**Advanced Software Engineering Final Project Report**

**Project Title:** Software-Based Load Balancer for Efficient Resource Management

**GitHub Repository Link:** <https://github.com/yashjosh-ux/LoadBalancer.git>

**Problem Description**

As demand grows, servers risk becoming overloaded, leading to potential downtimes, slower response times, and an overall subpar user experience. Traditional hardware load balancers are expensive and may lack the flexibility needed for rapid scaling in cloud-native environments. The project seeks to solve this by creating a flexible, cost-effective software load balancer capable of adapting to real-time traffic variations.

**Approach and Techniques**

**A diagram of a load balancing system

Description automatically generated**

**Figure:** Concept of Load Balancing

Key principles in software engineering were employed in the design and implementation of a software-based load balancer to address the problem. The following approaches and techniques were employed:

1. **Load Balancing Algorithms:**

* **Round-Robin Algorithm:** The algorithm spreads incoming requests in sequence across available servers, thus ensuring uniform traffic distribution.
* **Least-Connections Algorithm:** It routes the requests to a server with fewer active connections, balancing thus the load with server capacity.

1. **Health Monitoring:**

* Round-the-clock monitoring of the health of servers to detect non-responding servers. Traffic automatically gets rerouted when the detection of a failure occurs in an operational server.

1. **Incremental Development:**

* Implemented in an incremental fashion: first a simple skeleton of the load balancer was implemented, with a gradual implementation and testing of algorithms.

1. **Testing and Evaluation:**

* The performance of the system was measured under low, medium, and peak conditions using simulated traffic workloads. The metrics included efficiency of traffic distribution.

A screenshot of a computer

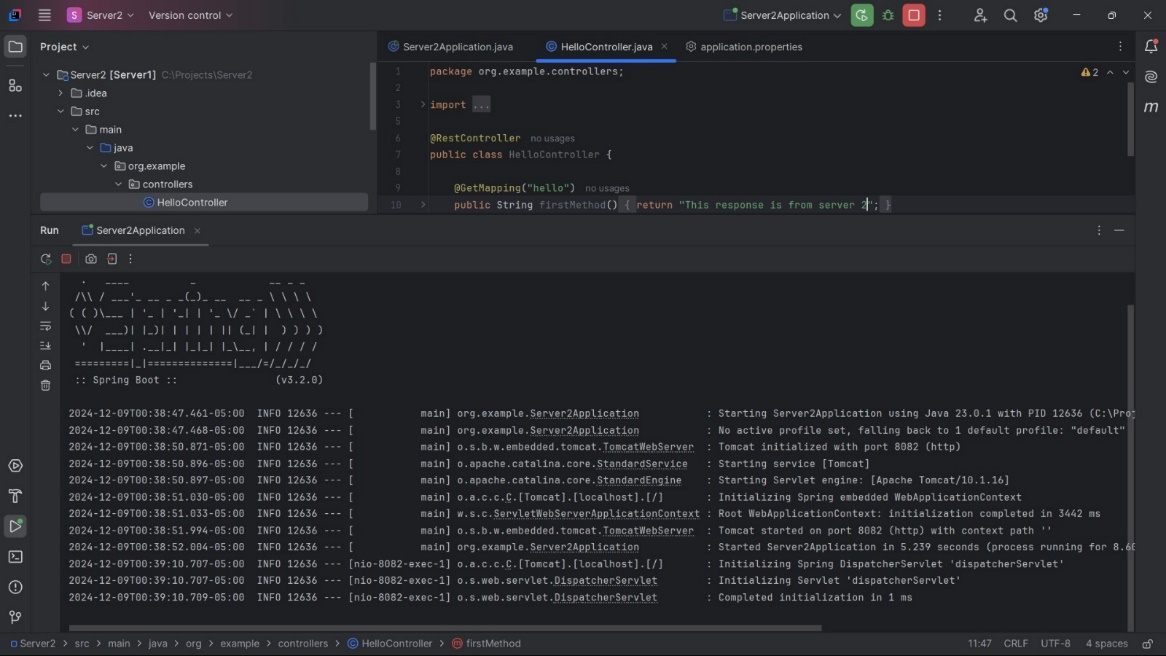
Description automatically generated

**Figure: Load Balancer Initiation**

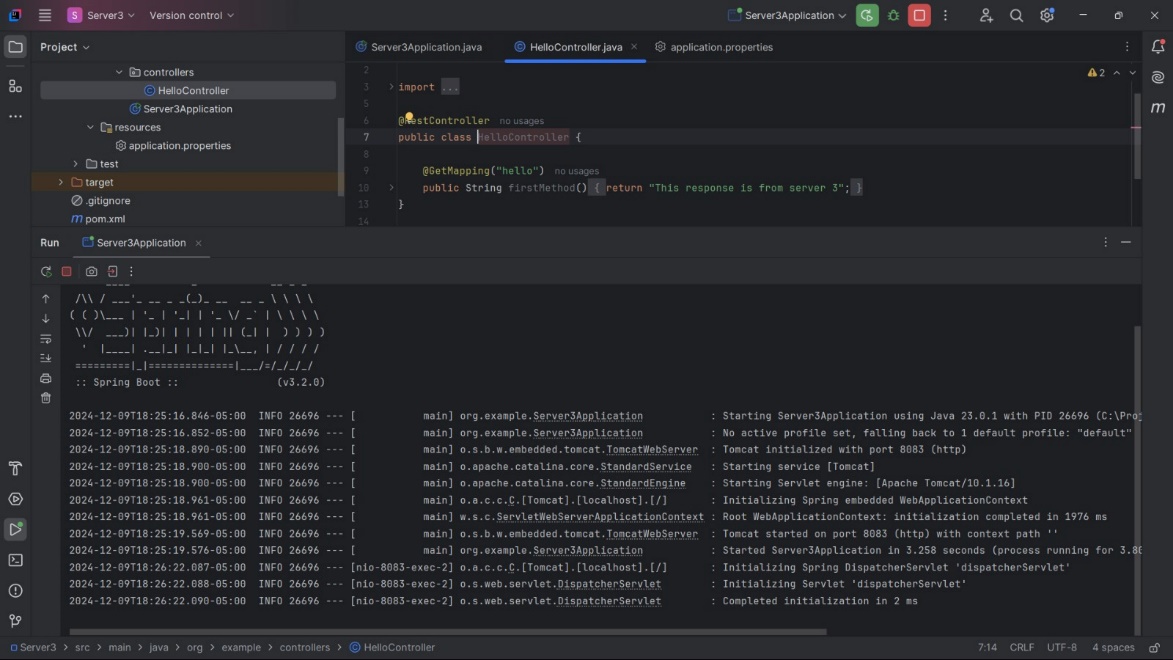
A screenshot of a computer

Description automatically generated

**Figure: Server 1 Initiation**

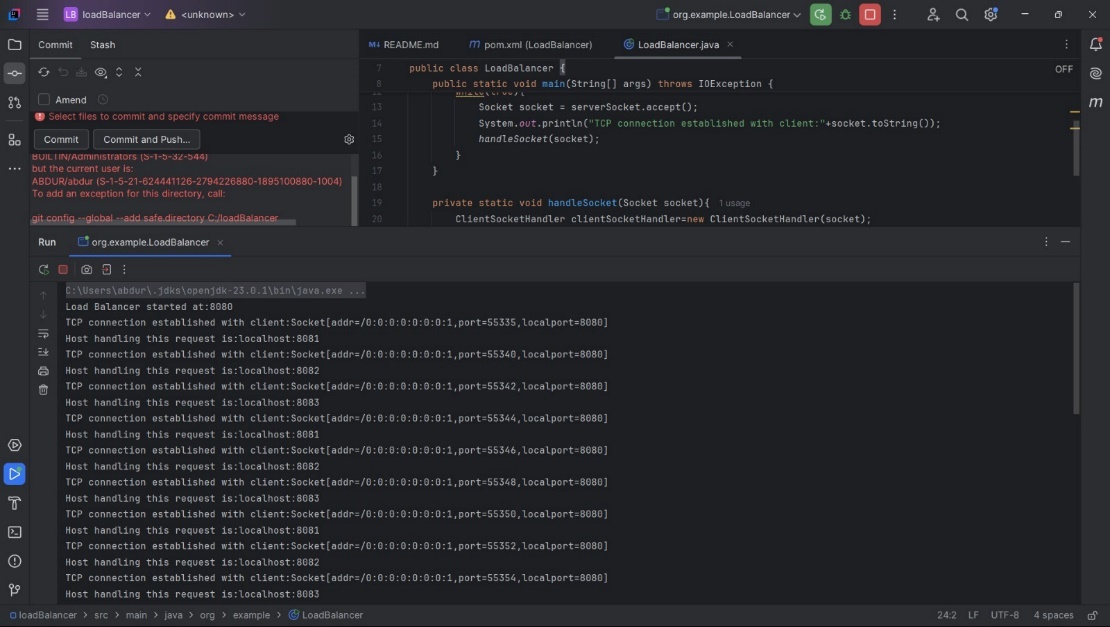


**Figure: Server 2 Initiation**

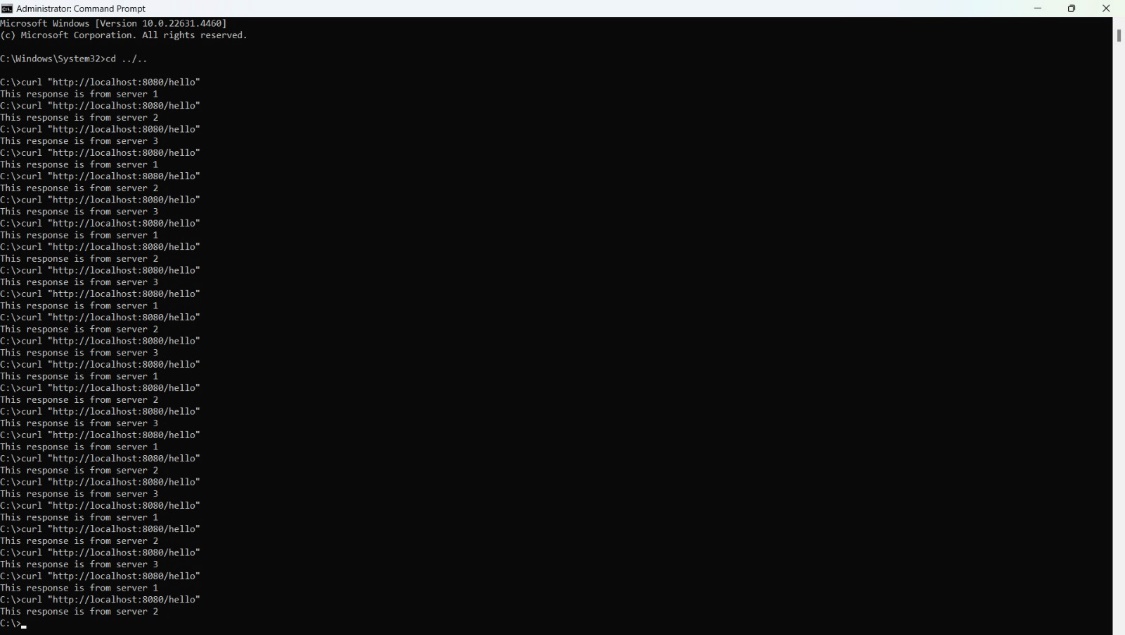


**Figure: Server 3 Initiation**

**Experimental Comparison and Results**

****

**Figure: TCP Connection Established with Client**

****

**Figure: Load Balancer Implementation**

* The load balancer was tested under different traffic conditions by generating artificial workloads. The following metrics were analyzed:

1. **Traffic Distribution Efficiency:**

* Both algorithms distributed the traffic between the servers evenly. In the round-robin algorithm, the load distribution was sequentially consistent; for the least-connections algorithm, it dynamically balanced loads with respect to server utilization.

1. **Scalability:**

* The load balancer scaled well with the introduction of more servers, with any increase in traffic distributed to them smoothly. Performance metrics had linear improvements in handling growing workloads.

**Conclusion**

In the work described, it was possible to realize an effective low-cost flexible software-based load balancer showing great enhancement in scalability and reliability. This may enable varied applications by easy adoption because of a modular design, offering an algorithm and guaranteeing a suitable use of resources in scalable applications. Future enhancements will include measuring response time and fault tolerance. It will also try to compare two different algorithms, i.e. Round-Robin Algorithm and Least-Connections Algorithm.