1. URL for git repository

https://github.com/SerendipityJerry/CS6650

2. Description of the client design and major classes

2.1 MultiThreadClient

The class performs multi-threaded client requests to a server. The program creates 80 consumer threads and 1 producer thread, each consumer thread processes 6250 requests, and the producer thread produces a total of 500000 requests. The purpose of the program is to measure the performance of the server by recording the response time of each request and generating statistics based on the response times that are recorded in an OutputRecord object and stored in a synchronized list.

The program uses the CountDownLatch to wait for all consumer threads to finish before printing out the statistics. The statistics include the mean response time, median response time, 99th percentile response time, minimum and maximum response time, and throughput.

Finally, the program writes records of the requests to a CSV file in the specified directory and generates a excel spreadsheet to show the average throughput of requests for the period of the test.

2.2 SwipeEventProducer

The class is used to generate random "SwipeEvent" objects and put them into a thread-safe LinkedBlockingQueue. Each SwipeEvent contains a random "siwpe behavior" (either "left" or "right"), a random swiper ID between 1 and 5000, a random swipee ID between 1 and 1000000, and a random comment with a length between 1 and 256 characters. The generated SwipeEvent is then added to the BlockingQueue.

The implementation uses ThreadLocalRandom to generate random numbers for the ID of the swiper and swipee and the length of the comment. The RandomStringUtils from the Apache Commons Lang library is used to generate the random comment string.

2.3 SwipeEventConsumer

The class is a multi-threaded consumer that is designed to consume SwipeEvent objects from the LinkedBlockingQueue and send them to a REST API. It does this by utilizing the SwipeApi class from the io.swagger.client package, which provides methods to interact with the REST API.

The consumer runs as a separate thread and retrieves SwipeEvent objects from the queue. The postWithEvent method is called with the retrieved SwipeEvent and

SwipeApi objects as arguments. This method creates a SwipeDetails object, sets its properties based on the SwipeEvent object, and sends it to the API using the swipeWithHttpInfo method. If the API returns a HTTP status code of 201 (created), the method returns true. If an exception occurs or the API returns a different status code, the method retries up to MAX_RETRY (5) times. If the method is not able to successfully send the request to the API after five retries, it returns false.

At the end of its execution, the consumer adds its successful and failed request counts to the shared successReq and failReq atomic integers, respectively, and decrements the latch count. The records list is updated with a new OutputRecord object containing the start time, response time, and HTTP status code of the request.

3. Client Part1 Screen Shot:

```
************************************************************
Processing Ends
************************************************************
Number of successful requests :10000
Number of failed requests :0
Total wall time: 316091
Throughput: 31 requests/second
```

Number of threads: 1
Number of requests: 10000

The average response time of a request is 31 ms. Assume we have 80 threads in multithreaded client, and the predicted throughput from Little's Law is 80/0.0316 = 2531 requests/second

```
************************************************************
Processing Ends
************************************************************
Number of successful requests :500000
Number of failed requests :0
Total wall time: 225965
Throughput: 2222 requests/second
```

Number of threads: 80
Number of requests: 500000

The actual throughput is close to Little's Law prediction.

4. Client Part2 Screen Shot:

```
Number of successful requests :10000
Number of failed requests :0
Total wall time: 316763
Mean response time (in milliseconds):31.5824
Median response time (in milliseconds):30.0
Throughput:31 requests/second
P99 (99th percentile) response time (in milliseconds):56.0
Min response time (in milliseconds):21.0
Max response time (in milliseconds):193.0
```

Number of threads: 1
Number of requests: 10000

```
Number of successful requests :500000
Number of failed requests :0
Total wall time: 223339
Mean response time (in milliseconds):44.297746
Median response time (in milliseconds):42.0
Throughput:2242 requests/second
P99 (99th percentile) response time (in milliseconds):91.0
Min response time (in milliseconds):18.0
Max response time (in milliseconds):376.0
```

Number of threads: 80
Number of requests: 500000

5. Plot Performance