EXPERIMENT NO. 09

Title: Develop a Web application using Hibernate framework.

Outcome: Students will be able to develop a Web application using Hibernate framework

Theory:

Prerequisites: Java 8 (1.8.0_261), Eclipse (with Dynamic Web Project support), Apache Tomcat 8.5/9, MySQL server, Hibernate 5.x JARs (or Maven), MySQL Connector/J.

---

0. Learning outcomes

- Create a Dynamic Web Project in Eclipse.
- Configure Hibernate (XML mapping approach).
- Persist Java objects to MySQL using Hibernate.
- Deploy and test a servlet/JSP web app on Tomcat.

---

1. Setup

1. Start MySQL server and Tomcat.
2. In Eclipse: File → New → **Dynamic Web Project**.
   - Project name: DemoHibernate
   - Target runtime: **Apache Tomcat 8.5/9**
   - Dynamic web module version: **3.1** (or compatible)
   - Finish.
3. Project layout we'll use (Maven-like but without Maven):

```
DemoHibernate/
├── src/main/java/        (Java classes)
├── src/main/resources/    (hibernate.cfg.xml, mapping files)
├── src/main/webapp/       (web content)
│   ├── WEB-INF/
│   │   └── web.xml
│   ├── index.jsp
│   └── success.jsp
└── WEB-INF/lib/          (Hibernate + driver jars)  (Eclipse will deploy them)
```

---

2. Create database and table

Open MySQL CLI or Workbench and run:

```
CREATE DATABASE hibernatedb;

USE hibernatedb;
```

```
CREATE TABLE USER2 (
  USER_ID INT NOT NULL AUTO_INCREMENT,
  USER_NAME VARCHAR(100),
  EMAIL VARCHAR(100),
  PRIMARY KEY (USER_ID)
);
```

Checkpoint: SELECT * FROM user; returns empty result set.

---

3. Add Hibernate & JDBC libraries (10–15 min)

If you are **not** using Maven:

1. Download required JARs (Hibernate 5.x distribution + MySQL connector).
2. Copy these to src/main/webapp/WEB-INF/lib/ (or Project → Properties → Java Build Path → Add JARs):
   - hibernate-core-5.x.x.jar
   - hibernate-commons-annotations-5.x.jar (or similar)
   - hibernate-jpa-2.1-api.jar
   - antlr.jar, javassist.jar, dom4j.jar (dependences)
   - log4j or slf4j jars (logging)
   - mysql-connector-java-8.x.jar
   - commons-logging.jar
     (If using Maven add appropriate dependencies instead.)

Tip: If a missing-class error appears at runtime, add the missing dependency jar.

---

4. Add configuration files

Create src/main/resources/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/hibernatedb</property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password">Niranjan.1</property>
```

```
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>

        <mapping resource="user.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Create src/main/resources/user.hbm.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.jwt.hibernate.bean.User" table="USER2">
        <id name="id" column="USER_ID">
            <generator class="native"/>
        </id>
        <property name="name" column="USER_NAME"/>
        <property name="email" column="EMAIL"/>
    </class>
</hibernate-mapping>
```

Checkpoint: Ensure both files are under src/main/resources so they appear in WEB-INF/classes at deployment.

---

5. Java classes

Create package com.jwt.hibernate.bean and add User.java:

```
package com.jwt.hibernate.bean;

public class User {
    private int id;
    private String name;
    private String email;

    public User() {}

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

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

Create com.example.util.HibernateUtil:

```
package com.example.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Create servlet com.example.servlet.SaveUserServlet:

```
package com.example.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;

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

import com.jwt.hibernate.bean.User;
import com.example.util.HibernateUtil;
/**
 * Servlet implementation class SaveUserServlet
 */
@WebServlet("/SaveUserServlet")
public class SaveUserServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		// TODO Auto-generated method stub
      String name = request.getParameter("name");
      String email = request.getParameter("email");

      User user = new User();
      user.setName(name);
      user.setEmail(email);

      Session session = HibernateUtil.getSessionFactory().openSession();
      Transaction tx = session.beginTransaction();
```

```
        session.save(user);
        tx.commit();
        session.close();

        response.sendRedirect("success.jsp");
    }
        }
```

---

6. JSP & web.xml (10–15 min)

Add src/main/webapp/index.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>Enter User Details</h2>
<form action="saveUser" method="post">
    Name: <input type="text" name="name"/><br/>
    Email: <input type="text" name="email"/><br/>
    <input type="submit" value="Save"/>
</form>
</body>
</html>
```

Add src/main/webapp/success.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
```

```
<h3>User saved successfully!</h3>
<a href="index.jsp">Add another user</a>
</body>
</html>
```

Configure web.xml (src/main/webapp/WEB-INF/web.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>DemoHibernate</display-name>
    <servlet>
        <servlet-name>SaveUserServlet</servlet-name>
        <servlet-class>com.example.servlet.SaveUserServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>SaveUserServlet</servlet-name>
        <url-pattern>/saveUser</url-pattern>
    </servlet-mapping>
</web-app>
```

---

7. Deployment assembly & build (5 min)

1. Right-click project → Properties → **Deployment Assembly**.
   Ensure:
   - /src/main/java → WEB-INF/classes
   - /src/main/resources → WEB-INF/classes
   - /src/main/webapp → /
2. Project → Clean → Build.
3. Run As → Run on Server (choose Tomcat). If already added to server, start server.

---

8. Test

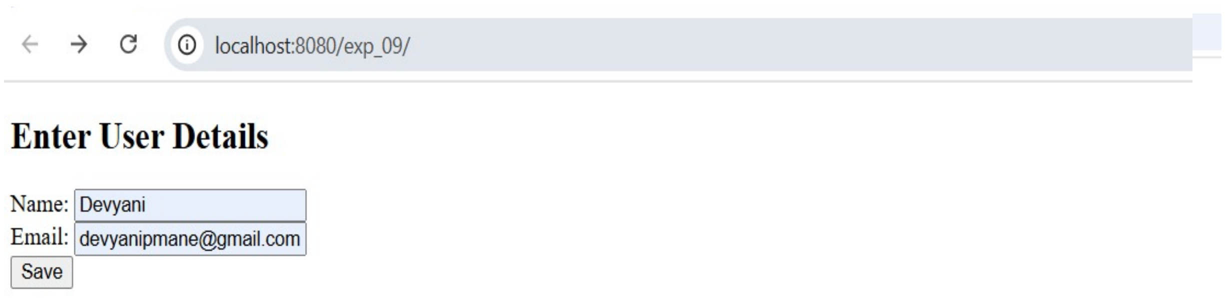
1. Open browser: http://localhost:8080/DemoHibernate/ (adjust context path if different).
2. Fill name & email → Submit. You should land on success.jsp.
3. Click **View All Users** → confirm the saved record is visible.
4. Verify in MySQL: SELECT * FROM user; — records should match.

Expected output: Console logs showing Hibernate SQL insert and select statements (because show_sql = true).

---

9. Troubleshooting (common errors & fixes)

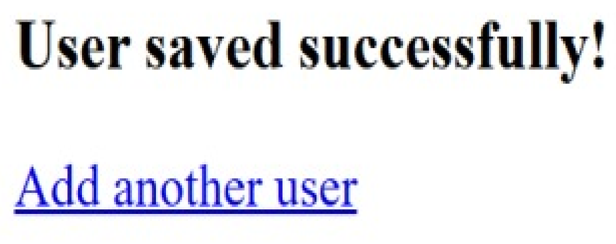
- **ClassNotFoundException: com.mysql.cj.jdbc.Driver**
  - Ensure mysql-connector-java-8.x.jar exists in WEB-INF/lib.
- **org.hibernate.MappingException: resource user.hbm.xml not found**
  - Ensure user.hbm.xml is in src/main/resources and mapping resource path in hibernate.cfg.xml matches. Example: mapping resource="user.hbm.xml" or com/example/bean/user.hbm.xml.
- **No Dialect mapping / SQL errors**
  - Verify correct hibernate.dialect property for your MySQL version: org.hibernate.dialect.MySQL5Dialect is OK for MySQL5/8 compatibility in many cases.
- **Servlet 404 for /saveUser**
  - Check web.xml and URL mappings. Confirm context path and mapping.
- **SessionFactory creation failed**
  - Look at server console logs — usually configuration error (typo in hibernate.cfg.xml) or missing jar.
- **Output:-**



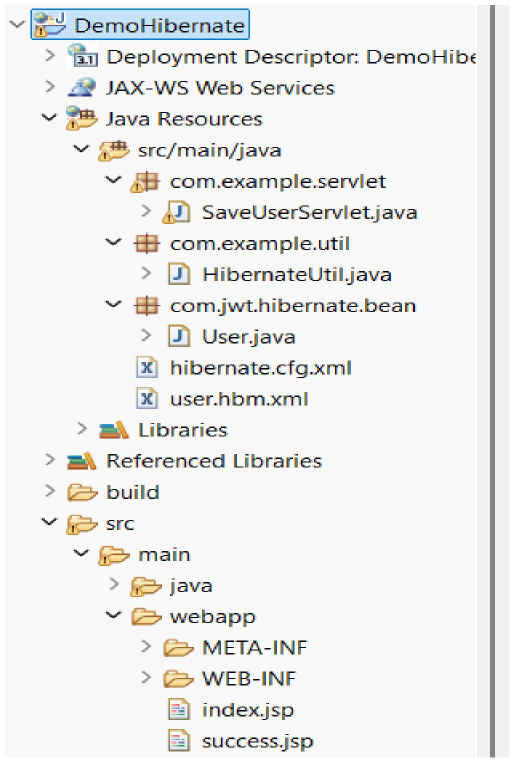
localhost:8080/exp_09/

Enter User Details

Name: Devyani
Email: devyanipmane@gmail.com



User saved successfully!



```
DemoHibernate
  > Deployment Descriptor: DemoHibe
  > JAX-WS Web Services
  v Java Resources
    v src/main/java
      v com.example.servlet
        > SaveUserServlet.java
      v com.example.util
        > HibernateUtil.java
      v com.jwt.hibernate.bean
        > User.java
        hibernate.cfg.xml
        user.hbm.xml
    > Libraries
  > Referenced Libraries
  > build
  v src
    v main
      > java
      v webapp
        > META-INF
        > WEB-INF
          index.jsp
          success.jsp
```

Conclusion : Thus, I have developed a Web application using Hibernate framework