Assignment 1 Report

Github Repo:

https://github.com/CloudyZ524/6650-Distributed-System/tree/main/Assignment/Assignment1

Client Design

1. MultiThreadLiftRideClient .java (Entry Point):

- Initializes the event generation and starts the multi-threaded posting process.
- Sets up the ThreadPoolExecutor for managing multiple threads.
- Waits for all tasks to complete and prints out the results

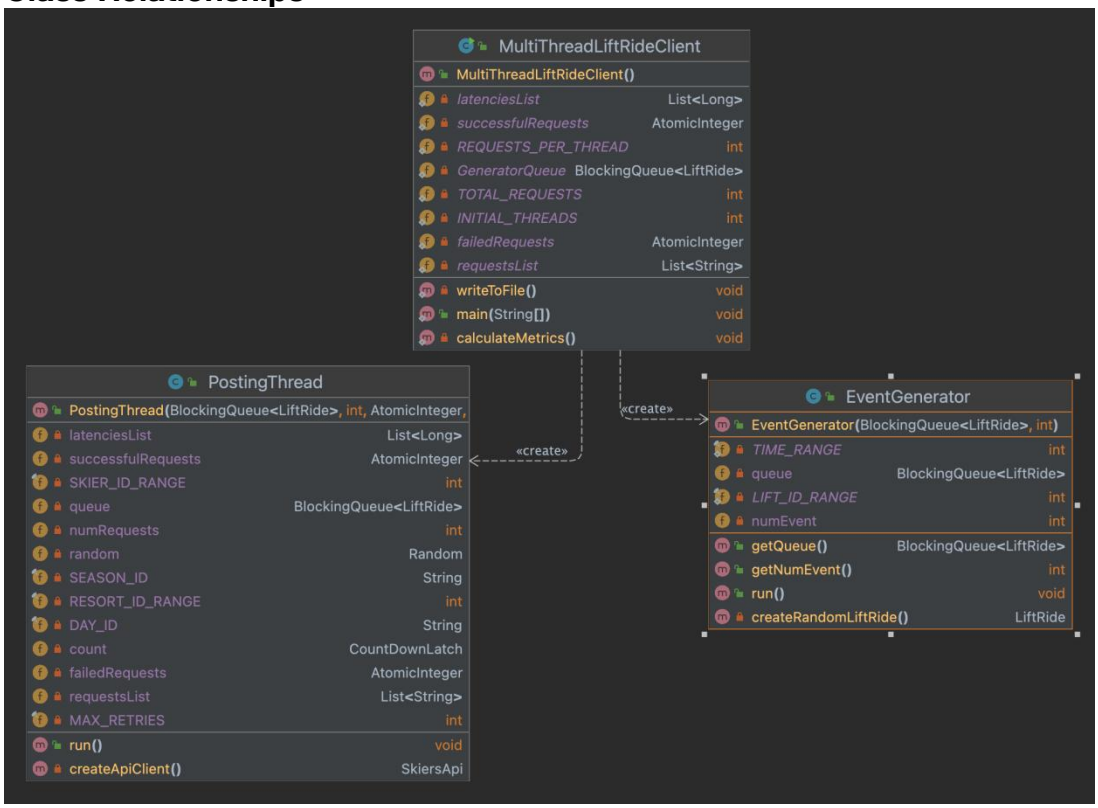
2. EventGenerator.java:

- Implements Runnable for concurrent execution.
- Generates a specified number of LiftRide events and adds them to a BlockingQueue.
- Uses random values to populate the LiftRide objects, simulating different lift rides.

3. PostingThread.java:

- Implements Runnable for concurrent execution.
- Takes LiftRide events from the BlockingQueue and posts them to a server using the SkiersApi.
- Handles retries for failed requests and collects latency metrics.

Class Relationships



Threading and Synchronization

Multi-threaded Approach:
The client utilizes Java's threading capabilities for concurrent HTTP requests through a ThreadPoolExecutor. This executor manages threads that execute tasks within the PostingThread class, acting as individual SingleClients.

CountDownLatch:
A CountDownLatch is employed to synchronize the start and completion of thread tasks, ensuring that the main flow waits for all threads to finish before proceeding. This mechanism ensures accurate timing and performance measurement by preventing premature termination or aggregation of results.

AtomicInteger
The use of AtomicInteger for tracking successful and failed requests across threads provides a thread-safe mechanism to update shared counters without race conditions, essential for accurate metrics collection in a concurrent environment.

BlockingQueue
The application introduces a ConcurrentLinkedQueue to store LiftRide objects, which encapsulate detailed information about each HTTP request. This queue allows for thread-safe additions, enabling detailed performance analysis post-execution.

Little's Law throughput predictions

```
Successful requests: 10000
Failed requests: 0
Total time: 14268 ms
Throughput: 700.8690776562938 requests/second
```

Baseline response time: 14.268 sec / 10000 = 0.0014268 sec/request

According to Little's Law, N = Throughput * Response Time.
If size of thread pool is 15, then N = 15.
Estimated throughput = N / Response Time
= 15 / 0.0014268
= 10531.0362 request/sec

The estimation is similar to the actual result shown below.

Client - Part 1

Size of thread pool : 15

```
Successful requests: 200000
Failed requests: 0
Total time: 19796 ms
Throughput: 10103.051121438675 requests/second
```

Client - Part 2

Size of thread pool : 15

```
Successful requests: 200000
Failed requests: 0
Total time: 17434 ms
Throughput: 11471.836641046231 requests/second

Mean response time: 1 ms
Median response time: 1 ms
p99 response time: 2 ms
Min response time: 0 ms
Max response time: 157 ms
```

Plot of Throughput