Lab Exercise 3- Pagination, Filtering, and Sorting Techniques using Java Servlets

Objective

To implement server-side pagination, filtering, and sorting using plain Java Servlets (no Spring), storing sample data in memory. The lab teaches how API endpoints support query parameters to control list behavior.

Skills Covered

1. Handling URL query parameters in Servlets
2. Implementing pagination logic
3. Implementing filtering using request parameters
4. Implementing sorting in ascending or descending order
5. Returning JSON responses
6. Testing using Postman / Bruno / Browser

---

1. Project Setup

Create a Dynamic Web Project in Eclipse:

Project Name: PaginationFilterSort

Package: com.api

Servlet Name: UserListServlet

Add Tomcat 9 or 10 as the server.

---

2. Create the Servlet: UserListServlet.java

```
package com.api;


import java.io.IOException;

import java.util.*;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.*;


@WebServlet("/users")

public class UserListServlet extends HttpServlet {


    private static final long serialVersionUID = 1L;


    // Sample dataset

    private List<Map<String, Object>> users = new ArrayList<>();


    @Override

    public void init() throws ServletException {

        // Initial dataset (20 records)

        for (int i = 1; i <= 20; i++) {

            Map<String, Object> user = new HashMap<>();

            user.put("id", i);

            user.put("name", "User " + i);
            user.put("age", 20 + (i % 5)); // random age range 20-24
            users.add(user);
        }
    }

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

        // Step 1: Read pagination parameters
        int page = parseIntOrDefault(request.getParameter("page"), 1);
        int size = parseIntOrDefault(request.getParameter("size"), 5);

        // Step 2: Read filtering parameter
        String filterName = request.getParameter("name"); // optional

        // Step 3: Read sorting parameter
        String sortBy = request.getParameter("sortBy"); // "id", "name", "age"
        String sortOrder = request.getParameter("order"); // "asc" or "desc"

        // Step 4: Apply filtering
        List<Map<String, Object>> filteredList = new ArrayList<>(users);
        if (filterName != null && !filterName.isEmpty()) {
            filteredList.removeIf(u ->
                !u.get("name").toString().toLowerCase().contains(filterName.toLowerCase())
            );
        }

        // Step 5: Apply sorting
        if (sortBy != null && !sortBy.isEmpty()) {
            filteredList.sort((u1, u2) -> {
                Comparable v1 = (Comparable) u1.get(sortBy);
                Comparable v2 = (Comparable) u2.get(sortBy);
                return v1.compareTo(v2);
            });

            if ("desc".equalsIgnoreCase(sortOrder)) {
                Collections.reverse(filteredList);
            }
        }

        // Step 6: Apply pagination
        int start = (page - 1) * size;
        int end = Math.min(start + size, filteredList.size());

        List<Map<String, Object>> paginatedList = new ArrayList<>();

        if (start < filteredList.size()) {
```

```
            paginatedList = filteredList.subList(start, end);
        }

        // Step 7: Build JSON manually
        String json = buildJsonResponse(page, size, filteredList.size(), paginatedList);

        response.setContentType("application/json");
        response.getWriter().write(json);
    }

    private int parseIntOrDefault(String value, int defaultValue) {
        try {
            return (value == null) ? defaultValue : Integer.parseInt(value);
        } catch (Exception e) {
            return defaultValue;
        }
    }

    private String buildJsonResponse(int page, int size, int total, List<Map<String,
Object>> list) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        sb.append("\"page\":").append(page).append(",");
        sb.append("\"size\":").append(size).append(",");
        sb.append("\"total\":").append(total).append(",");
        sb.append("\"data\":[");

        for (int i = 0; i < list.size(); i++) {
            Map<String, Object> u = list.get(i);
            sb.append("{");
            sb.append("\"id\":").append(u.get("id")).append(",");
            sb.append("\"name\":\"").append(u.get("name")).append("\",");
            sb.append("\"age\":").append(u.get("age"));
            sb.append("}");
            if (i < list.size() - 1) sb.append(",");
        }

        sb.append("]}");
        return sb.toString();
    }
}
```

---

3. Test the API

Endpoint

```
GET http://localhost:8080/PaginationFilterSort/users
```

A. Pagination

```
GET /users?page=2&size=5
```

Returns users 6–10.

B. Filtering

```
GET /users?name=User 1
```

Matches: User 1, User 10, User 11, User 12, etc.

C. Sorting

Ascending:

```
GET /users?sortBy=name&order=asc
```

Descending:

```
GET /users?sortBy=age&order=desc
```

D. Combine All

```
GET /users?page=1&size=3&name=User&sortBy=id&order=desc
```

---

4. Add Servlet Mapping in web.xml (only if not using annotations)

```
<servlet>

   <servlet-name>UserListServlet</servlet-name>

   <servlet-class>com.api.UserListServlet</servlet-class>

</servlet>



<servlet-mapping>

   <servlet-name>UserListServlet</servlet-name>

   <url-pattern>/users</url-pattern>

</servlet-mapping>
```

---

5. Output Example (page=1, size=3)

```
{
  "page": 1,
  "size": 3,
  "total": 20,
  "data": [
    { "id": 1, "name": "User 1", "age": 21 },
    { "id": 2, "name": "User 2", "age": 22 },
    { "id": 3, "name": "User 3", "age": 23 }
  ]
}
```