PROGRAM: COOKIES SESSION TRACKING (**EXPT NO:1**)

COMPLETE STEP-BY-STEP GUIDE (BEGINNER FRIENDLY)

This document explains **everything from scratch** to run **Servlet & JSP programs using Eclipse + Tomcat**. This is written so **any beginner** can follow without confusion.

STEP 1: CHOOSE CORRECT ECLIPSE PACKAGE (MOST IMPORTANT)

When downloading Eclipse, you will see many options.

☑ SELECT ONLY THIS:

Eclipse IDE for Enterprise Java and Web Developers

❌ Do NOT select: - Eclipse IDE for Java Developers - Eclipse IDE for C/C++ Developers

These do **NOT** support Servlets/JSP properly.

STEP 2: INSTALL ECLIPSE (ZIP METHOD)

If you downloaded ZIP file:

1. Right-click the ZIP file → **Extract All**
2. Extract to:

   `C:\eclipse`
3. Open folder → double-click:

   `eclipse.exe`
4. Choose Workspace:

   `C:\eclipse-workspace`

✓ Eclipse opens successfully

STEP 3: DOWNLOAD & INSTALL JDK 17 (FREEWARE)

Why JDK?

- Servlets & Tomcat need Java
- Without JDK, Tomcat will fail

Download:

- Download **JDK 17 (LTS)** from Oracle / OpenJDK

Install:

- Run installer
- Install to default location:

  ```
  C:\Program Files\Java\jdk-17
  ```

STEP 4: SET JAVA_HOME (MOST IMPORTANT STEP)

Open Environment Variables

1. Press **Windows + R**
2. Type:

   ```
   sysdm.cpl
   ```

3. Press Enter
4. Go to **Advanced** tab
5. Click **Environment Variables**

Create JAVA_HOME

Under **System Variables**: 1. Click **New** 2. Variable Name: `JAVA_HOME` 3. Variable Value: `C:\Program Files\Java\jdk-17` 4. Click OK

Update PATH Variable

1. Under System Variables → Select **Path** → Edit
2. Click **New**
3. Paste:

   ```
   %JAVA_HOME%\bin
   ```

4. Click OK → OK → OK

STEP 5: RESTART SYSTEM (COMPULSORY)

🔴 You **MUST** restart your computer now.

STEP 6: TEST JAVA INSTALLATION

After restart:

1. Open **Command Prompt**
2. Type:

```
java -version
```

Expected Output:

- Java version details displayed

✓ Java installed correctly

STEP 7: INSTALL APACHE TOMCAT (LOCAL SERVER)

Why Tomcat?

- Servlets & JSP cannot run alone
- Tomcat runs them on localhost (offline)

Download:

- **Apache Tomcat 9.x (Windows ZIP)**

⚠️ Choose ZIP, NOT EXE

STEP 8: EXTRACT TOMCAT

1. Right-click ZIP → Extract All
2. Extract to:

```
C:\tomcat9
```

Folder structure:

```
C:\tomcat9
 ├── bin
 ├── conf
 ├── webapps
```

STEP 9: TEST TOMCAT (MANUAL TEST – ONE TIME ONLY)

1. Open:

   `C:\tomcat9\bin`

2. Double-click:

   `startup.bat`

3. **Black window opens – DO NOT CLOSE IT**

4. Open browser → Type:

   `http://localhost:8080`

✓ Tomcat page opens → Tomcat installed correctly

⚠️ After this test, **DO NOT use startup.bat again**

STEP 10: USE GOOGLE CHROME (RECOMMENDED)

Why Chrome?

- Best support for HTML, CSS, JavaScript
- Best for AJAX, JSON, Servlets, JSP
- Industry standard
- Used in labs & exams

STEP 11: CREATE DYNAMIC WEB PROJECT IN ECLIPSE

1. Open Eclipse
2. File → New → **Dynamic Web Project**
3. Project Name:

   `cookie`

4. Target Runtime: **Apache Tomcat v9.0**

5. Finish

STEP 12: PROJECT STRUCTURE (IMPORTANT)

```
src/main/java        → Servlet (.java)
src/main/webapp     → HTML / JSP
WEB-INF             → web.xml
```

STEP 13: CREATE FILES IN ECLIPSE

Create Servlet (.java)

1. Right-click `src/main/java`
2. New → Other → Java → Package
3. Name:

   `com.example.cookies`
4. Right-click package → New → Class
5. Class Name:

   `PreferenceServlet`

   code:

   ```
   package com.example.cookies;


   import java.io.IOException;
   import java.io.PrintWriter;
   import javax.servlet.ServletException;
   import javax.servlet.http.Cookie;
   import javax.servlet.http.HttpServlet;
   import javax.servlet.http.HttpServletRequest;
   import javax.servlet.http.HttpServletResponse;
   ```

```
public class PreferenceServlet extends HttpServlet {


   protected void doPost(HttpServletRequest request, HttpServletResponse
response)

         throws ServletException, IOException {


      String theme = request.getParameter("theme");

      String language = request.getParameter("language");


      // Create cookies

      Cookie themeCookie = new Cookie("theme", theme);

      Cookie languageCookie = new Cookie("language", language);


      // Set cookie expiry (1 day)

      themeCookie.setMaxAge(24 * 60 * 60);

      languageCookie.setMaxAge(24 * 60 * 60);


      response.addCookie(themeCookie);

      response.addCookie(languageCookie);


      response.setContentType("text/html");

      PrintWriter out = response.getWriter();


      out.println("<h2>Preferences Saved Successfully!</h2>");

      out.println("<a href='preferences'>View Preferences</a>");

   }
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws ServletException, IOException {

    String theme = "Not Set";
    String language = "Not Set";

    Cookie[] cookies = request.getCookies();

    if (cookies != null) {
      for (Cookie c : cookies) {
        if (c.getName().equals("theme")) {
          theme = c.getValue();
        }
        if (c.getName().equals("language")) {
          language = c.getValue();
        }
      }
    }

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("<h2>Your Saved Preferences</h2>");
    out.println("Theme: " + theme + "<br>");
    out.println("Language: " + language + "<br><br>");
    out.println("<a href='preferences.html'>Change Preferences</a>");
```

```
    }
}
```

Create HTML File

1. Right-click `src/main/webapp`
2. New → HTML File
3. Name:

   `preferences.html`

```html
<!DOCTYPE html>
<html>
<head>
    <title>User Preferences</title>
</head>
<body>

<h2>Select Your Preferences</h2>

<form action="preferences" method="post">
    Theme:
    <select name="theme">
        <option value="Light">Light</option>
        <option value="Dark">Dark</option>
    </select>
    <br><br>

    Language:
    <select name="language">
        <option value="English">English</option>
        <option value="Tamil">Tamil</option>
        <option value="Hindi">Hindi</option>
    </select>
    <br><br>

    <input type="submit" value="Save Preferences">
</form>

</body>
</html>
```

Create web.xml

1. Right-click `src/main/webapp/WEB-INF`

2. Right-click `WEB-INF` → New → other->XML->XML File
3. Name:

   `web.xml`
4. Open with **Text Editor** (NOT Web XML Editor)

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         version="3.1">

    <servlet>
        <servlet-name>PreferenceServlet</servlet-name>
        <servlet-class>com.example.cookies.PreferenceServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>PreferenceServlet</servlet-name>
        <url-pattern>/preferences</url-pattern>
    </servlet-mapping>

</web-app>
```

STEP 14: ADD PROJECT TO TOMCAT

1. Open **Servers** tab
   (Go to the top menu bar and select Window.

   Hover over Show View.

   If you see Servers in the list, click it. And it will come under the bottom of the window

   If you don't see it, click 'Other...' at the bottom of the list.

   In the filter box, type Servers.)

   Expand the Server folder, select Servers, and click Open.
2. Right-click Tomcat → **Add and Remove**
3. Select `cookie` → Add → Finish
4. Start Tomcat from Eclipse

CLEAN PROJECT (IMPORTANT)

1. Click **Project → Clean**

2. Select **cookie**
3. Click **OK**

MOST IMPORTANT:

START TOMCAT ONLY FROM ECLIPSE

1. Open **Eclipse**
2. Go to the **Servers** tab
3. Right-click Tomcat → **Start**

EXPECTED MESSAGE

INFO: Server startup in XXXX ms

NO port error ❌

[Rare Case Error APPEARS (RARE CASE)

Then **another program is locking Java ports**.

QUICK FIX

1. Close:
    - Eclipse
    - Browser
2. Open **Command Prompt as Administrator**
3. Run:
4. netstat -ano | findstr :8080
5. netstat -ano | findstr :9090(mention the port id you want to give)
6. Note PID number
7. Open **Task Manager → End task** for that PID
8. Restart Eclipse → Start Tomcat **]**

STEP 15: RUN APPLICATION

Open Chrome and type:

`http://localhost:8080/cookie/preferences.html`

✓ Application runs successfully

For preference test:

http://localhost:8080/cookie/preferences

IMPORTANT RULES (REMEMBER FOREVER)

❌ Never start Tomcat twice ✓ Use **Eclipse OR startup.bat**, not both

❌ Do NOT put .java inside webapp ✓ .java → src/main/java

NOTE : CHECK THE CONTEXT ROOT IN WEB PROJECT SETTINGS TO KNOW THE ORIGINAL PROJECT NAME (RIGHT CLICK THE PROJECT NAME -> PROPERTIES->WEB PROJECT SETTINGS-> CONTEXT ROOT

1. Right-click your project
2. Click **Properties**
3. Select **Web Project Settings**
4. Look at:

Context Root

Example: WebSessionDemo

Remember this name exactly

USE CORRECT URL FORMAT

Correct URL structure:

http://localhost:8080/<ContextRoot>/<file-or-servlet>

http://localhost:8080/WebSessionDemo/preferences.html

CHECK THE PROJECT FACETS (RIGHT CLICK THE PROJECT NAME -> PROPERTIES-> PROJECT FACETS) TO CHECK THE DYNAMIC WEB SERVER MODULE,JAVA AND JAVASCRIPT ARE MARKED.

PROGRAM: Servlet to demonstrate the difference between HTTP GET and POST methods by creating a form and handling requests (**EXPT NO:2**)

SAME PROCEDURE SHOULD BE FOLLOWED BY CREATING A NEW DYNAMIC WEB PROJECT.

PROJECT DETAILS (IMPORTANT)

- **Dynamic Web Project Name:** login
- **Context Root:** login
- **Package Name:** log
- **Server:** Apache Tomcat 9
- **Browser:** Google Chrome
-

STEP 1: Create login.html

login → src → main → webapp → New → HTML File

File Name login.html

Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Login Page</title>
</head>
<body>


<h2>User Login</h2>


<form action="login" method="post">
```

```
    Username:
    <input type="text" name="username" required>
    <br><br>

    Password:
    <input type="password" name="password" required>
    <br><br>

    <input type="submit" value="Login">
</form>

</body>
</html>
```

STEP 2: Create Package log

login → src → main → java → New → Package

Package Name log

Click **Finish**

STEP 3: Create LoginServlet.java

login → src → main → java → log → New → Class

Class Name LoginServlet

Code:

```
package log;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class LoginServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    String username = request.getParameter("username");
    String password = request.getParameter("password");

    if (username.equals("admin") && password.equals("1234")) {

      HttpSession session = request.getSession();
      session.setAttribute("user", username);

      response.sendRedirect("welcome");

    } else {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h3>Invalid Login</h3>");
      out.println("<a href='login.html'>Try Again</a>");
    }
```

```
  }
}
```

STEP 4: Create WelcomeServlet.java

login → src → main → java → log → New → Class

code:

Class Name WelcomeServlet

Code:

```
package log;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class WelcomeServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    HttpSession session = request.getSession(false);

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
```

```
        if (session != null && session.getAttribute("user") != null) {
            String user = (String) session.getAttribute("user");
            out.println("<h2>Welcome, " + user + "</h2>");
            out.println("<a href='logout'>Logout</a>");
        } else {
            out.println("<h3>Session Expired. Please Login Again.</h3>");
            out.println("<a href='login.html'>Login</a>");
        }
    }
}
```

STEP 5: Create LogoutServlet.java

login → src → main → java → log → New → Class

Class Name LogoutServlet

Code:

```
package log;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class LogoutServlet extends HttpServlet {
```

```
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    HttpSession session = request.getSession(false);
    if (session != null) {
      session.invalidate();
    }

    response.sendRedirect("login.html");
  }
}
```

STEP 6: Create web.xml

login → src → main → webapp → WEB-INF → New → XML File

code:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
      version="3.1">

  <servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>log.LoginServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/login</url-pattern>
```

```
  </servlet-mapping>

  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>log.WelcomeServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/welcome</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>log.LogoutServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/logout</url-pattern>
  </servlet-mapping>

</web-app>
```

Save everything.

STEP 7: Run the Application

Clean Project

Project → Clean → login → OK

Restart Tomcat (ONLY from Eclipse)

Servers → Right-click Tomcat → Stop

Servers → Right-click Tomcat → Start

STEP 8: Test in Browser

http://localhost:8080/login/login.html

Login Credentials

Username: admin

Password: 1234

The real use of this program is to demonstrate SESSION TRACKING using HttpSession.

NOT:

- strong security
- real-world login system

Why this program exists (academic purpose)

Problem in web applications

HTTP is **stateless**
➞ Server does NOT remember the user between requests

Example:

1. You submit login form
2. Server processes it
3. Next page request comes
4. Server **forgets who you are**

What this program proves

This program shows how:

1. **User logs in**
2. **Server creates a session**
3. **User identity is stored**
4. **Server remembers the user**
5. **User can access protected pages**
6. **Session is destroyed on logout**

That's the **REAL purpose**.

Purely understanding purpose

Step-by-step flow (IMPORTANT)

1️⃣ **User enters username & password**

<form action="login" method="post">

➡ Request goes to LoginServlet

2️⃣ **Servlet validates credentials**

if (username.equals("admin") && password.equals("1234"))

⚠️ This is **dummy validation**, only to **simulate login**

3️⃣ **Session is created**

HttpSession session = request.getSession();

session.setAttribute("user", username);

✅ This is the **key line of the entire program**

Now server remembers:

user = admin

4️⃣ **User is redirected**

response.sendRedirect("welcome");

➞ New request
➞ Session is reused
➞ Data is still available

5️⃣ WelcomeServlet checks session

HttpSession session = request.getSession(false);

- If session exists → allow access
- If not → deny access

✅ This is **access control**

6️⃣ Logout destroys session

session.invalidate();

➞ User is forgotten completely

So what happens if password is wrong?

```
else {
   out.println("Invalid Login");
}
```

👉 **No session is created**
👉 Server does NOT remember the user
👉 Access is denied

This proves:

❌ Without session → no access

REAL-WORLD USE (conceptual)

This SAME logic is used in:

- College ERP systems
- Online exams
- Banking portals
- Gmail / Facebook login (advanced version)
- Shopping cart (Amazon, Flipkart)
- Admin dashboards

Only difference:

- Passwords come from database
- Sessions have timeout
- Security is stronger

EXPERIMENT 3 – JSP (GET & POST Methods)

AIM

To create a **JSP web page** that accepts user input using an **HTML form** and displays the submitted data using **GET and POST methods**.

SOFTWARE REQUIRED (FREEWARE)

- **Eclipse IDE for Enterprise Java**
- **Apache Tomcat 9**
- **Browser:** Chrome / Edge / Firefox (any one)

STEP 1: Open Eclipse

1. Open **Eclipse IDE**
2. Select your **Workspace**
3. Click **Launch**

STEP 2: Create Dynamic Web Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name:** JSPGetPostDemo
3. **Target Runtime:** Select **Apache Tomcat 9**
4. Click **Next → Next**
5. Tick ✓ **Generate web.xml**
6. Click **Finish**

STEP 3: Project Structure (Understand This)

You will see:

JSPGetPostDemo

└── src

└── src/main/webapp

  └── WEB-INF

    └── web.xml

👉 **All JSP and HTML files go inside**
src → main → webapp

STEP 4: Create JSP File

1. Right-click on
   src → main → webapp
2. Click **New → JSP File**
3. File name: index.jsp
4. Click **Finish**

STEP 5: Write JSP Code

index.jsp

📍 Location:
src → main → webapp → index.jsp

CODE:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
  <title>JSP GET and POST Example</title>
</head>
<body>

<h2>JSP Form using GET Method</h2>

<form method="get" action="index.jsp">
  Name: <input type="text" name="gname" required><br><br>
  Age: <input type="number" name="gage" required><br><br>
  <input type="submit" value="Submit using GET">
</form>

<hr>

<h2>JSP Form using POST Method</h2>

<form method="post" action="index.jsp">
  Name: <input type="text" name="pname" required><br><br>
```

```
  Age: <input type="number" name="pageValue" required><br><br>
  <input type="submit" value="Submit using POST">
</form>

<hr>

<h2>Submitted Data</h2>

<%
  String gname = request.getParameter("gname");
  String gage = request.getParameter("gage");

  String pname = request.getParameter("pname");
  String pageValue = request.getParameter("pageValue");

  if (gname != null && gage != null) {
%>
    <p><b>GET Method Output:</b></p>
    Name: <%= gname %><br>
    Age: <%= gage %><br>
<%
  }

  if (pname != null && pageValue != null) {
%>
    <p><b>POST Method Output:</b></p>
```

```
    Name: <%= pname %><br>
    Age: <%= pageValue %><br>
<%
  }
%>

</body>
</html>
```

STEP 6: Add Project to Server

1. Go to **Servers tab (bottom)**
2. Right-click → **Add and Remove**
3. Move JSPGetPostDemo to **Configured**
4. Click **Finish**

STEP 7: Run the Program

Server is already configured just run the program using URL

http://localhost:8080/JSPGetPostDemo/index.jsp

otherwise

1. Right-click on index.jsp
2. Click **Run As → Run on Server**
3. Choose **Tomcat 9**
4. Click **Finish**

Open URL in Browser

http://localhost:8080/JSPGetPostDemo/index.jsp

EXPERIMENT 4– SERVLET (GET & POST Methods)

AIM

To create a **Java Servlet** that demonstrates the difference between **HTTP GET and POST methods** by accepting user input through an HTML form and displaying the submitted data accordingly.

SOFTWARE REQUIRED (FREEWARE)

- Eclipse IDE for Enterprise Java
- Apache Tomcat 9
- Browser: Chrome / Edge / Firefox

STEP 1: Open Eclipse

1. Open **Eclipse IDE**
2. Select your **Workspace**
3. Click **Launch**

STEP 2: Create Dynamic Web Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name:** DoPostServletDemo
3. **Target Runtime:** Apache Tomcat 9
4. Click **Next → Next**
5. Tick ✓ **Generate web.xml**
6. Click **Finish**

STEP 3: Understand Project Structure

You will see:

DoPostServletDemo

```
├── src
│   └── main
│       └── java
└── src
    └── main
        └── webapp
            └── WEB-INF
                └── web.xml
```

👉 **HTML files** → src/main/webapp
👉 **Servlet (.java)** → src/main/java

STEP 4: Create HTML Form

📍 **Location:**

src → main → webapp

1. Right-click **webapp**
2. Click **New → HTML File**
3. File name: index.html
4. Click **Finish**

CODE: index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>GET and POST Servlet Demo</title>
</head>
<body>
```

```
<h2>Form using GET Method</h2>
<form action="DoPostServletDemo" method="get">
  Name: <input type="text" name="name"><br><br>
  Age: <input type="number" name="age"><br><br>
  <input type="submit" value="Submit using GET">
</form>

<hr>

<h2>Form using POST Method</h2>
<form action="DoPostServletDemo" method="post">
  Name: <input type="text" name="name"><br><br>
  Age: <input type="number" name="age"><br><br>
  <input type="submit" value="Submit using POST">
</form>

</body>
</html>
```

STEP 5: Create Servlet

📍 **Location:**

src → main → java

1. Right-click **java**
2. Click **New → Servlet**
3. Package name: com.demo
4. Class name: DoPostServletDemo

5. Click **Finish**

STEP 6: Write Servlet Code

CODE: DoPostServletDemo.java

```
package com.demo;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/DoPostServletDemo")
public class DoPostServletDemo extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String name = request.getParameter("name");
    String age = request.getParameter("age");
```

```
        out.println("<html><body>");
        out.println("<h2>GET Method Output</h2>");
        out.println("Name: " + name + "<br>");
        out.println("Age: " + age + "<br>");
        out.println("<a href='index.html'>Back</a>");
        out.println("</body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("name");
        String age = request.getParameter("age");

        out.println("<html><body>");
        out.println("<h2>POST Method Output</h2>");
        out.println("Name: " + name + "<br>");
        out.println("Age: " + age + "<br>");
        out.println("<a href='index.html'>Back</a>");
        out.println("</body></html>");
    }
}
```

In STEP 2: Tick ✓ Generate web.xml is not choosed then

CREATE web.xml

📍 **LOCATION**

login → src → main → webapp → WEB-INF

👆 **EXACT CLICKS**

1. Right-click **WEB-INF**
2. Click **New**
3. Click **Other**
4. Select **XML → XML File**
5. Click **Next**
6. File name:
7. web.xml
8. Click **Finish**

⚠️ **VERY IMPORTANT**

If Eclipse opens **Design view**, switch to:

👉 **Source** or **Text Editor**

✍️ **COPY-PASTE THIS CODE**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
      version="3.1">

  <servlet>
    <servlet-name>GetPostServlet</servlet-name>
```

```
    <servlet-class>log.GetPostServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>GetPostServlet</servlet-name>
    <url-pattern>/getpost</url-pattern>
  </servlet-mapping>

</web-app>
```

Press **CTRL + S**

[in THIS program you do NOT need to write or edit web.xml.
Because you are using **Annotation (@WebServlet)**.

So even though you **ticked "Generate web.xml"**, you **don't touch it**.

🧠 WHY IS web.xml NOT USED HERE?

There are **TWO ways** to connect a Servlet with a URL:

✅ METHOD 1: Using web.xml (OLD / CLASSICAL WAY)

You manually write:

<servlet>

<servlet-mapping>

👉 You used this in **HelloServlet**, **Cookie**, **Session** programs earlier.

✅ METHOD 2: Using Annotation (MODERN WAY)

You write **only this line in servlet**:

@WebServlet("/DoPostServletDemo")

👉 This **REPLACES web.xml mapping**

📌 **Your current program uses THIS method**

📌 WHAT HAPPENS INTERNALLY?

When Tomcat starts:

- It scans all servlet classes
- Finds this line 👇

@WebServlet("/DoPostServletDemo")

- Automatically maps:

URL → Servlet class

So Tomcat already knows:

/DoPostServletDemo → DoPostServletDemo.java

✓ That's why your program runs **WITHOUT touching web.xml**

❓ THEN WHY DID WE TICK "Generate web.xml"?

Good question 👌

Reasons:

1. Eclipse **creates it by default**
2. Some experiments **require web.xml**
3. Exams still **ask web.xml-based programs**
4. No harm in keeping it

👉 **Generated ≠ Mandatory to edit**

🧪 WHAT IF I DELETE web.xml?

✓ Program will **STILL WORK**
(because annotation is enough)

🧪 **WHAT IF I REMOVE @WebServlet?**

❌ Program will **NOT WORK**

(because no URL mapping exists)]

STEP 7: Add Project to Server

1. Go to **Servers** tab (bottom)
2. Right-click → **Add and Remove**
3. Move DoPostServletDemo to **Configured**
4. Click **Finish**

STEP 8: Run the Program

Method 1 (Easy)

1. Right-click index.html
2. Click **Run As → Run on Server**
3. Select **Tomcat 9**
4. Click **Finish**

OR Open Browser:

http://localhost:8080/DoPostServletDemo/index.html

WHAT IS GET AND POST? (IN SIMPLE WORDS)

GET and POST are **HTTP methods** used to **send data from a web page to the server**.

In our JSP program:

- The **browser sends data**

- The **JSP page receives and displays it**

🔹 **GET METHOD – EXPLANATION**

👉 **What GET does**

- Sends form data **through the URL**
- Data is **visible in browser address bar**
- Used for **small, non-sensitive data**

👉 **In our program (GET form)**

```
<form method="get" action="index.jsp">
  Name: <input type="text" name="gname">
  Age: <input type="number" name="gage">
  <input type="submit">
</form>
```

👉 **What happens when user submits**

If user enters:

- Name = Ravi
- Age = 20

Browser URL becomes:

http://localhost:8080/JSPGetPostDemo/index.jsp?gname=Ravi&gage=20

✓ Data is **attached to URL**
✓ JSP reads data using:

```
request.getParameter("gname");
request.getParameter("gage");
```

👉 **Output shown by JSP**

GET Method Output:

Name: Ravi

Age: 20

🔹 **POST METHOD – EXPLANATION**

👉 **What POST does**

- Sends data **inside request body**
- Data is **NOT visible in URL**
- Used for **secure or large data**

👉 **In our program (POST form)**

```
<form method="post" action="index.jsp">
  Name: <input type="text" name="pname">
  Age: <input type="number" name="pageValue">
  <input type="submit">
</form>
```

👉 **What happens when user submits**

If user enters:

- Name = Anu
- Age = 21

Browser URL remains:

http://localhost:8080/JSPGetPostDemo/index.jsp

✓ Data is **hidden**
✓ JSP still reads it using:

```
request.getParameter("pname");
request.getParameter("pageValue");
```

👉 **Output shown by JSP**

POST Method Output:

Name: Anu

Age: 21

🔹 HOW JSP HANDLES BOTH (VERY IMPORTANT)

In **index.jsp**, we wrote:

String gname = request.getParameter("gname");

String gage = request.getParameter("gage");

String pname = request.getParameter("pname");

String pageValue = request.getParameter("pageValue");

✓ Same request.getParameter() works for **both GET and POST**
✓ Difference is **how data is sent**, not how it is read

🔹 KEY DIFFERENCES (EXAM TABLE)

Feature	**GET**	**POST**
Data location	URL	Request body
Visibility	Visible	Hidden
Security	Less	More
Data size	Limited	Large
Usage	Search, filters	Login, forms

GET

- Search
- Filters
- Page navigation

POST

- Login forms

- Registration
- Passwords
- Personal data

Similarly,

When the **“Submit using GET”** button is clicked:

- The data is appended to the **URL**.
- Example URL:
- http://localhost:8080/DoPostServletDemo/DoPostServletDemo?name=Ravi&age=20

2. The request is handled by the **doGet() method** of the servlet.
3. The servlet reads the values using:
4. request.getParameter("name");
5. request.getParameter("age");
6. The servlet displays the output in the browser.

Displayed Output:

GET Method Output

Name: Ravi

Age: 20

✓ This shows that **GET sends data through the URL** and is **not secure**.

Output for POST Method

Step-by-step behavior:

1. The user enters **Name** and **Age** in the **POST form**.
2. When the **“Submit using POST”** button is clicked:
   - The data is sent in the **HTTP request body**.
   - The URL does **not** show the submitted data.

3. The request is handled by the **doPost() method** of the servlet.
4. The servlet reads the values using:
5. request.getParameter("name");
6. request.getParameter("age");
7. The servlet displays the output in the browser.

Displayed Output:

POST Method Output

Name: Ravi

Age: 20

✓ This shows that **POST sends data securely** and is suitable for **sensitive information**.

EXPERIMENT 5 – HELLO WORLD SERVLET

AIM

To write a simple Servlet program that displays **"Hello, World!"** when accessed through a web browser.

SOFTWARE REQUIRED (FREEWARE)

- **Eclipse IDE for Enterprise Java**
- **Apache Tomcat 9**
- **Browser:** Chrome / Edge / Firefox (any one)

STEP 1: Open Eclipse

1. Open **Eclipse IDE**
2. Select your **Workspace**
3. Click **Launch**

STEP 2: Create Dynamic Web Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name:** hello
3. **Target Runtime:** Select **Apache Tomcat 9**
    - If not selected → click **New Runtime** → choose Tomcat 9 → Finish
4. Click **Next**
5. Click **Next**
6. ✓ Tick **Generate web.xml**
7. Click **Finish**

STEP 3: Understand Project Structure

You will see this in **Project Explorer**:

hello

```
├── src
├── src/main/webapp
│   └── WEB-INF
│       └── web.xml
```

👉 **Servlet (.java) files → go inside src**
👉 **Browser files → go inside webapp**

STEP 4: Create Servlet Package

1. Right-click on **src**
2. Click **New → Package**
3. Package name: log
4. Click **Finish**

STEP 5: Create Servlet Class

1. Right-click on package **log**
2. Click **New → Class**
3. Enter:
    - **Class Name:** HelloServlet
4. ✓ Tick **public static void main** → ❌ NO (don't tick)
5. Click **Finish**

STEP 6: Write Servlet Code

📍 Location:

hello → src → log → HelloServlet.java

```
package log;


import java.io.IOException;
import java.io.PrintWriter;


import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class HelloServlet extends HttpServlet {


  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
```

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body>");
        out.println("<h1>Hello, World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

✅ Press **Ctrl + S** (Save)

STEP 7: Configure web.xml

📍 Location:

hello → src → main → webapp → WEB-INF → web.xml

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        version="3.1">

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
```

```
    <servlet-class>log.HelloServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

✅ Save file (**Ctrl + S**)

STEP 8: Add Project to Server

1. Go to **Servers tab** (bottom)
2. Right-click → **Add and Remove**
3. Move **hello** to **Configured**
4. Click **Finish**

STEP 9: Start Server

1. In **Servers tab**
2. Right-click **Tomcat v9.0 Server**
3. Click **Start**

✓ Wait until you see:

Server startup in xxxx ms

STEP 10: Run the Servlet

Open any browser and type:

http://localhost:8080/hello/hello

🎉 **OUTPUT**

You will see on browser:

Hello, World!

OUTPUT EXPLANATION

- The browser sends a **GET request** to the servlet.
- The doGet() method is executed.
- The servlet writes HTML output using PrintWriter.
- The message **"Hello, World!"** is displayed on the web page.

EXPERIMENT 6– AJAX (Fetching data from Text File)

AIM

To create a simple web page that fetches student data from a text file using **AJAX** and displays it dynamically **without reloading the page**.

SOFTWARE REQUIRED (FREEWARE)

• Eclipse IDE for Enterprise Java
• Apache Tomcat 9
• Browser: Chrome / Edge / Firefox

STEP 1: Open Eclipse

1. Open **Eclipse IDE**
2. Select your **Workspace**
3. Click **Launch**

STEP 2: Create Dynamic Web Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name:** AjaxTextDemo
3. **Target Runtime:** Apache Tomcat 9
4. Click **Next → Next**
5. Tick **✓ Generate web.xml**
6. Click **Finish**

STEP 3: Understand Project Structure

You will see:

AjaxTextDemo

```
├── src
└── src
    └── main
        └── webapp
            └── WEB-INF
                └── web.xml
```

👉 **HTML + TXT files go inside:**

src → main → webapp

STEP 4: Create Student Data Text File

📍 **Location:**
src → main → webapp

Steps:

1. Right-click **webapp**

2. Click **New → File**
3. File name: **students.txt**
4. Click **Finish**

CODE: students.txt

Roll No: 101, Name: Arun, Department: AI&DS

Roll No: 102, Name: Priya, Department: CSE

Roll No: 103, Name: Karthik, Department: IT

Roll No: 104, Name: Divya, Department: ECE

✓ Save the file

STEP 5: Create HTML File

📍 **Location:**
src → main → webapp

Steps:

1. Right-click **webapp**
2. Click **New → HTML File**
3. File name: **index.html**
4. Click **Finish**

STEP 6: Write AJAX Code

CODE: index.html

📍 src → main → webapp → index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>AJAX Text File Example</title>
```

```
  <script>
    function loadStudentData() {
      var xhr = new XMLHttpRequest();

      xhr.open("GET", "students.txt", true);

      xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {
          document.getElementById("output").innerHTML =
            "<pre>" + xhr.responseText + "</pre>";
        }
      };

      xhr.send();
    }
  </script>
</head>

<body>

<h2>Student Records (AJAX)</h2>

<button onclick="loadStudentData()">Load Student Data</button>

<br><br>
```

</body>

</html>

✓ Save the file

STEP 7: Add Project to Server

1. Go to **Servers tab**
2. Right-click → **Add and Remove**
3. Move **AjaxTextDemo** to *Configured*
4. Click **Finish**
5. Start the server (if not already started)

STEP 8: Run the Program

Method 1 (Easy)

1. Right-click **index.html**
2. Click **Run As → Run on Server**
3. Select **Tomcat 9**
4. Click **Finish**

OR use Browser:

http://localhost:8080/AjaxTextDemo/index.html

OUTPUT EXPLANATION

1. The page loads with a **button**:
   “Load Student Data”
2. When the button is clicked:

- AJAX sends a request to students.txt
- Page **does NOT reload**

3. The student data is fetched dynamically
4. Data appears inside the output box

✓ This proves **AJAX works without page refresh**

Important points:

What is AJAX?

AJAX stands for **Asynchronous JavaScript and XML**.
It allows web pages to update data **without reloading**.

Why Tomcat is needed?

Browsers block AJAX file access locally.
Tomcat provides a **local server environment**.

Is servlet used here?

❌ No
✓ Only **HTML + JavaScript**

Can this AJAX program run without Tomcat?

❌ **Directly opening index.html (double-click) → NO**

If you do this:

index.html (double-click)

Browser opens as:

file:///C:/...

👉 **AJAX will FAIL**

❌ **Reason (very important concept):**

Browsers **block AJAX requests** to local files for **security reasons**.

This error is called:

CORS / Local file restriction

[**Cross-Origin Resource Sharing** (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

An example of a cross-origin request: the front-end JavaScript code served from https://domain-a.com uses fetch() to make a request for https://domain-b.com/data.json.

For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts. For example, fetch() and XMLHttpRequest follow the same-origin policy. This means that a web application using those APIs can only request resources from the same origin the application was loaded from unless the response from other origins includes the right CORS headers.]

So:

- HTML opens ✓
- Button works ✓
- AJAX request ❌ (students.txt not loaded)

✅ **When DOES it work without Tomcat?**

✓ OPTION 1: Use any local server (Tomcat is just one)

Tomcat is **not special** — it's just a **local HTTP server**.

AJAX needs:

http://localhost

not:

file:///

✅ **OPTION 2: Use VS Code Live Server (EASIEST, NO TOMCAT)**

Steps:

1. Install **VS Code**
2. Install extension: **Live Server**
3. Right-click index.html
4. Click **Open with Live Server**

URL becomes:

http://127.0.0.1:5500/index.html

✓ AJAX works
✓ No Tomcat
✓ Best for beginners

✅ OPTION 3: Use Python Simple Server (NO TOMCAT)

If Python is installed:

1. Open Command Prompt
2. Go to project folder:

cd path_to_your_webapp

3. Run:

python -m http.server 8080

4. Open browser:

http://localhost:8080

✓ AJAX works
✓ No Tomcat

🧠 FINAL TRUTH (EXAM / LAB ANSWER)

❓ Is Tomcat mandatory?

NO

❓ Is a server mandatory?

YES

AJAX **requires an HTTP server**.

📌 **Why colleges use Tomcat then?**

Because:

- Already needed for **Servlets & JSP**
- Works offline
- Industry standard
- Same server for **HTML + AJAX + Servlet**

So for labs:

Tomcat is the safest choice

EXPERIMENT 7: AJAX WITH JSON

AIM

To create a simple web application that uses **AJAX** to fetch **student data from a JSON file** and display it dynamically on a web page **without reloading the page**.

SOFTWARE REQUIRED (FREEWARE)

• Eclipse IDE for Enterprise Java
• Apache Tomcat 9
• Browser: Google Chrome

STEP 1: Open Eclipse

1. Open **Eclipse IDE**
2. Select Workspace
3. Click **Launch**

STEP 2: Create Dynamic Web Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name:** AjaxJsonDemo
3. Target Runtime: **Apache Tomcat 9**
4. Click **Next → Next**
5. ✓ Tick **Generate web.xml**
6. Click **Finish**

STEP 3: Understand Project Structure

AjaxJsonDemo

└── src

└── src/main/webapp

  └── WEB-INF

    └── web.xml

👉 HTML / JSON / JS files go inside
src → main → webapp

STEP 4: Create JSON File

📍 Location:

src → main → webapp

Steps:

1. Right-click **webapp**
2. New → File
3. File name: **students.json**
4. Click Finish

CODE: students.json

```
[
 { "id": 101, "name": "Arun", "dept": "AI&DS" },
 { "id": 102, "name": "Meena", "dept": "CSE" },
 { "id": 103, "name": "Karthik", "dept": "IT" }
]
```

STEP 5: Create HTML File

📍 Location:

src → main → webapp

1. Right-click **webapp**
2. New → HTML File
3. File name: **index.html**
4. Click Finish

CODE: index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>AJAX with JSON</title>

  <script>
    function loadStudents() {
      var xhr = new XMLHttpRequest();
      xhr.open("GET", "students.json", true);
```

```
        xhr.onreadystatechange = function () {
          if (xhr.readyState == 4 && xhr.status == 200) {
            var students = JSON.parse(xhr.responseText);

            var output = "<table
border='1'><tr><th>ID</th><th>Name</th><th>Department</th></tr>";

            for (var i = 0; i < students.length; i++) {
              output += "<tr><td>" + students[i].id +
                  "</td><td>" + students[i].name +
                  "</td><td>" + students[i].dept + "</td></tr>";
            }

            output += "</table>";
            document.getElementById("result").innerHTML = output;
          }
        };

        xhr.send();
      }
    </script>
</head>

<body>
<h2>AJAX with JSON Example</h2>

<button onclick="loadStudents()">Load Student Data</button>
```


</body>

</html>

STEP 6: Add Project to Server

1. Open **Servers** tab
2. Right-click Tomcat → **Add and Remove**
3. Move **AjaxJsonDemo** to *Configured*
4. Click **Finish**

STEP 7: Run Program

Open browser:

http://localhost:8080/AjaxJsonDemo/index.html

OUTPUT EXPLANATION

• Page loads normally
• Button click triggers AJAX request
• JSON file is fetched **without page reload**
• Student data appears dynamically in table

EXPERIMENT 8: AJAX WITH SERVLET

AIM

To develop a web application where **AJAX fetches data from a Java Servlet**, demonstrating server-side processing and dynamic response handling.

SOFTWARE REQUIRED (FREEWARE)

- Eclipse IDE for Enterprise Java
- Apache Tomcat 9
- Browser: Google Chrome

STEP 1: Create New Dynamic Web Project

1. File → New → Dynamic Web Project
2. Project Name: **AjaxServletDemo**
3. Target Runtime: Apache Tomcat 9
4. Next → Next
5. ✓ Generate web.xml
6. Finish

STEP 2: Project Structure

AjaxServletDemo

```
├── src/main/java
└── src/main/webapp
    └── WEB-INF
        └── web.xml
```

STEP 3: Create Servlet

📍 Location:

src → main → java

1. Right-click java
2. New → Servlet
3. Package name: **com.ajax**
4. Class name: **StudentServlet**

5. Finish

CODE: StudentServlet.java

```
package com.ajax;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/students")
public class StudentServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    response.setContentType("application/json");
    PrintWriter out = response.getWriter();

    out.print("[");
    out.print("{\"id\":201,\"name\":\"Ravi\",\"dept\":\"AI\"},");
    out.print("{\"id\":202,\"name\":\"Sita\",\"dept\":\"DS\"},");
    out.print("{\"id\":203,\"name\":\"Kumar\",\"dept\":\"CSE\"}");
    out.print("]");
  }
```

}

STEP 4: Create HTML File

📍 Location:

src → main → webapp

1. Right-click webapp
2. New → HTML File
3. File name: **index.html**
4. Finish

CODE: index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>AJAX with Servlet</title>

  <script>
    function loadStudents() {
      var xhr = new XMLHttpRequest();
      xhr.open("GET", "students", true);

      xhr.onreadystatechange = function () {
        if (xhr.readyState == 4 && xhr.status == 200) {
          var students = JSON.parse(xhr.responseText);

          var output = "<ul>";
```

```
            for (var i = 0; i < students.length; i++) {
               output += "<li>" + students[i].name +
                     " - " + students[i].dept + "</li>";
            }
            output += "</ul>";

            document.getElementById("result").innerHTML = output;
          }
        };

        xhr.send();
      }
   </script>
</head>

<body>
<h2>AJAX with Servlet Example</h2>

<button onclick="loadStudents()">Get Student Data</button>

<div id="result"></div>

</body>
</html>
```

STEP 5: Add Project to Server

1. Servers tab
2. Add and Remove
3. Add **AjaxServletDemo**
4. Restart Tomcat

STEP 6: Run Program

http://localhost:8080/AjaxServletDemo/index.html

OUTPUT EXPLANATION

• Browser loads HTML page
• Button triggers AJAX request
• Servlet executes on server
• JSON response returned
• Data displayed dynamically
• No page refresh

🔑 **DIFFERENCE (VERY IMPORTANT FOR EXAM)**

AJAX with JSON	**AJAX with Servlet**
Static data	Dynamic data
No Java code	Uses Java Servlet
Frontend only	Frontend + Backend
Simple	Industry-level

EXPERIMENT 9 (Client-Side JSON)

STEP 1: Create Project in Eclipse

1. Open Eclipse → Workspace → Launch
2. File → New → Dynamic Web Project
    - Project Name: JSONClientDemo

- Target Runtime: Apache Tomcat 9 (optional, we just need workspace structure)
- Tick ✓ Generate web.xml → Finish

Folder structure you will use:

JSONClientDemo

└── src

└── src/main/webapp

  ├── index.html

  └── students.json

STEP 2: Create JSON File

1. Right-click src/main/webapp → New → File → Name: students.json

Code for students.json

```
[
  {"id":1,"name":"Ash","branch":"AI & DS","age":21},
  {"id":2,"name":"Priya","branch":"CSE","age":22},
  {"id":3,"name":"Muthu","branch":"IT","age":20},
  {"id":4,"name":"Anu","branch":"ECE","age":23}
]
```

STEP 3: Create HTML File

1. Right-click src/main/webapp → New → HTML File → Name: index.html

Code for index.html

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

```
  <title>Student Records - JSON Demo</title>
  <style>
    table, th, td {
      border: 1px solid black;
      border-collapse: collapse;
      padding: 5px;
    }
    th {
      background-color: #f2f2f2;
    }
  </style>
</head>
<body>

<h2>Student Records</h2>
<button onclick="loadStudents()">Load Students</button>
<br><br>
<table id="studentTable">
  <tr>
    <th>ID</th>
    <th>Name</th>
    <th>Branch</th>
    <th>Age</th>
  </tr>
</table>
```

```
<script>
function loadStudents() {
  fetch("students.json")
  .then(response => response.json())
  .then(students => {
    let table = document.getElementById("studentTable");

    // Clear previous rows except header
    table.innerHTML =
"<tr><th>ID</th><th>Name</th><th>Branch</th><th>Age</th></tr>";

    students.forEach(student => {
      let row = table.insertRow();
      row.insertCell(0).innerHTML = student.id;
      row.insertCell(1).innerHTML = student.name;
      row.insertCell(2).innerHTML = student.branch;
      row.insertCell(3).innerHTML = student.age;
    });
  })
  .catch(err => alert("Error loading JSON. Open using a local server, not file:// URL."));
}
</script>

</body>
</html>
```

STEP 4: Open in Browser

✅ **Important:** You **cannot open index.html directly via file:// in Chrome/Edge**.

Options to make it work:

Option A: Use Tomcat / Eclipse

1. Right-click index.html → **Run As → Run on Server**
2. Choose **Tomcat 9**
3. Open:

http://localhost:8080/JSONClientDemo/index.html

4. Click **Load Students** → Table populates ✅

Option B: Use lightweight local server

- If you don't want Tomcat, use Node.js http-server:

cd path\to\src\main\webapp

npx http-server

- Open the URL shown (http://127.0.0.1:8080) → Click **Load Students**

STEP 5: OUTPUT EXPLANATION

ID	Name	Branch	Age
1	Ash	AI & DS	21
2	Priya	CSE	22
3	Muthu	IT	20
4	Anu	ECE	23

Explanation:

- The **Load Students** button triggers JavaScript fetch()
- students.json is read and parsed as JSON
- JavaScript dynamically populates the table
- No page reload occurs — purely **client-side dynamic update**

💡 **Summary:**

- Your project **does not need a servlet**
- Chrome/Edge **require a local server** to fetch JSON
- Tomcat or Node.js http-server is the easiest way

EXPERIMENT 10 – LIBRARY MANAGEMENT SYSTEM USING SERVLET/JSP + JAVASCRIPT VALIDATION + DB CONNECTIVITY

AIM

To develop a Library Management System that validates user input on the registration form using JavaScript and stores the data into a MySQL database using Servlet/JSP.

SOFTWARE REQUIRED (FREEWARE)

- Eclipse IDE for Enterprise Java
- Apache Tomcat 9
- MySQL 9.6
- Browser: Chrome / Edge / Firefox

STEP 1 – Create Dynamic Web Project

1️⃣ Open Eclipse IDE
2️⃣ Go to: File → New → Dynamic Web Project
3️⃣ Fill in details:

- **Project Name:** LibraryManagementSystem
- **Target Runtime:** Apache Tomcat v9.0
- **Dynamic Web Module Version:** 4.0
  4️⃣ Click: Next → Next → Finish
  ✅ Project created successfully

STEP 2 – Understand Project Structure

You will see:

LibraryManagementSystem

├— Java Resources

│ └— src

└— Webapp

├— META-INF

└— WEB-INF

- All JSP/HTML files → WebContent
- All Servlet files → Java Resources/src

STEP 3 – Create Registration JSP Page

1️⃣ src->main->Right-click **Webapp → New → JSP File**
2️⃣ File name: register.jsp
3️⃣ Paste the following code:

```
<!DOCTYPE html>

<html>

<head>

<title>Library Registration</title>


<script>

function validateForm() {

  let fname = document.forms["regForm"]["fname"].value;

  let lname = document.forms["regForm"]["lname"].value;

  let email = document.forms["regForm"]["email"].value;

  let password = document.forms["regForm"]["password"].value;

  let mobile = document.forms["regForm"]["mobile"].value;

  let address = document.forms["regForm"]["address"].value;
```

```
let emailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;
let namePattern = /^[A-Za-z]+$/;

if (!namePattern.test(fname) || fname.length < 6) {
   alert("First Name must contain alphabets and minimum 6 characters");
   return false;
}

if (lname == "") {
   alert("Last Name cannot be empty");
   return false;
}

if (password.length < 6) {
   alert("Password must be at least 6 characters");
   return false;
}

if (!email.match(emailPattern)) {
   alert("Enter valid Email ID");
   return false;
}

if (mobile.length != 10 || isNaN(mobile)) {
   alert("Mobile number must contain 10 digits");
```

```
    return false;
  }

  if (address == "") {
    alert("Address cannot be empty");
    return false;
  }

  alert("Form Submitted");
  return true;
}
</script>

</head>
<body>

<h2>Library Registration Form</h2>

<form name="regForm" action="RegisterServlet" method="post" onsubmit="return validateForm()">
First Name: <input type="text" name="fname"><br><br>
Last Name: <input type="text" name="lname"><br><br>
Email: <input type="text" name="email"><br><br>
Password: <input type="password" name="password"><br><br>
Mobile: <input type="text" name="mobile"><br><br>
Address: <textarea name="address"></textarea><br><br>
<input type="submit" value="Register">
```

</form>

</body>

</html>

✅ This validates:

- First Name ≥ 6 chars, alphabets only
- Last Name not empty
- Password ≥ 6 chars
- Valid Email format
- Mobile = 10 digits
- Address not empty

STEP 4 – Create Servlet

1️⃣ Right-click **Java Resources → New → Servlet**
2️⃣ Fill details:

- **Package:** com.library
- **Class Name:** RegisterServlet
  3️⃣ Click **Finish**

4️⃣ Paste this code in RegisterServlet.java:

```
package com.library;


import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/RegisterServlet")
public class RegisterServlet extends HttpServlet {

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String fname = request.getParameter("fname");
    String lname = request.getParameter("lname");
    String email = request.getParameter("email");
    String password = request.getParameter("password");
    String mobile = request.getParameter("mobile");
    String address = request.getParameter("address");

    try {
      Class.forName("com.mysql.cj.jdbc.Driver");

      Connection con = DriverManager.getConnection(
```

```
            "jdbc:mysql://localhost:3306/librarydb?useSSL=false&serverTimezone=UTC",
            "root",
            "0000"
        );

        PreparedStatement ps = con.prepareStatement(
            "INSERT INTO users(fname,lname,email,password,mobile,address) VALUES
(?,?,?,?,?,?)"
        );

        ps.setString(1, fname);
        ps.setString(2, lname);
        ps.setString(3, email);
        ps.setString(4, password);
        ps.setString(5, mobile);
        ps.setString(6, address);

        ps.executeUpdate();

        out.println("<h2>Registration Successful</h2>");
        out.println("<a href='register.jsp'>Go Back</a>");

        con.close();

    } catch (Exception e) {
        e.printStackTrace();
        out.println("<h3>Error Occurred</h3>");
```

```
            out.println(e);
        }
    }
}
```

✅ **Note:** @WebServlet("/RegisterServlet") automatically maps the servlet URL.

STEP 5 – MySQL Database Setup

Check MySQL Installation (Very Important)

https://www.softpedia.com/get/Internet/Servers/Database-Utils/MySQL-4.shtml#download

then

DOWNLOAD: External mirror - v9.6.0 Innovation

Then initial proceedings should be done.

Once everything is done then

1️⃣ Open **MySQL Command Line**
2️⃣ Create database & table:

```
CREATE DATABASE librarydb;

USE librarydb;


CREATE TABLE users (
    fname VARCHAR(50),
    lname VARCHAR(50),
    email VARCHAR(100),
    password VARCHAR(50),
    mobile VARCHAR(15),
    address VARCHAR(200)
);
```

[example: mysql> CREATE DATABASE librarydb;

Query OK, 1 row affected (0.106 sec)

mysql> USE librarydb;

Database changed

mysql> CREATE TABLE users (

-> fname VARCHAR(50),

-> lname VARCHAR(50),

-> email VARCHAR(100),

-> password VARCHAR(50),

-> mobile VARCHAR(15),

-> address VARCHAR(200)

->);

Query OK, 0 rows affected (0.336 sec)**]**

STEP 6 – MySQL Connector Setup

1️⃣ Download MySQL Connector JAR:
https://dev.mysql.com/downloads/connector/j/

select:

- **Platform:** Operating System Independent

Download:
mysql-connector-j-9.0.xx.zip (based upon the installed version) (5.1mb)

2️⃣ Extract and save: C:\mysql-connector-j-9.6.0
3️⃣ Copy mysql-connector-j-9.6.0.jar to:
C:\tomcat9\lib

C:\tomcat9\lib\mysql-connector-j-9.6.0.jar

4️⃣ Add to Eclipse Build Path:

- Right-click Project (LibraryManagementSystem)→ **Build Path → Configure Build Path**
- Libraries → **Classpath → Add External JARs** → select mysql-connector-j.jar
- **Use Classpath**, NOT Module Path

✅ Why Classpath: Tomcat cannot read JDBC drivers from Module Path

You MUST have MySQL JAR in BOTH PLACES

✅ **1. Tomcat (MANDATORY)**

C:\tomcat9\lib\mysql-connector-j-9.6.0.jar

✅ **2. Eclipse Project**

Right-click Project → Properties → Java Build Path → Libraries
Add JAR to **Classpath** (NOT Modulepath)

[C:\mysql-connector-j-9.6.0 choose libraries in this path to avoid controversy]

STEP 7 – Run the Project

Add Project to Server

1. Go to **Servers tab** (bottom)
2. Right-click → **Add and Remove**
3. Move LibraryManagementSystem to **Configured**
4. Click **Finish**

Start Server

5. In **Servers tab**
6. Right-click **Tomcat v9.0 Server**
7. Click **Start**

1️⃣ Start Tomcat 9 in eclipse
2️⃣ Open chrome browser: http://localhost:8080/LibraryManagementSystem/register.jsp

3️⃣ Fill registration form → Click **Register**
4️⃣ If validation passes → Alert "Form Submitted" → Servlet stores data in MySQL → Shows "Registration Successful" Go Back

STEP 8 – Verify Database

SELECT * FROM users;

Example Output:

fname	lname	email	password	mobile	address
Aswin	Kumar	aswinja@examly.io	124325	1234567890	Some Address

✅ This confirms **Servlet ↔ Database ↔ JSP** connectivity is working perfectly.

ERRORS FACED IN THE PROJECT AND THEIR RESOLUTIONS

During the development and execution of the **Library Management System using JSP, Servlet, JavaScript, Tomcat 9, and MySQL**, the following errors were encountered. Each issue was analyzed and resolved as explained below.

ERROR 1: java.lang.ClassNotFoundException: com.mysql.cj.jdbc.Driver

Error Description

After submitting the registration form, the application displayed the following runtime error:

java.lang.ClassNotFoundException: com.mysql.cj.jdbc.Driver

Cause

This error occurred because the **MySQL JDBC Driver (Connector/J)** was not properly loaded by the application.
Tomcat could not locate the JDBC driver class at runtime.

Resolution

The issue was resolved by ensuring that the **MySQL Connector JAR file is present in both mandatory locations**:

✅ **1. Tomcat Library (MANDATORY for Runtime)**

C:\tomcat9\lib\mysql-connector-j-9.6.0.jar

Tomcat loads JDBC drivers only from its lib directory at runtime.

✅ **2. Eclipse Project Classpath (MANDATORY for Compilation)**

Steps:

1. Right-click Project → **Properties**
2. Java Build Path → **Libraries**
3. Click **Add External JARs**
4. Select:

C:\mysql-connector-j-9.6.0\mysql-connector-j-9.6.0.jar

5. Ensure it is added to **Classpath**
6. Click **Apply → Close**

❌ **Module Path was NOT used**, because:

- Servlet/JSP projects are **not modular**
- Tomcat does not load JDBC drivers from Module Path

Final Outcome

After placing the JAR in both locations, the JDBC driver loaded successfully and the database connection worked.

ERROR 2: HTTP Status 404 – RegisterServlet Not Found

Error Description

When accessing:

http://localhost:8080/LibraryManagementSystem/RegisterServlet

The browser displayed:

HTTP Status 404 – Not Found

Cause

The servlet URL mapping was missing or incorrectly defined.
Tomcat could not recognize the servlet endpoint.

Resolution

Since **Tomcat 9.0** is used, **annotation-based servlet mapping** was applied correctly.

The following two lines were verified and added to the servlet code:

import javax.servlet.annotation.WebServlet;

@WebServlet("/RegisterServlet")

These annotations explicitly map the servlet class to the URL /RegisterServlet.

Final Outcome

Once the servlet annotation was correctly configured, Tomcat successfully routed requests from register.jsp to RegisterServlet.

ERROR 3: Servlet Import Errors (jakarta vs javax)

Error Description

Compilation errors appeared when using:

import jakarta.servlet.*;

Cause

The project was running on **Apache Tomcat 9**, which uses the **javax.servlet** namespace. The **jakarta.servlet** package is applicable only for **Tomcat 10 and above**.

Resolution

All servlet imports were corrected to use **javax**:

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

Final Outcome

Servlet compiled and executed correctly under Tomcat 9.

ERROR 4: Confusion Between Module Path and Classpath

Error Description

Despite adding the MySQL JAR, the ClassNotFoundException persisted.

Cause

The JDBC driver JAR was initially added to **Module Path** instead of **Classpath**.

Resolution

The JAR was removed from Module Path and added strictly to:

Java Build Path → Libraries → Classpath

Reason

- Module Path is used only for Java 9+ modular applications
- Servlet/JSP applications are **non-modular**
- Tomcat reads JDBC drivers only from Classpath

Final Outcome

The driver was detected correctly and database connectivity succeeded.

ERROR 5: Database Connection Failure (Initial Setup)

Error Description

Data was not inserted into the database during initial attempts.

Cause

- Database or table was not created
- MySQL service was not running

Resolution

The following commands were executed in MySQL Command Line:

CREATE DATABASE librarydb;

USE librarydb;

```
CREATE TABLE users (
   fname VARCHAR(50),
   lname VARCHAR(50),
   email VARCHAR(100),
   password VARCHAR(50),
   mobile VARCHAR(15),
   address VARCHAR(200)
);
```

Additionally, MySQL service was verified to be running.

Final Outcome

Data was successfully inserted and retrieved using:

SELECT * FROM users;

SUMMARY OF ERRORS & FIXES

Error Faced	Root Cause	Solution
ClassNotFoundException	JDBC JAR missing	Add JAR to Tomcat lib + Classpath
HTTP 404	Servlet not mapped	Use @WebServlet("/RegisterServlet")
Import errors	Wrong servlet package	Use javax.servlet for Tomcat 9
JDBC not loading	JAR in Module Path	Move to Classpath
DB insert failed	DB/Table not created	Create DB and table

EXPERIMENT 11 – JavaScript Registration Form Validation

AIM

To create a registration form in HTML and validate user input using JavaScript for:

- First Name

- Last Name
- Password
- Email ID
- Mobile Number
- Address

SOFTWARE REQUIRED (FREEWARE)

- **Eclipse IDE for Enterprise Java** (or Eclipse IDE for Java Developers)
- **Browser**: Chrome / Edge / Firefox

Note: This is **pure client-side**. No Tomcat required.

STEP 1: Open Eclipse

1. Launch Eclipse IDE
2. Choose your workspace (e.g., C:\eclipse-workspace)
3. Click **Launch**

STEP 2: Create a Static Web Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name**: RegistrationValidationDemo
    - **Target Runtime**: <None> (since it's client-side only)
3. Click **Next → Next**
4. **Do NOT** tick Generate web.xml (optional for client-side only)
5. Click **Finish**

You will see the project in **Project Explorer**:

RegistrationValidationDemo

├— src

└── WebContent

STEP 3: Create HTML File

📍 **Location**: WebContent

1. Right-click **WebContent → New → HTML File**
2. **File Name**: registration.html
3. Click **Finish**

STEP 4: Copy-Paste HTML + JavaScript Code

📍 **registration.html**

```
<!DOCTYPE html>
<html>
<head>
  <title>Registration Form Validation</title>
  <script>
    function validateForm() {
      let firstName = document.forms["regForm"]["fname"].value.trim();
      let lastName = document.forms["regForm"]["lname"].value.trim();
      let password = document.forms["regForm"]["password"].value.trim();
      let email = document.forms["regForm"]["email"].value.trim();
      let mobile = document.forms["regForm"]["mobile"].value.trim();
      let address = document.forms["regForm"]["address"].value.trim();

      // First Name validation
      let nameRegex = /^[A-Za-z]{6,}$/;
      if (!nameRegex.test(firstName)) {
```

```
    alert("First Name must be alphabets and at least 6 characters.");
    return false;
}

// Last Name validation
if (lastName === "") {
    alert("Last Name cannot be empty.");
    return false;
}

// Password validation
if (password.length < 6) {
    alert("Password must be at least 6 characters.");
    return false;
}

// Email validation
let emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-z]{2,}$/;
if (!emailRegex.test(email)) {
    alert("Enter a valid email address (example: name@domain.com).");
    return false;
}

// Mobile number validation
let mobileRegex = /^[0-9]{10}$/;
if (!mobileRegex.test(mobile)) {
```

```
            alert("Mobile number must be exactly 10 digits.");
            return false;
        }

        // Address validation
        if (address === "") {
            alert("Address cannot be empty.");
            return false;
        }

        alert("Form submitted successfully!");
        return true;
    }
  </script>
</head>
<body>

<h2>Registration Form</h2>
<form name="regForm" onsubmit="return validateForm()">
  First Name: <input type="text" name="fname"><br><br>
  Last Name: <input type="text" name="lname"><br><br>
  Password: <input type="password" name="password"><br><br>
  Email: <input type="email" name="email"><br><br>
  Mobile: <input type="text" name="mobile"><br><br>
  Address: <textarea name="address"></textarea><br><br>
  <input type="submit" value="Register">
```

</form>

</body>

</html>

STEP 5: Run the HTML File in Browser

1. Right-click **registration.html** → **Open With → Web Browser (Chrome recommended)**
2. Or double-click registration.html → Opens in default browser

STEP 6: Test Validations

Field	**Example Invalid Input**	**Expected Result**
First Name	abc, 123	Alert: "First Name must be alphabets and at least 6 characters."
Last Name	Blank	Alert: "Last Name cannot be empty."
Password	12345	Alert: "Password must be at least 6 characters."
Email	abc@, xyz.com	Alert: "Enter a valid email address."
Mobile	12345, abcdefghij	Alert: "Mobile number must be exactly 10 digits."
Address	Blank	Alert: "Address cannot be empty."

- Fill **all fields correctly** → Alert: "Form submitted successfully!"

STEP 7: Output Explanation

- The **JavaScript function** validateForm() checks all fields when the **Submit** button is clicked.
- Invalid fields trigger **alerts** and **block form submission**.
- Correct data triggers a **success alert**.

- Works **purely client-side**, so no server or Tomcat needed.

✅ **PROJECT STRUCTURE IN ECLIPSE**

RegistrationValidationDemo

├── src

└── WebContent

  └── registration.html

EXPERIMENT 12 – Number to Words (0 to 999) using JavaScript

AIM

To create an **HTML page with JavaScript** that accepts a number between **0 and 999** from a text field and displays the **number in words**, while rejecting:

- Numbers with **4 or more digits**
- **Alphabets**
- **Special characters**

SOFTWARE REQUIRED (FREEWARE)

- Eclipse IDE (any version)
- Browser: Google Chrome / Edge / Firefox
  ✅ **No Tomcat required**

STEP 1: Open Eclipse

1. Open **Eclipse IDE**
2. Select your **Workspace**
3. Click **Launch**

STEP 2: Create New Project

1. Go to **File → New → Dynamic Web Project**
2. Enter:
    - **Project Name**: NumberToWords
    - **Target Runtime**: <None>
3. Click **Next → Next**
4. ❌ Do **NOT** tick *Generate web.xml*
5. Click **Finish**

STEP 3: Project Structure

You will see:

NumberToWords

├── src

└── WebContent

👉 **HTML + JavaScript go inside WebContent**

STEP 4: Create HTML File

📍 Location:
WebContent

1. Right-click **WebContent**
2. Click **New → HTML File**
3. File Name: numbertowords.html
4. Click **Finish**

STEP 5: COPY–PASTE CODE

📍 **numbertowords.html**

<!DOCTYPE html>

```
<html>
<head>
  <title>Number to Words</title>

  <script>
    function convertToWords() {

      var num = document.getElementById("number").value;

      // Validation
      if (num === "") {
        alert("Please enter a number");
        return;
      }

      if (!/^[0-9]+$/.test(num)) {
        alert("Only numbers are allowed");
        return;
      }

      if (num.length > 3) {
        alert("Enter a number between 0 and 999 only");
        return;
      }

      num = parseInt(num);
```

```
  var ones = ["Zero","One","Two","Three","Four","Five","Six","Seven","Eight","Nine"];
  var teens = ["Ten","Eleven","Twelve","Thirteen","Fourteen","Fifteen",
          "Sixteen","Seventeen","Eighteen","Nineteen"];
  var tens = ["","","Twenty","Thirty","Forty","Fifty","Sixty","Seventy","Eighty","Ninety"];

  var words = "";

  if (num >= 100) {
    words += ones[Math.floor(num / 100)] + " Hundred ";
    num = num % 100;
  }

  if (num >= 10 && num <= 19) {
    words += teens[num - 10];
  } else if (num >= 20) {
    words += tens[Math.floor(num / 10)] + " ";
    words += ones[num % 10];
  } else if (num > 0) {
    words += ones[num];
  }

  document.getElementById("output").innerHTML = words;
}
```

```
  </script>
</head>

<body>

<h2>Number to Words (0 to 999)</h2>

Enter Number:
<input type="text" id="number">
<br><br>

<input type="button" value="Convert" onclick="convertToWords()">

<h3>Output:</h3>
<p id="output"></p>

</body>
</html>
```

STEP 6: Run the Program

1. Right-click **numbertowords.html**
2. Click **Open With → Web Browser**
3. Choose **Chrome (recommended)**

STEP 7: SAMPLE OUTPUT

Input	**Output**
5	Five

23	Twenty Three
105	One Hundred Five
999	Nine Hundred Ninety Nine

INVALID INPUT BEHAVIOR

Input	**Result**
1000	❌ Alert: Enter 0–999
ab12	❌ Only numbers allowed
@#	❌ Only numbers allowed
empty	❌ Please enter a number