Waterfall Chat

Test Plan Document

Table of Contents

1 Introduction

This document will describe and outline the test plan for the Waterfall Chat, which will ensure that it meets all requirements initially outlined for production. It will describe test information in detail for all units of the program, as well as the entire system.

1.1 Purpose and Scope

As stated above, the test plan will ensure that the program meets all of the requirements laid out previously in the design process, by allowing all use cases and components, as well as the overall system to be systematically tested. This will involve giving a passing case and an exceptional case for each contingency.

1.2 Target Audience

This document is intended for Tegan, the author, to allow her to effectively test the program. It's also intended for the TA, and Professor Xie, to allow them to gain an understanding of my test process, and the overall success of my program as it relates to the initial design requirements.

1.3 Terms and Definitions

- User - A client (user of the program) that logs into the server.
- Server - A program that accept connections from multiple users, and allows them to communicate with each other.
- Chatroom - An online space that allows users to communicate with each other by sending public messages to everyone else in the room.
- Private Messaging - The sending of a message from one user to another user.

2 Test Plan Description

In this section, we're going to go over the overall structure of the test plan. This is including the scope, which is what will and will not be tested, the testing schedule, which is a timeline for when certain criteria and modules will be tested, along with the release criteria, which is going to be the maximum allowed fault tolerance and the criteria the program must meet before deployment.

2.1 Scope of Testing

As I (Tegan) am the only designer, programmer, and tester, I plan to approach testing in a fairly holistic manner. It won't be difficult to access all the modules individually, and design a comprehensive test plan that addresses the majority of the potential issues.

I'll be testing most individual methods, as well as the larger sub-systems such as those that take input from the output of another method. Above that, the individual client programs will be tested once completed, as well as the server, and then finally all of the units together. I will not be testing the interaction of large volumes of clients, as I believe that testing 5 to 10 will give accurate results to an order of magnitude more (50-100).

2.2 Testing Schedule

Most individual modules will be tested as written. Since the code is usually <20 lines, this is the quickest way to test the low level components. As the complexity rises, larger components that interact with others will be tested at longer intervals, such as when sub-systems get completed (such as account creation, or client connection.) Once most components are written, and most subsystems are working, larger scale testing will ensue

to ensure that all of the modules effectively communicate with each other and work well together.

2.3 Release Criteria

As this is a term project, and the scope is well within what I'm capable of programming and testing myself, the release criteria will be such that nearly all cases will be tested and working before deployment. As I don't have the option of pushing updates and fixing issues post-deployment, I'm going to strive to achieve success in all of my approaches to testing.

3 Unit Testing

Here I will be describing the individual unit tests of my program. For each one I will describe the details of the test, the inputs and outputs, as well as the expected result. These will be listed as test cases for each unit, along with what determines a pass or fail for that criteria.

3.1 Server Account Creation

This unit encompases all of the utility and functionality of creating user accounts on the server. This involves account creation and storage, and credential setting. It will take input of a username and password. It will then search the list of users already on the server, and determine whether the new username is a duplicate. If not, it will create a new node in the hash table corresponding to that user, and return a success. If a duplicate username is found, it will return a failure. Successful deployment will hinge on the entirety of this unit being 100% functional.

3.1.1 Test Case 1: Valid Username and Password

For this test case, we will attempt to create a new user account with a valid username and password. The server should take this information, search the userlist, and create a new account if the credentials are valid and the username is not taken, along with returning true. This will indicate a success. If the username is already in the list, it should return false, which will also be a success. If the behavior deviates from this, it will indicate a unit failure.

3.1.2 Test Case 2: Invalid Username and Password

For this test case, we will attempt to create a new user account with a invalid username and password. The server should take this information, search the userlist, and find a match, of which it returns false. If the behavior deviates from this, it will indicate a unit failure.

3.2 Server Login Credentials Validation

This unit encompases all of the utility of checking a user's login credentials. It will take input of a username and password. It will then search the list of users already on the server, and determine whether the user has an account. If not, it will return false indicating a failure. If the username is found, it will then check the passwords against each other. Should they match, it will return true, otherwise, it will return false. Successful deployment will hinge on the entirety of this unit being 100% functional.

3.2.1 Test Case 1: Invalid Username

For this test case, we will attempt to login with a username that doesn't have an account on the server. The unit should return false, indicating that the account doesn't exist. Any behavior other than this indicates a failure of the module.

3.2.2 Test Case 2: Valid Username and Invalid Password

For this test case, we will attempt to login with a valid username and an invalid password. The module should return false, indicating the password doesn't match. Any behavior other than this indicates a failure of the module.

3.2.3 Test Case 3: Valid Username and Password

For this test case, we will attempt to login with a valid username and password. The module should check these, find the username, find that the password matches, and return true. Any behavior other than this will indicate a module failure.

3.3 Server Account Logout (Disconnect)

This unit encompases the ability of the server to allow a client to logout, or disconnect from the server. Successful deployment *does not* hinge on the entirety of this unit being 100% functional, as the functional requirements will not be hindered by this test.

3.3.1 Test Case 1: Invalid Username

For this test case, we will attempt to disconnect a user that is not currently logged in, or that does not currently have an account on the server. The module will take in a username, search the list of users, and return false indicating that the user was not found. Any behavior other than this indicates a failure of the module.

3.3.2 Test Case 2: Valid Username

For this test case, we will attempt to disconnect a user that is currently logged in. The module will take in a username, search the list of users, and return true indicating that the user was not found. Any behavior other than this indicates a failure of the module.

3.4 Server Display Account Records

This unit encompases the ability of the server to show chat records for a given user. The server will take in a username and return the chat records if the user was found. If the user was not found, it will return a failure flag, indicating the user was not found. Successful deployment hinges on the entirety of this unit being 100% functional.

3.4.1 Test Case 1: Invalid Username

For this test case, we will attempt to show records for a user that is not currently logged in, or that does not currently have an account on the server. We will pass in a non-valid username, and the module should return a failure flag. Any behavior other than this indicates a failure of the module.

3.4.2 Test Case 2: Valid Username

For this test case, we will attempt to show records for a user that is currently logged in. We will pass in a valid username, and the module should return all of the chat records for that user, in an array of strings, or similar manner. Any behavior other than this indicates a failure of the module.

3.5 Client Send Public Message

This unit encompases the ability of the client to send a message to all users currently logged in on the server. The client will send out a socket message containing the message to be sent, as well as a flag indicating it should be sent to all users. Successful deployment hinges on the entirety of this unit being 100% functional.

3.5.1 Test Case 1: Sending a message

For this test case, we will attempt to send a message to all users. The user will type a message into the send box, and if no user is selected when they press enter, a message will be sent out using a UDP socket, to the server, along with a flag indicating it's a public message. Any behavior other than this indicates a failure of the module.

3.6 Client Send Private Message

This unit encompases the ability of the client to send a message to all users currently logged in on the server. The client will send out a socket message containing the

message to be sent, as well as a flag indicating it should be sent to all users. Successful deployment hinges on the entirety of this unit being 100% functional.

3.6.1 Test Case 1: Sending a message

For this test case, we will attempt to send a message to a single user. The user will type a message into the send box, and if a user is selected when they press enter, a message will be sent out using a UDP socket, to the server, along with a flag indicating it's a private message, along with a third field indicating which user to send the message to. Any behavior other than this indicates a failure of the module.

4 Integration Testing

In this section we will explore and lay out integration testing for the chat program. Compared to unit testing, this is a more holistic approach that encompases the entirety of the program, rather than testing that encompases individual modules. Success of the program and ensurance of meeting all of the initial design requirements hinges on the entirety of this section being tested with successful outcomes. As this is testing the entirety of the program, these tests will be performed after the majority of the code (>95%) has been written, as opposed to the unit testing, many of which are tested as they are written.

4.1 Client to Server Communication

In this test, we are going to ensure that the client server communications are functioning as intended. Successful deployment hinges on the entirety of this test being 100% functional.

This will involve the following:

- Connecting the client to the server, and ensuring a valid connection.
    - Success will be dependant on a valid client ID being assigned to the client by the server, and the server storing this client ID.
- Creating a new user account from the client on the server.
    - Success will be dependant on the server creating the new user account from the client.
- Logging in to the server from the client.
    - Success will be dependant on the server matching the client ID with the username logged in, and the client acknowledging this.
- Checking user records.
    - Success will be dependant on the client requesting user account records from the server, and the server responding correctly to that client with the relevant information.

4.2 Multi-Client to Server Communication

For this test, we are merely going to ensure that the above test (Ref. 4.1) will work given multiple clients. This test will be done by launching between five and ten client programs, connecting them all to the server, and creating unique users for all of those clients. Once they are all logged in, each individual client will be tested for its unique functionality, which should all be successful. Successful deployment hinges on the entirety of this test being 100% functional.

4.3 Client to Client Messaging

For this test, we are going to test communication between clients, which is the main purpose of the program. This will be accomplished similarly to the above test, in which 5 to 10 clients will be spawned, and each with a unique user logged in. Each client will then need to demonstrate that it can communicate publicly with all other clients, as well as individually (private) messaging each other client.

This is the reason for limiting my testing clients to between 5 and 10, as the number of tests and messages are going to be $\mathbf{n^2 - n}$. I believe with reasonable certainty that testing at this volume will successfully extrapolate out to an order of magnitude higher (such that testing 5-10 clients simultaneously will simulate testing of 50-100 clients; my original design upper limit.) Successful deployment hinges on the entirety of this test being 100% functional.