**Software Engineering Coursework**

**Napier Bank Message Filtering**

**Guilherme Almeida – Napier BEng Software Engineering**

# Requirement Specification

## Brief

I’ve been hired by Napier Bank, a medium-sized local bank that has thousands of users. My job is to create a software, named Napier Bank Messaging (NBM), which will validate, sanitise, and filter all incoming messages to Napier Bank, the messages are in form of SMS, emails (regular emails and SIR emails) and Tweets.

## Project Requirements

This software must feature one main page which will allow the user to input any type of message in two different text boxes, one for the message ID, which consists in a single capital letter (S, for SMS; T, for Tweets; E, for Email), and one for the Message body. The software should then be able to identify the type of message according to the ID, for SIR emails, it will identify according to the subject. The software should then filter and organize the inputted message and output it into a JSON file. The software should be able to identify website links in emails and quarantine them into a list, identify mentions and hashtags, creating a trend list with the hashtags at the end of the session, and create a list of the SIR reports.

## Functional Requirements

|  |  |
| --- | --- |
| Requirement ID | Requirement |
| 1.1 | Input Message ID |
| 1.2 | Input Message Body |
| 2.1 | Filter Message |
| 2.2 | End session |

### Functional requirement description

* 1. Input Message ID

Description: User inputs message ID in appropriate text box

* 1. Input Message Body

Description: User inputs Message Body in appropriate text box

2.1 Filter Message

Description: User clicks Filter button and the software will display a success or error message, informing the user of any changes he must make to the input in order for the software to work.

Constraints: Message Id must feature either S T or E as the first character. ID shouldn’t be bigger than 10 characters. Input text boxes can’t be empty.

2.2 End Session

Description: User clicks End session button and the above-mentioned lists are displayed.

## Non-Functional Requirements

## Resource Requirements

## 

|  |
| --- |
| Resources |
| Visual studio |
| Monitor |
| Keyboard |
| Mouse |
| Computer |

## Use Case Diagram

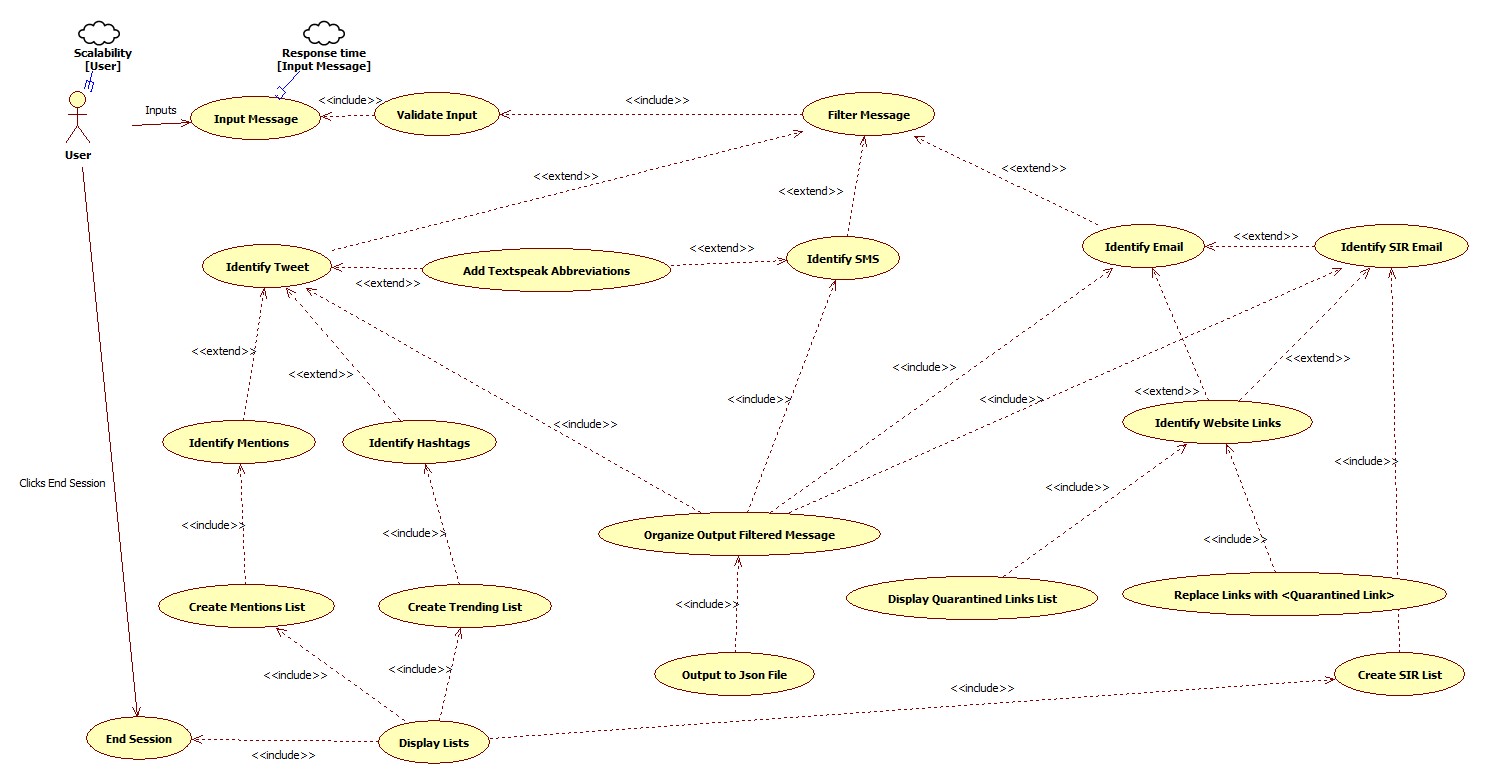


Fig.1 - refer to Documents folder in order to have a better clear view of the diagram)

## Class Diagram

Graphical user interface, application

Description automatically generated

# Testing

## Test plan

### Objectives

The objective of this testing is to confirm that the application meets its functional requirements, previously discussed in the planning phase, this will be done by checking if there are any errors in the functioning of the app.

### Scope

The scope of this testing is to verify the functionality of the back-end, the usability of the user interface and user inputs and outputs of the system.

### Test Items

* Functional Acceptance Test Log
* Test Plan

### Testing Methods

The method used for this testing will be **white box testing** which consists of reading and understanding the code to see if we can find any errors and then running the program with specific data to check if we get the expected result, this method is to be used by people with knowledge of the code being tested.

### Testing Levels

The level of testing will be **Acceptance Testing** where we will be using different inputs to test if the expected output is achieved

### Testing Strategy

The testing strategy used will be Top-Down strategy, this is a strategy where the main modules will be tested first which will give us an early visualization of the final application functionality, only then we move to the less important modules of the application, this way, we can make sure that the foundation of the application is working before going into further details.

### Environmental Needs

|  |  |
| --- | --- |
| **Hardware** | **Software** |
| PC- minimum hardware requirements | Visual Studio |
|  | Word |

### Features to be tested

* Message Filtering and Re-organize
* Mentions, SIR, websites and trending lists creation and display
* Filtered and re-organized message output to .Json file
* Buttons Functionalities

## Functional Acceptance Test Log

|  |  |  |
| --- | --- | --- |
| Input | Expected Result | Output |
| ID: E123456789  Message: [test@gmail.com](mailto:test@gmail.com) Shopping List Today. This is the shopping list for today | [test@gmail.com](mailto:test@gmail.com)  Shopping List Today.  This is the shopping list for today | As expected |
| ID: E123456789 Message: [testgmail.com](mailto:test@gmail.com) Shopping List Today. This is the shopping list for today | Error, with suitable message box displayed. | As expected |
| ID: E123 Message: [testgmail.com](mailto:test@gmail.com) Shopping List Today. This is the shopping list for today | Error, with suitable message box displayed. | As expected |
| ID: T123456789  Message: @testuser Hi, this is a test LOL @softwaredevelopment #testing | @testuser Hi, this is a test LOL <Laughing Out Loud> | As expected |
| ID: S123456789 Message: +4412345678910 Hello this is a test sms LOL | +4412345678910  Hello this is a test sms LOL <Laughing Out Loud> | As expected |
| ID: E123456798 Message:  [test@gmail.com](mailto:test@gmail.com) SIR 20/12/2022 Sort Code: 23-43-54 Nature of Incident: Theft  There has been an incident last week. | [test@gmail.com](mailto:test@gmail.com)  SIR 20/12/2022  Sort Code: 23-43-54  Nature of Incident: Theft  There has been an incident last week. | As expected |
| End session | Displays the different lists | As expected |

# Version Control

In order to keep software development smooth and efficient, version control need to be used so that different software developers can work together. If no version is being used, what may happen is that different developers will be working on the same code at the same time, then needing to manually merge the two different versions, which normally causes bugs or breaks the code.

Version control is a software package that will analyse, record and allow you to tag the changes made to the code so that anyone who looks at it knows exactly what they are working with, and what changes were made.

There are 3 types of version control systems

* Local
* Centralised
* Distributed

The key concepts in Version Control that need to be thought about are:

* Repository (“A central place that holds the master copy of all versions of your project’s file”)
* Revision (“Most version control systems use a numbering system for the versions of a file”)
* Workspace (“A local copy of everything you need from the repository to work on your part of the project”)
* Branching(“People check stuff out, make revisions, commit the changes back in, and update their local workspaces”)
* Merging(“Say you fixed a bug in the release branch and realised that the same bug will be present in the mainline code – it would be nice to merge the change back to the mainline code”)

The strategy used for this application would be Git Flow Branching, Git is a Distributed version control system. Git Flow consists of two branches, the main branch and the development branch. Developers will make all the changes into the development branch, which will then be tested and moved into the main branch if the testing is successful.

# Evolution Strategy

The NBM software would have a low maintenance cost as it is a simple application with few features, evolution to the software would be adding more types of message types which would be easy as the base for it already exists.