A Messaging App

Test Plan Document

Table of Contents

1 Introduction

The project that this document discusses is a project to create a messaging app. This document is the test-plan document which will go over the test-plan for the project. The document will discuss the test-plan that the project will implement in order to guarantee functionality targets; please see the table of contents for a full illustration of topics.

1.1 Purpose and Scope

The purpose of this document is to outline the test-plan for the messaging app in order to give a clear path of implementation of the testing suite. The testing suite itself will be the method of guaranteeing that the project has met the full functionality requirements. However, the document will not go into much detail about the design of the app, or the actual implementation of features; it is about the testing of the application, not the implementation of it. For implementation details, please see the design document; for the full list of requirements, please see the requirements document.

1.2 Target Audience

The target audience of this app will be people who prefer a very minimalist approach to messaging.

1.3 Terms and Definitions

- Client: A client is a single instance of the app that a user will use to log into the server.
- Server: A server is a single instance of a server app that will be used as the main interface between all clients.
- I/O: Input/Output is all of the input and output into and from a system.
- User: A person that uses a client in order to use the messaging app.
- Socket: A "communication tunnel" that allows for I/O between server and client.
- Message: A catch-all object for any interaction between the server and client; can be a message from a client to client, client to global, client to group, or command to server.
- global: Global is the global chat that all users are automatically in. Any message to global is broadcast to all users.
- group message: group message is a message that is sent to a group (or subset) of users.
- Unit: A singular concept of the program. In this case, an example of unit would be the Server. It may or may not be a single class but it encompasses a singular concept.
- Scenario: A scenario is something that might happen in the messaging app in real world usage. Example: User 1 joins, sends a message to global, then quits.

- Integration Testing: A method of testing that combines multiple units together in order to test how they interact; more abstract than Scenario Testing.
- Scenario Testing: A method of testing that plays out a scenario to ensure that multiple units work well together.
- Big Bang approach: A method of integration testing that combines all or almost all units at once to test at one time; this will use Scenario Testing, specifically, due to how well it fits the application.

2 Test Plan Description

Always assume "clean environment" is given. Not all unit tests shown here will be implemented as some are duplicates; they are given for clarity of coverage and duplicates will be merged as convenient.

2.1 Scope of Testing

2.1.1 Testing Schedule

2.1.2 Release Criteria

3 Unit Testing

3.1 Server

The server is what holds all of the clients together and organizes them. It handles: authenticating the clients, connecting the clients, and disconnecting the clients; broadcasting data to the clients, receiving data from the clients, and receiving commands from the clients; and the logging of messages as well as keeping chat histories for the clients.

Testing on this unit will make sure that these things are, at a minimum level, functional under normal situations; testing will not assume actual distributed internet connections as regular java objects will be passed through sockets. Testing will be designed to ensure that normal behavior is satisfied with basic fault tolerance, sanitation, and verification utilized; IO and sockets can be messy and the app will not attempt to be overly robust in fault tolerance.

3.1.1 socket

The socket handling will handle everything related to sockets: in this case, the authentication of clients, connecting of clients, and disconnecting of clients.

- Listen for client: Expect correct "hearing" when a finite number of well-defined clients request connection.
- Authenticate clients: Expect already created user to be authenticated with correct credentials, expect new user prompt to be triggered given un-recognized credentials, expect re-try prompt to be triggered given recognized username but invalid credentials.

  - Gives chat history when connected: Expect a client with pre-existing history to display correct chat history when connected, expect a client with no history to display correct history, expect a client with multiple chats to display correct histories for all chats.
  - Announces connection: Expect all clients to receive broadcast whenever a client connects, expect disconnected clients to receive any missed connected broadcast as chat history.

- Disconnect client: Expect client to be disconnected correctly on both server and client end, expect logs to be cleaned and closed correctly.
  - Announces disconnection: Expect all clients to receive broadcast whenever a client connects, expect disconnected clients to receive any missed connected broadcast as chat history.

3.1.2 broadcast

- Logs messages: Expect any message to be stored correctly in its appropriate data structure, expect no cross contamination.
- Displays messages: Expect a server command to be handled and not displayed to anyone except own client; expect one to one message, one to group, and one to global, to be displayed correctly and stored correctly. Expect a client joining a pre-existing multi chat to receive prior history correctly.

3.1.3 logs all messages

- Client to global: Expect message to be stored correctly.
- Client to client: Expect message to be stored correctly.
- Client to clients: Expect message to be stored correctly.

3.1.4 displays all messages

- Client to global: Expect a server command to be handled and not displayed to anyone except own client; expect one to one message, one to group, and one to global, to be displayed correctly and stored correctly. Expect a client joining a pre-existing multi chat to receive prior history correctly.
- Client to client: Expect a server command to be handled and not displayed to anyone except own client; expect one to one message, one to group, and one to global, to be displayed correctly and stored correctly. Expect a client joining a pre-existing multi chat to receive prior history correctly.
- Client to clients: Expect a server command to be handled and not displayed to anyone except own client; expect one to one message, one to group, and one to global, to be displayed correctly and stored correctly. Expect a client joining a pre-existing multi chat to receive prior history correctly.

3.1.5 handles commands

Expect a server command to be handled and not displayed to anyone except own client. Expect a server command to be executed and change state correctly. Expect that such changed state is stored and saved properly.

3.2 Client

The client is what holds all of the information about the client together and organizes it. It handles: User data, such as the username, password, display name, and any possible implemented options; requesting a connection/authentication from server; sending a message, receiving a message, and displaying a message.

Testing on this unit will make sure that these things are, at a minimum level, functional under normal situations; testing will not assume actual distributed internet connections as regular java objects will be passed through sockets. Testing will be designed to ensure that normal behavior is satisfied with basic fault tolerance, sanitation, and verification utilized; IO and sockets can be messy and the app will not attempt to be overly robust in fault tolerance.

3.2.1 User Data

In general, expect that fields are created correctly. Expect that fields are updated/saved correctly if modified with appropriate and well-formed commands.

- Username: Expect that username is not allowed to be changed. Expect that username is not allowed to be a duplicate of someone else's.
- Password.
- Display name: Expect that display name is not allowed to be a duplicate of someone else's.

3.2.2 Socket

- Connects to server: Expect that an authentication is successful if credentials are valid. Expect that a user creation process is successful if new credentials are valid. Expect that an invalid authentication prompts a re-try.

3.2.3 Message

In general, all properties will be tested with server command, one to one, one to group, and one to global messages.

- Sends message: Expect that sending a message sends to server properly. Expect that failure notifies user somehow. Expect that no message is sent to someone that does not exist. Expect that no message is accessible by someone that does not have permission.
- Receives message: Expect that messages are sent to user correctly. Expect that no message is received that is not meant for user.
- Displays message: Expect that all messages are displayed correctly.

3.3 Message

3.3.1 From

- One client: Expect that a message cannot be from two clients.

3.3.2 To

This will be merged with message testing inside Client Unit.

- One client:
- All clients:
- List/group of clients:
- Command to server:

3.3.3 Data

- Message content: Expect that message content is not garbled. Expect that empty messages are not sent. Expect that rapid sending of messages does not change order of messages.
- Command to server: Expect that no chat message is interpreted as command. Expect that no command is interpreted as chat message.

3.4 Client GUI

3.4.1 interface

- Login: Expect that login has an input for all necessary values. Expect that non-existing credentials prompt a creation or re-try.
- Chat interface: Expect that chat interface has a chat box, a chat window(s) and/or chat window with tabs.
- Supports multiple “windows” (either tabs or multi-window) Expect that all chats can be accessed by however method is chosen.
- Chat box: Expect that the chat box can receive input from keyboard. Expect that the chat box does not trigger on empty input.
- Mouse-friendly: Expect that all functionality can be achieved through mouse input. In practice this might not be tested in an automated way, depending on testing library support for this.

4 Integration Testing

Integration testing here will be slightly different. I have opted for a Big Bang approach, which is where all units are smashed together and the entire app as a whole is tested. The reason I have opted for this is because the messaging app itself is not very complicated; thus, multiple levels of integration testing or any level of testing above integration testing will yield rapidly diminishing results with regards to overhead on my part vs benefit the app receives in robustness due to testing.

In all reality, the app is small enough that "Integration Testing" can be achieved merely by compiling and running the entire program and testing myself. However, in order to speed up and solidify the process, small chat bots will likely be utilized and pre-scripted scenarios will be executed. So, the test cases will be "scenarios" and not a more traditional Use Case.

4.1 Scenario 1

This scenario is designed to be entirely comprehensive all by itself. It will be implemented bits at a time until the entire scenario is implemented as a test. This will serve as an incremental process of testing that can be done while development is being completed. It will also remove the immediate necessity to have multiple scenarios without removing the possibility of having them in the future should more features be added that require scenario testing.

- User 1 joins. User 1 sends message to global.
- User 2 joins. User 1 sends message to User 2.
- User 3 joins. User 2 sends message to User 1 and User 2.
- User 4 joins. User 4 sends message to global.
- Users 2 adds User 3 to group message of User 1 and User 2.
- User 1 changes nickname.
- User 3 sends message to group message.
- User 3 leaves. User 2 sends group message. User 1 adds User 4.
- User 4 sends message to User 3. User 4 sends message to User 2. User 4 sends message to group message.
- User 3 tries to change username. User 4 changes nickname to User 1's old nickname.
- User 3 joins again. User 3 sends message to group message.
- User 1, 4 quit. User 2 sends message to global.
- User 2, 3 quit.
- User 5 joins. User 5 sends message to global.
- User 1, 2, 3, 4 join. User 5, 4, 3, 2, 1 quit.