David Kim

Term Project: *Java Messenger*

Test Plan Document

Table of Contents

1. Introduction

This document describes in detail the software testing requirements for Java Messenger, a simple online instant messenger program. This document will describe the problems Java Messenger intends to address and testing requirements of the proposed system. This document is intended for the stakeholders of the application and to assist in the development process of Java Messenger as well as serve as a reference to clarify any future issues the stakeholders may encounter.

1.1 Purpose and Scope

The purpose of this Test Plan Document is to describe the testing approach and overall framework that will guide the testing of Java Messenger. This document will cover testing and execution strategies as well as test management strategies.

This Test Plan also describes the integration and system tests that will be executed on the architectural prototype following integration of the subsystems and components for Java Messenger. It is critical that all system and subsystem interfaces be tested. Testing of system functionality and features will be done throughout development to ensure software correctness.

1.2 Target Audience

- Project team member, David Kim, will perform tasks specified in this document as well as providing input and recommendations on this document.
- The stakeholders include Dr. Fei Xie and Bin Lin, and they will provide project feedback.

1.3 Terms and Definitions

Term	Definition
User	Someone who interacts with the messaging service
Stakeholder	Any person who has interaction with the system who is not a developer
Redundancy	The duplication of critical components or functions of a system with the intention of increasing reliability of the system, usually in the form of a backup or fail-safe, or to improve actual system performance
Server	A computer or computer program that manages access to a centralized resource or service in a network
GUI	The graphical user interface is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.
SDLC	Software Development Life Cycle
Socket	A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a port number.
Backend	Backend refers to the parts that do the work behind-the-scenes while the user is unaware of or cannot see. This includes databases, services, etc.

2. Test Plan Description

This section describes the scope of testing, the testing schedule and the release criteria. This section also discusses test objectives and testing strategies and methodologies used throughout development.

2.1 Scope of Testing

The scope of the test plan includes the JMClient and JMServer packages. Each package contains multiple classes that needs to be tested for proper functionality. Testing will be used to make sure messages are properly sent between the client and server application.

Testing strategies include unit testing and integration testing. JUnit, an automated unit testing framework for the Java programming language, will be used for writing and running automated tests. JUnit was chosen because it is open source and runs in most IDE's. The purpose of unit testing is to make sure critical defects and programming errors are resolved prior to integration testing. Integration testing makes sure that data flow between modules and methods is properly handled and behaving as expected.

Performance tests are not included within this document.

2.2 Testing Schedule

Date	Deliverable Name	Author	Reviewer
5/30/17	Overall Test Plan	David Kim	Bin Lin
5/31/17	Unit Test Cases	David Kim	Bin Lin
6/1/17	Unit Test Execution	David Kim	Bin Lin
6/2/17	Integration Test Cases	David Kim	Bin Lin
6/3/17	Integration Test Execution	David Kim	Bin Lin
6/8/17	Final Report	David Kim	Bin Lin

Milestone List:

5/30/17 - The Test Plan is due.

6/8/17 - The Project Report and Final Deliverables are due.

2.3 Release Criteria

Release criteria refers to the minimum conditions for the software to be deliver-ready. The following table lists the basic requirements:

Release Criteria
100% Test Scripts executed
95% pass rate of Test Scripts
No major defects or errors
All expected and actual results are recorded and documented within test scripts

2.3.1 Component Pass/Fail Criteria

Unit tests executed on components only pass when they satisfy the Object Design Specification for an individual component. This includes positive and negative tests and boundary tests. If a test fails to meet the specifications of the component, an issue will be raised and handled and resolved by the project team.

2.3.2 Integration Pass/Fail Criteria

Integration tests executed on integrated components only pass when they satisfy both the Object Design Specification as well as the System

Architecture Specification. This includes positive and negative tests, boundary tests and tests that involve the interface environment such as the physical connection to the JMServer. If a test fails to meet either specification, an issue will be raised and handled and resolved by the project team.

3. Unit Testing

This section describes the unit tests that are intended to be executed during the early stages of development. Unit testing is the base level of software testing where the smallest individual units/components of a system are tested. The purpose is to validate that each unit of the software performs as expected. Unit testing is carried out prior to integration testing.

3.1 ClientTest

This unit manages all functionality central to the client application. Functionalities include establishing and maintaining a socket connection, instantiating a client window interface, opening I/O streams and setting up the history functionality. Testing will be limited to making sure that the client application maintains basic functionality without raising runtime errors.

3.1.1 testRun()

This test case instantiates a new ClientWindow within a new Client object. It is expected to successfully run until the ClientWindow is closed. This test will fail if a new Client object cannot be created; otherwise it passes.

3.1.2 testTimeStamp()

This test case asserts that the return value is equal to the current timeStamp in the following form: 5/29/17 6:02 PM. If the method assertEquals() fails,

the test fails; otherwise it passes

3.2 ServerUnitTest

This unit manages all functionality central to the server application. Functionalities include listening for a socket connection, instantiating a server window interface and opening I/O streams. Testing will be limited to making sure that the server application maintains basic functionality without raising runtime errors while properly sending correctly formatted messages to each client.

3.2.1 testRun()

This test case instantiates a new ServerWindow within a new Server object. It is expected to successfully run until the ServerWindow is closed. This test will fail if a new Server object cannot be created; otherwise it passes.

3.2.2 testStart()

This test case starts a new Thread. It is expected to stay open until the stop() method is called on the thread, in which case the thread is sent an interrupt signal and stops the thread. This test will fail if a new thread cannot be instantiated or the newly instantiated thread does not remain open; otherwise it passes.

3.2.3 testStop()

This test case stops a running thread. It is expected to send an interrupt signal to the thread which stops the thread. This test will fail if the thread continues running after the stop() method is called; otherwise it passes.

3.2.4 testFindUserThread()

This test case finds a user thread by a String input representing the username. It is expected to return the thread of the given username. This

test will fail if the thread that's returned is not equal to the expected user thread instance; otherwise it passes.

3.2.5 testRemove()

This test case removes a client thread by an input representing the user ID. It is expected to successfully close a client thread. This test will fail if the thread that's being closed doesn't match the given user thread; otherwise it passes.

3.3 MessageUnitTest

This unit manages a message along with its content. This unit has a single accessor method which returns the formatted message content. Testing will be limited to making sure that the message properly returns the message content in the correct format. This is crucial to the system architecture because all message handlers expect input in this particular format.

3.3.1 testToString()

This test case returns the message content. It is expected to return the message content in a specific message format. This test will fail if the returned String object isn't formatted properly; otherwise it passes.

3.4 HistoryUnitTest

This unit manages the chat history functionality. Functionality that will be tested is the getTagValue() method. Testing will be limited to making sure that the client application properly extracts metadata from an external XML file.

3.4.1 testGetTagValue()

This test case returns a String representing an external XML file's tag values, which is then displayed in the historyWindow. It is expected to return

properly ordered past messages by extracting tag values from the XML file. This test will fail if the returned String object isn't formatted correctly; otherwise it passes.

4. Integration Testing

This section describes the integration tests that will be run throughout the development process. Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this type of testing is to reveal any problems or issues within the interaction between integrated units. A bottom-up approach is followed during this phase of testing.

4.1 ClientIntegrationTest

4.1.1 testClientWindowHandler()

This test case handles incoming message from the server. It is expected to direct the message to the intended recipient based on the message type. The following lists the different message types to be tested: message, login, test, newuser, register and signout. This test will fail if a message isn't properly redirected based on its type; otherwise it passes.

4.1.2 testSend()

This test case sends a message to the server. In order for this test case to pass, a JMServer must be running at the time of the test. This test case requires a dummy message to be sent to the server. This test will fail if no incoming message is received after a message is sent; otherwise it passes.

4.2 ServerIntegrationTest

4.2.1 testHandler()

This test case handles incoming message from the client. It is expected to

direct the message to the intended recipient based on the message type. The following lists the different message types to be tested: message, login, test, newuser, register and signout. This test will fail if a message isn't properly redirected based on its type; otherwise it passes.

4.2.3 testSendUserList()

This test case sends to a client an updated userlist which includes everyone currently logged into Java Messenger. A username is required to update the userlist. Then, a message with type "newuser" should be sent to each client reflecting the updated userlist. This test will fail if a newuser message isn't sent to all clients; otherwise it passes.

4.3 HistoryIntegrationTest

4.3.1 testAddMessage()

This test case saves a message to an external XML file containing a log of past messages. The addMessage() method requires a message object and a String object representing when the message was sent or received. This test will fail if a new entry isn't created in the external XML file; otherwise it passes.

4.3.2 testFillTable()

This test case displays content from an external XML file containing past messages. The fillTable() method requires a historyWindow object with a historyTable in order to display the history log. This test will fail if past messages are not properly loaded and displayed in the historyTable; otherwise it passes.