Course code:	CSE3427
Course Name:	Java Full Stack Development
Credit Structure: (L-T-P-C)	2-0-2-3
Semester:	Even Semester 2025-26
Name of the Faculty:	Dr. T Ramesh
Program:	B.Tech
Year:	2025-2026
Section:	CSE & CSE Allied

Sl.No	List of Lab Experiments
1.	Use Serialization and Deserialization mechanism to develop a console application.
2.	Build a console application by using Collection framework and Annotation.
3.	Build a console application by using Collection framework and Lambda Expression
4.	Develop a console application that connect with MySQLDatabase and perform database transactions.
5.	Build a web application to connect with a database using Servlet that perform database manipulations.
6.	Build web application to connect with a database using JSP that perform database manipulations.
7.	Construct a login application in respecting the MVC model.
8.	Implement a web application based on the MVC design pattern, to create an **Employee Registration** module using **JSP**, **Servlet**, **JDBC**, and **MySQL** database.
9.	Create Student mark processing project using Hibernate with Maven.
10.	Create a User Registration project using **JSP**, **Servlet**, **Hibernate Framework**, and **MySQL** database.
11.	Develop a User Login Form and will validate username and password with the MySQL database using the Hibernate framework.
12.	Build a complete Hibernate application with HQL CRUD operations using **MAVEN** ,**JSP**, **Servlet**, **Hibernate Framework**, JPQL and **MySQL** database.
13.	Build CRUD RESTful API using Spring Boot 3, Spring Data JPA (Hibernate), and MySQL database.
14.	Build login or sign-in and registration or signup REST API using Spring boot, Spring Security, Hibernate, and MySQL database.
15.	Create Spring web application to implement SpringMVC framework using eclipse IDE

16.	Create Spring application to implement Spring AOP using eclipse IDE.

Ex No: 1 Use Serialization and Deserialization mechanism to develop a console application.

AIM:

To implement the Serialization and Deserialization mechanism using File and demonstrated in a Java console application.

ALGORITHM:

Step 1: Creating a Student class and are serializing the object of the Student class.

Step 2: Serialization of an Object of type Student. A text file called f.txt is created with the help of the FileOutputStream class. Serializing the object by using the writeObject() method of ObjectOutputStream class.

Step 3: For deserializing the object by using the readObject() method of ObjectInputStream class.

PROGRAM:

Student.java

```
import java.io.Serializable;

public class Student implements Serializable {
     int id;
     String name;
     float  fees;
 public Student(int id, String name, float fees) {
    this.id = id;
    this.name = name;
    this.fees = fees;
 }
public String toString() {
    return id + "   " + name + "  " + fees + "\n";
}
}
```

Persist.java

```
import java.io.*;
class Persist{
```

```
 public static void main(String args[]){
  try{
        Student s1 =new Student(1,"ram",10000.00f);
        FileOutputStream fout=new FileOutputStream("d:\\f.txt");
        ObjectOutputStream out=new ObjectOutputStream(fout);
         out.writeObject(s1);
        out.flush();
        out.close();
        System.out.println("success");
  }catch(Exception e){
        System.out.println(e);
}
 }
}
```

Depersist.java

```
import java.io.*;
class Depersist {
 public static void main(String args[]){
       try{
            FileInputStream fin =  new FileInputStream("d:\\f.txt");
           ObjectInputStream in=new ObjectInputStream(fin);
           Student s=(Student)in.readObject();
           System.out.println(s);
          in.close();
  }catch(Exception e){
        System.out.println(e);
  }
 }
}
```

RESULT:

Thus the Serialization and Deserialization using File was implemented in a Java console application.

Expt 2:

Build a console application by using Collection framework and Annotation.

AIM:

To implement the collection framework and annotation by develop a java console application.

ALGORITHM:

Step 1 : Student class contains fields and age and a parameterized constructor.
Step 2: AgeComparator class defines comparison logic based on the age.
Step 3: NameComparator class provides comparison logic based on the name.
Step 4: FeesComparator class provides comparison logic based on the fees.
Step 5: Main class printing the values of the object by sorting on the basis of name, age and fees.

PROGRAM:

```
import java.io.*;
import java.util.*;

class Student {
        int rollno;
        String name;
        float fees;
        String branch;
        int year;
        int sem;
        int age;
   static String clg;

   public Student(int rollno,String name,float fees,String branch,int year,int sem,int age) {
        this.rollno = rollno;
        this.name = name;
        this.fees = fees;
        this.branch = branch;
        this.year = year;
        this.sem = sem;
        this.age = age;
        clg="PU";
   }
   @Override
   public String toString() {
        return  rollno + " "+ name + " " + fees + " " + branch + " " + year + sem + " " + age + " "
+ clg + "\n";
   }
```

```
}
class AgeComparator implements Comparator {
	public int compare(Object o1, Object o2) {
		Student s1=(Student)o1;
		Student s2=(Student)o2;
		if(s1.age==s2.age)
			return 0;
		else if(s1.age>s2.age)
			return 1;
		else
			return -1;
	}
}
class NameComparator implements Comparator{
	public int compare(Object o1, Object o2) {
		Student s1=(Student)o1;
		Student s2=(Student)o2;
		return  s1.name.compareTo(s2.name);
}
}
class FeesComparator implements Comparator {
	public int compare(Object o1,Object o2) {
		Student s1=(Student)o1;
		Student s2=(Student)o2;
		if(s1.fees==s2.fees)
			return 0;
		else if(s1.fees>s2.fees)
			return 1;
		else
			return -1;
	}
}

public class Temp1 {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ArrayList sl=new ArrayList();
		sl.add(new Student(1,"Shiva",10000.00f,"cse",1,1,18));
		sl.add(new Student(2,"Venky",15000.00f,"ise",1,2,20));
		sl.add(new Student(3,"Jesus",17000.00f,"ece",1,1,19));
		sl.add(new Student(3,"Alla",12000.00f,"eee",1,1,19));
		sl.add(new Student(3,"Budha",11000.00f,"mech",1,1,21));
		System.out.println("Sorting by Name");
		System.out.println("_________________");
		Collections.sort(sl,new NameComparator());
```

```
                Iterator itr=sl.iterator();
                while(itr.hasNext()){
                        Student st=(Student)itr.next();
                        System.out.println(st. );
                }
                System.out.println("Sorting by age");
                System.out.println("_________________");
                Collections.sort(sl,new AgeComparator());
                itr=sl.iterator();
                while(itr.hasNext()) {
                        Student st=(Student)itr.next();
                        System.out.println(st );
                }
                System.out.println("Sorting by fees");
                System.out.println("_________________");
                Collections.sort(sl,new FeesComparator());
                 itr=sl.iterator();
                while(itr.hasNext()){
                        Student st=(Student)itr.next();
                          System.out.println(st);
                  }
        }
}
```

RESULT:

Thus the collection framework was implemented by using its interfaces such as Collection, Iterator and Comparator that can be demonstrated with a java console application.

Expt3:

Build a console application by using Collection framework and Lambda Expression

AIM:

To implement the collection framework, access the collection by using Lambda Expression and demonstrate it with a java console application.

ALGORITHM:

Step 1 : Student class contains fields such as rno,name,fees and age and a parameterized constructor.

Step 2: To sort the names by using Collections sort method and Lambda expression.Access the collection by using Lambda expression within for statement.

Step 2: To sort the age by using Collections sort method and Lambda expression.Access the collection by using Lambda expression within for statement.

Step 2: To sort the fees by using Collections sort method and Lambda expression.Access the collection by using Lambda expression within for statement.

Step 5: Main class printing the values of the object by sorting on the basis of name, age and fees.

Program:

```
import java.util.*;

 class Student{
    int rno;
    String name;
    int age;
    float fees;

    public Student(int rno, String name, int age, float fees) {
        super();
        this.rno = rno;
        this.name = name;
        this.age = age;
        this.fees = fees;
    }


    @Override
    public String toString() {
        return rno + "   " + name + "  " + age + "   " + fees;
    }
}
class temp {
    public static void main(String[] args) {
       List <Student> s = new <Student>ArrayList();
       s.add(new Student(1,"abc",20,20000.00f));
       s.add(new Student(2,"xyz",15,15000.00f));
       s.add(new Student(3,"def",10,10000.00f));

       System.out.println("Sorting on the basis of name...");

        // implementing lambda expression
        Collections.sort(s,(s1,s2)->{return s1.name.compareTo(s2.name);});
          for(Student i:s){
            System.out.println(i);
        }
        System.out.println("Sorting by age");

        Collections.sort(s,(s1,s2)-> s1.age - s2.age);
        s.forEach((l)->System.out.println(l));

        System.out.println("Sorting by Fees");
```

```
        Collections.sort(s,(s1,s2)-> (int)s1.fees - (int)s2.fees);
        s.forEach((l)->System.out.println(l));

    }
}
```

RESULT:

Thus the collection framework was implemented with Collection, and access by using Lambda expression that can be demonstrated with a java console application.

Expt4:

Develop a console application that connect with MySQLDatabase and perform database transactions.

AIM:

To develop a Java console application that demonstrates the connection with MySQL database and perform various operations on it.

ALGORITHM:

Step 1: create employee database by using create database employee; and use employee; commands.

Step 2: create a table in the mysql database by create table emp(eno **int**(10),name varchar(40),age **int**(3));

Step 3: **Register the JDBC driver:** to initialize a driver so that open a communication channel with the database.

Step 4: **Open a connection:** use the *getConnection()* method to create a Connection object, which represents a physical connection with the database.

Step 5: **Execute a query:** requires to use an object of type Statement for building and submitting an SQL statement to the database.

Step 6: **Extract data from the result set:** use the appropriate *getXXX()* method to retrieve the data from the result set.

Step 7: **Clean up the environment:** to explicitly close all database resources versus relying on the JVM's garbage collection.

PROGRAM:

```
import java.sql.*;
import java.util.*;
public class Temp2 {

        public static void main(String[] args) {
                // TODO Auto-generated method stub
```

```
                try{
                        Class.forName("com.mysql.jdbc.Driver");
                        Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/employee?characterEncoding=
latin1","root","root");
                        Statement stmt=con.createStatement();
                int ans=1;
                do {
                   System.out.println("1. Insert a record ");
                   System.out.println("2. Delete a record ");
                   System.out.println("3. Modify/Edit a record ");
                   System.out.println("4. Display list of records ");
                   Scanner sc = new Scanner(System.in);
                   System.out.println("Enter your choice:");
                   int ch = sc.nextInt();
                   String ename;
                   int eno,age;
                   String query="";

                   switch(ch) {
                     case 1:
                        System.out.println("Enter employee number:");
                        eno = sc.nextInt();
                        System.out.println("Enter employee name:");
                        ename = sc.next();
                        System.out.println("Enter employee age:");
                        age = sc.nextInt();
                        query = "INSERT INTO emp " + "VALUES (" + eno+ ",'" + ename+"',"+
age+")";
                        stmt.executeUpdate(query);
                        break;
                     case 2:
                        System.out.println("Enter employee number:");
                        eno = sc.nextInt();
                        query = "delete from  emp where eno='"+eno+"'";
                        stmt.executeUpdate(query);
                        System.out.println("Record is deleted from the table
successfully.....");
                        break;
                     case 3:
                        PreparedStatement ps = null;
                        query = "update emp set name=? where eno=? ";
                        ps = con.prepareStatement(query);
                        System.out.println("Enter employee number:");
                        eno = sc.nextInt();
                        System.out.println("Enter employee name:");
```

```
                        ename = sc.next();
                        ps.setString(1, ename);
                        ps.setInt(2, eno);
                        ps.executeUpdate();
                        System.out.println("Record is updated successfully.....");
                        break;
                    case 4:
                        ResultSet rs=stmt.executeQuery("select * from emp");
                                while(rs.next())
                                    System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"
"+rs.getInt(3));
                        }
                        System.out.println("Enter another(1/0)");
                        ans = sc.nextInt();
                  }while(ans==1);

                        con.close();
                }catch(Exception e){ System.out.println(e);}
        }

}
```

RESULT:

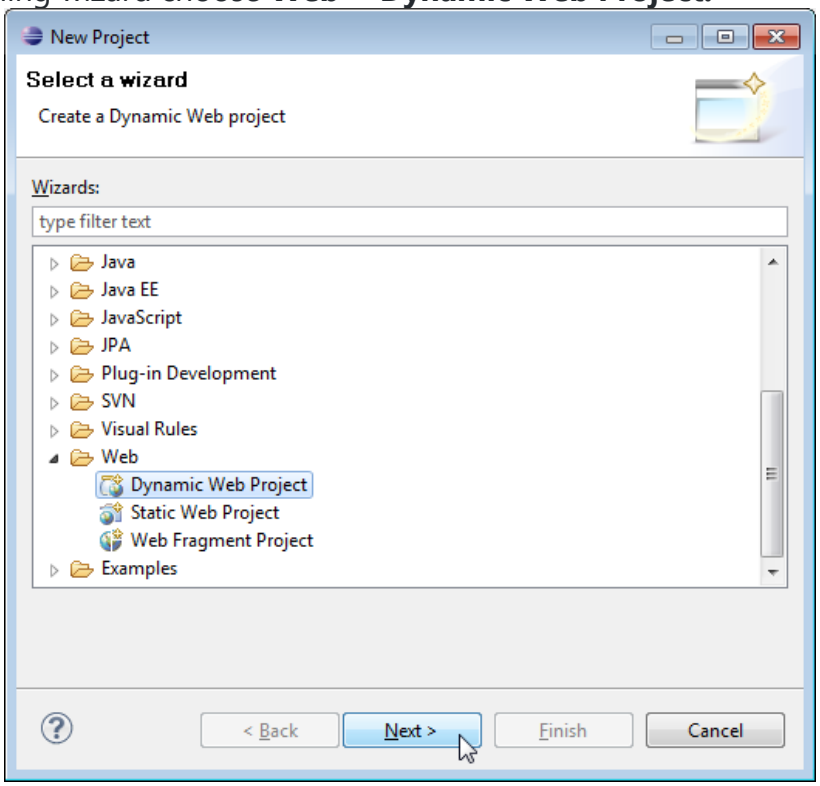
Thus the Java console application that demonstrated the connection with MySQL database and perform various operations on it.

Expt5:
Build a web application to connect with a database using Servlet that perform database manipulations.

AIM:
To build a simple Student Mark Processing system using Servlet, JDBC, and MySQL database.
ALGORITHM

1. Create an Eclipse Dynamic Web Project

To create a new dynamic Web project in Eclipse:
1. On the main menu select **File > New > Project....**
2. In the upcoming wizard choose **Web > Dynamic Web Project.**



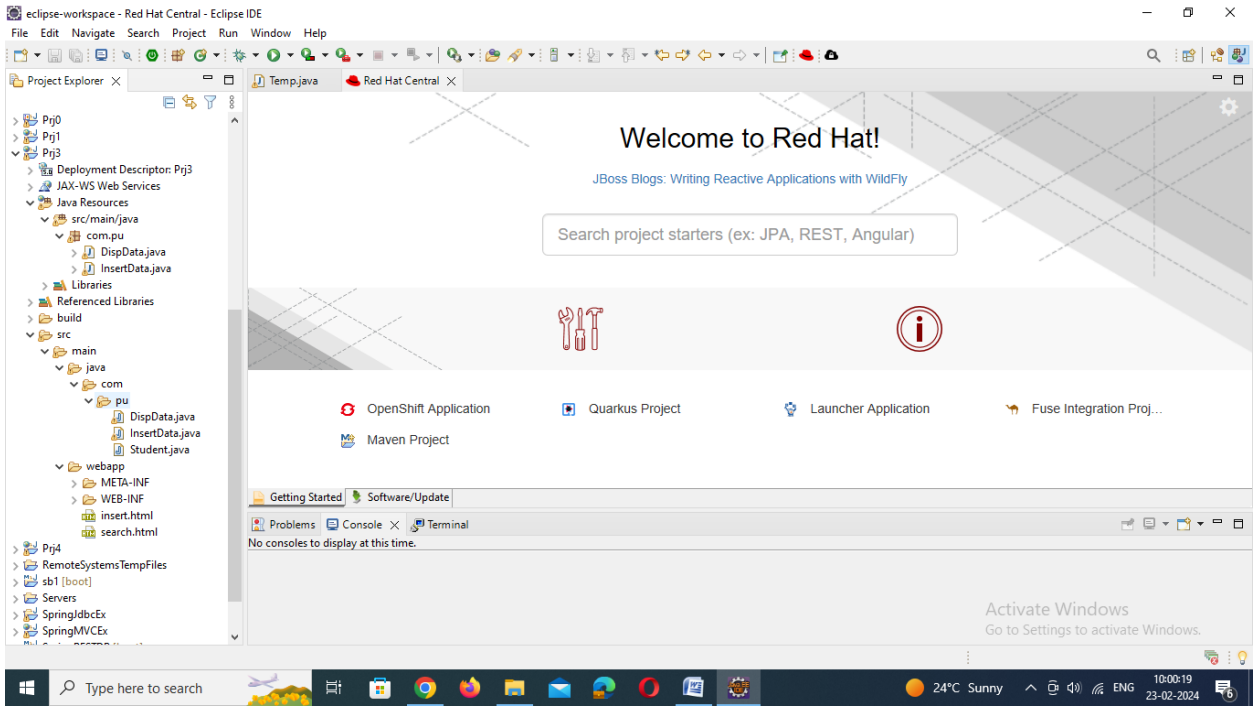
3. Click **Next**.
4. Enter project name as "Exp5";
5. Make sure that the target runtime is set to Apache Tomcat with the currently supported version.

2. Add Dependencies

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

- servlet-api.jar
- mysql-connector.jar

3. Project Structure



4. MySQL Database Setup

- Let's create a database named "student" in MySQL and create an `mark` table using below DDL script:

  Create database student;
  Use student;
  Create table mark(rollno int,name varchar(50),section varchar(1),s1 int,s2 int,s3 int, s4 int, s5 int,s6 int,l1 int,l2 int)

Step 2: To create a Insert HTML file to accept the user input marks in html form.

Step 3: To create a InsertData Servlet program to process the user request and store the marks into database.
Step 5: To create a Search HTML file to accept the user input as roll number in html form.
Step 6: To create a DispData Servlet program to process the user request and retrieve the selected roll number and process the result status from database.

main.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<marquee>Student Database Management System</marquee>
<ul>
<li><a href="./list.html">List</a></li>
<li><a href="./insert.html">Insert</a></li>
<li><a href="./delete.html">Delete</a></li>
<li><a href="./update.html">Update</a></li>
<li><a href="./search.html">Search</a></li>
</ul>
</body>
</html>
```

insert.html

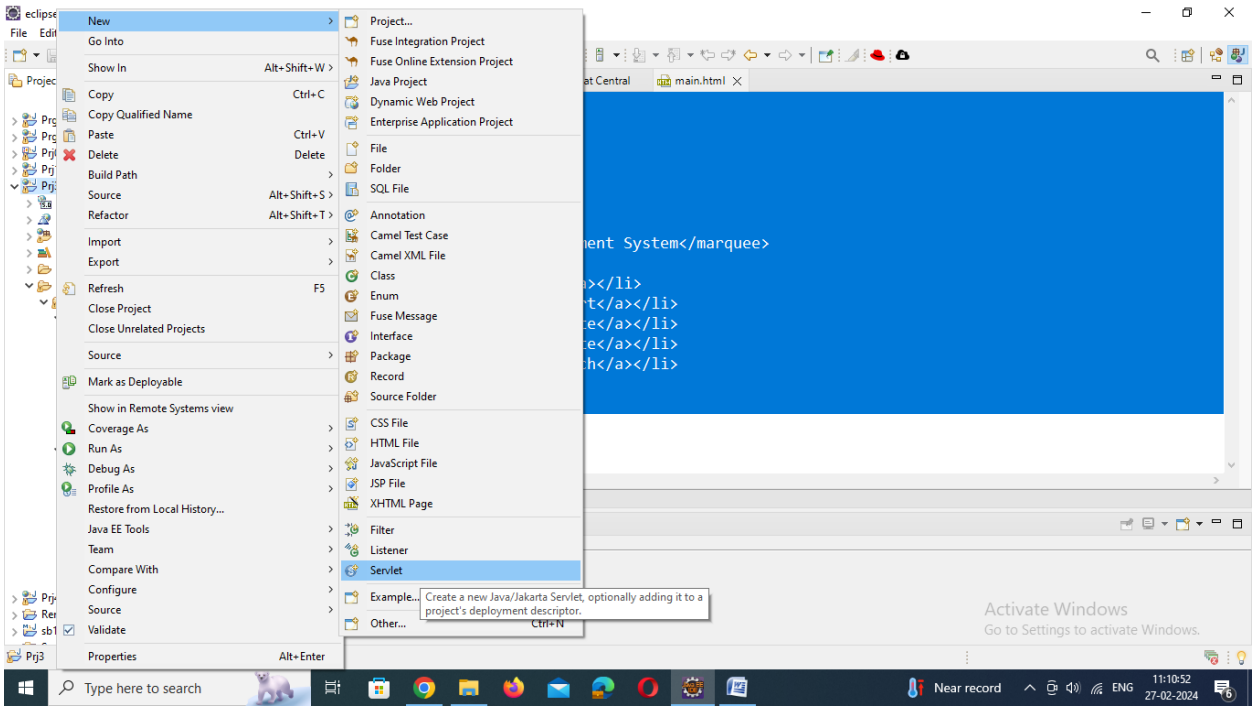
```
<!DOCTYPE html>
<html>
<head>
<title>Insert title here</title>
</head>
<body>
<form action="./InsertData" method="post">
<table>
<tr><td>Enter Roll No:</td>
<td><input type="text" name="rollno"/></td>
</tr>
<tr><td>Enter Name:</td>
<td><input type="text" name="name"/></td>
</tr>
<tr><td>Enter Section:</td>
<td><select name="section">
<option>A</option>
<option>B</option>
```

```
<option>C</option>
<option>D</option>
</select>
</td>
</tr>
<tr><td>Enter Subjet1 Mark:</td>
<td><input type="text" name="sub1"/></td>
</tr>
<tr><td>Enter Subject2 Mark:</td>
<td><input type="text" name="sub2"/></td>
</tr>
<tr><td>Enter Subjet3 Mark:</td>
<td><input type="text" name="sub3"/></td>
</tr>
<tr><td>Enter Subject4 Mark:</td>
<td><input type="text" name="sub4"/></td>
</tr>
<tr><td>Enter Subject5 Mark:</td>
<td><input type="text" name="sub5"/></td>
</tr>
<tr><td>Enter Subject6 Mark:</td>
<td><input type="text" name="sub6"/></td>
</tr>
<tr><td>Enter Lab1 Mark:</td>
<td><input type="text" name="lab1"/></td>
</tr>
<tr><td>Enter Lab2 Mark:</td>
<td><input type="text" name="lab2"/></td>
</tr>
<tr><td><input type="submit"/></td></tr>
</table>
</form>
</body>
</html>
```

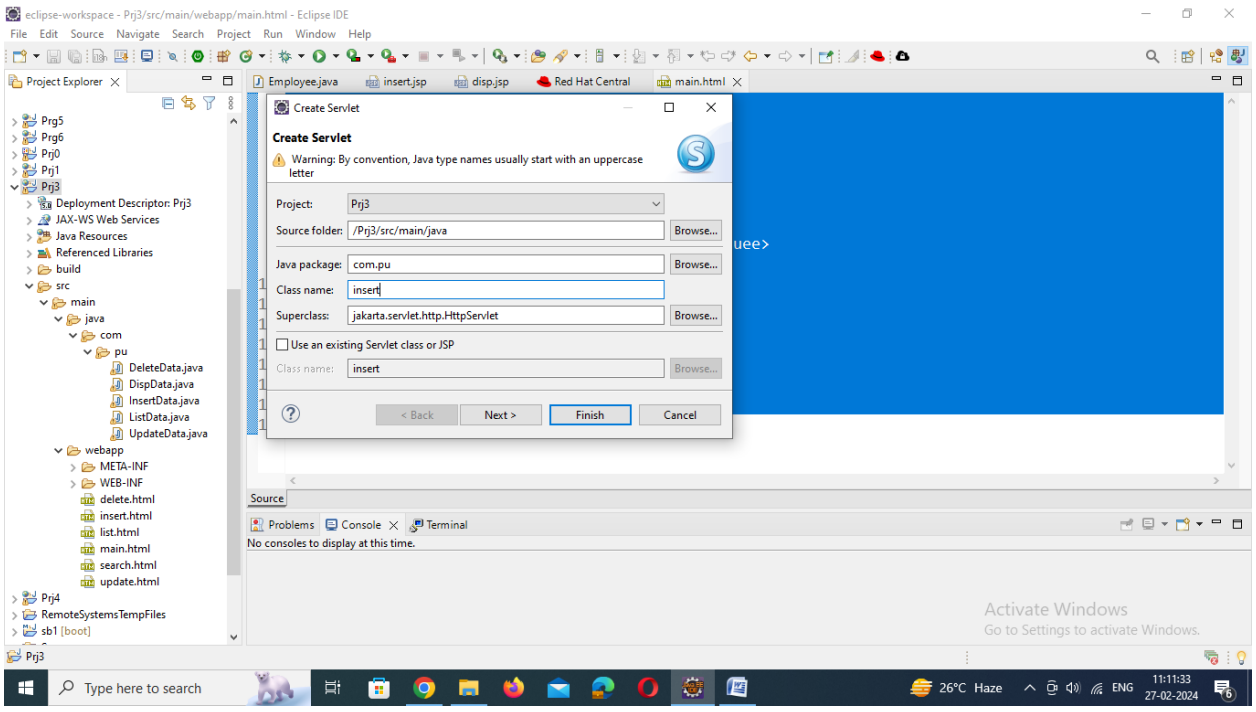
6. Create an InsertData Servlet program

1.Select the project

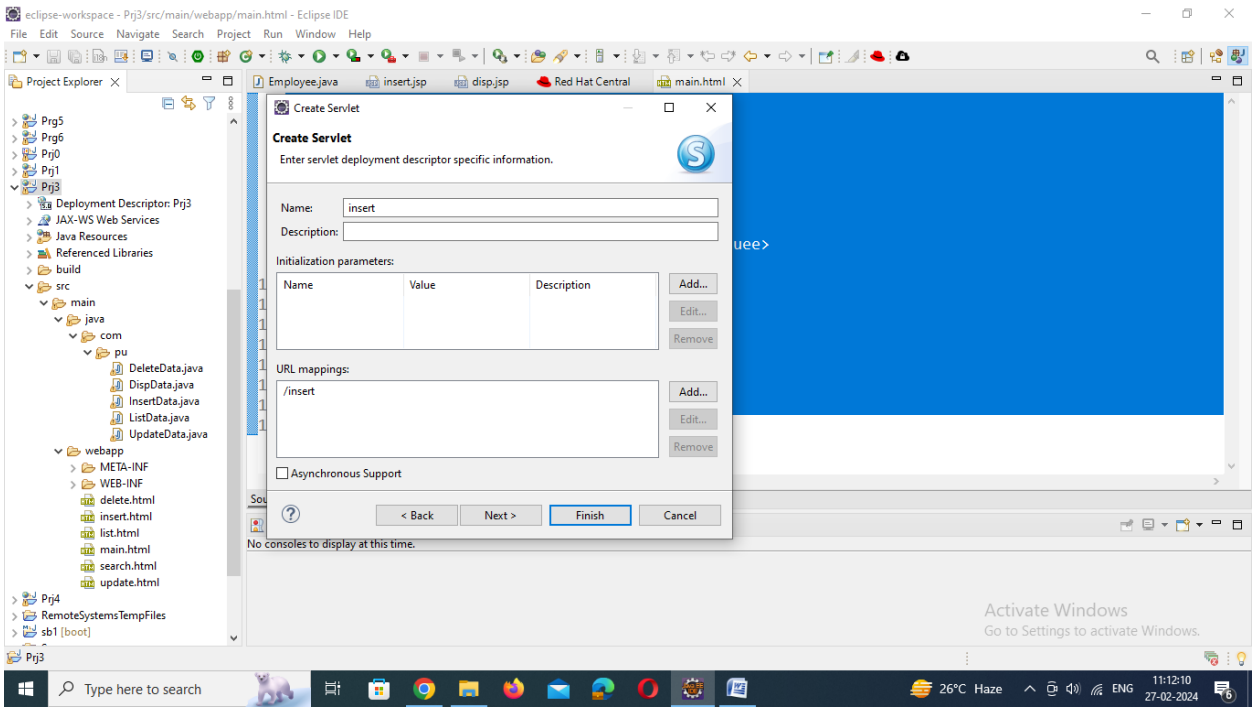
2.


New
Go Into
Show In
Copy
Copy Qualified Name
Paste
Delete
Build Path
Source
Refactor
Import
Export
Refresh
Close Project
Close Unrelated Projects
Source
Mark as Deployable
Show in Remote Systems view
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Team
Compare With
Configure
Source
Validate
Properties
Project...
Fuse Integration Project
Fuse Online Extension Project
Java Project
Dynamic Web Project
Enterprise Application Project
File
Folder
SQL File
Annotation
Camel Test Case
Camel XML File
Class
Enum
Fuse Message
Interface
Package
Record
Source Folder
CSS File
HTML File
JavaScript File
JSP File
XHTML Page
Filter
Listener
Servlet
Example...
Other...
Create a new Java/Jakarta Servlet, optionally adding it to a project's deployment descriptor.
Activate Windows
Go to Settings to activate Windows.
Type here to search
Near record
ENG
11:10:52
27-02-2024

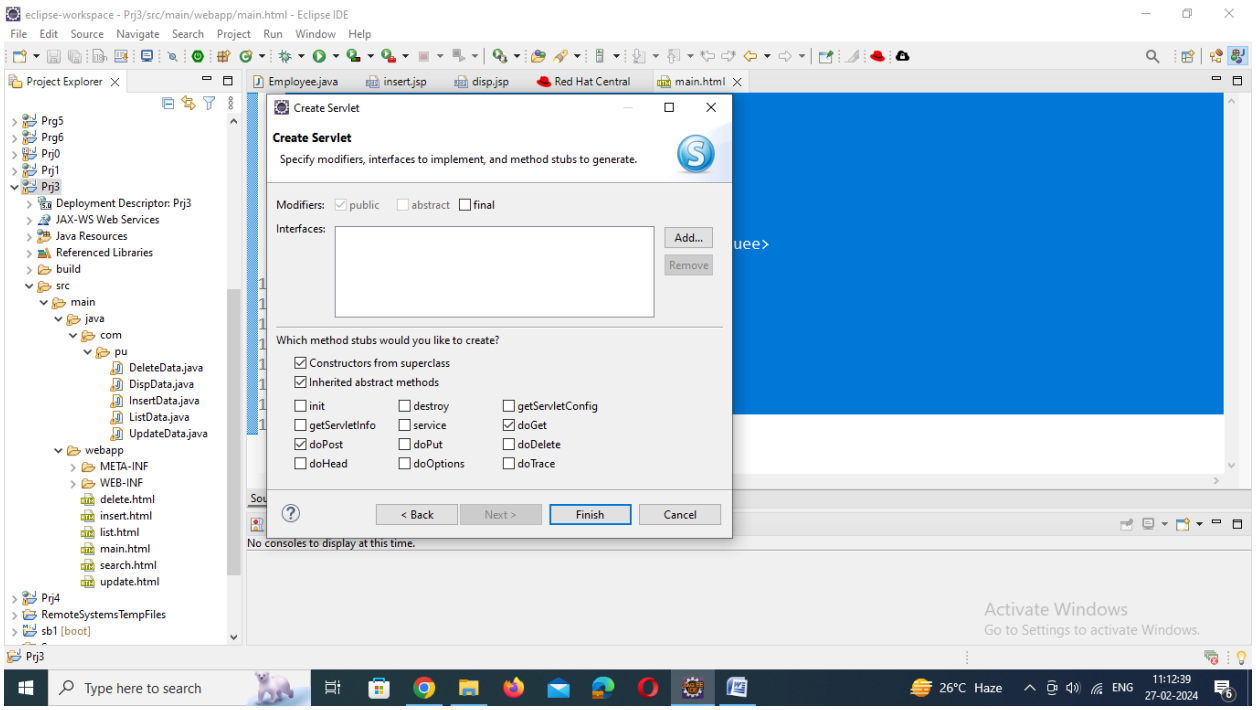
3.


eclipse-workspace - Prj3/src/main/webapp/main.html - Eclipse IDE
File Edit Source Navigate Search Project Run Window Help
Project Explorer
Prg5
Prg6
Prj0
Prj1
Prj3
Deployment Descriptor: Prj3
JAX-WS Web Services
Java Resources
Referenced Libraries
build
src
main
java
com
pu
DeleteData.java
DispData.java
InsertData.java
ListData.java
UpdateData.java
webapp
META-INF
WEB-INF
delete.html
insert.html
list.html
main.html
search.html
update.html
Prj4
RemoteSystemsTempFiles
sb1 [boot]
Employee.java
insert.jsp
disp.jsp
Red Hat Central
main.html
Create Servlet
Create Servlet
Warning: By convention, Java type names usually start with an uppercase letter
Project: Prj3
Source folder: /Prj3/src/main/java
Java package: com.pu
Class name: insert
Superclass: jakarta.servlet.http.HttpServlet
Use an existing Servlet class or JSP
Class name: insert
< Back
Next >
Finish
Cancel
Source
Problems
Console
Terminal
No consoles to display at this time.
Activate Windows
Go to Settings to activate Windows.
Type here to search
26°C Haze
ENG
11:11:33
27-02-2024

4.



5.



6.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
try{
    Class.forName("com.mysql.jdbc.Driver");
```

```
   Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/god?characterEncoding=latin1
","root","admin123");
   PreparedStatement st = con.prepareStatement("insert into mark values(?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?)");
    st.setInt(1, Integer.valueOf(request.getParameter("rollno")));
    st.setString(2, request.getParameter("name"));
    st.setString(3, request.getParameter("section"));
    st.setInt(4, Integer.valueOf(request.getParameter("sub1")));
    st.setInt(5, Integer.valueOf(request.getParameter("sub2")));
    st.setInt(6, Integer.valueOf(request.getParameter("sub3")));
    st.setInt(7, Integer.valueOf(request.getParameter("sub4")));
    st.setInt(8, Integer.valueOf(request.getParameter("sub5")));
    st.setInt(9, Integer.valueOf(request.getParameter("sub6")));
    st.setInt(10, Integer.valueOf(request.getParameter("lab1")));
    st.setInt(11, Integer.valueOf(request.getParameter("lab2")));
    st.executeUpdate();
    st.close();
    con.close();
    PrintWriter out = response.getWriter();
    out.println("<html><body><b>Successfully Inserted"
    + "</b></body></html>");
}catch(Exception e){
     System.out.println(e);}
}
}
```

Search the data in the database

search.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="./DispData">
Enter your Roll No:<input type="text" name="rollno"/><br/>
<input type="submit" value="search"/>
</form>
</body>
</html>
```

DispData.java

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      try{
        String rno=request.getParameter("rollno");
        int rn=Integer.valueOf(rno);
        Class.forName("com.mysql.jdbc.Driver");
        Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/god?characterEncoding=latin1
","root","admin123");
        PreparedStatement ps=con.prepareStatement("select * from mark where rollno=?");
        ps.setInt(1,rn);
        out.print("<table width=50% border=1>");
        out.print("<caption>Result:</caption>");
        ResultSet rs=ps.executeQuery();
       /* Printing column names */
        ResultSetMetaData rsmd=rs.getMetaData();
        int total=rsmd.getColumnCount();
        out.print("<tr>");
        for(int i=1;i<=total;i++)
        {
          out.print("<th>"+rsmd.getColumnName(i)+"</th>");
        }
        out.print("<th>Status</th></tr>");
       /* Printing result */
       while(rs.next())
       {
          out.print("<tr><td>"+rs.getInt(1)+"</td><td>"+rs.getString(2)+
"</td><td>"+rs.getString(3)+ rs.getInt(4)+"</td><td>" + rs.getInt(5)+"</td><td>" +
rs.getInt(6)+"</td><td>"+ rs.getInt(7)+"</td><td>" + rs.getInt(8)+"</td><td>"+
rs.getInt(9)+"</td><td>"+ rs.getInt(10)+"</td><td>"+rs.getInt(11)+"</td><td>");
if(rs.getInt(4)>=40 && rs.getInt(5)>=40 && rs.getInt(6)>=40 && rs.getInt(7)>=40 &&
rs.getInt(8)>=40 && rs.getInt(9)>=40 && rs.getInt(10)>=40 && rs.getInt(11)>=40)
out.print("<td>Pass</td>"+"</td></tr>");
else
out.print("<td>Fail</td>"+"</td></tr>");
}
out.print("</table>");
}catch (Exception e2) {e2.printStackTrace();}
finally{out.close();}
}
}
```

Delete record in the table.

delete.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="./DeleteData" method="post">
Enter your Roll No:<input type="text" name="rollno"/><br/>
  <input type="submit" value="delete"/>
</form>
</body>
</html>
```

DeleteData.java(Servlet program)

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
          // TODO Auto-generated method stub
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          try{
          String rno=request.getParameter("rollno");
          int rn=Integer.valueOf(rno);


          Class.forName("com.mysql.jdbc.Driver");
          Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306
/god?characterEncoding=latin1","root","admin123");

          PreparedStatement ps=con.prepareStatement("delete
from  mark where rollno=?");
          ps.setInt(1,rn);
          ps.executeUpdate();
          }catch (Exception e2) {e2.printStackTrace();}
```

```
            finally{out.close();}

            }
      }
```

List the records fron the table

list.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action="./ListData">


<input type="submit" value="List"/>
</form>
</body>
</html>
```

ListData.java(Servlet Program)

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
            // TODO Auto-generated method stub
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            try{


            Class.forName("com.mysql.jdbc.Driver");
            Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306
/god?characterEncoding=latin1","root","admin123");
```

```
          PreparedStatement ps=con.prepareStatement("select *
from mark");


          out.print("<table width=50% border=1>");
          out.print("<caption>Result:</caption>");

          ResultSet rs=ps.executeQuery();

          /* Printing column names */
          ResultSetMetaData rsmd=rs.getMetaData();
          int total=rsmd.getColumnCount();
          out.print("<tr>");
          for(int i=1;i<=total;i++)
          {
          out.print("<th>"+rsmd.getColumnName(i)+"</th>");
          }

          out.print("<th>Status</th></tr>");

          /* Printing result */

          while(rs.next())
          {

out.print("<tr><td>"+rs.getInt(1)+"</td><td>"+rs.getString(2
)+ "</td><td>"+rs.getString(3)+"</td><td>"+
rs.getInt(4)+"</td><td>" + rs.getInt(5)+"</td><td>" +
rs.getInt(6)+"</td><td>"+ rs.getInt(7)+"</td><td>" +
rs.getInt(8)+"</td><td>"+ rs.getInt(9)+"</td><td>"+
rs.getInt(10)+"</td><td>"+rs.getInt(11)+"</td><td>");
                if(rs.getInt(4)>=40 && rs.getInt(5)>=40 &&
rs.getInt(6)>=40 && rs.getInt(7)>=40 && rs.getInt(8)>=40 &&
rs.getInt(9)>=40 && rs.getInt(10)>=40 && rs.getInt(11)>=40)
                 out.print("Pass</td>"+"</tr>");
                else
                 out.print("Fail</td>"+"</tr>");
          }
```

```
            out.print("</table>");

            }catch (Exception e2) {e2.printStackTrace();}

            finally{out.close();}

            }
      }
```

Update record in the table

update.html

```
<!DOCTYPE html>
<html>
<head>

<title>Insert title here</title>
</head>
<body>
Enter the update data details
<form action="./UpdateData" >
     <table>
        <tr><td>Enter Roll No:</td>
        <td><input type="text" name="rollno"/></td>
        </tr>
        <tr><td>Enter Name:</td>
        <td><input type="text" name="name"/></td>
        </tr>
        <tr><td>Enter Section:</td>
        <td><select name="section">
            <option>A</option>
            <option>B</option>
            <option>C</option>
            <option>D</option>
        </select>
        </td>
        </tr>
        <tr><td>Enter Subjet1 Mark:</td>
        <td><input type="text" name="sub1"/></td>
        </tr>
```

```
        <tr><td>Enter Subject2 Mark:</td>
        <td><input type="text" name="sub2"/></td>
        </tr>
        <tr><td>Enter Subjet3 Mark:</td>
        <td><input type="text" name="sub3"/></td>
        </tr>
        <tr><td>Enter Subject4 Mark:</td>
        <td><input type="text" name="sub4"/></td>
        </tr>
        <tr><td>Enter Subject5 Mark:</td>
        <td><input type="text" name="sub5"/></td>
        </tr>
        <tr><td>Enter Subject6 Mark:</td>
        <td><input type="text" name="sub6"/></td>
        </tr>
        <tr><td>Enter Lab1 Mark:</td>
        <td><input type="text" name="lab1"/></td>
        </tr>
        <tr><td>Enter Lab2 Mark:</td>
        <td><input type="text" name="lab2"/></td>
        </tr>
       <tr><td><input type="submit"/></td></tr>
      </table>
    </form>
</body>
</html>
```

UpdateData.java(Servlet)

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        try{
        String rno=request.getParameter("rollno");
        int rn=Integer.valueOf(rno);


        Class.forName("com.mysql.jdbc.Driver");
```

```
          Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306
/god?characterEncoding=latin1","root","admin123");

          PreparedStatement st=con.prepareStatement("update
mark set
name=?,section=?,s1=?,s2=?,s3=?,s4=?,s5=?,s6=?,l1=?,l2=?
where rollno=?");

          st.setString(1, request.getParameter("name"));
          st.setString(2, request.getParameter("section"));
          st.setInt(3,
Integer.valueOf(request.getParameter("sub1")));
          st.setInt(4,
Integer.valueOf(request.getParameter("sub2")));
          st.setInt(5,
Integer.valueOf(request.getParameter("sub3")));
          st.setInt(6,
Integer.valueOf(request.getParameter("sub4")));
          st.setInt(7,
Integer.valueOf(request.getParameter("sub5")));
          st.setInt(8,
Integer.valueOf(request.getParameter("sub6")));
          st.setInt(9,
Integer.valueOf(request.getParameter("lab1")));
          st.setInt(10,
Integer.valueOf(request.getParameter("lab2")));
          st.setInt(11,
Integer.valueOf(request.getParameter("rollno")));
          st.executeUpdate();


        out.println("<html><body><b>Successfully Updated"
                        + "</b></body></html>");
        st.close();
          con.close();
     }catch(Exception e){ System.out.println(e);}
```

```
        finally{out.close();}

        }


    }
```

RESULT:

Thus the dynamic web application has been created and process the exam marks with servlet programs.

Expt 6:

AIM:

Build web application to connect with a database using JSP that perform database manipulations.

ALGORITHM

Step 1 : In MySQL database server to create the table by
Create table employees(eno int,name varchar(30),gender varchar(1),dept varchar(10),salary float(10,2));
Step 2: To open Eclipse IDE and create a Dynamic Web Project.
Step 2: To create a Insert JSP file to accept the user input of employee details in a form. After entered the data the information will be store into the employees table.
Step 6: To create a JSP Disp program to process the user request and retrieve employees from the employess table in a database.

PROGRAM:

Insert.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="#ffffcc">
<font size="+3" color="green"><br>Welcome to Presidency University !</font>
<FORM action="insert.jsp" method="get">
<TABLE style="background-color: #ECE5B6;" WIDTH="30%" >
<TR>
<TH width="50%">Employee No:</TH>
<TD width="50%"><INPUT TYPE="text" NAME="eno"></TD>
</tr>
<TR>
```

```
<TH width="50%">Employee Name</TH>
<TD width="50%"><INPUT TYPE="text" NAME="name"></TD>
</tr>
<tr>
<TH width="50%">Employee Gender</TH>
<TD width="50%"><INPUT TYPE="text" NAME="gender"></TD>
</tr>
<tr>
<TH width="50%">Employee Department</TH>
<TD width="50%"><INPUT TYPE="text" NAME="dept"></TD>
</tr>
<tr>
<TH width="50%">Employee Salary</TH>
<TD width="50%"><INPUT TYPE="text" NAME="salary"></TD>
</tr>
<TR>
<TH></TH>
<TD width="50%"><INPUT TYPE="submit" VALUE="submit"></TD>
</tr>
</TABLE>
<%
int uq=0;
try {
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/god?characterEncoding=latin1
","root","admin123");
PreparedStatement pstatement = con.prepareStatement("INSERT INTO EMPLOYEES
VALUES(?, ?, ?, ?, ?)");
pstatement.setInt(1, Integer.valueOf(request.getParameter("eno")));
pstatement.setString(2, request.getParameter("name"));
pstatement.setString(3, request.getParameter("gender"));
pstatement.setString(4, request.getParameter("dept"));
pstatement.setFloat(5, Integer.valueOf(request.getParameter("salary")));
uq=pstatement.executeUpdate();
pstatement.close();
con.close();
}
catch(Exception ex) {
//out.println("Unable to connect to database.");
}
if (uq != 0) {
%>
<br>
<TABLE style="background-color: #E3E4FA;"
WIDTH="30%" border="1">
```

```
<tr><th>Data is inserted successfully in database.</th></tr>
</table>
<%
} %>
</form>
</body>
</html>
```

Disp.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2>Employee Details</h2>
<%
try {
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/god?characterEncoding=latin1
","root","admin123");
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select * from employees");
%>
<TABLE cellpadding="15" border="1" style="background-color: #ffffcc;">
<tr><th>E.No</th><th>Name</th><th>Gender</th><th>Department</th><th>Salary</th><tr>
<%
while (rs.next()) {
%>
<TR>
<TD><%=rs.getInt(1)%></TD>
<TD><%=rs.getString(2)%></TD>
<TD><%=rs.getString(3)%></TD>
<TD><%=rs.getString(4)%></TD>
<TD><%=rs.getFloat(5)%></TD>
</TR>
<% } %>
<%
// close all the connections.
```

```
rs.close();
st.close();
con.close();
} catch (Exception ex) {
%>
<font size="+3" color="red"></b>
<%
out.println("Unable to connect to database.");
}
%>
</TABLE><TABLE>
<TR>
<TD><FORM ACTION="disp.jsp" method="get" >
<button type="submit"><-- back</button></TD>
</TR>
</TABLE>
</font>
</body>
</html>
```

RESULT:

Thus the dynamic web application has been created and process the employees data with JSP programs.

Expt 7:

Construct a login application in respecting the MVC model.

AIM:

To demonstrate the web development by using MVC design pattern.

ALGORITHM

Step 1 : Create a LoginBean class acting as a model which defines the business logic of the system and also represents the state of the application.

Step 2: Create index.jsp will be a view class which can accept the user login credentials.

Step 3: Create ControllerServlet.java *Class* which acts as a controller is like an interface between Model and View. It receives the user requests from the view layer and processes them, including the necessary validations. The requests are then sent to model for data processing. Once they are processed, the data is again sent back to the controller and then displayed on the view.

Step 4: Create login-success.jsp and login-error.jsp pages represents the output of the application or the user interface. It displays the data fetched from the model layer by the controller and presents the data to the user whenever asked for. It receives all the information it needs from the controller.

PROGRAM:

index.jsp(View)

```
<form action="ControllerServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
```

```
<input type="submit" value="login">
</form>
```
LoginBean.java(Model)
```
package com.pu;
import java.lang.*;
public class LoginBean {
private String name,password;
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;
}
public boolean validate(){
if(password.equals("admin")){
return true;
}
else{
return false;
}
}
}
```
login-success.jsp(view)
```
<%@page import="com.pu.LoginBean"%>
<p>You are successfully logged in!</p>
<%
LoginBean bean=(LoginBean)request.getAttribute("bean");
out.print("Welcome, "+bean.getName());
%>
```
login-error.jsp(View)
```
<p>Sorry! username or password error</p>
<%@ include file="index.jsp" %>
```
ControllerServlet.java(Controller)
```

response.setContentType("text/html");
PrintWriter out=response.getWriter();
String name=request.getParameter("name");
String password=request.getParameter("password");
LoginBean bean=new LoginBean();
bean.setName(name);
```

```
bean.setPassword(password);
request.setAttribute("bean",bean);
boolean status=bean.validate();
if(status){
RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
rd.forward(request,response);
}
else{
RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
rd.forward(request,response);
}
```

RESULT:

Thus the dynamic web pages were developed and demonstrated by using MVC design pattern.

Expt8:

Implement a web application based on the MVC design pattern, to create an **Employee Registration** module using **JSP**, **Servlet**, **JDBC**, and **MySQL** database.

AIM:

To implement a web application based on the MVC design pattern with data base.

ALGORITHM

Step 1 : create a database named "employee" in MySQL. Let's create an *employee1* table using below DDL script:

```
CREATETABLEemployee1
(idint,first_namevarchar(20),last_namevarchar(20),usernamevarchar(250),passwordvarchar(20)
,addressvarchar(45),contactvarchar(45));
```

Step 2: Create the Employee class which will act as our **Model** class.

Step 3: *EmployeeDao* class that contains JDBC code to connect with the MySQL database.

Step 4: create an *EmployeeServlet* class to process HTTP request parameters and redirect to the appropriate JSP page after request data stored in the database:

Step 5: design an employee registration HTML form employeeregister.jsp

Step 6:After an employee successfully registered then employeedetails.jsp show a successful message on screen

---

PROGRAM:

Employee.java

```
package com.pu;
```

```
import java.io.*;

publicclass Employee {
	privateintid;
	private String firstName;
	private String lastName;
	private String username;
	private String password;
	private String address;
	private String contact;

	public int getId() {
		return  id;
    }
	public void setId(intid) {
		this.id = id;
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}
	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 getAddress() {
		return address;
  }
	public void setAddress(String address) {
```

```
		this.address = address;
	}
	public String getContact() {
		return contact;
   }
	public void setContact(String contact) {
		this.contact = contact;
   }
}
```

EmployeeServlet.java

```
package com.pu;

import java.sql.*;
import jakarta.servlet.*;
import java.io.*;

publicclass EmployeeServlet extends HttpServlet {

	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		int id = Integer.valueOf(request.getParameter("id"));
		String fn=request.getParameter("firstName");
		String ln=request.getParameter("lastName");
		String un=request.getParameter("username");
		String pwd=request.getParameter("password");
		String addr=request.getParameter("address");
		String cont=request.getParameter("contact");
		Employee emp = new Employee();
		emp.setId(id);
		emp.setFirstName(fn);
		emp.setLastName(ln);
		emp.setUsername(un);
		emp.setPassword(pwd);
		emp.setAddress(addr);
		emp.setContact(cont);
		request.setAttribute("emp",emp);
		int result = 0;
	try{
		Class.forName("com.mysql.cj.jdbc.Driver");
```

```
            Connection connection =
        DriverManager.getConnection("jdbc:mysql://localhost:3306/god","root","admin123");
            String query = "INSERT INTO employee1 (id, first_name, last_name, username,
        password, address, contact) VALUES (?, ?, ?, ?, ?,?,?);";
            PreparedStatement preparedStatement = connection.prepareStatement(query);
        preparedStatement.setInt(1, emp.getId());
        preparedStatement.setString(2, emp.getFirstName());
        preparedStatement.setString(3, emp.getLastName());
        preparedStatement.setString(4, emp.getUsername());
        preparedStatement.setString(5, emp.getPassword());
        preparedStatement.setString(6, emp.getAddress());
        preparedStatement.setString(7, emp.getContact());
            System.out.println(preparedStatement);
        result = preparedStatement.executeUpdate();
            RequestDispatcher rd=request.getRequestDispatcher("employeedetail.jsp");
        rd.forward(request,response);

        } catch (Exception e) {
            System.err.println(e);
        }
            }

        }
```

employeeregister.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPEhtml>
<html>
<head>
<metacharset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Employee Register Form</h1>
<formaction="EmployeeServlet"method="post">
<tablestyle="with: 50%">
<tr>
<td>Registration Id</td>
<td><inputtype="text"name="id"/></td>
</tr>
<tr>
<td>First Name</td>
<td><inputtype="text"name="firstName"/></td>
```

```
</tr>
<tr>
<td>Last Name</td>
<td><inputtype="text"name="lastName"/></td>
</tr>
<tr>
<td>UserName</td>
<td><inputtype="text"name="username"/></td>
</tr>
<tr>
<td>Password</td>
<td><inputtype="password"name="password"/></td>
</tr>
<tr>
<td>Address</td>
<td><inputtype="text"name="address"/></td>
</tr>
<tr>
<td>Contact No</td>
<td><inputtype="text"name="contact"/></td>
</tr>
</table>
<inputtype="submit"value="Submit"/></form>
</body>
</html>
```

employeedetail.jsp

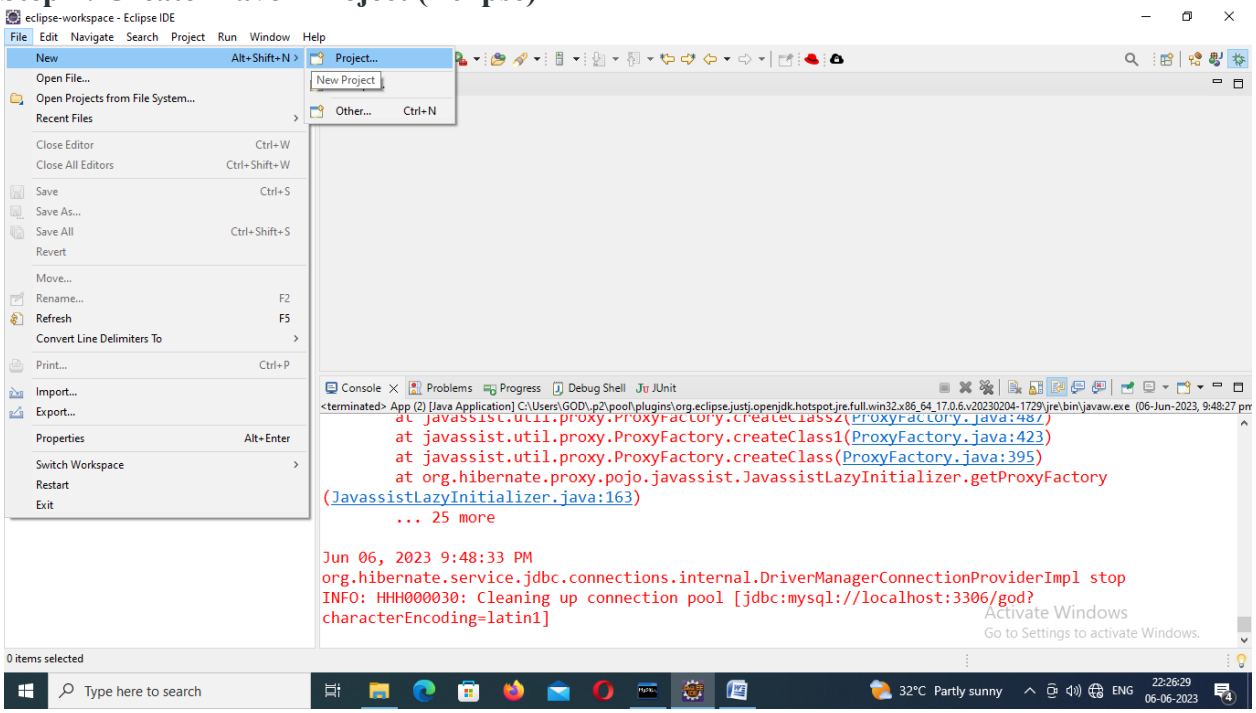
```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@pageimport="com.pu.*"%>
<!DOCTYPEhtml>
<html>
<head>
<metacharset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%
  out.print("You are successfully registered");
%>
</body>
</html>
```

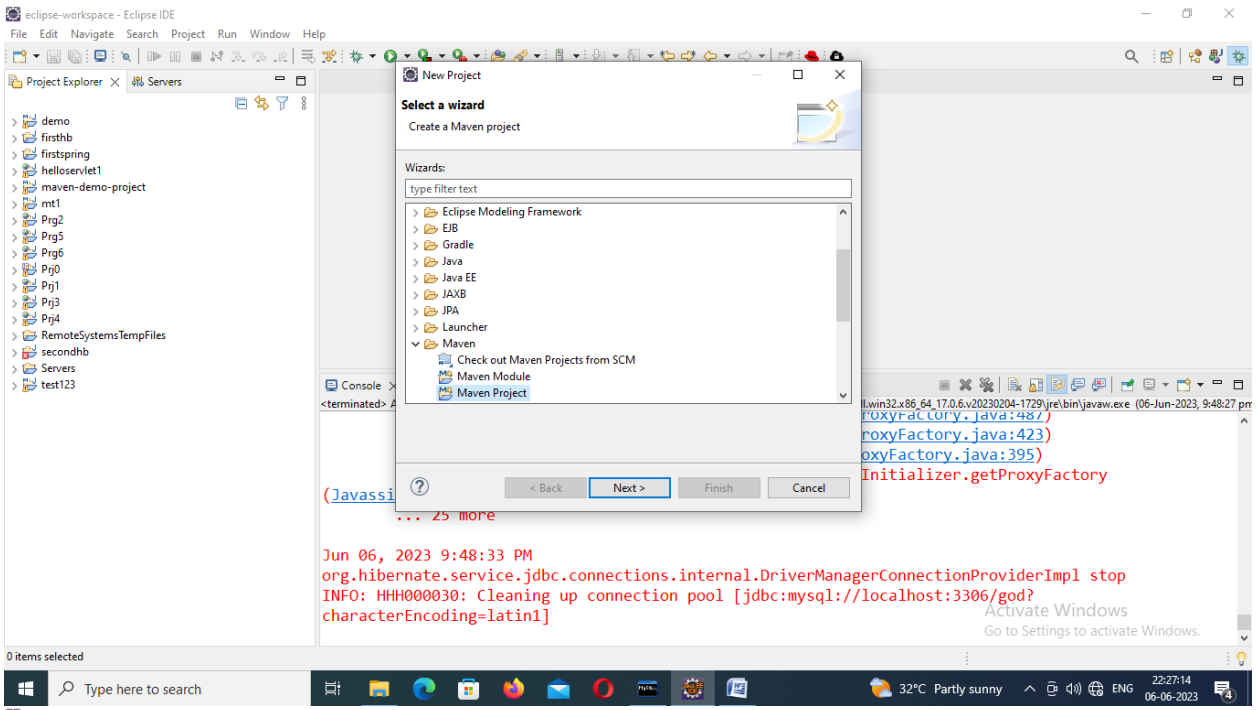
RESULT:

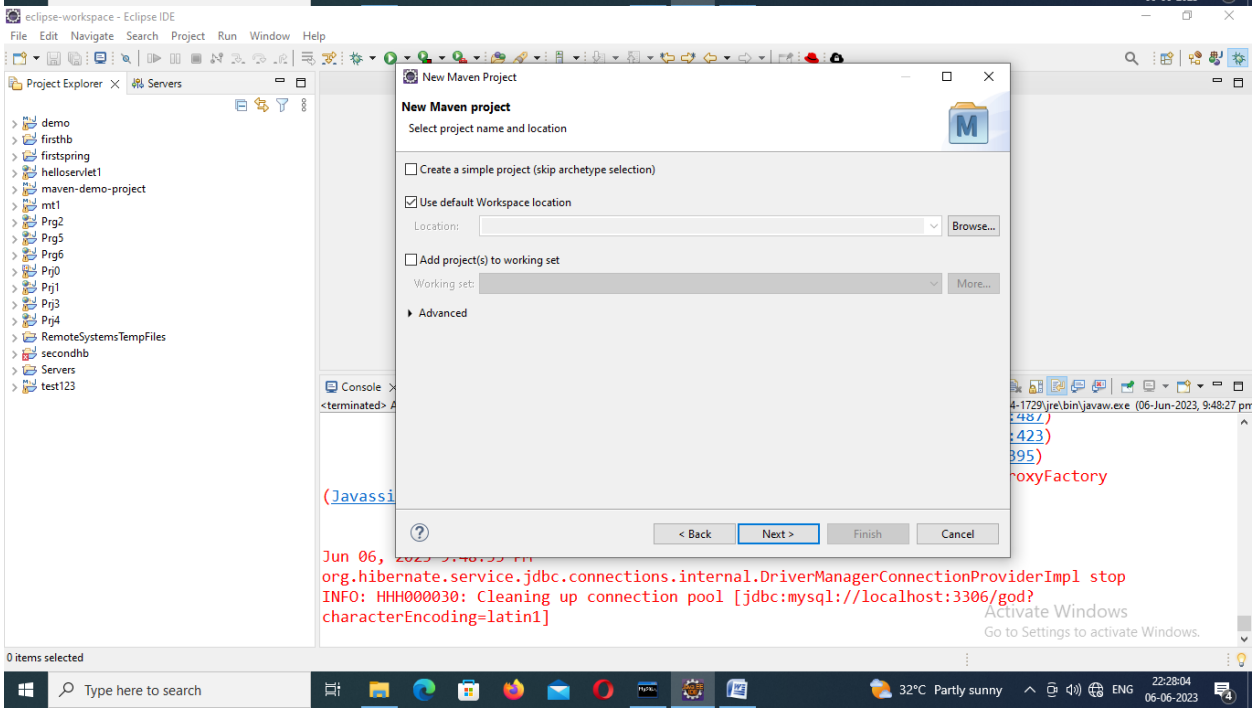
Thus, to implement a simple MVC web application using JSP and Servlet.

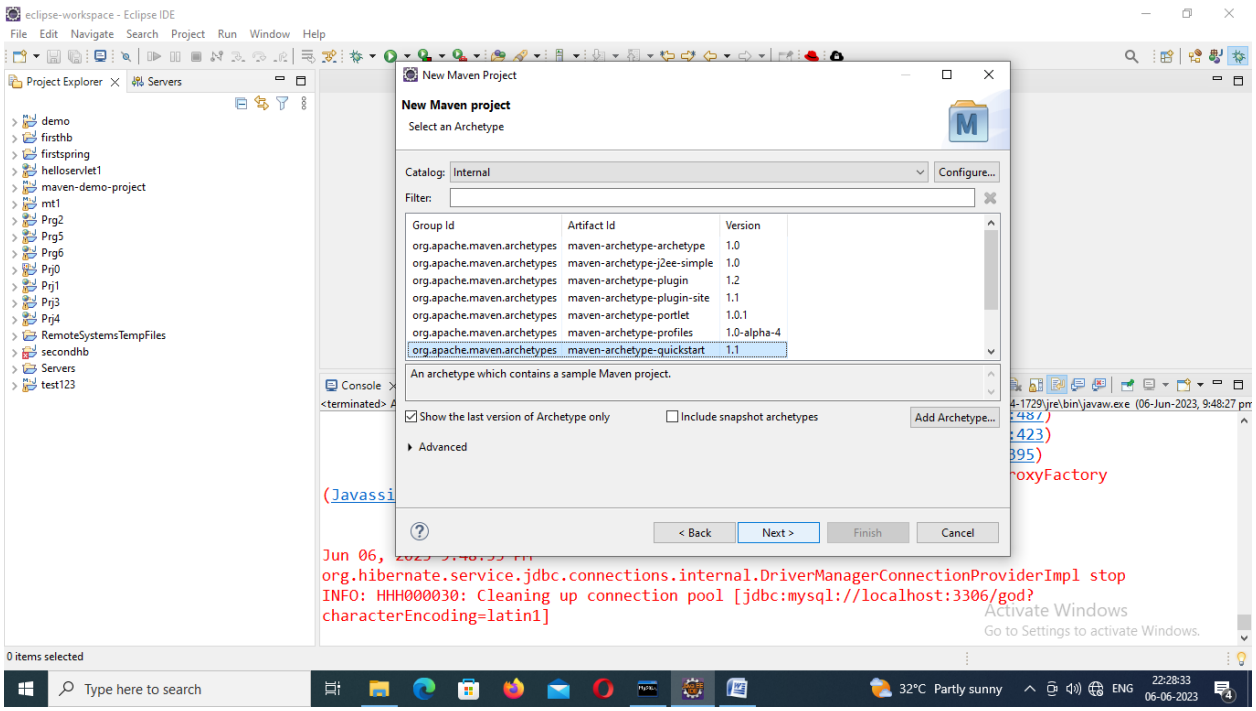
Expt9:
Create Student mark processing project using Hibernate with Maven.

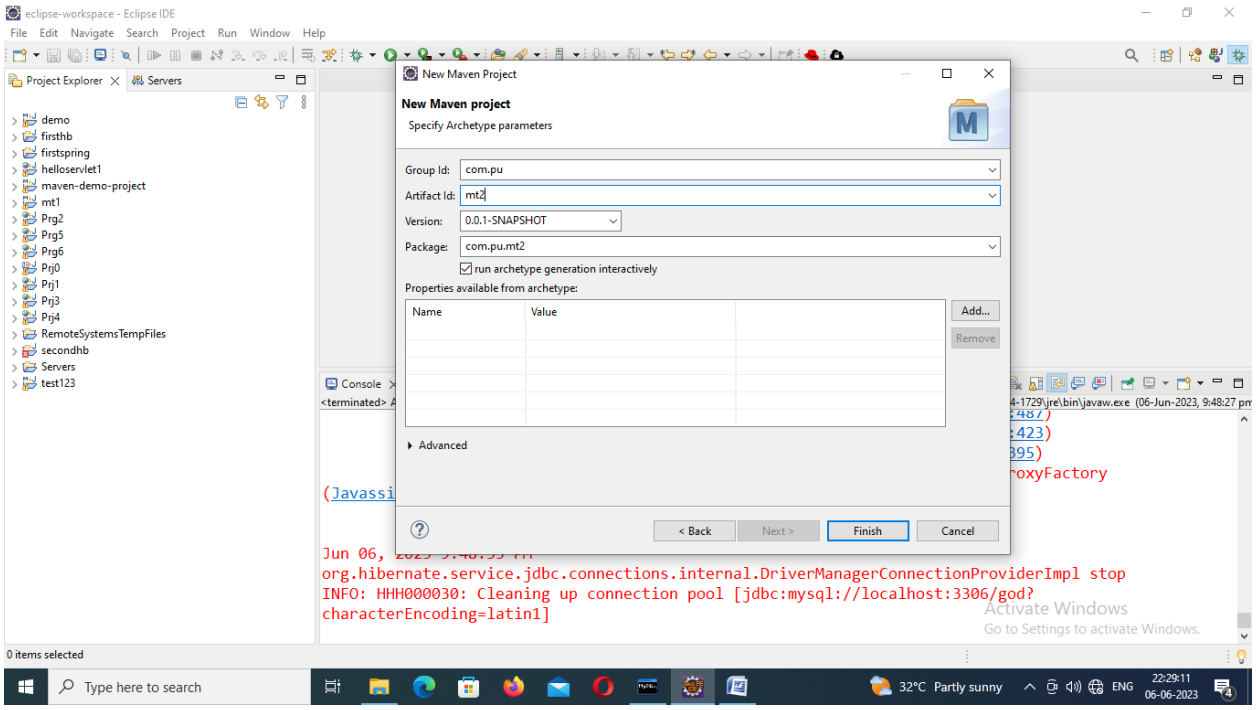
AIM:
To create Hibernate 6 Application and connect with the MySQL database bye using Maven as a dependency management tool.
PROCEDURE:
Step 1: Create Maven Project (Eclipse)

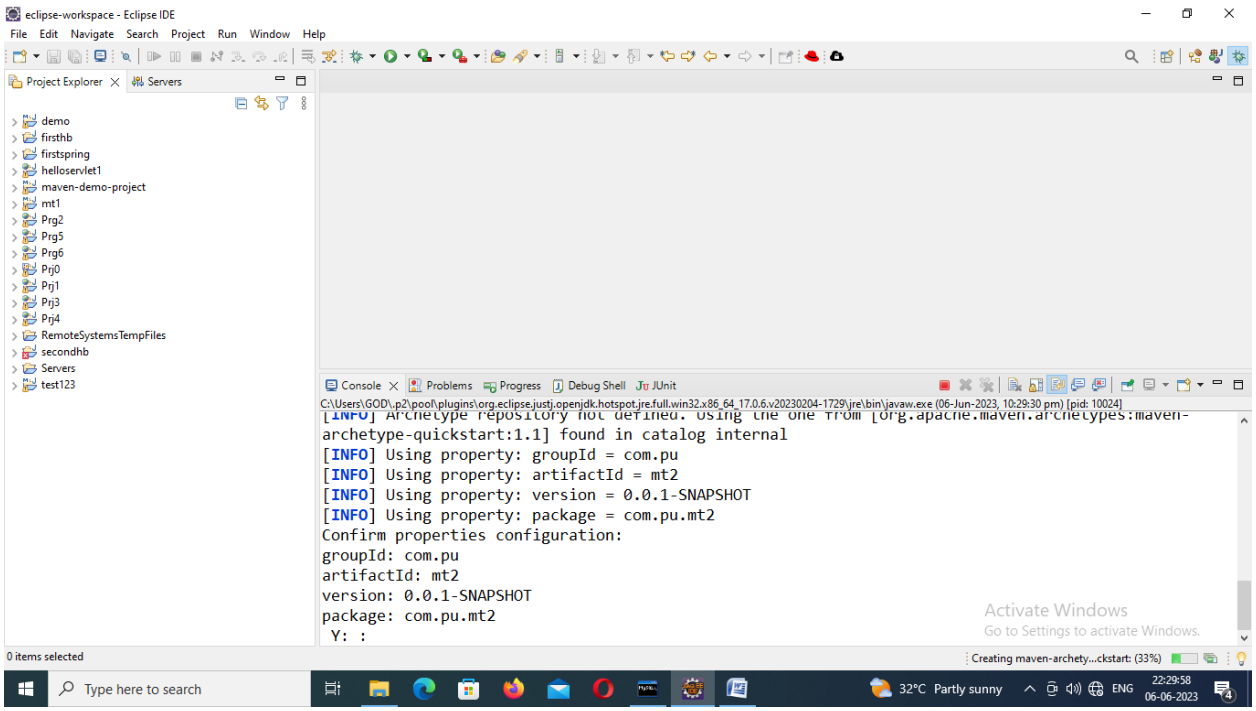



eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
Servers
demo
firsthb
firstspring
helloservlet1
maven-demo-project
mt1
Prg2
Prg5
Prg6
Prj0
Prj1
Prj3
Prj4
RemoteSystemsTempFiles
secondhb
Servers
test123
New Project
Select a wizard
Create a Maven project
Wizards:
type filter text
Eclipse Modeling Framework
EJB
Gradle
Java
Java EE
JAXB
JPA
Launcher
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
< Back
Next >
Finish
Cancel
Console
<terminated>
... 25 more
Jun 06, 2023 9:48:33 PM
org.hibernate.service.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/god?
characterEncoding=latin1]
Activate Windows
Go to Settings to activate Windows.
0 items selected
Type here to search
32°C Partly sunny
ENG
22:27:14
06-06-2023

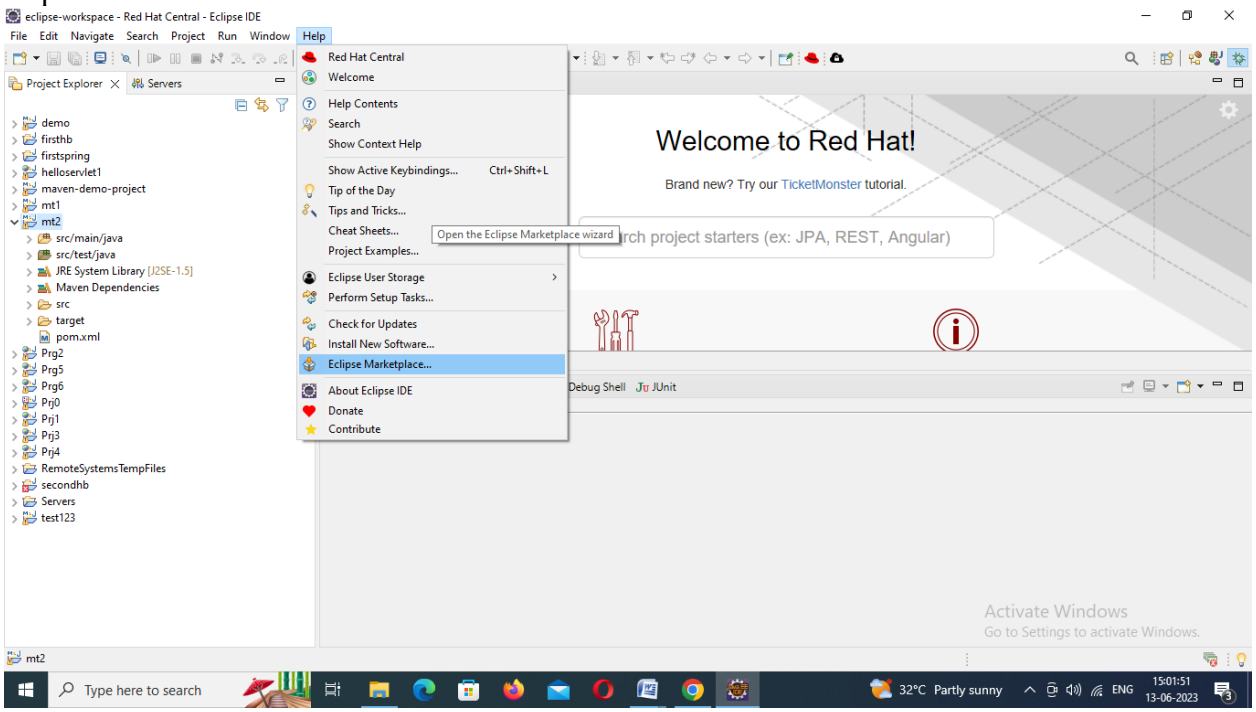

eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
Servers
demo
firsthb
firstspring
helloservlet1
maven-demo-project
mt1
Prg2
Prg5
Prg6
Prj0
Prj1
Prj3
Prj4
RemoteSystemsTempFiles
secondhb
Servers
test123
New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel
Console
<terminated>
Jun 06,
org.hibernate.service.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/god?
characterEncoding=latin1]
Activate Windows
Go to Settings to activate Windows.
0 items selected
Type here to search
32°C Partly sunny
ENG
22:28:04
06-06-2023

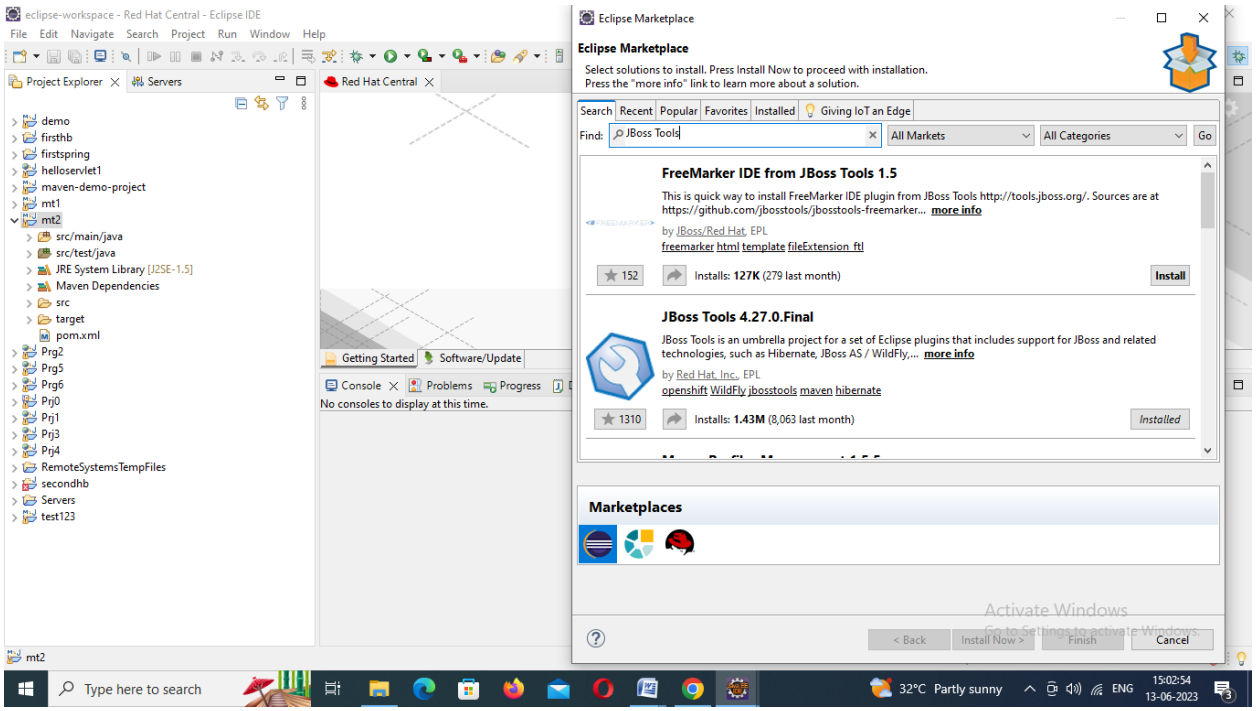

eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
Servers
demo
firsthb
firstspring
helloservlet1
maven-demo-project
mt1
Prg2
Prg5
Prg6
Prj0
Prj1
Prj3
Prj4
RemoteSystemsTempFiles
secondhb
Servers
test123
New Maven Project
New Maven project
Select an Archetype
Catalog: Internal
Configure...
Filter:
Group Id
Artifact Id
Version
org.apache.maven.archetypes maven-archetype-archetype 1.0
org.apache.maven.archetypes maven-archetype-j2ee-simple 1.0
org.apache.maven.archetypes maven-archetype-plugin 1.2
org.apache.maven.archetypes maven-archetype-plugin-site 1.1
org.apache.maven.archetypes maven-archetype-portlet 1.0.1
org.apache.maven.archetypes maven-archetype-profiles 1.0-alpha-4
org.apache.maven.archetypes maven-archetype-quickstart 1.1
An archetype which contains a sample Maven project.
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Finish
Cancel
Console
org.hibernate.service.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/god?
characterEncoding=latin1]
Activate Windows
Go to Settings to activate Windows.
0 items selected
Type here to search
32°C Partly sunny
ENG
22:28:33
06-06-2023


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
Servers
demo
firsthb
firstspring
helloservlet1
maven-demo-project
mt1
Prg2
Prg5
Prg6
Prj0
Prj1
Prj3
Prj4
RemoteSystemsTempFiles
secondhb
Servers
test123
New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.pu
Artifact Id: mt2
Version: 0.0.1-SNAPSHOT
Package: com.pu.mt2
run archetype generation interactively
Properties available from archetype:
Name
Value
Add...
Remove
Advanced
< Back
Next >
Finish
Cancel
Console
org.hibernate.service.jdbc.connections.internal.DriverManagerConnectionProviderImpl stop
INFO: HHH000030: Cleaning up connection pool [jdbc:mysql://localhost:3306/god?
characterEncoding=latin1]
Activate Windows
Go to Settings to activate Windows.
0 items selected
Type here to search
32°C Partly sunny
ENG
22:29:11
06-06-2023



Step 2: Install JBoss Tools

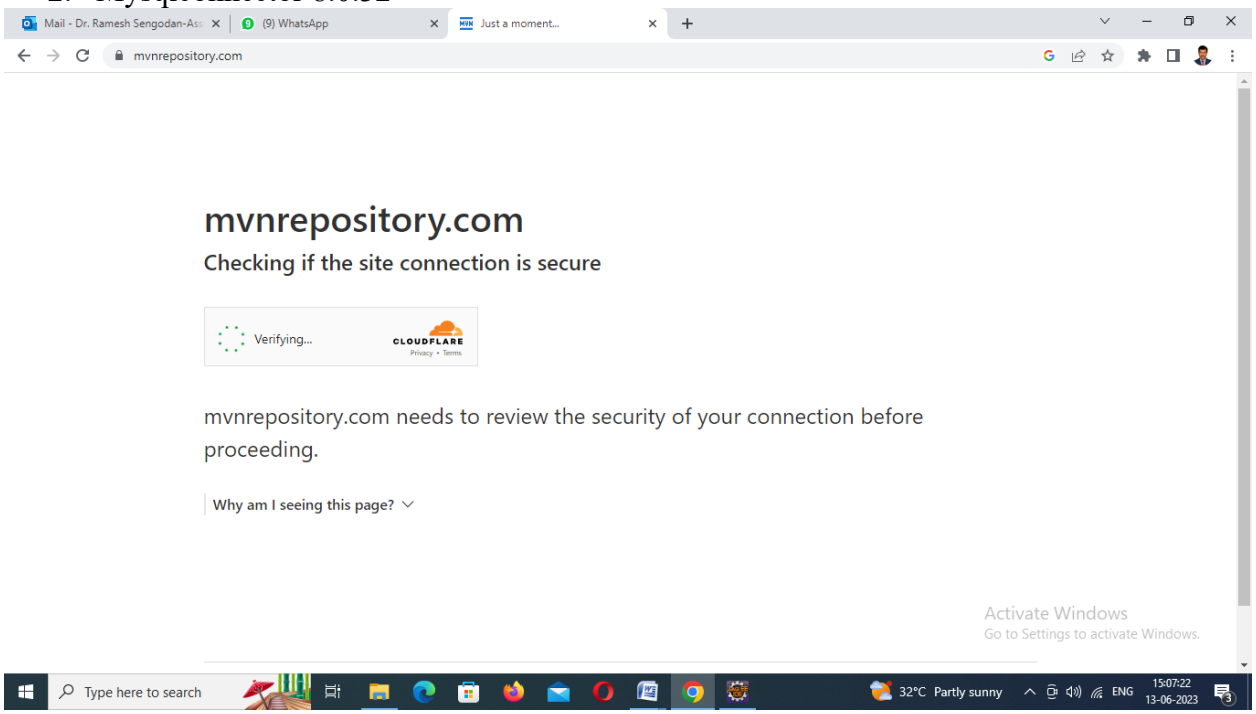


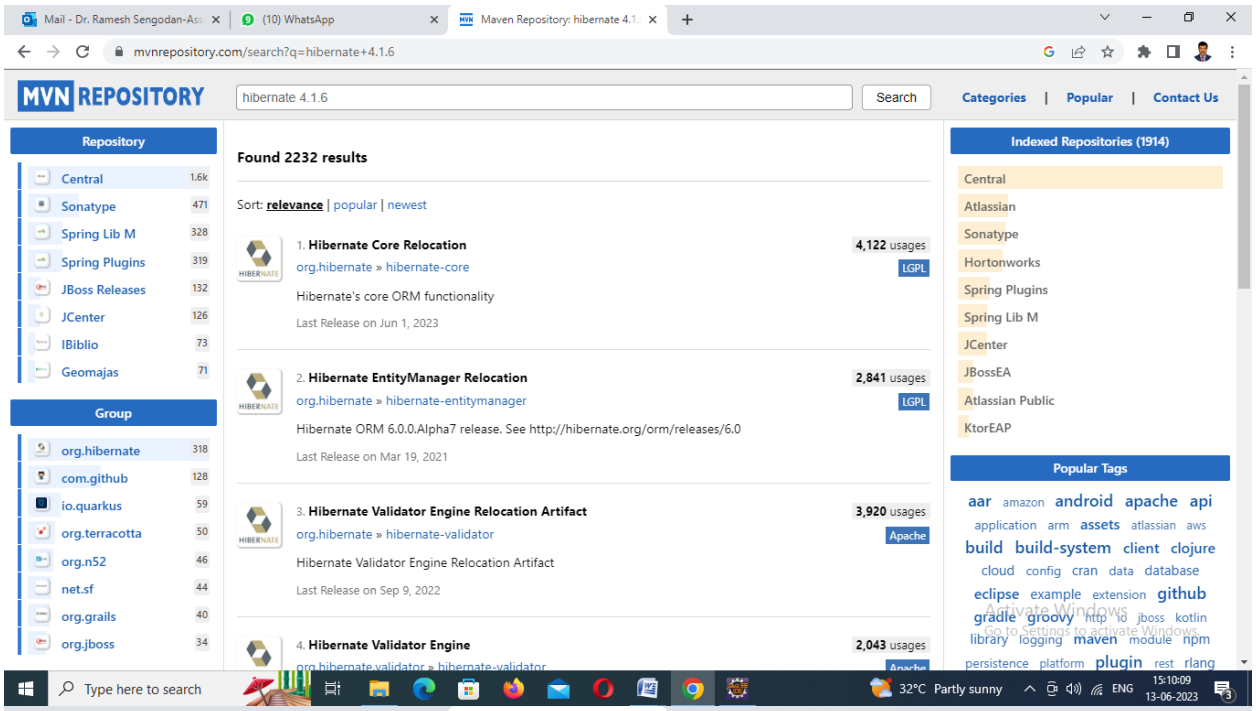


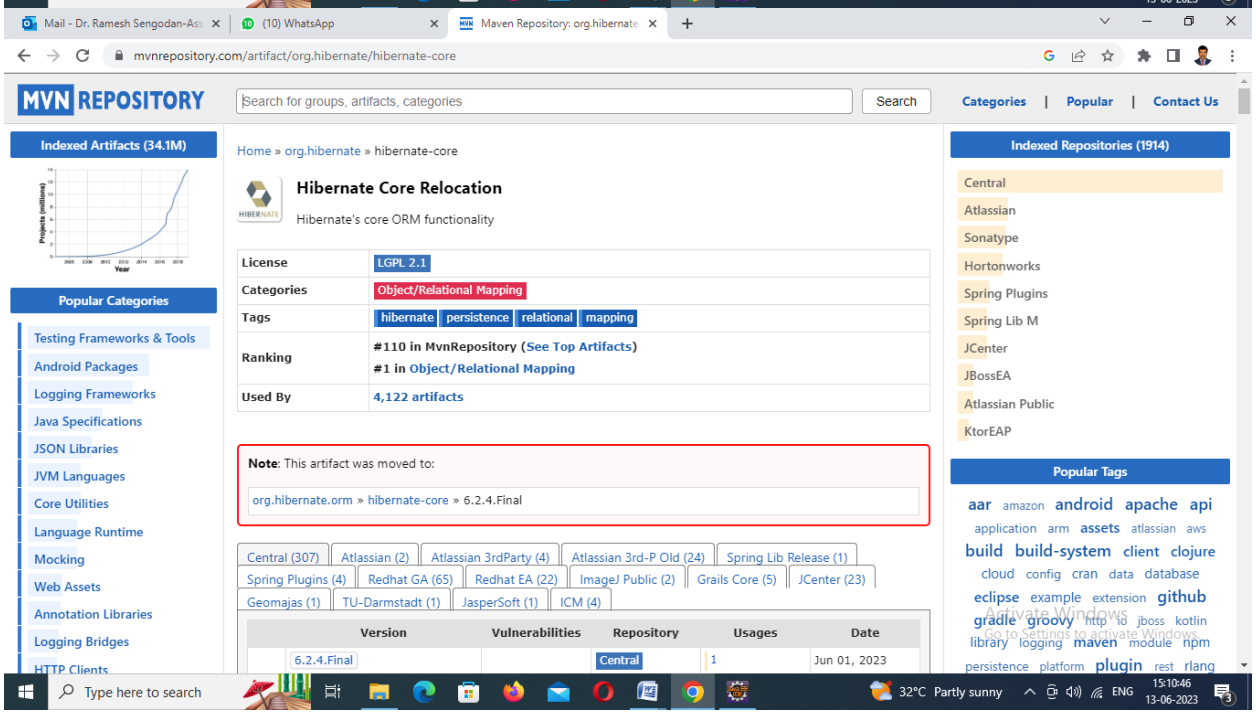
Step 3: Add the dependency to the pom.xml file

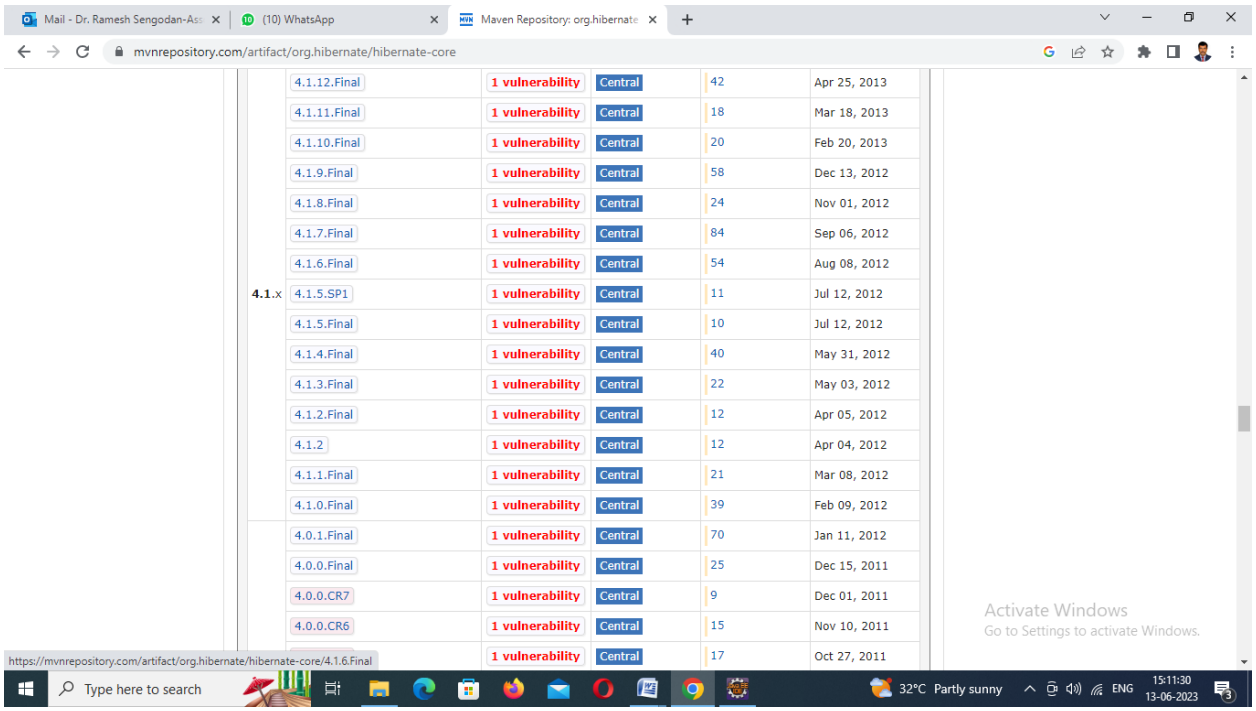
Include following dependencies available in mvnrepository.com in pom.xml file

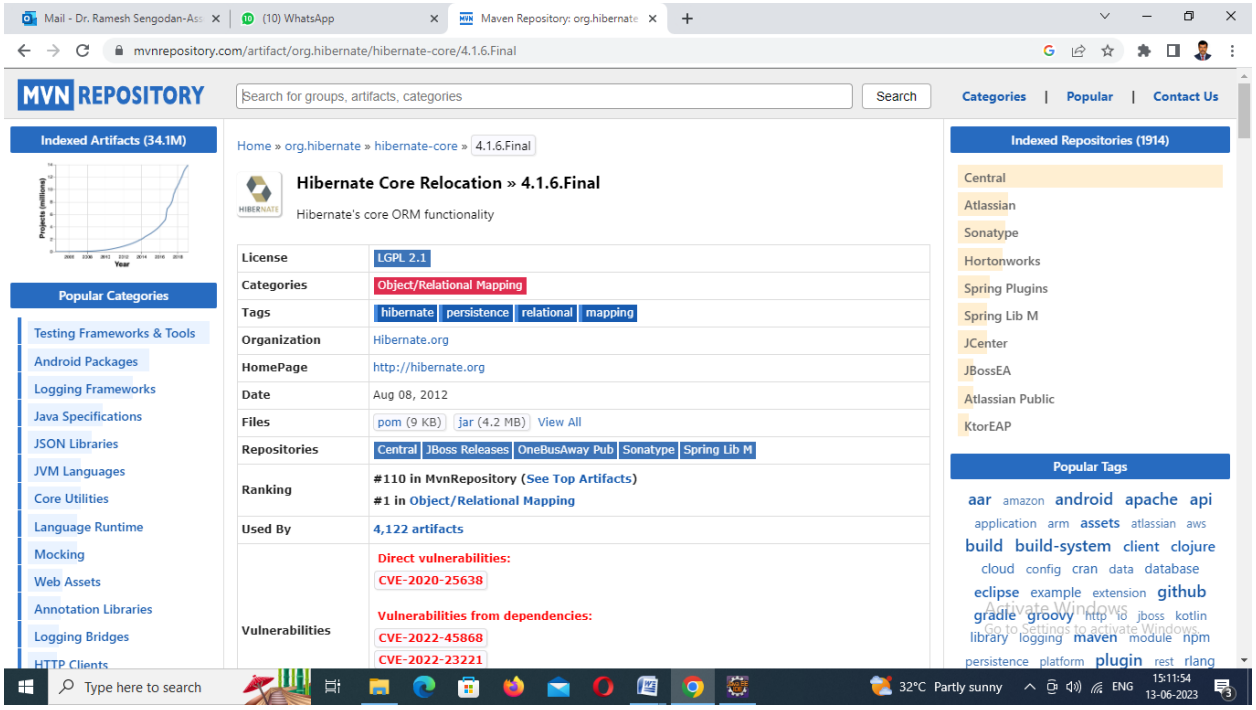
1. Hibernate core 4.1.6
2. Mysqlconnector 8.0.32

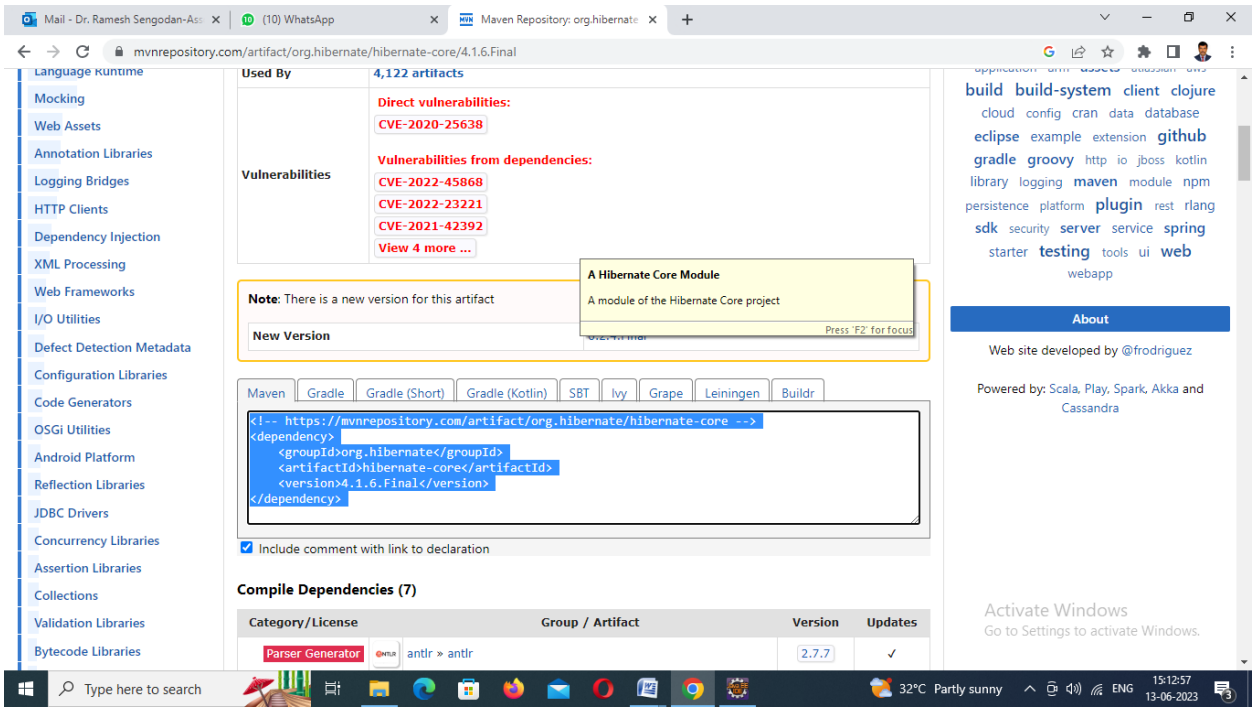


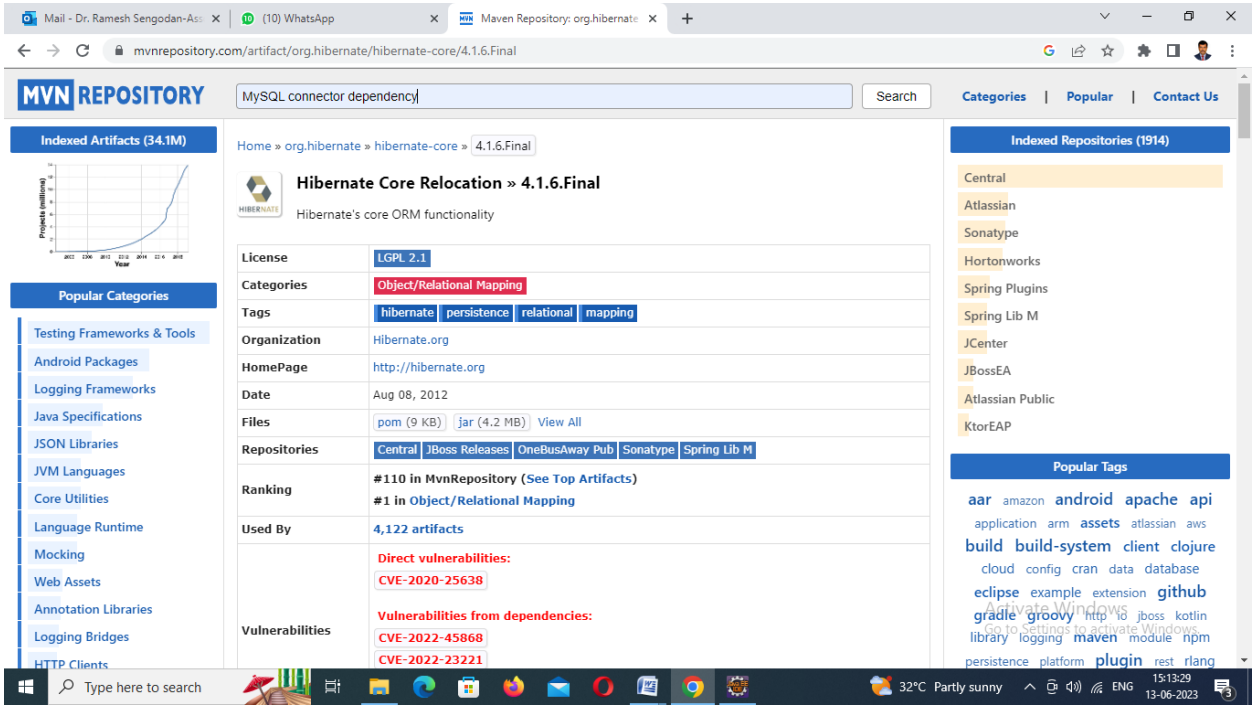

Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: hibernate 4.1
mvnrepository.com/search?q=hibernate+4.1.6
MVN REPOSITORY
hibernate 4.1.6
Search
Categories | Popular | Contact Us
Repository
Central 1.6k
Sonatype 471
Spring Lib M 328
Spring Plugins 319
JBoss Releases 132
JCenter 126
IBiblio 73
Geomajas 71
Group
org.hibernate 318
com.github 128
io.quarkus 59
org.terracotta 50
org.n52 46
net.sf 44
org.grails 40
org.jboss 34
Found 2232 results
Sort: relevance | popular | newest
1. Hibernate Core Relocation
4,122 usages
org.hibernate » hibernate-core
LGPL
Hibernate's core ORM functionality
Last Release on Jun 1, 2023
2. Hibernate EntityManager Relocation
2,841 usages
org.hibernate » hibernate-entitymanager
LGPL
Hibernate ORM 6.0.0.Alpha7 release. See http://hibernate.org/orm/releases/6.0
Last Release on Mar 19, 2021
3. Hibernate Validator Engine Relocation Artifact
3,920 usages
org.hibernate » hibernate-validator
Apache
Hibernate Validator Engine Relocation Artifact
Last Release on Sep 9, 2022
4. Hibernate Validator Engine
2,043 usages
Indexed Repositories (1914)
Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP
Popular Tags
aar amazon android apache api
application arm assets atlassian aws
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http io jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:10:09
13-06-2023

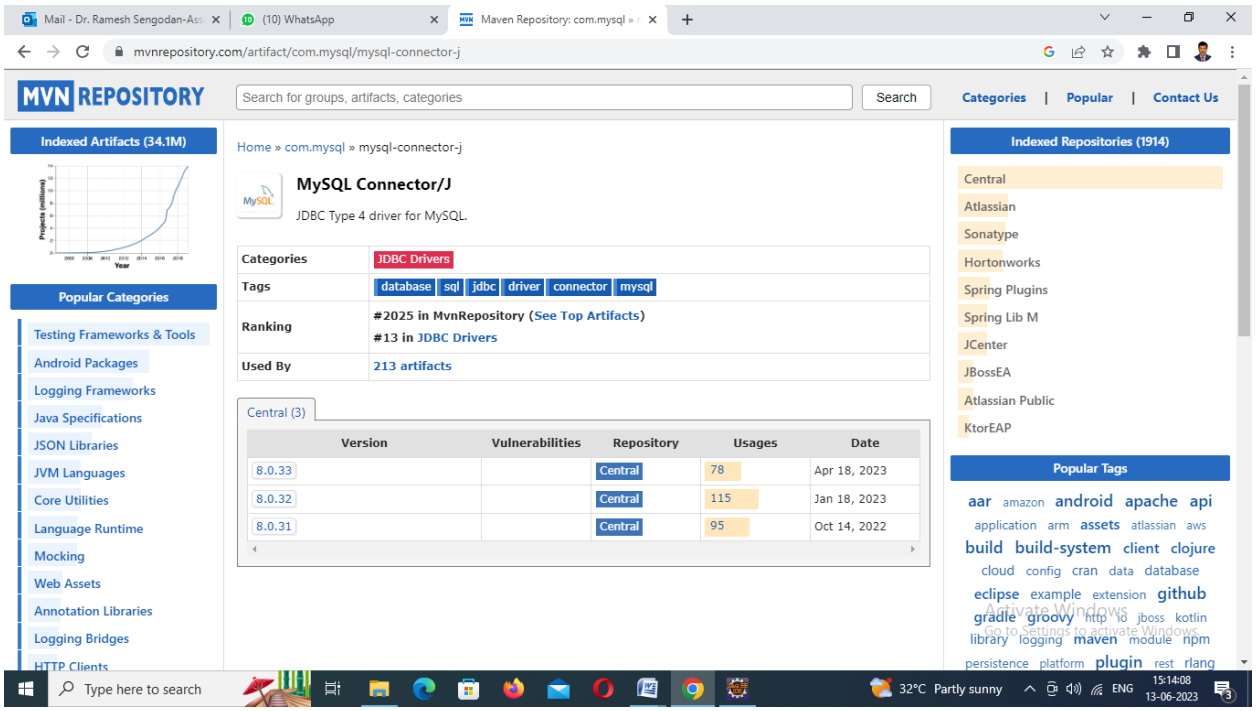

Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: org.hibernate
mvnrepository.com/artifact/org.hibernate/hibernate-core
MVN REPOSITORY
Search for groups, artifacts, categories
Search
Categories | Popular | Contact Us
Indexed Artifacts (34.1M)
Projects (millions)
Year
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Core Utilities
Language Runtime
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Home » org.hibernate » hibernate-core
Hibernate Core Relocation
Hibernate's core ORM functionality
License LGPL 2.1
Categories Object/Relational Mapping
Tags hibernate persistence relational mapping
Ranking #110 in MvnRepository (See Top Artifacts)
#1 in Object/Relational Mapping
Used By 4,122 artifacts
Note: This artifact was moved to:
org.hibernate.orm » hibernate-core » 6.2.4.Final
Central (307) Atlassian (2) Atlassian 3rdParty (4) Atlassian 3rd-P Old (24) Spring Lib Release (1)
Spring Plugins (4) Redhat GA (65) Redhat EA (22) ImageJ Public (2) Grails Core (5) JCenter (23)
Geomajas (1) TU-Darmstadt (1) JasperSoft (1) ICM (4)
Version Vulnerabilities Repository Usages Date
6.2.4.Final Central 1 Jun 01, 2023
Indexed Repositories (1914)
Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP
Popular Tags
aar amazon android apache api
application arm assets atlassian aws
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http io jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:10:46
13-06-2023

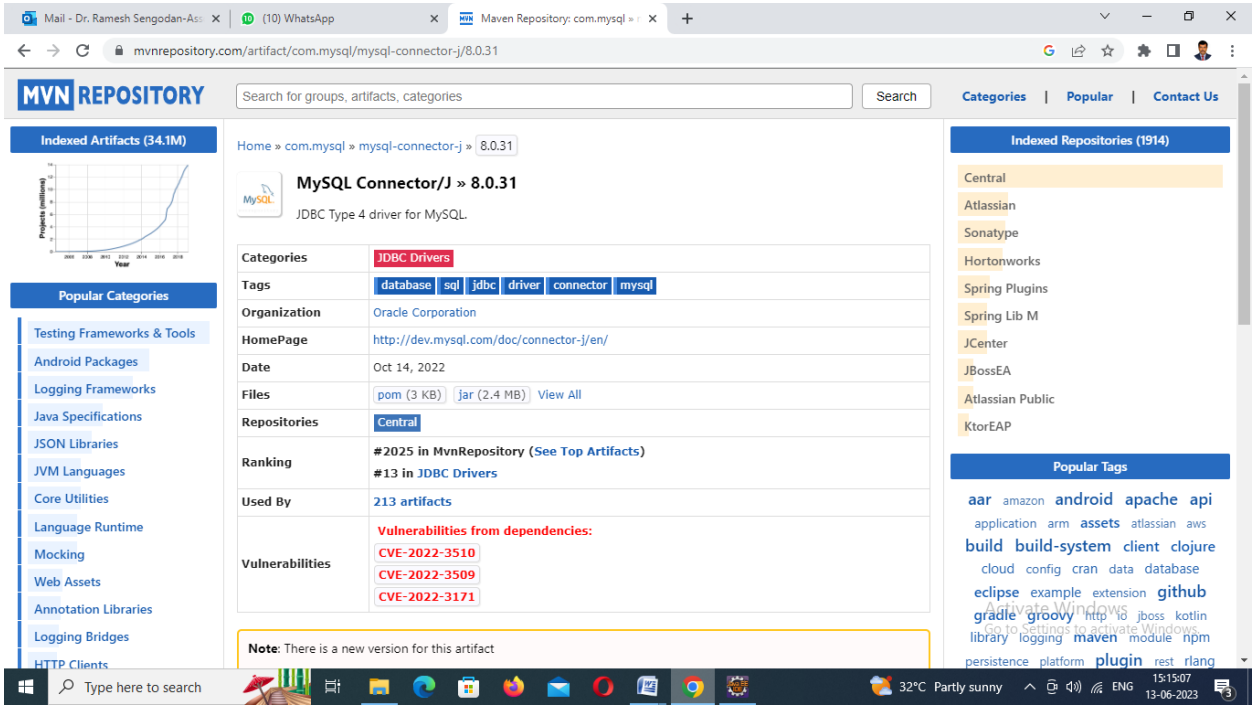

Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: org.hibernate
mvnrepository.com/artifact/org.hibernate/hibernate-core
4.1.x
4.1.12.Final 1 vulnerability Central 42 Apr 25, 2013
4.1.11.Final 1 vulnerability Central 18 Mar 18, 2013
4.1.10.Final 1 vulnerability Central 20 Feb 20, 2013
4.1.9.Final 1 vulnerability Central 58 Dec 13, 2012
4.1.8.Final 1 vulnerability Central 24 Nov 01, 2012
4.1.7.Final 1 vulnerability Central 84 Sep 06, 2012
4.1.6.Final 1 vulnerability Central 54 Aug 08, 2012
4.1.5.SP1 1 vulnerability Central 11 Jul 12, 2012
4.1.5.Final 1 vulnerability Central 10 Jul 12, 2012
4.1.4.Final 1 vulnerability Central 40 May 31, 2012
4.1.3.Final 1 vulnerability Central 22 May 03, 2012
4.1.2.Final 1 vulnerability Central 12 Apr 05, 2012
4.1.2 1 vulnerability Central 12 Apr 04, 2012
4.1.1.Final 1 vulnerability Central 21 Mar 08, 2012
4.1.0.Final 1 vulnerability Central 39 Feb 09, 2012
4.0.1.Final 1 vulnerability Central 70 Jan 11, 2012
4.0.0.Final 1 vulnerability Central 25 Dec 15, 2011
4.0.0.CR7 1 vulnerability Central 9 Dec 01, 2011
4.0.0.CR6 1 vulnerability Central 15 Nov 10, 2011
1 vulnerability Central 17 Oct 27, 2011
Activate Windows
Go to Settings to activate Windows.
https://mvnrepository.com/artifact/org.hibernate/hibernate-core/4.1.6.Final
Type here to search
32°C Partly sunny
ENG
15:11:30
13-06-2023

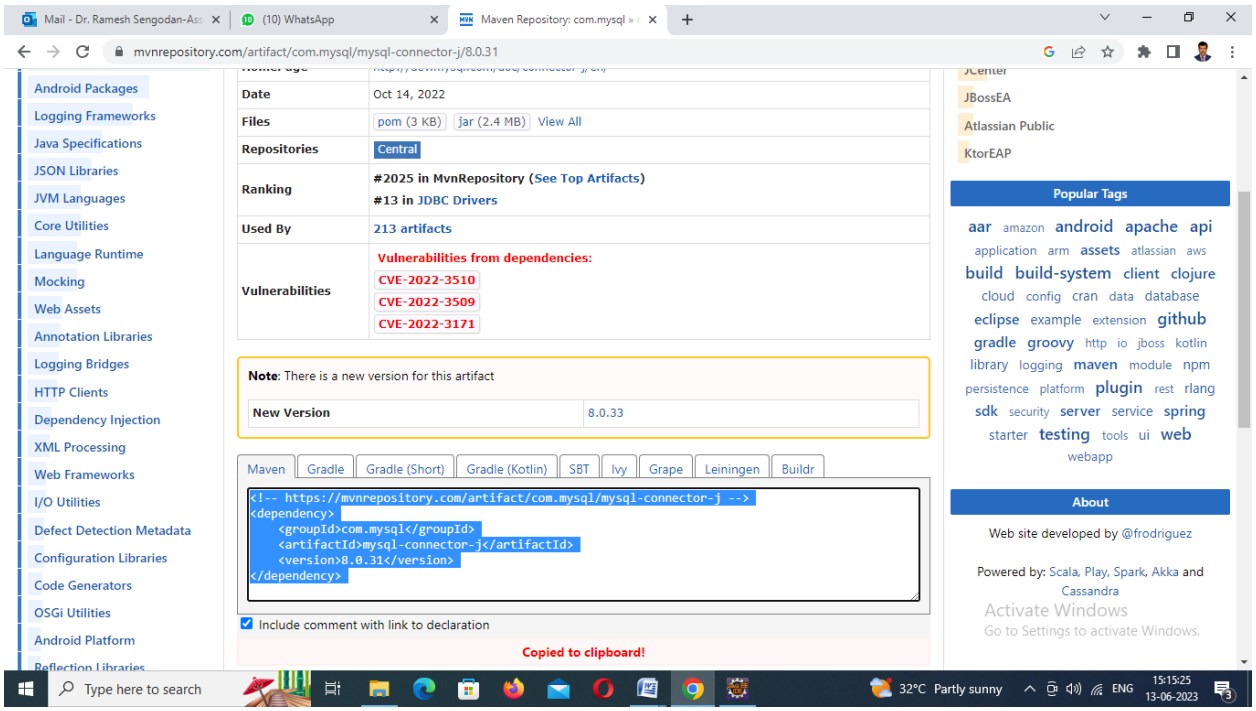

Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: org.hibernate
mvnrepository.com/artifact/org.hibernate/hibernate-core/4.1.6.Final
MVN REPOSITORY
Search for groups, artifacts, categories
Search
Categories | Popular | Contact Us
Indexed Artifacts (34.1M)
Projects (millions)
Year
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Core Utilities
Language Runtime
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Home » org.hibernate » hibernate-core » 4.1.6.Final
Hibernate Core Relocation » 4.1.6.Final
Hibernate's core ORM functionality
License LGPL 2.1
Categories Object/Relational Mapping
Tags hibernate persistence relational mapping
Organization Hibernate.org
HomePage http://hibernate.org
Date Aug 08, 2012
Files pom (9 KB) jar (4.2 MB) View All
Repositories Central JBoss Releases OneBusAway Pub Sonatype Spring Lib M
Ranking #110 in MvnRepository (See Top Artifacts)
#1 in Object/Relational Mapping
Used By 4,122 artifacts
Vulnerabilities
Direct vulnerabilities:
CVE-2020-25638
Vulnerabilities from dependencies:
CVE-2022-45868
CVE-2022-23221
Indexed Repositories (1914)
Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP
Popular Tags
aar amazon android apache api
application arm assets atlassian aws
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http id jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:11:54
13-06-2023


Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: org.hibernate
mvnrepository.com/artifact/org.hibernate/hibernate-core/4.1.6.Final
Language Runtime
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Dependency Injection
XML Processing
Web Frameworks
I/O Utilities
Defect Detection Metadata
Configuration Libraries
Code Generators
OSGi Utilities
Android Platform
Reflection Libraries
JDBC Drivers
Concurrency Libraries
Assertion Libraries
Collections
Validation Libraries
Bytecode Libraries
Used By
4,122 artifacts
Vulnerabilities
Direct vulnerabilities:
CVE-2020-25638
Vulnerabilities from dependencies:
CVE-2022-45868
CVE-2022-23221
CVE-2021-42392
View 4 more ...
A Hibernate Core Module
A module of the Hibernate Core project
Press 'F2' for focus
Note: There is a new version for this artifact
New Version
Maven
Gradle
Gradle (Short)
Gradle (Kotlin)
SBT
Ivy
Grape
Leiningen
Buildr
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>4.1.6.Final</version>
</dependency>
Include comment with link to declaration
Compile Dependencies (7)
Category/License
Group / Artifact
Version
Updates
Parser Generator
antlr » antlr
2.7.7
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http io jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
sdk security server service spring
starter testing tools ui web
webapp
About
Web site developed by @frodriguez
Powered by: Scala, Play, Spark, Akka and Cassandra
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:12:57
13-06-2023


Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: org.hibernate
mvnrepository.com/artifact/org.hibernate/hibernate-core/4.1.6.Final
MVN REPOSITORY
MySQL connector dependency
Search
Categories | Popular | Contact Us
Indexed Artifacts (34.1M)
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Core Utilities
Language Runtime
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Home » org.hibernate » hibernate-core » 4.1.6.Final
Hibernate Core Relocation » 4.1.6.Final
Hibernate's core ORM functionality
License
LGPL 2.1
Categories
Object/Relational Mapping
Tags
hibernate
persistence
relational
mapping
Organization
Hibernate.org
HomePage
http://hibernate.org
Date
Aug 08, 2012
Files
pom (9 KB)
jar (4.2 MB)
View All
Repositories
Central
JBoss Releases
OneBusAway Pub
Sonatype
Spring Lib M
Ranking
#110 in MvnRepository (See Top Artifacts)
#1 in Object/Relational Mapping
Used By
4,122 artifacts
Vulnerabilities
Direct vulnerabilities:
CVE-2020-25638
Vulnerabilities from dependencies:
CVE-2022-45868
CVE-2022-23221
Indexed Repositories (1914)
Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP
Popular Tags
aar amazon android apache api
application arm assets atlassian aws
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http io jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:13:29
13-06-2023


Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: com.mysql
mvnrepository.com/artifact/com.mysql/mysql-connector-j
MVN REPOSITORY
Search for groups, artifacts, categories
Search
Categories | Popular | Contact Us
Indexed Artifacts (34.1M)
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Core Utilities
Language Runtime
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Home » com.mysql » mysql-connector-j
MySQL Connector/J
JDBC Type 4 driver for MySQL.
Categories: JDBC Drivers
Tags: database sql jdbc driver connector mysql
Ranking: #2025 in MvnRepository (See Top Artifacts)
#13 in JDBC Drivers
Used By: 213 artifacts
Central (3)
Version | Vulnerabilities | Repository | Usages | Date
8.0.33 | Central | 78 | Apr 18, 2023
8.0.32 | Central | 115 | Jan 18, 2023
8.0.31 | Central | 95 | Oct 14, 2022
Indexed Repositories (1914)
Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP
Popular Tags
aar amazon android apache api
application arm assets atlassian aws
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http io jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:14:08
13-06-2023


Mail - Dr. Ramesh Sengodan-As
(10) WhatsApp
Maven Repository: com.mysql
mvnrepository.com/artifact/com.mysql/mysql-connector-j/8.0.31
MVN REPOSITORY
Search for groups, artifacts, categories
Search
Categories | Popular | Contact Us
Indexed Artifacts (34.1M)
Popular Categories
Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
JVM Languages
Core Utilities
Language Runtime
Mocking
Web Assets
Annotation Libraries
Logging Bridges
HTTP Clients
Home » com.mysql » mysql-connector-j » 8.0.31
MySQL Connector/J » 8.0.31
JDBC Type 4 driver for MySQL.
Categories: JDBC Drivers
Tags: database sql jdbc driver connector mysql
Organization: Oracle Corporation
HomePage: http://dev.mysql.com/doc/connector-j/en/
Date: Oct 14, 2022
Files: pom (3 KB) jar (2.4 MB) View All
Repositories: Central
Ranking: #2025 in MvnRepository (See Top Artifacts)
#13 in JDBC Drivers
Used By: 213 artifacts
Vulnerabilities from dependencies:
CVE-2022-3510
CVE-2022-3509
CVE-2022-3171
Note: There is a new version for this artifact
Indexed Repositories (1914)
Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP
Popular Tags
aar amazon android apache api
application arm assets atlassian aws
build build-system client clojure
cloud config cran data database
eclipse example extension github
gradle groovy http io jboss kotlin
library logging maven module npm
persistence platform plugin rest rlang
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C Partly sunny
ENG
15:15:07
13-06-2023



pom.xml file

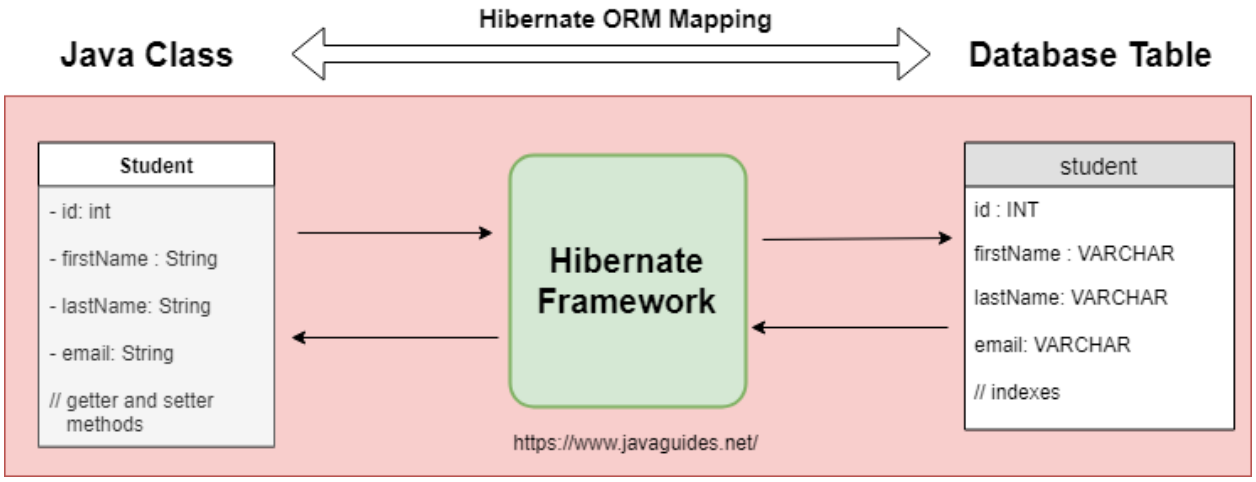
- XML

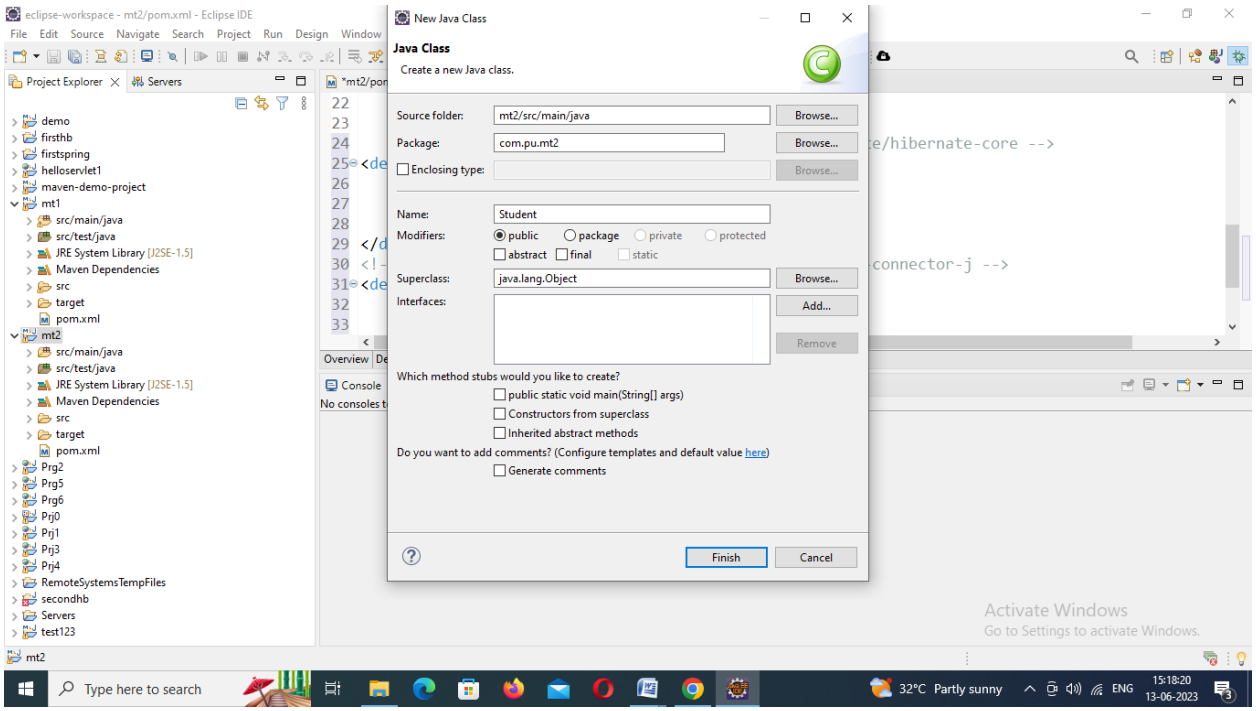
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

  <modelVersion>4.0.0</modelVersion>


  <groupId>com.spring.hibernate</groupId>

  <artifactId>Spring-Hibernate</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <packaging>jar</packaging>
```

```
  <name>Spring-Hibernate</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-
core -->
    <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.14.Final</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.mysql/mysql-
connector-j -->
```

```
    <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>8.0.31</version>
    </dependency>
  </dependencies>
</project>
```

Step 4: Add POJO and main classes for working with the functionality To create a POJO(Module) class file.





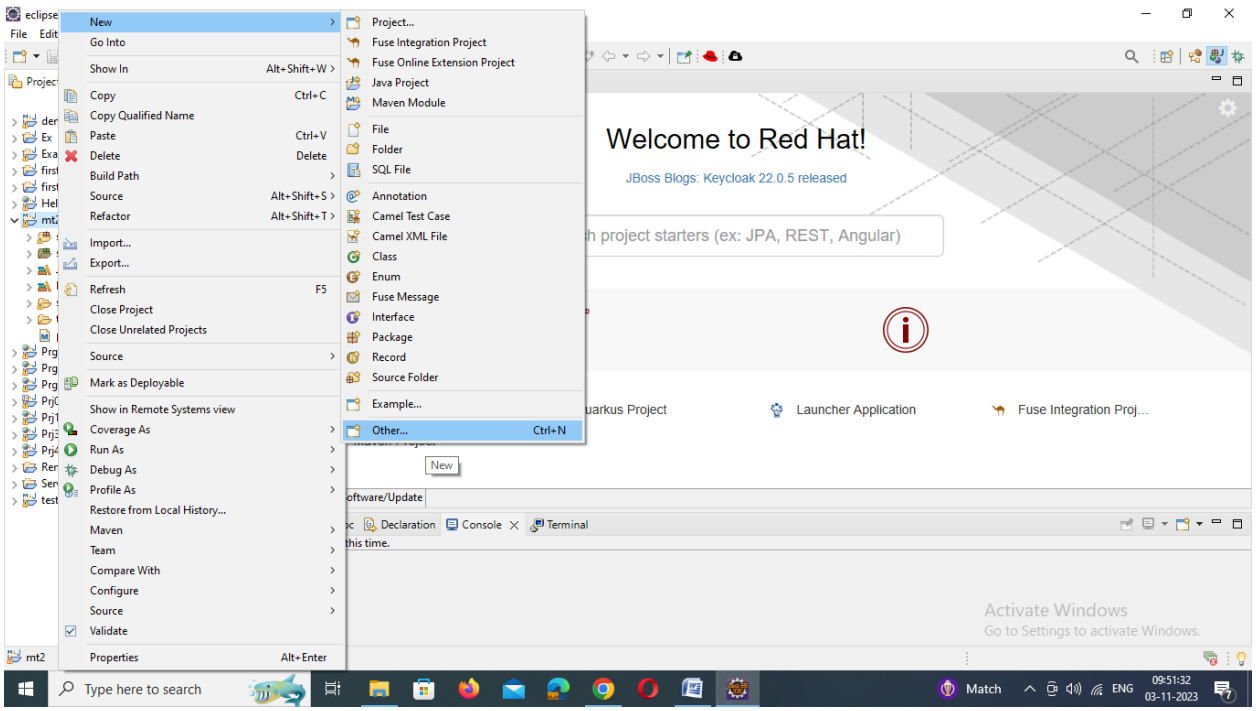
Student.java

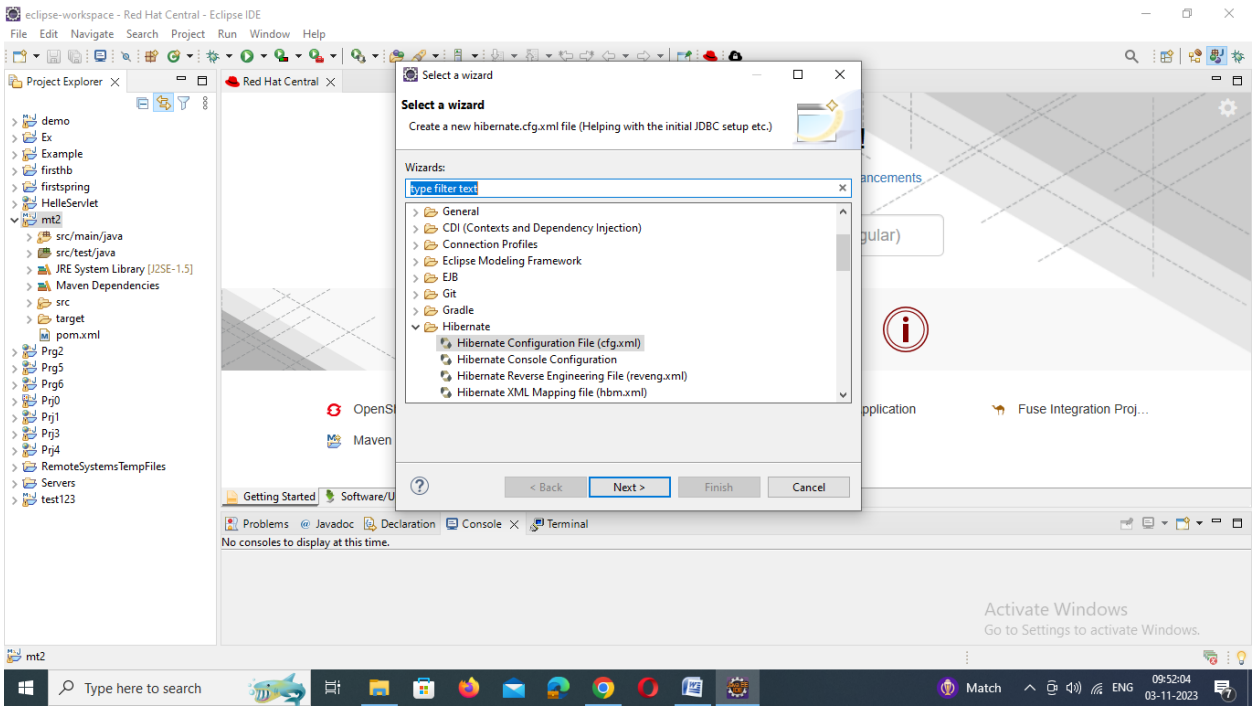
```
package com.pu.test123;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Student {
	@Id
	private int roll;
	private String name;
	private int marks;
	public int getRoll() {
		return roll;
	}
	public void setRoll(int roll) {
		this.roll = roll;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public int getMarks() {
		return marks;
	}
	}
```

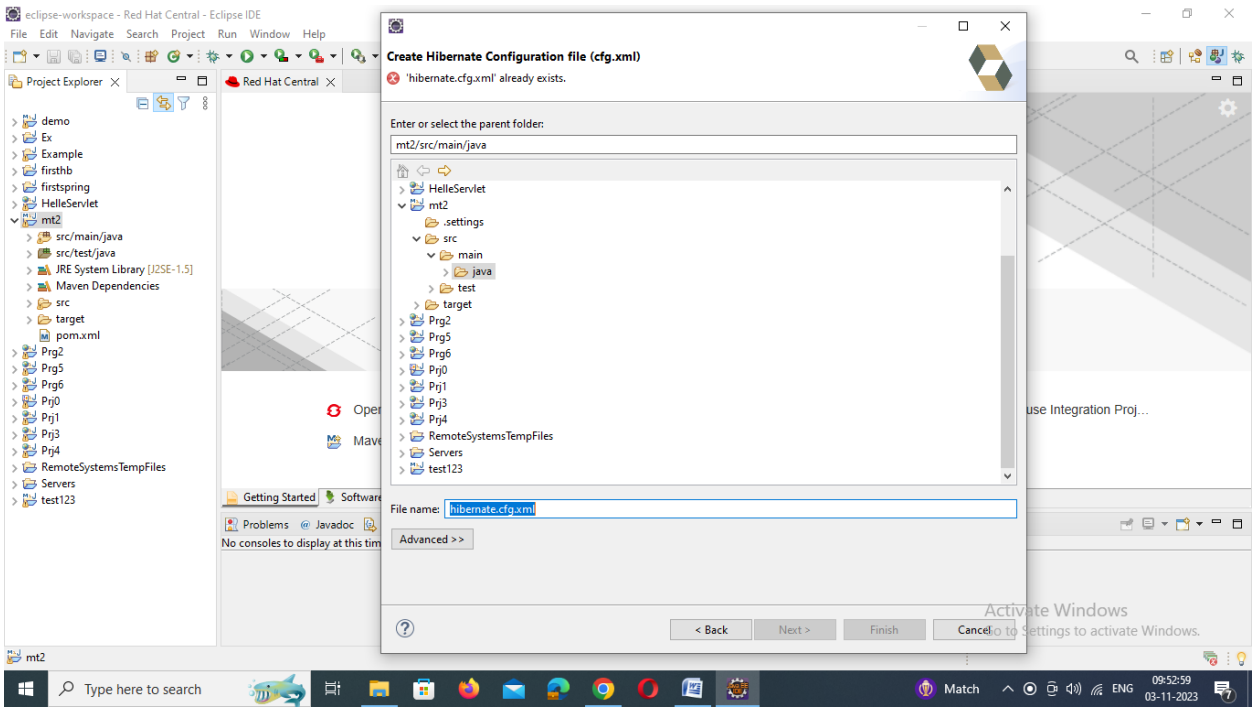
```
	public void setMarks(int marks) {
		this.marks = marks;
	}


}
```

Step 5: Add hibernate.cfg.xml file for database parameters

To create hibernate configuration file


New
Go Into
Show In
Copy
Copy Qualified Name
Paste
Delete
Build Path
Source
Refactor
Import...
Export...
Refresh
Close Project
Close Unrelated Projects
Source
Mark as Deployable
Show in Remote Systems view
Coverage As
Run As
Debug As
Profile As
Restore from Local History...
Maven
Team
Compare With
Configure
Source
Validate
Properties
Project...
Fuse Integration Project
Fuse Online Extension Project
Java Project
Maven Module
File
Folder
SQL File
Annotation
Camel Test Case
Camel XML File
Class
Enum
Fuse Message
Interface
Package
Record
Source Folder
Example...
Other...
Welcome to Red Hat!
JBoss Blogs: Keycloak 22.0.5 released
Launcher Application
Fuse Integration Proj...
Activate Windows
Go to Settings to activate Windows.


Select a wizard
Select a wizard
Create a new hibernate.cfg.xml file (Helping with the initial JDBC setup etc.)
Wizards:
type filter text
General
CDI (Contexts and Dependency Injection)
Connection Profiles
Eclipse Modeling Framework
EJB
Git
Gradle
Hibernate
Hibernate Configuration File (cfg.xml)
Hibernate Console Configuration
Hibernate Reverse Engineering File (reveng.xml)
Hibernate XML Mapping file (hbm.xml)
< Back
Next >
Finish
Cancel
No consoles to display at this time.
Activate Windows
Go to Settings to activate Windows.



Step 6. MySQL Database Setup

Let's create a database named "demo" in MySQL:

```
CREATE DATABASE 'demo';
```

Note: Hibernate will create **Student** tables automatically
Create table Student(roll int,name varchar(30),marks int);

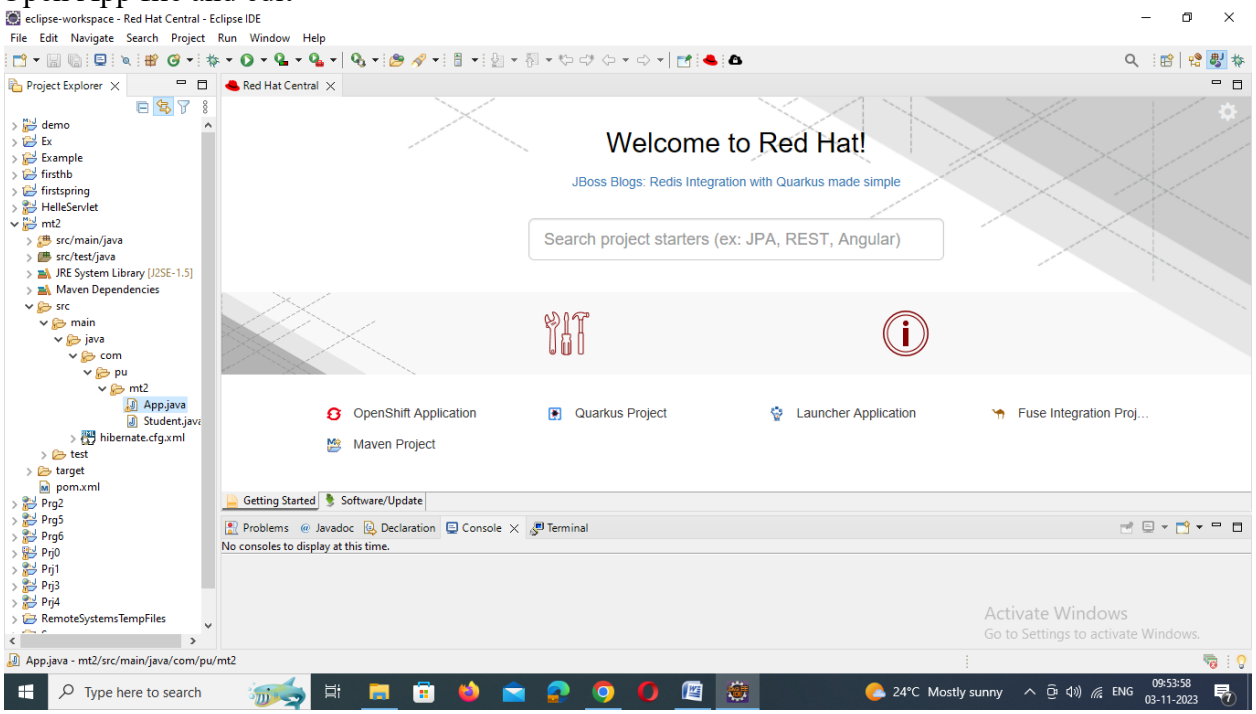
Hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/god?characterEncoding=latin1</
property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">admin123</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="show_sql">true</property>
<property name="format_sql">true</property>
```

```
<property name="hbm2ddl.auto">update </property>
<mapping resource="com.pu.student.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

Step 7: Create the main App class and Run an Application

Open App file and edit



App.java

```
package com.pu.test123;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
/**
 * Hello world!
 *
 */
public class App
{
   public static void main( String[] args )
   {
     Student student = new Student();
     student.setRoll(1);
```

```
        student.setName("god");
        student.setMarks(100);
            //1. create Configuration object
        Configuration configuration=new
Configuration().configure().addAnnotatedClass(Student.class);
        //2. create Session Factroy object
        SessionFactory sessionFactory=configuration.buildSessionFactory();
        //3. Create Session object
        Session session=sessionFactory.openSession();
        //4. Begin your transaction
        Transaction transaction=session.beginTransaction();
        //5.Save your object to database
        session.save(student);
        //6/ Commit your transaction
        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```

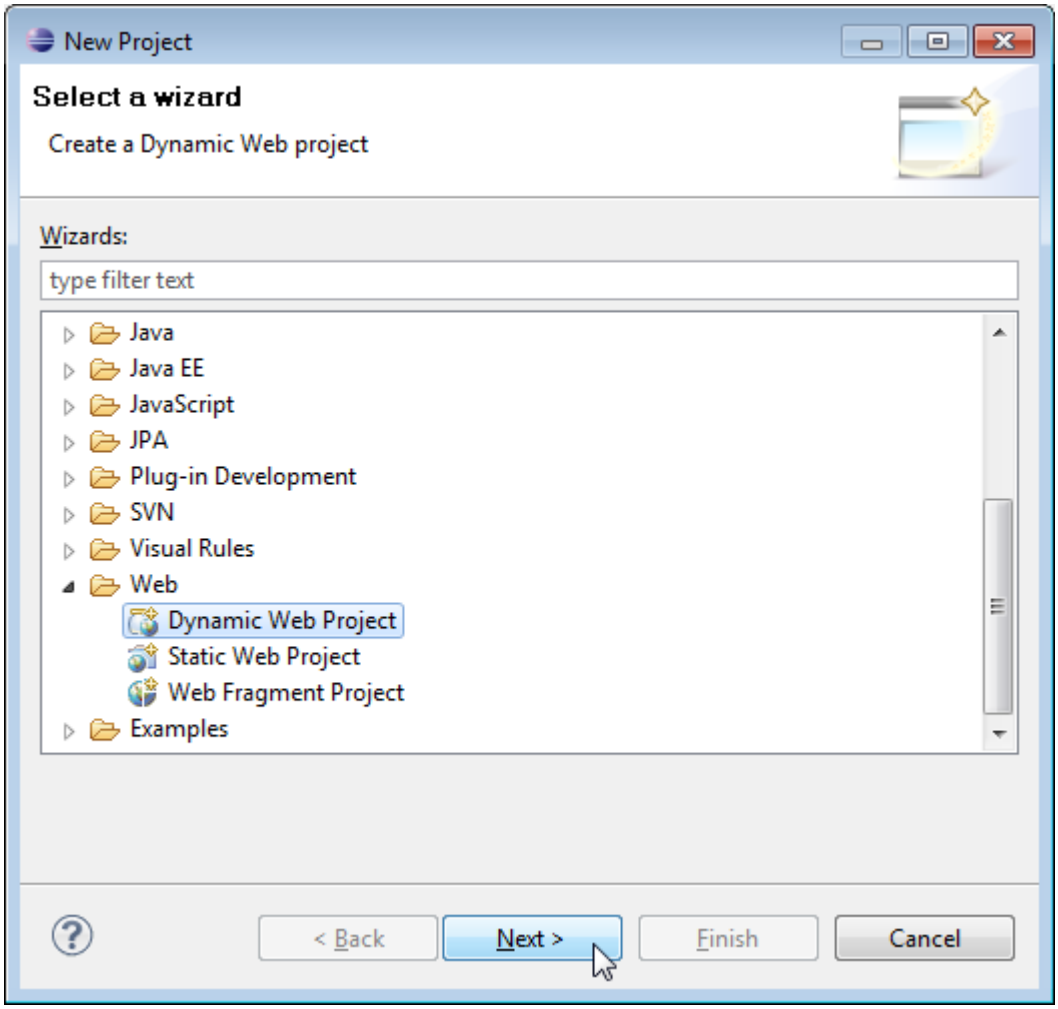
RESULT:
Thus created a step-by-step hibernate application to demonstrate the use of Java-based configuration without using *hibernate.cfg.xml* to connect the *MySQL* database.

Expt10:
Develop a User Registration Form using **JSP**, **Servlet**, MySQL database using the Hibernate framework.
AIM:
To create a User Registration project using **JSP**, **Servlet**, **Hibernate Framework**, and **MySQL** database.
1. Creating a User Registration form using JSP
2. Submit the User Registration form with a POST request
3. After form submission the corresponding servlet will get called
4. UserController class handles all the request parameters and sends a request to the UserDao class to save this data to the database.

PROCEDURE:

1. Create a Dynamic Web Project in Eclipse IDE

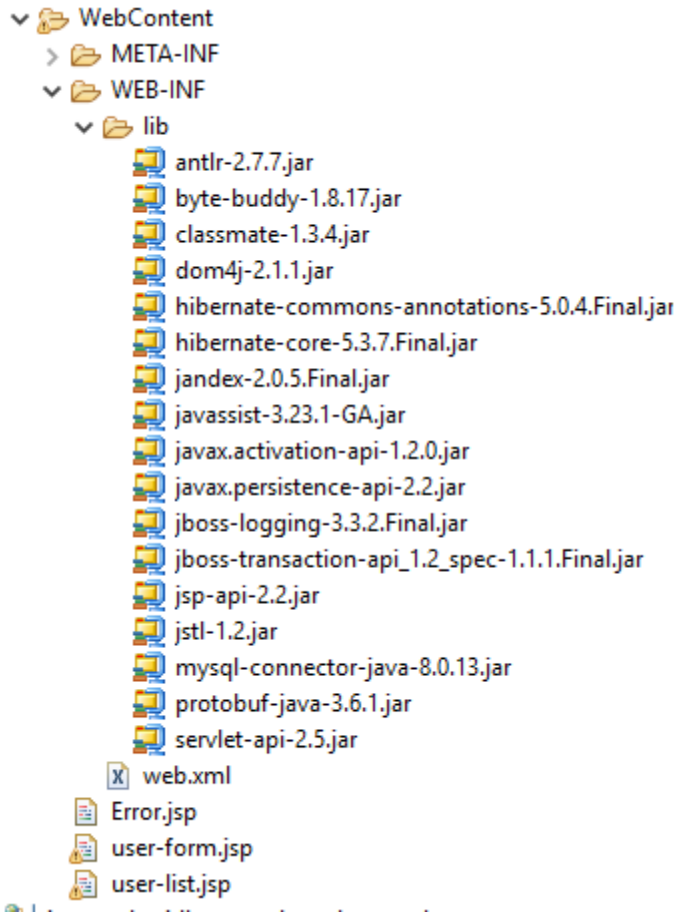
To create a new dynamic Web project in Eclipse:
1. On the main menu select **File > New > Project...**
2. In the upcoming wizard choose **Web > Dynamic Web Project.**



3. Click **Next**.
4. Enter project name as "hb1";
5. Make sure that the target runtime is set to Apache Tomcat with the currently supported version.

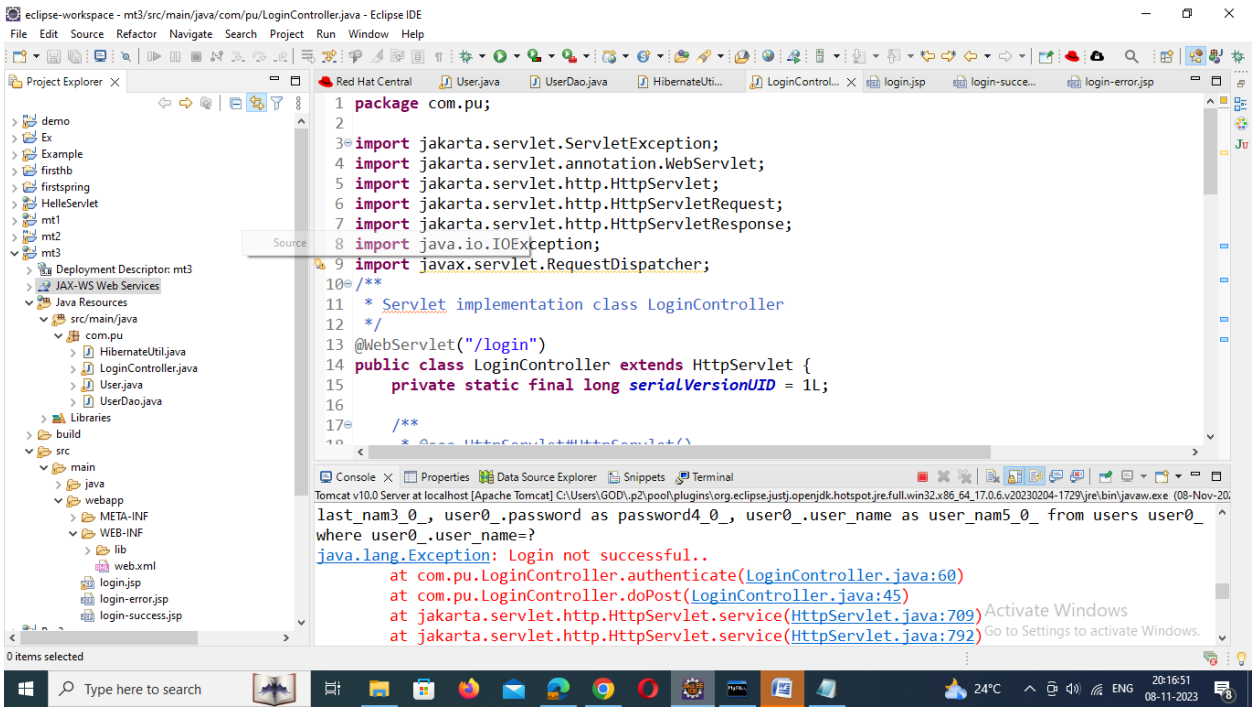
2. Add Jar Dependencies

You can get these jar dependencies from the GitHub repository (the link given at the end of this tutorial). Add the latest release of the below jar files to the lib folder:



3. Project Structure

Create project structure or packaging structure as per the below screenshot:



4. MySQL Database Setup

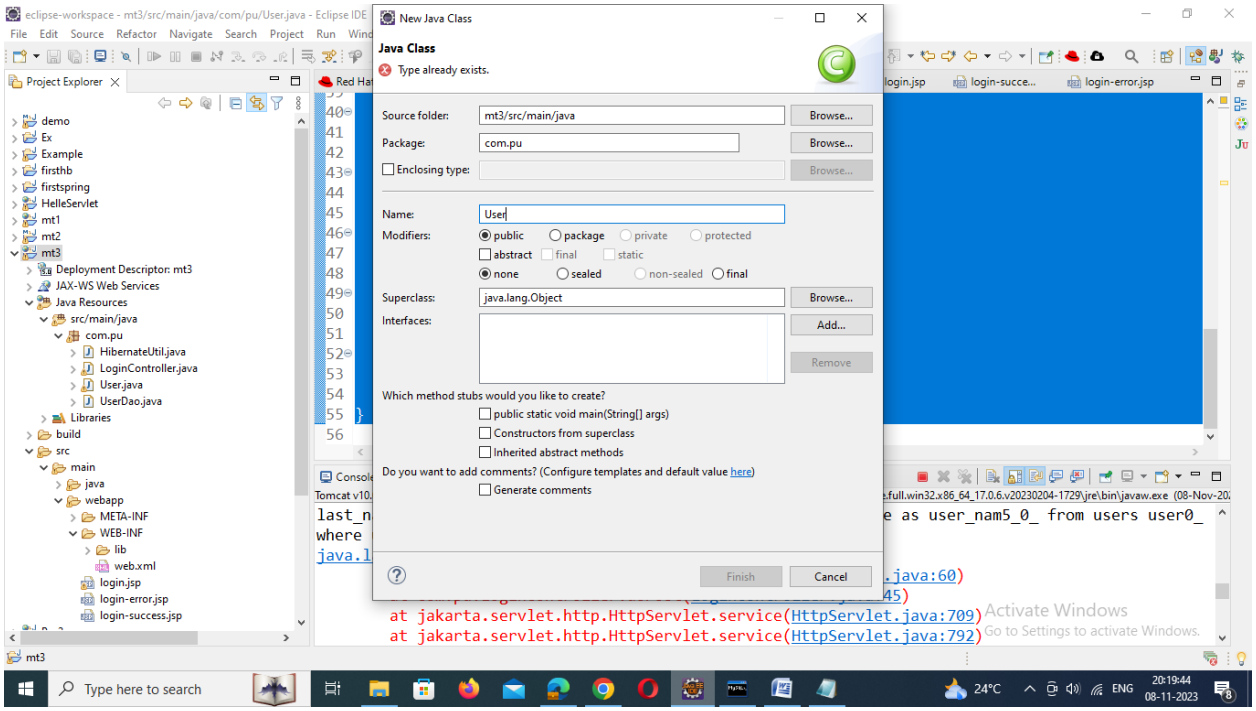
Let's create a database named "demo" in MySQL:

```
CREATE DATABASE 'demo';
```

Note: Hibernate will create **users** tables automatically. registered users so we will use the same **users** table to authenticate a User in this example.

5. Create a JPA Entity - User.java

Next, create User JPA Entity class and add the following content to it:



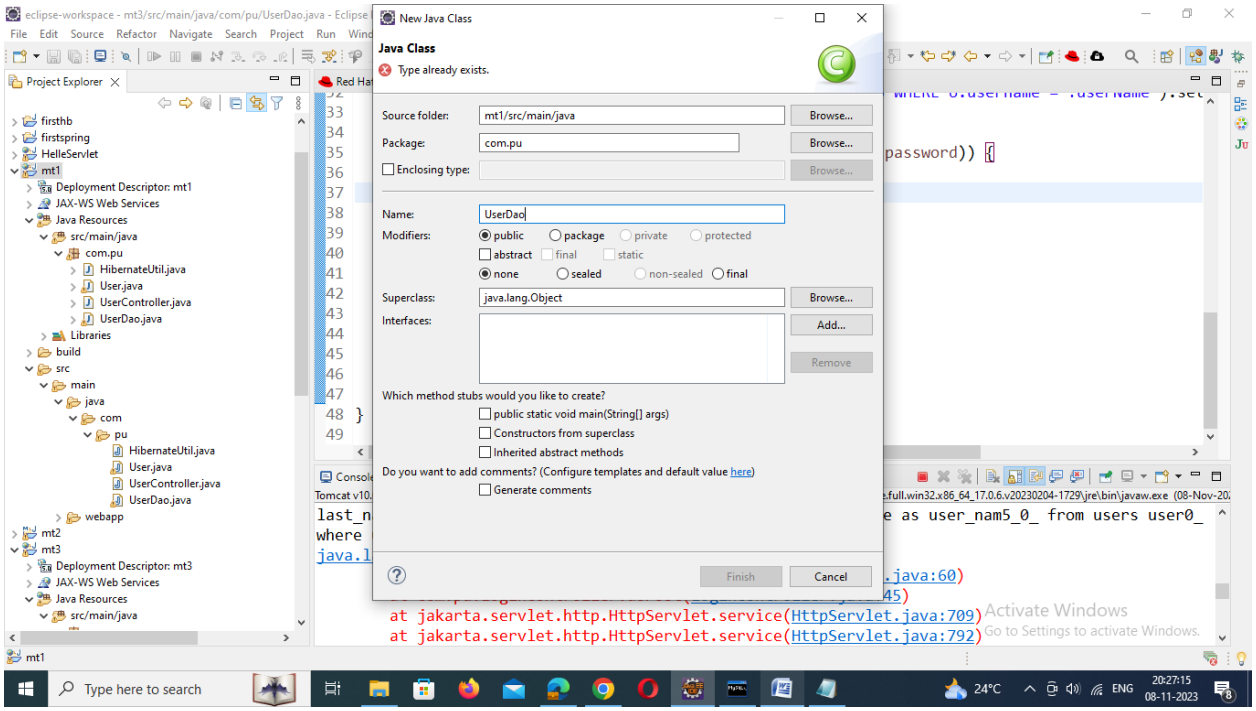
User.java

```
package com.pu;
import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "users")
public class User implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
```

```
    private String lastName;

    @Column(name = "user_name")
    private String username;

    @Column(name = "password")
    private String password;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    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;
    }
}
```

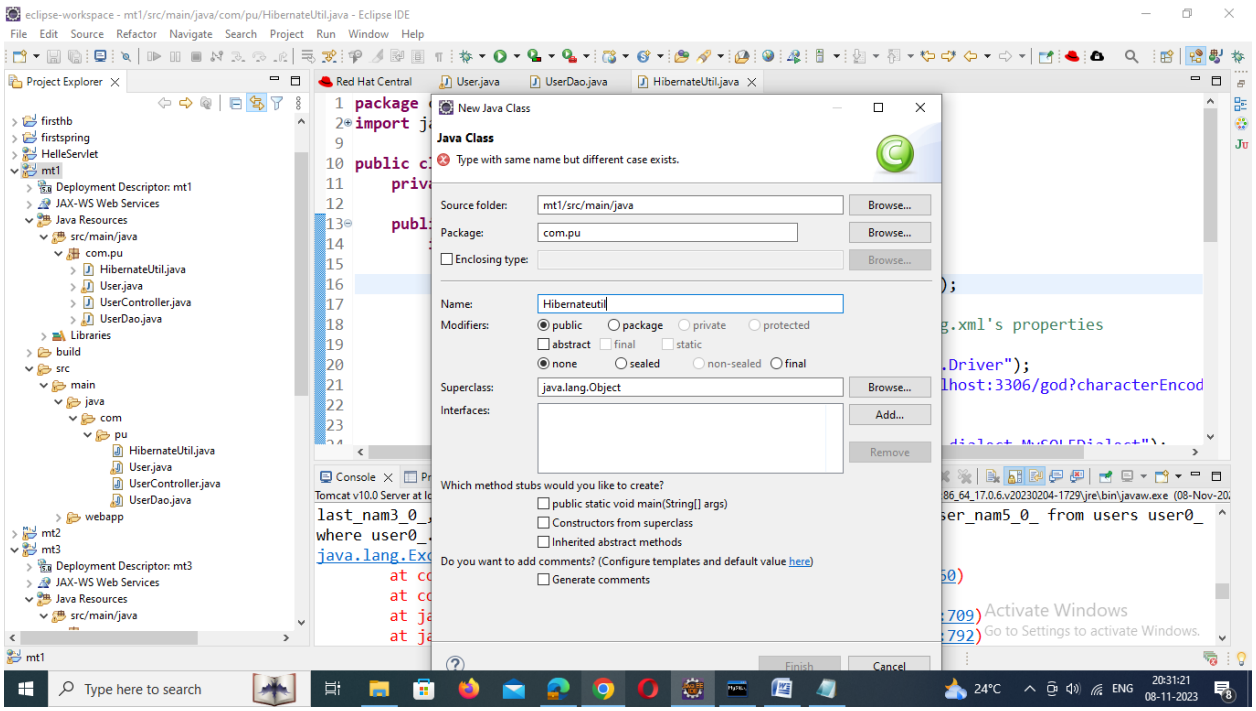
6. Create UserDao to Save Registered User into Database

Let's create a *UserDao* class and add the *saveUser()* method to save Users into the *users* table in a database using Hibernate.



UserDao.java

```
package com.pu;

import org.hibernate.Session;
import org.hibernate.Transaction;

public class UserDao {
        public void saveUser(User user) {
      Transaction transaction = null;
      try (Session session = HibernateUtil.getSessionFactory().openSession()) {
         // start a transaction
         transaction = session.beginTransaction();
         // save the student object
         session.save(user);
         // commit transaction
         transaction.commit();
      } catch (Exception e) {
         if (transaction != null) {
            transaction.rollback();
         }
         e.printStackTrace();
      }
   }
```



7. Hibernate Java-Based Configuration

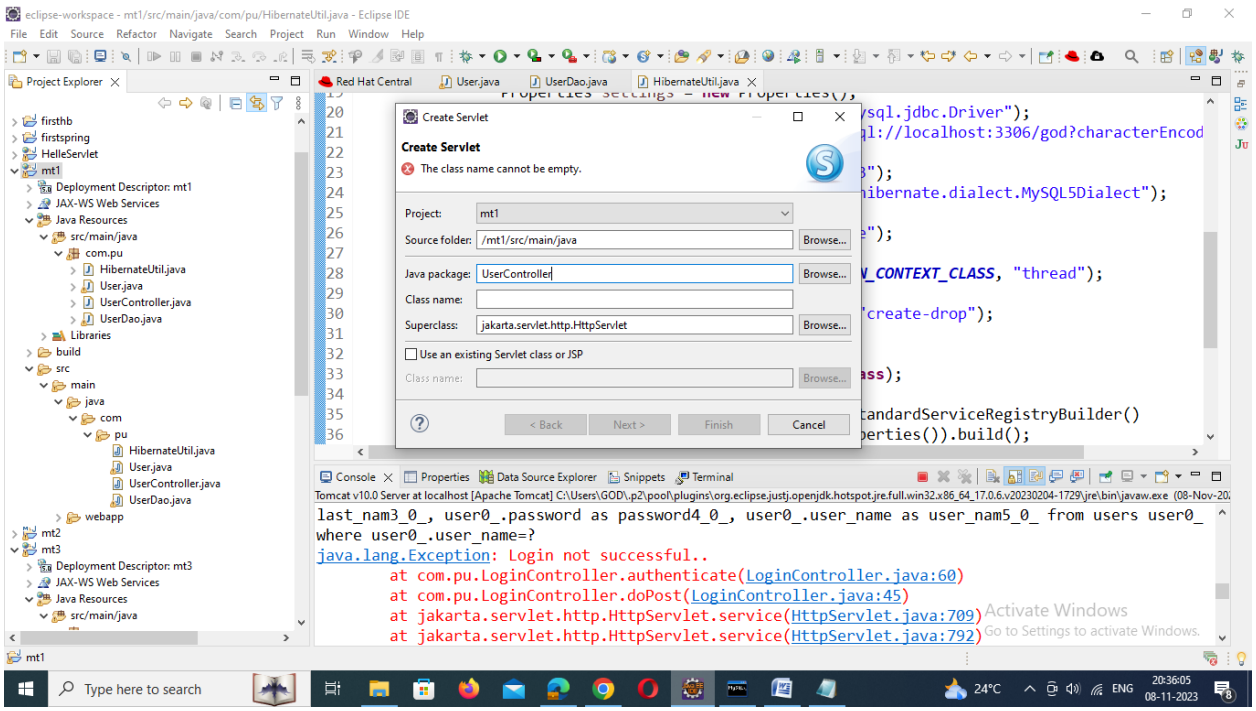
HibernateUtil.java

```
package com.pu;

import java.util.Properties;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtil {
        private static SessionFactory sessionFactory;

   public static SessionFactory getSessionFactory() {
      if (sessionFactory == null) {
         try {
            Configuration configuration = new Configuration();

            // Hibernate settings equivalent to hibernate.cfg.xml's properties
            Properties settings = new Properties();
            settings.put(Environment.DRIVER, "com.mysql.jdbc.Driver");
            settings.put(Environment.URL,
"jdbc:mysql://localhost:3306/demo?characterEncoding=latin1");
            settings.put(Environment.USER, "root");
            settings.put(Environment.PASS, "admin123");
```

```
            settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");

            settings.put(Environment.SHOW_SQL, "true");

            settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

            settings.put(Environment.HBM2DDL_AUTO, "create-drop");

            configuration.setProperties(settings);
            configuration.addAnnotatedClass(User.class);

            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();
            System.out.println("Hibernate Java Config serviceRegistry created");
            sessionFactory = configuration.buildSessionFactory(serviceRegistry);
            return sessionFactory;

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return sessionFactory;
  }
}
```

8. Create a UserController.java

Now, let's create UserController (servlet) that acts as a page controller to handle all requests from the client:




```java
package com.pu;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.RequestDispatcher;
import java.io.*;

/**
 * Servlet implementation class UserController
 */
@WebServlet("/register")
public class UserController extends HttpServlet {
	private static final long serialVersionUID = 1L;
	private UserDao userDao;
    /**
     * @see HttpServlet#HttpServlet()
     */
    public UserController() {
        super();
        // TODO Auto-generated constructor stub
    }
    public void init() {
```

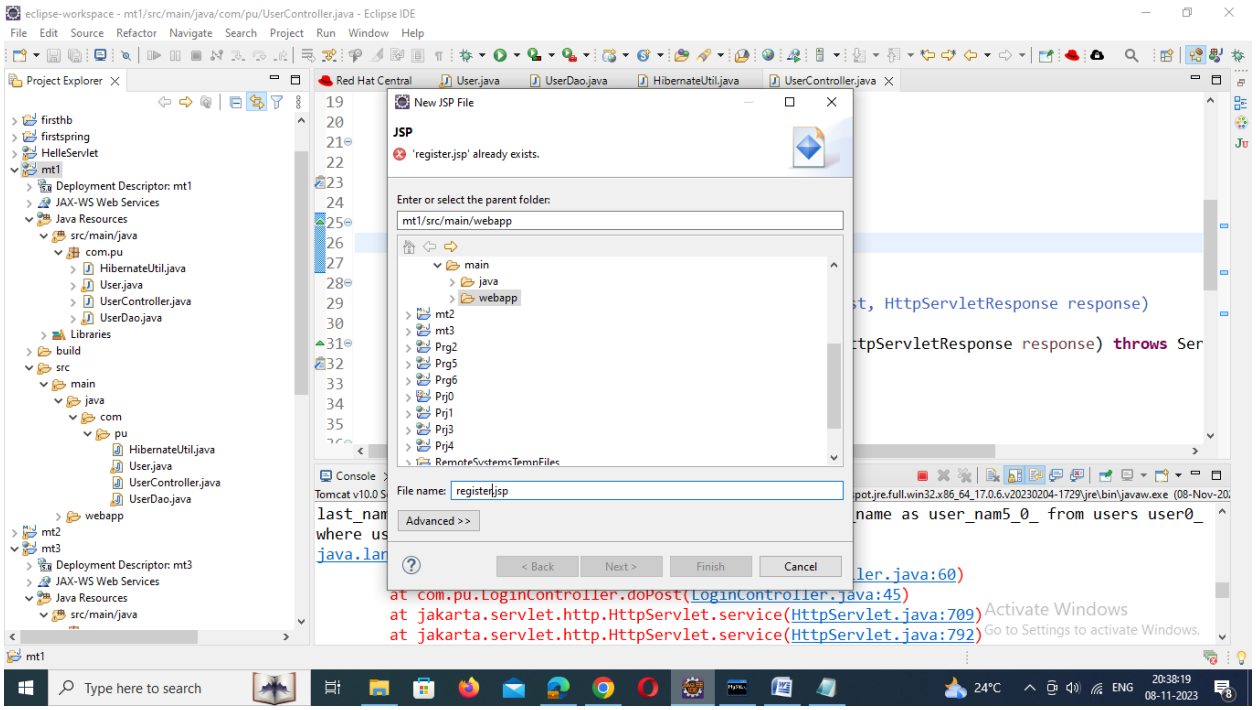
```
        userDao = new UserDao();
    }
        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                // TODO Auto-generated method stub
                response.sendRedirect("register.jsp");
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                // TODO Auto-generated method stub
                 register(request, response);
        }
         private void register(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {
                String firstName = request.getParameter("firstName");
                String lastName = request.getParameter("lastName");
                String username = request.getParameter("username");
                String password = request.getParameter("password");

                User user = new User();
                user.setFirstName(firstName);
                user.setLastName(lastName);
                user.setUsername(username);
                user.setPassword(password);

                userDao.saveUser(user);

                RequestDispatcher dispatcher = request.getRequestDispatcher("register-
success.jsp");
                dispatcher.forward(request, response);
            }

}
```

9. Create a View - register.jsp

Let's design a user registration HTML form with the following fields:

- firstName

- lastName
- username
- password

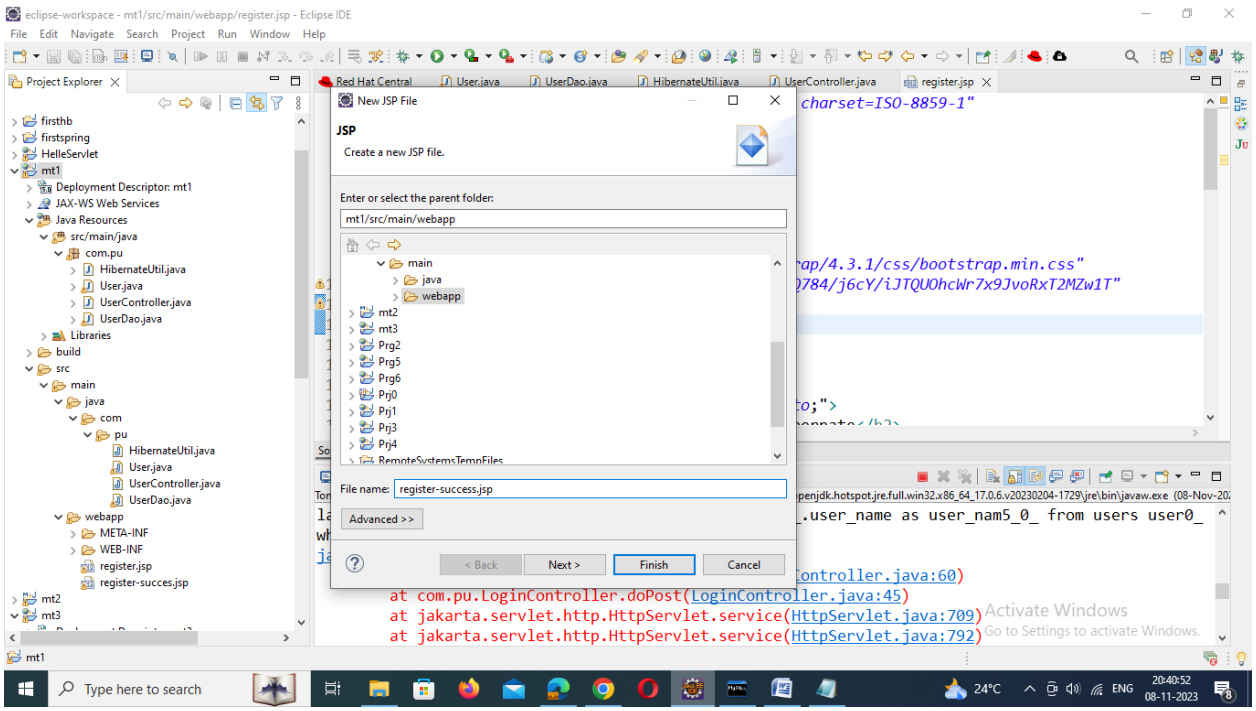



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
 integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
 crossorigin="anonymous">

</head>
<body>
<div class="container">
 <div class="row text-center" style="color: tomato;">
  <h2>User Registration with JSP, Servlet and Hibernate</h2>
 </div>
 <hr>
  <div class="row col-md-10 col-md-offset-3">

   <div class="card card-body">
```

```
<h2>User Register Form</h2>
<div class="col-md-8 col-md-offset-3">

 <form action="<%=request.getContextPath()%>/register" method="post">

  <div class="form-group">
   <label for="uname">First Name:</label> <input type="text"
    class="form-control" id="uname" placeholder="First Name"
    name="firstName" required>
  </div>

  <div class="form-group">
   <label for="uname">Last Name:</label> <input type="text"
    class="form-control" id="uname" placeholder="last Name"
    name="lastName" required>
  </div>

  <div class="form-group">
   <label for="uname">User Name:</label> <input type="text"
    class="form-control" id="username" placeholder="User Name"
    name="username" required>
  </div>

  <div class="form-group">
   <label for="uname">Password:</label> <input type="password"
    class="form-control" id="password" placeholder="Password"
    name="password" required>
  </div>

  <button type="submit" class="btn btn-primary">Submit</button>

 </form>
 </div>
 </div>
 </div>
 </div>
</body>
</html>
```

10. Create a View - register-success.jsp

Let's create a *register-success.jsp* page and add the following code to it:




```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
 integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
 crossorigin="anonymous">
</head>
<body>
<div class="container">
  <div class="row col-md-10 col-md-offset-3">
   <div class="card card-body">
    <h1>User successfully registered!</h1>
   </div>
  </div>
 </div>
</body>
</html>
```

RESULT:

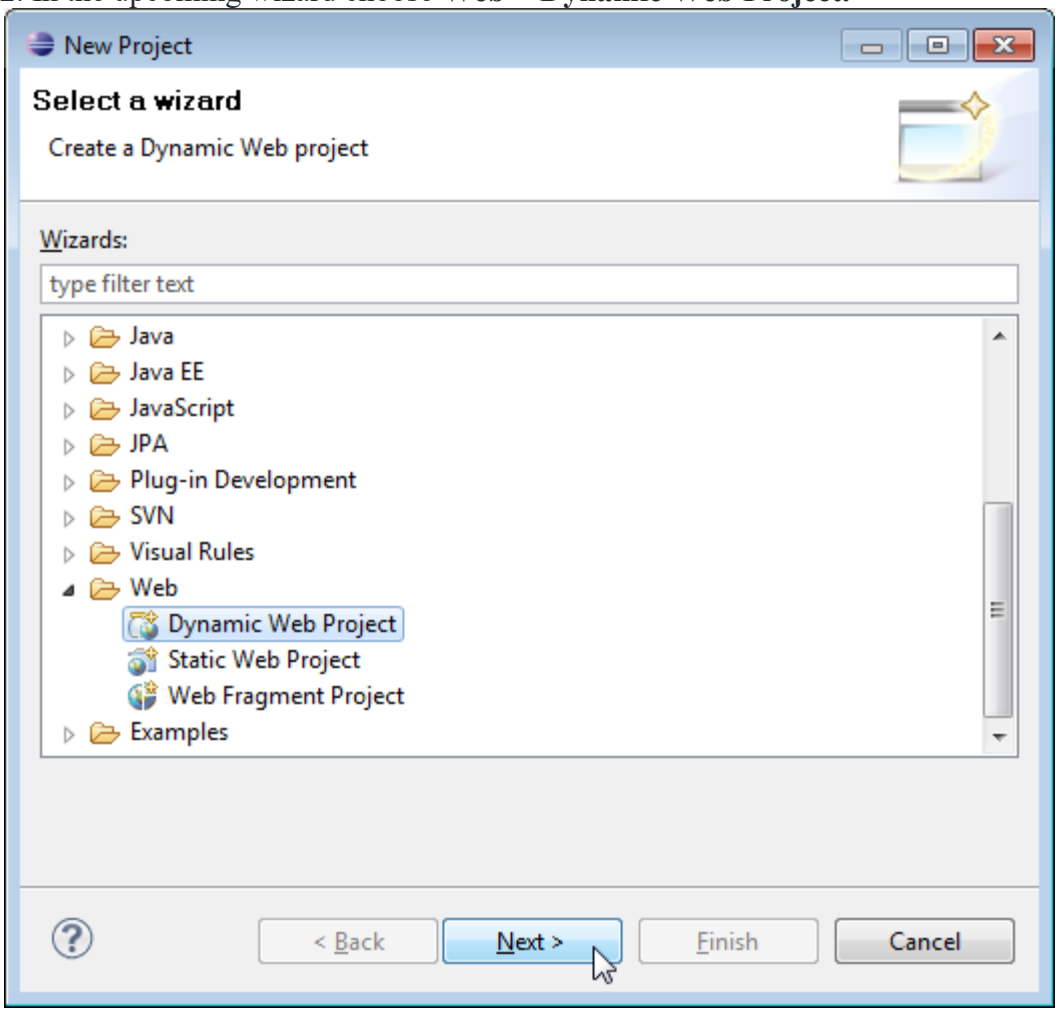
Thus created a registration form using JSP, Servlet, Hibernate, and MySQL.

Expt 11:
Develop a User Login Form and will validate username and password with the MySQL database using the Hibernate framework.
AIM:
To develop a User Login Form and we will validate username and password with the MySQL database using the Hibernate framework.
1. Creating a User Login form using JSP to enter the user credentials
2. . After form submission of the login form the corresponding servlet will get called
3. UserController class handles all the request parameters and sends a request to the UserDao class to check this data in the database.
4. Login Succes & Login Error page to show the outcome.

PROCEDURE:

1. Create a Dynamic Web Project in Eclipse IDE

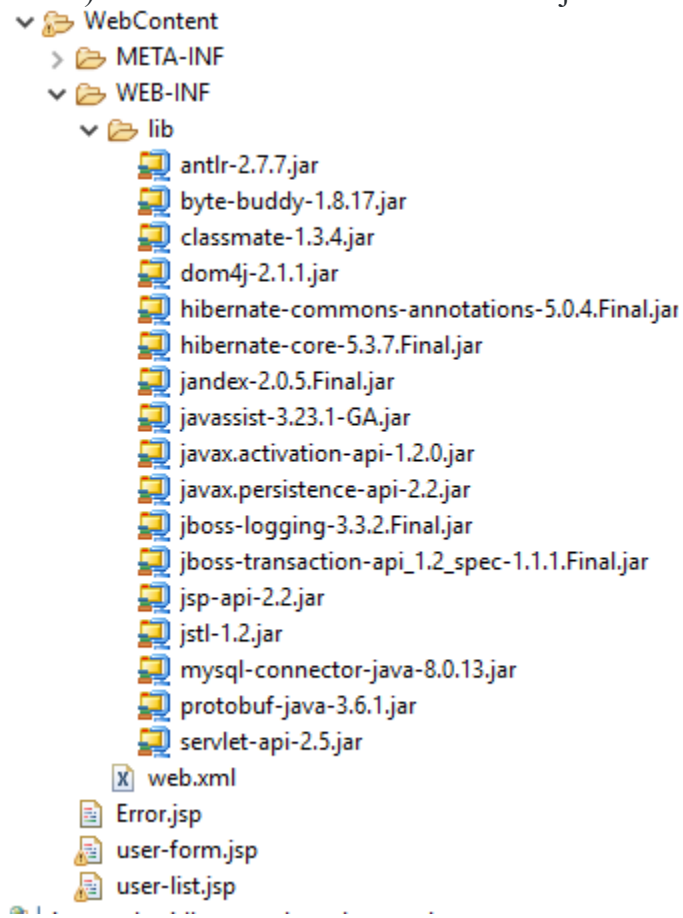
To create a new dynamic Web project in Eclipse:
1. On the main menu select **File > New > Project...**
2. In the upcoming wizard choose **Web > Dynamic Web Project.**



3. Click **Next**.
4. Enter project name as "hb2";
5. Make sure that the target runtime is set to Apache Tomcat with the currently supported version.

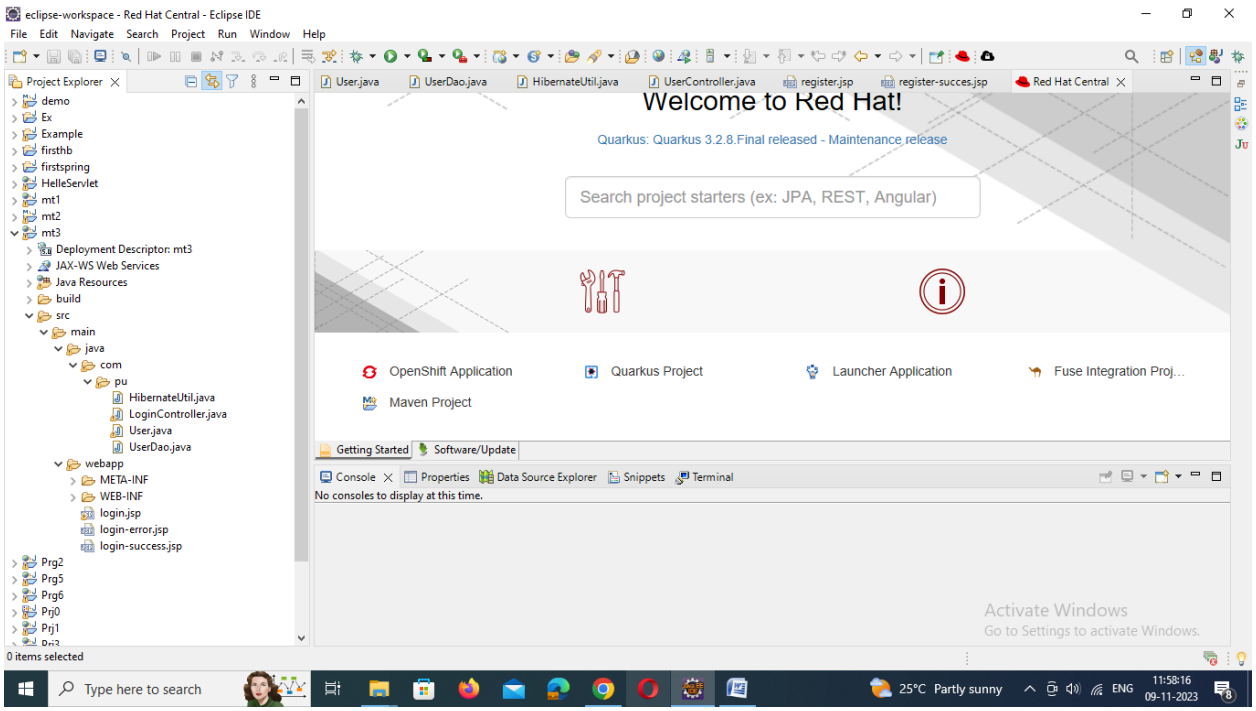
2. Add Jar Dependencies

You can get these jar dependencies from the GitHub repository (the link given at the end of this tutorial). Add the latest release of the below jar files to the lib folder:



3. Project Structure

Create project structure or packaging structure as per the below screenshot:



4. MySQL Database Setup

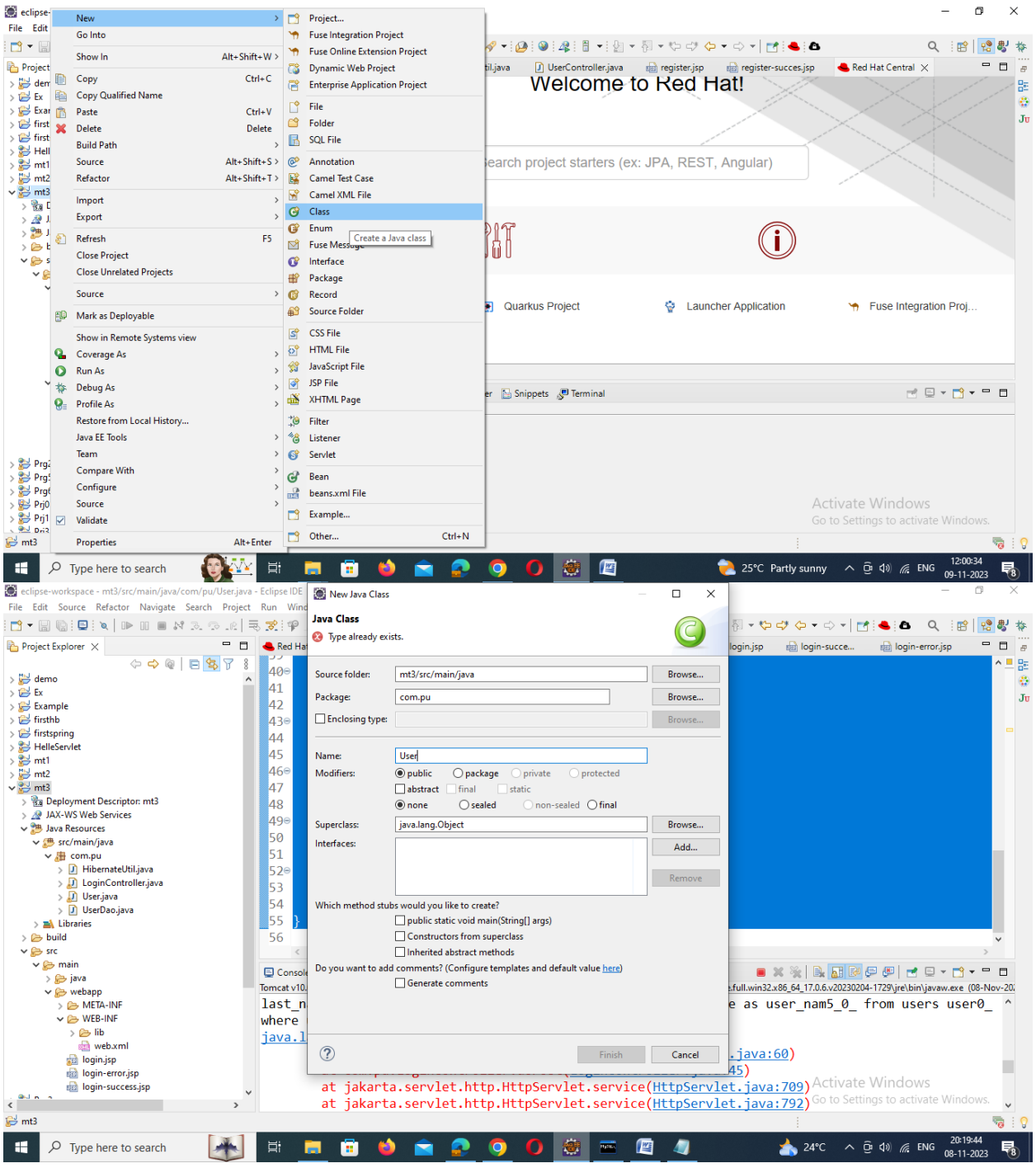
Let's create a database named "demo" in MySQL:

```
CREATE DATABASE 'demo';
```

Note: Hibernate will create **users** tables automatically. In the previous [Hibernate Registration Form Example with JSP, Servlet, and MySQL] article, we have registered users so we will use the same **users** table to authenticate a User in this example.

5. Create a JPA Entity - User.java

Next, create User JPA Entity class and add the following content to it:



User.java

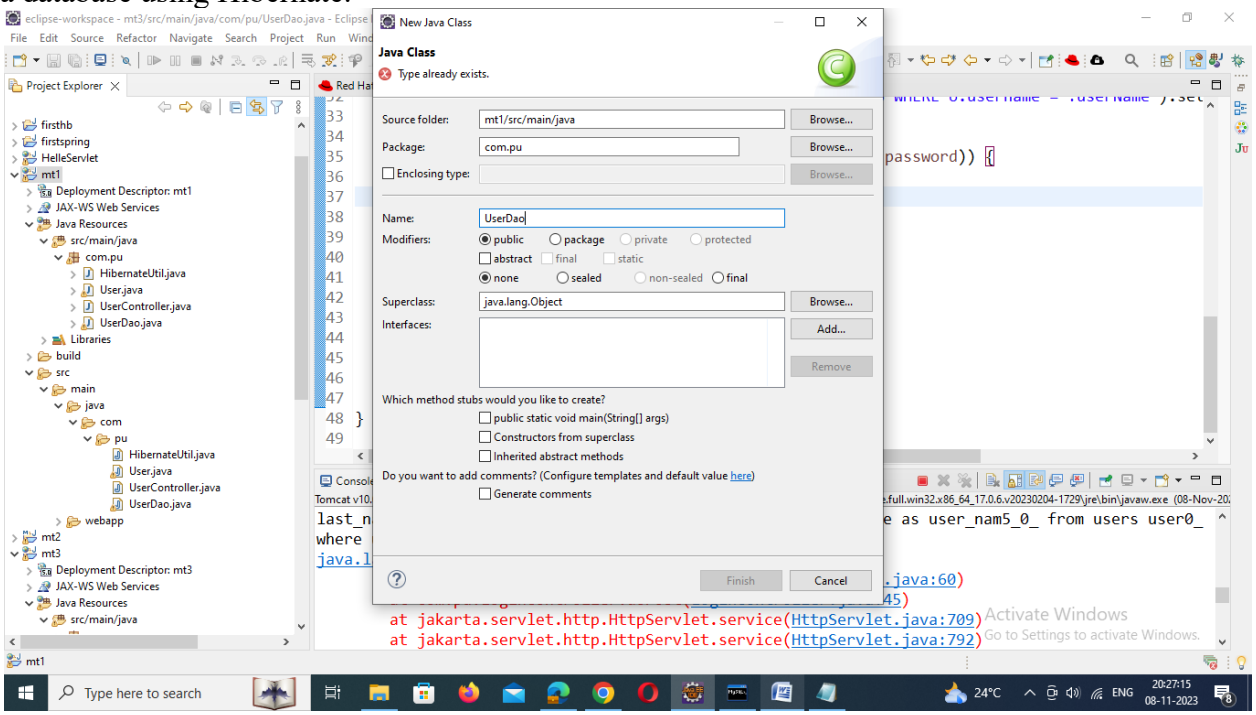
```
package com.pu;
import java.io.Serializable;
```

```java
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "users")
public class User implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "user_name")
    private String username;

    @Column(name = "password")
    private String password;

    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    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;
    }
}
```

6. Create UserDao.java

Let's create a *UserDao* class and add the *saveUser()* method to save Users into the *users* table in a database using Hibernate.

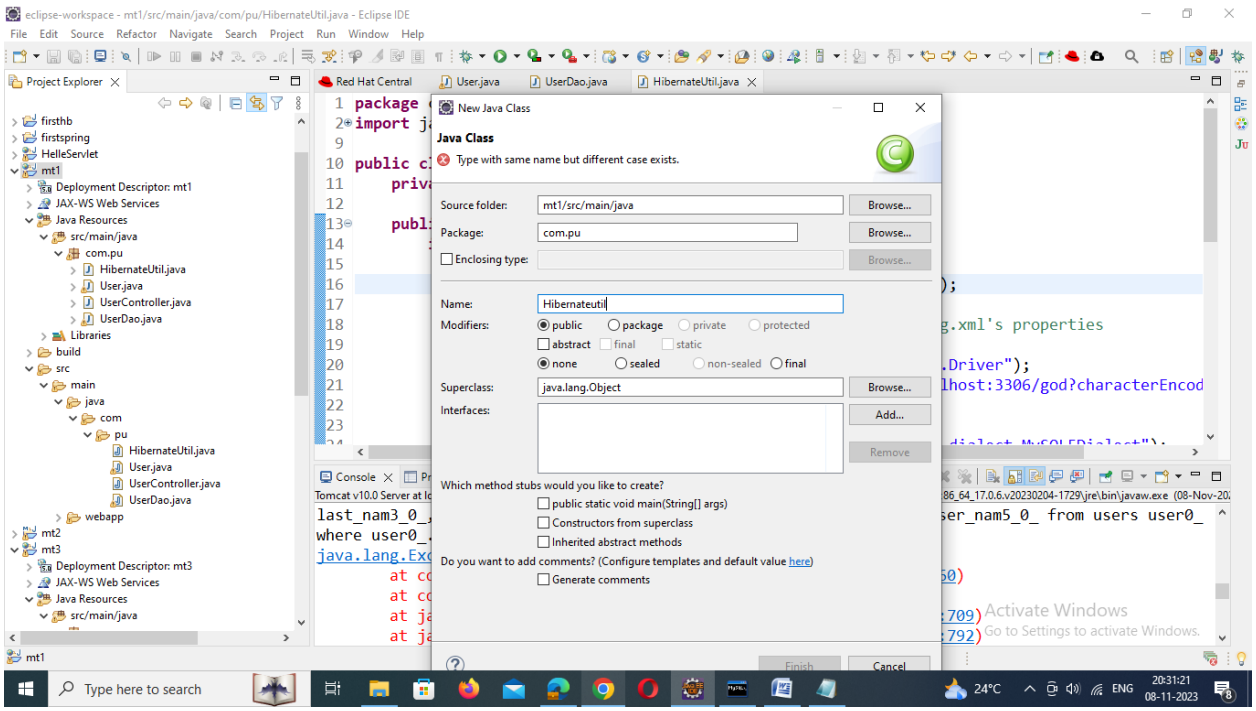


UserDao.java

```
package com.pu;

import org.hibernate.Session;
import org.hibernate.Transaction;

public class UserDao {
        public void saveUser(User user) {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        // start a transaction
        transaction = session.beginTransaction();
        // save the student object
        session.save(user);
        // commit transaction
```

```
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public boolean validate(String userName, String password) {

        Transaction transaction = null;
        User user = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            // start a transaction
            transaction = session.beginTransaction();
            // get an user object
            user = (User) session.createQuery("FROM User U WHERE U.username =
:userName").setParameter("userName", userName)
                .uniqueResult();

            if (user != null && user.getPassword().equals(password)) {
                return true;
            }
            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
        return false;
    }
}
```

7. Hibernate Java-Based Configuration



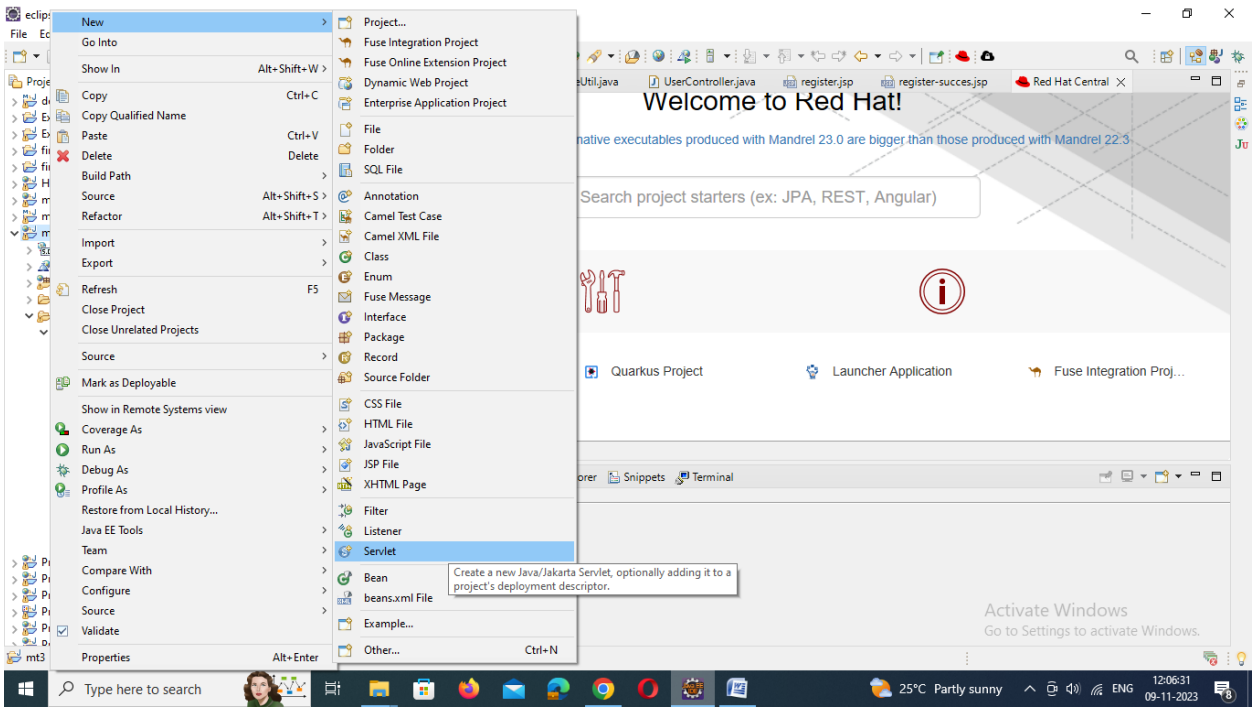
HibernateUtil.java

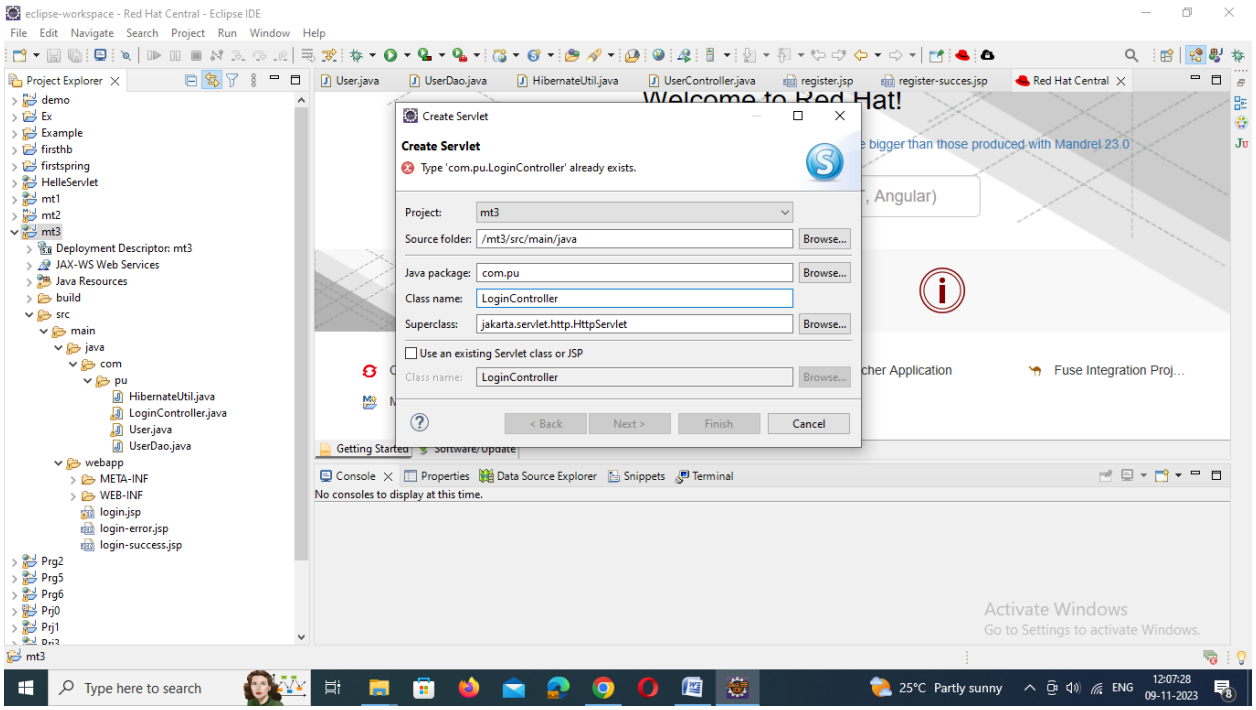
```
package com.pu;

import java.util.Properties;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtil {
        private static SessionFactory sessionFactory;

   public static SessionFactory getSessionFactory() {
      if (sessionFactory == null) {
         try {
            Configuration configuration = new Configuration();

            // Hibernate settings equivalent to hibernate.cfg.xml's properties
            Properties settings = new Properties();
            settings.put(Environment.DRIVER, "com.mysql.jdbc.Driver");
            settings.put(Environment.URL,
"jdbc:mysql://localhost:3306/god?characterEncoding=latin1");
            settings.put(Environment.USER, "root");
            settings.put(Environment.PASS, "admin123");
```

```
            settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");

            settings.put(Environment.SHOW_SQL, "true");

            settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

            settings.put(Environment.HBM2DDL_AUTO, "create-drop");

            configuration.setProperties(settings);
            configuration.addAnnotatedClass(User.class);

            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();
            System.out.println("Hibernate Java Config serviceRegistry created");
            sessionFactory = configuration.buildSessionFactory(serviceRegistry);
            return sessionFactory;

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return sessionFactory;
  }
}
```

8. Create a LoginController.java

Now, let's create LoginController (servlet) that acts as a page controller to handle all requests from the client:






```
package com.pu;
```

```
package com.pu;

import jakarta.servlet.ServletException;
```

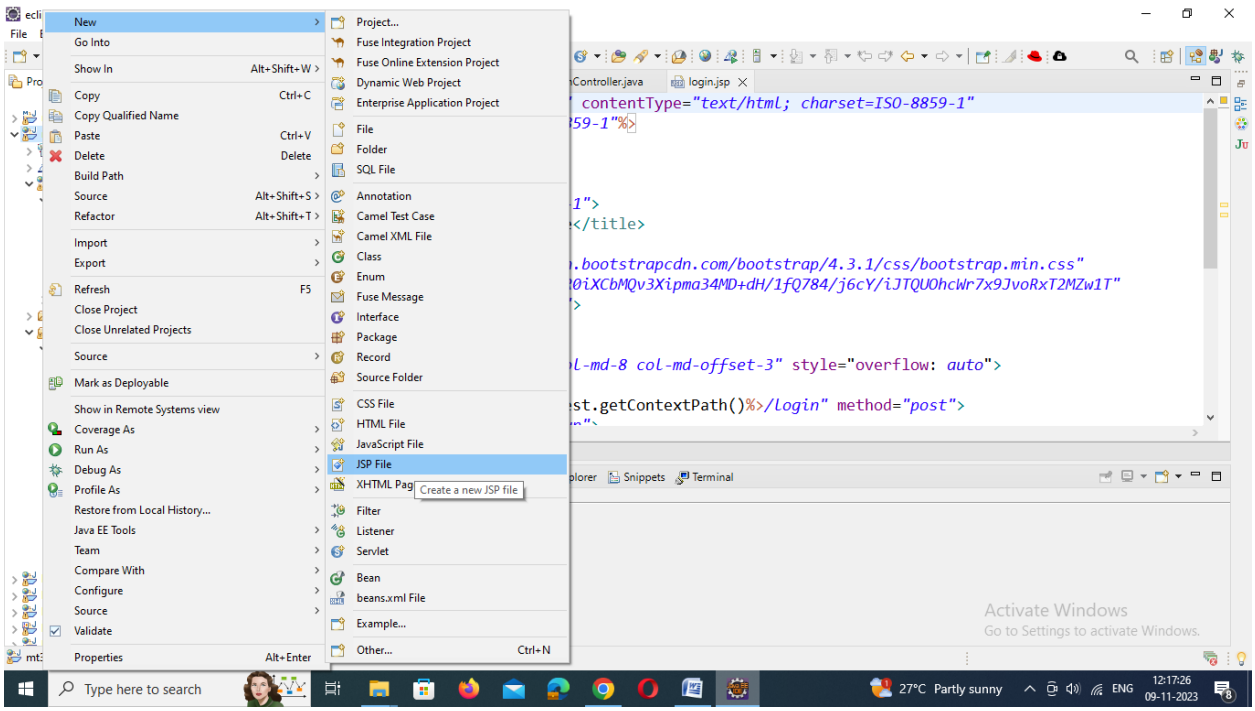
```java
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
/**
 * Servlet implementation class LoginController
 */
@WebServlet("/login")
public class LoginController extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
	private UserDao loginDao;

    public void init() {
        loginDao = new UserDao();
    }

    public LoginController() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub
		response.sendRedirect("login.jsp");
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		// TODO Auto-generated method stub
		try {
	authenticate(request, response);
	} catch (Exception e) {
		// TODO Auto-generated catch block
```

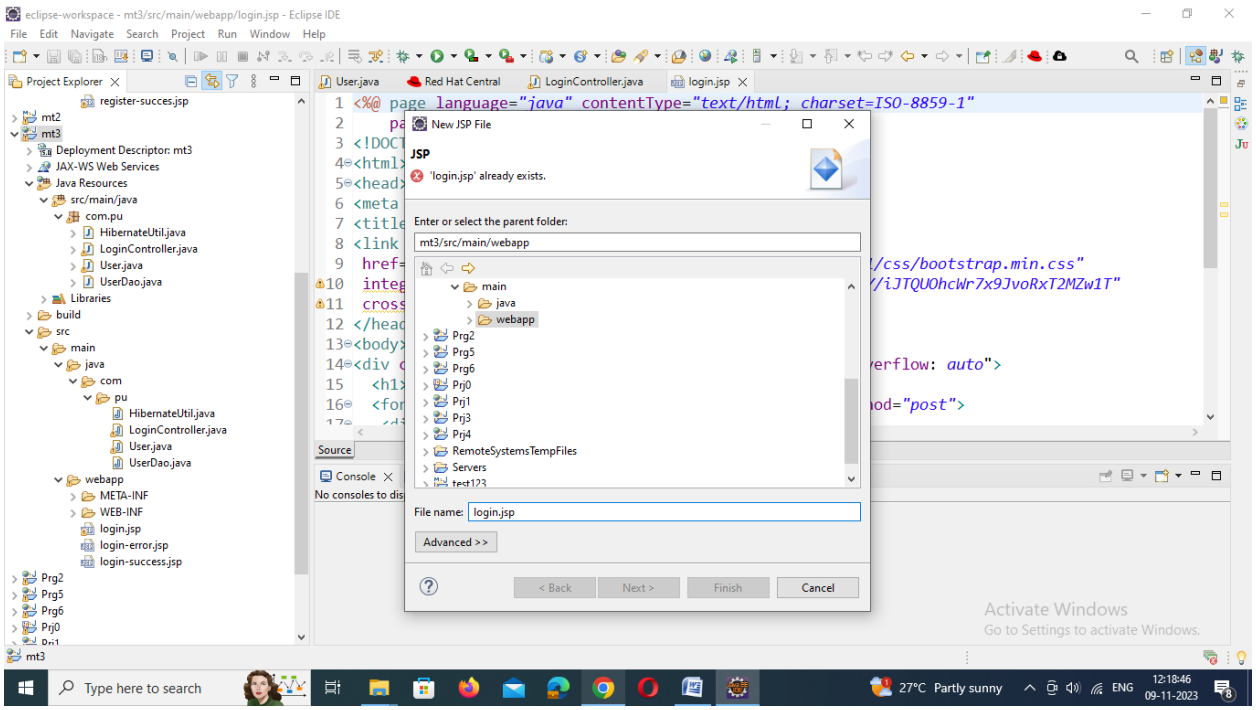
```
            e.printStackTrace();
        }
            }
             private void authenticate(HttpServletRequest request, HttpServletResponse response)
                                    throws Exception {
                                       String username = request.getParameter("username");
                                       String password = request.getParameter("password");

                                       if (loginDao.validate(username, password)) {
                                          jakarta.servlet.RequestDispatcher dispatcher =
request.getRequestDispatcher("login-success.jsp");
                                          dispatcher.forward(request, response);
                                       } else {
                                           jakarta.servlet.RequestDispatcher dispatcher =
request.getRequestDispatcher("login-error.jsp");
                                                   dispatcher.forward(request, response);
                                       }
                                    }
}
```

9. Login form - login.jsp

Next, create a *login.jsp* page under the "WebContent" folder and add the following content to it:

- username
- password



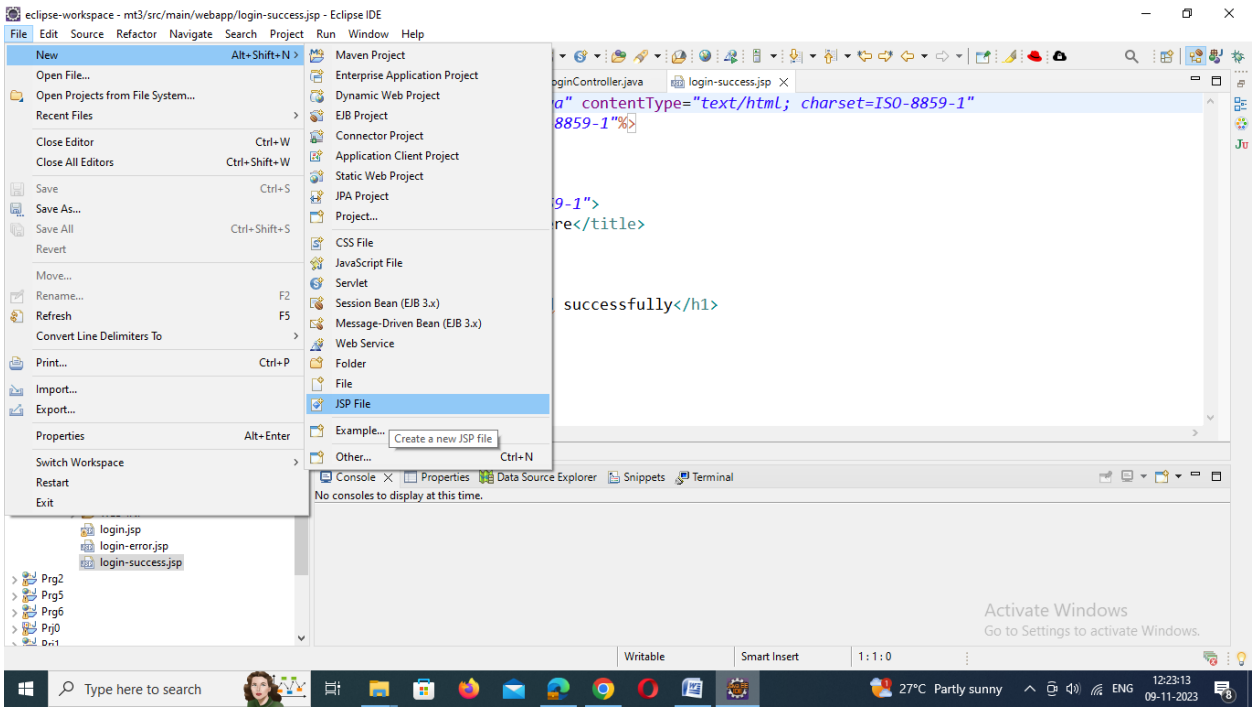


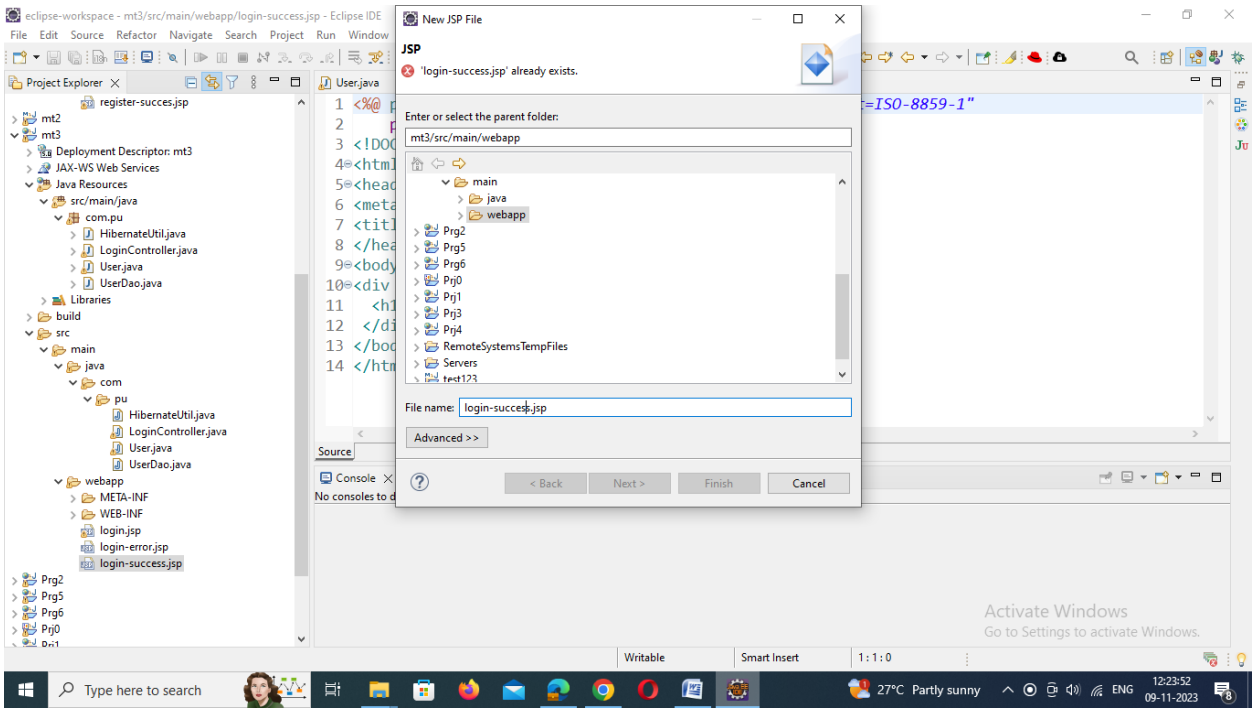

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
```

```
<link rel="stylesheet"
 href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
 integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
 crossorigin="anonymous">
</head>
<body>
<div class="container col-md-8 col-md-offset-3" style="overflow: auto">
  <h1>Login Form</h1>
  <form action="<%=request.getContextPath()%>/login" method="post">
   <div class="form-group">
    <label for="uname">User Name:</label> <input type="text"
     class="form-control" id="username" placeholder="User Name"
     name="username" required>
   </div>
   <div class="form-group">
    <label for="uname">Password:</label> <input type="password"
     class="form-control" id="password" placeholder="Password"
     name="password" required>
   </div>
   <button type="submit" class="btn btn-primary">Submit</button>
  </form>
 </div>
</body>
</html>
```

10. Login success page - login-success.jsp

Next, create a *login-success.jsp* page under the "WebContent" folder and add the following content to it:





11. Login Error page - login-error.jsp

Next, create a *login-error.jsp* page under the "WebContent" folder and add the following content to it:

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

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<div align="center">
  <h1>Login not successful..</h1>
 </div>
</body>
</html>
```

RESULT:
Thus, Creating a login form is a fundamental part of many web applications, and integrating technologies like JSP, Servlet, Hibernate Framework, and MySQL database makes the process both efficient and robust.

Expt12:
Build a complete Hibernate application with HQL CRUD operations using **MAVEN** ,**JSP**, **Servlet**, **Hibernate Framework**, JPQL and **MySQL** database.
AIM:
To build a Hibernate application with HQL CRUD operations using **MAVEN** ,**JSP**, **Servlet**, **Hibernate Framework** and **MySQL** database.

PROCEDURE:

1. Create a Simple Maven Project

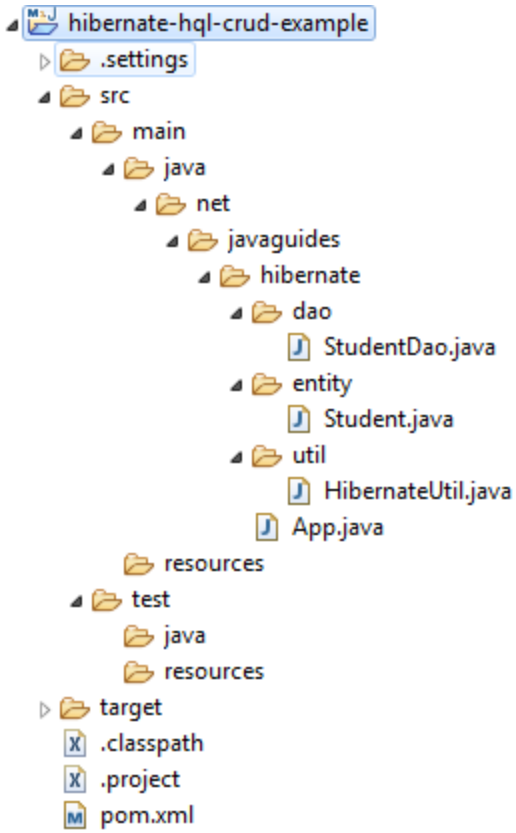
Use the **How to Create a Simple Maven Project in Eclipse** article to create a simple Maven project in Eclipse IDE.

1. Create a Simple Maven Project

Use the How to Create a Simple Maven Project in Eclipse article to create a simple Maven project in Eclipse IDE.

2. Project Directory Structure

The project directory structure for your reference -



3. Add jar Dependencies to pom.xml

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>net.javaguides.hibernate</groupId>
      <artifactId>hibernate-tutorial</artifactId>
      <version>0.0.1-SNAPSHOT</version>
   </parent>
   <artifactId>hibernate-hql-crud-example</artifactId>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
      <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
      <dependency>
         <groupId>mysql</groupId>
```

```
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.32</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.1.7.Final</version>
        </dependency>
    </dependencies>
    <build>
        <sourceDirectory>src/main/java</sourceDirectory>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

4. Creating the JPA Entity Class(Persistent class)

Let's create a Student entity class under net.javaguides.hibernate.entity package with the following code:

```
package net.javaguides.hibernate.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "student")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;
```

```
@Column(name = "last_name")
private String lastName;

@Column(name = "email")
private String email;

public Student() {

}

public Student(String firstName, String lastName, String email) {
   this.firstName = firstName;
   this.lastName = lastName;
   this.email = email;
}

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 getLastName() {
   return lastName;
}

public void setLastName(String lastName) {
   this.lastName = lastName;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
```

```
        this.email = email;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ",
email=" + email + "]";
    }
}
```

5. Create a Hibernate configuration file - Java Configuration

The HibernateUtil Java configuration file contains information about the database and mapping file.
Let's create a **HibernateUtil** file and write the following code in it.

```
package net.javaguides.hibernate.util;

import java.util.Properties;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;

import net.javaguides.hibernate.entity.Student;

public class HibernateUtil {
    private static SessionFactory sessionFactory;
    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                Configuration configuration = new Configuration();

                // Hibernate settings equivalent to hibernate.cfg.xml's properties
                Properties settings = new Properties();
                settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
                settings.put(Environment.URL,
"jdbc:mysql://localhost:3306/hibernate_db?useSSL=false");
                settings.put(Environment.USER, "root");
                settings.put(Environment.PASS, "root");
                settings.put(Environment.SHOW_SQL, "true");

                settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");
```

```
            settings.put(Environment.HBM2DDL_AUTO, "create-drop");

            configuration.setProperties(settings);

            configuration.addAnnotatedClass(Student.class);

            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                .applySettings(configuration.getProperties()).build();

            sessionFactory = configuration.buildSessionFactory(serviceRegistry);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    return sessionFactory;
  }
}
```

Create StudentDao Class - INSERT, UPDATE, SELECT, and DELETE HQL operations

Let's create a separate StudentDao class with the following code:

```
package net.javaguides.hibernate.dao;

import java.util.List;

import jakarta.persistence.Query;

import org.hibernate.Session;
import org.hibernate.Transaction;

import net.javaguides.hibernate.entity.Student;
import net.javaguides.hibernate.util.HibernateUtil;

public class StudentDao {

  public void insertStudent() {
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
      // start a transaction
      transaction = session.beginTransaction();

      String hql = "INSERT INTO Student (firstName, lastName, email) " +
        "SELECT firstName, lastName, email FROM Student";
```

```
            Query query = session.createQuery(hql);
            int result = query.executeUpdate();
            System.out.println("Rows affected: " + result);

            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public void updateStudent(Student student) {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            // start a transaction
            transaction = session.beginTransaction();

            // save the student object
            String hql = "UPDATE Student set firstName = :firstName " + "WHERE id =
:studentId";
            Query query = session.createQuery(hql);
            query.setParameter("firstName", student.getFirstName());
            query.setParameter("studentId", 1);
            int result = query.executeUpdate();
            System.out.println("Rows affected: " + result);

            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public void deleteStudent(int id) {

        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            // start a transaction
            transaction = session.beginTransaction();
```

```
            // Delete a student object
            Student student = session.get(Student.class, id);
            if (student != null) {
                String hql = "DELETE FROM Student " + "WHERE id = :studentId";
                Query query = session.createQuery(hql);
                query.setParameter("studentId", id);
                int result = query.executeUpdate();
                System.out.println("Rows affected: " + result);
            }

            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }

    public Student getStudent(int id) {

        Transaction transaction = null;
        Student student = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            // start a transaction
            transaction = session.beginTransaction();

            // get an student object
            String hql = " FROM Student S WHERE S.id = :studentId";
            Query query = session.createQuery(hql);
            query.setParameter("studentId", id);
            List results = query.getResultList();

            if (results != null && !results.isEmpty()) {
                student = (Student) results.get(0);
            }
            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
        return student;
```

```
    }

    public List < Student > getStudents() {
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            return session.createQuery("from Student", Student.class).list();
        }
    }
}
```

7. Create the main App class and Run an Application

Let's test Hibernate application to connect to the MySQL database.

```
package net.javaguides.hibernate;

import java.util.List;

import net.javaguides.hibernate.dao.StudentDao;
import net.javaguides.hibernate.entity.Student;

public class App {
    public static void main(String[] args) {
        StudentDao studentDao = new StudentDao();
        Student student = new Student("Ramesh", "Fadatare", "rameshfadatare@javaguides.com");
        studentDao.saveStudent(student);
studentDao.insertStudent();

        // update student
        Student student1 = new Student("Ram", "Fadatare", "rameshfadatare@javaguides.com");
        studentDao.updateStudent(student1);

        // get students
        List < Student > students = studentDao.getStudents();
        students.forEach(s - > System.out.println(s.getFirstName()));

        // get single student
        Student student2 = studentDao.getStudent(1);
        System.out.println(student2.getFirstName());

        // delete student
        studentDao.deleteStudent(1);
    }
}
```

Output

```
Nov 24, 2018 4:40:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProvider
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
Nov 24, 2018 4:40:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProvider
INFO: HHH10001005: using driver [com.mysql.cj.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/hiberna
Nov 24, 2018 4:40:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProvider
INFO: HHH10001001: Connection properties: {user=root, password=****}
Nov 24, 2018 4:40:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProvider
INFO: HHH10001003: Autocommit mode: false
Nov 24, 2018 4:40:33 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProvider
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Nov 24, 2018 4:40:33 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
Hibernate: drop table if exists student
Nov 24, 2018 4:40:34 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolato
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.intern
Nov 24, 2018 4:40:34 PM org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolato
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.intern
Hibernate: create table student (id integer not null auto_increment, email varchar(255), first_name va
Nov 24, 2018 4:40:34 PM org.hibernate.tool.schema.internal.SchemaCreatorImpl applyImportSources
INFO: HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.ScriptSourceInputNon
Hibernate: insert into student (email, first_name, last_name) values (?, ?, ?)
Nov 24, 2018 4:40:35 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: insert into student ( first_name, last_name, email ) select student0_.first_name as col_0_0
Rows affected: 1
Hibernate: update student set first_name=? where id=?
Rows affected: 1
Hibernate: select student0_.id as id1_0_, student0_.email as email2_0_, student0_.first_name as first_
Ram
Ramesh
Hibernate: select student0_.id as id1_0_, student0_.email as email2_0_, student0_.first_name as first_
Ram
Hibernate: select student0_.id as id1_0_0_, student0_.email as email2_0_0_, student0_.first_name as fi
Hibernate: delete from student where id=?
Rows affected: 1
```

RESULT:
Thus to create Hibernate application with Hibernate Query Language INSERT, UPDATE, SELECT, and DELETE statements.

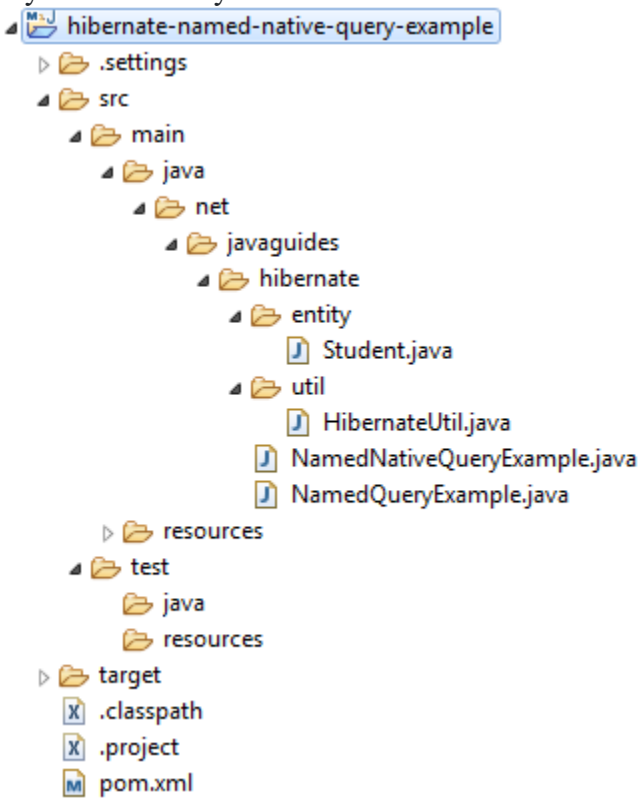
Expt 12a. JPQL

1.Create a Simple Maven Project

Use the **How to Create a Simple Maven Project in Eclipse** article to create a simple Maven project in Eclipse IDE.

2. Project Directory Structure

The project directory structure for your reference -



3. Add jar Dependencies to pom.xml

```
<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/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
```

```
        <groupId>net.javaguides.hibernate</groupId>
        <artifactId>hibernate-tutorial</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>hibernate-named-query-example</artifactId>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.32</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.1.7.Final</version>
        </dependency>
    </dependencies>
    <build>
        <sourceDirectory>src/main/java</sourceDirectory>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Create a Hibernate configuration file - Java Configuration

The *HibernateUtil* Java configuration file contains information about the database and mapping file.
Let's create a *HibernateUtil* file and write the following code in it.

```
package net.javaguides.hibernate.util;
```

```
import java.util.Properties;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;

import net.javaguides.hibernate.entity.Student;

public class HibernateUtil {
   private static SessionFactory sessionFactory;
   public static SessionFactory getSessionFactory() {
      if (sessionFactory == null) {
         try {
            Configuration configuration = new Configuration();

            // Hibernate settings equivalent to hibernate.cfg.xml's properties
            Properties settings = new Properties();
            settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
            settings.put(Environment.URL,
"jdbc:mysql://localhost:3306/hibernate_db?useSSL=false");
            settings.put(Environment.USER, "root");
            settings.put(Environment.PASS, "root");

            settings.put(Environment.SHOW_SQL, "true");

            settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

            settings.put(Environment.HBM2DDL_AUTO, "create-drop");

            configuration.setProperties(settings);

            configuration.addAnnotatedClass(Student.class);

            ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
               .applySettings(configuration.getProperties()).build();

            sessionFactory = configuration.buildSessionFactory(serviceRegistry);
         } catch (Exception e) {
            e.printStackTrace();
         }
      }
      return sessionFactory;
   }
}
```

Using @NamedQuery and @NamedQueries Annotations

Let's create a *Student* entity class and annotate it with the *@NamedQuery* and *@NamedQueries* annotation for using single or multiple-named HQL/JPQL query expressions as follows.

```
package net.javaguides.hibernate.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "student")

//Using @NamedQuery for single JPQL or HQL
@NamedQuery(name = "GET_STUDENTS_COUNT", query = "select count(1) from Student")

//Using @NamedQueries for multiple JPQL or HQL
@NamedQueries({ @NamedQuery(name = "GET_STUDENT_BY_ID", query = "from Student
where id=:id"),
                @NamedQuery(name = "GET_ALL_STUDENTS", query = "from Student")
})
public class Student {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "id")
        private int id;

        @Column(name = "first_name")
        private String firstName;

        @Column(name = "last_name")
        private String lastName;

        @Column(name = "email")
        private String email;

        public Student() {

        }

        public Student(String firstName, String lastName, String email) {
                this.firstName = firstName;
                this.lastName = lastName;
                this.email = email;
        }
```

```
	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 getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	@Override
	public String toString() {
		return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" +
lastName + ", email=" + email + "]";
	}

}
```

Using Named Query for JPQL

Now, create a main class to execute the named query using the *Session.createNamedQuery()* method:

```
package net.javaguides.hibernate;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;

import net.javaguides.hibernate.entity.Student;
import net.javaguides.hibernate.util.HibernateUtil;

public class NamedNativeQueryExample {
   @SuppressWarnings("unchecked")
   public static void main(String[] args) {

      saveStudent();

      Transaction transaction = null;
      try (Session session = HibernateUtil.getSessionFactory().openSession()) {
         transaction = session.beginTransaction();
         // Executing named queries

         List < Object > totalStudents =
session.createNamedQuery("GET_STUDENTS_COUNT").getResultList();
         System.out.println("Total Students: " + totalStudents.get(0));

         List < Object > student =
session.createNamedQuery("GET_STUDENT_BY_ID").setParameter("id", 1).getResultList();

         System.out.println(student.get(0));

         List < Student > students = session.createNamedQuery("GET_ALL_STUDENTS",
Student.class).getResultList();
         for (Student student1: students) {
            System.out.println("ID : " + student1.getId() + " \tNAME : " +
student1.getFirstName());
         }

         transaction.commit();

      } catch (Exception e) {
         if (transaction != null) {
            transaction.rollback();
         }
         e.printStackTrace();
      }
```

```
    }

    private static void saveStudent() {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {

            // start a transaction
            transaction = session.beginTransaction();
            // create new student
            Student student = new Student("Ramesh", "Fadatare",
"rameshfadatare@javaguides.com");
            // save the student object
            session.persist(student);
            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }
    }
}
```

Using @NamedNativeQuery and @NamedNativeQueries Annotations

Let's create a *Student* entity class and annotate it with the *@NamedNativeQuery* and *@NamedNativeQueries* annotation for using single or multiple-named native SQL query expressions as follows.

```
package net.javaguides.hibernate.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "student")

@NamedNativeQuery(name = "GET_STUDENTS_COUNT", query = "select count(1) from
student")

@NamedNativeQueries({
    @NamedNativeQuery(name = "GET_STUDENT_BY_ID", query = "select * from student
where id=:id"),
    @NamedNativeQuery(name = "GET_ALL_STUDENTS", query = "select * from student",
resultClass = Student.class)
```

```
})
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    public Student() {

    }

    public Student(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    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 getLastName() {
        return lastName;
    }
```

```
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ",
email=" + email + "]";
    }
}
```

Note that we used *@NamedNativeQuery* and *@NamedNativeQueries* annotations to define named queries with native SQL statements.

Using Named Query for Native SQL Statements

```
package net.javaguides.hibernate;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;

import net.javaguides.hibernate.entity.Student;
import net.javaguides.hibernate.util.HibernateUtil;

public class NamedNativeQueryExample {
    @SuppressWarnings("unchecked")
    public static void main(String[] args) {

        saveStudent();

        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {
            transaction = session.beginTransaction();
            // Executing named queries
```

```
            List < Object > totalStudents =
session.createNamedQuery("GET_STUDENTS_COUNT").getResultList();
            System.out.println("Total Students: " + totalStudents.get(0));

            List < Object > student =
session.createNamedQuery("GET_STUDENT_BY_ID").setParameter("id", 1).getResultList();

            System.out.println(student.get(0));

            List < Student > students = session.createNamedQuery("GET_ALL_STUDENTS",
Student.class).getResultList();
            for (Student student1: students) {
                System.out.println("ID : " + student1.getId() + " \tNAME : " +
student1.getFirstName());
            }

            transaction.commit();

        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
        }

    }

    private static void saveStudent() {
        Transaction transaction = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession()) {

            // start a transaction
            transaction = session.beginTransaction();
            // create new student
            Student student = new Student("Ramesh", "Fadatare",
"rameshfadatare@javaguides.com");
            // save the student object
            session.persist(student);
            // commit transaction
            transaction.commit();
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
```

```
        }
    }
}
```

Expt13:

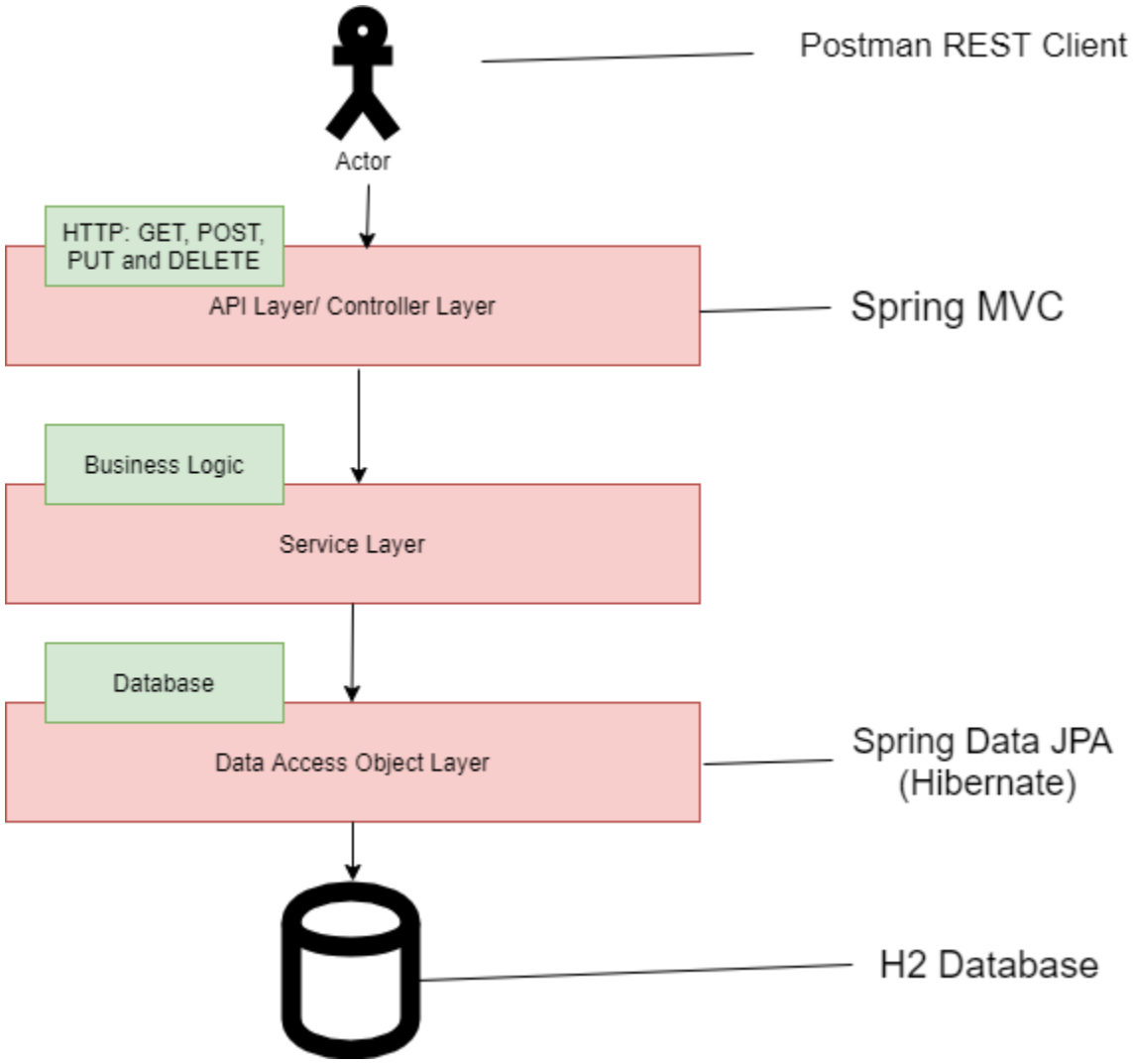
Build CRUD RESTful API using Spring Boot 3, Spring Data JPA (Hibernate), and MySQL database.

AIM:

To build a simple **User Management Application** which has CRUD Rest APIs. Following five REST APIs (Controller handler methods) are created for User resource:
has below CRUD Rest APIs

Sr. No.	API Name	HTTP Method	Path	Status Code	Description
(1)	GET Users	GET	`/api/v1/users`	200 (OK)	All User resources are fetched.
(2)	POST User	POST	`/api/v1/users`	201 (Created)	A new User resource is created.
(3)	GET User	GET	`/api/v1/users/{id}`	200 (OK)	One User resource is fetched.
(4)	PUT User	PUT	`/api/v1/users/{id}`	200 (OK)	User resource is updated.
(5)	DELETE User	DELETE	`/api/v1/users/{id}`	204 (No Content)	User resource is deleted.

High-level architecture of Spring boot project



1. Creating a Spring Boot Application

There are many ways to create a Spring Boot application. You can refer below articles to create a Spring Boot application.
>> **Create Spring Boot Project With Spring Initializer**
>> **Create Spring Boot Project in Spring Tool Suite [STS]**
Refer to the next step to create a project packaging structure.

2. Maven dependencies

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

3. Configuring MySQL Database

Since we're using MySQL as our database, we need to configure the database **URL**, **username**, and **password** so that Spring can establish a connection with the database on startup.
Open *src/main/resources/application.properties* file and add the following properties to it:

```
spring.datasource.url = jdbc:mysql://localhost:3306/usersDB?useSSL=false
spring.datasource.username = root
spring.datasource.password = root


## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

Don't forget to change the *spring.datasource.username* and *spring.datasource.password* as per your MySQL installation. Also, create a database named **usersDB** in MySQL before proceeding to the next section.
You don't need to create any tables. The tables will automatically be created by hibernate from the *User* entity that we will define in the next step. This is made possible by the property *spring.jpa.hibernate.ddl-auto = update.*

4. Create User JPA Entity

Let's create a *User* model or domain class with the following fields:

- id - primary key
- firstName - user first name
- lastName - user last name
- emailId - user email ID

```
package net.javaguides.springboot.entity;

import jakarta.persistence.*;

@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "email")
    private String email;

    public User() {

    }

    public User(String firstName, String lastName, String email) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public long getId() {
        return id;
    }
    public void setId(long id) {
```

```
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

All your domain models must be annotated with *@Entity* annotation. It is used to mark the class as a persistent Java class.
@Table annotation is used to provide the details of the table that this entity will be mapped to.
@Id annotation is used to define the primary key.
@GeneratedValue annotation is used to define the primary key generation strategy. In the above case, we have declared the primary key to be an Auto Increment field.
@Column annotation is used to define the properties of the column that will be mapped to the annotated field. You can define several properties like name, length, nullable, updateable, etc.

5. Define UserRepository

Let's create a *UserRepository* to access User's data from the database.
Well, Spring Data JPA has comes with a *JpaRepository* interface which defines methods for all the CRUD operations on the entity, and a default implementation of *JpaRepository* called *SimpleJpaRepository*.

```
package net.javaguides.springboot.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import net.javaguides.springboot.entity.User;

@Repository
```

```
public interface UserRepository extends JpaRepository<User, Long>{

}
```

6. Creating Custom Business Exception

We'll define the Rest APIs for creating, retrieving, updating, and deleting a *User* in the next step. The APIs will throw a *ResourceNotFoundException* whenever a *User* with a given id is not found in the database.
Following is the definition of *ResourceNotFoundException*:

```
package net.javaguides.springboot.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1 L;

    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

7. Creating UserController - Building CRUD Rest APIs

Let's create the REST APIs for creating, retrieving, updating and deleting a *User*:

```
package net.javaguides.springboot.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import net.javaguides.springboot.entity.User;
import net.javaguides.springboot.exception.ResourceNotFoundException;
import net.javaguides.springboot.repository.UserRepository;

@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserRepository userRepository;

    // get all users
    @GetMapping
    public List < User > getAllUsers() {
        return this.userRepository.findAll();
    }

    // get user by id
    @GetMapping("/{id}")
    public User getUserById(@PathVariable(value = "id") long userId) {
        return this.userRepository.findById(userId)
            .orElseThrow(() - > new ResourceNotFoundException("User not found with id :" +
userId));
    }

    // create user
    @PostMapping
    public User createUser(@RequestBody User user) {
        return this.userRepository.save(user);
    }

    // update user
    @PutMapping("/{id}")
    public User updateUser(@RequestBody User user, @PathVariable("id") long userId) {
        User existingUser = this.userRepository.findById(userId)
            .orElseThrow(() - > new ResourceNotFoundException("User not found with id :" +
userId));
        existingUser.setFirstName(user.getFirstName());
        existingUser.setLastName(user.getLastName());
        existingUser.setEmail(user.getEmail());
        return this.userRepository.save(existingUser);
    }

    // delete user by id
    @DeleteMapping("/{id}")
```

```
    public ResponseEntity < User > deleteUser(@PathVariable("id") long userId) {
        User existingUser = this.userRepository.findById(userId)
            .orElseThrow(() - > new ResourceNotFoundException("User not found with id :" +
userId));
        this.userRepository.delete(existingUser);
        return ResponseEntity.ok().build();
    }
}
```

8. Running the Application

We have successfully developed all the CRUD Rest APIs for the *User* model. Now it's time to deploy our application in a servlet container(embedded tomcat).
Two ways we can start the standalone Spring boot application.
1. From the root directory of the application and type the following command to run it -

```
$ mvn spring-boot:run
```

2. From your IDE, run the SpringBootCrudRestApplication.main() method as a standalone Java class that will start the embedded Tomcat server on port 8080 and point the browser to **http://localhost:8080/**.

RESULT:
Thus, successfully built a Restful CRUD API using Spring Boot, MySQL, JPA, and Hibernate.

Expt14:

AIM:
To build login or sign-in and registration or signup REST API using Spring boot, Spring Security, Hibernate, and MySQL database.

PROCEDURE:

1. Create a Spring boot application

Spring Boot provides a web tool called **Spring Initializer** to bootstrap an application quickly. Just go to **https://start.spring.io/** and generate a new spring boot project.

Use the below details in the Spring boot creation:

Project Name: springboot-blog-rest-api

Project Type: Maven

Choose dependencies: Spring Web, Lombok, Spring Data JPA, Spring Security, Dev Tools and MySQL Driver

Package name: net.javaguides.springboot

Packaging: Jar

Download the Spring Boot project as a zip file, unzip it and import it in your favorite IDE.

2. Maven Dependencies

Here is the pom.xml file for your reference:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.0.0</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.springboot.blog</groupId>
	<artifactId>springboot-blog-rest-api</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>springboot-blog-rest-api</name>
	<description>Spring boot blog application rest api&amp;apos;s</description>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
```

```
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-devtools</artifactId>
                <scope>runtime</scope>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <optional>true</optional>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-validation</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-security</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
            </dependency>
    </dependencies>

    <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                    <configuration>
                        <excludes>
                            <exclude>

    <groupId>org.projectlombok</groupId>

    <artifactId>lombok</artifactId>
                            </exclude>
                        </excludes>
                    </configuration>
                </plugin>
```

```
            </plugins>
        </build>

</project>
```

3. Configure MySQL Database

Let's first create a database in MySQL server using the below command:

```
create database myblog
```

Since we're using MySQL as our database, we need to configure the database **URL**, **username**, and **password** so that Spring can establish a connection with the database on startup.
Open src/main/resources/application.properties file and add the following properties to it:

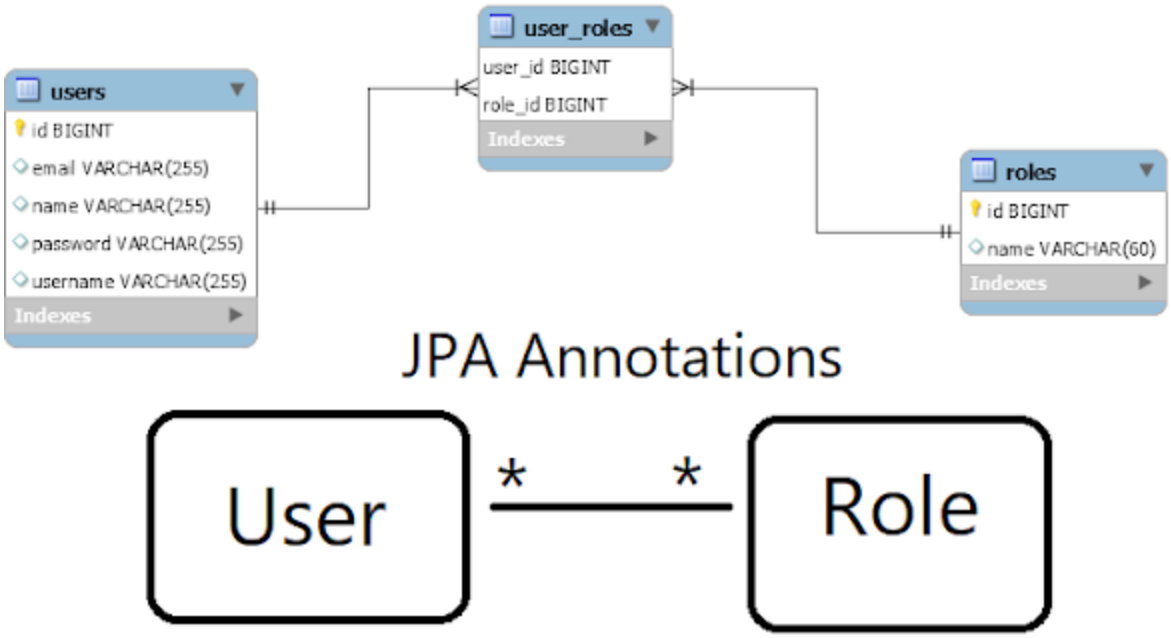
```
spring.datasource.url =
jdbc:mysql://localhost:3306/myblog?useSSL=false&serverTimezone=UTC
spring.datasource.username = root
spring.datasource.password = root

# hibernate properties
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

logging.level.org.springframework.security=DEBUG
```

Don't forget to change the spring.datasource.username and spring.datasource.password as per your MySQL installation.

You don't need to create any tables. The tables will automatically be created by Hibernate from the Employee entity that we will define in the next step. This is made possible by the property spring.jpa.hibernate.ddl-auto = update.

4. Model Layer - Create JPA Entities

In this step, we will create *User* and *Role* JPA entities and establish **MANY-to-MANY** relationships between them. Let's use JPA annotations to establish **MANY-to-MANY** relationships between *User* and *Role* entities.



User JPA Entity

```
package com.springboot.blog.entity;

import lombok.Data;

import jakarta.persistence.*;
import java.util.Set;

@Data
@Entity
@Table(name = "users", uniqueConstraints = {
        @UniqueConstraint(columnNames = {"username"}),
        @UniqueConstraint(columnNames = {"email"})
})
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String name;
    private String username;
    private String email;
    private String password;
```

```
    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id", referencedColumnName = "id"),
        inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName = "id"))
    private Set<Role> roles;
}
```

Role JPA Entity

```
package com.springboot.blog.entity;

import lombok.Getter;
import lombok.Setter;

import jakarta.persistence.*;

@Setter
@Getter
@Entity
@Table(name = "roles")
public class Role {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    @Column(length = 60)
    private String name;
}
```

5. Repository Layer

UserRepository

```
package com.springboot.blog.repository;

import com.springboot.blog.entity.User;
import org.springframework.data.domain.Example;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;
```

```
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
    Optional<User> findByUsernameOrEmail(String username, String email);
    Optional<User> findByUsername(String username);
    Boolean existsByUsername(String username);
    Boolean existsByEmail(String email);
}
```

RoleRepository

```
package com.springboot.blog.repository;

import com.springboot.blog.entity.Role;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface RoleRepository extends JpaRepository<Role, Long> {
    Optional<Role> findByName(String name);
}
```

6. Service Layer - CustomUserDetailsService

Let's write a logic to load user details by name or email from the database.

Let's create a *CustomUserDetailsService* which implements the *UserDetailsService* interface (Spring security in-build interface) and provides an implementation for the *loadUserByUername()* method:

```
import com.springboot.blog.entity.User;
import com.springboot.blog.repository.UserRepository;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Set;
import java.util.stream.Collectors;

@Service
public class CustomUserDetailsService implements UserDetailsService {
```

```
    private UserRepository userRepository;

    public CustomUserDetailsService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String usernameOrEmail) throws
UsernameNotFoundException {
        User user = userRepository.findByUsernameOrEmail(usernameOrEmail,
usernameOrEmail)
                .orElseThrow(() ->
                        new UsernameNotFoundException("User not found with username or email: "+
usernameOrEmail));

        Set<GrantedAuthority> authorities = user
                .getRoles()
                .stream()
                .map((role) -> new
SimpleGrantedAuthority(role.getName())).collect(Collectors.toSet());

        return new org.springframework.security.core.userdetails.User(user.getEmail(),
                user.getPassword(),
                authorities);
    }
}
```

Spring Security uses the UserDetailsService interface, which contains the loadUserByUsername(String username) method to look up UserDetails for a given username.

The UserDetails interface represents an authenticated user object and Spring Security provides an out-of-the-box implementation of org.springframework.security.core.userdetails.User.

7. Spring Security Configuration

Let's create a class *SecurityConfig* and add the following configuration to it:

```
package com.springboot.blog.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
```

```
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConf
iguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    private UserDetailsService userDetailsService;

    public SecurityConfig(UserDetailsService userDetailsService){
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public static PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authenticationManager(
                            AuthenticationConfiguration configuration) throws Exception {
        return configuration.getAuthenticationManager();
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http.csrf().disable()
                .authorizeHttpRequests((authorize) ->
                        //authorize.anyRequest().authenticated()
                        authorize.requestMatchers(HttpMethod.GET, "/api/**").permitAll()
                                .requestMatchers("/api/auth/**").permitAll()
                                .anyRequest().authenticated()
```

```
            );

        return http.build();
    }
}
```

In Spring Security 5.6, we can enable annotation-based security using the *@EnableMethodSecurity* annotation on any *@Configuration* instance. *@EnableMethodSecurity* enables @PreAuthorize, @PostAuthorize, @PreFilter, and @PostFilter by default.

We are allowing anyone to access login REST API with the below security configuration:

```
authorize.requestMatchers(HttpMethod.GET, "/api/**").permitAll()
```

We are using the Spring security provided *BCryptPasswordEncoder* class to encrypt the passwords.

8. DTO or Payload Classes

Let's create DTO classes to transfer data or payload between client and server and vice-versa.

LoginDto

```
package com.springboot.blog.payload;

import lombok.Data;

@Data
public class LoginDto {
    private String usernameOrEmail;
    private String password;
}
```

SignUpDto

```
package com.springboot.blog.payload;

import lombok.Data;

@Data
public class SignUpDto {
    private String name;
```

```
    private String username;
    private String email;
    private String password;
}
```

9. Controller Layer - Login/Sign-in and Register/SignUp REST APIs

Now it's time to code Login/Sign-in and Register/SignUp REST APIs. Let's create a class *AuthController* and add the following code to it:

```
package com.springboot.blog.controller;

import com.springboot.blog.entity.Role;
import com.springboot.blog.entity.User;
import com.springboot.blog.payload.LoginDto;
import com.springboot.blog.payload.SignUpDto;
import com.springboot.blog.repository.RoleRepository;
import com.springboot.blog.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Collections;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RoleRepository roleRepository;
```

```
    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostMapping("/signin")
    public ResponseEntity<String> authenticateUser(@RequestBody LoginDto loginDto){
        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                loginDto.getUsernameOrEmail(), loginDto.getPassword()));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        return new ResponseEntity<>("User signed-in successfully!.", HttpStatus.OK);
    }

    @PostMapping("/signup")
    public ResponseEntity<?> registerUser(@RequestBody SignUpDto signUpDto){

        // add check for username exists in a DB
        if(userRepository.existsByUsername(signUpDto.getUsername())){
            return new ResponseEntity<>("Username is already taken!",
HttpStatus.BAD_REQUEST);
        }

        // add check for email exists in DB
        if(userRepository.existsByEmail(signUpDto.getEmail())){
            return new ResponseEntity<>("Email is already taken!", HttpStatus.BAD_REQUEST);
        }

        // create user object
        User user = new User();
        user.setName(signUpDto.getName());
        user.setUsername(signUpDto.getUsername());
        user.setEmail(signUpDto.getEmail());
        user.setPassword(passwordEncoder.encode(signUpDto.getPassword()));

        Role roles = roleRepository.findByName("ROLE_ADMIN").get();
        user.setRoles(Collections.singleton(roles));

        userRepository.save(user);

        return new ResponseEntity<>("User registered successfully", HttpStatus.OK);

    }
}
```

Login/Sign in REST API:

```
    @PostMapping("/signin")
    public ResponseEntity<String> authenticateUser(@RequestBody LoginDto loginDto){
        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                loginDto.getUsernameOrEmail(), loginDto.getPassword()));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        return new ResponseEntity<>("User signed-in successfully!.", HttpStatus.OK);
    }
```

Register/SignUp REST API:

```
    @PostMapping("/signup")
    public ResponseEntity<?> registerUser(@RequestBody SignUpDto signUpDto){

        // add check for username exists in a DB
        if(userRepository.existsByUsername(signUpDto.getUsername())){
            return new ResponseEntity<>("Username is already taken!",
HttpStatus.BAD_REQUEST);
        }

        // add check for email exists in DB
        if(userRepository.existsByEmail(signUpDto.getEmail())){
            return new ResponseEntity<>("Email is already taken!", HttpStatus.BAD_REQUEST);
        }

        // create user object
        User user = new User();
        user.setName(signUpDto.getName());
        user.setUsername(signUpDto.getUsername());
        user.setEmail(signUpDto.getEmail());
        user.setPassword(passwordEncoder.encode(signUpDto.getPassword()));

        Role roles = roleRepository.findByName("ROLE_ADMIN").get();
        user.setRoles(Collections.singleton(roles));

        userRepository.save(user);

        return new ResponseEntity<>("User registered successfully", HttpStatus.OK);

    }
```

10. Run Spring Boot Application

We have successfully developed Login and Registration Rest APIs.
Now it's time to deploy our application in a servlet container(embedded tomcat).
Two ways we can start the standalone Spring boot application.
1. From the root directory of the application and type the following command to run it -

```
$ mvn spring-boot:run
```

2. From your IDE, run the Application.main() method as a standalone Java class that will start the embedded Tomcat server on port 8080 and point the browser to **http://localhost:8080/**.

Important

Once you start the Spring boot application, make sure that you add the role **ROLE_ADMIN** record in the *roles* table.

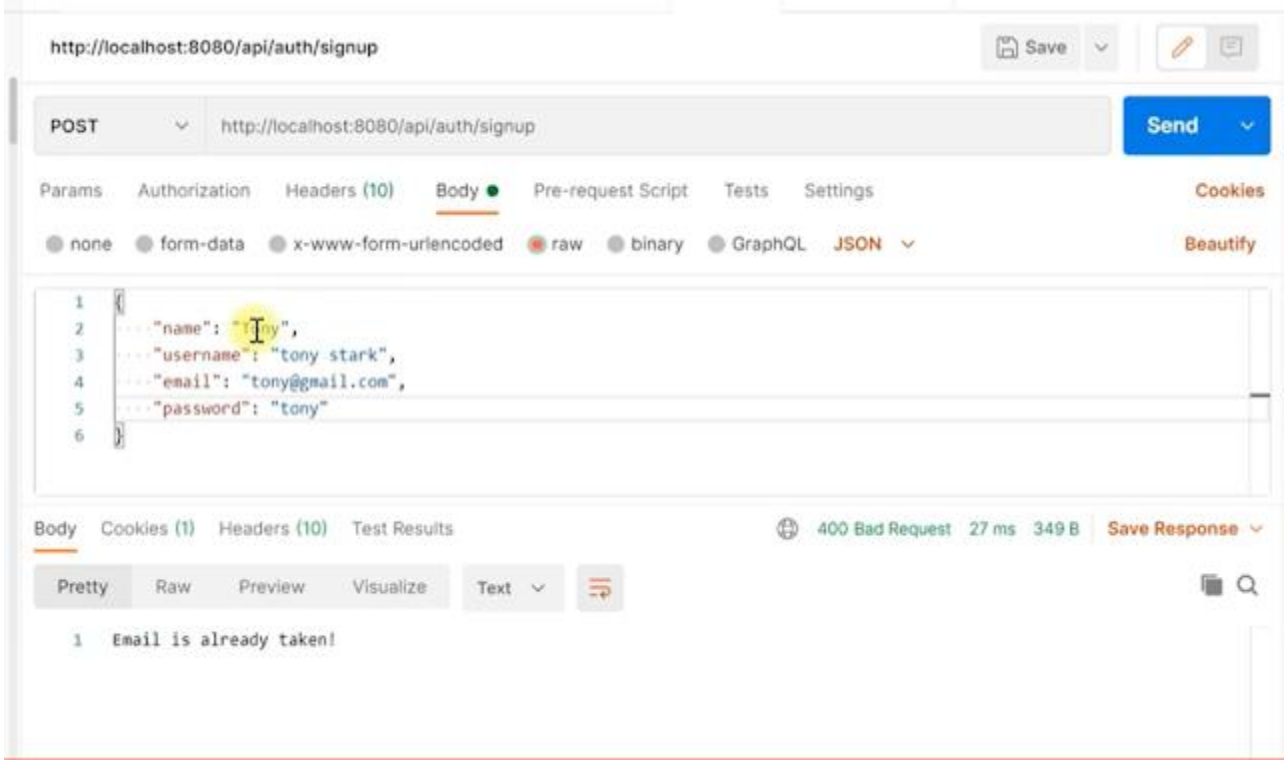
Execute the below INSERT SQL statements to add the **ROLE_ADMIN** role to the *roles* database table:

```
INSERT INTO roles VALUES('ROLE_ADMIN');
```

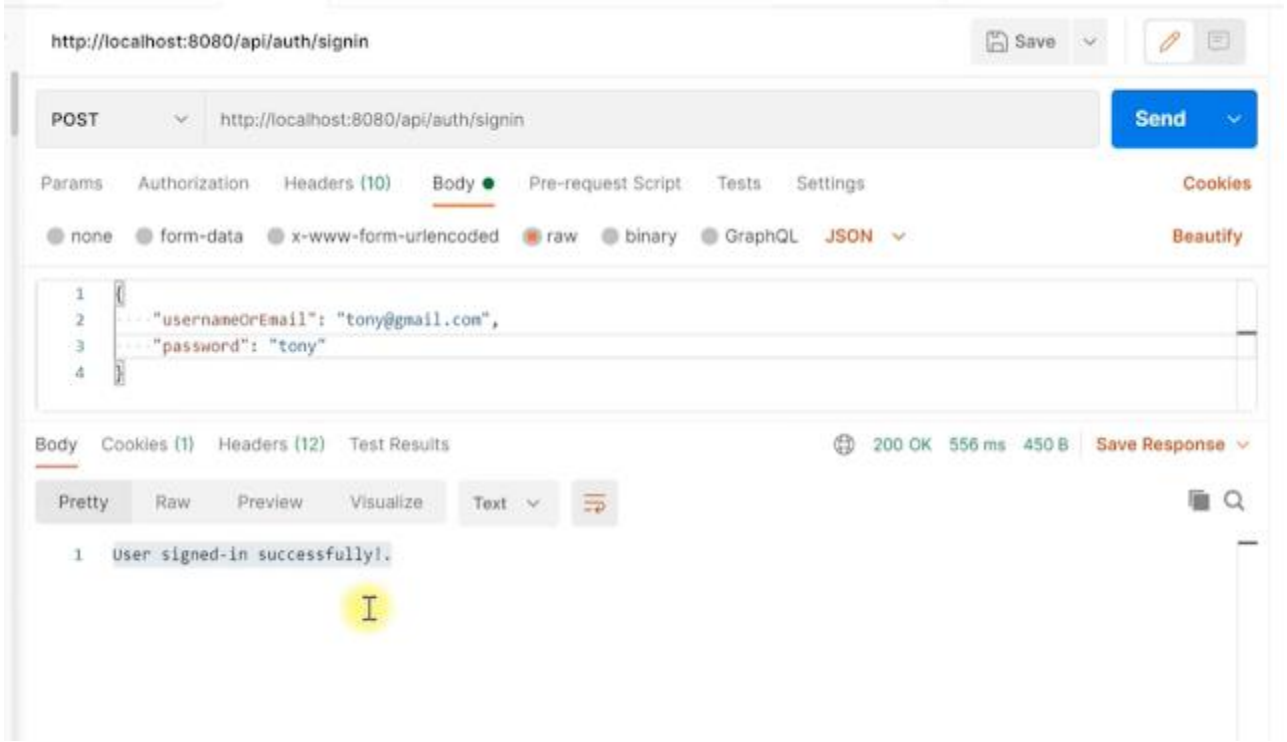
11. Test using Postman

Refer to the below screenshots to test Login and Registration REST API using Postman:

SignUp REST API:



SignIn/Login REST API:



Expt15:

AIM:
To create a simple Spring MVC web application using Java-based configuration that is we configure the Spring `DispatcherServlet` and spring beans configuration using all Java Code (no XML).

PROCEDURE:

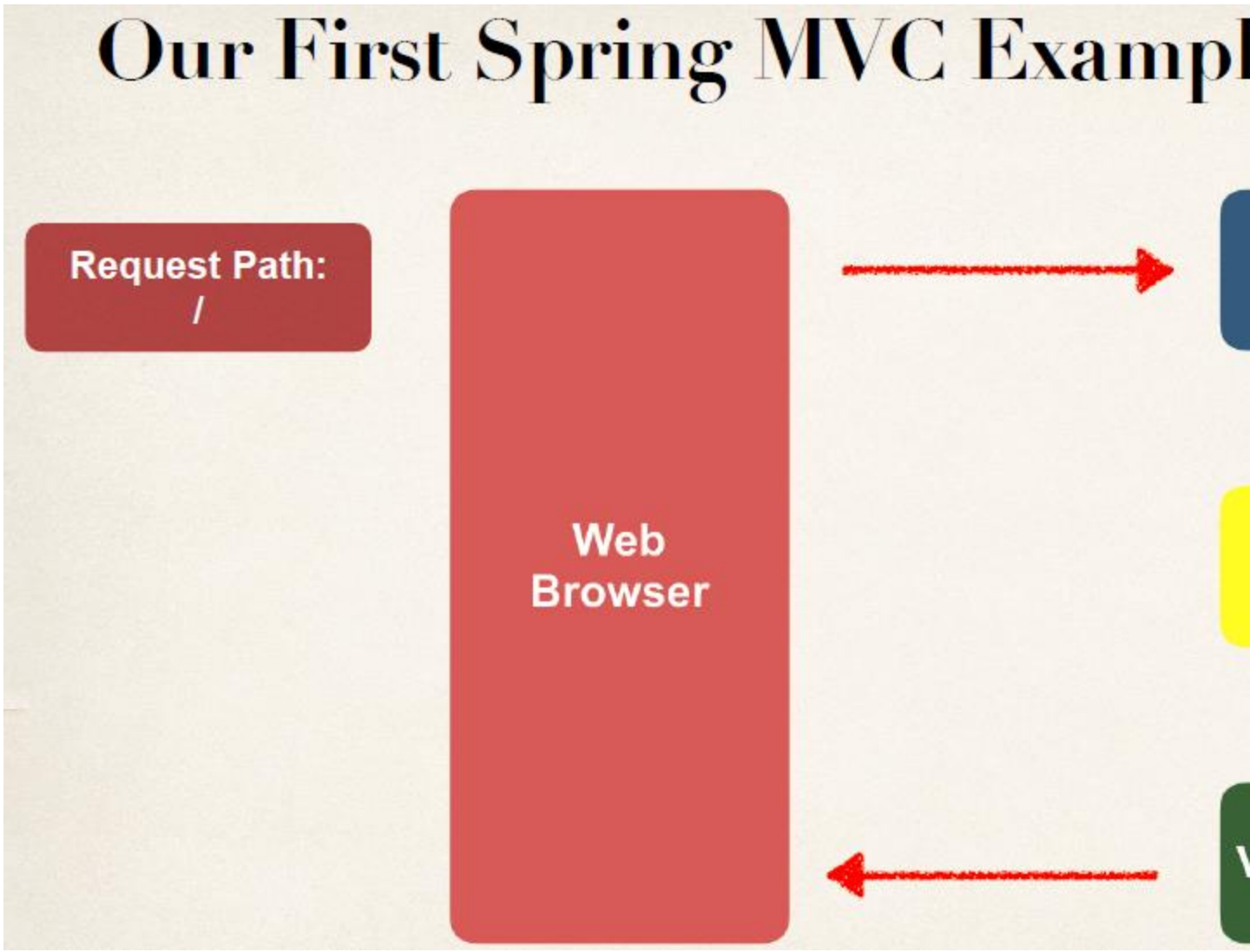
Tools and technologies used

Here are the tools and technologies that we will be using to build a simple Spring MVC web application:

- Spring MVC - 5.1.0 RELEASE
- JDK - 1.8 or later
- Maven - 3.5.1
- Apache Tomcat - 8.5
- IDE - STS/Eclipse Neon.3
- JSTL - 1.2.1

Spring MVC Hello World Application Flow

The following image shows the typical request-response flow in Spring MVC application:



Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet −

- After receiving an HTTP request, The *Front controller* or *DispatcherServlet* consults the *HandlerMapping* to call the appropriate Controller.
- The Controller takes the request and calls the appropriate service methods based on the used GET or POST method. The service method will set model data based on defined business logic and returns view name to the *DispatcherServlet*.
- The *DispatcherServlet* will take help from *ViewResolver* to pick up the defined view for the request.
- Once a view is finalized, The *DispatcherServlet* passes the model data to the view, which is finally rendered, on the browsers.

Development Steps

1. Create a Maven Web Application
2. Add Dependencies - pom.xml File
3. Project Structure
4. Spring Configuration - AppConfig.java
5. Servlet Container Initialization - MySpringMvcDispatcherServletInitializer.java
6. Model Class - HelloWorld.java
7. Controller Class - HelloWorldController.java
8. View - helloworld.jsp
9. Build + Deploy + Run an application
10. Demo

1. Create a Maven Web Application

Let's create a Maven-based web application either using a command line or from Eclipse IDE.
1. Use **Guide to Create a Maven Web Application** link to create a maven project using a command line.
2. Use **Create Maven Web Application using the Eclipse IDE** link to create a maven web application using IDE Eclipse.
Once you created a maven web application, refer below pom.xml file jar dependencies.

2. Add Dependencies - pom.xml File

Refer the following pom.xml file and add jar dependencies to your pom.xml.

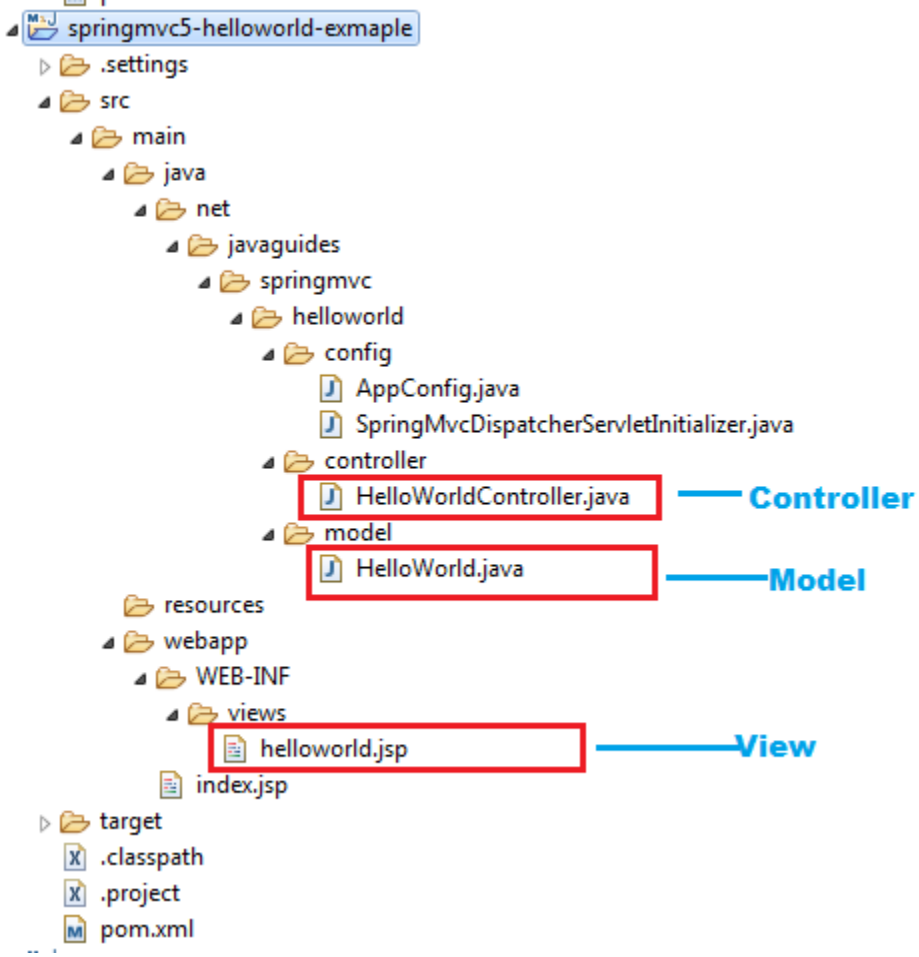
```
<project xmlns="https://maven.apache.org/POM/4.0.0"
 xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>net.javaguides.springmvc</groupId>
   <artifactId>springmvc5-helloworld-exmaple</artifactId>
   <packaging>war</packaging>
   <version>0.0.1-SNAPSHOT</version>
   <name>springmvc5-helloworld-exmaple Maven Webapp</name>
   <url>http://maven.apache.org</url>
   <properties>
     <failOnMissingWebXml>false</failOnMissingWebXml>
   </properties>
   <dependencies>
     <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
     <dependency>
       <groupId>org.springframework</groupId>
       <artifactId>spring-webmvc</artifactId>
       <version>5.1.0.RELEASE</version>
```

```
        </dependency>

        <!-- JSTL Dependency -->
        <dependency>
            <groupId>javax.servlet.jsp.jstl</groupId>
            <artifactId>javax.servlet.jsp.jstl-api</artifactId>
            <version>1.2.1</version>
        </dependency>
        <dependency>
            <groupId>taglibs</groupId>
            <artifactId>standard</artifactId>
            <version>1.1.2</version>
        </dependency>

        <!-- Servlet Dependency -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>

        <!-- JSP Dependency -->
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
            <version>2.3.1</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <sourceDirectory>src/main/java</sourceDirectory>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Note that we are using **spring-webmv**c dependency for a Spring MVC web application.
Next, let's create a standard project structure, please refer below diagram.

3. Project Structure

Standard project structure for your reference -



As the name suggests Spring MVC, look at the diagram we are using the Model-View-Controller approach.
Model - HelloWorld.java
View - helloworld.java
Controller - HelloWorldController.java
Next step, we will configure Spring beans using Java-based configuration.

4. Spring Configuration - AppConfig.java

Create an AppConfig class and annotated with **@Configuration**, @EnableWebMvc, and @ComponentScan annotations as follows.

```
package net.javaguides.springmvc.helloworld.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

/**
 * @author Ramesh Fadatare
 */

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = {
   "net.javaguides.springmvc.helloworld"
})
public class AppConfig {

   @Bean
   public InternalResourceViewResolver resolver() {
      InternalResourceViewResolver resolver = new InternalResourceViewResolver();
      resolver.setViewClass(JstlView.class);
      resolver.setPrefix("/WEB-INF/views/");
      resolver.setSuffix(".jsp");
      return resolver;
   }
}
```

Let's understand few important annotations as below:

- The **@Configuration** is a class-level annotation indicating that an object is a source of bean definitions.
- The @EnableWebMvc enables default Spring MVC configuration and provides the functionality equivalent to **mvc:annotation-driven/** element in XML based configuration.
- The @ComponentScan scans the stereotype annotations (@Controller, @Service etc...) in a package specified by basePackages attribute.

InternalResourceViewResolver

This ViewResolver allows us to set properties such as prefix or suffix to the view name to generate the final view page URL:

```
   @Bean
   public InternalResourceViewResolver resolver() {
      InternalResourceViewResolver resolver = new InternalResourceViewResolver();
      resolver.setViewClass(JstlView.class);
      resolver.setPrefix("/WEB-INF/views/");
      resolver.setSuffix(".jsp");
```

```
        return resolver;
    }
```

We only need a simple JSP page, placed in the /WEB-INF/view folder. In this example, we have used helloworld.jsp as the view - **/WEB-INF/views/helloworld.jsp.**

5. Servlet Container Initialization - SpringMvcDispatcherServletInitializer.java

Let's configure Spring MVC DispatcherServlet and set up URL mappings to Spring MVC DispatcherServlet.
Create a SpringMvcDispatcherServletInitializer class by extending the AbstractAnnotationConfigDispatcherServletInitializer class as follows.

```
package net.javaguides.springmvc.helloworld.config;

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

/**
 * @author Ramesh Fadatare
 */
public class SpringMvcDispatcherServletInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class <?> [] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class <?> [] getServletConfigClasses() {
        return new Class[] {
            AppConfig.class
        };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] {
            "/"
        };
    }
}
```

6. Model Class - HelloWorld.java

As we know that Spring MVC uses a model-view-controller design pattern. Let's create a model - HelloWorld.java with message and DateTime fields. This model data will be rendered on a view (JSP file).

```
package net.javaguides.springmvc.helloworld.model;

public class HelloWorld {
   private String message;
   private String dateTime;
   public String getMessage() {
      return message;
   }
   public void setMessage(String message) {
      this.message = message;
   }
   public String getDateTime() {
      return dateTime;
   }
   public void setDateTime(String dateTime) {
      this.dateTime = dateTime;
   }
}
```

Next, we will create a Controller to handle a request, populate the model with some data and returns the view to a browser.

7. Controller Class - HelloWorldController.java

Create a HelloWorldController class annotated with @Controller annotation under net.javaguides.springmvc.helloworld.controller package and write the following code in it.

```
package net.javaguides.springmvc.helloworld.controller;

import java.time.LocalDateTime;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

import net.javaguides.springmvc.helloworld.model.HelloWorld;

/**
 * @author Ramesh Fadatare
 */
```

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloworld")
    public String handler(Model model) {

        HelloWorld helloWorld = new HelloWorld();
        helloWorld.setMessage("Hello World Example Using Spring MVC 5!!!");
        helloWorld.setDateTime(LocalDateTime.now().toString());
        model.addAttribute("helloWorld", helloWorld);
        return "helloworld";
    }
}
```

8. View - helloworld.jsp

Let's create an helloworld.jsp file under src/main/webapp/WEB-INF/views folder and write the following code in it.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head><%@ page isELIgnored="false" %>
<meta charset="ISO-8859-1">
<title>Spring 5 MVC - Hello World Example | javaguides.net</title>
</head>
  <body>
    <h2>${helloWorld.message}</h2>
    <h4>Server date time is : ${helloWorld.dateTime}</h4>
  </body>
</html>
```

9. Build + Deploy + Run an application

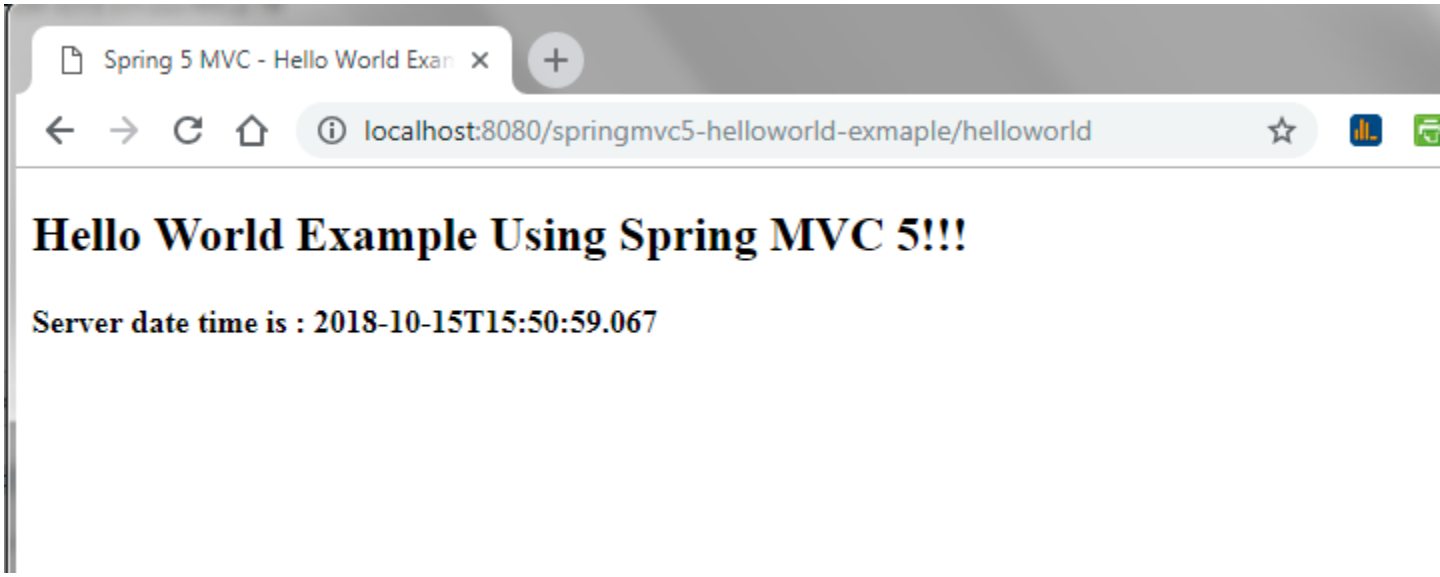
As we are using maven build tool so first, we will need to build this application using the following maven command:

```
clean install
```

Once build success, then we will run this application on tomcat server 8.5 in IDE or we can also deploy war file on the external tomcat **webapps** folder and run the application.

10. Demo

Once an application is up and running in tomcat then hit this link into browser: **http://localhost:8080/springmvc5-helloworld-exmaple/helloworld**
We will see below the page on the browser with the current date-time and a message from a model.



RESULT:
Thus developed a very simple Spring MVC web application successfully.