Phase-End-Project 2

JSP Servlet Hibernate CRUD Application

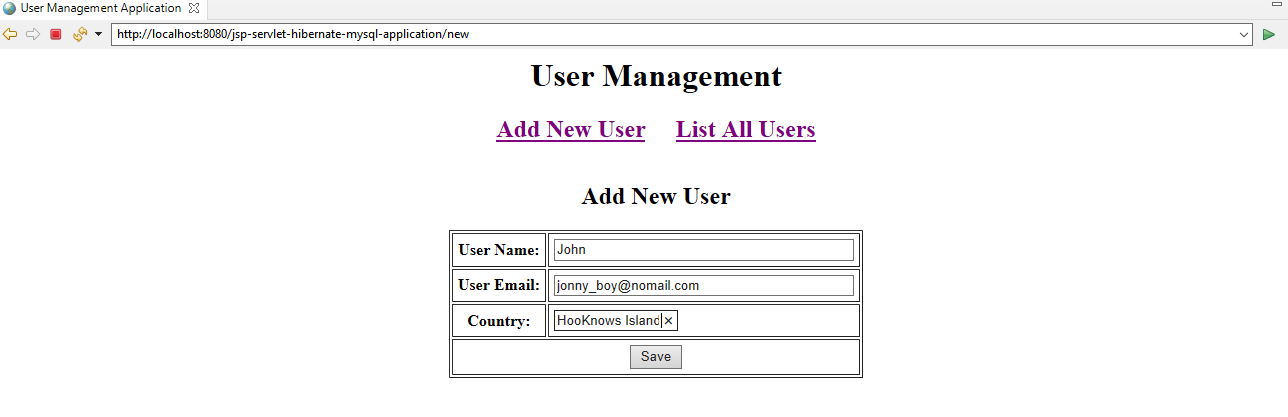
In this phase-end project, we are building a simple **User Management** web application using JSP, Servlet, and Hibernate. This web application manages a collection of **users** with the basic feature: list, insert, update, delete (or CURD operations - Create, Update, Read and Delete).

We will use **[Hibernate Java-based configuration](https://www.javaguides.net/2018/11/hibernate-5-java-configuration-example.html" \t "https://www.javaguides.net/2019/03/_blank)** without using *hibernate.cfg.xml* to connect MySQL database.

We will develop below simple basic features in our **User Management** web application:

1. Create a User
2. Update a User
3. Delete a User
4. Retrieve a User
5. List of all Users

The application looks something like this:



# Tools and technologies used

* JSP - 2.2 +
* Hibernate - 5+
* IDE - STS/Eclipse Neon.3
* JDK - 1.8 or later
* Apache Tomcat - 8.5
* JSTL - 1.2.1
* Servlet API - 2.5
* MySQL - mysql-connector-java-8.0.13.jar

The jars are available in the Phase-End-Project 2 jars folder.

# Development Steps

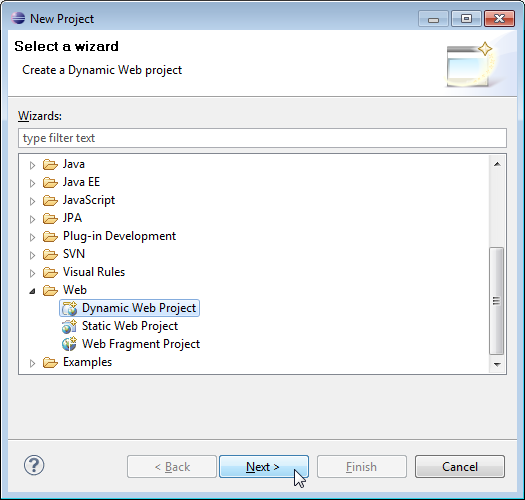
1. Create an Eclipse Dynamic Web Project
2. Add Dependencies
3. Project Structure
4. MySQL Database Setup
5. Create a JPA entity - User.java
6. Hibernate Java-based configuration
7. Create a UserDAO.java
8. Create a UserServlet.java
9. Creating User Listing JSP Page - user-list.jsp
10. Create a User Form JSP Page - user-form.jsp
11. Creating Error JSP page
12. Deploy and Test the Application

# 1. Create an Spring ToolSuite Dynamic Web Project

To create a new dynamic Web project in STS:

1. On the main menu select **File > New > Project....**

2. In the upcoming wizard choose **Web > Dynamic Web Project.**

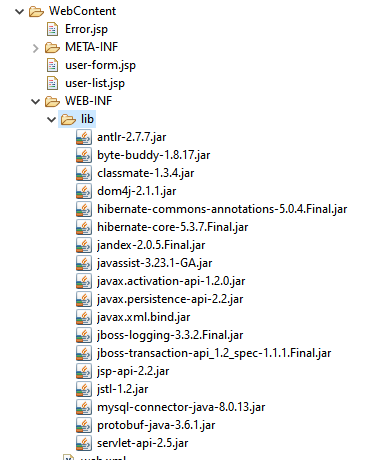
**[](https://3.bp.blogspot.com/-CeA278XIG4g/XDiv1ePWVlI/AAAAAAAAFZI/2mvCujNS2Co_gp1UqevtbZk0qhDsaa9DwCLcBGAs/s1600/create-web-proj-1.png)**

3. Click **Next**.

4. Enter project name as "jsp-servlet-hibernate-mysql-application";  
5. Make sure that the target runtime is set to Apache Tomcat with the currently supported version.

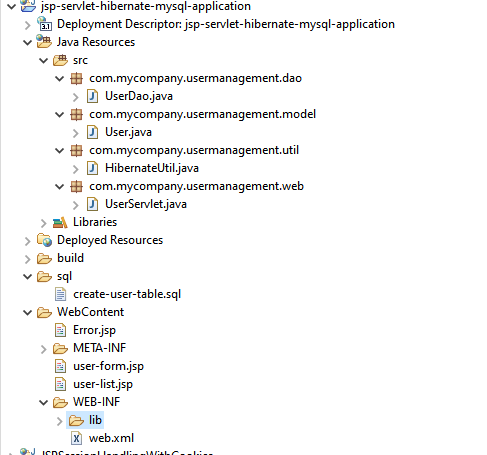
# 2. Add Dependencies

Add the latest release of below jar files to the lib folder:



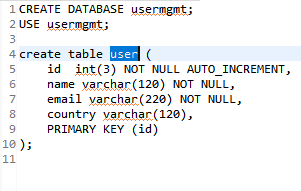
# 3. Project Structure

Standard project structure for your reference -

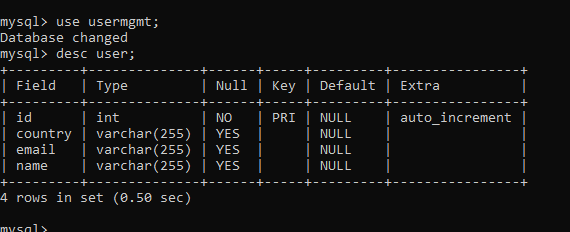


# 4. MySQL Database Setup

Let's create a database named "usermgmt" in MySQL. Now, create a **user** table using below DDL script:



You can use either MySQL Command Line Client or MySQL Workbench tool to create the database. The above a **user**table looks like this:

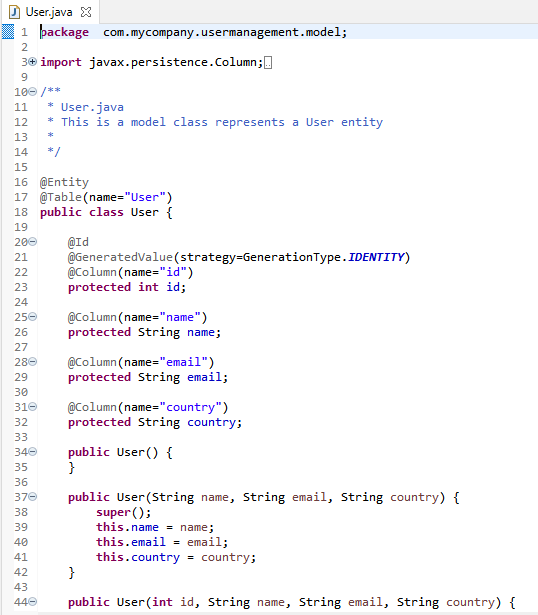


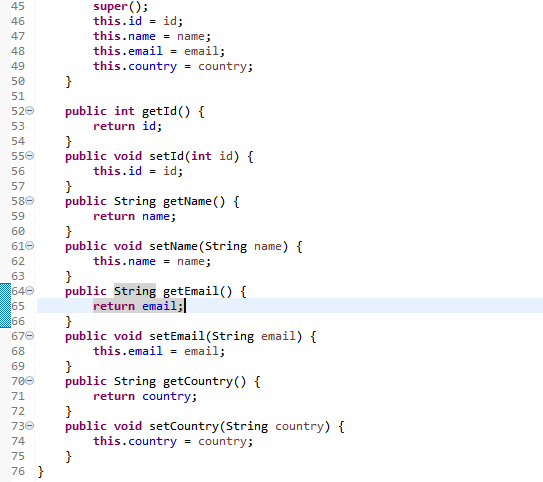
# 5. Create a JPA Entity - User.java

Let's create a *User* persistent class that is mapped to a database table - user.

A simple Persistent class should follow some rules:

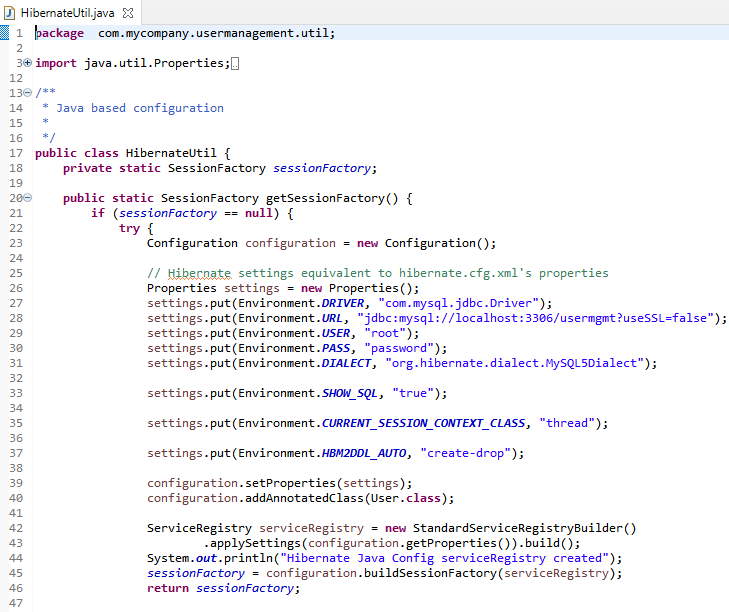
* **A no-arg constructor:** It is recommended that you have a default constructor with at least package visibility so that hibernate can create the instance of the Persistent class by newInstance() method.
* **Provide an identifier property:** It is better to assign an attribute as an id. This attribute behaves as a primary key in a database.
* **Declare getter and setter methods:** The Hibernate recognizes the method by getter and setter method names by default.
* **Prefer non-final class:**Hibernate uses the concept of proxies, which depends on the persistent class. The application programmer will not be able to use proxies for lazy association fetching.

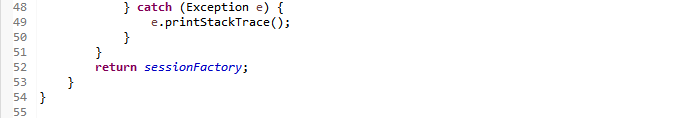




# 6. Hibernate Java-based configuration

 Java code for Hibernate settings equivalent to hibernate.cfg.xml's properties:



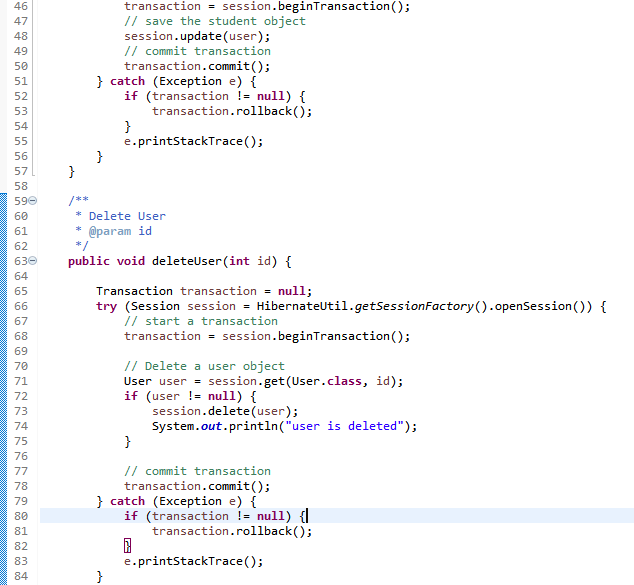


* ServiceRegistry holds the services that Hibernate will need during bootstrapping and at runtime.
* StandardServiceRegistryBuilder - Builder for standard ServiceRegistry instances.
* HibernateUtil - This is a helper class to bootstrap hibernate SessionFactory. In most Hibernate applications, the SessionFactory should be instantiated once during application initialization. The single instance should then be used by all code in a particular process, and any Session should be created using this single SessionFactory. The SessionFactory is thread-safe and can be shared; a Session is a single-threaded object.

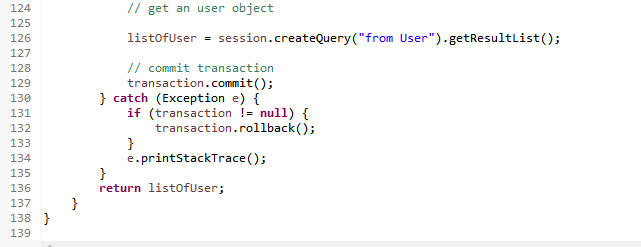
# 7. Create a UserDAO.java

Let's create a *UserDAO* class which is a Data Access Layer (DAO) class that provides CRUD (Create, Read, Update, Delete) operations using for the **user table**in a database using Hibernate. Here’s the full source code of the *UserDAO*:





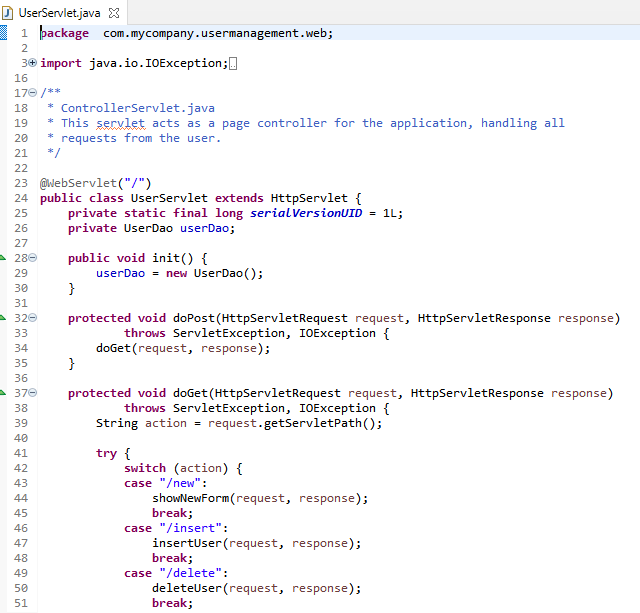




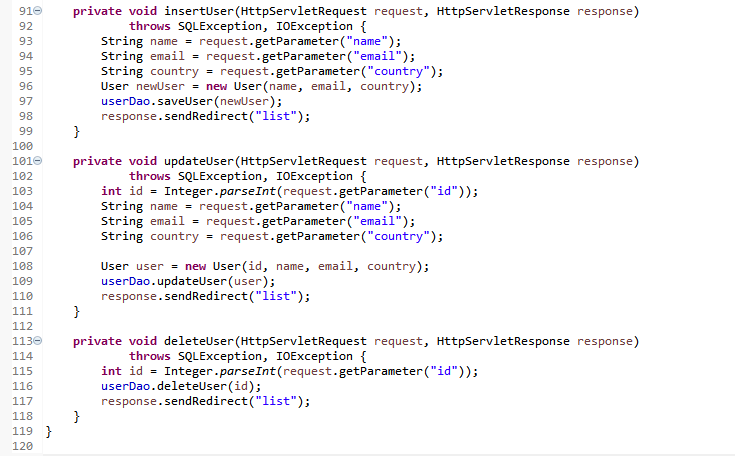
# 8. Create a UserServlet.java

Now, let's create a *UserServlet* that acts as a page controller to handle all requests from the client.

Let’s look at the code first:



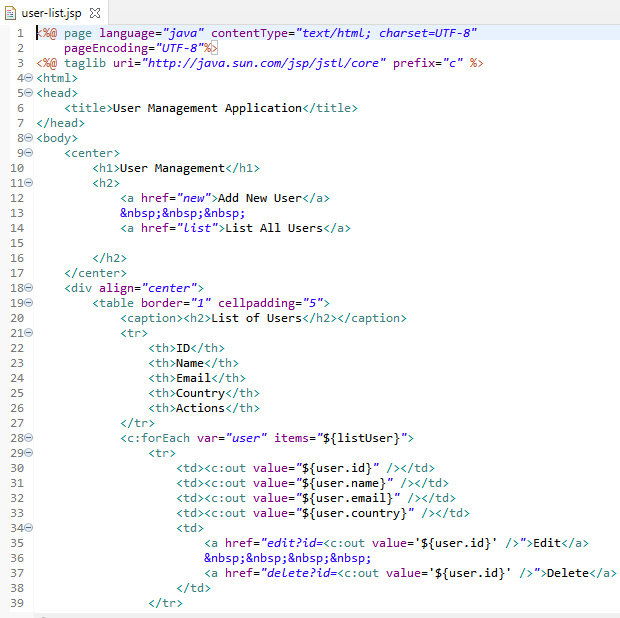




# 9. Creating User Listing JSP Page - user-list.jsp

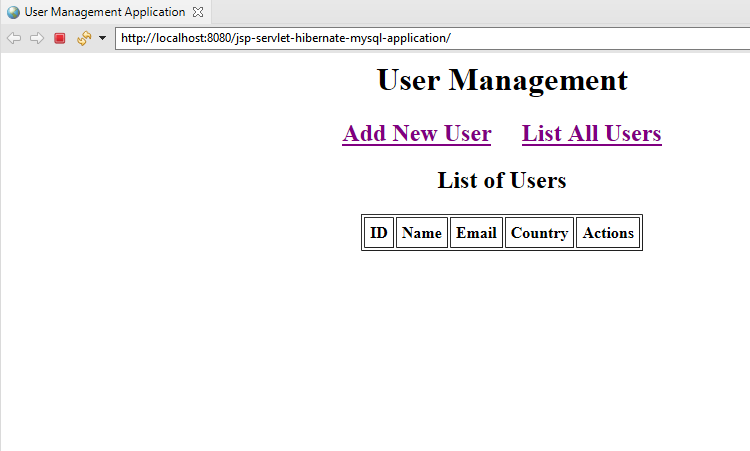
Next, create a JSP page for displaying all **users**from the database.

Let's create a *user-list.jsp* page under the **WebContent** directory in the project with the following code:



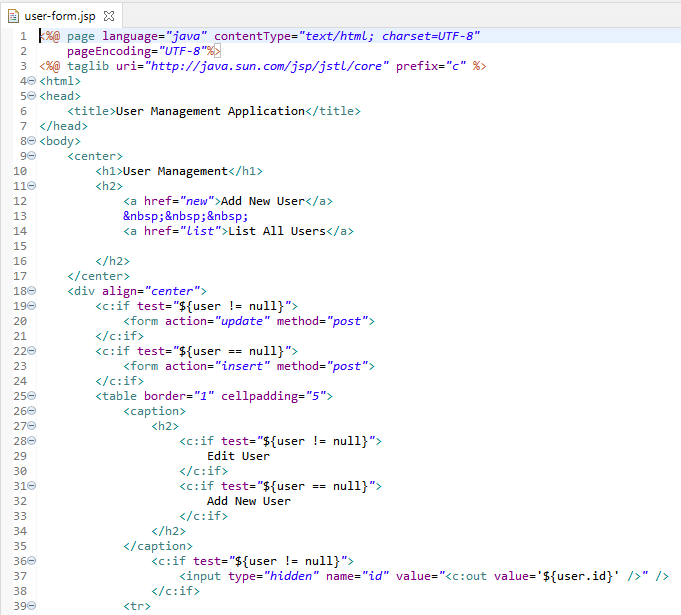


Once you will deploy the above JSP page in tomcat and open it in the browser looks something like this:



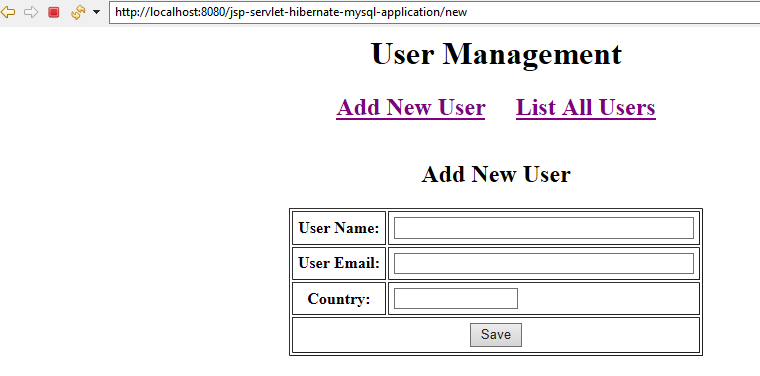
# 10. Create a User Form JSP Page - user-form.jsp

Next, we create a JSP page for creating a new User called *user-form.jsp*. Here’s its full source code:

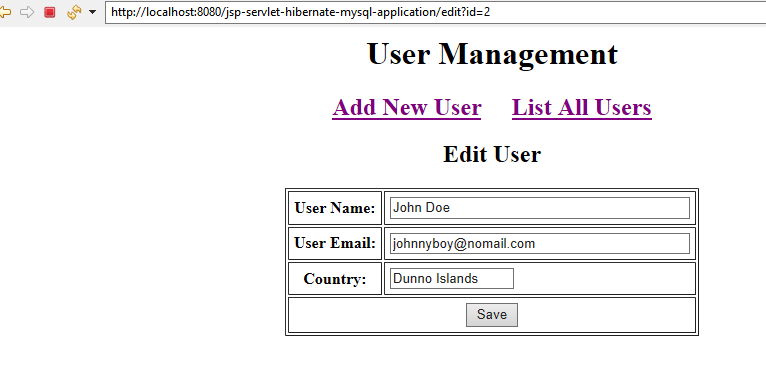




Once you will deploy the above JSP page in tomcat and open it in the browser looks something like this:



The above page acts for both functionalities to create a new User and Edit the same user. The edit page looks like this:

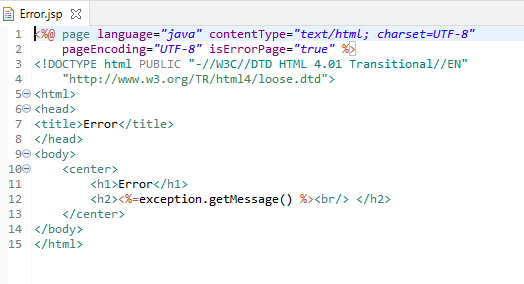


# 11. Creating Error JSP page

Here’s the code of the *Error.jsp* page which simply shows the exception message:

# 11. Creating Error JSP page

Here’s the code of the *Error.jsp* page which simply shows the exception message:



# 12. Deploy and Test the Application

It's time to see a demo of the above **User Management** web application. Deploy this web application to the tomcat server.

From the Home Page, you can access the various features and functionalities of the application.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*