A Messaging App

Analysis and Design

Table of Contents

1 Introduction

The project that this document discusses is a project to create a messaging app. This document is the analysis and design document which will go over both the analysis and the design for the project. The document will discuss the design considerations, a system overview, an architecture of the system, and give a detailed system design. 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 and analyze the design for the messaging app, the technology stack it will be using, and technicalities to be taken under consideration in the implementation of the design. However, the document will not go into much detail about the actual implementation of the app, or nitty-gritty details that detract from presenting a coherent overview of the design.

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

- Bus Factor: The higher the bus factor, the more probable it is that a bus running over the lead developer of a project will devastate the progress of the project and cripple its ability to be completed. Often used as a slightly morbid way of determining how robust a project's state of documentation and organization is. That is, it should not be dependent on the people as that indicates a lack of necessary documentation and paperwork.
- Version Control: Version Control (VCS) is a system that stores and logs code for organization and retrieval.
- Commits: A way of adding in and storing changes to a codebase over time into a VCS database.
- 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.
- Client-Server Architecture: This is a style of designing interaction between multiple clients where many clients talk to one server.
- I/O: Input/Output is all of the input and output into and from a system.
- Asynchronous: The concept that something can happen in the background without the system paying explicit attention to it.

- Non-blocking: An activity has the ability to perform in the background and have other tasks start and finish while it runs.

2 Design Considerations

This section goes over the design considerations for the application to be developed. It has a constraints and dependencies subsection, which will go over functional and non-functional constraints and dependencies that were required for the app. This will be followed by a methodology subsection that will discuss different types of methodologies that will be used in the design as well as their justification.

To review, the project here is a messaging app that acts, in short, as a stripped down IRC client. Users log into the app and chat either globally or one-on-one or in group chats. The UI will be graphical and user-friendly in nature. In order to serve more efficiently the target audience of the app, the feature set will be kept to a minimum. Thus, the program will not deal with extensive user customization, nor arbitrary execution of user-designed scripting, macros, or aliasing of commands.

2.1 Constraints and Dependencies

2.1.1 Functional Dependency 1: UI

There must be a UI (User Interface). This UI will be user-friendly in all aspects including, but not limited to, accessibility, input agnostic, window size/shape agnostic, intuitive, easy to use, easy to read, and unobtrusive.

2.1.2 Functional Dependency 2: Client-Server Architecture

The design of the app necessitates a client-server architecture so that multiple clients can connect to one server in order to send messages back and forth.

The client itself will require authentication by a user before being able to join the server. This will necessitate account creation and storage.

2.1.3 Functional Dependency 3: Chatting

The experience will be such that the user of the application will be able to chat globally, in groups, or one-on-one.

Further, all of this chatting will be stored to disk in an appropriately safe way so that the user can seamlessly look it up again.

2.1.4 Non-Functional Dependency 1: VCS

Version control should have a sane history, with clear commits and a coherent story being told through commit messages.

2.1.5 Non-Functional Dependency 2: Documentation

Should have comprehensive documentation. The documentation should be clear enough in all aspects of the program that the bus factor of the project approaches zero.

2.1.6 Non-Functional Dependency 3

The I/O of the entire system, client and server, must be asynchronous and non-blocking. This will allow for messages to be sent simultaneously without error and will allow for concurrent chatting being done by different users on the system.

2.2 Methodology

I am a huge fan of the Actor model for messaging based systems. It has seen huge success in telephone systems and other major messaging platforms with platforms such as Erlang; it has strong potential and fits the use-case perfectly. As far as datatypes, objects, and system components go, I am a fan of thinking about the data first, behavior second, and "classes" last (if ever). Data is often intuitively hierarchical, behavior is often easily taxonmized and thus is represented well with composition, and implementation is most naturally modeled through interfaces.

3 System Overview

This section will contain a high level description of the system. This high level description will hopefully prepare the reader to understand the architecture of the system.

Firstly, the messaging experience itself will be split into two parts: the application and the server. The server will be the central hub through which all user clients connect with and interface through. The actual messaging application will be a simple user client with a GUI that connects automatically to the server.

4 System Architecture

The system-level architecture is the main architecture of the application. A short summary of the system-level architecture will be here, and an introduction to the following subsystems

4.1 Subsystem, Component, or Object 1

Provide a detailed, high level description of the first subsystem, component or object of your system. Your choice of terms should reflect the development methodology you chose in the design considerations section. You should provide a detailed description of the interface. A high level description of the internal architecture is also appropriate here. You may want to outline any important algorithms as well. Be sure to include supporting diagrams, such as UML diagrams, interaction diagrams and data flow diagrams, wherever appropriate

4.1.1 Sub-subsystem, Sub-component, or Sub-object 1-1

Some of the larger subsystems, components or objects may need to be broken down further to complete the high-level picture. The same guidelines for content apply in this section as in its parent section.

4.1.2 Sub-subsystem, Sub-component, or Sub-object 1-2

Will add

4.2 Subsystem, Component, or Object 2

Will add

4.3 Subsystem, Component, or Object 3

Will add

5 Detailed System Design

5.1 Subsystem, Component, or Object 1

Here, you will provide a highly detailed description of your subsystem, component or object. You will describe your implementation in a fair amount of detail. It is quite common to see pseudo code or even source listed in this section for portions of the object. List the data structures you will use, helper functions that you will need, and any other facet of the implementation that helps the reader understand your implementation. It is unlikely that you will need to introduce any new diagrams in this section

5.1.1 Sub-subsystem, Sub-component, or Sub-object 1-1

Some of the larger subsystems, components or objects may need to be broken down further to complete the high-level picture. The same guidelines for content apply in this section as in its parent section.

5.1.2 Sub-subsystem, Sub-component, or Sub-object 1-2

Will add

5.2 Subsystem, Component, or Object 2

Will add

5.3 Subsystem, Component, or Object 3

Will add

6 Addendum

As you can see, this document isn't quite complete yet. I had a really rough week, my hearing aid broke, and my girlfriend experienced 3 deaths of people she knew. So, not a lot got done. I will continue to revise and update this document as time goes on. I accept that this is not a valid excuse and accept the late penalties. However, in the interest of complete

documentation, some explanation is better than nothing and a half-complete document is more useful than no document.