# Final report

## Introduction

The problem is to produce a messaging app home page. This will allow users to view their messages and save/delete them, view and edit contacts, view and edit their profile and finally organize the information in separate ways e.g. alphabetically.

## Main requirements

Functional Requirements

## FR1: Upon launching the app, the landing page shall display a list of chat messages sorted from newest to oldest, the user's name, and menu options to view/edit profile, view contacts, and create a new chat.

## FR2: Each chat message on the landing page shall display the sender's name, read/unread status, and the date and time it was received.

## FR3: Users shall be able to select/open a chat message on the landing page to view its contents, reply/add to the chat, and return to the main landing page.

## FR4: Users shall have the ability to view and edit their profile details, including user ID, name, telephone number, and optional profile picture.

## FR5: Users shall be able to maintain a list of contacts, including adding, removing, or editing contact details.

## FR6: Users shall have the ability to organize contacts in different ways, such as displaying them alphabetically or by most recently added.

## FR7: Upon selecting a contact, the app shall display their profile information and list the three most recent chat messages received from them, sorted from most recent to least recent.

## FR8: Users shall have the option to search through chat messages for keywords, with the app displaying a list of chats containing the keyword(s) and allowing the user to open any chat from the search results.

## FR9: Users shall be able to delete entire chats, and optionally, individual messages within a chat.

## FR10: The app shall support the functionality to save and load all app contents, including profile information, contact details, and chat messages, from disk files.

## Structures and why they were implemented

Linked list:  
We decided to use a linked list to store the messages for each contact. Each Contact contains a MessageLog which is the linked list of Messages. We used a linked list for the messages, as they can be added to, and deleted from, easily. Unlike stacks or queues, which area Last-In-First-Out, and First-In-Last-Out respectively, meaning that messages couldn’t be deleted from inside the list, only from the start in stacks, and the end in queues.

Binary Tree:  
We decided to use a binary tree to store our contacts. This was initially because they were sorted automatically, even when a new one was added. We started by sorting the binary tree alphabetically, before switching to by the time the most recent message was sent, but because these weren’t unique, we eventually decided to keep our binary tree, but used an ID as a unique identifier. This meant that we couldn’t sort the contacts alphabetically, or by most recent message sent, and complicated some algorithms, but worked out in the end.

## Classes

* Contact – Contains the contact information.
* ContactButton - The contact button class is used to create a button containing information about contacts and the most recently sent messages with them.
* DeleteButton - Class for button next to messages in messageArea which deletes that message when clicked.
* LandingPage - Landing page for the application.
* LandingPageMainPanel – Class that creates a JPanel with a name.
* Message – used to store information about the messages.
* MessageLog – Linked list of messages between the user and a contact.
* MessagePage - Class representing the message page of the program.
* MessagePanel – The message panel, containing a name.
* Profile – The class used to store the user’s profile information.
* Tree – Binary tree of contacts.

## Class Diagram

## Test Plan

## Design

We Want a GUI based design. Which allows a user to see the entire list/most of the chats at one time. They will be able to select one which will show the messages. At the top of the chat, it will provide the user with relevant information about the contact e.g. name, favourite.

|  |  |  |
| --- | --- | --- |
| Task | Lead Contributor | Regular Contributor |
| GUI implementation | Sam | Jonah, Harrison |
| GUI design | Sam | Harrison |
| Documentation | All |  |
| Binary Trees | Harrison | Sam |
| Linked list | Sam, Harrison | Jonah |
| Class Design | Sam | Jonah |

## Task Allocation

## Self-evaluation

We feel like our project went well. Initially we had four members, with the fourth being Ruairidh Munro, but after getting in touch with him and adding him to our group chat, he became unresponsive, and didn’t make any input into our project. Down a team member, we decided to strip back slightly on some of our ambitions for the project. This included not implementing a “favourite contact” system or being able to sort the contacts in different ways. Despite this, we worked well as a team, Sam took on more of a leadership role, assisting in all aspects of the project. He especially focused on the GUI and the interconnection of the data structures. Harrison focused on the implementation of the data structures, particularly the binary tree. Jonah floated around, making regular contributions to all aspects of the project. We had near daily meetings, at Sam’s flat, where we worked on the project together. We also went to the library regularly to avoid distractions and focus solely on our project. We created a GitHub repository, to help manage the sharing of files, and we all used IntelliJ IDEA as our IDE, for consistency. We also communicated in a Snapchat and a discord group chat. Overall, we feel like the project went very well, especially considering that we were down a team member. Our data structures are implemented fully and correctly, and we explored many bonus requirements.