**LAB3 --- ADVANCE WEB SORTING**

**1. Introduction**

This project aims to design and implement a RESTful API following HATEOAS principles using Java Servlets. It will also integrate several sorting algorithms (Heap Sort, Quick Sort, Merge Sort, Radix Sort, and Bucket Sort) for data processing. The application will be deployed on an Apache Tomcat server, which will handle the HTTP protocols and ensure proper request and response management.

In this project, I implemented the RESTful API around Student management system.

**2. Objectives**

1. Design and implement a RESTful API using Java Servlets that follows HATEOAS (Hypermedia As The Engine of Application State) principles.
2. Configure and deploy the application on an Apache Tomcat web server, ensuring proper handling of HTTP protocols and integration with the API.
3. Develop a servlet-based web application that showcases the use of dependency injection, basic servlets, and application configuration.
4. Implement and integrate sorting algorithms such as Heap Sort, Quick Sort, Merge Sort, Radix Sort, and Bucket Sort for data manipulation.
5. Provide a user interface for selecting and executing different sorting algorithms on a sample dataset.

**3. Technologies Used**

1. **Java Servlets:** For building the RESTful API and handling HTTP requests and responses.
2. **Apache Tomcat:** To deploy and run the application as a web server.
3. **HATEOAS Principles:** For designing the API to provide clients with discoverable links to related resources.
4. **Sorting Algorithms:** Implementations of Heap Sort, Quick Sort, Merge Sort, Radix Sort, and Bucket Sort.
5. **Maven:** For project management and dependencies (if applicable).

**4. System Architecture**

The application is built using the **Model-View-Controller (MVC)** pattern :

1. **Model:** Data processing of the student data
2. **View:** Simple user interface for interacting with the API and selecting sorting algorithms.
3. **Controller:** Servlets that manage HTTP requests and responses, implementing the RESTful API.

**A. HTTP Protocols & Apache Tomcat Web Server**

1. **Apache Tomcat Configuration:**
   1. The application is deployed on **Apache Tomcat** to handle HTTP requests and responses.
   2. The web server listens for HTTP requests, processes them through servlets, and returns responses in the appropriate format (e.g., JSON).
   3. Tomcat is configured to handle multiple servlets and map them to relevant API endpoints.
2. **HTTP Protocols Handling:**
   1. The server is configured to ensure proper handling of HTTP requests such as GET, POST, PUT, and DELETE.
   2. Response status codes (e.g., 200 OK, 404 Not Found) are managed within the servlet logic to give appropriate feedback to users.

**API Design**

1. **RESTful API with HATEOAS:**
   1. A RESTful API is designed following **HATEOAS** principles, where the API responds with links to related resources.
   2. Example: When retrieving a resource, the response includes a link to edit or delete it, ensuring the API is self-descriptive.
2. **CRUD Operations:**
   1. Implemented basic **CRUD (Create, Read, Update, Delete)** operations for managing a sample dataset (e.g., numbers or items) using servlets.
   2. Each resource in the dataset has its own endpoint, and links to related actions are provided using HATEOAS.

**Sample endpoints:**

1. GET /api/items

* Description: Retrieves the list of students.

[

{

"id": 1,

"name": "John Doe",

"age": 20,

"grade": "A",

"links": {

"self": "/api/items/1",

"edit": "/api/items/1/edit",

"delete": "/api/items/1/delete"

}

},

{

"id": 2,

"name": "Jane Smith",

"age": 22,

"grade": "B",

"links": {

"self": "/api/items/2",

"edit": "/api/items/2/edit",

"delete": "/api/items/2/delete"

}

}

]

2. POST /api/items

1. Description: Creates a new student.
2. Sample Request Body:

{

"name": "New Student",

"age": 18,

"grade": "C"

}

1. Sample Response:

{

"message": "Student created successfully",

"location": "/api/items/3"

}

3. GET /api/items/{id}

* Description: Retrieves a specific student by ID.
* Sample Response:

{

"id": 1,

"name": "John Doe",

"age": 20,

"grade": "A",

"links": {

"self": "/api/items/1",

"edit": "/api/items/1/edit",

"delete": "/api/items/1/delete"

}

}

4. PUT /api/items/{id}

1. Description: Updates an existing student by ID.
2. Sample Request Body:

{

"name": "Johnathan Doe",

"age": 21,

"grade": "A+"

}

1. Sample Response:

{

"message": "Student updated successfully"

}

5. DELETE /api/items/{id}

1. Description: Deletes a student by ID.
2. Sample Response:

{

"message": "Student deleted successfully"

}

These endpoints provide a basic CRUD (Create, Read, Update, Delete) structure for managing students, along with sample JSON bodies.

**C. Servlet-based Spring Integration**

Since **Spring Framework** was not used, the servlet-based approach allows us to simulate dependency injection and basic functionality:

1. **Servlet Configuration:**
   1. Servlets are configured in the **web.xml** file or by using **annotations** (@WebServlet).
2. **Dependency Injection:**
   1. Servlets simulate dependency injection by utilizing initialization parameters and context configuration for managing the required objects.

**D. Sorting Algorithms**

Five different sorting algorithms are implemented for data processing:

1. **Heap Sort:**
   1. A comparison-based sorting algorithm that uses a binary heap data structure.
2. **Quick Sort:**
   1. A divide-and-conquer algorithm that partitions the dataset into subarrays and recursively sorts them.
3. **Merge Sort:**
   1. A divide-and-conquer algorithm that splits the array into smaller subarrays, sorts them, and merges them.
4. **Radix Sort:**
   1. A non-comparative sorting algorithm that sorts numbers digit by digit.
5. **Bucket Sort:**
   1. A sorting algorithm that distributes the elements into buckets, sorts them individually, and then merges them.

**6. Implementation Details**

**Servlets for CRUD Operations**

1. **HomeController.java**: Handles all CRUD operations.
   1. In the doGet() method, the servlet retrieves all students and sends them in the response with a link to each item for further actions.
   2. In the doPost() method, the servlet creates a new student and responds with a success message and a link to view the newly created item.
   3. In the doPut() method to update an existing student
   4. In the doDelete() method to delete a student by Id

**8. Conclusion**

This project successfully demonstrates the design and implementation of a RESTful API using Java Servlets, following HATEOAS principles. It also integrates multiple sorting algorithms into the application, providing a user-friendly interface to interact with different sorting techniques. The project is deployed on an Apache Tomcat server, ensuring proper HTTP request handling and scalability.

**9. Future Enhancements**

1. Add more advanced features such as authentication, authorization, or advanced error handling.
2. Implement asynchronous processing for sorting algorithms to improve performance for larger datasets.
3. Enhance the user interface with more interactive elements like charts to visualize the sorting process.