SET09102 Software Engineering Coursework

Software Development Report for Napier Bank Message Filtering System

Mansour Sami

Overview

In this report, I described the software development efforts to create Napier Bank Message Filtering System as a partial fulfillment of Software Engineering course. In section 1, requirement specification is explained. Class diagram is presented in section 2. In section 3, GUIs used throughout the project are shown. Although additional requirements were explained in the requirement specification stage, in Section 4 we summarized them for clarity. Testing is comprehensively covered in Section 5. Version control and evolution strategy are explained in sections 6 and 7. Based on project description and after requirement specifications, Star UML application was used to create the Class Diagram and the Use Case Diagram. The programming language Java (openjdk-19 (java version 19.0.1)) is used. For GUI the JavaFx library is used. The libraries used in the project are the standard Java library, standard JavaFx library, and Lombok library. Finally, to make the GUIs (FXML files), I used scene builder.

1. Requirement Specification:

Two forms of requirement specifications were used in this project. First, the List of functionalities and secondly the Use Case diagram

List of functionalities derived from project description are:

1. Handling the Message types SMS, Email, Significant Incident Report, and Tweet stored in a single stored text file.
2. Detecting message types.
3. Making a dictionary from TextSpeak.CSV file.
4. Detecting Text to speak abbreviations in the SMS and Tweet and adding the long format to the text output while the input is not changed.
5. Detecting the URLs in the text of the E-mail and Significant incident report.
6. Quarantining the detected URLs.
7. Detecting Hashtags in the Tweets.
8. Creating a Hashtag List out of the used Hashtags in the Tweet.
9. Making a Trending list that is based on the Hashtags and the number of times that were been used in a Tweet.
10. Detecting all the IDs in the Tweets text.
11. Adding all the IDs to a list called Mentions.
12. **Additional requirement**: Reading input from another format except for JSON format.
13. Output the messages in JSON format.
14. Saving the messages that were transformed to a JSON format in the output file.
15. Taking testing message as a normal message.
16. Making Testing Input.
17. User Interface for user and system to interact with each other.

18. Users can give testing message data throughout the UI.
19. Messages are displayed with the UI.
20. Testing Messages are redisplayed by the UI.
21. UI shows the SIR list.
22. UI shows the Mention List.
23. UI shows the trending list.

How Each Function Works:

1. Each message type has its special function and format within the classes, that allows the system to put the inputted message in the correct format.
2. The system can detect messages by its header in classes like Email and SIR. The system reads and depending on the format decides whether it is Email or SIR.
3. The system opens the TextSpeak.CSV file and reads it line by line and makes a dictionary where keys are the short format and the long formats are the values.
4. The system reads all the SMS text and Tweet Text and checks if the word exists in the TextSpeak dictionary and if exists we add the long format to the text next to the short format inside the <>. E.g. Input: “Short format”, Output: “Short format<Long format>”
5. The system reads the text of E-mail and SIR and if there was a string with the pattern of an URL we define it as an URL.
6. The system takes the detected URLs in the message and replaces them with the “<URLs Quarantined>” and then we add the URL to a list that we save the quarantined URLs for each E-mail.
7. The system reads all the Tweet texts and if there is a hashtag (starts with # and then contains characters) it will detect it by its format
8. The system makes a Hashtag list for that Tweet from the hashtags that were in the text.
9. A trending list is a list that contains all the hashtags that were used in all the tweets and saves the number of times that those hashtags were used.
10. We read the Tweet text and if there was a word with the format of the IDs we define it as an ID (ID format starts with an ‘@’ and is followed with characters).
11. Adding all the IDs of a Tweet to a list called Mentions.
12. Additional Requirement: The system takes the input to form a CSV file the format of messages is different from the JSON format.
13. The system transforms message by its defined function to a JSON format. (Java has a library for it but we use the handmade one. Compared to the library version, the library version takes more space and takes more time to do but in the handmade version each MessageToJson function has its special design for its own message time and each function has differences from the other one in the other message type.)
14. All the messages are saved in the output format after they are transformed into the JSON format.
15. Users can make a testing input in each of the 4 types of messages in the system.
16. The system processes the test message as a normal message and does all the processing on it.

17. The user interface is designed so that the user and the system can interact with each other through it.
18. The user sends the data of the testing input via the UI to the system.
19. All the messages after they are processed, are displayed to the user in their detected format.
20. All the testing messages are displayed like a normal message.
21. UI displays all the data inside of the SirList to the User.
22. UI displays all the data in the MetionsList to the User.
23. UI displays all the Hashtags and the number of times they were used by the User.

Use Case Diagram:

High level use case diagram of the system is shown in Figure 1. The system can get inputs from the input text file (Input Database actor) and then reads each message, detects each message type, and transforms it to the respective format. That is, the message is transformed into the JSON format based on the description stated in the above section and assignment sheet. The system saves the JSON format into the output file and shows the message based on the type of message into the relevant GUI to the user. It is possible to make a testing input inside the system and gave it to the system for doing the same stages as the normal message. In Figure 2, the use case is more detailed and included all the specialized message type modifications that is requested to give a more detailed view of the system.

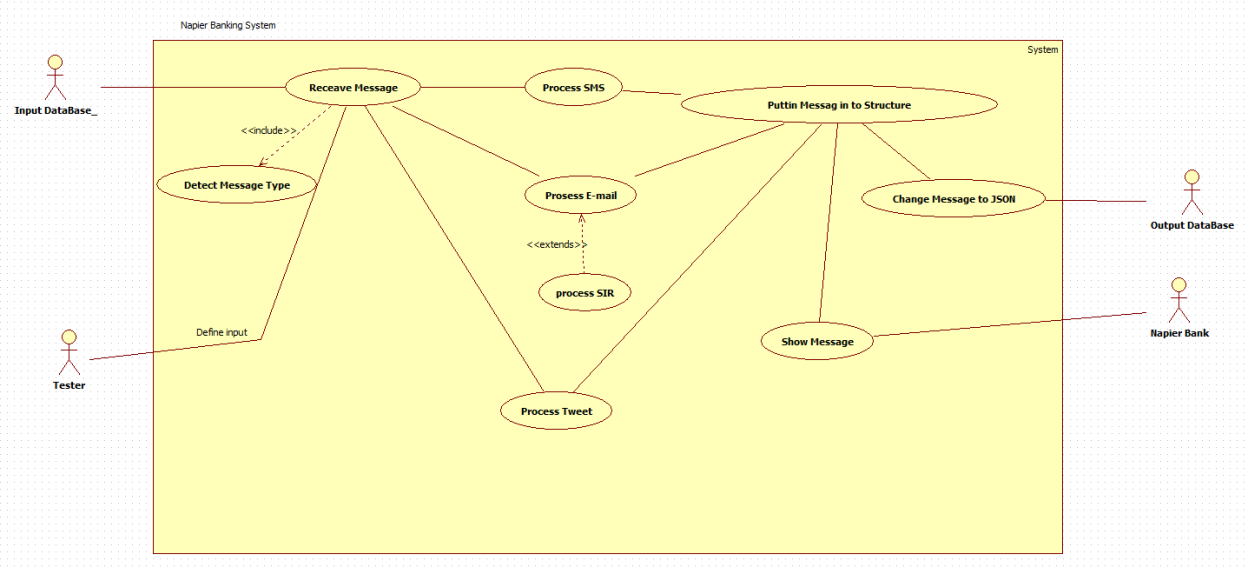

FIGURE 1 HIGH LEVEL USE CASE DIAGRAM OF THE SYSTEM

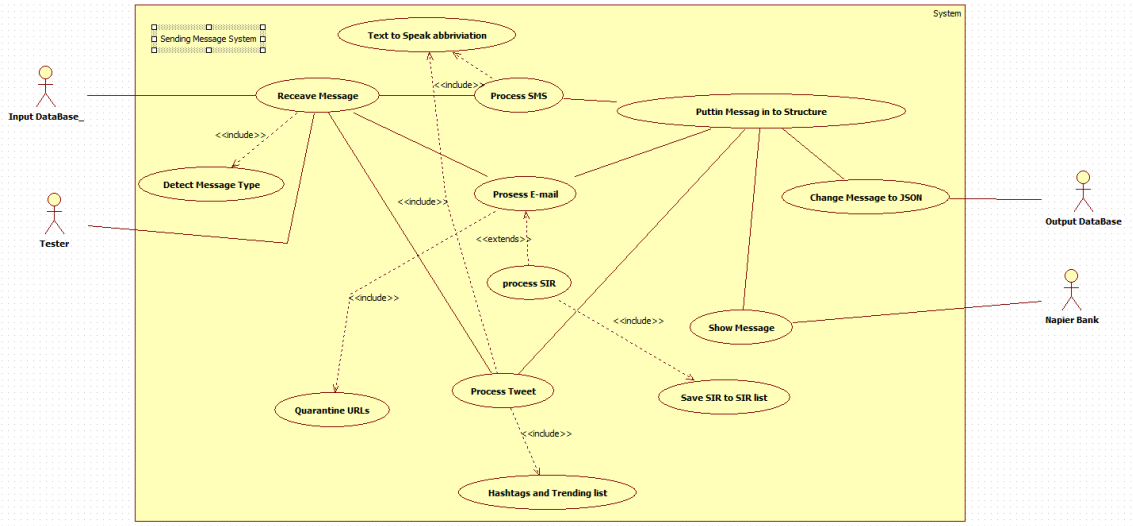

FIGURE 2 SPECIFIED VERSION OF THE USE CASE DIAGRAM

2. Class Diagram

The Class Diagram is shown in Figure 3 and the relationships among different classes and control classes for GUI are also included in the diagram.

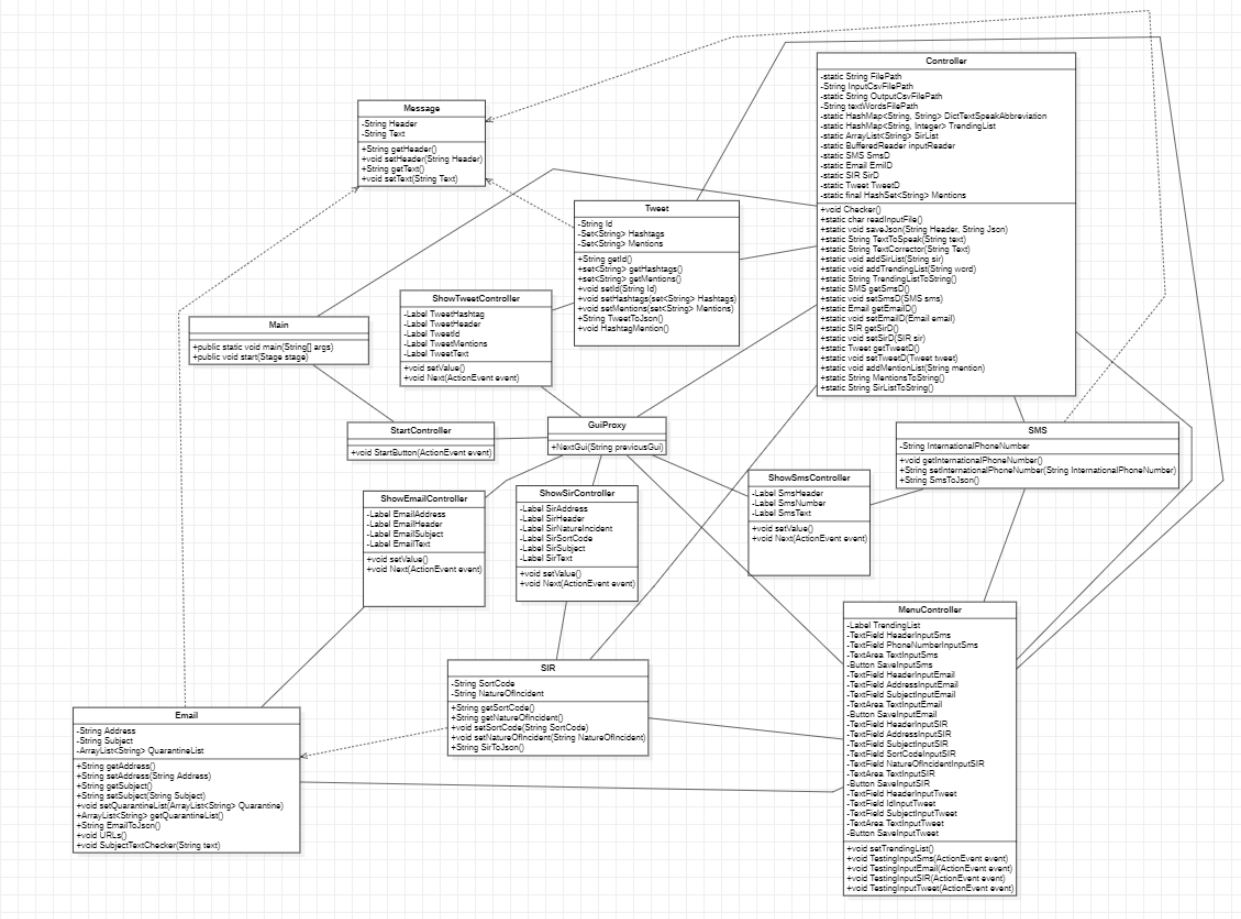

FIGURE 3 THE CLASS DIAGRAM OF THE WHOLE SYSTEM

3. GUI

Here we present NBS the used GUIs and say their functionality. To save space, I have included the final operations system view of the GUIs. The system starts with GUI shown in Figure 4. Sample output GUIs for each message is also shown in Figures 5 to 8. In Figure 5, the output for an SMS. Sample email message output is shown in Figure 6. Figure 7 is the output of SIR. Finally, Tweet output is shown in Figure 8.

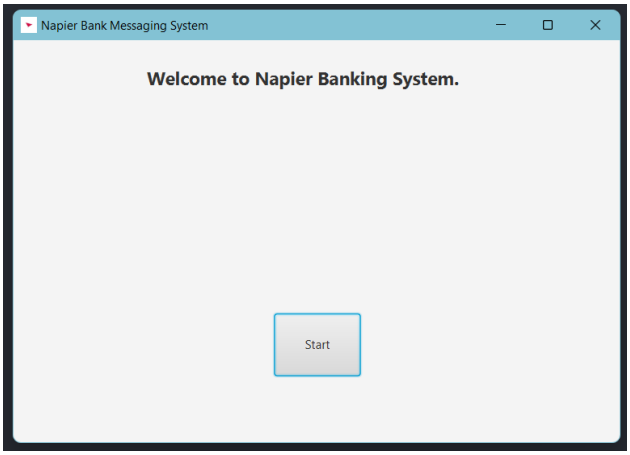

FIGURE 4 START.FXML

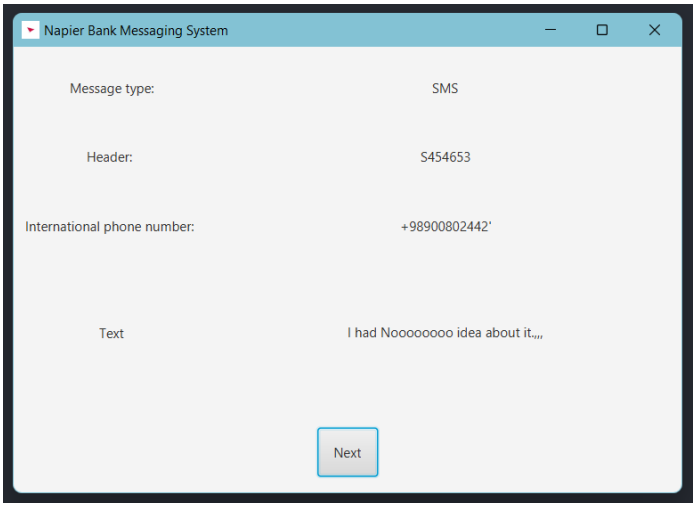

FIGURE 5 SHOWSMS.FXML FOR SHOWING THE SMS MESSAGE TYPES

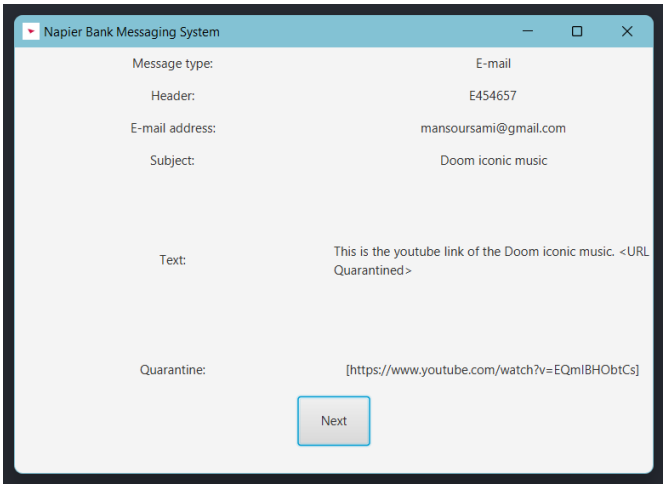

FIGURE 6 SHOWEMAIL.FXML FOR SHOWING THE STANDARD E-MAIL MESSAGE TYPE

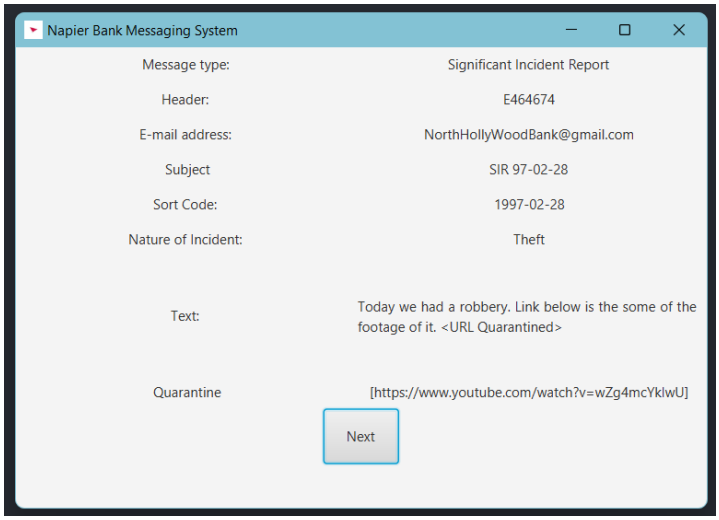

FIGURE 7 SHOWSIR.FXML FOR SHOWING SIGNIFICANT INCIDENT REPORT E-MAIL MESSAGE TYPE

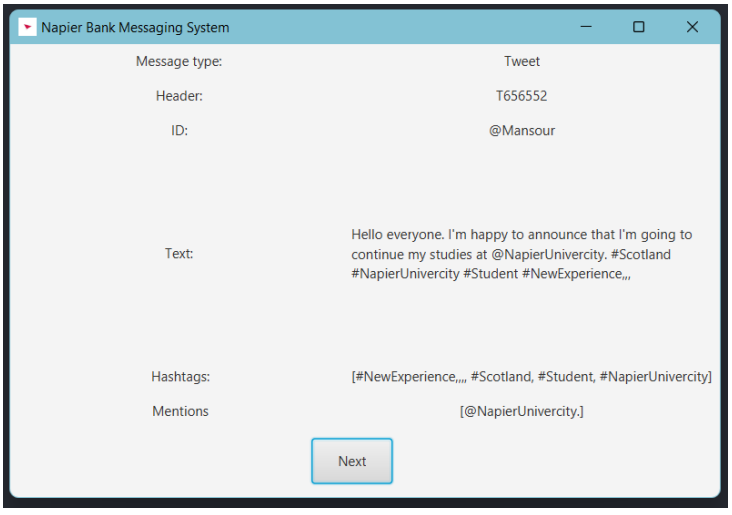

FIGURE 8 SHOWTWEET.FXML FOR SHOWING TWEET MESSAGE TYPE

Figure 9 is the welcome GUI. Sample input GUIs for each message is also shown in Figures 10 to 13. In Figure 10, the output for an SMS. Sample email message output is shown in Figure 11. Figure 12 is the output of SIR. Finally, Tweet output is shown in Figure 13. Finally, figure 14 shows the SIR list, mentions list and trending list.

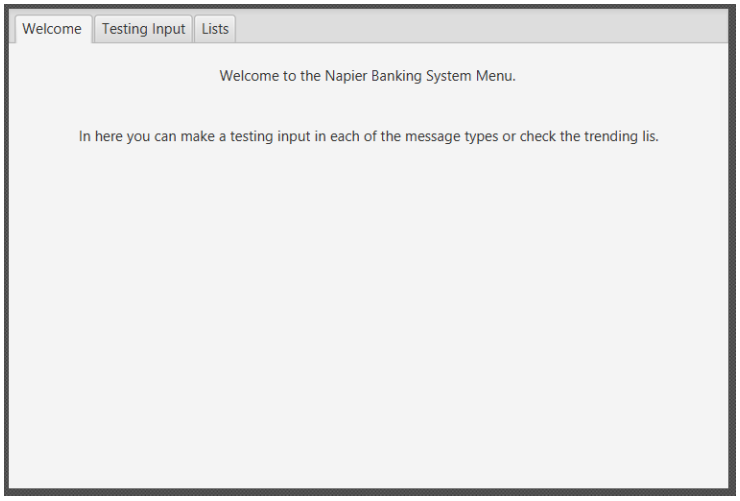

FIGURE 9 WELCOME TAB IN MENU.FXML

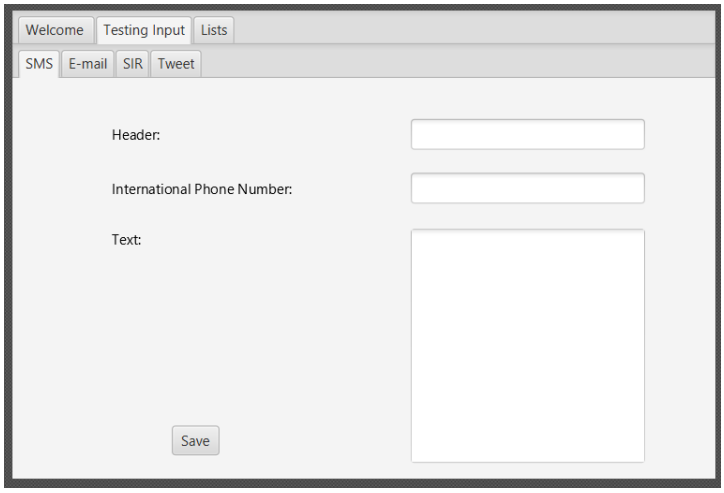

FIGURE 10 SMS TAB IN TESTING INPUT TAB IN MENU.FXML FOR MAKING SMS TESTING MESSAGE

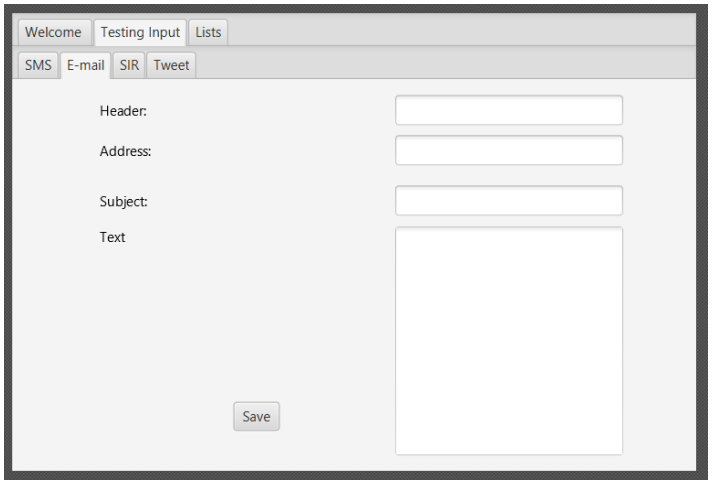

FIGURE 11 E-MAIL TAB IN TESTING INPUT TAB IN MENU.FXML FOR MAKING E-MAIL TESTING MESSAGE

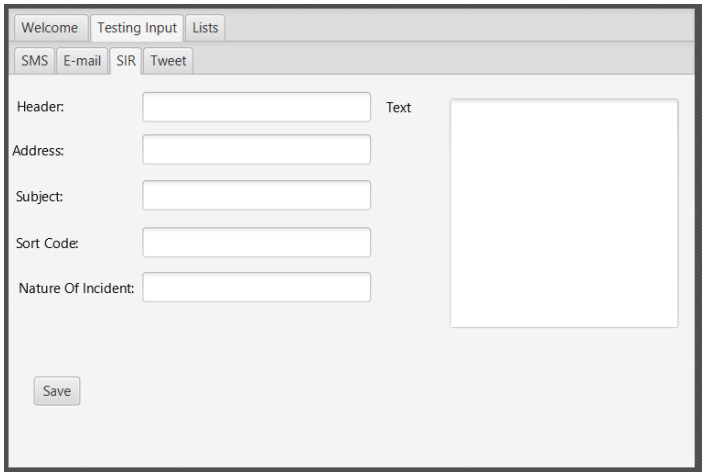

FIGURE 12 SIR TAB IN TESTING INPUT TAB IN MENU.FXML FOR MAKING SIGNIFICANT INCIDENT REPORT TESTING MESSAGE

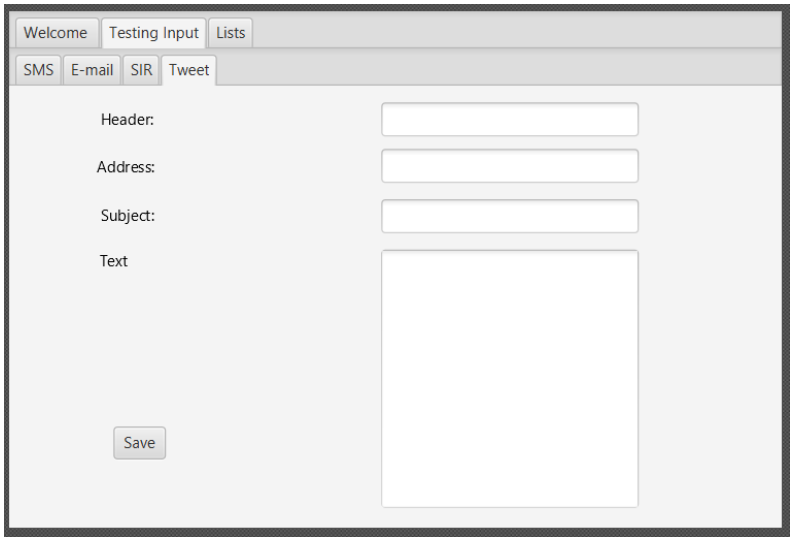

FIGURE 13 TWEET TAB IN TESTING INPUT MENU.FXML FOR MAKING TWEET TESTING MESSAGE

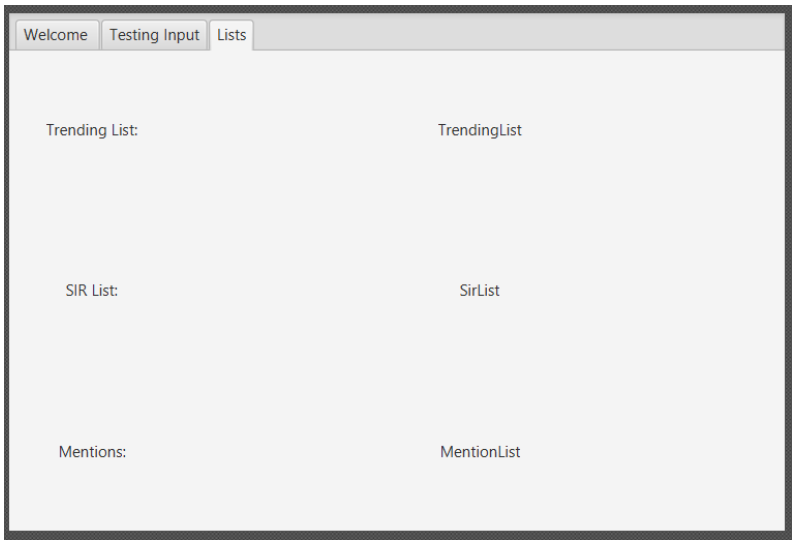

FIGURE 14 LISTS TAB IN MENU.FXML FOR SHOWING THE TRENDING LIST, SIR LIST AND MENTIONS

4. Additional Requirements

The input file based on our system design is reading from text files that are stored and the output is in JSON format. The csv input format is used.

5. Testing:

The type of testing is defect testing. The main strategy used for this program is the typical testing process that the level of working is bottom up. However, the tests are designed based on requirements and can be traced back to our requirements. All of the test cases were designed by looking inside the box and knowing how the system works and the goal was to trigger all the functionalities and reveal as many bugs as possible.

Overall strategy:

Unit testing => Integration testing => System testing => Validation testing => Acceptance testing => Regression testing

I. Unit testing: In this part, we wrote each function and test it depending on its functionality to reach the limits and see the system reaction throughout all the possibilities possible for that function.
II. Integration testing: Testing each part of the system as one. E.g. testing to see if the system can get SMS then putting the value in the right class and values, then doing the text-to-speak abbreviation on the text and returning a JSON made from the SMS. For this part, each message type was tested with and without its processing and each processing function is used in different situations.

III. System testing: At this level, we are testing the whole system to see how the system behaves as one. For this testing, we make some sample input files for the system to process aside from the testing inputs.
IV. Validation testing: Validation testing is the only test that I went through during the whole process and each time after each function is written is done. But aside it was done alongside the testing input after the system testing I did validation testing for the whole system.
V. Acceptance testing: Because I was the only user to test the program, I put myself as a User, and by using the given PDF for the requirements, I have done the testing.
VI. Regression testing: After going through the process of testing we have a regression testing we have gone through the whole process of testing again to make sure there is no bug or problem in the whole system.

Test Plan:

To make sure each requirement has been implemented and works properly I have designed tests for each functionality that I have mentioned. Tests for the following scenarios were implemented.

1. Each message type has its special function and format within the class that allows the system to easily put the inputted message in the right format. So we design different message types to trigger each message function and to test them if they work correctly or not.
2. The system can detect messages by its header in classes like Email and SIR, the system reads the message and depending on the format of the message decides whether it is Email or SIR. Give the system different message types. If we gave SMS, Twitter, SIR, SMS, and Email in a line the output should be SMS, Twitter, SIR, SMS, and Email.
3. The system opens the TextSpeak.CSV file and reads it line by line and makes a dictionary where keys are the short format and the long formats are the values. By making the dictionary and testing it by giving the different short forms and then by getting the long formats and checking them we know if the system has bugs or not. E.g. Input: “GMTA”, Output: “Great minds think alike”
4. All the SMS text and Tweet Text were read and checked if the word exists in the TextSpeak dictionary and if existed, the long format were added to the text next to the short format inside the <>. E.g. Input: “Short format”, Output: “Short format<Long format>”. We test it by making SMS messages containing the text speak and checking if it correctly changed the text or not. E.g. Input: “CYA”, Output: “CYA<See ya>”.
5. The system reads the text of E-mail and SIR and if there was a string with the pattern of an URL we define it as an URL. We check it by giving it different inputs that some contain URLs and some do not and we check how the system reacts through the inputs.
6. The system takes the detected URLs in the message and replaces them with the “<URLs Quarantined>” and then we add the URL to a list that we save the quarantined URLs for each E-mail. We gave Email and SIR inputs that have URLs and some that do not and we check if in the output those which have URLs

are changed and those which do not contain URLs did not change and we check the quarantined list if they changed or not depending on the input. E.g. Input: Email Text == "https://www.amazon.co.uk/ref=nav_logo", Output: "<URLs Quarantined>" QuarantineList = "https://www.amazon.co.uk/ref=nav_logo".

7. The system reads all the Tweet texts and if there is a hashtag (starts with # and then contains characters) it will detect it by its format. For testing, we gave different inputs some contained hashtags and some do not and we write the system in a way after each tweet printed all the hashtags, and then by the output, we check if the program is working correctly or not. E.g. Input: Tweet Text == "#ComputerScienceStudent", Output: "#ComputerScienceStudent",
8. The system makes a Hashtag list for that Tweet from the hashtags that were in the text. The test of this part is similar to the previous part.
9. A trending list is a list that contains all the hashtags that were used in all the tweets and saves the number of times that those hashtags were used. For testing this part we only need to input different messages and some tweets that contain different hashtags and check if the trending list is correct or not.
10. We read the Tweet text and if there was a word with the format of the IDs we define it as an ID (ID format starts with an '@' and is followed with characters). We gave Tweets with some containing IDs and then check if the outputs are correct are not.
11. Adding all the IDs of a Tweet to a list called Mentions.
12. Additional Requirement: The system takes the input to form an .CSV file the format of messages is different from the JSON format. We need to check if we can read from the CSV file correctly and test if the system can read the message correctly or not.
13. The system transforms messages by its defined function to JSON format. (Java has a library for it but we use the handmade one. Compared to the library version, the library version takes more space and takes more time to do but in the handmade version each MessageToJson function has its special design for its own message time and each function has differences from the other one in the other message type.) When we write the functions of each message type transform to JSON we check it with the output. E.g. Input: SMS Header == "S9543" PhoneNumber == "+448248545" Text == "Hey Brother. Call me ASAP!", Output: "{Header: S9543, PhoneNumber: "+448248545, Text: Hey Brother. Call me ASAP<As Soon As Possible>!}"
14. All the messages are saved in the output format after they are transformed into the JSON format. To test this part, we gave different types of the message as input and then check the output file that is all the JSON formats are correct are not.
15. Users can make a testing input in each of the 4 types of messages in the system.
16. The system processes the test message as a normal message and does all the processing on it. We test it by checking if the test message is processed correctly or not and if we are sure of our functions we check if they are the same or not.
17. The user interface is designed so that the user and the system can interact with each other through it. Test it by checking if all the inputs are gotten correctly and if it is possible to do all the wanted tasks.

18. The user sends the data of the testing input via the UI to the system. Testing it with if all the inputs are correctly inputted or not.
19. All the messages after they are processed, are displayed to the user in their detected format. We test it by checking if the text it is going to display is the same as the processed output or not.
20. All the testing messages are displayed like a normal messages. We test it to see if there is a difference or not.
21. UI displays all the data inside of the SirList to the User. We check if it is displayed or not and if it is correct or not.
22. UI displays all the data in the MetionsList to the User. The test is the same as the previous function.
23. UI displays all the Hashtags and the number of times they were used by the User. The test is the same as the previous function.

Sample Test Case:

The test case below is one of the test cases designed for system testing that can trigger almost all of the functions. For triggering the testing input you should give the data by hand. (This test case is used for making the Visual Demo)

S485454653,+98900802442',I had Noooooooo idea about it.,,,
E162454657,mansoursami@gmail.com,Doom iconic music,This is the youtube link of the Doom iconic music. https://www.youtube.com/watch?v=EQmIBHObtCs,,
E468254674,NorthHollyWoodBank@gmail.com,SIR 97-02-28,1997-02-28,Theft,Today we had a robbery. Link below is the some of the footage of it.
https://www.youtube.com/watch?v=wZg4mcYkIwU
T656543252,@Mansour,Hello everyone. I'm happy to announce that I'm going to continue my studies at @NapierUnivercity. #Scotland #NapierUnivercity #Student #NewExperience,,,
S454853723,+44645582542',"Hey man, HRU, Call me ASAP",,,
E566417713,QatarAirWays@gmail.com,SIR 14-08-05,2014-08-05,BombThreat,The news about the bomb threat is now online (https://www.youtube.com/watch?v=j4KJkFf9CAE). We do not know what is exactly happening inside.
E458443865,auto-confirm@amazon.co.uk,Your Amazon.co.uk order.,"Hello Thanks for your order. We'll let you know once your item(s) have dispatched. Your estimated delivery date is indicated below. You can view the status of your order or make changes to it by visiting Your Orders on Amazon.co.uk.",,
T656543282,@elonmusk,What do you think of the culture war?,,,
T781235933,@BigMikeJ73,"""Take advantage ,do your best, Don't stress, you was granted everything inside this planet""",,,

With this, we can trigger all the functionalities and one of the unique points of this test case is that it's more like an actual message not randomly generated (All the links are real and the Significant incident report messages were designed based on actual events and all of the tweets except the first one are actual tweet that you can find in the Twitter platform. The second Email is the actual email of Amazon).

6. Version Control Plan:

The program has 3 main versions. First version is purely depending on the class diagram that was designed for the system without the GUI. Second version is where we add the GUIs to the program and add Controller Classes to the program and Class Diagram. The third version is after when we have revision testing and this lead to the hugest impact to the Class diagram.

I used local repository for this project because it is easier to work with, it is easier to access to it and lastly because I was working solo.

I designed the system based two different ways the user is supposed to use the system. The first way is we make the functions and work with them by the program terminal. In the second line, we make the GUI and make the controllers. Not to mention that several projects are the sub-functions or functions and the only reason that those exists are for writing the function and testing it on its own and without needing the function to interact with other functions. After both paths are finished we start to match the GUI to the functions and do the test to make sure everything is working properly.

The first step is to design the Use Case Diagram based on understanding the requirements. After that we design the class diagram for the system, the main differences it has from the original class diagram aside from the class and design have some differences it is that was designed to work and communicate through the terminal, not the GUI. After having a solid and stable class diagram we start to make the program. In the first, we have 2 different path that works separately from each other. The first path is that we make the functions and classes without the GUI and only communicate through the terminal. The second path contains the GUI design and how it is going to communicate with the user and the program.

In the first path, we make the classes and the functions work and how the system is going to communicate with each part. By testing these parts and going through the whole testing process we reach a point where we have a solid program, not to mention that the class diagram will evolve throughout the process.

The second step is about the GUI design. There are questions we should answer. The questions are, in which way we should design it to fulfill the requirements, and can the user use that easily or not? After we answer these questions and designed our GUI we start to make each GUIs controller, so when we have to connect them to the project we have fewer problems.

After finishing the two paths we combine the two paths into one by combining the system and the GUIs. In this process, the class diagram will evolve even more because we are adding the controller classes and each GUI has its unique controller class.

When we are finished by testing the programs and going through the testing, we start the regression testing. At this level the difference is now we are not communicating through the terminal, we are communicating through the GUIs (the UI).

After finishing the test, we start to check our Class diagram and the Use Case diagram to make sure that they are not separated from the system and that they can give exact data about the program itself.

7. Evolution Strategy:

In evolution, we have three types of activities: bug fixing, modifying software to work in a new environment and implementing new or changed requirements. To ease bug fixing, I have refactored my code so it is more readable. In addition, I tried to implement a class diagram with few number dependencies. To work in a new environment, I used Java instead of C# since I thought Java can be easier to implement in Linux or Unix systems than C#. Moreover, Java has proven its veracity among different machines and environments. Finally, to add functionality or change requirements, I have used *Proxy design pattern* that is based on the Open-Closed Principle (OPC). that is, the modules should be open for extension but closed for modifications. Assuming we will have a new multimedia message format that we will need in future to add the existing system. In this case, because of the design this would be simple to add functionality. The design requires only adding a concrete 'Multimedia-Message' class and if the attributes of that message type are different it needs another FXML file displaying the message for the user. There should be editing in the Controller class and the GuiProxy class. For the testing input, we need to add a tab to the testing tab in the Menu.FXML file for getting the testing input.