2019

Software Engineering Coursework

EUSTON LEISURE MESSAGE FILTERING SERVICE

MURPHY, BROGAN, 40304890

Introduction

Euston leisure required a program that can filter any incoming messages that will check if the message is an email, tweet or SMS. This report will include a requirement specification agreed with Euston leisure, non-functional requirements, a UML use case that ties in with the requirement specification, a UML class diagram outlining all classes that were implemented detailing their properties and behaviours. A test plan will also be included within this document that will outline the reason for testing, scope and objectives, also data and event validation test logs. This document will include a version control plan which will detail my use of GitHub. Finally, the document will include an evolution strategy which will describe how I would add to the program in the future and how I will complete parts I missed.

My requirements specification outlines what was agreed with Euston Leisure, but I have written in brackets beside each requirement that wasn't completed to the standard specified what I managed to implement instead, if I implemented part of the requirement or didn't manage to implement it at all. I designed the system with the idea of creating a base message class with properties, methods and a constructor and have an SMS, Tweet and message class inherit from it and within my main window class use button event handlers and methods to perform all the functionality. Due to timing constraints, I was only able to create a test plan but in an ideal world I'd have relevant methods, comprehensive test cases and their outputs and an analysis of testing.

Requirements Specification

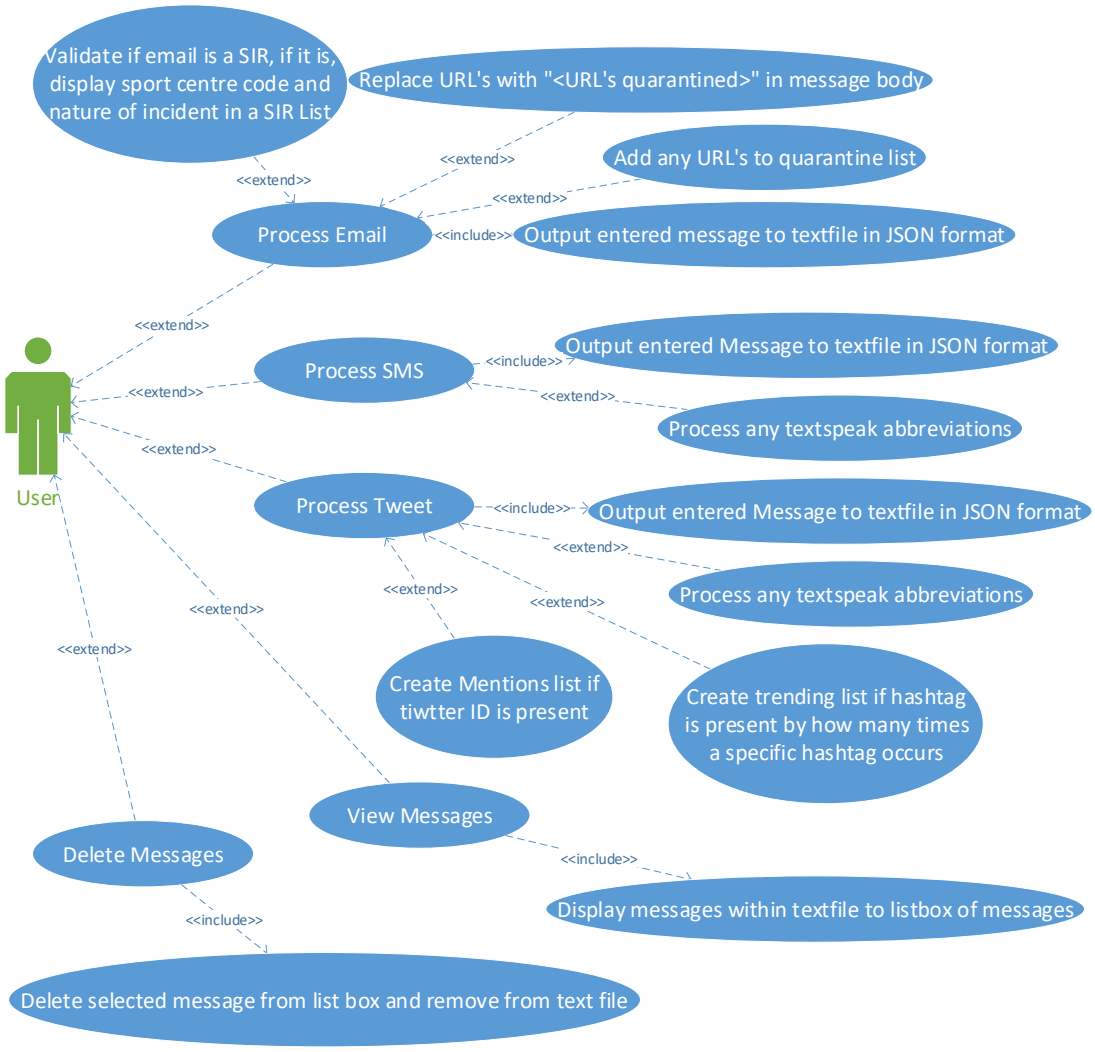
- MessageID must start with either "S", "E", "T" and follow with 9 numbers
- Only emails are to contain a subject
- All Messages must be outputted to a file in JSON format and can be read from the file
- Program must be able to take in manual inputs with the text boxes or read in a file for the input(I could only implement manual inputs)
- SMS Messages must:
    - Have the letter S at the beginning of MessageID
    - Sender must be an international phone number(my implementation only checks for the number to be 10 digits and only contain numerical values)
    - The message body must be a maximum of 140 characters long
    - Must process textspeak abbreviations so that if an abbreviation is entered it will display the expanded version of the abbreviation e.g. "OMG <Oh My God>"
- Email Messages

- Have the letter E at the beginning of MessageID
- Sender must be a valid email e.g. alan@gmail.com
- Subject must be a maximum of 20 characters
- Message body must be a maximum of 1028 characters
- If the message body contains any hyperlinks, the hyperlinks must be stored in a quarantine list and replaced with "<URL Quarantined> within the message body
- Emails must be able to have the Significant Incident Report (SIR) functionality which entails the subject formatted as "SIR" and a valid date, and the message body must contains "Sport Centre Code" which will follow with the code e.g. 22-364-90. The message body must also contain "Nature of Incident" with one of the following f
    - Theft of Properties
    - Staff Attack
    - Device Damage
    - Raid
    - Customer Attack
    - Staff Abuse
    - Bomb Threat
    - Terrorism
    - Suspicious Incident
    - Sport Injury
    - Personal Info Leak
- The Sport centre code and nature of incident must then be written to a SIR list at the end of inputting an SIR email

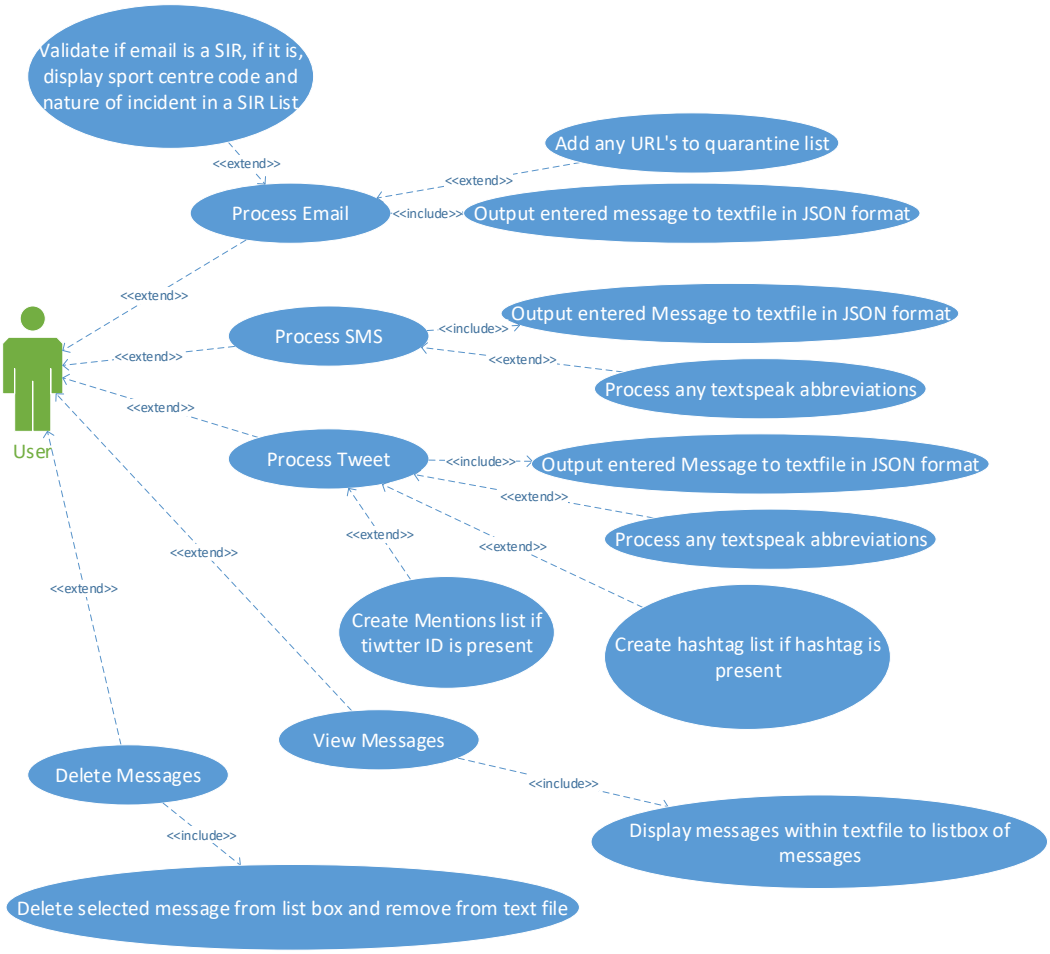
- Tweets:
    - Must contain the letter T at the beginning of MessageID
    - Sender must take the form of a twitter ID e.g. @BroganMurphy4, this means that the sender must begin with the @ sign and follow with a maximum of 15 characters
    - Message body is a maximum of 140 characters
    - Message body must deal with textspeak abbreviations, same as SMS by expanding the abbreviations to their full form, e.g. "TTYL <Talk To You Later>
    - Hashtags within message body will be added to a list which will count the amount of times this particular hashtag occurs to create a hashtag list (My program outputs hashtags within the message body to a hashtag list but doesn't count how many times they occur)
    - Mentions (embedded twitter ID's) e.g. "love the @Metallica album so much" (@Metallica being the embedded twitter ID) will be added to a mentions list

Use Case Diagrams

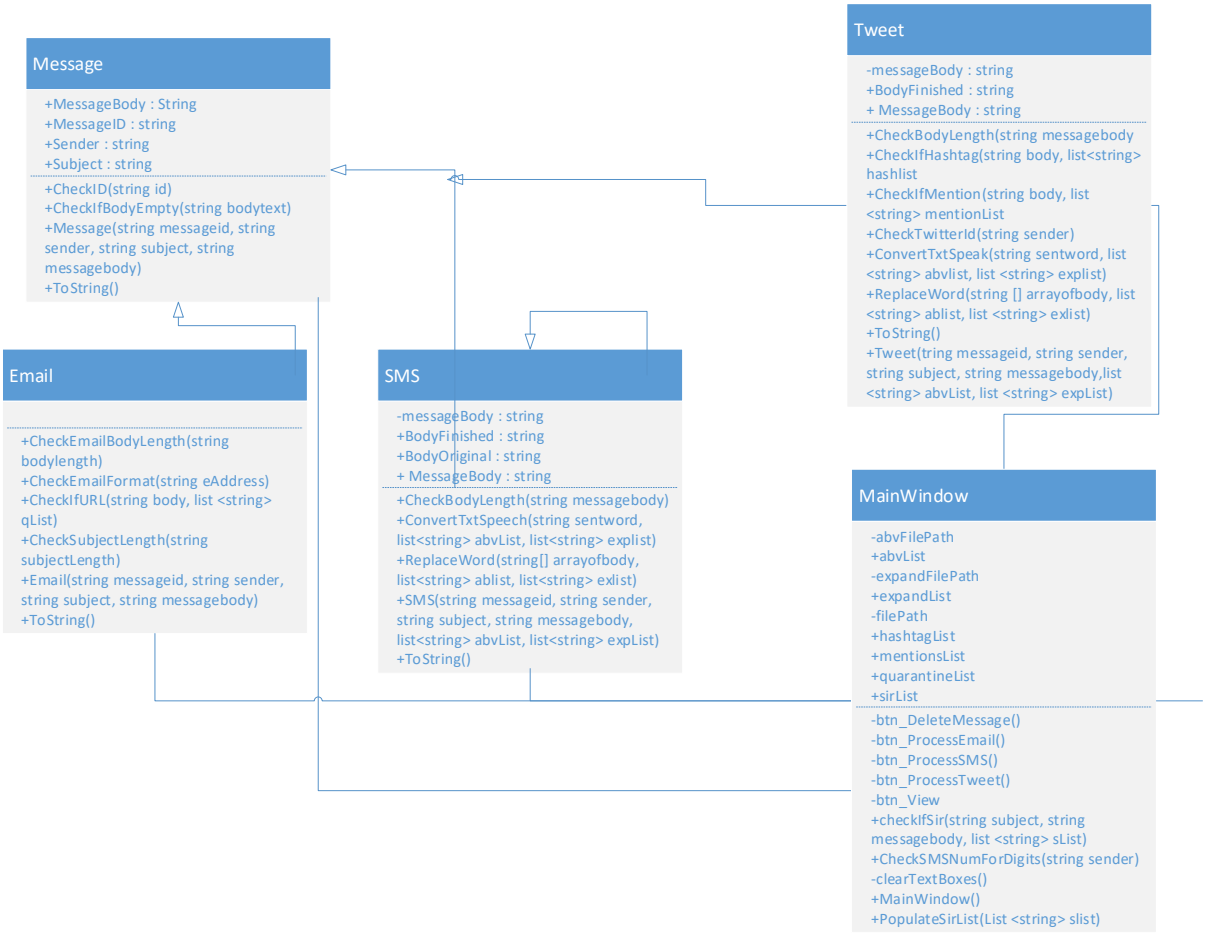
Before implementation:



After Implementation:


Validate if email is a SIR, if it is, display sport centre code and nature of incident in a SIR List
Add any URL's to quarantine list
Process Email
Output entered message to textfile in JSON format
Process SMS
Output entered Message to textfile in JSON format
Process any textspeak abbreviations
Process Tweet
Output entered Message to textfile in JSON format
Process any textspeak abbreviations
User
Create Mentions list if tiwtter ID is present
Create hashtag list if hashtag is present
View Messages
Delete Messages
Display messages within textfile to listbox of messages
Delete selected message from list box and remove from text file
<<extend>>
<<include>>

Class Diagram



Non-Functional Requirements

- Security – the program's data should not be accessed by anyone else apart from those who are registered to view it
- Usability – The program should be straight forward to use and should require as little as possible training time for anyone who must use it.
- Robust – the program must be robust so that any errors or correct data will not be able to infiltrate the text files.
- Efficiency – The program must not take too long performing and use as little processing power as possible.
- Ethical – The program must adhere to any relevant laws.

Test Plan

The message filtering service required testing to ensure the functionality of the program lived up to the requirements set and to ensure each button performs its task correctly, this will be performed by three types of testing;

- Data validation testing
- Event validation testing
- Stress testing

Data validation will be used to ensure that all validation of data matches the requirements that Euston Leisure set using, normal, boundary and extreme test cases

Event validation will be used to ensure that every button on the program works as expected, this will be done by testing the button itself, producing an expected result and what actually happens.

Stress testing can be used to ensure the text files are able to take a large number of JSON messages, it could also be used ensure all list boxes are able to cope with a large amount of items being added to them.

The whole objective of testing is to identify and fix all errors that weren't found during the implementation of the program quickly and efficiently

Data Validation Testing

Test Case	Test Item	Expected Result	Actual Result	Comments
Txt_MessageID	E123456789	Input accepted	As expected	
	S123456789	Input accepted	As expected	
	T123456789	Input accepted	As expected	
	e123454321	Input accepted	As expected	
	s121234346	Input accepted	As expected	
	t154326789	Input accepted	As expected	
	543678345	Error	As expected	
	P908765432	Error	As expected	
	!qwertyfghj	Error	As expected	

Test Case	Test Item	Expected Result	Actual Result	Comments
Txt_Sender	g@hotmail.com	Input accepted	As expected	
	hi@live.co.uk	Input accepted	As expected	
	brogan@gmail.com	Input accepted	As expected	
	56456@fdd.com	Input accepted	As expected	
	fjfgjjfg@no.com	Input accepted	As expected	
	66878@kfkds.com	Input accepted	As expected	
	543678345@fogd	Error	As expected	
	!!!!@///.com	Error	As expected	
	""	Error	As expected	

Test Case	Test Item	Expected Result	Actual Result	Comments
Txt_Subject	Hi	Input accepted	As expected	
	Hello	Input accepted	As expected	
	yes	Input accepted	As expected	
	!"£$%^	Input accepted	As expected	
	qqqqqqqqqqqqqqqqqq	Input accepted	As expected	
	1	Input accepted	As expected	
	wwwwwwwwwwwwwwwwwww	Error	As expected	
	""	Error	As expected	
	wwwwwwwwwwwwwwwwwwwww	Error	As expected	

Test Case	Test Item	Expected Result	Actual Result	Comments
Txt_MessageText	Hi www.google.com	Input accepted	As expected	
	Hello World	Input accepted	As expected	
	Nice hat	Input accepted	As expected	
	'Input with 1027 characters'	Input accepted	As expected	
	h	Input accepted	As expected	
	!!!"""£$£$$	Input accepted	As expected	
	'input with 1030 characters'	Error	As expected	
	""	Error	As expected	

Event validation testing

Control	User interaction	Expected Result	Actual Result	Comments
Btn_ProcessEmail	User clicks on button	Processes text boxes above and stores a JSON format of the textboxes in a text file	As expected	
Btn_ProcessSMS	User clicks on button	Processes text boxes above and stores a JSON format of the textboxes in a text file	As expected	
Btn_ProcessTweet	User clicks on button	Processes text boxes above and	As expected	

		stores a JSON format of the textboxes in a text file		
Btn_View	User clicks on button	Displays all messages in text file in JSON format	As expected	
Btn_DeleteMessage	User clicks on button	Deletes the selected index of the list box and removes it from text file	As expected	

I created these test logs using white box testing as my test method.

Version Control

Version control aided my software greatly. I used GitHub which meant I could create a repository to commit and push code to it while being able to add a comment outlining changes/progress made to the solution. Using GitHub also means that if this implementation was a team project, each member would be able to check out the latest version of the file and comment and commit changes to it collaboratively, meaning that it would be clear how a team member has contributed to each version. A brief plan of using GitHub for this implementation if it was a team project would be a simple case of adding all the team members to the project so they have access to the repository and are able to commit changes simultaneously. This also allows branching so that different members can work on different versions of the software.

Evolution Strategy

I would firstly implement requirements that I did not match during the implementation, which are

- No use of an input file; would save time and effort using the program
- No quarantining of URL's; could pose security risks as some URL's are malicious
- Fixed trending list; displaying how many times a hash tag occurs within a message, this would be useful as it could outline customer feelings towards Euston Leisure e.g. #EustonLeisureIsGreat. It could also be used to track how many occurrences there are of hashtags created by EL.

Maintenance would be relatively easy for the program as no updates would be necessary. However I would update the user interface with some colour to make it better looking.