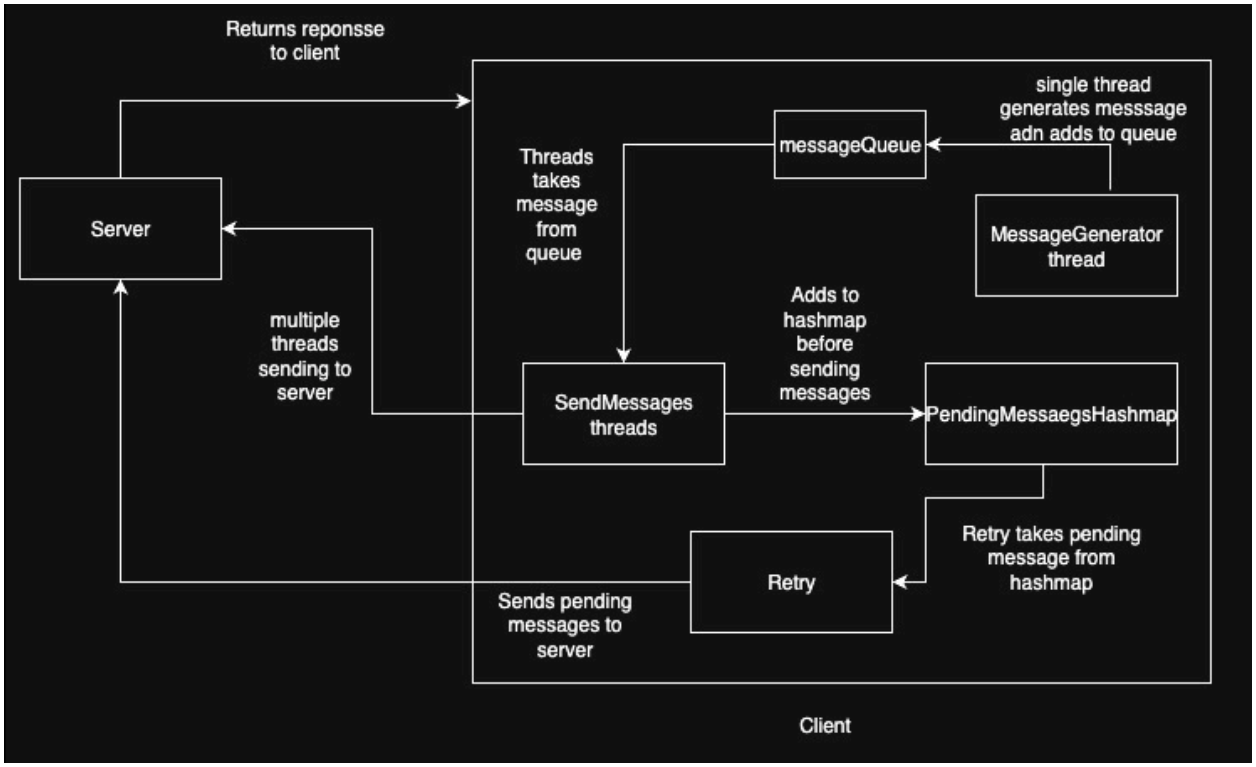
Assignment 1

Git repository URL

/server	https://github.com/RonaldRommel/Chatflow/tree/main/server
/client-part1	https://github.com/RonaldRommel/Chatflow/tree/main/client-part1
/client-part2	https://github.com/RonaldRommel/Chatflow/tree/main/client-part2
/results	https://github.com/RonaldRommel/Chatflow/tree/main/results

Design

Architecture Diagram



Major Classes and their relationships

Class	Responsibility	Relationships
Server		
WebSocketServerApplication	Entry point of the server application. Bootstraps Spring Boot or similar backend and initializes WebSocket configuration.	Instantiates WebSocketConfig and ServerController.
WebSocketHandler	Core handler for WebSocket sessions. Receives messages from clients, parses them, and broadcasts responses. Handles message types (JOIN, CHAT, ACK).	Called by the WebSocket framework when a message/event occurs. Interacts with ServerController.
ServerController	Provides REST endpoints such as /health for readiness/liveness probes and monitoring.	Exposed separately via HTTP.
WebSocketConfig	Configures WebSocket endpoints, message broker, allowed origins, and handlers.	Registers WebSocketHandler. Used by WebSocketServerApplication.
Client		
ClientApplication	Entry point for client execution. Initializes configuration, threads, and test parameters (user count, message rate, etc.).	Creates and manages multiple LoadTestClient instances.
LoadTestClient	Represents a single simulated WebSocket user. Connects to the server, sends ChatMessages, receives ChatResponses. Tracks latency and throughput.	Uses ChatMessage, PendingMessage, BufferedCSVWriter, StreamingLatency. Interacts with server over WebSocket.
ChatMessage	Represents outgoing messages (e.g., join room, send chat text). Contains messageId, roomId, timestamp, and message type.	Created by LoadTestClient and stored in PendingMessage. Serialized and sent via WebSocket.

ChatResponse	Represents a message received from the server (acknowledgment or chat broadcast). Used to measure round-trip latency.	Processed by LoadTestClient. Correlates with PendingMessage via messageId.
PendingMessage	Stores metadata about an in-flight message (send timestamp, retries, etc.) until a response is received.	Managed by LoadTestClient's pending message map. Updated on send/ack.
BufferedCSVWriter	Writes latency and throughput metrics to CSV in batches (buffered mode to avoid I/O overhead).	Used by LoadTestClient or StreamingLatency to persist metrics.
StreamingLatency	Continuously collects latency data (from PendingMessage updates) and computes moving averages or percentiles.	Used by LoadTestClient for live latency reporting; outputs to BufferedCSVWriter.
ThroughputChart	Aggregates message send/ack counts per time window to visualize system throughput over time.	Consumes metrics from StreamingLatency or logs produced by multiple clients.

Threading Model Explanation

Component	Threading Model	Description
WebSocketHandler	Event-driven, non-blocking I/O	The WebSocket server runs on a reactive thread model. Each connection is handled asynchronously no dedicated thread per connection. Incoming messages are processed in the I/O thread and dispatched to internal handlers via lightweight callbacks.
ServerController (/health)	Separate HTTP thread pool	REST endpoints like **/health** execute on a dedicated HTTP thread pool (e.g., Tomcat worker threads). These threads are independent of WebSocket I/O threads, ensuring that health checks do not interfere with live message handling.

Thread	Role	Interactions
MessageGenerator Thread (1 per client)	Continuously generates chat messages at a configured rate and enqueues them into a shared messageQueue.	Produces messages for worker threads.
SendMessages Threads (N per client)	Multiple worker threads consume messages from the queue and send them to the WebSocket server. Each message is recorded in the PendingMessagesHashMap along with its timestamp for latency tracking.	Producer-consumer relationship with MessageGenerator.
Retry Thread (1 per client)	Periodically scans the PendingMessagesHashMap to find messages that have not received a server ACK within a timeout. Retries those messages and updates retry metrics.	Ensures reliability and consistency.

WebSocket Connection Management Strategy

The client manages multiple persistent WebSocket connections to simulate real-time chat users sending and receiving messages concurrently.

1) **Connection Pool Design**
    - **a)** The client pre-creates a pool of WebSocket connections (POOL_SIZE = 20) during startup.
    - **b)** Each connection is associated with a unique room ID to simulate traffic across different chat rooms.
    - **c)** Connections are stored in a shared list connectionPool, which is used by sender threads to distribute messages evenly.
2) **Connection Lifecycle**

Stage	Description

Creation	Each connection is instantiated as a WebSocketClient object with custom handlers for onOpen, onMessage, onClose, and onError.
Open	On successful handshake, connectionCount increments and the client becomes ready to send/receive messages.
Active	Connections remain open throughout the test phase, handling both outgoing messages and incoming acknowledgments concurrently.
Close	After all test phases complete, connections are closed gracefully using closeBlocking() to ensure full cleanup.
Error Handling	Failed or broken connections are logged via onError. Metrics track broken connections for reliability analysis.

3) **Message Routing & Connection Selection**
   When sending messages:
   a) Threads pick an active WebSocket connection from the pool in a round-robin manner
   b) This ensures even load distribution across all open connections.
   c) If a connection is temporarily closed, the loop skips it and tries the next available one.

Little's Law Calculation and Predictions

Formula:

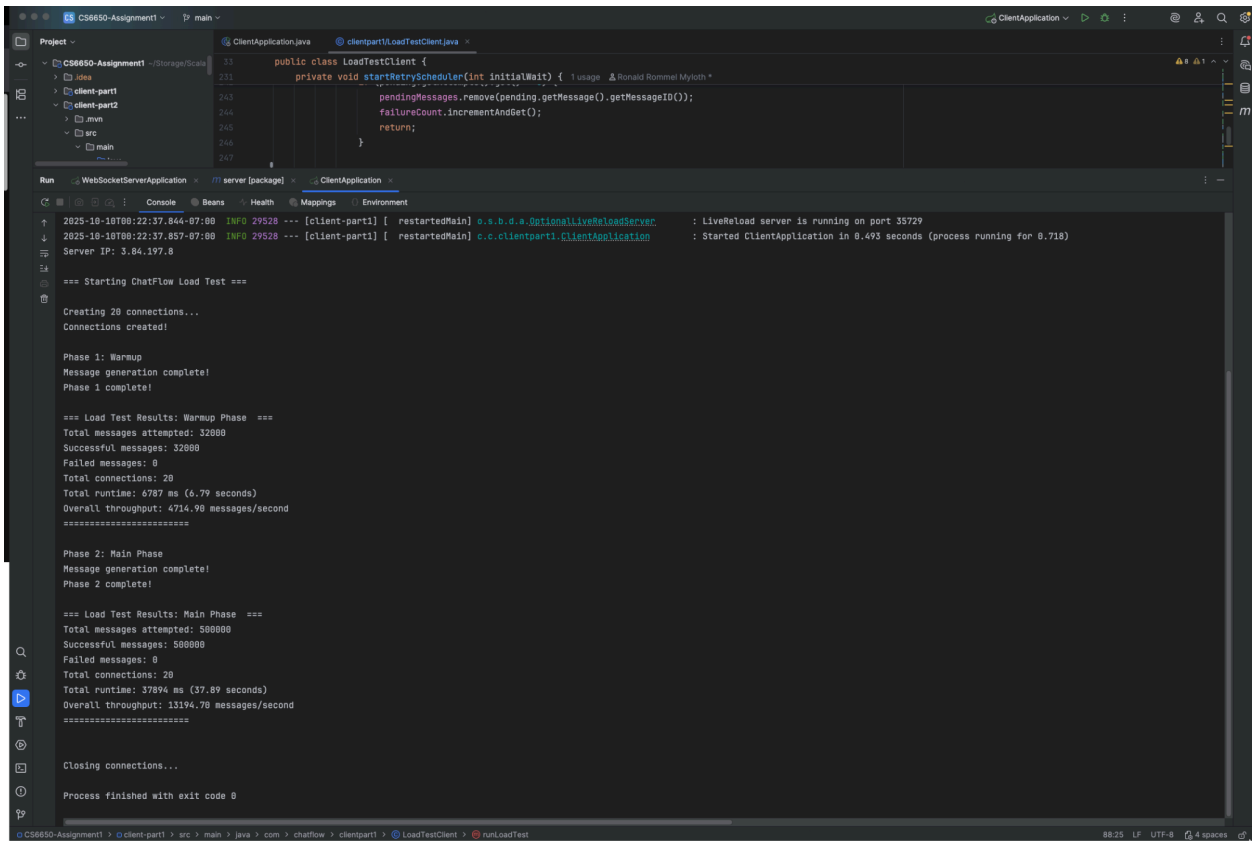
$$L=\lambda \times W$$

Prediction:
RTT for 1 message: 150ms
Suppose each thread can send 100messages/sec.
With 20 connections its 2000 messages
Total messages = 2,000
Test duration ≈ 0.15 s → λ ≈ 2000 / 0.15 = 13333.33 messages/s

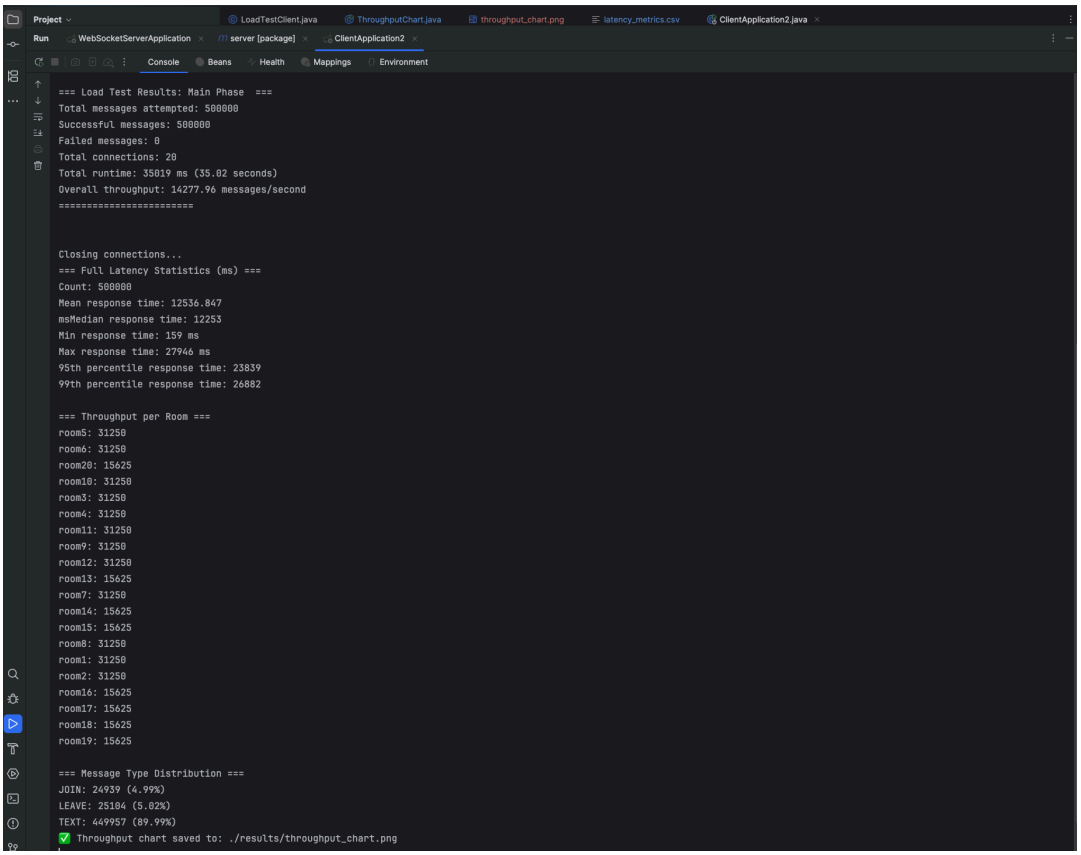
Actual:
Mean latency (W) = 15,114 ms = 15.114 s
Total messages = 500,000
Test duration ≈ 37.89 s → λ ≈ 500000 / 37.89 = 13196.10 messages/s

Test Results

Screenshot of Part 1 output (basic metrics)



Screenshot of Part 2 output (detailed metrics)



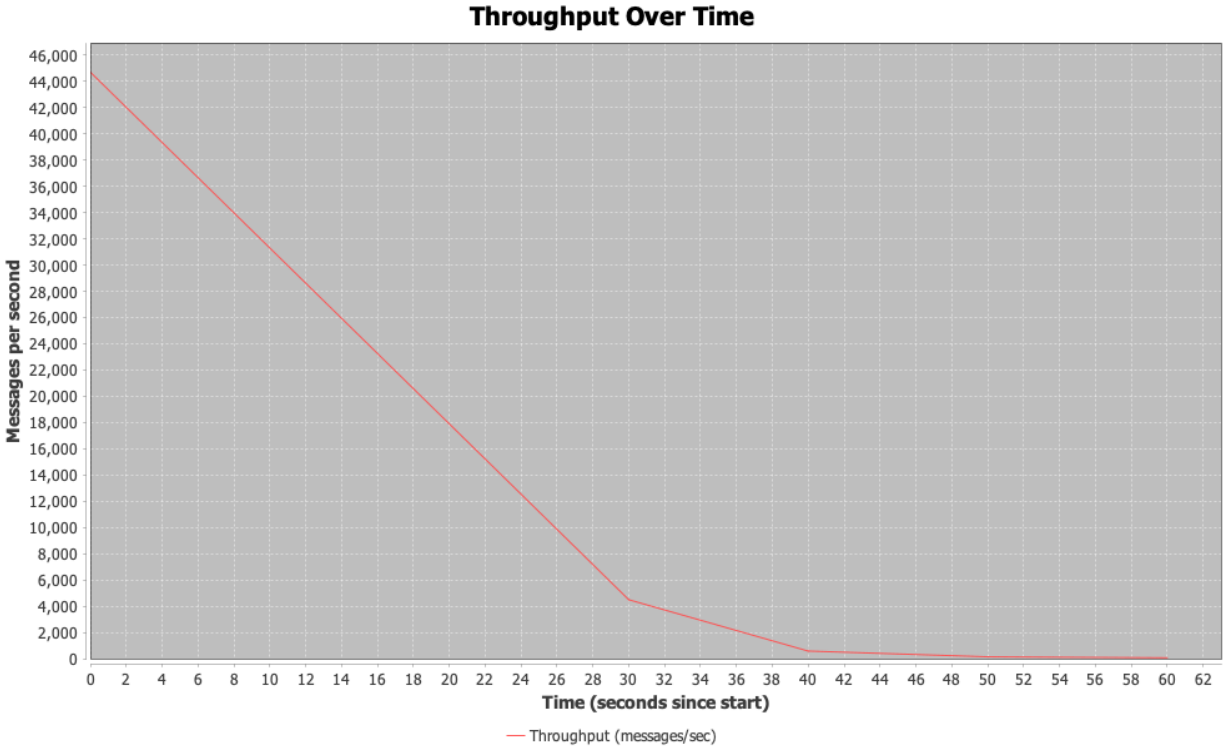
```
=== Load Test Results: Main Phase  ===
Total messages attempted: 500000
Successful messages: 500000
Failed messages: 0
Total connections: 20
Total runtime: 35019 ms (35.02 seconds)
Overall throughput: 14277.96 messages/second
========================


Closing connections...
=== Full Latency Statistics (ms) ===
Count: 500000
Mean response time: 12536.847
msMedian response time: 12253
Min response time: 159 ms
Max response time: 27946 ms
95th percentile response time: 23839
99th percentile response time: 26882

=== Throughput per Room ===
room5: 31250
room6: 31250
room20: 15625
room10: 31250
room3: 31250
room4: 31250
room11: 31250
room9: 31250
room12: 31250
room13: 15625
room7: 31250
room14: 15625
room15: 15625
room8: 31250
room1: 31250
room2: 31250
room16: 15625
room17: 15625
room18: 15625
room19: 15625

=== Message Type Distribution ===
JOIN: 24939 (4.99%)
LEAVE: 25104 (5.02%)
TEXT: 449957 (89.99%)
✅ Throughput chart saved to: ./results/throughput_chart.png
```

Performance analysis charts



Evidence of EC2 deployment

```
~/Storage/Scalable Systems — ec2-user@ip-172-31-31-253:~ — ssh -i chatflow-key.pem ec2-user@3.84.197.8        ~/Storage/Scalable Systems/CS6650-Assignment1/server — -zsh

(base) ronaldrommelmyloth@Ronald-Macbook Scalable Systems % ssh -i chatflow-key.pem ec2-user@3.84.197.8
       #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Fri Oct 10 06:24:51 2025 from 24.19.200.175
[ec2-user@ip-172-31-31-253 ~]$ java -jar server-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.6)

2025-10-10T07:29:14.228Z  INFO 4461 --- [server] [           main] c.c.server.WebSocketServerApplication    : Starting WebSocketServerApplication v0.0.1-SNAPSHOT using Java 17.0.16 with PID 4461 (/home/ec
2-user/server-0.0.1-SNAPSHOT.jar started by ec2-user in /home/ec2-user)
2025-10-10T07:29:14.233Z  INFO 4461 --- [server] [           main] c.c.server.WebSocketServerApplication    : No active profile set, falling back to 1 default profile: "default"
2025-10-10T07:29:16.869Z  INFO 4461 --- [server] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2025-10-10T07:29:16.888Z  INFO 4461 --- [server] [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-10-10T07:29:16.889Z  INFO 4461 --- [server] [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.46]
2025-10-10T07:29:16.938Z  INFO 4461 --- [server] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-10-10T07:29:16.940Z  INFO 4461 --- [server] [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2586 ms
2025-10-10T07:29:17.912Z  INFO 4461 --- [server] [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 1 endpoint beneath base path '/actuator'
2025-10-10T07:29:18.004Z  INFO 4461 --- [server] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2025-10-10T07:29:18.096Z  INFO 4461 --- [server] [           main] c.c.server.WebSocketServerApplication    : Started WebSocketServerApplication in 4.658 seconds (process running for 5.38)
```