

3.1 Set Up a JDBC Environment

It is assumed that you have a MySQL database server setup already

This section will guide you to:

- Set up Eclipse to work with JDBC
- Create an HTML page to call a servlet
- Create a servlet that will initialize JDBC and then close it

Development Environment

- Eclipse IDE for Enterprise Java Developers v2019-03 (4.11.0)
- Apache Tomcat Server v9.0
- JRE: OpenJDK Runtime Environment 11.0.2
- MySQL Connector for Java 8.0.16

This guide has twelve subsections, namely:

- 3.1.1 Creating a dynamic web project
- 3.1.2 Adding the jar files for MySQL connection for Java
- 3.1.3 Creating an HTML page index.html
- 3.1.4 Creating a DBConnection class to initiate a JDBC connection in code
- 3.1.5 Creating a config.properties file to store JDBC credentials
- 3.1.6 Creating a DemoJDBC servlet
- 3.1.7 Configuring web.xml
- 3.1.8 Checking for servlet-api.jar
- 3.1.9 Building the project
- 3.1.10 Publishing and starting the project
- 3.1.11 Running the project
- 3.1.12 Pushing the code to your GitHub repositories

Step 3.1.1: Creating a dynamic web project

- Go the **File** menu. Choose **New->Dynamic Web Project**
- Enter the project name as **JDBCSetup**. Click on **Next**
- Enter nothing in the next screen and click on **Next**
- Check the checkbox **Generate web.xml deployment descriptor** and click on **Finish**
- This will create the project files in the Project Explorer

Step 3.1.2: Adding the jar files for MySQL connection for Java

- Go to <https://dev.mysql.com/downloads/connector/j/>
- Choose **Platform Independent** from the **Select Operating System** dropdown
- Click on the **Download** button for **Platform Independent Zip Archive**
- Extract **mysql-connector-java-8.0.16.jar** from the zip and add it to your project's **WebContent/WEB-INF/lib** folder

Step 3.1.3: Creating an HTML page index.html

- In the Project Explorer, expand the project **JDBCSetup**
- Expand **WebContent**. Right click on **WebContent**. Choose **New->HTML File**
- Enter the filename as **index.html** and click on **Finish**
- Enter the following code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JDBC Setup</title>
</head>
<body>
<a href="init">Initialize JDBC</a><br>
</body>
</html>
```

- Click on the **Save** icon

Step 3.1.4: Creating a DBConnection class to initiate a JDBC connection in code

- In the Project Explorer, expand **JDBCSetup->Java Resources**

- Right click on **src** and choose **New->Class**
- In **Package**, enter **com.ecommerce** and in **Name** enter **DBConnection** and click on **Finish**
- Enter the following code:

```
package com.ecommerce;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {

    private Connection connection;

    public DBConnection(String dbURL, String user, String pwd) throws ClassNotFoundException,
SQLException{

        Class.forName("com.mysql.jdbc.Driver");
        this.connection = DriverManager.getConnection(dbURL, user, pwd);
    }

    public Connection getConnection(){
        return this.connection;
    }

    public void closeConnection() throws SQLException {
        if (this.connection != null)
            this.connection.close();
    }
}
```

Step 3.1.5: Creating a config.properties file to store JDBC credentials

- In the Project Explorer, expand the project **JDBCSetup**
- Expand **WebContent**. Right click on **WebContent**. Choose **New->File**
- Enter the filename as config.properties and click on **Finish**
- Enter the following data:

```
url=jdbc:mysql://localhost:3306/ecommerce
userid=root
password=master
```

Step 3.1.6: Creating a DemoJDBC servlet

- In the Project Explorer, expand **JDBCSetup->Java Resources**
- Right click on **src** and choose **New->Servlet**
- In **Class Name**, enter **DemoJDBC** and click on **Finish**
- Enter the following code:

```
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.math.BigDecimal;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

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 com.ecommerce.DBConnection;

/**
 * Servlet implementation class DemoJDBC
 */
@WebServlet("/DemoJDBC")
public class DemoJDBC extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public DemoJDBC() {
        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

        try {
            PrintWriter out = response.getWriter();
            out.println("<html><body>");

            InputStream in = getServletContext().getResourceAsStream("/WEB-INF/config.properties");
            Properties props = new Properties();
```

```

        props.load(in);

        DBConnection conn = new DBConnection(props.getProperty("url"), props.getProperty("userid"),
props.getProperty("password"));
        out.println("DB Connection initialized.<br>");

        conn.closeConnection();
        out.println("DB Connection closed.<br>");

        out.println("</body></html>");
        conn.closeConnection();

    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

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

```

Step 3.1.7: Configuring web.xml

- In the Project Explorer, expand **JDBCSetup->WebContent->WEB-INF**
- Double click on **web.xml** to open it in the editor
- Enter the following script:

```

<?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_4_0.xsd" id="WebApp_ID" version="4.0">
    <display-name>JDBCSetup</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <servlet>

```

```
<servlet-name>DemoJDBC</servlet-name>
<servlet-class>DemoJDBC</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DemoJDBC</servlet-name>
  <url-pattern>/init</url-pattern>
</servlet-mapping>

</web-app>
```

Step 3.1.8: Checking for servlet-api.jar

- Before building the project, we need to confirm that **servlet-api.jar** has been added to the project
- In the Project Explorer, right click on **JDBCSetup** and choose **Properties**
- Select **Java Build Path** from the options on the left
- Click on **Libraries** tab on the right
- Under **ClassPath**, expand the node that says **Apache Tomcat**
- If there is an existing entry for **servlet-api.jar**, then click on **Cancel** and exit the window
- If it is not there, then click on **Classpath** entry and click on **Add External JARs** button on the right
- From the **file** list, select **servlet-api.jar** file and click on **Ok**
- Click on **Apply and Close**

Step 3.1.9: Building the project

- From the **Project** menu at the top, click on **Build**
- If any compile errors are shown, fix them as required

Step 3.1.10: Publishing and starting the project

- If you do not see the **Servers** tab near the bottom of the IDE, go to **Window** menu and click on **Show View->Servers**

- Right click the **Server** entry and choose **Add and Remove**
- Click the **Add** button to move **JDBCSetup** from the **Available** list to the **Configured** list
- Click on **Finish**
- Right click the **Server** entry and click on **Publish**
- Right click the **Server** entry and click on **Start**
- This will start the server

Step 3.1.11: Running the project

- To run the project, open a web browser and type: **http://localhost:8080/JDBCSetup**

Step 3.1.12: Pushing the code to your GitHub repositories

Open your command prompt and navigate to the folder where you have created your files

```
cd
<folder
path>
```

Initialize your repository using the following command:

```
git init
```

Add all the files to your git repository using the following command:

```
git add .
```

Commit the changes using the following command:

```
git
commit
. -m "Changes have been committed."
```

Push the files to the folder you initially created using the following command:

```
git push
-u origin
master
```