

# **Assignment 3 Report: Twinder Adding Persistence and Reads**

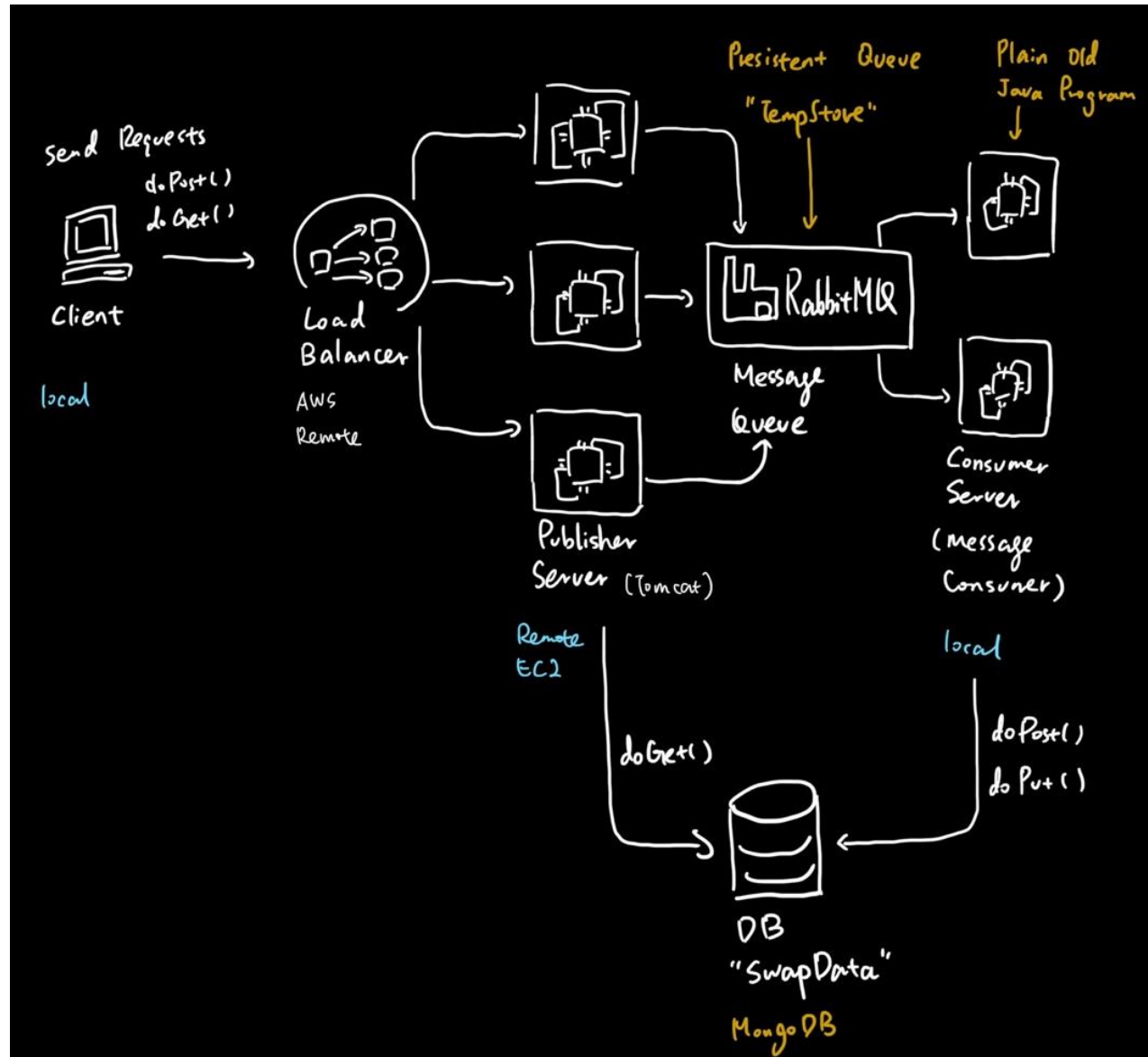
**CS6650 Distributed System**

**Zidi Xia**

**The URL of git repo**

[https://github.com/sharonzidi/cs6650\\_twinder/tree/main/Assignment3](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.

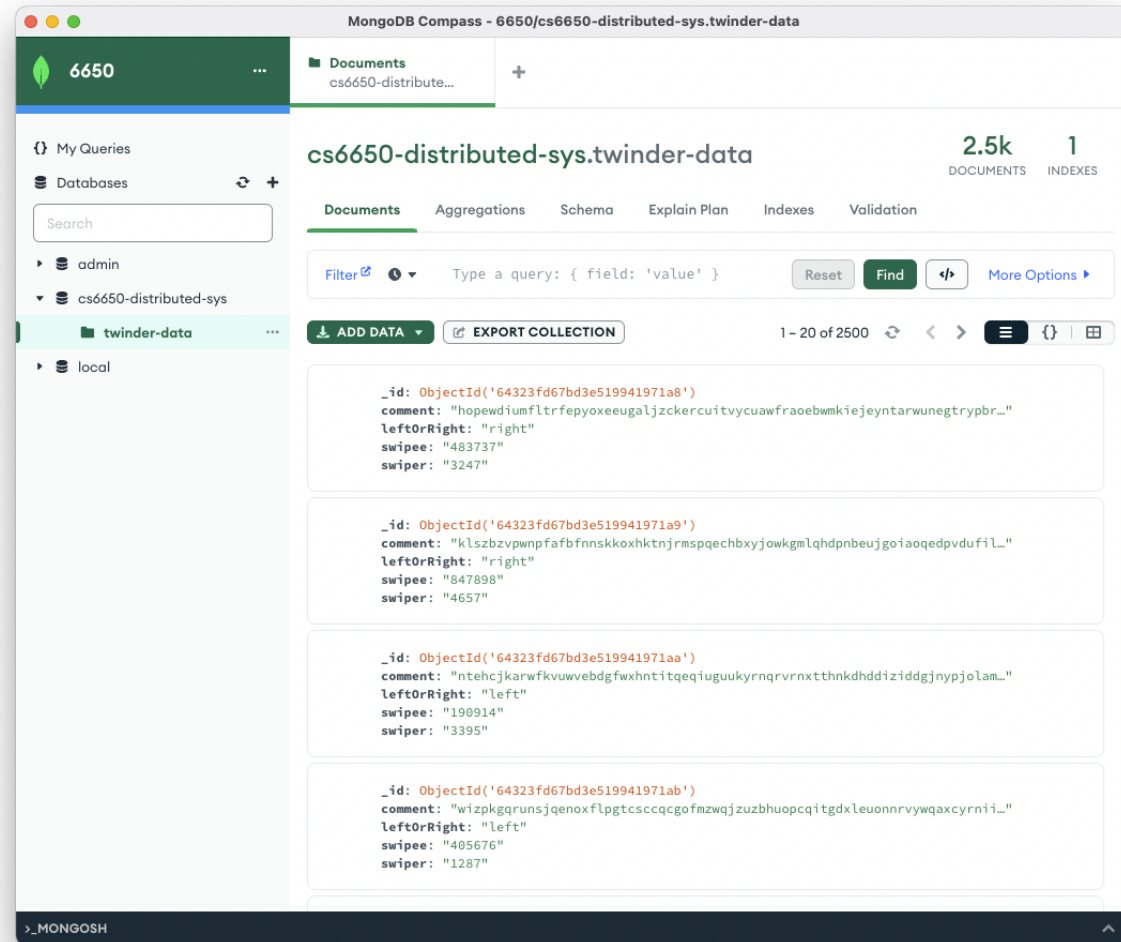
**Message Queue (RabbitMQ):** Using RMQ to manage message queues since it provides a flexible and scalable way to implement messaging between distributed systems.

**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.

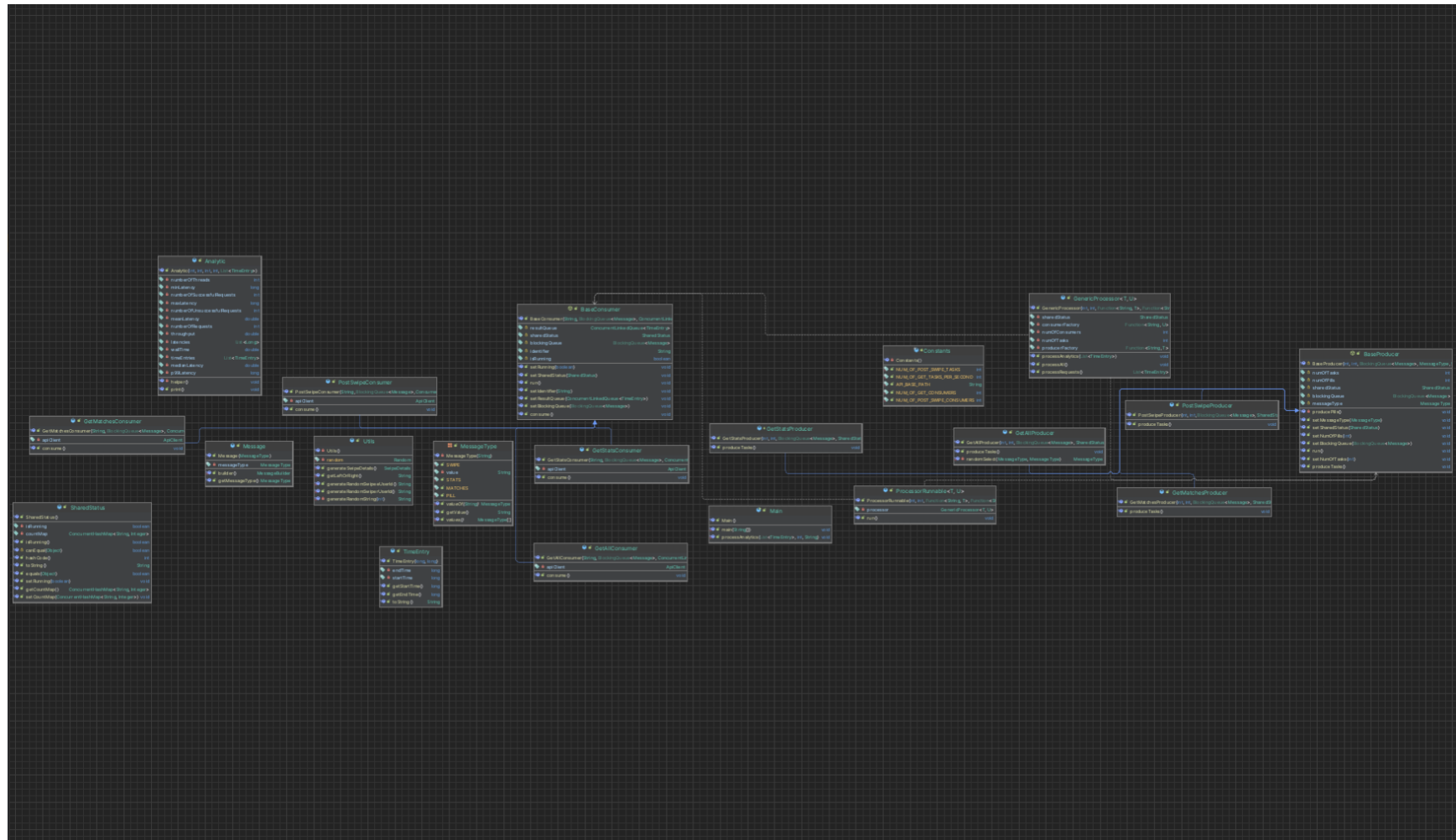
**Database (MongoDB Atlas):** The project uses MongoDB Atlas to stores information about users and swipe events. MongoDB's flexibility, scalability, and performance make it a powerful choice for this project, and works great with large and complex data requirements.

## 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.



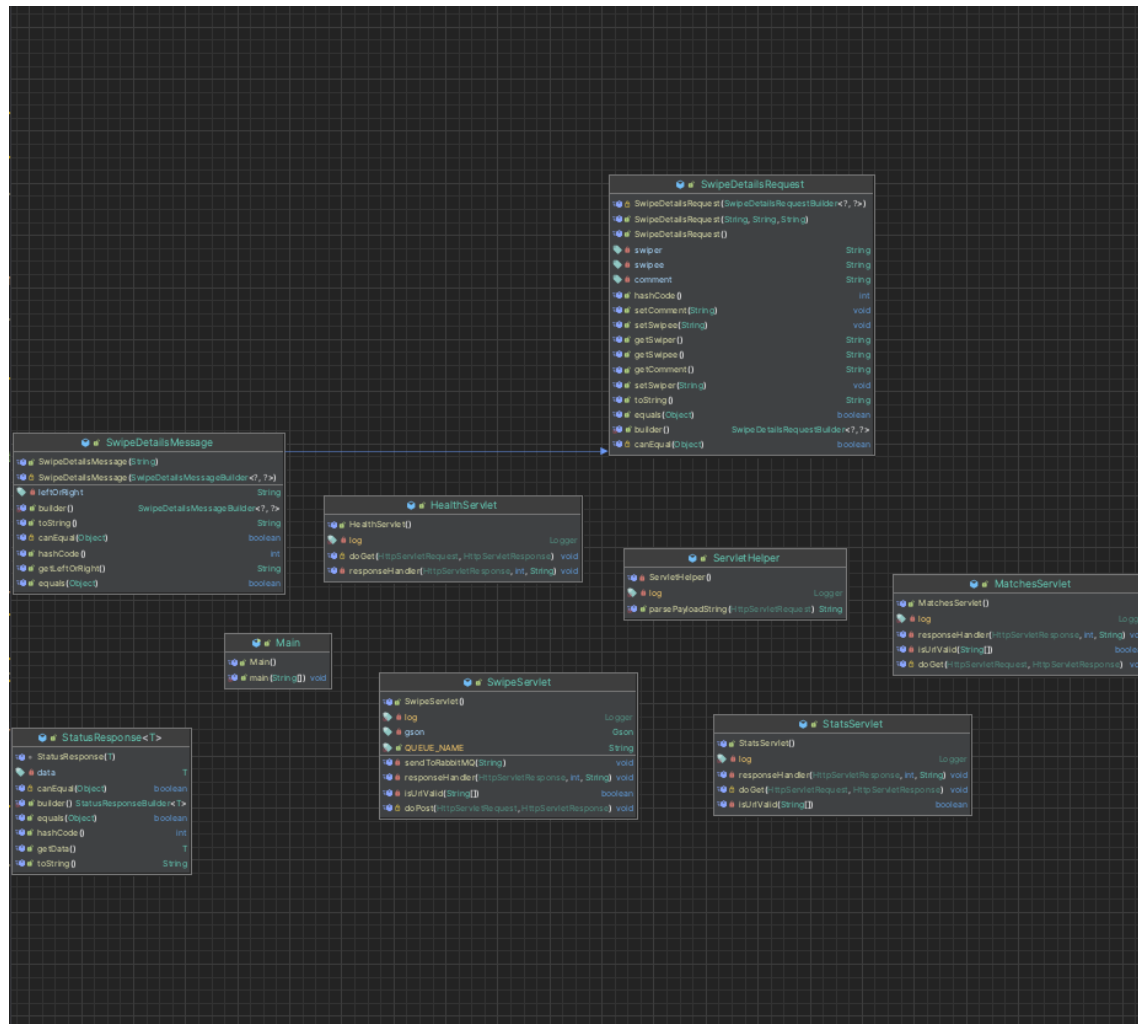
## Twinder Client UML



## Twinder Publisher Server UML

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

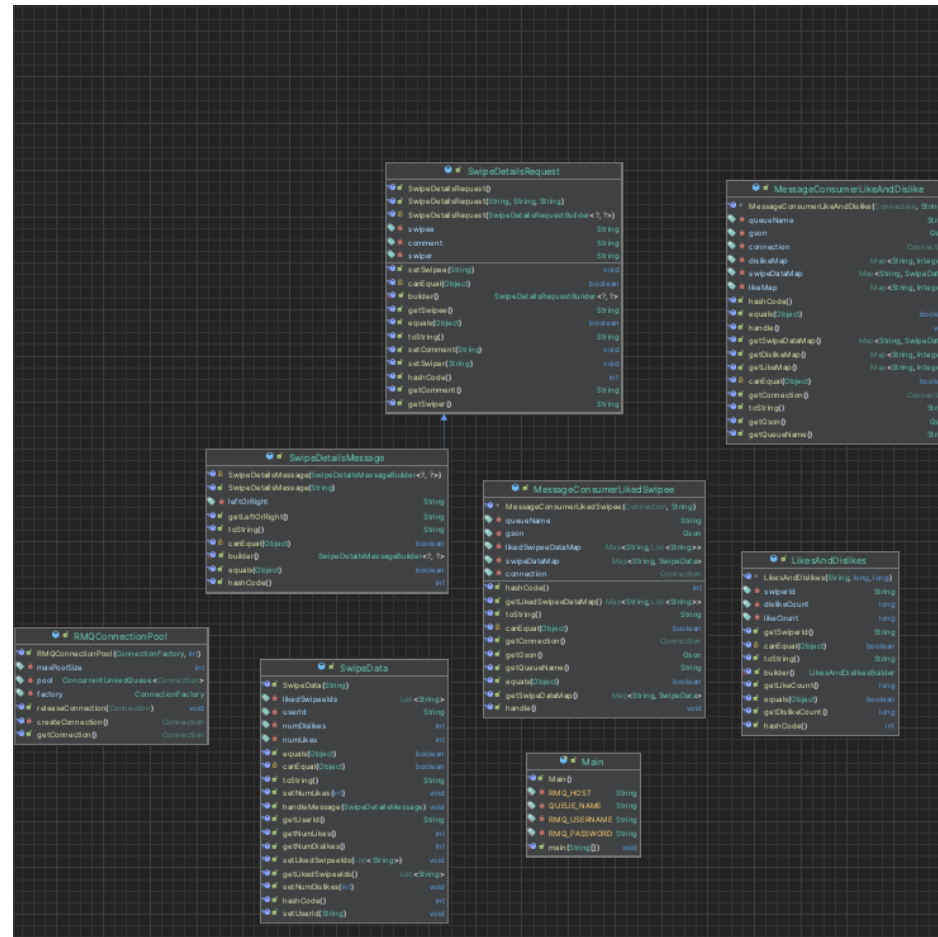
Implements the GET requests and retrieves results from the *SwipeData* database directly.



## 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

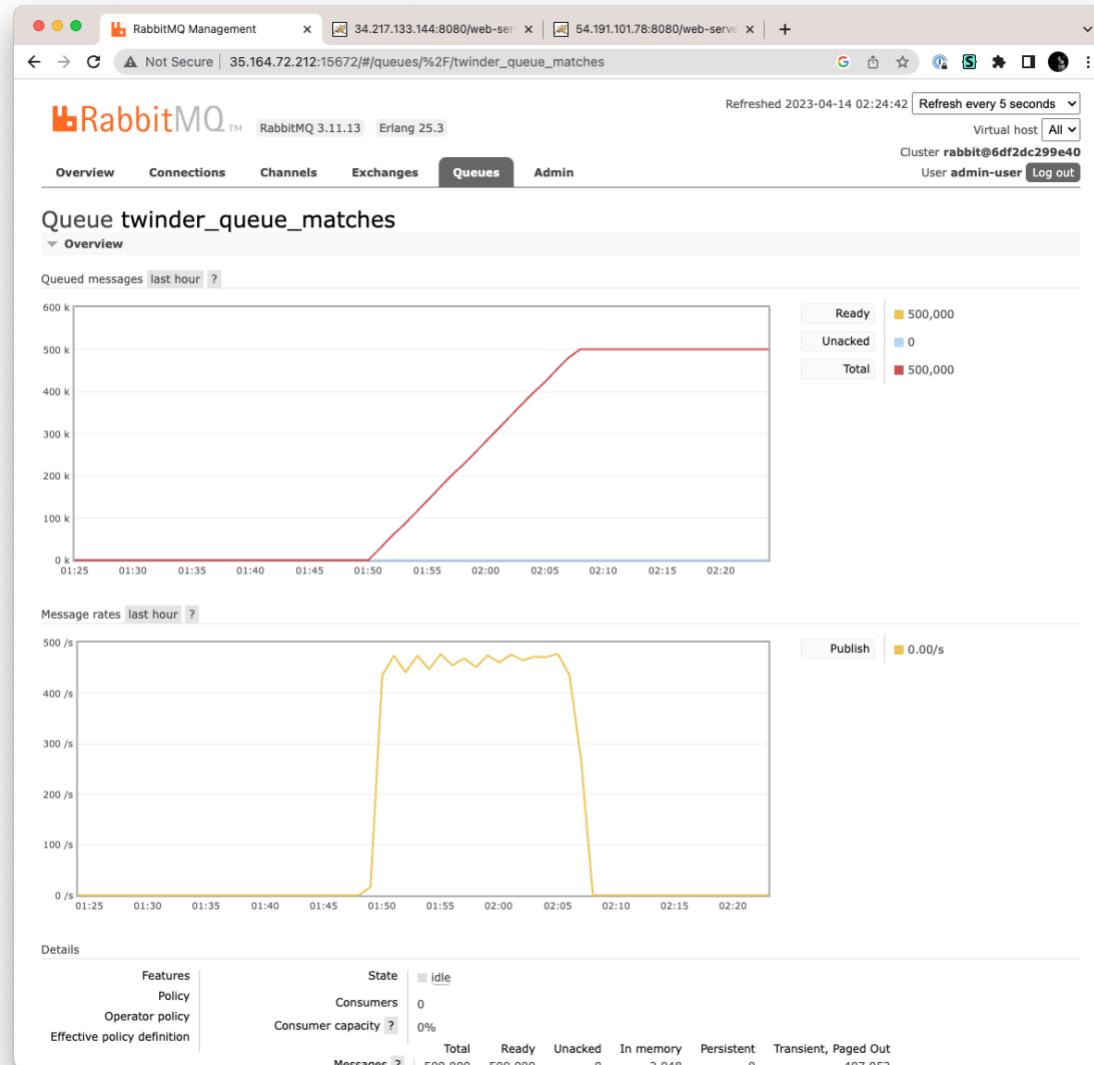
The screenshot shows the RabbitMQ Management interface in a web browser. The page title is "Queues" and it shows a list of 2 queues. The table below provides details for each queue.

Overview				Messages			Message rates			
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver	get	ack
twinder_queue_matches	classic		idle	500,000	0	500,000	0.00/s			
twinder_queue_stats	classic		idle	500,000	0	500,000	0.00/s			

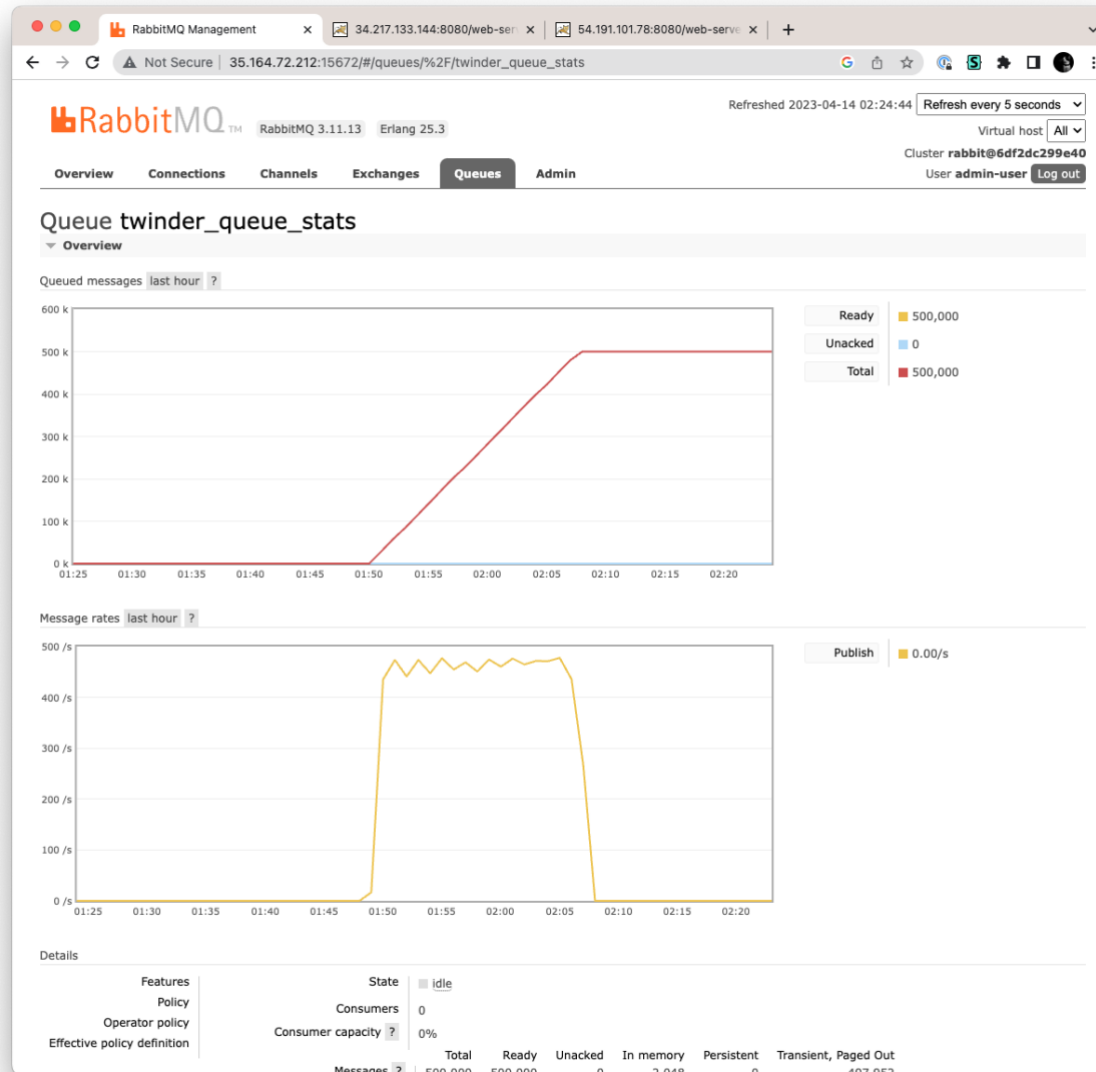
Below the table, there is a link to "Add a new queue".



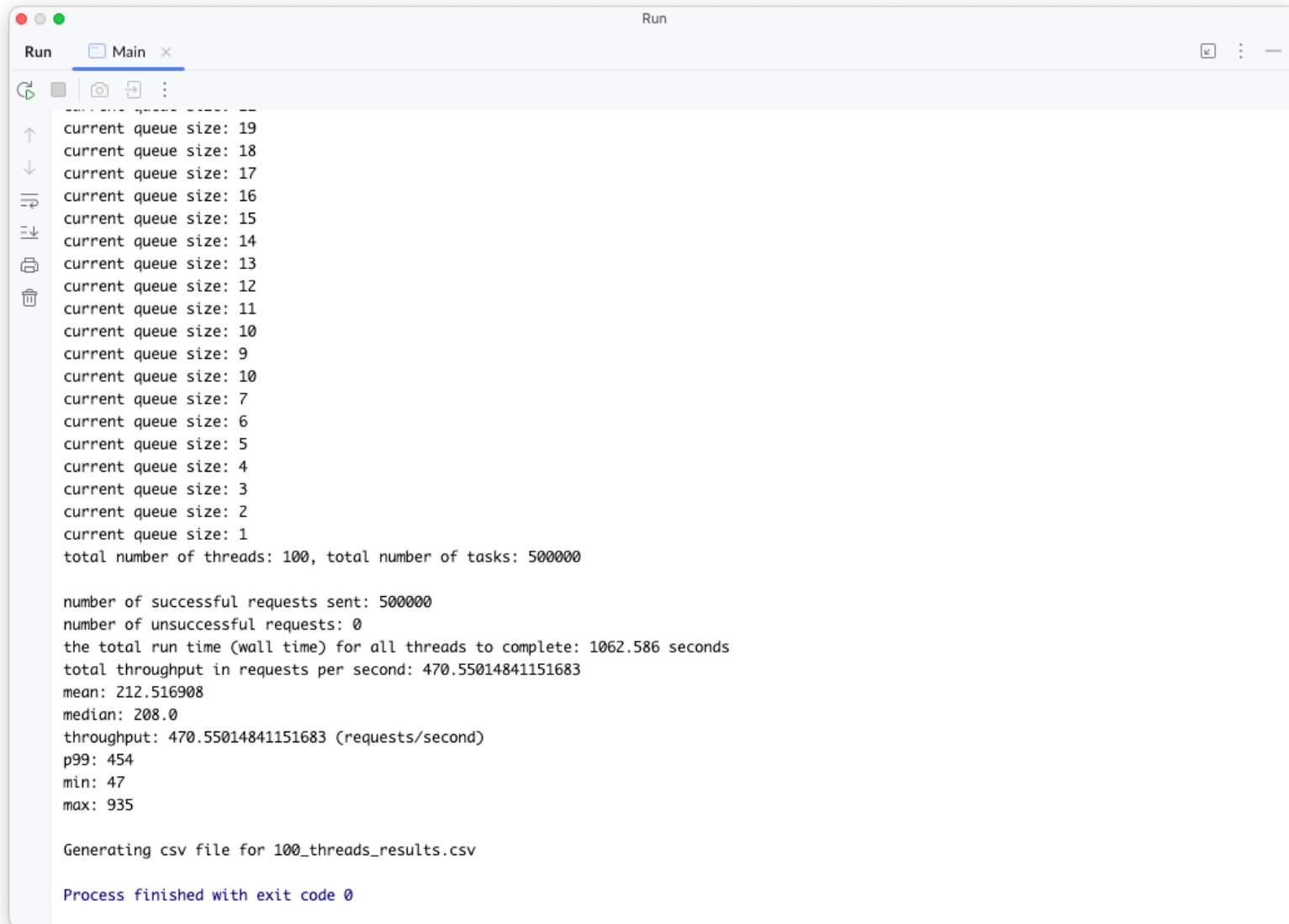
## A3 RMQ Screenshots with Flat Line Profile Twinder\_queue\_matches queue



## Twinder\_queue\_stats queue



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



The screenshot shows a terminal window titled "Run" with a tab labeled "Main". The terminal displays the output of a test run, including queue sizes, total tasks, successful and unsuccessful requests, run time, throughput, and percentile values. The process finished with exit code 0.

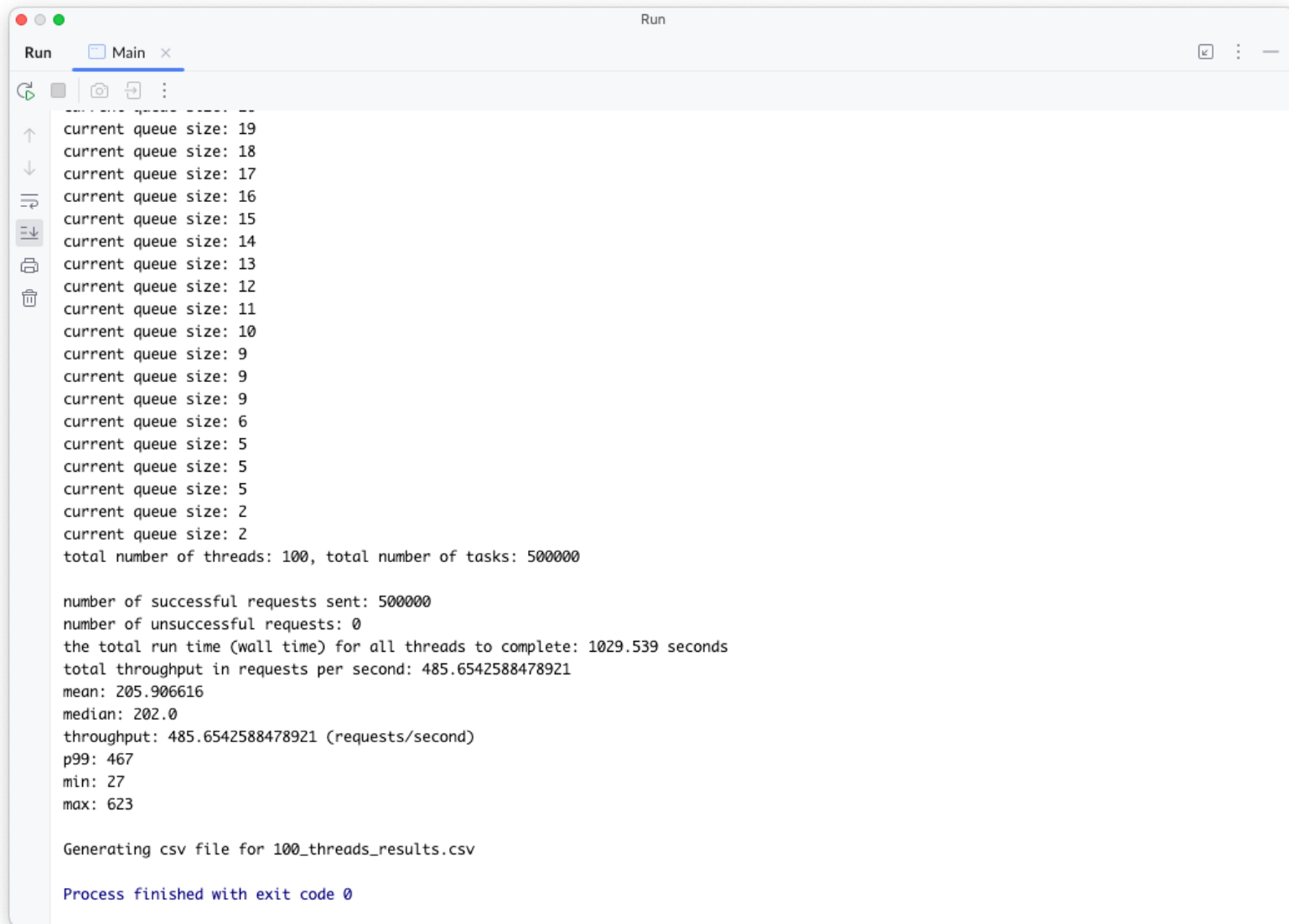
```
current queue size: 19
current queue size: 18
current queue size: 17
current queue size: 16
current queue size: 15
current queue size: 14
current queue size: 13
current queue size: 12
current queue size: 11
current queue size: 10
current queue size: 9
current queue size: 10
current queue size: 7
current queue size: 6
current queue size: 5
current queue size: 4
current queue size: 3
current queue size: 2
current queue size: 1
total number of threads: 100, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 1062.586 seconds
total throughput in requests per second: 470.55014841151683
mean: 212.516908
median: 208.0
throughput: 470.55014841151683 (requests/second)
p99: 454
min: 47
max: 935

Generating csv file for 100_threads_results.csv

Process finished with exit code 0
```

## A2 performance screenshots:



The screenshot shows a terminal window titled "Run" with a tab labeled "Main". The terminal displays the following output:

```
-----
current queue size: 19
current queue size: 18
current queue size: 17
current queue size: 16
current queue size: 15
current queue size: 14
current queue size: 13
current queue size: 12
current queue size: 11
current queue size: 10
current queue size: 9
current queue size: 9
current queue size: 9
current queue size: 6
current queue size: 5
current queue size: 5
current queue size: 5
current queue size: 2
current queue size: 2
total number of threads: 100, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 1029.539 seconds
total throughput in requests per second: 485.6542588478921
mean: 205.906616
median: 202.0
throughput: 485.6542588478921 (requests/second)
p99: 467
min: 27
max: 623

Generating csv file for 100_threads_results.csv

Process finished with exit code 0
```

## 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.