The Github link: https://github.com/Crystal0GMY/MultithreadPost

Client Design Overview

Major Classes and Packages

The client design is composed of the following major classes:

1. MultithreadClient.java

- **Role**: This class serves as the entry point to the program and is responsible for initiating the load testing process. It handles the configuration, such as the server URL, number of threads, and requests per thread.
- **Responsibilities**:
  - Sets up and manages the concurrent execution of requests via an `ExecutorService` (multithreading).
  - Tracks the completion of all threads.
  - Passes control to the `HTTPClientThread` class for each simulated client.

2. HTTPClientThread.java

- **Role**: This class is responsible for managing the HTTP requests sent by a simulated client. It handles the logic for sending requests, logging responses, and retrying failed requests.
- **Responsibilities**:
  - Sends HTTP POST requests to the server using the `HttpClient` class.
  - Records the start and end time of each request to compute latency.
  - Logs the data (timestamp, request type, latency, response code, throughput) into a CSV file.
  - Handles retries if a request fails, ensuring that the simulation can continue in case of temporary failures.
  - Computes response time and stores it for later analysis.

3. EventGeneratorThread.java

- **Role**: This class generates random event data that is sent in the HTTP POST requests. It simulates a series of skier lift events with random attributes (e.g., resort ID, season ID, skier ID).
- **Responsibilities**:
  - Generates synthetic data that mimics real-world events, such as ski resort information and user activity.

- Ensures randomness in generated data to simulate various user requests during the load testing.

4. LatencyAnalyzer.java

- **Role**: This class is responsible for analyzing the collected latency data after the test has completed. It computes important performance metrics such as the mean, median, min, max, p99 latency, and throughput.
- **Responsibilities**:
  - Reads the logged data from the CSV file.
  - Computes the **mean** and **median** response times.
  - Identifies the **p99** latency (the 99th percentile) for assessing high-latency outliers.
  - Calculates **throughput**.

Packages and Relationships

The project is structured into the following main packages:

- **client1**: Contains all the key classes for client (`MultithreadClient`, `HTTPClientThread`, `EventGeneratorThread`, and `LatencyAnalyzer`).
- **Server**: Contains the servlet.

Relationships:

- The `MultithreadClient` class manages the setup and execution of `HTTPClientThread` instances.
- Each `HTTPClientThread` interacts with the `EventGeneratorThread` to generate the data for the HTTP POST requests.
- Once all threads are complete, `LatencyAnalyzer` reads the collected data and generates a report with key performance metrics.

Throughput Prediction Using Little's Law

Throughput Calculation

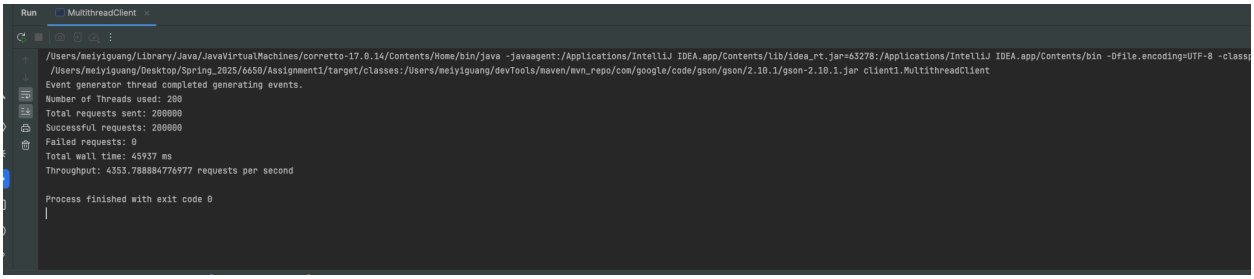
The throughput can be derived by monitoring the total time of the test and the total number of requests sent. Given that the client logs the start and end times of requests, as well as their latencies, we can compute throughput as:

Throughput = Total Number of Requests / Total Time Taken = 200,000 / (50,612 / 1,000) = 3,951 requests per second.

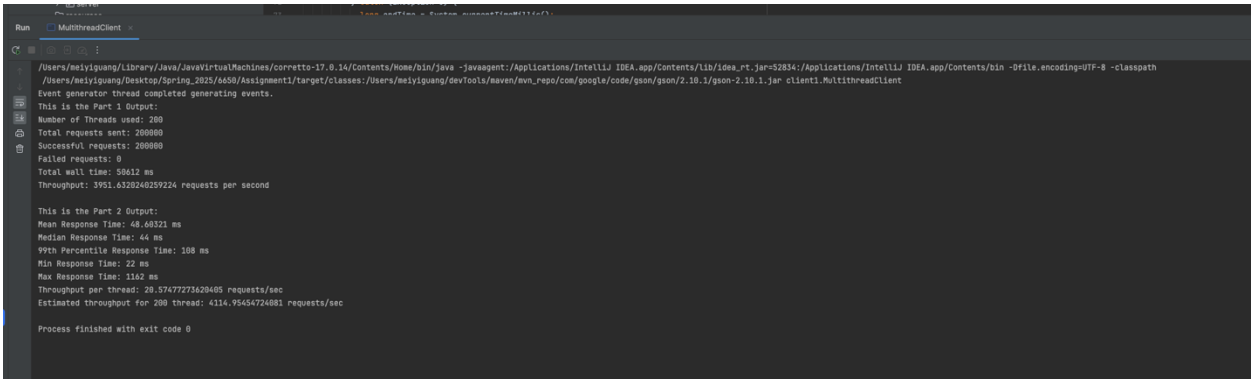
This helps in predicting how many requests per second the system can handle under a given load, providing valuable insights into its performance.

Screenshots

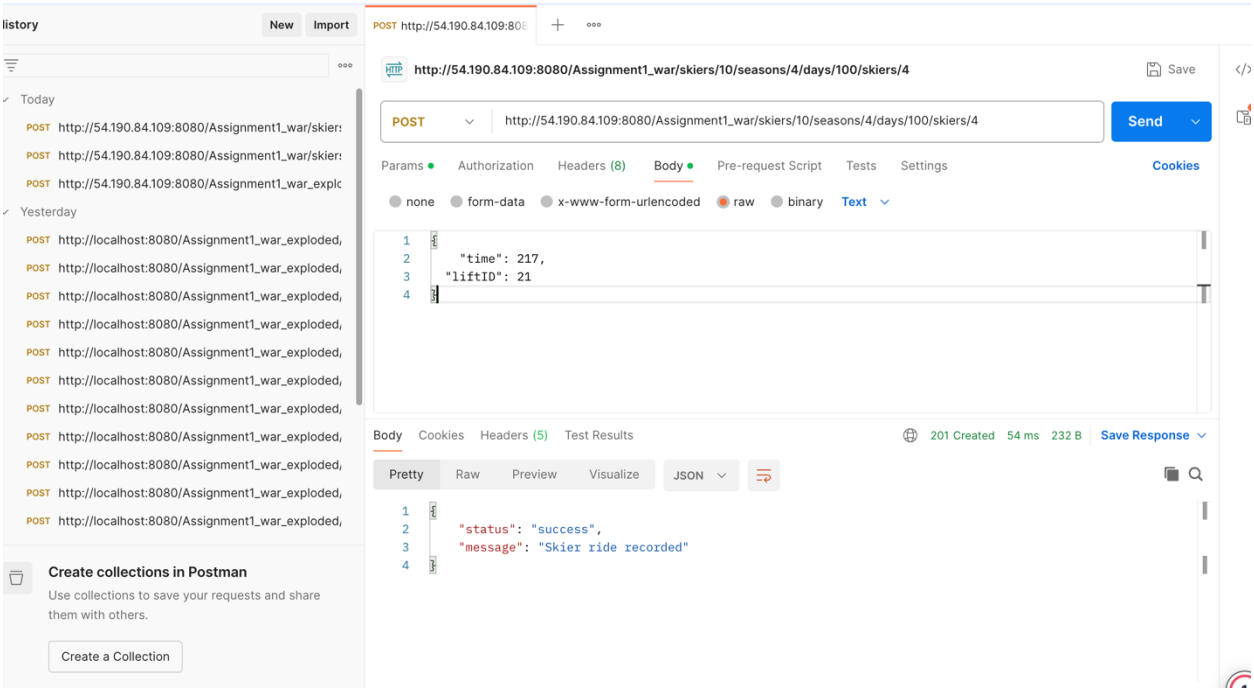
Client (Part 1) - screenshot of the output window with the wall time and throughput.



Client (Part 2) – screenshot of the output window for each run with the specified performance statistics listed at the end.



Postman testing page showing the URL (the IP address may have changed because the learner lab terminated the EC2 instance I used while I designed my HTTPClient)



EC2 instance page