# Final report

## Introduction

The problem we were given was to create a messaging application using our knowledge of data structures in java. The program had to allow users to view, edit and delete their contacts, as well as show messages with each contact. The program also had to include a user profile that could be edited.

## 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.

We were able to complete requirements 1 through 5 successfully. We tried to do requirement 6 but had problems with the sorting, so couldn’t complete it. Requirement 7, we developed further that the original brief, showing all the messages with a given contact, instead of the most recent three. Requirement 8 we were able to do successfully, allowing the user to search for a keyword, and be given a list of used of that word, which upon clicking, took the user to chat where the word was used. We half-completed requirement 9, by letting the user delete individual chats, but not able to delete entire chats, unless by deleting the messages one at a time. Requirement 10 we were able to do fully, by automatically saving/loading to/from a folder, that is created by default when first closing the application. This takes away the saving/loading from the user, and does it automatically, when opening and closing the program, which brings a more seamless experience.

## Design

We aim to create a GUI based design, which will allow 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.

## 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.

## A computer screen with blue squares Description automatically generatedClass Diagram

Class diagram was created automatically by Visual Paradigm.

## Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test Description | Test Data | Expected Result | Worked? |
| Add a new contact to the tree | Name: “Brian”, Number: “07123456789” | New Contact added to correct spot in tree with name Brian and number 07123456789 | Y |
| Delete existing contact from tree | Name: “Jonah” | Contact with name “Jonah” should be deleted from the tree and the other nodes should fill in respectively | Y |
| Attempt to delete contact with invalid name | Name: “Enter Message Here…” | No contact removed | Y |
| Search for message word that doesn’t exist | Search term: “SAM IS AWESOME” | No search results appear | Y |
| Search for message word that does exist | Search term: “Hello” | All instances of “Hello” are displayed | Y |
| Change Name and number of Profile | Name: “Sam Watts”  Number: “07936011901” | Name on profile changed to Sam Watts and number changed to 07936011901 | Y |
| Add Profile picture of User’s Choosing | Bear Image | Profile picture will change from blank profile to bear | Y |
| Delete message from a chat | “Goodbye Ruairidh” Message deleted from chat | Message will be deleted from the chat and not appear on further openings of the chat (including when program closed) | Y |
| Adding new message to chat | “Good Morning Harrison” message sent on chat | Message will appear on the chat window and be present when reopening the chats (including when program closed) | Y |

|  |  |  |
| --- | --- | --- |
| 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 |
| Image creation | Sam |  |
| Testing | Jonah | Sam |
| File Handling | Sam | Jonah |

## Task Allocation

## Project Evaluation

During the development of our project, we encountered many barriers. One of which was when trying to get the contact buttons to update the contents of the message area. We eventually found a solution, by removing the existing message area. We were able to make a new one, with the updated contact. This remove proved useful throughout the project, as whenever we wanted to add or delete, we called it, so a new component could be added. One problem we couldn’t fix, was when adding a new contact to the tree, the GUI had some graphical problems, causing a line to appear down the centre of the contact buttons. We found that going back to the landing page fixed the problem, but it wasn’t a permanent solution. One problem we had was when saving to a file, if the files folder, which is where the information was saved, didn’t exist, an error would be thrown. This was solved by creating a new folder, if one didn’t exist. There were many problems with the layout of the pages. These were mostly all solved by trial and error, using all of the Java Swing layout managers, in different configurations, until one worked. One other problem we had was when we tried to convert the binary tree into an array, we came up with a solution, that recursively saved the left, current and right nodes into separate arrays, then combined them using a premade java array combining function.

Meeting Matrix

During the completion of this project we had far too many meetings to recount. From spending lab sessions huddled around a desk to lounging with our laptops on our knees in Sam’s flat kitchen we have put in many hours towards this project. In this table we will try to give a brief idea of the kinds of things we did in some meetings (dates may not be completely accurate)

|  |  |  |
| --- | --- | --- |
| Date | Aim | Occurred |
| 22nd March | Implement the Deletion method into GUI | Deletion Method was completed and began to create a button to call it in the GUI |
| 18th March | Create the view/edit profile method | Beginning of view/edit profile created, just need to add in the profile picture section |
| 14th March | Create Text Messages to display on Message Frame outside Landing Page Frame | Default messages created in separate frame callable by buttons on landing page |
| 8th March | Begin planning for project | Interim report completed with plan for project workload etc. |

## 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 in an effort to make up for our failed attempts at some base requirements.