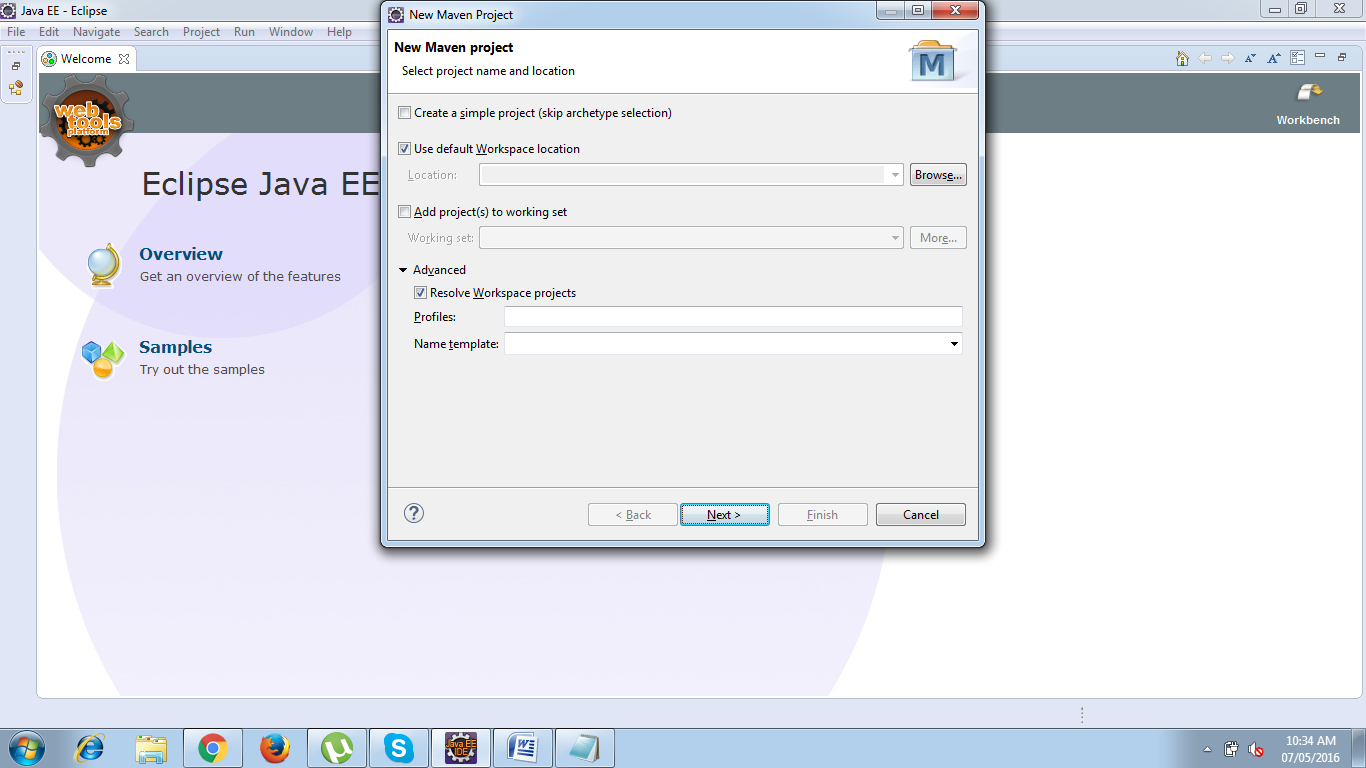
**Step 1.** Create a New Maven Project

Click 'File' -> 'New' -> 'Other' -> 'Maven Project' and then click 'Next'.

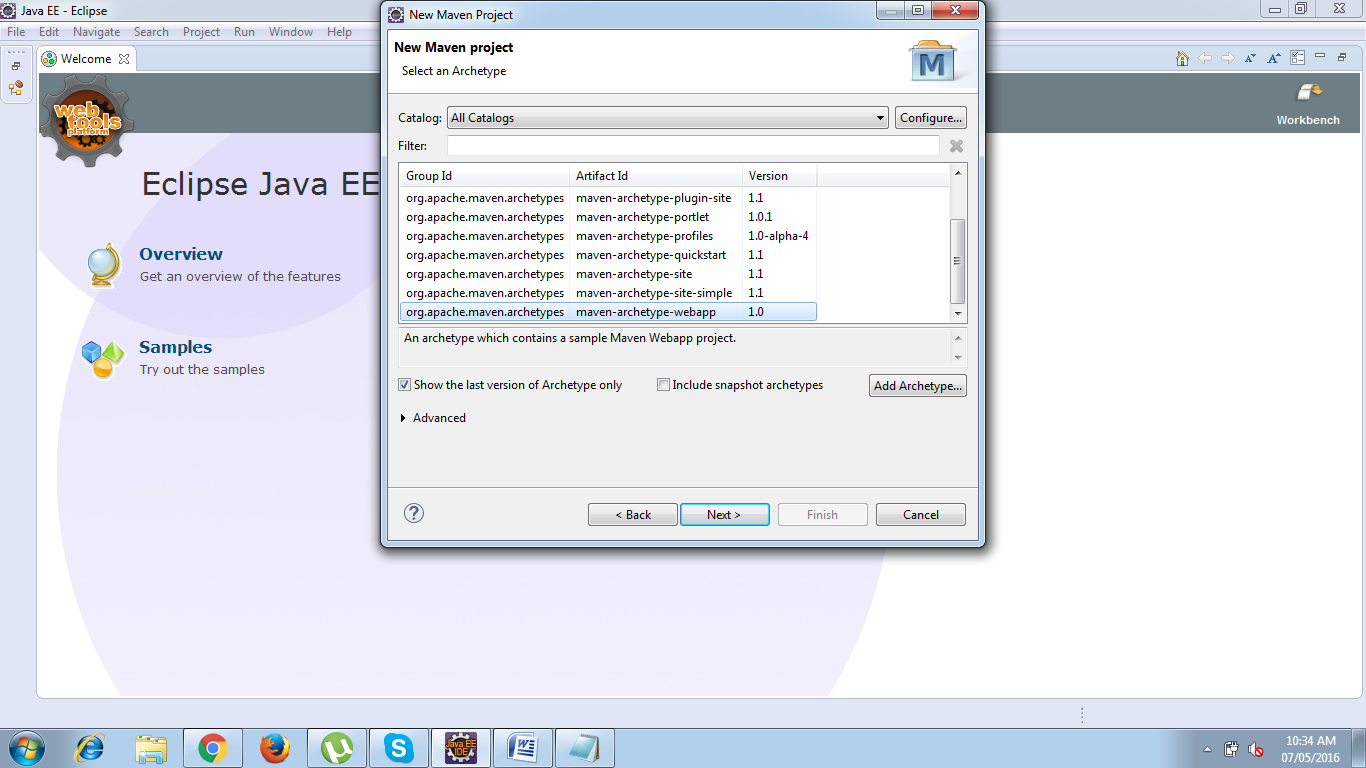


**Step 2.** Select project name and location.

Use the default Workspace location or specify the location if necessary.



##### **Step 3.** Select an archetype



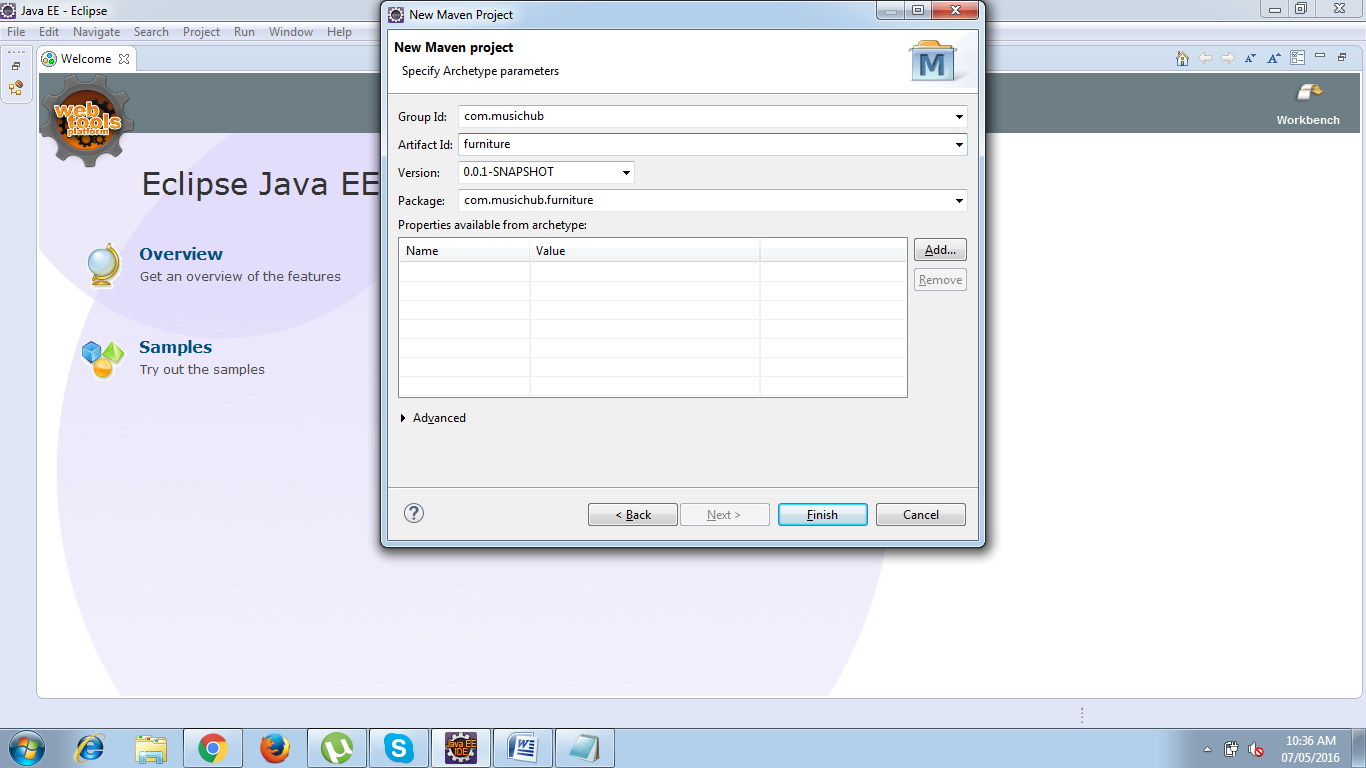
**Step 4.** Enter the Group id and Artifact Id

Enter the Group Id and Artifact Id and click 'Finish'

groupId will identify your project uniquely across all projects, so we need to enforce a naming schema. It has to follow the package name rules, what means that has to be at least as a domain name you control

artifactId is the name of the jar without version. If you created it then you can choose whatever name you want with lowercase letters and no strange symbols.

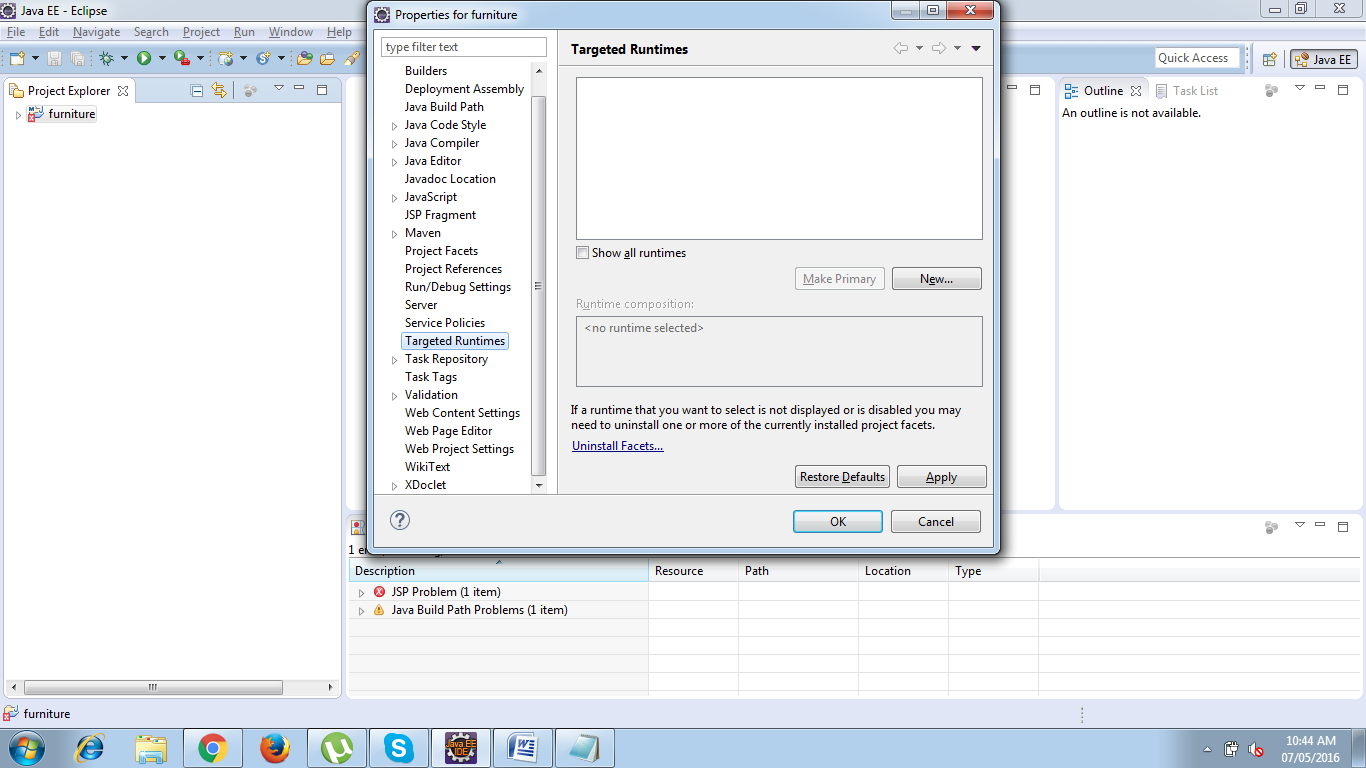
version if you distribute it then you can choose any typical version with numbers and dots (1.0, 1.1, 1.0.1, ...). Don't use dates as they are usually associated with SNAPSHOT (nightly) builds. If it's a third party artifact, you have to use their version number whatever it is, and as strange as it can look.



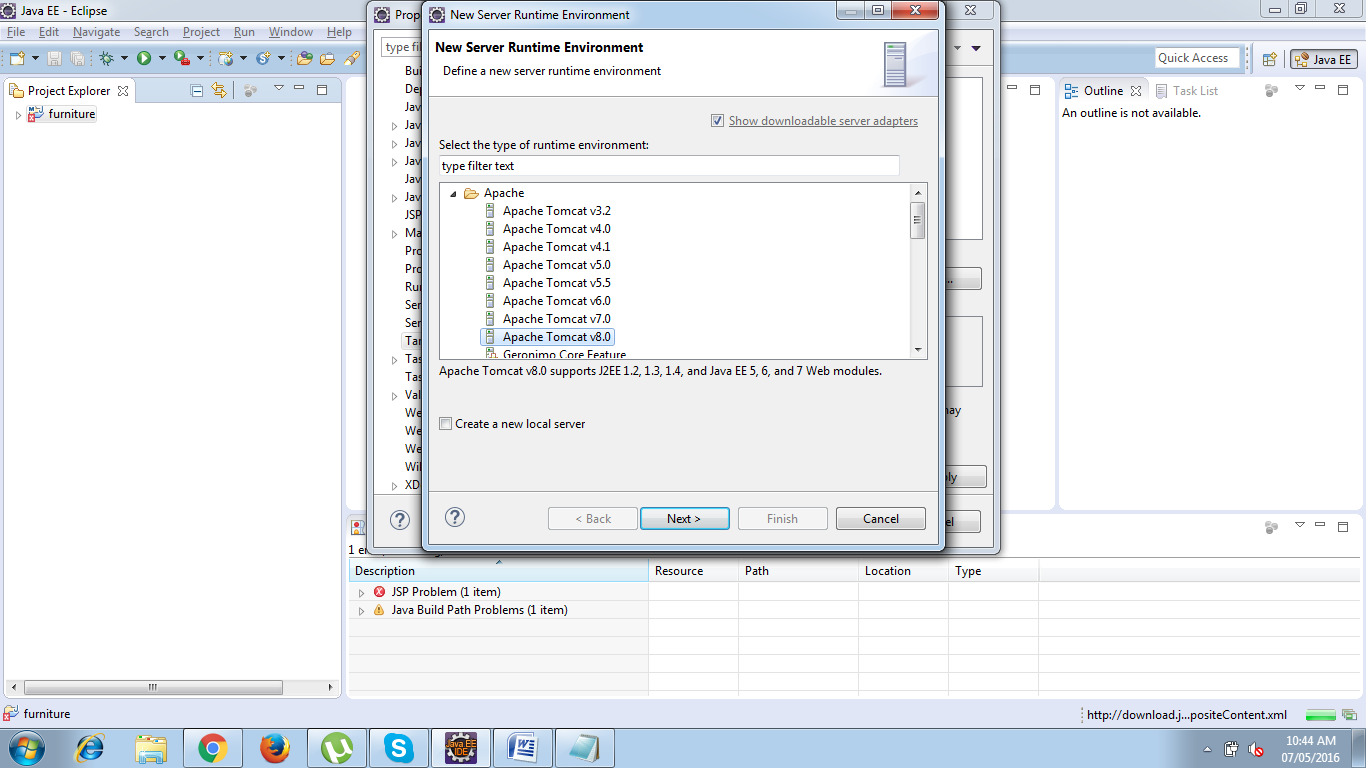
Your project will be ready but with an error due to runtime is not configured for the newly created Workspace.

Updating Target Runtime Server’s :

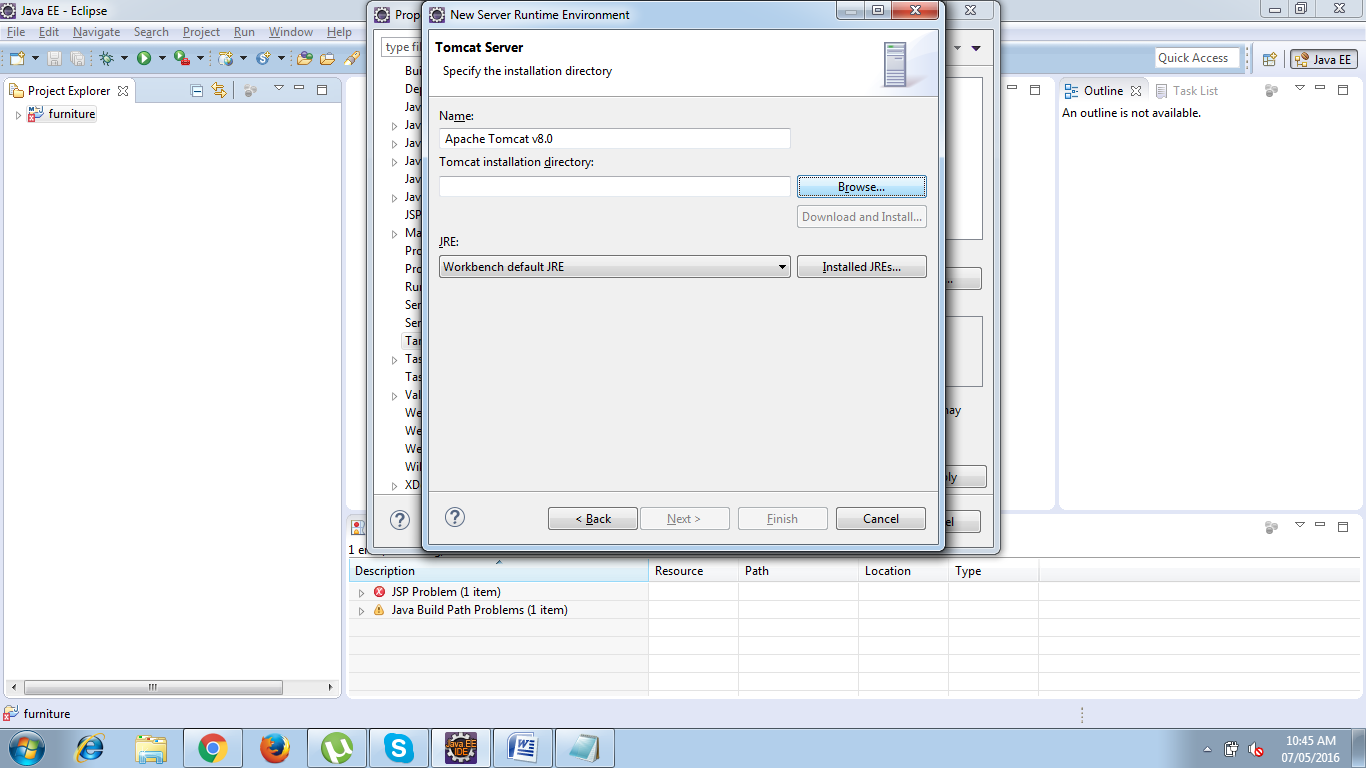
Right Click Project root folder and choose properties :

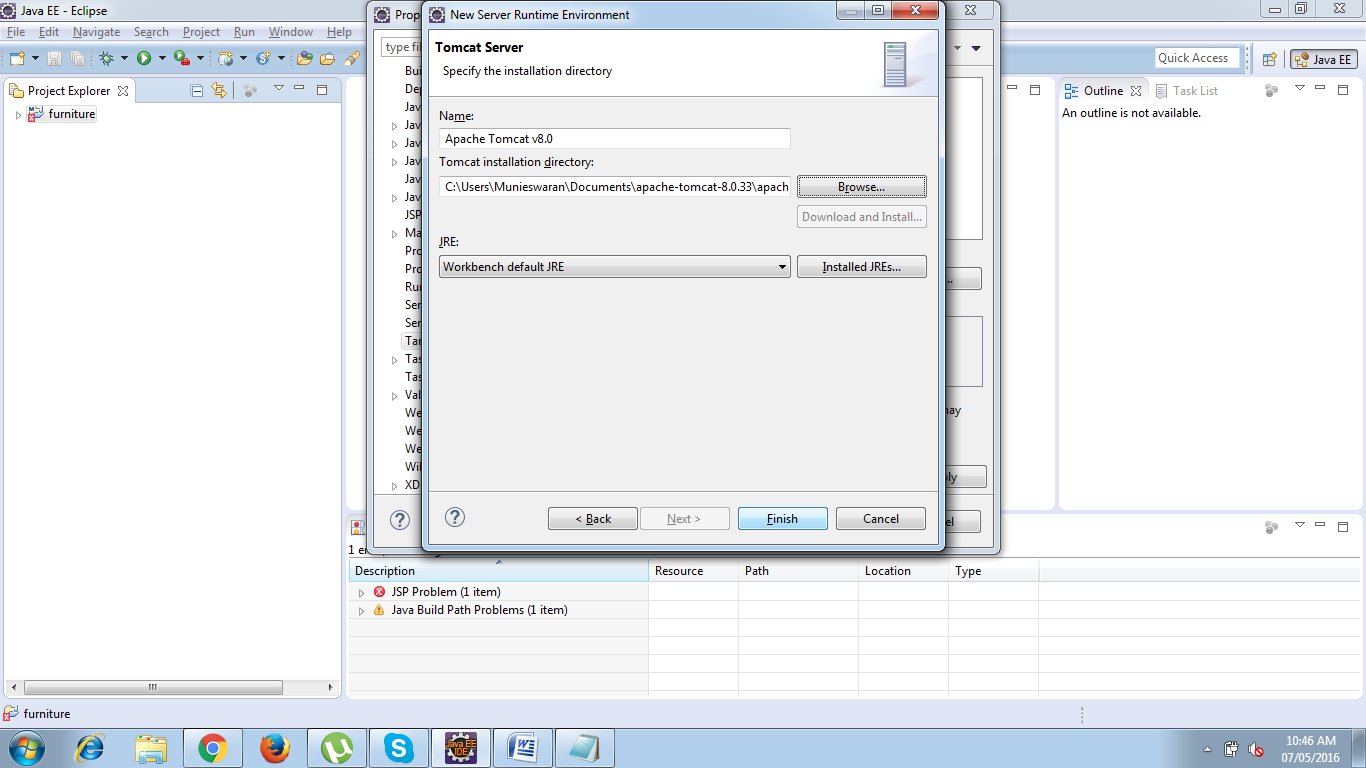


Choose Targeted Runtime which will open New Server Runtime Environment window :



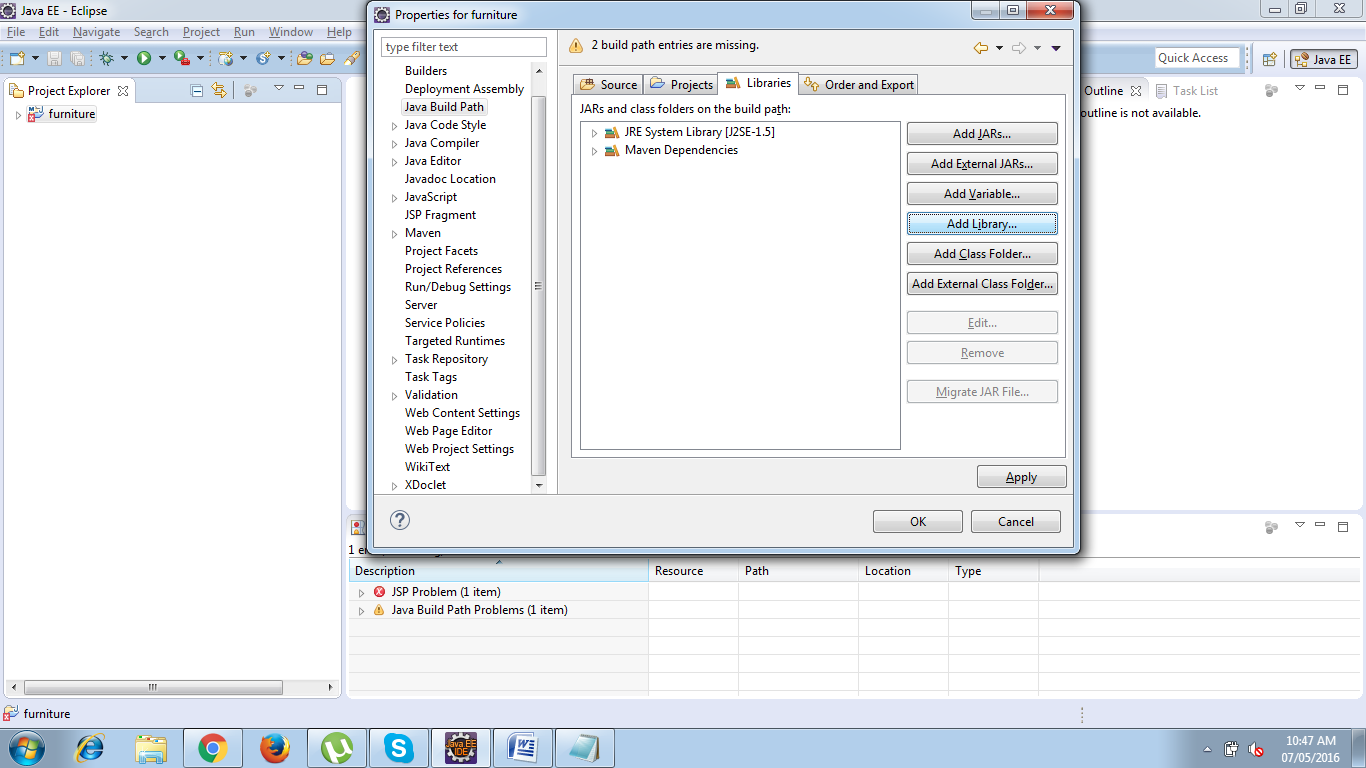
Select Apache Tomcat v8.0 and Click Next -> Choose Tom cat Installation Directory



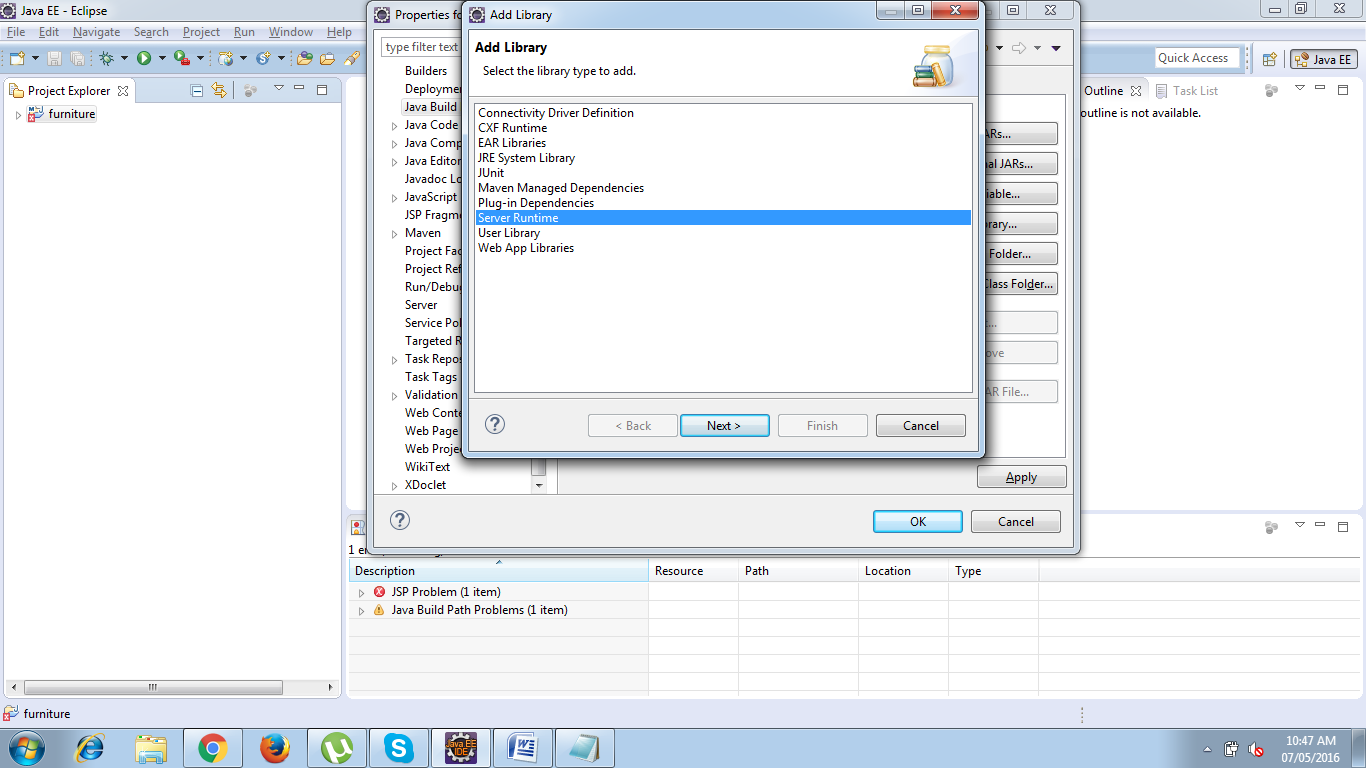


**Click Finish**

Adding build path Right click project and choose properties -> Select Java Build path -> Choose Libraries -> Select Add libraries .

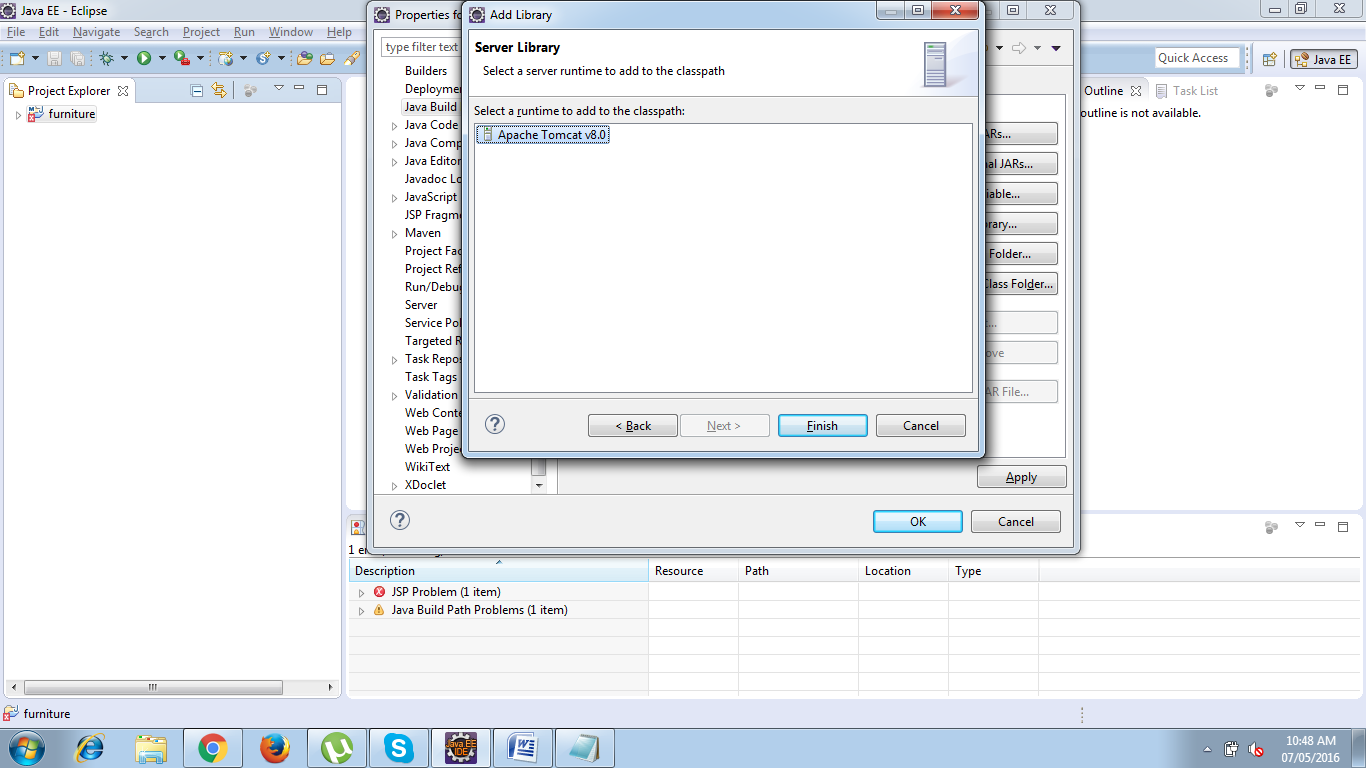


Choose Server Runtime -> Click Next

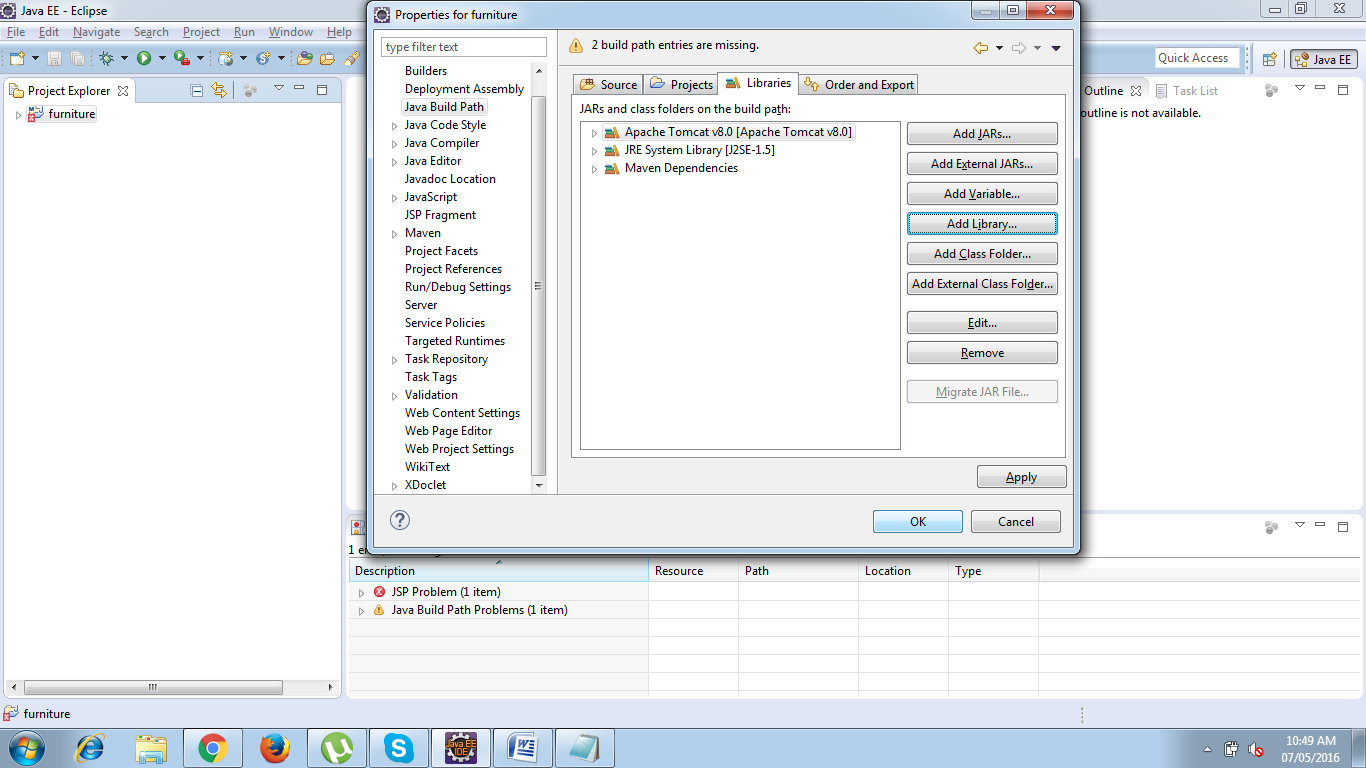


Select a runtime to add to the class path ->Apache Tomcat V8.0

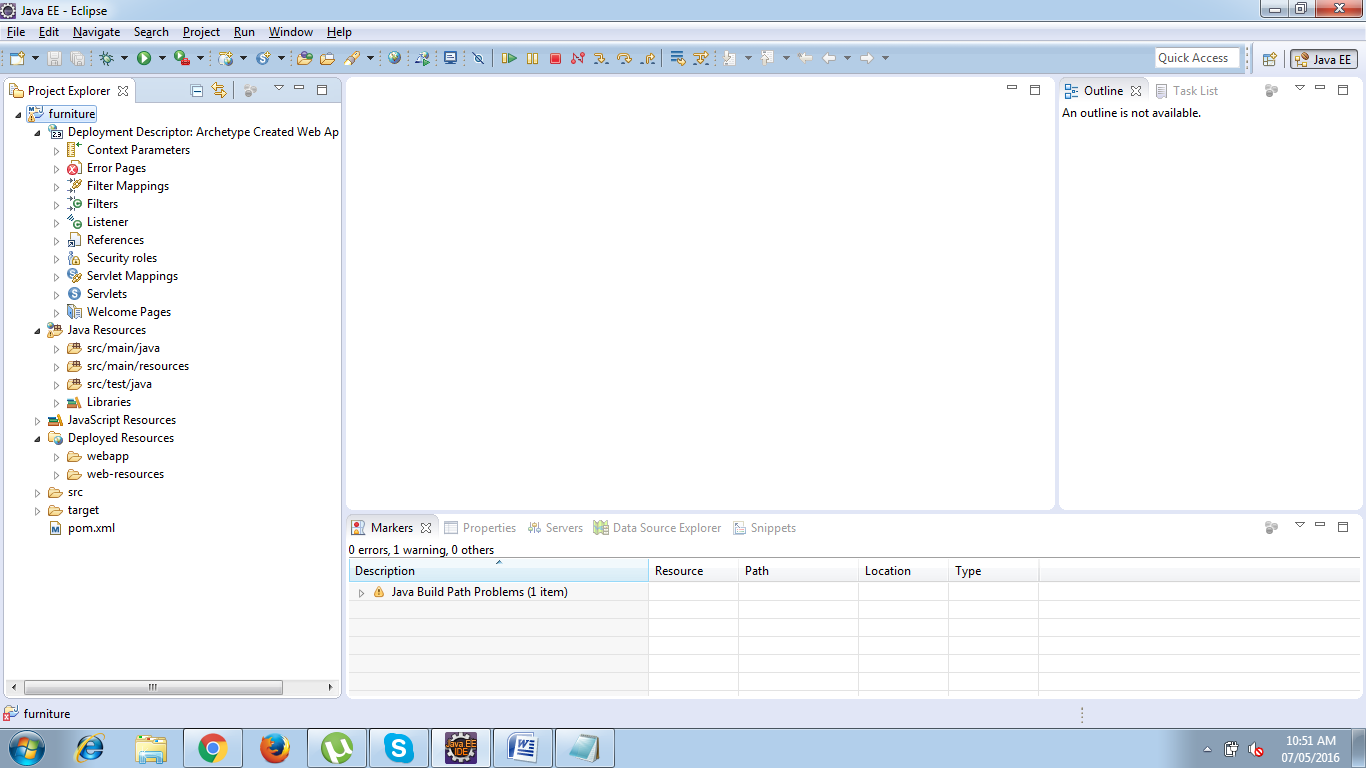
And click finish.



Click OK in the properties window.

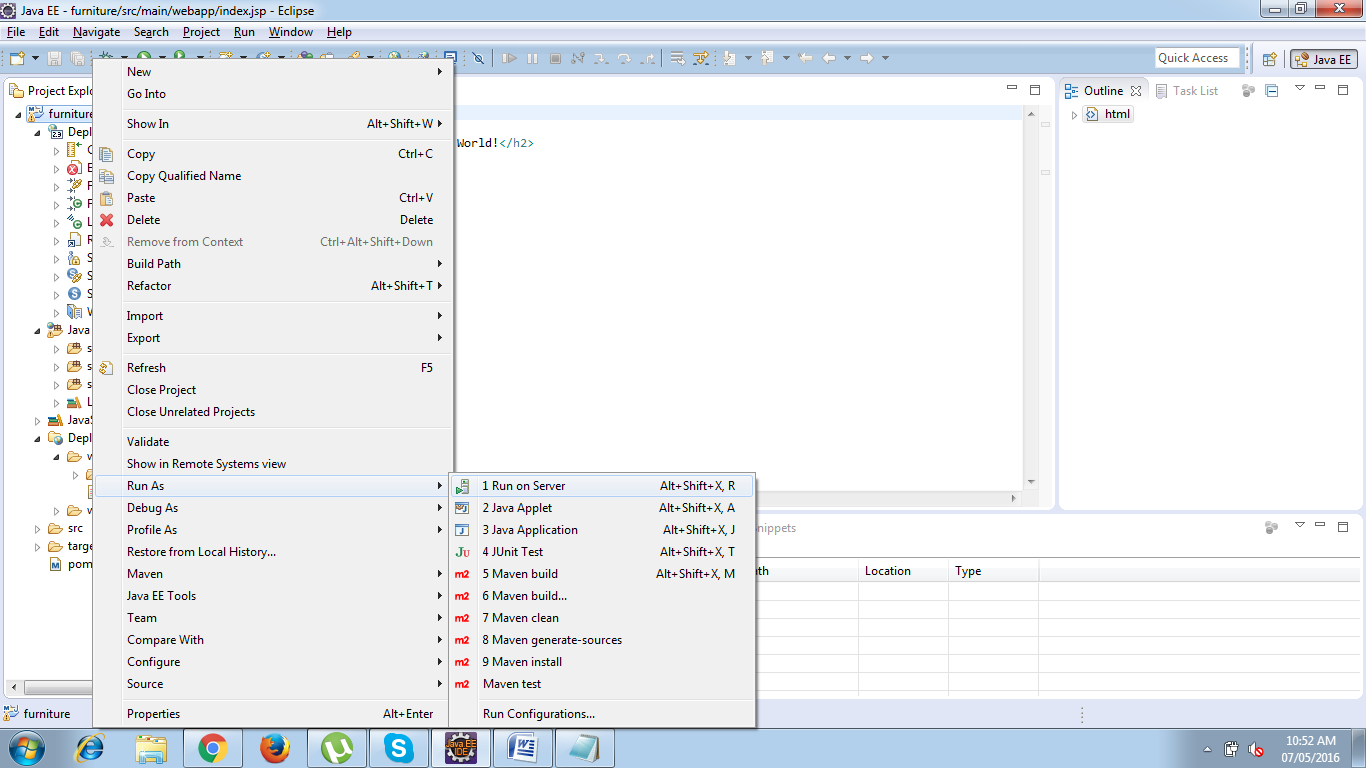


Finally your project is now ready to develop.

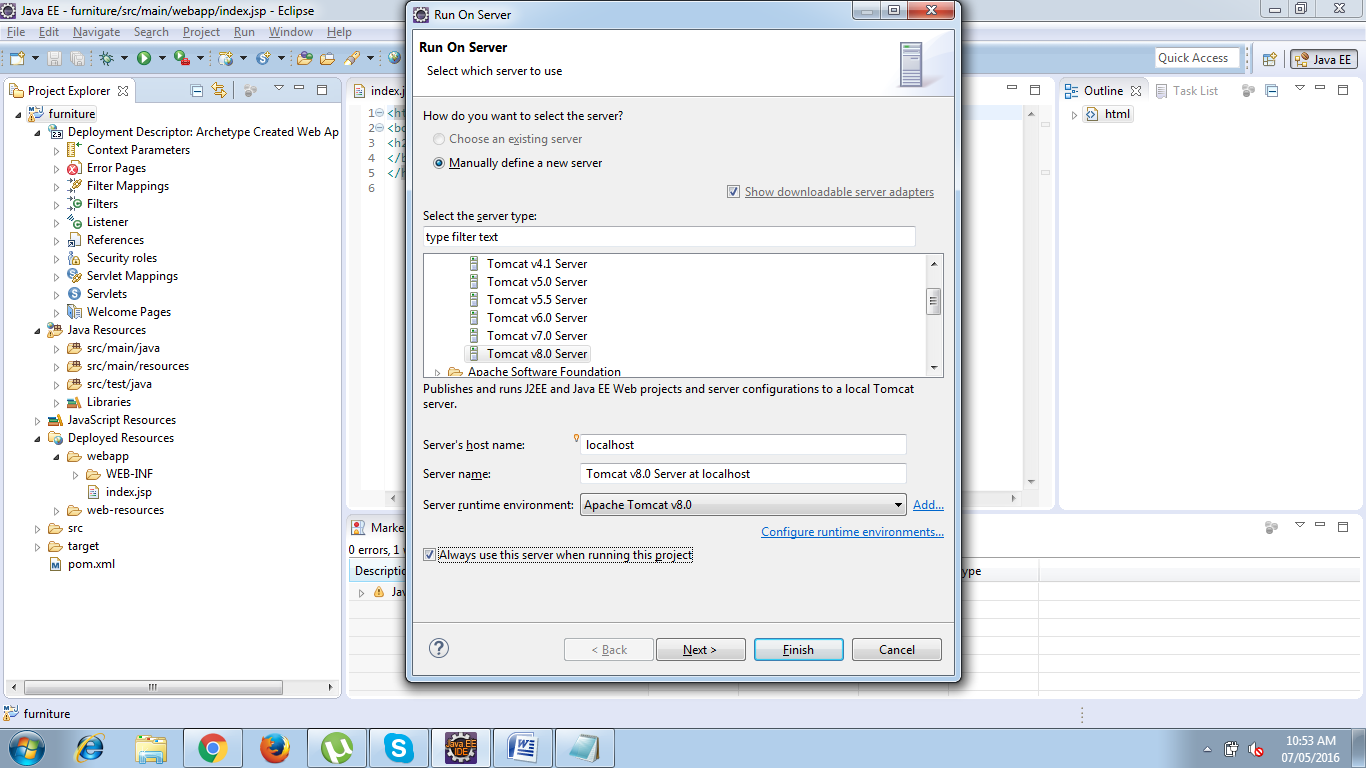


**Testing the Environment**

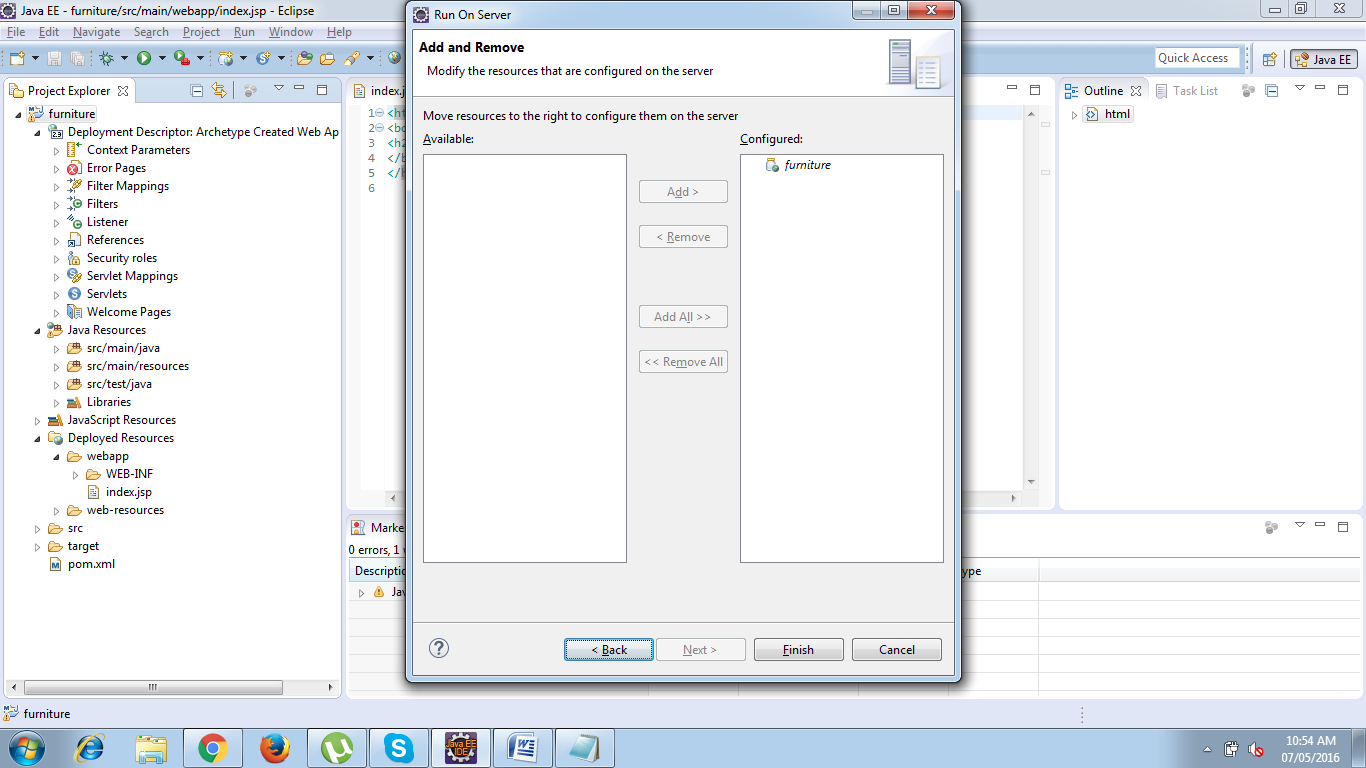
Right Click the project and choose RUN AS -> Run on Server option

****

Run on Server window will be now opened , Select the Server to use and select always use this server when running the project check box.



Click Next -> Move the resources to be configured and click finish



Now your server will start up and your project will be deployed into Tomcat Server and you will see the index page on your default embedded browser.

## Adding Dependencies :

The following are the dependencies which I used in my project

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.todo</groupId>

<artifactId>todo</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>smiley Maven Webapp</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.7.5</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.5.Final</version>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-dbcp2</artifactId>

<version>2.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.5</version>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>1.4.192</version>

</dependency>

<dependency>

<groupId>commons-io</groupId>

<artifactId>commons-io</artifactId>

<version>2.3</version>

</dependency>

</dependencies>

<build>

<finalName>todo</finalName>

</build>

</project>

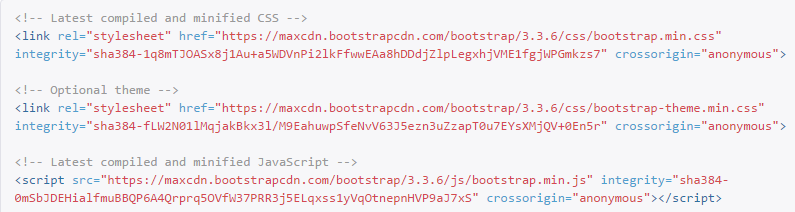
**Add a bootstrap reference to the web page**

<script type="text/javascript" src="webjars/jquery/2.1.1/jquery.min.js"></script>

<script type="text/javascript" src="webjars/bootstrap/3.2.0/js/bootstrap.min.js"></script>

**Bootstrap CDN**

BootstrapCDN is a free and public content delivery network. Users of BootstrapCDN can load CSS, JavaScript and images remotely, from its servers.



# MAVEN MVC PATTERN

## Implementing the VIEW COMPONENT

* Move your html to /WEB-Content/index.jsp
* In your main source java file create a package com.todo.configuration in that add three java classes namely, ApplicationConfiguration.java,TodoConfiguration.java,Todo  
  Intializer.java
* make a request to localhost:8080/todo/

**ApplicationConfiguration.java**

package com.Todo.configuration;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.orm.hibernate4.HibernateTransactionManager;

import org.springframework.orm.hibernate4.LocalSessionFactoryBuilder;

import org.springframework.transaction.annotation.EnableTransactionManagement;

import com.Todo.model.Todo;

@Configuration

@ComponentScan("com")

@EnableTransactionManagement

public class ApplicationConfiguration {

@Autowired

@Bean(name="dataSource")

public DataSource getDataSource()

{

System.out.println("get datasource method called");

DriverManagerDataSource dataSource=new DriverManagerDataSource();

dataSource.setDriverClassName("org.h2.Driver");

dataSource.setUrl("jdbc:h2:tcp://localhost/~/krish007");

dataSource.setUsername("sa");

dataSource.setPassword("");

return dataSource;

}

private Properties getHibernateProperties()

{

Properties properties=new Properties();

properties.setProperty("hibernate.show\_sql", "true");

properties.setProperty("hibernate.dialect", "org.hibernate.dialect.H2Dialect");

properties.setProperty("hibernate.hbm2ddl.auto", "update");

return properties;

}

@Autowired

@Bean(name="sessionFactory")

public SessionFactory getSessionFactory(DataSource dataSource)

{

LocalSessionFactoryBuilder localSessionFactoryBuilder=new LocalSessionFactoryBuilder(dataSource);

localSessionFactoryBuilder.addProperties(getHibernateProperties());

localSessionFactoryBuilder.addAnnotatedClass(Todo.class);

return localSessionFactoryBuilder.buildSessionFactory();

}

@Autowired

@Bean(name="transactionManager")

public HibernateTransactionManager getTransactionManager(SessionFactory sessionFactory)

{

HibernateTransactionManager hibernateTransactionManager=new HibernateTransactionManager(sessionFactory);

return hibernateTransactionManager;

}

}

**TodoConfiguration.java**

package com.Todo.configuration;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages="com.todo.configuration")

public class TodoConfiguration {

}

**TodoIntializer.java**

**package** com.Todo.configuration;

**import** org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

**public** **class** TodoInitializer **extends** AbstractAnnotationConfigDispatcherServletInitializer {

**protected** Class<?>[] getRootConfigClasses() {

// **TODO** Auto-generated method stub

**return** **new** Class[]{TodoConfiguration.**class**,ApplicationConfiguration.**class**};

}

**protected** Class<?>[] getServletConfigClasses() {

// **TODO** Auto-generated method stub

**return** **null**;

}

**protected** String[] getServletMappings() {

// **TODO** Auto-generated method stub

**return** **new** String[]{"/"};

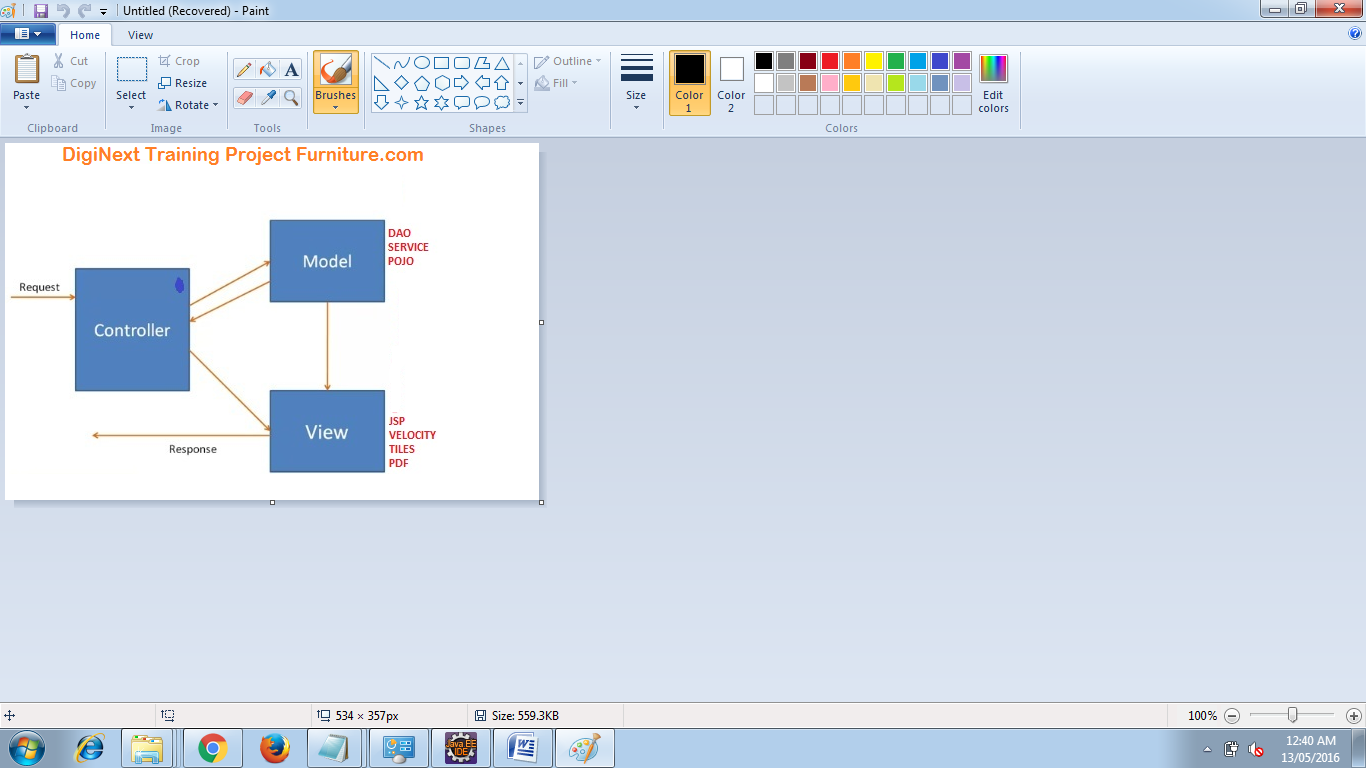
}

}

Adding the Front Controller and SPRING MVC Framework :

Model view controller is a software architecture design pattern. It provides solution to layer an application by separating three concerns business, presentation and control flow.

* The Model can be some DAO layer or some Service Layers which give some information about request or requested information or Model can be a POJO which encapsulates the application data given by the controller.
* The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The Controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

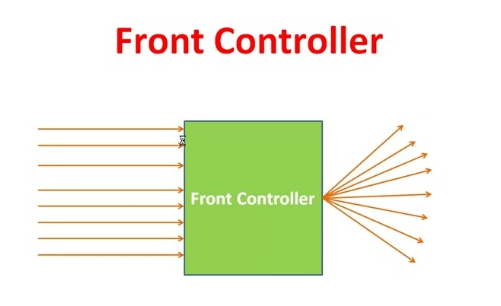


**Advantages of Spring MVC Framework**

* Supports RESTful URLs.
* Annotation based configuration(i.e. you may reduce the metadata file or less of configuration).
* Supports to plug with other MVC frameworks like Struts, Struts2, WebWorks etc.
* Flexible in supporting different view types like JSP, velocity, XML, PDF, Tiles etc.

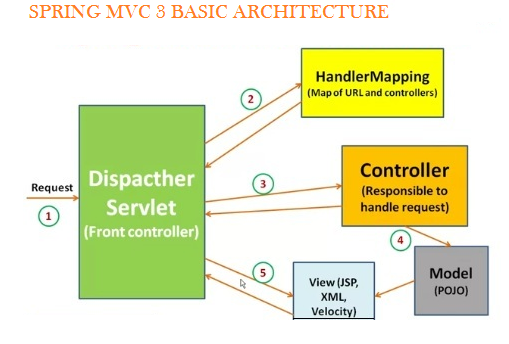
**Front Controller:** Front Controller is very important component one which route the all the requests into framework control that means when ever requests land on different controllers it queues that request to the controller of framework without this MVC framework will not may be able to take control of the request at landing at the application. So front controller is not only capture the request but also the following responsibility-

* It initialize the framework to cater to the requests.
* Load the map of all URLs and the components responsible to handle the request.
* Prepare the map for the views.



## Spring 3 MVC Basic Architecture :

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.



In Spring 3 MVC framework Dispatcher Servlet access Front Controller which handles all coming requests and queses for forward to the different controller.

1. Whenever request lands the dispatcher servlet consult with HandlerMapping

(HandlerMapping- is a component which have the map of URL and Controller which need to be invoked for that particular request which lands with URL)

2. then Dispatcher servlet has information about which is controller need to be invoked

3. then that controller will be invoked

4. and Controller can request the model for some information (about some DAO, Service layer or Data in POJO, or data in database using business logic)

5. once process has been done then dispatcher servlet get the response then dispatcher servlet will get view resolver to build the view and view resolver look out what view has being configured it has been JSP, Velocity, XML etc. based this configuratin view has been prepared and the information from model i.e. POJO it will be put on the view and response will be send back to browser.

**@Controller :**

* Used at the class level
* Tells the spring framework that the marked class acts as a controller.

@Controller

public class EmployeeController{

}

**@RequestMapping :**

* Can be used at the class level and method level in controllers.
* Arguments
* URL[]
* HTTP Methods[]-GET, POST, DELETE, TRACE, OPTIONS, HEAD, PUTS. Defaults method supported is GET
* params[]-used to check if a request parameter matches with a value and only if the conditions passes the method or controller processes the request. (eg. @RequestMapping params="myName=guest" )
* headers[]-used to check if a request header matches with a value and only if the condition passes the method or controller processes the request (eg. @RequestMapping headers="myheader=guestHadder")

# Defining a Controller

. The @Controller annotation indicates that a particular class serves the role of a controller. The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

@Controller

package com.Todo.controller;

import java.util.List;

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

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

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

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

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

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

import com.Todo.dao.TodoDao;

import com.Todo.model.Todo;

@RestController

public class TodoController {

@Autowired

TodoDao todoDao;

@RequestMapping(value="/addTodo",headers="Accept=application/json",method=RequestMethod.POST)

public void addTodo(@RequestBody Todo todo)

{

todoDao.addTodo(todo);

}

@RequestMapping(value="/viewAllTodo",headers="Accept=application/json",method=RequestMethod.GET)

public List<Todo> viewTodo()

{

return todoDao.viewTodo();

}

@RequestMapping(value="/deleteTodo/{id}",headers="Accept=application/json",method=RequestMethod.DELETE)

public void deleteTodo(@PathVariable int id)

{

todoDao.deleteTodo(id);

}

@RequestMapping(value="/updateTodo",headers="Accept=application/json",method=RequestMethod.PUT)

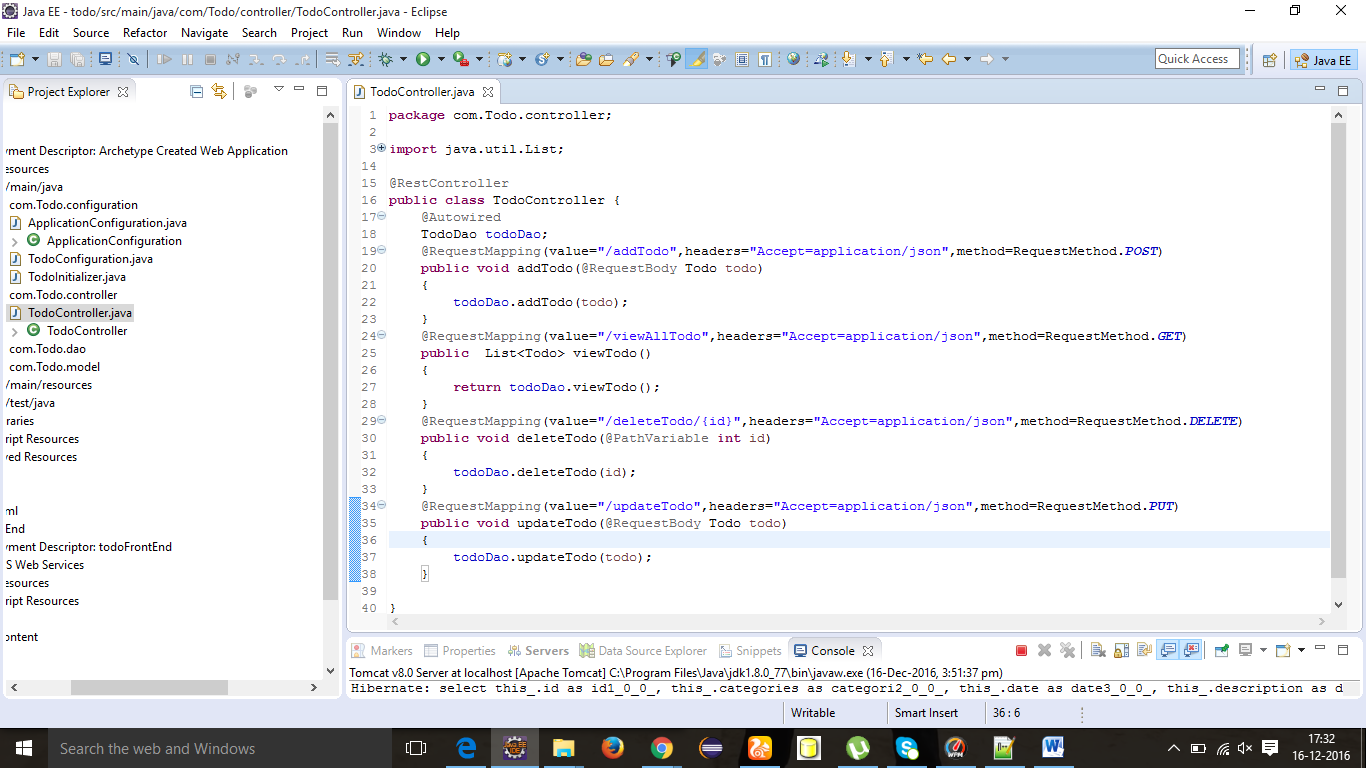
public void updateTodo(@RequestBody Todo todo)

{

todoDao.updateTodo(todo);

}

}

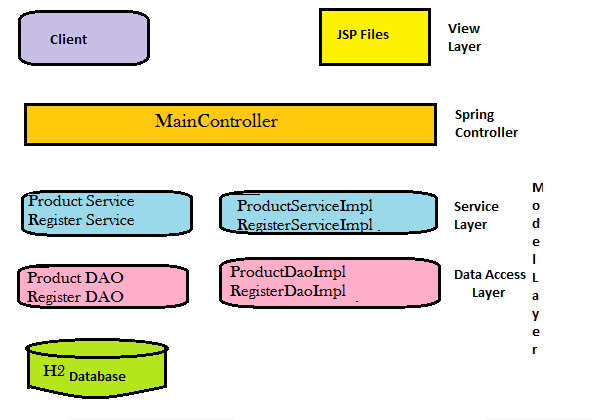


Spring MVC Form Model Implementation

Create a Java class Product and under the com.todo.model

Model Objects

* are very common, and are used in almost all applications
* are often central to an application, since they usually model problem domain objects
* often map roughly to the records of a corresponding database table
* are often used as return values for Data Access Object methods
* can be used to implement the Model in a Model-View-Controller pattern

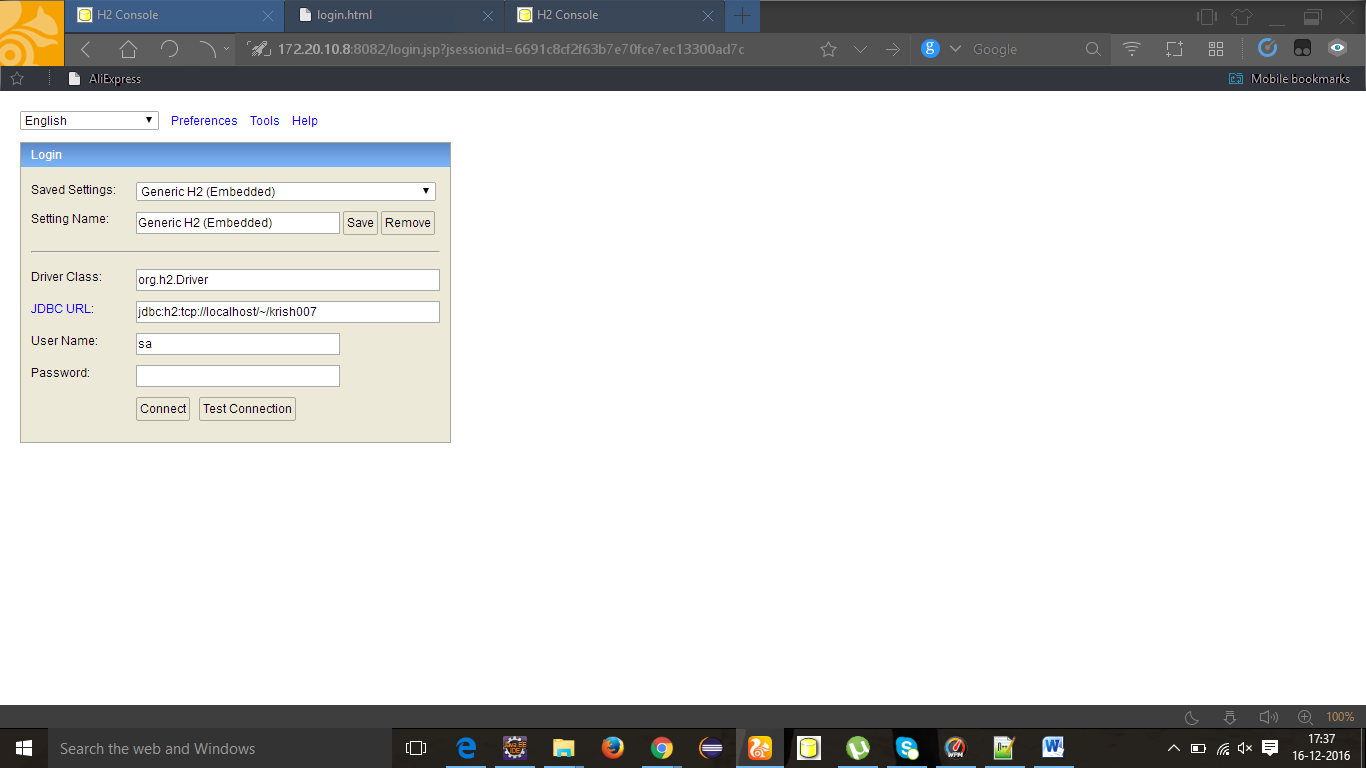
CRUDOperationsSpring3Hibernate Integration

**Files used**

* The Spring MVC Controller
* Client side validation is done by using Angular-JS
* The html page and JavaScript files used to perform UI creation
* And the last Spring MVC configuration file

Step1: Ensure the H2 RDBMS is running on the back ground.

Step2: Ensure all database properties are correct which can be retrieved from the home screen of H2 DB. Make sure you are logined into Server and not Client



Step 3 : Add Annotation to the Model classes Annotations is the powerful way to provide the metadata for the Object and Relational Table mapping. All the metadata is clubbed into the POJO java file along with the code this helps the user to understand the table structure and POJO simultaneously during the development.

**@Entity Annotation**

The EJB 3 standard annotations are contained in the javax.persistence package, so we import this package as the first step. Second we used the @Entity annotation to the Employee class which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

**@Id and @GeneratedValue Annotations**

Each entity bean will have a primary key, which you annotate on the class with the @Id annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.

By default, the @Id annotation will automatically determine the most appropriate primary key generation strategy to be used but you can override this by applying the @GeneratedValue annotation which takes two parameters strategy and generator which I'm not going to discuss here, so let us use only default the default key generation strategy. Letting Hibernate determine which generator type to use makes your code portable between different databases.

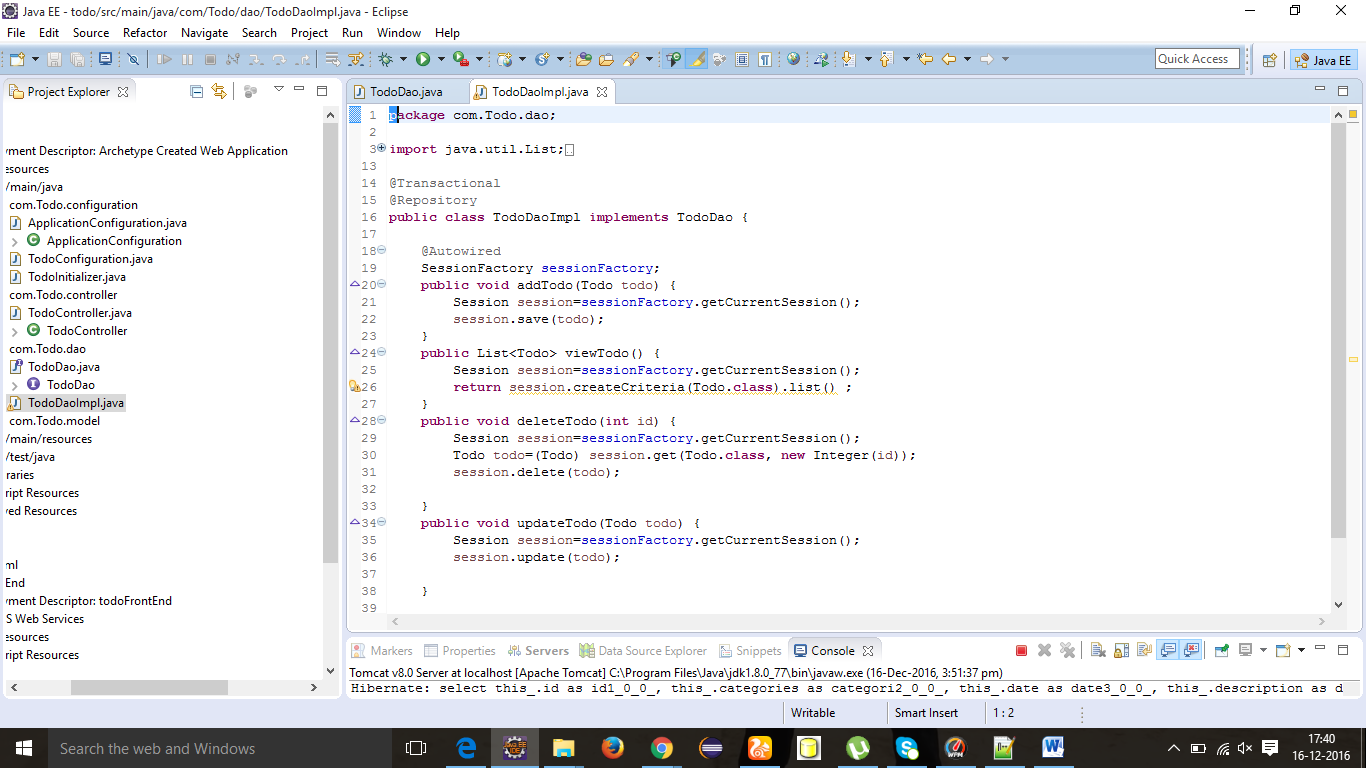
## Create DAO Class

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate, JPA in a consistent way. This allows one to switch between the aforementioned persistence technologies fairly easily and it also allows one to code without worrying about catching exceptions that are specific to each technology.

@Repository – Indicates DAO component in the persistence layer

@Controller – Indicates a controller component in the presentation layer.

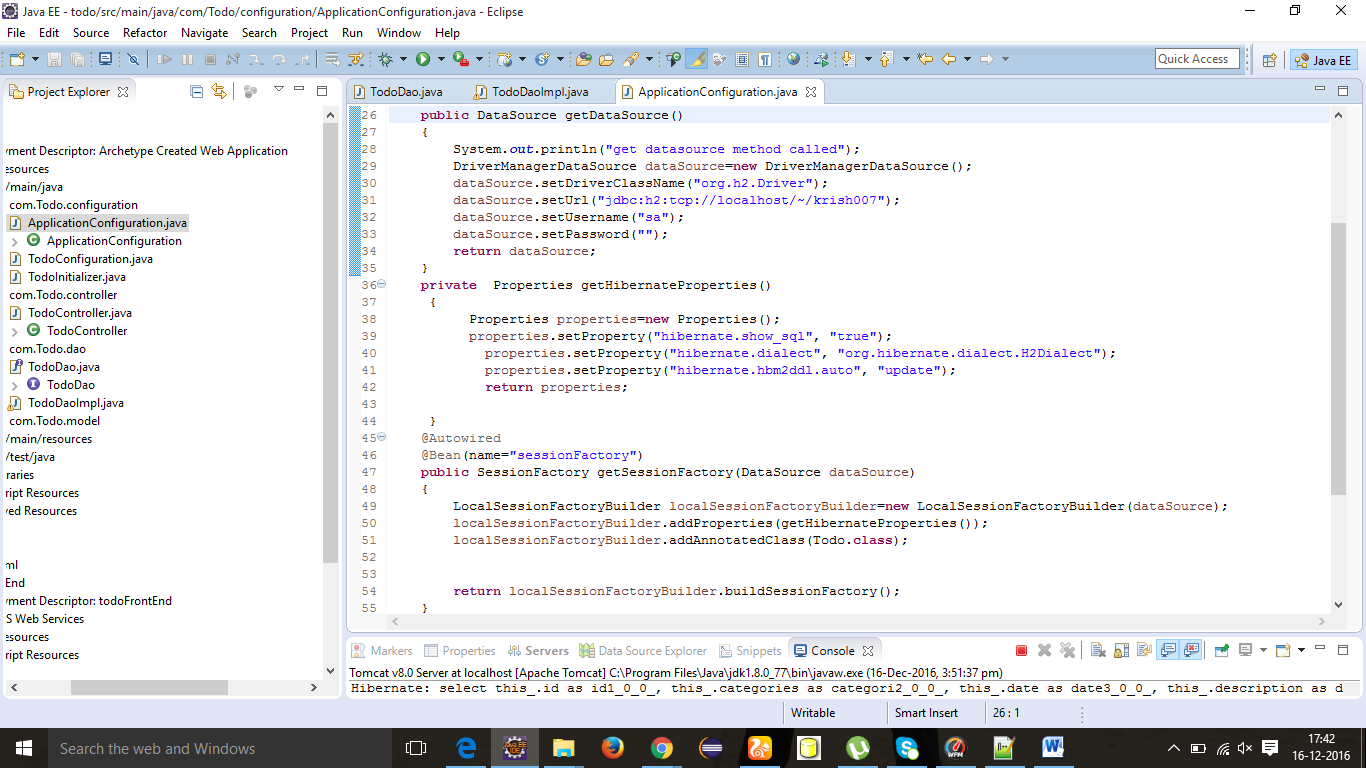
@Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished. The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.



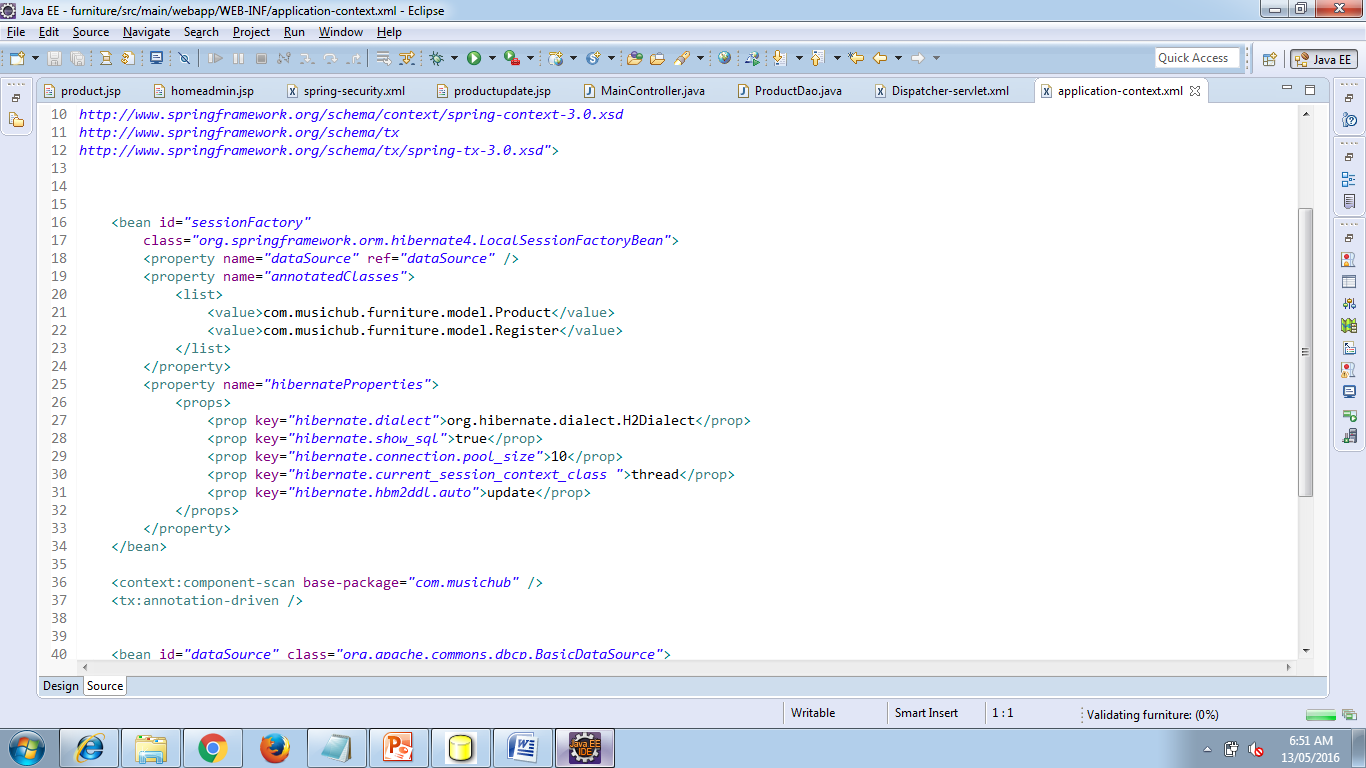
SessionFactory setup in a Spring container

We can directly obtain a Hibernate session using the getSession() method of the Hibernate SessionFactory. Wire a LocalSessionFactoryBean into the DAO as a SessionFactory object and then use the getSession method of this object to obtain the session.

To avoid tying application objects to hard-coded resource lookups, you can define resources such as a JDBC DataSource or a Hibernate SessionFactory as beans in the Spring container. Application objects that need to access resources receive references to such predefined instances through bean references.



The above DAO follows the dependency injection pattern: it fits nicely into a Spring IoC container, just as it would if coded against Spring’s HibernateTemplate. Of course, such a DAO can also be set up in plain Java (for example, in unit tests). Simply instantiate it and call setSessionFactory(..) with the desired factory reference. As a Spring bean definition, the DAO would resemble the following:



The main advantage of this DAO style is that it depends on Hibernate API only; no import of any Spring class is required. This is of course appealing from a non-invasiveness perspective, and will no doubt feel more natural to Hibernate developers.

## Declarative transaction demarcation

Spring’s declarative transaction support, which enables you to replace explicit transaction demarcation API calls in your Java code with an AOP transaction interceptor. This transaction interceptor can be configured in a Spring container using either Java annotations or XML.This declarative transaction capability allows you to keep business services free of repetitive transaction demarcation code and to focus on adding business logic, which is the real value of your application.

<bean id=*"transactionManager"*

class=*"org.springframework.orm.hibernate4.HibernateTransactionManager"*>

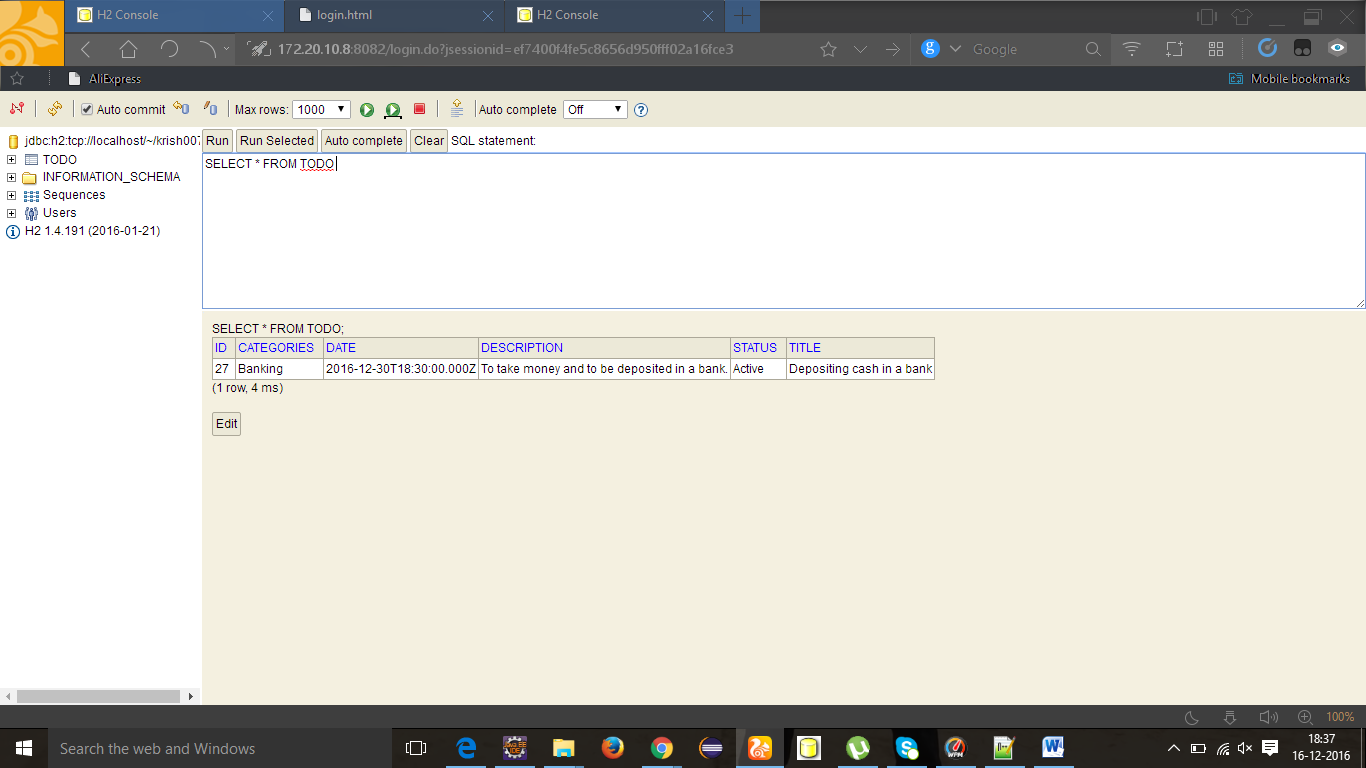
<property name=*"sessionFactory"* ref=*"sessionFactory"* />

</bean>

Now we have to annotate the service layer with @Transactional annotations and instruct the Spring container to find these annotations and provide transactional semantics for these annotated methods.

**Database**

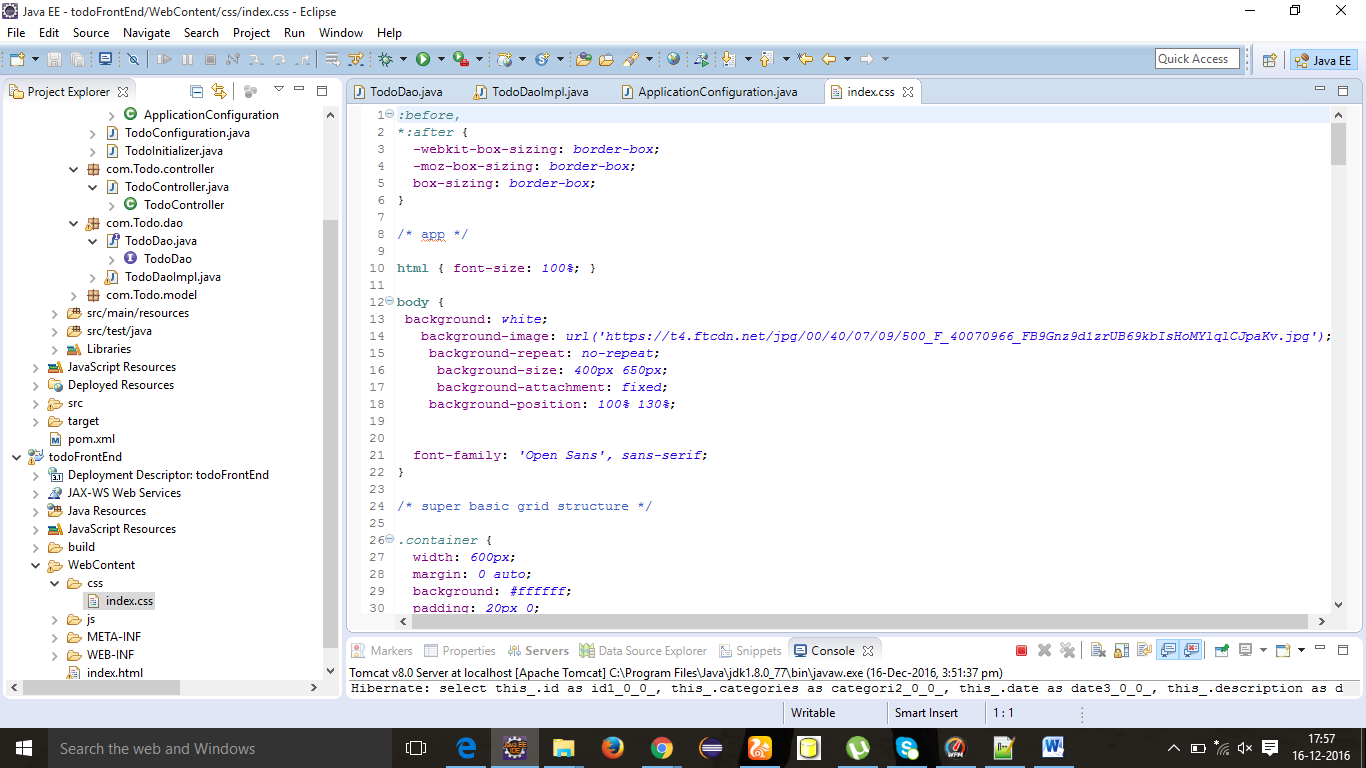
I used H2 database in that I created one table named TODO



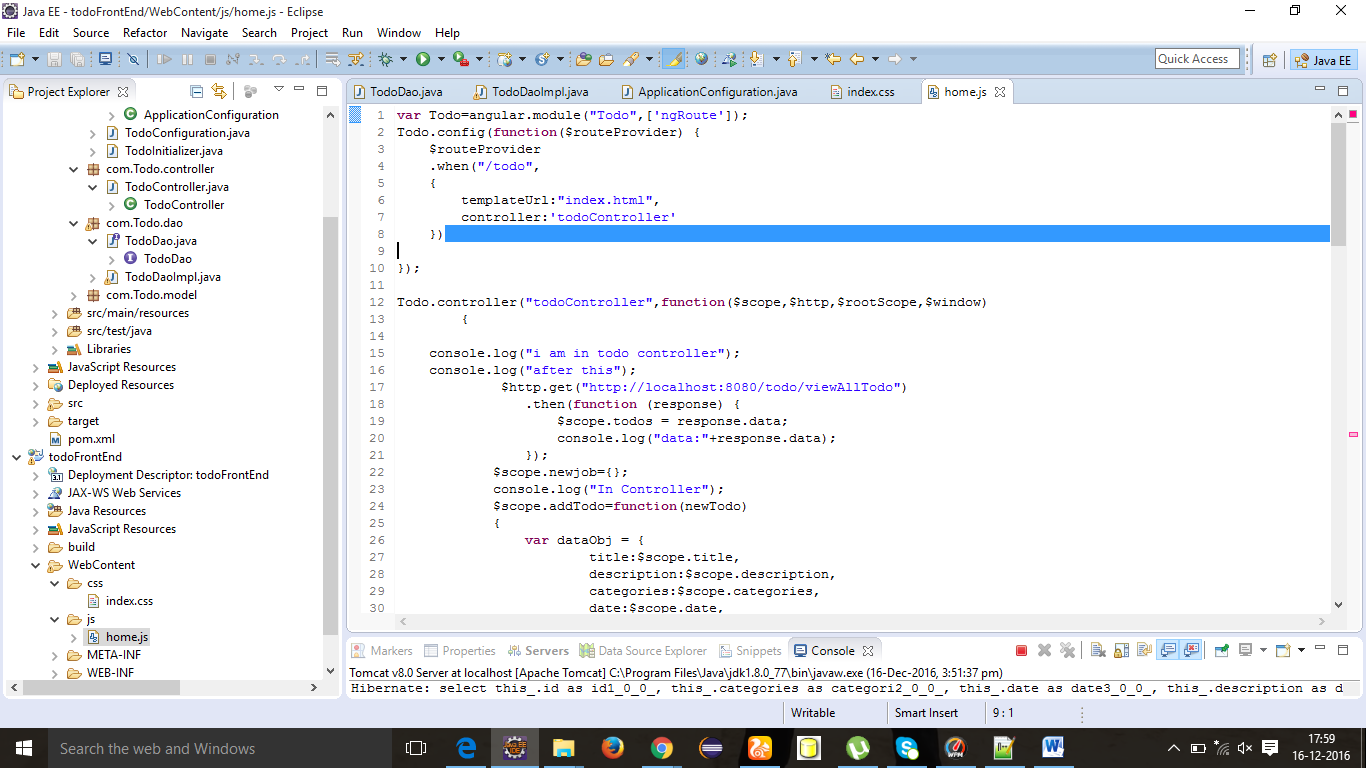
**Designing Front End with Dynamic Web Project**

In that I have used CSS Folder and JS Folder and there is also a index.html Page. In CSS folder there is a page called index.css in which I had written the following code:

**CSS:**



**JS:**

****

**Index.html:**

<!DOCTYPE html>

<html ng-app=*"Todo"* >

<head>

<title>Todo List</title>

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

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

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"*></script>

<script src=*"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"*></script>

<link rel=*"stylesheet"* href=*"//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css"* />

<link rel=*"stylesheet"* href=*"//netdna.bootstrapcdn.com/font-awesome/4.0.0/css/font-awesome.css"* />

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"*></script>

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

<script src=*"http://ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular.min.js"*></script>

<script src=*"//ajax.googleapis.com/ajax/libs/angularjs/1.2.25/angular-route.js"*></script>

<script src=*"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"*></script>

<script src=*"js/home.js"*></script>

<link href=*"css/index.css"* type=*"text/css"* rel=*"Stylesheet"* />

<style>

**input***.ng-invalid.ng-dirty*{border:*0.5px solid red*;}

</style>

</head>

<body ng-controller=*"todoController"*>

<div style="margin-top:*150px*;">

<div class=*"container"*>

<div class=*"content"* >

<center><img src=*"http://vignette2.wikia.nocookie.net/gtawiki/images/e/ee/GTAWiki-Todo-Checklist.png/revision/latest?cb=20151027125135"* class=*"img-responsive margin"* style="width:*10%*" ></center><h1> To-Do List</h1><br><br>

<form class=*"form-todo"* name=*"todo"*>

<div class=*"inputContainer"*>

<input type=*"text"* class=*"form-control"* name=*"title"* id=*"title"* placeholder=*"Enter Title"* required=*""* ng-model=*"title"*>

<center><label for=*"title"*>Title</label></center>

</div>

<span ng-show=*"todo.title.$dirty && todo.title.$invalid"*><center><font color=*"red"*>\*The title is required.</font></center></span>

<div class=*"inputContainer"*>

<input type=*"text"* id=*"description"* name=*"description"* class=*"form-control"* placeholder=*"Enter Description"* required=*""* ng-model=*"description"* ng-minlength=*"8"* ng-maxlength=*"50"* >

<center><label for=*"description"*>Description</label></center>

</div>

<span ng-show=*"todo.description.$dirty && todo.description.$error.minlength"*><center><font color=*"red"*>\*Too Short!</font></center></span>

<span ng-show=*"todo.description.$dirty && todo.description.$error.maxlength"*><center><font color=*"red"*>\*Too Long!</font></center></span>

<span ng-show=*"todo.description.$dirty && todo.description.$invalid"*><center><font color=*"red"*>\*The description is required.</font></center></span>

<div class=*"inputContainer half last"*>

<div class=*"inputContainer"*>

<input type=*"text"* id=*"categories"* name=*"categories"* class=*"form-control"* placeholder=*"Enter categories"* required=*""* ng-model=*"categories"*>

<center><label for=*"categories"*>categories</label></center>

</div>

<span ng-show=*"todo.categories.$dirty && todoForm.categories.$invalid"*><center> <font color=*"red"*>\*The categories is required.</font></center></span>

</div>

<div class=*"inputContainer half last right"*>

<input type=*"date"* id=*"date"* name=*"date"* class=*"taskDate"* placeholder=*"Enter Dead Line"* required=*""* ng-model=*"date"*>

<label for=*"date"*>Task Date</label>

</div>

<span ng-show=*"todo.date.$dirty && todo.date.$invalid"*><center><font color=*"red"*>\*The date is required.</font></center></span>

<div class=*"row"*>

<button ng-disabled=*"todo.$invalid"* class=*"btn btn-primary btn-sm "* ng-click=*"addTodo()"*></i>Add task</button>

</div>

</form>

<div class=*"container"*>

<ul class=*"nav nav-tabs"*>

<li class=*"active"*><a data-toggle=*"tab"* href=*"#alltasks"*>All Tasks</a></li>

<li><a data-toggle=*"tab"* href=*"#pending"*>Pending Tasks</a></li>

<li><a data-toggle=*"tab"* href=*"#complete"*>Completed Tasks</a></li>

</ul>

<div class=*"tab-content"*>

<div id=*"alltasks"* class=*"tab-pane fade in active"*>

<table class=*"table "*>

<tbody>

<tr ng-repeat=*"resource in todos | filter:title "*>

<td>&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Title :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;<span ><font size=*"3"*><b>{{resource.title}}</b></font></span> <br><br>

&nbsp;&nbsp;&nbsp;decription:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span>{{resource.description}}</span><br><br>

&nbsp;&nbsp;&nbsp;categories:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span >{{resource.categories}}</span><br><br>

&nbsp;&nbsp;&nbsp;Dead Line: &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong ><i class=*"fa fa-calendar"*></i>{{resource.date | date : 'mediumDate'}}</strong> </li>

</td>

<td><br><br><br> &nbsp; &nbsp;&nbsp;Status :&nbsp;<span ><font color=*"red"*>{{resource.status}}</font></span></td>

</tr>

</tbody>

</table>

</div>

<div id=*"pending"* class=*"tab-pane fade"*>

<table class=*"table "*>

<tbody>

<tr ng-repeat=*"resource in todos | filter:{status:'Active'} | filter:title"*>

<td>&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Title :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;<span >{{resource.title}}</span> <br><br>

&nbsp;&nbsp;&nbsp;decription:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span>{{resource.description}}</span><br><br>

&nbsp;&nbsp;&nbsp;categories:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span >{{resource.categories}}</span><br><br>&nbsp;&nbsp;&nbsp;

&nbsp; &nbsp;&nbsp;Status :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;<span ><font color=*"red"*>{{resource.status}}</font></span><br><br>

&nbsp;&nbsp;&nbsp;Dead Line: &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong ><i class=*"fa fa-calendar"*></i>{{resource.date | date : 'mediumDate'}}</strong> </li>

</td>

<td><br><br><br><button type=*"button"* class=*"btn btn-primary btn-sm "* data-toggle=*"modal"* data-target=*"#edittodo"* ng-click=*"editTodo(resource)"*>Edit</button></td>

<td><br><br><br><button type=*"button"* class=*"btn btn-danger btn-sm "* ng-click=*"deleteTodo(resource)"*>Delete</button></td>

<td><br><br><br><button type=*"button"* class=*"btn btn-success"* ng-click=*"completeTodo(resource)"*>Done!</button></td>

</tr>

</tbody>

</table>

</div>

<div id=*"complete"* class=*"tab-pane fade"*>

<table class=*"table "*>

<tbody>

<tr ng-repeat=*"resource in todos | filter:{status:'Complete'} | filter:detail"*>

<td>&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Title :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;<span >{{resource.title}}</span> <br><br>

&nbsp;&nbsp;&nbsp;decription:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span>{{resource.description}}</span><br><br>

&nbsp;&nbsp;&nbsp;categories:&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span >{{resource.categories}}</span><br><br>&nbsp;&nbsp;&nbsp;

&nbsp; &nbsp;&nbsp;Status :&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;<span ><font color=*"green"*><img src=*"https://cdn.pixabay.com/photo/2016/03/31/14/37/check-mark-1292787\_960\_720.png"* style="width:*5%*" >{{resource.status}}</font></span><br><br>

&nbsp;&nbsp;&nbsp;Dead Line: &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<strong ><i class=*"fa fa-calendar"*></i>{{resource.date | date : 'mediumDate'}}</strong> </li>

</td>

<td><br><br><br><button type=*"button"* class=*"btn btn-danger btn-sm "* ng-click=*"deleteTodo(resource)"*>Delete</button></td>

</tr>

</tbody>

</table>

</div>

<!-- Modal -->

<div class=*"modal fade"* id=*"edittodo"* role=*"dialog"*>

<div class=*"modal-dialog"*>

<!-- Modal content -->

<div class=*"modal-content"*>

<div class=*"modal-header"*>

<button type=*"button"* class=*"close"* data-dismiss=*"modal"*>&times;</button>

<h4 class=*"modal-title"*>Update Task</h4>

</div>

<div class=*"modal-body"*>

<form>

<div class=*"inputContainer"*>

<input type=*"text"* class=*"form-control"* id=*"title"* placeholder=*"Enter company name"* ng-model=*"todoDataToEdit.title"*>

<center><label for=*"title"*>Title</label></center>

</div>

<div class=*"inputContainer"*>

<input type=*"text"* id=*"description"* class=*"form-control"* placeholder=*"What do you need to do?"* ng-model=*"todoDataToEdit.description"*>

<center><label for=*"description"*>Description</label></center>

</div>

<div class=*"inputContainer half last"*>

<div class=*"inputContainer"*>

<input type=*"text"* id=*"categories"* class=*"form-control"* placeholder=*"Enter categories"* ng-model=*"todoDataToEdit.categories"*>

<center><label for=*"categories"*>categories</label></center>

</div>

</div>

<div class=*"inputContainer half last right"*>

<input type=*"date"* id=*"date"* class=*"taskDate"* ng-model=*"todoDataToEdit.date"*>

<label for=*"date"*>Task Date</label>

</div>

</form>

<div class=*"form-group"*>

<div class=*"col-sm-offset-2 col-sm-10"*>

<button type=*"submit"* class=*"btn btn-primary btn-sm"* ng-click=*"saveEdit()"* data-dismiss=*"modal"*>Save Edit</button>

</div>

</div>

</form>

</div>

<div class=*"modal-footer"*>

<button type=*"button"* class=*"btn btn-default"* data-dismiss=*"modal"*>Close</button>

</div>

</div>

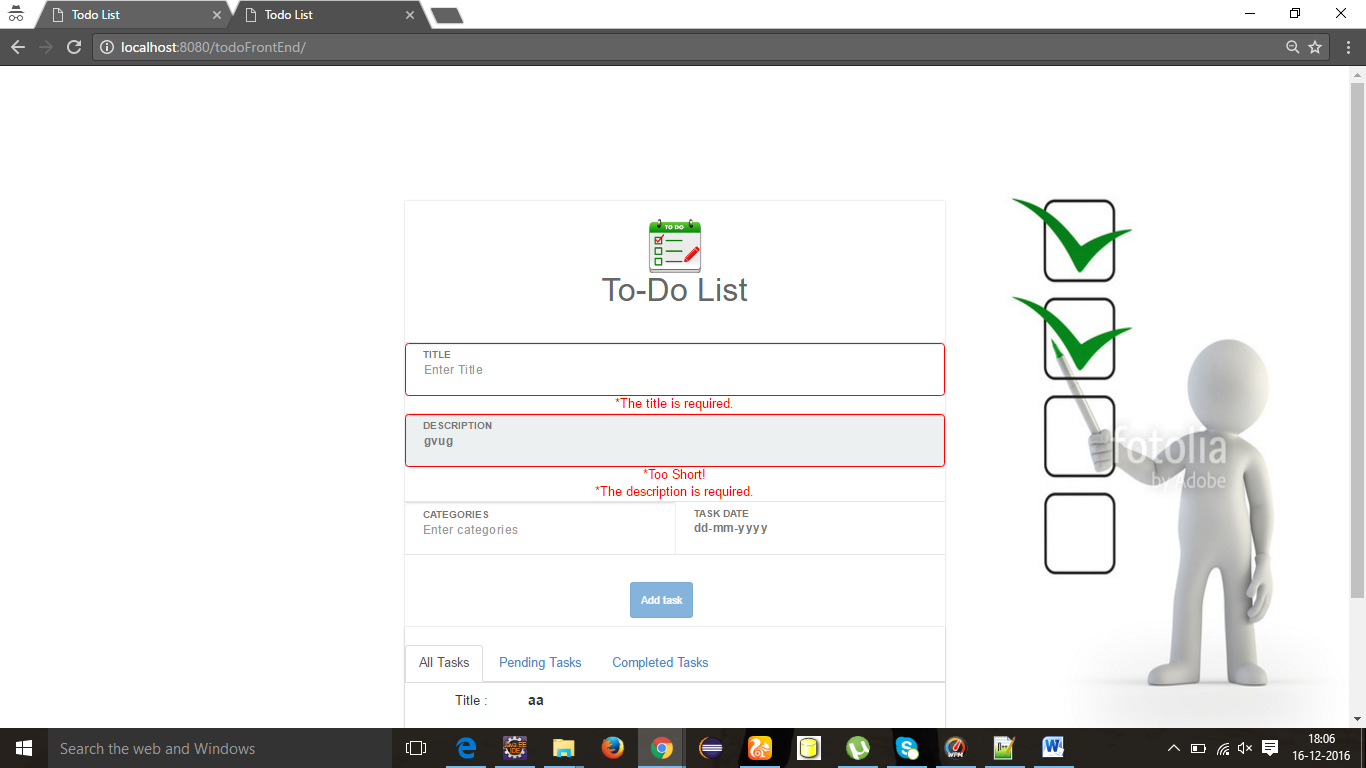
</div>

</div>

</body>

</html>

**Anjular-JS Validations:**

****

**Save & Run:**

The output of the project is

