JSP, Servlet, JDBC in Java by Zohair Ahmed

YouTube ZohairAhmed

Welcome to this tutorial on JSP, Servlet, and JDBC integration in Java.
Check out my YouTube channel for video guides and tutorials!

MySQL table preview that displays the structure of the `users` table from the database. It

Column Name	Data Type	Description	Notes
id	INT (Auto-Increment, PK)	Unique user identifier	Primary key, auto-generated
name	VARCHAR(100)	User's full name	Max 100 characters
email	VARCHAR(100)	User's email address	Max 100 characters

`UserServlet` Explained — Step by Step

```
@WebServlet("/UserServlet")
public class UserServlet extends HttpServlet {
```

- This servlet listens at the URL pattern `/UserServlet`.
- It extends `HttpServlet` to handle HTTP requests.

---

1. Database Connection Constants

```
private static final String URL = "jdbc:mysql://localhost:3306/testdb";
private static final String USER = "root";
private static final String PASS = "password"; // Change to your DB password
```

- These constants store the MySQL connection details.
- `URL` includes the DB hostname, port, and database name.
- `USER` and `PASS` are your DB credentials.

---

2. The `doGet()` Method

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

- Handles all **GET** requests — usually for showing pages and deleting users.

2.1 Reading the `action` parameter

```
String action = request.getParameter("action");
if (action == null) action = "list";
```

- The servlet decides what to do based on `action` parameter in the URL.
- If no action specified, it defaults to `"list"` (show all users).

2.2 Load JDBC driver and open DB connection

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

- Loads the MySQL JDBC driver to enable Java-MySQL communication.
- Establishes a connection to the database.

2.3 Handling `delete` action

```
if (action.equals("delete")) {
    int id = Integer.parseInt(request.getParameter("id"));
    PreparedStatement ps = conn.prepareStatement("DELETE FROM users WHERE id = ?");
    ps.setInt(1, id);
    ps.executeUpdate();
    ps.close();
    conn.close();
    response.sendRedirect("UserServlet");
    return;
}
```

- Reads `id` parameter from the request (which user to delete).
- Prepares a SQL DELETE statement with `id` as parameter to avoid SQL injection.
- Executes the delete.
- Closes resources.
- Redirects to `/UserServlet` (default action: list all users) — so user sees updated list.
- `return` to stop further processing.

2.4 Handling `edit` action

```
if (action.equals("edit")) {
    int id = Integer.parseInt(request.getParameter("id"));
    PreparedStatement ps = conn.prepareStatement("SELECT * FROM users WHERE id = ?");
    ps.setInt(1, id);
    ResultSet rs = ps.executeQuery();
    if (rs.next()) {
        request.setAttribute("userId", rs.getInt("id"));
        request.setAttribute("userName", rs.getString("name"));
        request.setAttribute("userEmail", rs.getString("email"));
    }
    rs.close();
```

```
    ps.close();
    conn.close();
    request.getRequestDispatcher("userform.jsp").forward(request, response);
    return;
}
```

- Gets `id` from request to know which user to edit.
- Runs SQL SELECT query to fetch user info.
- Reads user data from the `ResultSet`.
- Sets the user data as **request attributes** to pre-fill the form fields.
- Closes DB resources.
- Forwards the request to `userform.jsp` — the JSP will use attributes to show current user data.
- `return` to stop processing.

2.5 Default action: list all users

```
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM users");
request.setAttribute("users", rs);
request.getRequestDispatcher("userlist.jsp").forward(request, response);
rs.close();
stmt.close();
conn.close();
```

- Runs SQL query to get all users.
- Stores the `ResultSet` in the request attribute named `"users"`.
- Forwards to `userlist.jsp` to display the users.
- Closes DB resources.

---

3. The `doPost()` Method

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

- Handles **POST** requests — form submissions for add/update user.

3.1 Read form parameters

```
String id = request.getParameter("id");
String name = request.getParameter("name");
String email = request.getParameter("email");
```

- Gets submitted form data from the request.

3.2 Open DB connection again

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

3.3 Insert or Update logic

```
if (id == null || id.isEmpty()) {
    // Insert new user
    PreparedStatement ps = conn.prepareStatement("INSERT INTO users(name,email)
VALUES(?,?)");
    ps.setString(1, name);
    ps.setString(2, email);
    ps.executeUpdate();
    ps.close();
} else {
    // Update existing user
    PreparedStatement ps = conn.prepareStatement("UPDATE users SET name=?, email=?
WHERE id=?");
    ps.setString(1, name);
    ps.setString(2, email);
    ps.setInt(3, Integer.parseInt(id));
    ps.executeUpdate();
    ps.close();
}
conn.close();
response.sendRedirect("UserServlet");
```

- If `id` is missing → this is a **new user**, so INSERT into DB.
- If `id` exists → this is an **update** for an existing user.
- Uses prepared statements to prevent SQL injection.
- Executes SQL queries.
- Closes connection.
- Redirects to list page to show updated user list.

Summary: What Happens Internally?

Action	Request Type	Parameters	Servlet Behavior	JSP Forwarded To
List Users	GET	None	Select all users, set in request attribute	`userlist.jsp`
Add New User	GET	None (opens form)	Show empty form	`userform.jsp`
Save New User	POST	name, email	Insert user in DB, redirect to list	Redirect to list
Edit User Form	GET	action=edit, id	Select user by ID, set attributes for form pre-fill	`userform.jsp`
Update User	POST	id, name, email	Update user record, redirect to list	Redirect to list
Delete	GET	action=delete,	Delete user record, redirect	Redirect

User		id	to list	to list

A clear, step-by-step explanation of both JSP files — **`userlist.jsp`** and **`userform.jsp`** — to help understand how they work and fit into the servlet-JSP-JDBC CRUD flow.

`userlist.jsp` Explained — Displaying User List

```
<%@ page import="java.sql.*" %>
<%
    ResultSet rs = (ResultSet) request.getAttribute("users");
%>
```

- **Import `java.sql.*` :** We need `ResultSet` to iterate through the database query results.
- **Retrieve the `users` attribute:** The servlet sets this attribute with a `ResultSet` containing all users.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>User List</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.4.3/dist/css/bootstrap.min.css"
rel="stylesheet" />
</head>
<body class="bg-light">
<div class="container mt-5">
    <h2 class="mb-4">User List</h2>
    <a href="userform.jsp" class="btn btn-primary mb-3">Add New User</a>
```

- **HTML structure:** Sets up a Bootstrap styled container.
- **Add New User button:** Links to `userform.jsp` (empty form for adding a new user).

```
<table class="table table-bordered table-hover bg-white">
    <thead class="table-dark">
        <tr>
            <th>ID</th><th>Name</th><th>Email</th><th style="width:
150px;">Actions</th>
        </tr>
    </thead>
    <tbody>
    <%
        while(rs.next()) {
    %>
        <tr>
            <td><%= rs.getInt("id") %></td>
```

```
            <td><%= rs.getString("name") %></td>
            <td><%= rs.getString("email") %></td>
            <td>
                <a href="UserServlet?action=edit&id=<%= rs.getInt("id") %>" class="btn
btn-sm btn-warning">Edit</a>
                <a href="UserServlet?action=delete&id=<%= rs.getInt("id") %>"
class="btn btn-sm btn-danger" onclick="return confirm('Are you sure?');">Delete</a>
            </td>
        </tr>
    <%
        }
        rs.close();
    %>
    </tbody>
</table>
```

- **Table headers:** Define columns for ID, Name, Email, and Actions.
- **Iterate over users:**
  - `while(rs.next())` loops over each row in the `ResultSet`.
  - `<%= ... %>` outputs user data in table cells.
- **Actions:**
  - **Edit:** Links to the servlet with `action=edit` and the user's `id`.
  - **Delete:** Links to the servlet with `action=delete` and user's `id` — shows a JS confirm dialog.
- **Close ResultSet:** Good practice to release DB resources.

```
</div>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.4.3/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>
```

- Includes Bootstrap JS bundle for responsive UI components.

What Happens When This JSP Runs?

- Servlet queries users and passes a `ResultSet` named `"users"`.
- JSP loops over this data and creates a table row for each user.
- Users see a styled list with options to **edit** or **delete**.
- Clicking buttons sends requests back to `UserServlet` with appropriate actions.

`userform.jsp` Explained — Add/Edit User Form

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>User Form</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.4.3/dist/css/bootstrap.min.css"
rel="stylesheet" />
</head>
<body class="bg-light">
<div class="container mt-5">
```

- Basic HTML setup with Bootstrap styles.

```
<h2 class="mb-4"><%= request.getAttribute("userId") == null ? "Add New User" : "Edit
User" %></h2>
```

- **Dynamic heading:**
    - If `userId is null` → we are adding a new user.
    - Otherwise → editing an existing user.

```
<form action="UserServlet" method="post" class="bg-white p-4 rounded shadow-sm">
    <input type="hidden" name="id" value="<%= request.getAttribute("userId") != null ?
request.getAttribute("userId") : "" %>" />
```

- The form posts to the servlet (`UserServlet`) via **POST** method.
- Hidden field for `id` — **empty if adding** and **filled if editing**, so servlet knows whether to insert or update.

```
    <div class="mb-3">
        <label for="name" class="form-label">Name</label>
        <input id="name" name="name" type="text" class="form-control" required
               value="<%= request.getAttribute("userName") != null ?
request.getAttribute("userName") : "" %>" />
    </div>
    <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input id="email" name="email" type="email" class="form-control" required
               value="<%= request.getAttribute("userEmail") != null ?
request.getAttribute("userEmail") : "" %>" />
    </div>
```

- Two input fields:
    - **Name** — text input, required
    - **Email** — email input, required
- Each input's `value` attribute is **pre-filled** if editing (`request.getAttribute(...)` returns a value).
- If adding, inputs start empty.

```
    <button type="submit" class="btn btn-success">Save</button>
    <a href="UserServlet" class="btn btn-secondary ms-2">Back to List</a>
</form>
```

- Submit button sends the form data to `UserServlet`.
- Back button returns user to the user list.

```
</div>
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.4.3/dist/js/bootstrap.bundle.min.js">
</script>
</body>
</html>
```

- Bootstrap JS for UI enhancements.

What Happens When This JSP Runs?

- For **Add New User**, fields are blank; user fills out name and email.
- For **Edit User**, fields are pre-filled with the user's current data.
- User submits the form:
    - Data is sent via POST to `UserServlet`.
    - Servlet decides to insert or update based on presence of `id`.

Summary

JSP File	Purpose	Important Features
`userlist.jsp`	Show all users with action links	Loops over ResultSet, creates table rows, uses `<%= %>` to display data
`userform.jsp`	Add or edit user form	Uses request attributes to pre-fill form fields, hidden `id` to distinguish add/update

---