URL: https://github.com/YeZhang125/cs6650-assignment1.git

Client Architecture Overview

Key Components and Structure

The client implementation comprises the following essential classes:

1. MultiThreadedLiftRideClient.java

- **Purpose:** Acts as the program's main entry point, orchestrating the load testing process.
- **Functions:**
    - Configures parameters such as server URL, number of threads, and requests per thread.
    - Utilizes `ExecutorService` to manage concurrent execution.
    - Monitors thread completion and delegates task execution to `HTTPClientThread`.

2. HTTPClientThread.java

- **Purpose:** Handles HTTP request execution and logs responses from simulated clients.
- **Functions:**
    - Sends HTTP POST requests via `HttpClient`.
    - Logs request details, including timestamp, type, latency, response code, and throughput, to a CSV file.
    - Implements retry logic for failed requests.
    - Calculates and stores response times for further evaluation.

3. EventProducer.java

- **Purpose:** Generates randomized event data for HTTP POST requests, simulating skier lift events.
- **Functions:**
    - Produces synthetic data representing ski resort activities with randomized attributes (e.g., resort ID, season ID, skier ID).
    - Ensures varied request simulation to enhance realism in load testing.

4. LatencyComputationForClient2.java

- **Purpose:** Processes collected latency data post-test to derive performance metrics.
- **Functions:**
    - Reads logged data from CSV files.

    - Computes statistics such as mean, median, minimum, maximum, and p99 latencies.
    - Determines system throughput.

Project Structure and Class Interactions

Modules:

- **client1:** Houses the primary classes (`MultiThreadedLiftRideClient`, `HTTPClientThread`, `EventProducer`, `LatencyComputationForClient2`).
- **server:** Contains the servlet responsible for request handling.

Class Interactions:

- `MultiThreadedLiftRideClient` initiates and supervises multiple `HTTPClientThread` instances.
- Each `HTTPClientThread` collaborates with `EventProducer` to generate request data.
- After execution, `LatencyComputationForClient2` evaluates and reports system performance.

Throughput Performance Estimation Using Little's Law

Throughput Calculation:

Throughput is determined by evaluating the total test duration and the number of processed requests:

Given Total Requests = 200,000, Throughput = Total Number of Requests / Total Response Time = 200,000 / 27,865 ms = 7177 requests / sec

This estimation helps assess system capacity and efficiency under varying load conditions.

Screenshots

Client 1

```
/Users/yezhang/Library/Java/JavaVirtualMachines/corretto-17.0.14/Contents/Home/bi
======= Client 1 Output =======
Number of Threads: 200
Successful requests: 200000
Failed requests: 0
Total requests sent: 200000
Total response time: 30161 ms
Throughput: 6631.07987135705 requests per second
```

Client 2

```
======= Client 1 Output =======
Number of Threads: 200
Successful requests: 200000
Failed requests: 0
Total requests sent: 200000
Total response time: 30161 ms
Throughput: 6631.07987135705 requests per second

======= Client 2 Output =======
Mean Response Time: 28.4722 ms
Median Response Time: 27 ms
Min Response Time: 10 ms
Max Response Time: 264 ms
99th Percentile Response Time: 73 ms
Throughput per thread: 35.1219786317882 requests/sec
Estimated throughput for 200 thread: 7024.39572635764 requests/sec

Process finished with exit code 0
```

EC2 Instance

http://44.243.12.46:8080/assignment1_war/skiers/10/seasons/2025/days/1/skiers/4

Save

POST http://44.243.12.46:8080/assignment1_war/skiers/10/seasons/2025/days/1/skiers/4 Send

Params Authorization Headers (8) Body Scripts Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
  "time": 217,
  "liftID": 21
}
```

Body Cookies Headers (5) Test Results 201 Created 46 ms 222 B

JSON Preview Visualize

```
{
    "message": "Skier processed successfully"
}
```