Spring MVC + Hibernate + MySQL project set up for Register User

Contents

[1. Environment: 1](#_Toc473069140)

[2. Set up new project 1](#_Toc473069141)

[2.1. Create Spring project 1](#_Toc473069144)

[2.2. Create Entity class 5](#_Toc473069145)

[2.3. Create DAO 7](#_Toc473069146)

[2.4. Create Service 7](#_Toc473069147)

[2.5. Create Controller 7](#_Toc473069148)

[2.6. Create View 8](#_Toc473069149)

[3. Build your web application 8](#_Toc473069150)

[4. Deploy and run the application 9](#_Toc473069151)

# Environment:

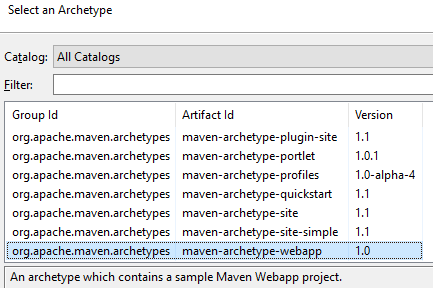
* JDK 1.8
* Maven 3.x
* Tomcat 8
* MySQL 5.x
* Eclipse Neon
* Spring 4
* Hibernate 4

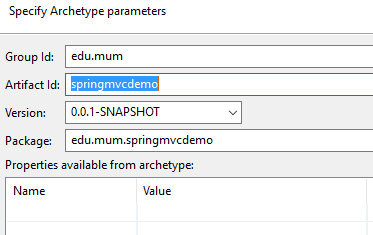
# Set up new project



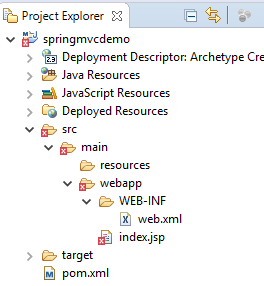
## Create Spring project

* + 1. Create maven project
* Open Eclipse
* File -> New -> Maven Project





Now your springmvcdemo project should look like below



You will need to create folder “***src/main/java***” as it’s missing during the project creattion

* + 1. Dependencies - pom.xml

This step is about to configure dependency library which we’re going to reference for the project

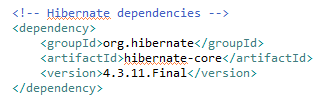
* Add Spring dependencies



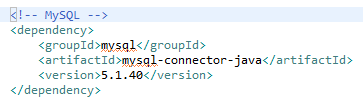
* Add Servlet, JSP, JSTL dependencies



* Add Hibernate dependencies



* Add MySQL dependencies



* Right click on project name -> Run As -> Maven install

The project should be built successfully

* + 1. Spring configuration
* web.xml



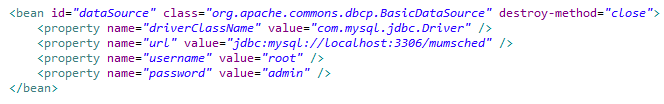
Update web.xml with DispatcherServlet and read the Spring configuration from springmvc-config.xml which is created by us

* springmvc-config.xml: create this file under folder WEB-INF/config

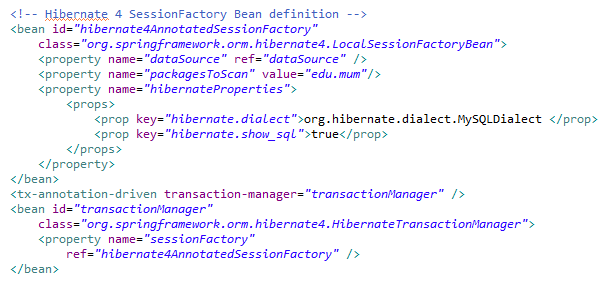
+ Configure Spring MVC view resolver



+ Configure datasource



+ Configure session factory and transaction manager



## Create Entity class

Suppose that we have a table name USER stored in the MySQL database

*CREATE TABLE `mumsched`.`user` (*

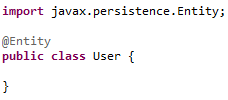
*`id` INT NOT NULL AUTO\_INCREMENT,*

*`username` VARCHAR(45) NOT NULL,*

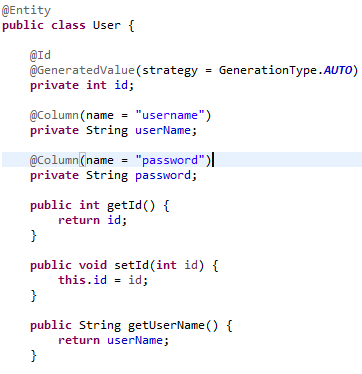
*`password` VARCHAR(45) NOT NULL,*

*PRIMARY KEY (`id`));*

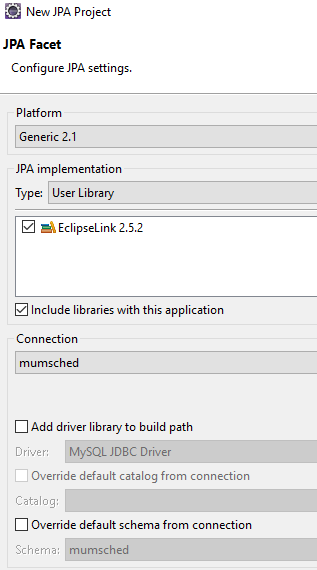
* + 1. Manually creation
* Create new package ***edu.mum.entity*** under ***src/main/java***
* Right click on this package -> New -> Class
* Add ***@Entity*** to the class



* Add fields matching to USER table columns, and all getter, setter



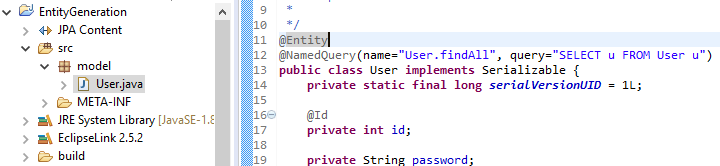
* + 1. Automatic generation
* File -> New -> Other -> JPA -> JPA Project



* Right click on created project -> New -> JPA Entities from Tables

(Select “user” as table you want to generate entity class)

* User.java file should be generated automatically



* Then you just need to copy this file and put it into springmvcdemo project

## Create DAO

* Create package ***edu.mum.dao*** under ***src/main/java***
* New -> Class

Create abstract class AbtractDAO which includes common operations to database such as find, persist, update, delete …

* New -> Interface

Create interface UserDAO which includes particular operations relating to User only

* New -> Class

Create UserDAOImpl which extends AbstractDAO and implement UserDAO and add *@Repository* to this new class

C:\Users\TuyenLam\AppData\Local\Temp\x10sctmp24.png

## Create Service

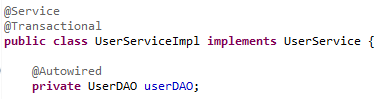
* Create package ***edu.mum.service*** under ***src/main/java***
* New -> Class

Create UserService interface which includes business logic operations and interact with the DAO layer

* New -> Class

Create UserServiceImpl class which implements the UserService interface, and add *@Service, @Transactional* to it

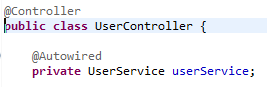
This service needs to invoke a UserDAO to access to DAO layer so you add it and put @Autowired so that the container can ininitial it for your usage



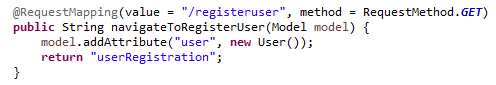
## Create Controller

* Create package ***edu.mum.controller*** under ***src/main/java***
* New -> Class

Create UserController class which handles the request, invoke the service, and return the response. Add @Controller to this class, add the service with @Autowired so that the container can initialize it for your usage



* Add request mappings as operations where you handle the request and return the response. e.g:



It means that when user sends a GET request to the url “/registeruser”, this operation will be invoked and it eventually tells the view resolver to return the userRegistration.jsp and display to user

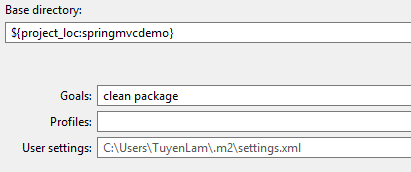
## Create View

* Create new folder WEB-INF/jsp where you put all the jsp files for the view
* Right click on that folder -> New -> JSP file

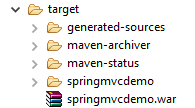
Create userRegistration.jsp file

# Build your web application

* Right click on the project -> Run As -> Maven build … -> enter “clean package” into the Goal

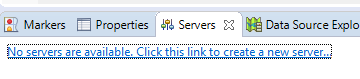


* After running, a bunch of stuffs is generated under ***target*** folder including the war for the deployment

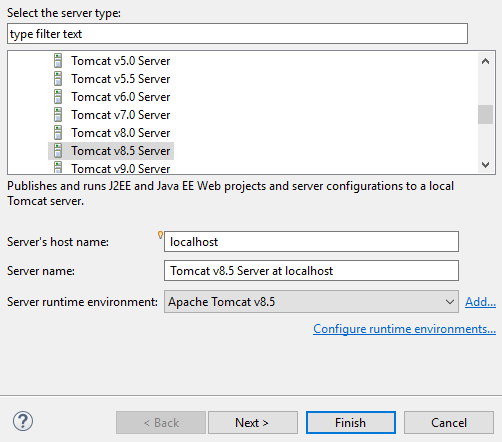


# Deploy and run the application

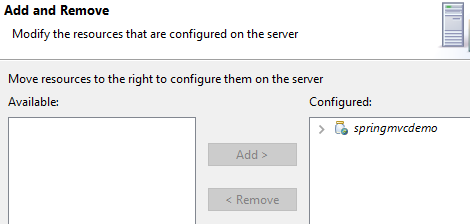
* Create a new server and deploy the web application to the server



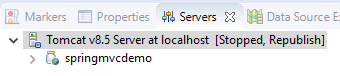
By clicking on the link. In case you don’t see that tab, simply add it by go to Window -> Show View -> Other -> Server -> Servers



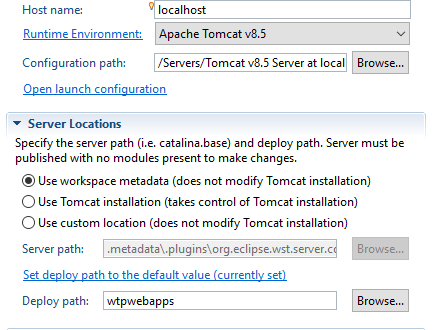
Add the web application that you want to deploy



After clicking on Finish button, here is what you should see

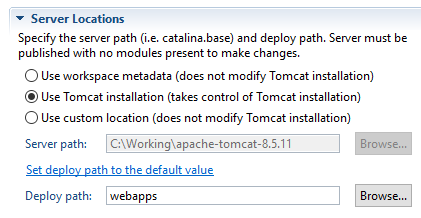


As my experience, you would need to change some information after createing a new server. Go double click on “Tomcat v8.5 Server at localhost”



Server Locations is where you would want to change:

* Check “User Tomcat installation” to point to your install Tomcat
* Change Deploy path to “webapps”



Then click “Ctrl-C” to save your changes

* Run the server

Right click on “Tomcat v8.5 Server at localhost” -> Start (or Debug) to run your web application

Then access the url <localhost:8080/springmvcdemo/registeruser> and enjoy ☺

