Assignment 3 Report: Twinder Adding Persistence and Reads

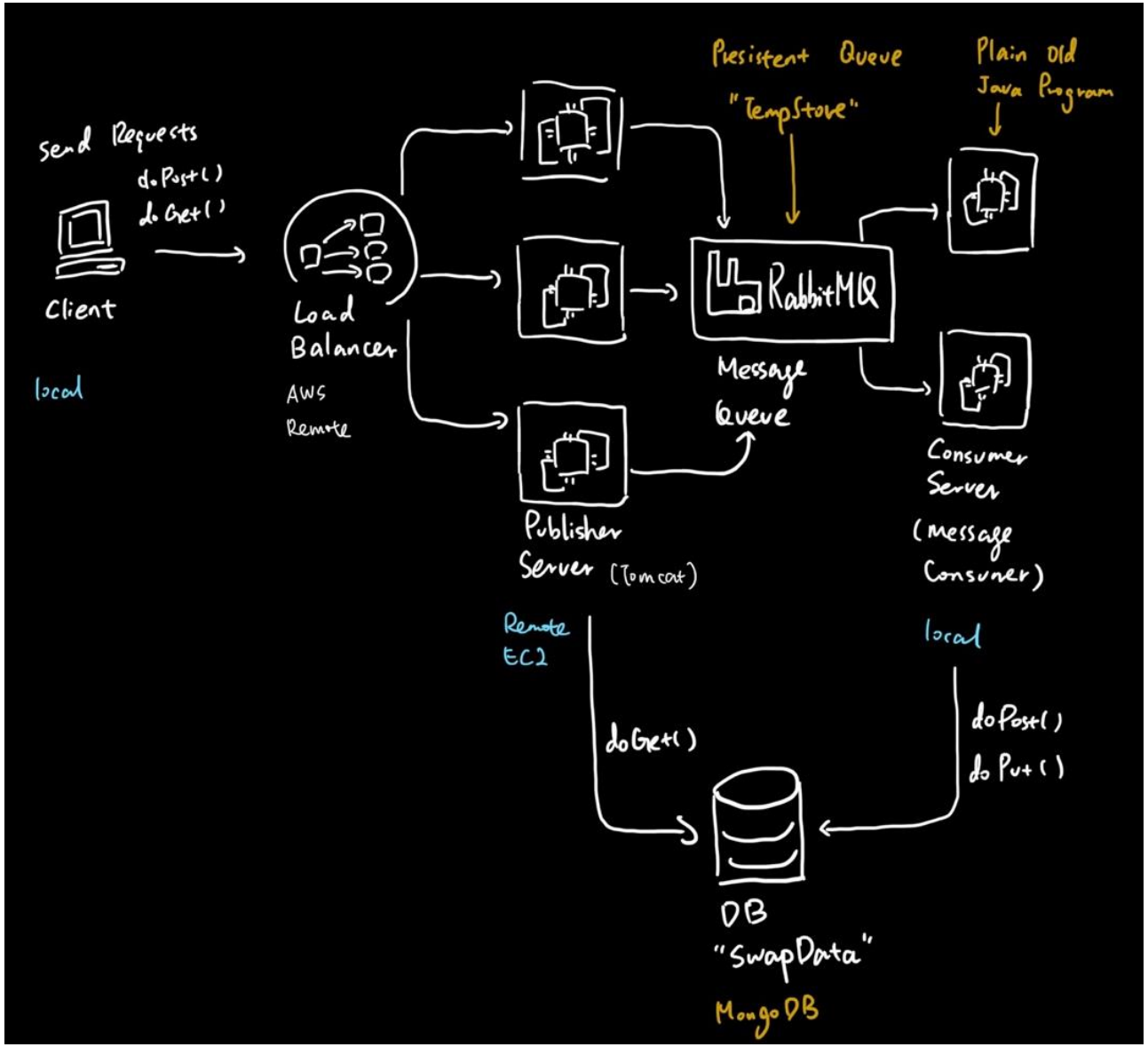
CS6650 Distributed System

Zidi Xia

The URL of git repo

https://github.com/sharonzidi/cs6650_twinder/tree/main/Assignment3

Twinder Architecture Design



Client (local): The client generates Swipe events and send to server through load balancer.

Load balancer: The load balancer used to distribute requests among multiple web servers and ensure that all users receive timely and efficient responses.

Publisher Server (Tomcat): The publisher server receiving request from LB and send messages to message queue. It implements the GET requests and retrieves results from the database directly.

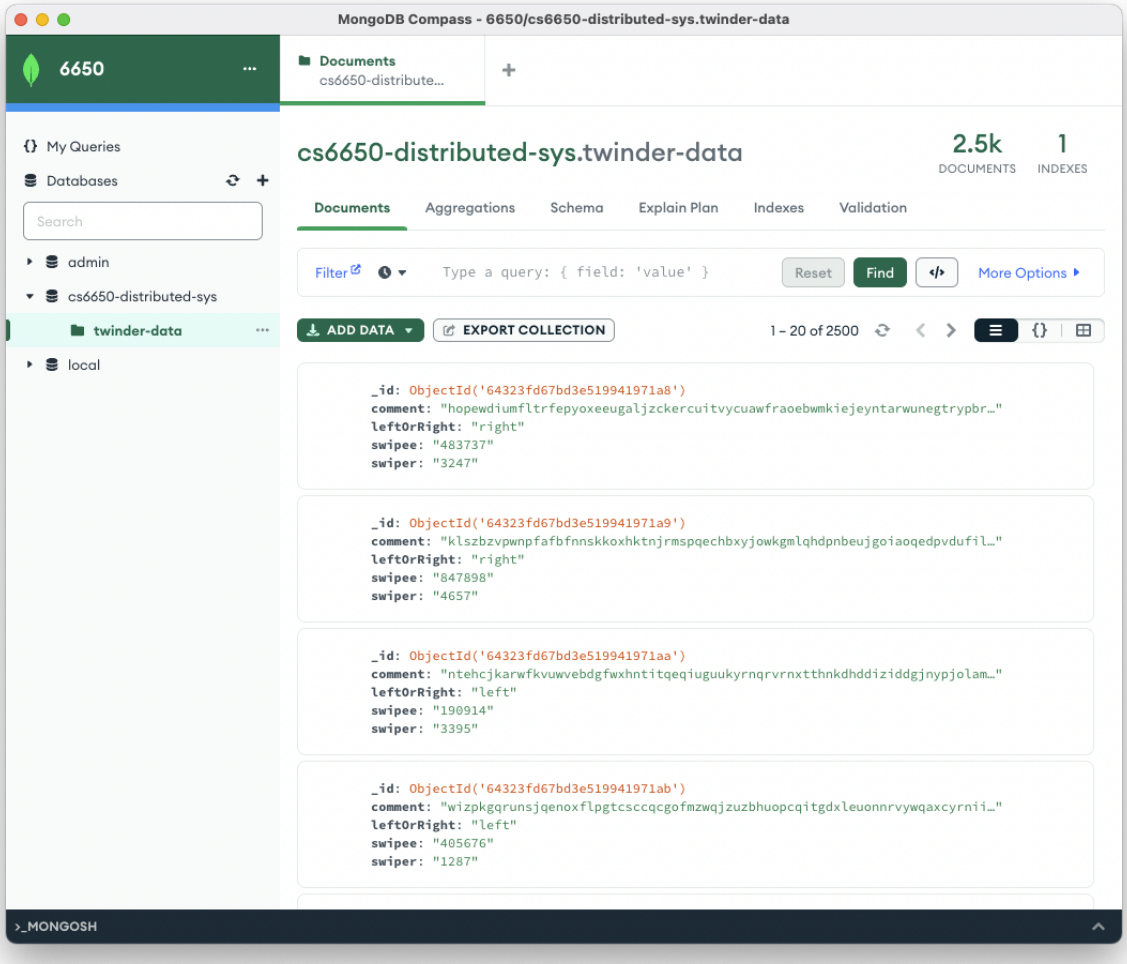
Message Queue (RabbitMQ): Using RMQ to manage message queues since it provides a flexible and scalable way to implement messaging between distributed systems.

Consumer Server (Plain Old Java Program): The consumer reads new Swipe events from RMQ and updates the database that stores information about users and swipe events.

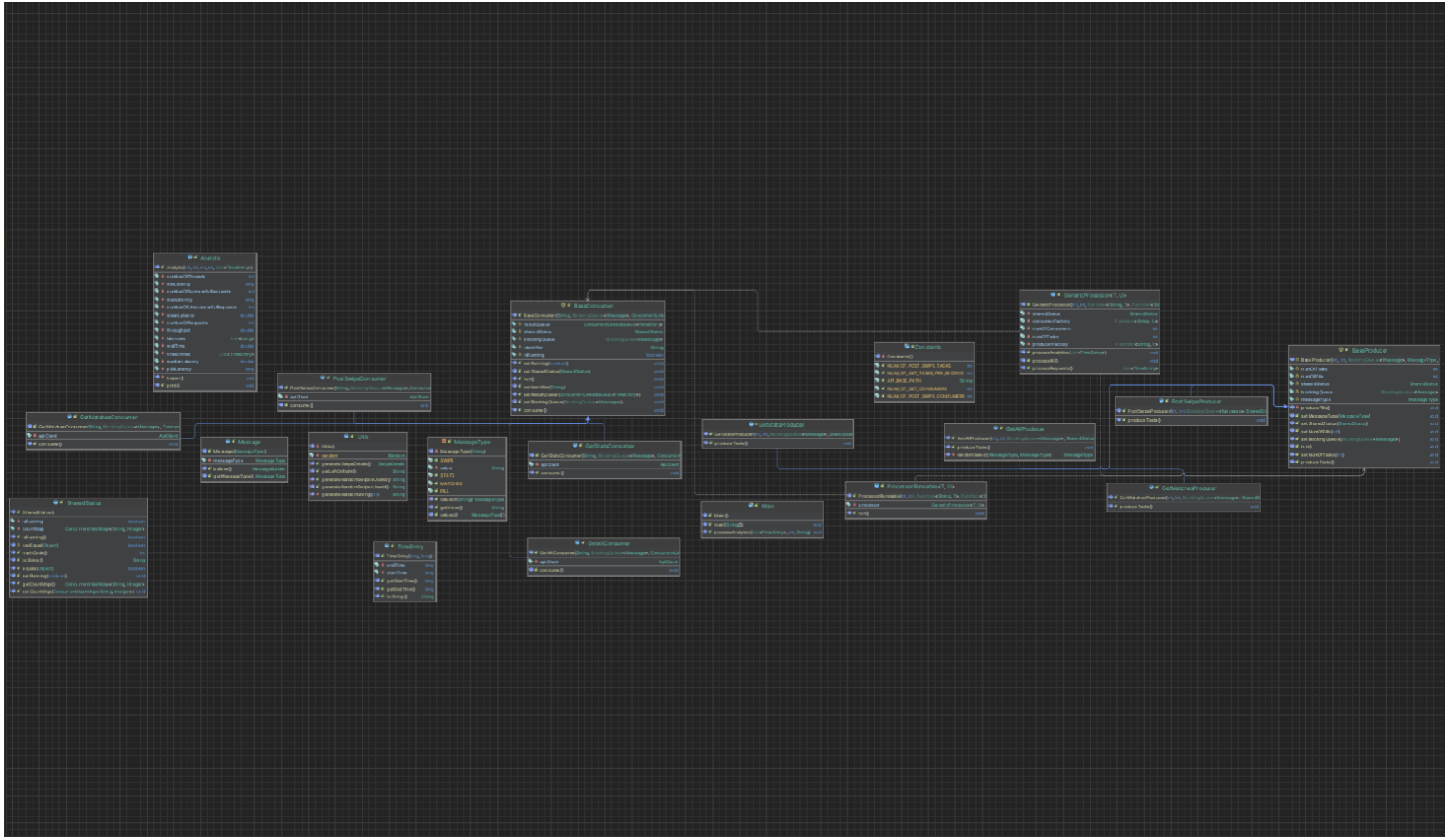
Database (MongoDB Atlas): The project uses MongoDB Atlas to stores information about users and swipe events. MongoDB's flexibility, scalability, and performance make it a powerful choice for this project, and works great with large and complex data requirements.

Database Design

This project uses MongoDB to store data. MongoDB contains cs6650-disxtributed-sys database. In the cs6650-disxtributed-sys database, there is a collection called twinder-data which contains a group of documents. A document is a set of key-value pairs that represent a single instance of swipe data. MongoDB documents are stored in BSON (Binary JSON) format, which is a binary representation of JSON documents.



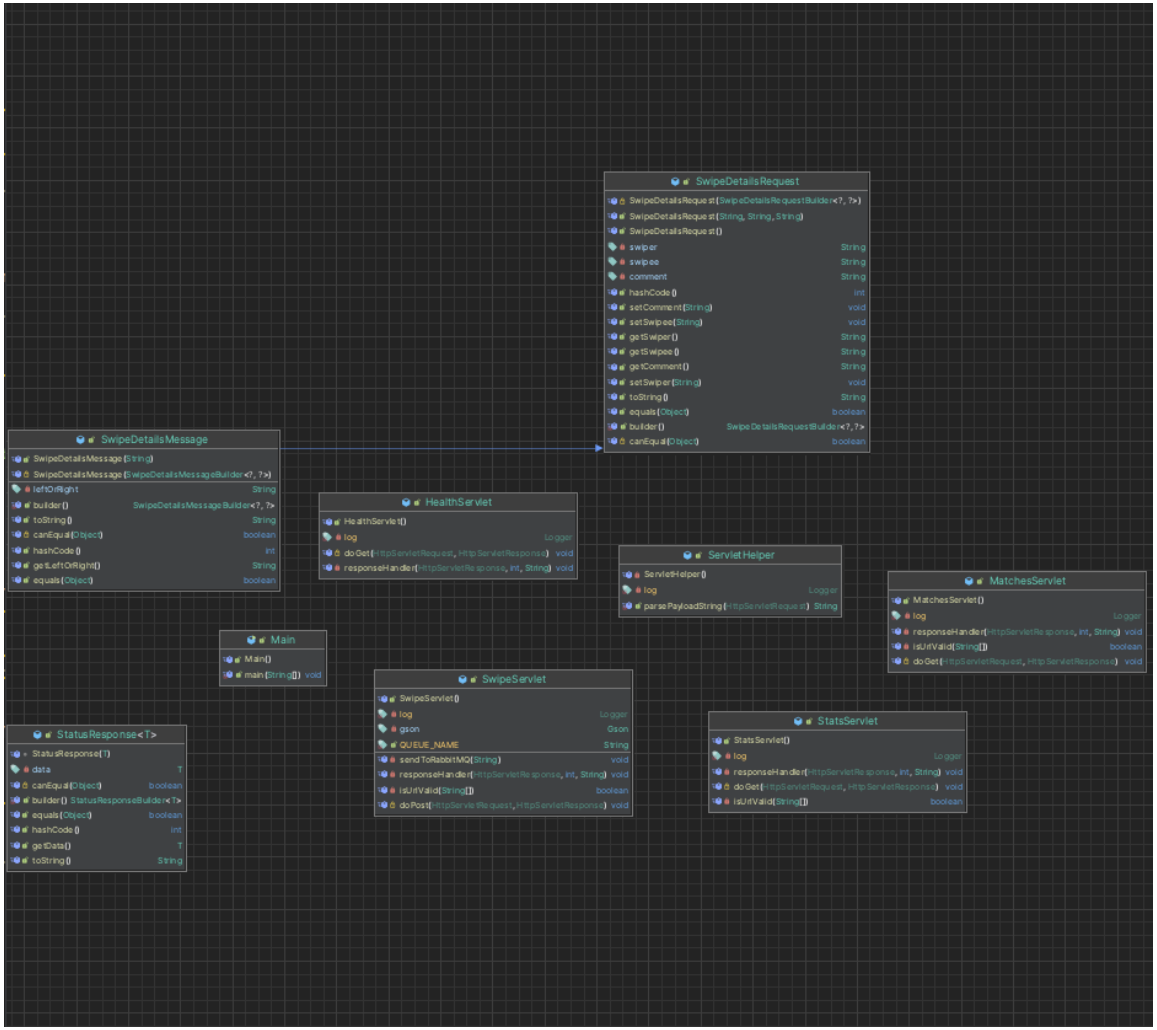
Twinder Client UML



Twinder Publisher Server UML

Send Swipe data as payload to remote queue and return success to the client.

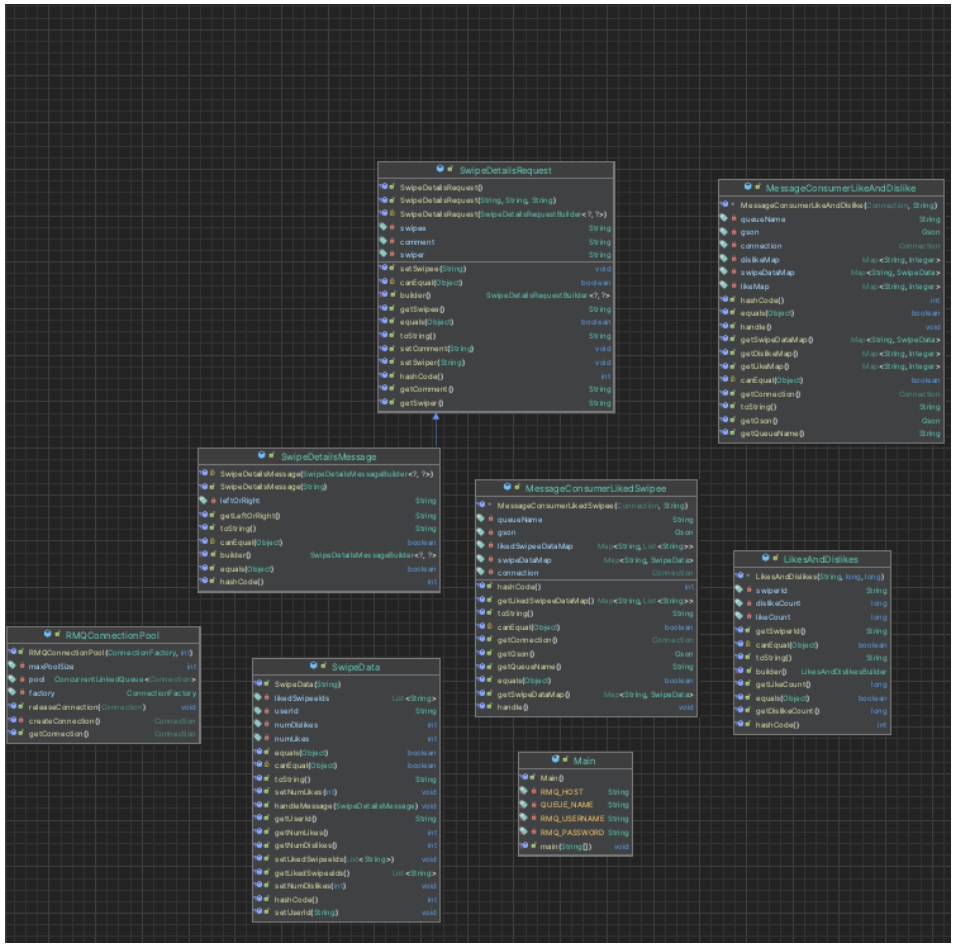
Implements the GET requests and retrieves results from the *SwipeData* database directly.



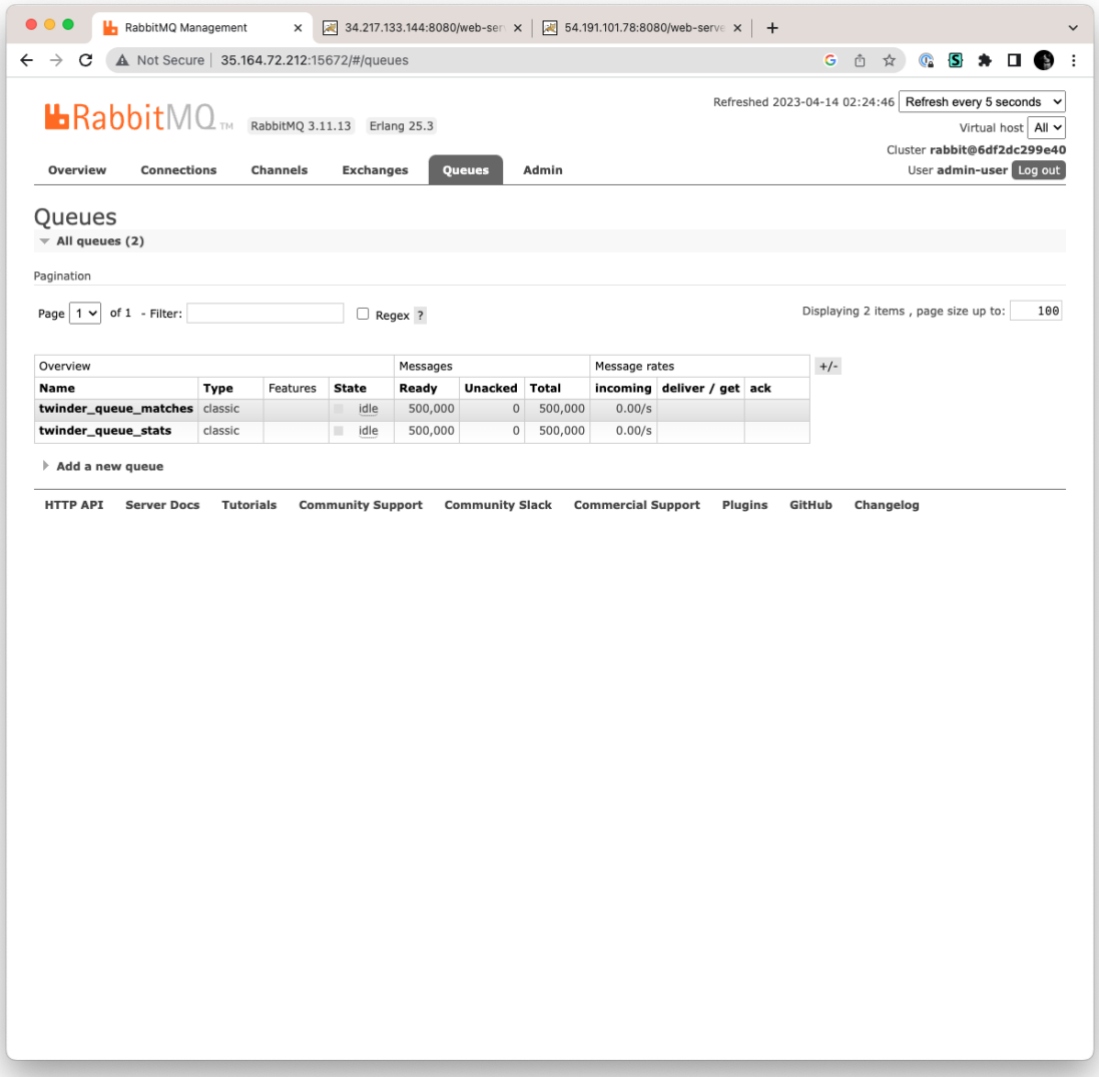
Twinder Consumer Client UML

Pull messages off the queue and consume each message.

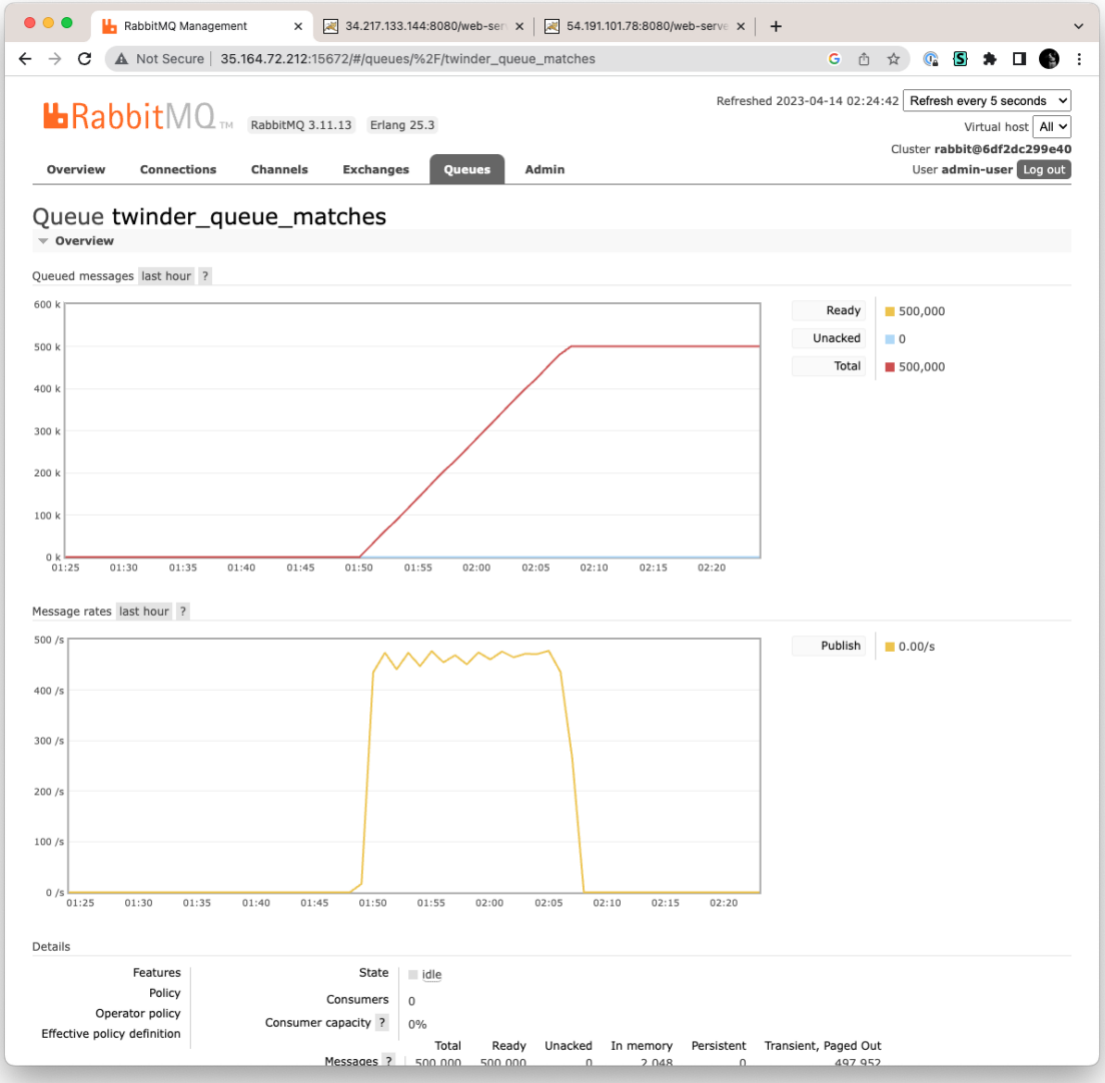
Reads new Swipe events from *TempStore* and updates a database that stores information about users and swipe events.



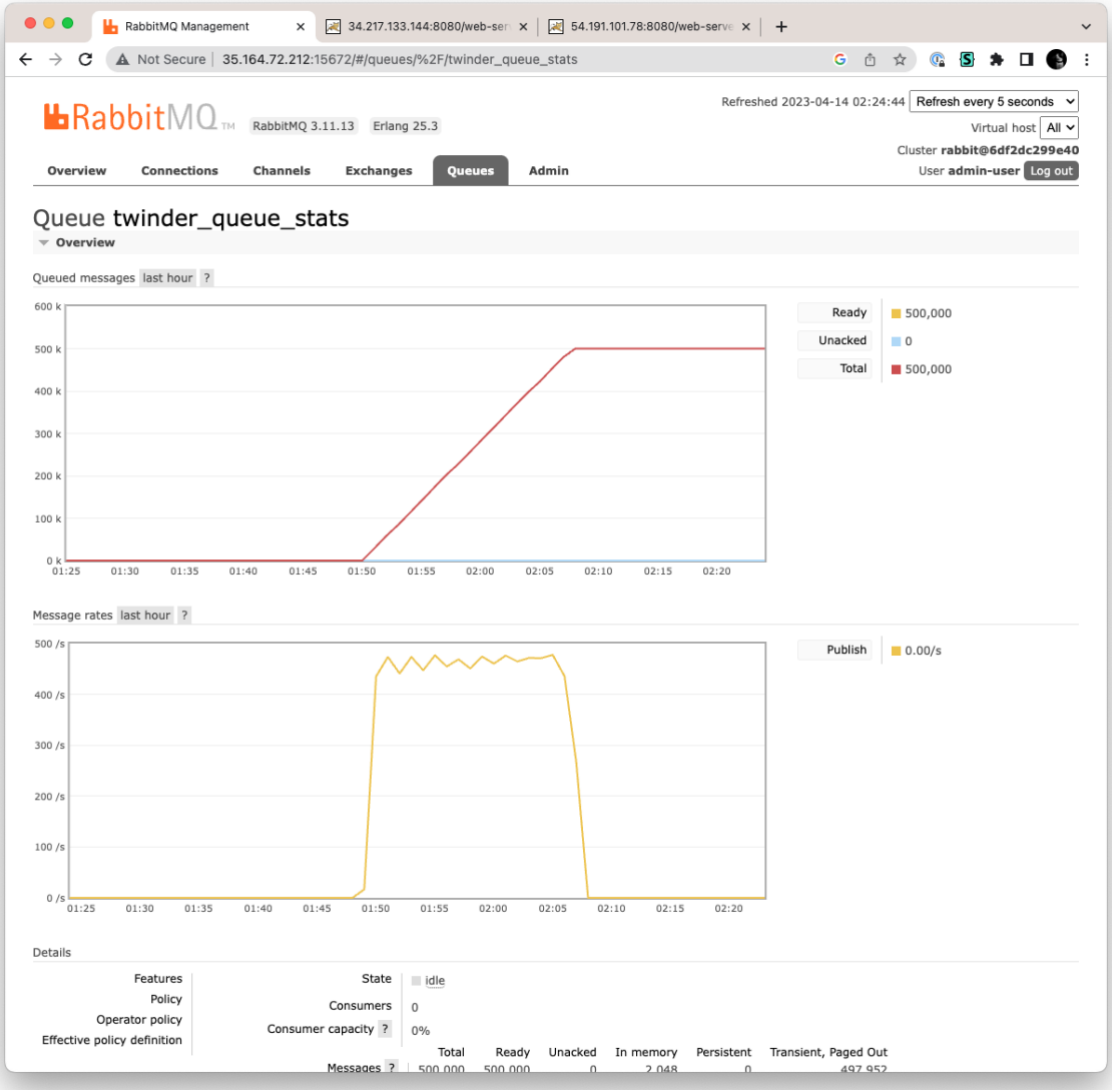
A3 RMQ Screenshots with Queue Size



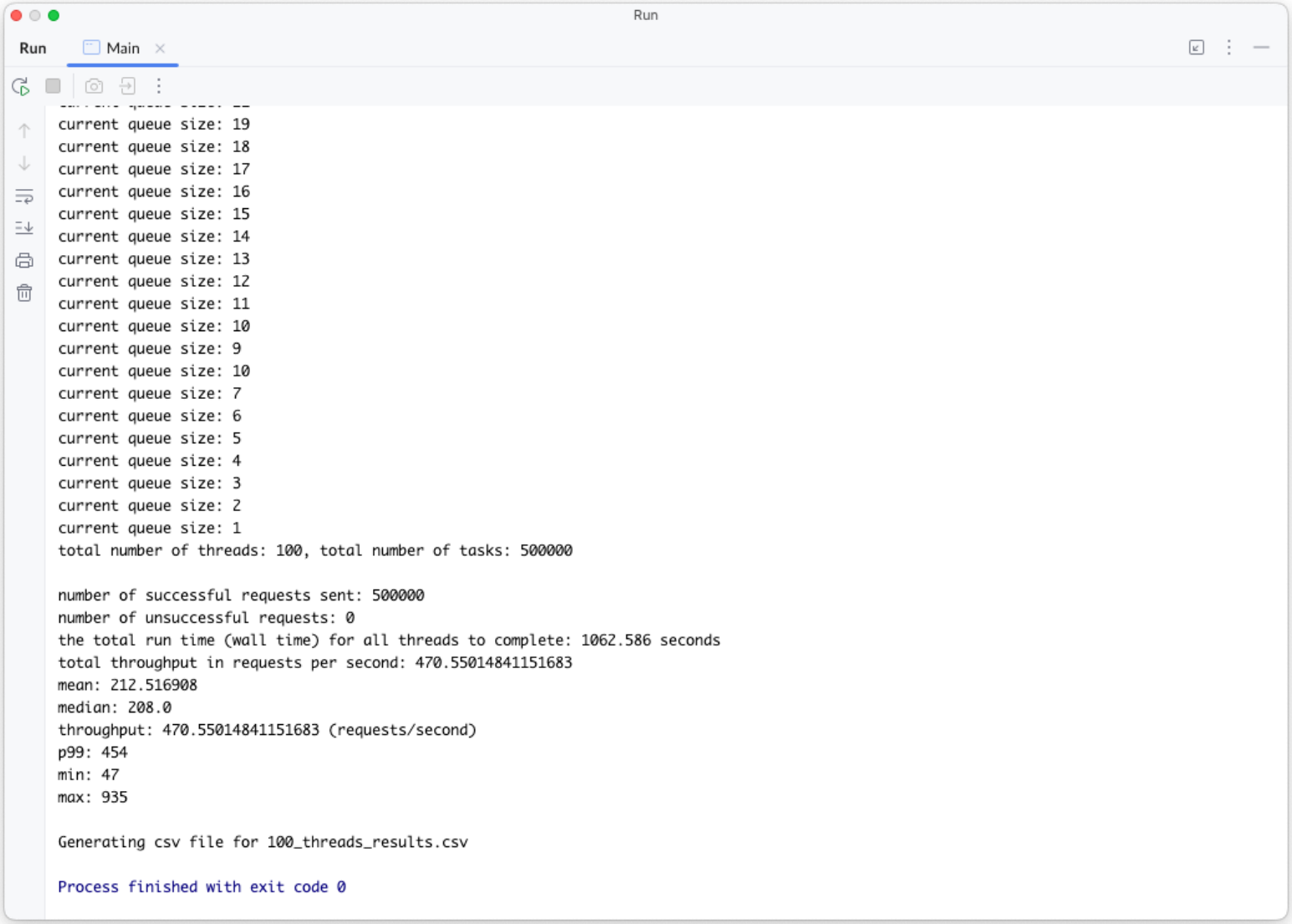
A3 RMQ Screenshots with Flat Line Profile
Twinder_queue_matches queue



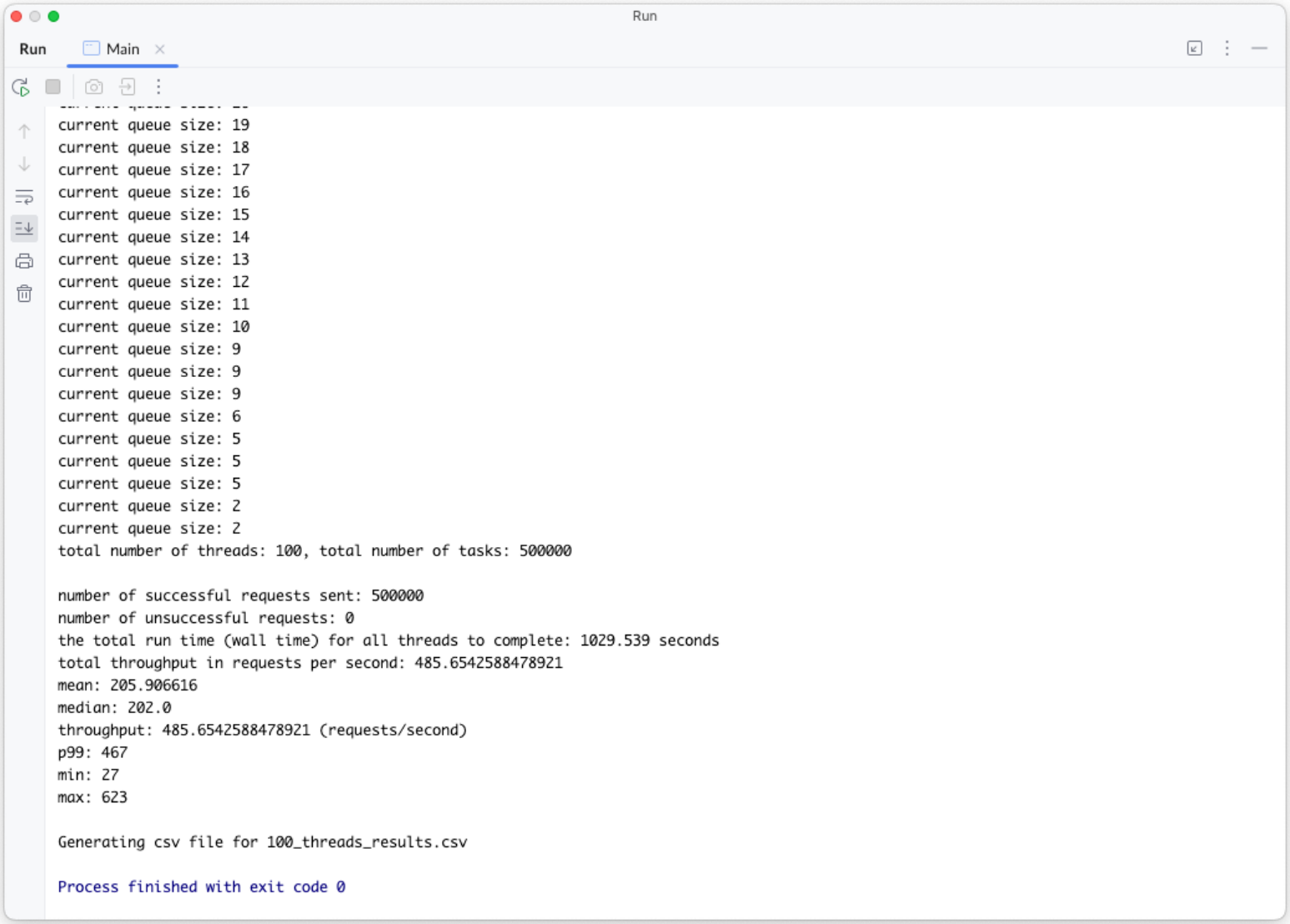
Twinder_queue_stats queue



A3 Client test run screenshots showing results for GET and POST requests and best performance:



A2 performance screenshots:



A2 & A3 performance Comparisons

A2

Total number of threads: 100
Total number of tasks: 500k
Wall time for all threads to complete: **1029.539 seconds**
Total throughput in requests per second: **485.654**
Mean: 205.906
Median: 202
P99: 467
Min: 27
Max: 623

A3

Total number of threads: 100
Total number of tasks: 500k
Wall time for all threads to complete: **1062.586 seconds**
Total throughput in requests per second: **470.550**
Mean: 212.516
Median: 208.0
P99: 454
Min: 47
Max: 935

As we can see from the above screenshots and data, the throughput and latencies at the client in Assignment 3 is within 10% of Assignment 2. The GET request mean latency no longer than the mean POST request latency.