SET09102 Software Engineering

NAPIER BANK MESSAGE FILTERING SERVICE

40283965

Contents

Introduction

Napier Bank require a prototype application that will validate, sanitize, and categorise incoming messages in the form of tweets, emails, and SMS text messages. The system will allow the user to enter these messages and will write each message in JSON format to a file. This system was created using C# in Visual Studio, as a Windows Forms project.

Requirement Specification

Functional Requirements

The system will allow the user to select whether they are sending information in the form of a tweet, standard email, significant incident report, or SMS text message. The messages will be saved in JSON format in a file. Once the user has selected their desired message type, they will be taken to the relevant form where they will input their message. If a Tweet is selected, they will enter their username (a maximum of 15 characters) and their message which may not be more than 140 characters long. If the message is a standard email, then the user will enter their email address, a subject (a maximum of 20 characters) and the message text (a maximum of 1028 characters). If a Significant Incident Report is selected, the user will select the nature of the incident they are reporting as well as entering their message and email address, they will also select the date of the incident they are reporting. If the message type is an SMS text message, the user will enter their phone number, and a message of no more than 140 characters. If the user sends a Tweet or an SMS Text Message that contains any textspeak abbreviations, these will be expanded, for example, if the user sends “Got your message ROTFL can’t wait to see you”, the system will save this as “Got your message ROTFL <Rolls on the floor laughing> can’t wait to see you”. Any hashtags or mentions (words/phrases beginning with characters # or @ respectively) will be stored into ‘Trending’ and ‘Mentions’ lists respectively. Any URLs contained in Email messages will be removed and replaced by “<URL Quarantined>” in the message, for example “Visit us at http://www.welikelinks.com” becomes “Visit us at <URL Quarantined>”. A Message ID will also be given to each message.

Non-Functional Requirements

The system will have a short response time, meaning the user will not have to wait after clicking any buttons. It will be easy to use with a consistent layout throughout so the user does not get confused about how to use a specific form. This also means that labels and buttons will be labelled so the user knows what the relevant textboxes are for and what the buttons do. The information given back to the user will be the same as the information that was received, showing the accuracy of the data the program displays.

The ‘Main Menu’ Form

The Main Menu form will be the first thing the user will see when they run the program. It will present the user with a menu, displaying 6 buttons. These buttons will be the ‘Send Tweet’ button, the ‘Send Standard Email’ button, the ‘Send SMS Text Message’ button, the ‘Send Significant Incident Report’ button, the ‘View Messages’ button, and the ‘Quit’ button. Once clicked, these buttons will take the user to the relevant form, with the exception of the ‘Quit’ button, which will exit the program.

The 'Send Tweet' Form

The Send Tweet form will provide the user with two textboxes, a submit button and a back button. The first textbox will be for the user's Twitter ID, an '@' symbol will appear before the textbox to be followed by their username, this means that the user does not need to enter the '@' symbol, and validation will take place to ensure that the user only enters letters, numbers or underlines. The username should not be any longer than 15 characters long. The second textbox will be for the user's message, which will be a maximum of 140 characters. At the bottom of the form there will be a button labelled 'Submit' which will validate the information entered in the text boxes, if there is any invalid input, the user will be prompted to try again, otherwise the information will be saved in the file and the user will be returned to the main menu. Any textspeak used by the user will be expanded as described in the functional requirements, for example the text "ROTFL" will be replaced with "ROTFL<Rolling on the floor laughing>" in the body of the message. The back button will take the user back to the main menu. Any hashtags (words entered directly after a #) will also be stored in a separate Trending.json file, and each mention (words entered directly after a @) will be stored into a Mentions.json file.

The 'Send Email' Form

The Send Email form will provide the user with three text boxes, a submit button, and a back button. The first text box is where the user will enter their email address. The second textbox will contain the subject of the email, which will be no more than 20 characters long. The third textbox will be for the email itself, which will have a limit of 1028 characters in length. The submit button will check for validation (valid email, and that the subject and message length are valid) before writing the information to a file in JSON format, and returning the user to the main menu. Any links contained in the email will be replaced with "<URL Quarantined>" in the body of the message, i.e. "Check out www.unrelatedstuff.com" is stored in the messages file as "Check out <URL Quarantined>". The program will validate the email address entered by the user to make sure it has been entered in the correct format. The back button will take the user back to the main menu.

The 'Send Significant Incident Report' Form

The Send Significant Incident Report form will provide the user with two textboxes, a date picker and a list box. The first textbox will be for the user's email address. The date picker will be where the user selects the date of the incident being reported. The list box provides a list of types of incidents, of which the user will select one. The second and final textbox will be where the message will be entered. The submit button will be at the bottom of the form, and will validate the input and write it to the file. URLs entered in the message will be replaced in the body with "<URL Quarantined>" as above in the 'Send Email' form. The email address entered by the user is checked to ensure it is a valid email i.e. in the correct format. The back button will return the user to the main menu.

The 'Send SMS Text Message' Form

The Send SMS Text Message form will allow the user to enter their phone number in one textbox, and their message in another, and a button to submit this message. When the message is submitted, the program will check that the phone number entered is a valid number, and will check that the message is no longer than 140 characters long. Any textspeak contained in the SMS message will be expanded and explained in the JSON file, for example if a message contains the text "LOL", this text will be replaced by "LOL<Laughing out loud>" in the JSON file. The submit button will make the program carry out these checks before writing the information to the JSON file, unless any information is invalid in which case an error message will appear and the user will be prompted to try again. There will also be a back button to direct the user back to the main menu.

The 'View Messages' Form

The View Messages form will display five buttons, a 'View Tweets' button, a 'View SMS Text Messages' button, a 'View Significant Incident Reports' button, a 'View Email Messages' button, and a 'Back' button. Each of these buttons will redirect the user to the form they have requested.

The 'View Tweets' Form

The View Tweets form will contain one label and three buttons. The label will contain text loaded from the "Tweets.json" file. The buttons will be labelled 'View Trending', 'View Mentions' and 'Back'. Each of which loading the relevant form for the user.

The 'View SMS Text Messages' Form

The View SMS Text Messages form will contain one label and one button. The label will contain text that has been loaded from the 'SMS.json' file. The button, labelled 'Back', will redirect the user back to the 'View Messages' form.

The 'View Significant Incident Reports' Form

The View Significant Incident Reports form will contain one label and one button. The label will display text that will be retrieved from the 'SIRs.json' file. The button will be labelled 'Back' and will take the user back to the 'View Messages' form when clicked.

The 'View Email Messages' Form

The View Email Messages form will have one label and a 'Back' button. The label will display the text taken from the 'Emails.json' file. The 'Back' button will take the user back to the 'View Messages' form.

The 'View Trending' Form

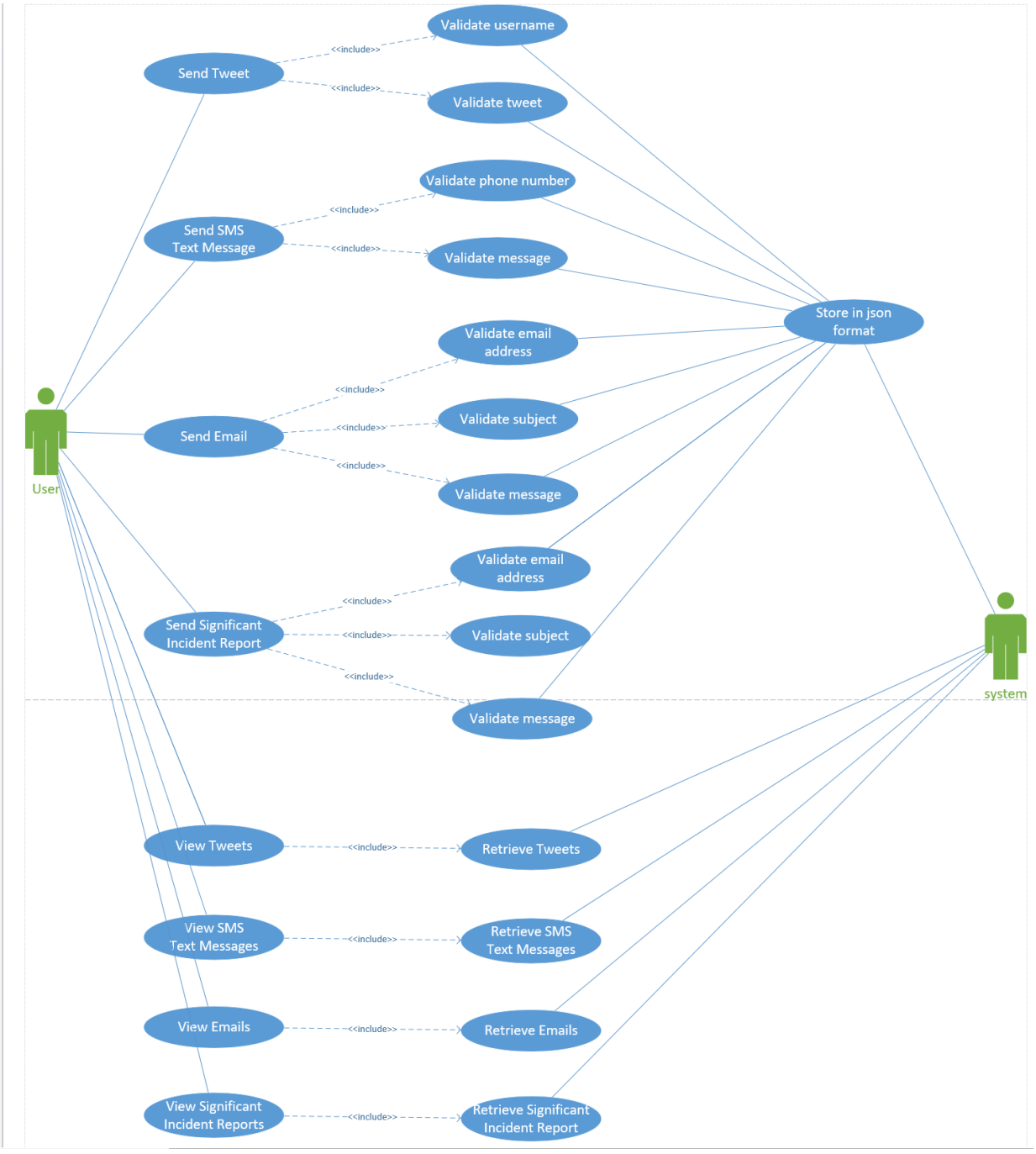
The View Trending form has one label and a 'Back' button. The program will retrieve text from the 'Trending.json' file and display it in the label on the form. The 'Back' button will take the user back to the 'View Tweets' form.

The 'View Trending' Form

The View Mentions form has a label and a button. The program will read text from the 'Mentions.json' file and display it in the label on the form. The 'Back' button will take the user back to the 'View Tweets' form.

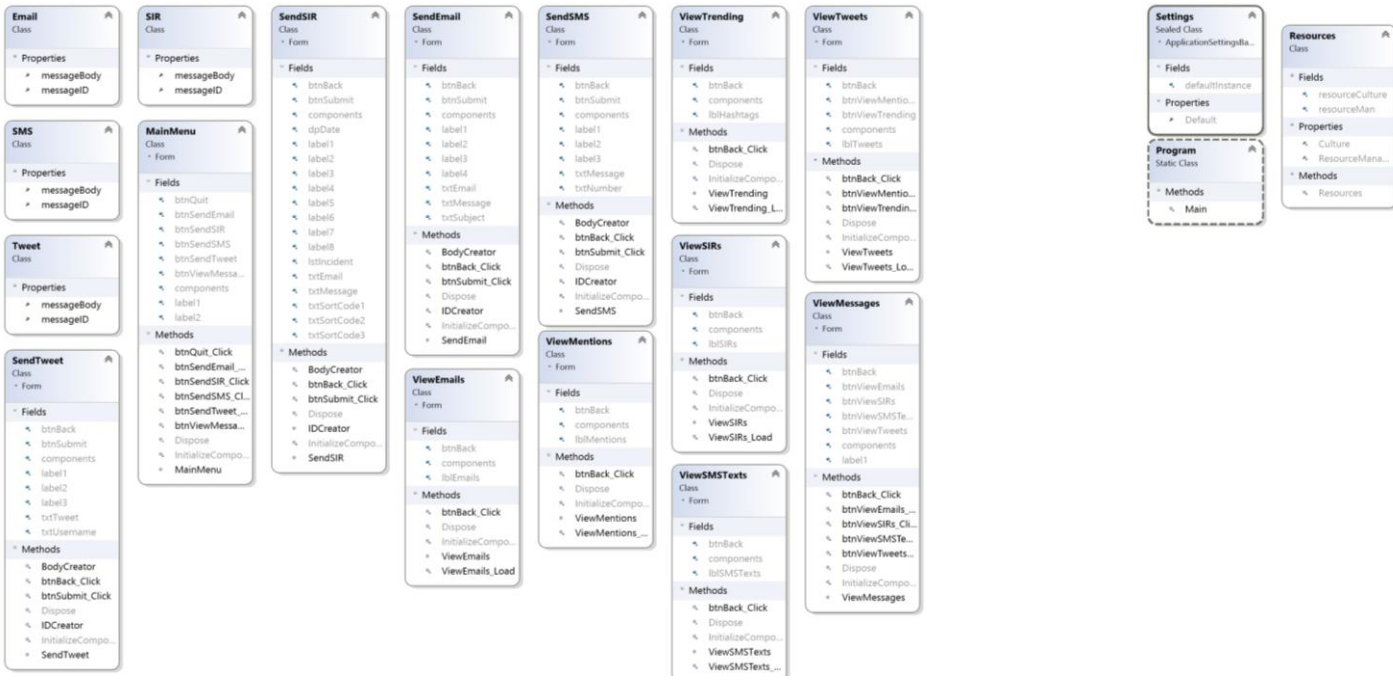
Use Case Diagram

A use case diagram represents the user's interaction with the system. The example below shows the different actions that the user can take while using the Napier Bank Message Filtering Service system. For example, when a user sends a tweet to the system, it first validates the username and the tweet entered, before writing the message and storing it in a JSON format, the same applies for sending an SMS message, where validation is needed for the phone number and the message and so on. It also shows the case of the user viewing a specific type of message, for example if the user wishes to view the tweets sent to the system, the system retrieves these tweets from the relevant file so that they are there for the user to see.



Class Diagram

The following class diagram, created in Visual Studio, displays all the classes (including the forms) used for the system. It displays the fields, methods and properties of these classes which gives an insight to how the system works and what goes on 'behind the scenes' while the user is running the program. On the right hand side of this class diagram, are the classes not created by the programmer.



Testing and Analysis

With the Agile method being adopted for this project, testing was carried out from beginning to end to ensure a constant idea of whether or not the program was working. This included testing parts of the system as they were implemented, for example when a new button was added to a form, and the code added to it, the program would be tested to make sure that the button redirected the user to the relevant form, even if there was nothing on the new form at that point, it was known that the button worked, and would not be a problem that would appear later on in the implementation stage. One downfall of this method is that it may consume slightly more time than required. For example, if no tests were taken during the implementation, and the program was only tested once the product was completed, there is a chance (albeit a very small one) that the program could be problem free, which would make testing during implementation seem to take up more time which could be spent 'fine tuning' the program or perfecting other things such as its aesthetics. Testing was also carried out via Visual Studio using its built in 'Test' feature. The test cases used for testing the Napier Bank Message Filtering system are displayed below along with the results of the tests.

Navigation

These tests show if the program navigates between forms as it should (i.e. clicking the 'Send Tweet' button should not take the user to the 'Send SMS Text Message' form). These are all buttons, and results can be seen in the table below.

Form	Test Case - Action	Expected Result	As expected?
MainMenu	btnSendTweet – Click	User should be navigated to the 'Send Tweet' form.	Yes.
MainMenu	btnSendSMS – Click	User should be navigated to the 'Send SMS Text Message' form.	Yes.
MainMenu	btnSendSIR – Click	User should be navigated to the 'Send Significant Incident Report' form.	Yes.
MainMenu	btnSendEmail – Click	User should be navigated to the 'Send Email' form.	Yes.
SendTweet, SendSMS, SendSIR, SendEmail	btnSubmit – Click	If input is accepted as valid, the user should be navigated to the 'Main Menu' form.	Yes.
MainMenu	btnViewMessages – Click	User should be navigated to the 'View Messages' form.	Yes.
SendTweet, SendSMS, SendEmail, SendSIR, ViewMessages	btnBack – Click	User should be navigated to the 'Main Menu' form.	Yes.
ViewMessages	btnViewTweets – Click	User should be navigated to the 'View Tweets' form.	Yes.
ViewMessages	btnViewSMSTexts – Click	User should be navigated to the 'View SMS Text Messages' form.	Yes.
ViewMessages	btnViewSIRs – Click	User should be navigated to the 'View Significant Incident Reports' form.	Yes.
ViewMessages	btnViewEmails – Click	User should be navigated to the 'View Email Messages' form.	Yes.
ViewTweets	btnViewTrending – Click	User should be navigated to the 'View Trending' form.	Yes.

ViewTweets	btnViewMentions – Click	User should be navigated to the 'View Mentions' form.	Yes.
ViewTweets, ViewSMSTexts, ViewSIRs, ViewEmails	btnBack – Click	User should be navigated to the 'View Messages' form.	Yes.
ViewTrending, ViewMentions	btnBack – Click	User should be navigated to the 'View Tweets' form.	Yes.

Sending a Tweet – Validation

These tests check that the validation on the 'SendTweet' form works as it should. When the 'Submit' button is clicked, the system checks that the information entered is valid. Neither of the boxes may be left blank. The txtUsername field should contain letters, numbers, or underscores and may be a maximum of 15 characters long. The txtTweet field may not be longer than the 140 character limit. If any input is invalid, a message box will appear to inform the user of this and prompt them to try again.

Test Case – Input	**Expected Result**	**As Expected?**
txtUsername - "" (left blank)	Invalid input message will appear as the username cannot be blank.	Yes.
txtUsername – "MoreThan15Characters"	Invalid input message will appear as username is longer than 15 character limit.	Yes.
txtUsername – "Test user"	Invalid input message will appear as username cannot contain spaces.	Yes.
txtUsername – "Test-user"	Invalid input message will appear as username cannot contain characters other than letters, numbers and underscores.	Yes.
txtUsername – "TestUser"	Input accepted.	Yes.
txtTweet – "" (left blank)	Invalid input message will appear as tweet message cannot be left blank.	Yes.
txtTweet – "This tweet contains more than 140 characters, therefore it is not valid input and should not be accepted and the user should be prompted to try again"	Invalid input message will appear as the tweet is longer than the 140 character limit.	Yes.
txtTweet – "This tweet should be accepted as it is less than 140 characters long"	Input accepted.	Yes.

Sending an SMS Text Message – Validation

This tests the functionality of the validation on the 'SendSMS' form. Neither of the boxes may be left blank. The txtNumber box must consist of no less than 8 and no more than 12 numeric characters. The txtMessage box may not exceed the 140 character limit. Once the 'Submit' button is clicked, the program will check these boxes to make sure that the input is valid, if not then a message box will appear to tell the user that the input was invalid and that they should try again.

Test Case – Input	**Expected Result**	**As Expected?**
txtNumber = "" (left blank)	Invalid input message will appear as the 'Phone Number' field cannot be left blank.	Yes.
txtNumber – "Words"	Invalid input message will appear as the input is not numeric.	Yes.
txtNumber – "1234567"	Invalid input message will appear as there is not enough numbers.	Yes.
txtNumber – "1234567891011"	Invalid input message will appear as there are too many numbers.	Yes.
txtNumber – "07252637282"	Input accepted.	Yes.
txtMessage – "" (left blank)	Invalid input message will appear as the message cannot be blank.	Yes.
txtMessage – "This message contains more than 140 characters, therefore it is not valid input and should not be accepted and the user should be prompted to try again"	Invalid input message will appear as the message contains more than 140 characters.	Yes.
txtMessage – "This message should be accepted as it is less than 140 characters long"	Input accepted.	Yes.

Sending an Email – Validation

This tests how the 'SendEmail' form validates input, and if it recognises the valid email address format. The txtEmail box should be a valid email address (e.g. testuser@test.com). The txtSubject box can contain up to 20 characters. The txtMessage box can contain up to 1028 characters. None of the input boxes may be left blank. When the 'Submit' button is clicked, the program checks these boxes to ensure that input is not invalid, if invalid input is found a message box will appear to let the user know and prompting them to try again.

Test Case – Input	**Expected Result**	**As Expected?**
txtEmail – "" (left blank)	Invalid input message will appear as the email address cannot be blank.	Yes.
txtEmail – "email"	Invalid input message will appear as it is not in valid email format.	Yes.
txtEmail – "email@email"	Invalid input message will appear as it is not in valid email format.	Yes. The first regular expression used to validate emails only checked for an '@' symbol, this meant these types of input were accepted despite

		not being valid. The code was changed and the case was retested.
txtEmail – “test@test.”	Invalid input message will appear as it is not in valid email format.	Yes. As above. Retested and found that it validates correctly.
txtEmail – “e:mail@email.com”	Invalid input message will appear as illegal character entered.	Yes.
txtEmail – “testuser@test.com”	Input accepted.	Yes.
txtSubject – “” (left blank)	Invalid input message will appear as the subject cannot be blank.	Yes.
txtSubject – “More than 20 characters”	Invalid input message will appear as the subject is longer than the 20 character limit.	Yes.
txtSubject – “Testing Input”	Input accepted.	Yes.
txtMessage – “” (left blank)	Invalid input message will appear as the message cannot be blank.	Yes.
txtMessage – “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet mattis magna, et faucibus sapien. Quisque mollis dui ut turpis rutrum, nec ultricies ex ullamcorper. Morbi id efficitur ante. Proin eleifend pellentesque viverra. Proin sed nibh massa. Pellentesque nibh velit, fringilla non nisl quis, pharetra consectetur velit. Sed mollis a turpis molestie gravida. Pellentesque vel ante dolor. Quisque id est elementum, posuere tortor eget, suscipit justo. Donec sollicitudin quam nunc eget interdum ultricies. Nulla facilisi. Quisque in lectus in nisi tristique facilisis vitae a sem. In placerat eros tempus mi scelerisque, in consectetur ex maximus. Suspendisse fermentum tempus nunc eget tincidunt. Aenean ex justo, iaculis ut lobortis et, luctus id risus. Vivamus accumsan dolor vitae dictum fringilla.	Invalid input message will appear as the message is longer than the 1028 character limit.	Yes.

Integer nec pretium elit. Fusce interdum, purus sed vulputate mattis, orci mi pellentesque felis, id tristique nisi dui at sem. Fusce eu orci nec sem iaculis tincidunt. Quisque quis nunc nunc. Nunc dolor nisi metus.”		
txtMessage – “Testing the body of a message.”	Input accepted.	Yes.

Sending a Significant Incident Report – Validation

This tests the validation on the ‘SendSIR’ form. The system checks the email address entered to check it is in the valid format, it checks that the user has selected a Nature of Incident, and checks that the message entered is not empty, and is no more than 1028 characters long.

Test Case – Input	**Expected Result**	**As expected?**
txtEmail – “” (left blank)	Invalid input message will appear as the email address may not be blank.	Yes.
txtEmail – “test”	Invalid input message will appear as the input is not a valid email address.	Yes.
txtEmail – “test@test”	Invalid input message will appear as the input is not a valid email address.	Yes.
txtEmail – “test@test.com”	Input accepted.	Yes.
lstIncident – No option selected	Invalid input as no option is selected on the list.	Yes.
lstIncident – “Theft” selected	Input accepted.	Yes.
txtMessage – “” (left blank)	Invalid input message will appear as the message may not be blank.	Yes.
txtMessage – “Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sit amet mattis magna, et faucibus sapien. Quisque mollis dui ut turpis rutrum, nec ultricies ex ullamcorper. Morbi id efficitur ante. Proin eleifend pellentesque viverra. Proin sed nibh massa. Pellentesque nibh velit, fringilla non nisl quis, pharetra consectetur velit. Sed mollis a turpis molestie gravida. Pellentesque vel ante dolor. Quisque id est elementum, posuere tortor eget, suscipit justo. Donec	Invalid input message appears as the message is longer than the 1028 character limit.	Yes.

sollicitudin quam nunc eget interdum ultricies. Nulla facilisi. Quisque in lectus in nisi tristique facilisis vitae a sem. In placerat eros tempus mi scelerisque, in consectetur ex maximus. Suspendisse fermentum tempus nunc eget tincidunt. Aenean ex justo, iaculis ut lobortis et, luctus id risus. Vivamus accumsan dolor vitae dictum fringilla. Integer nec pretium elit. Fusce interdum, purus sed vulputate mattis, orci mi pellentesque felis, id tristique nisi dui at sem. Fusce eu orci nec sem iaculis tincidunt. Quisque quis nunc nunc. Nunc dolor nisi metus.”		
txtMessage – “Testing Significant Incident Report”	Input accepted.	Yes.

Textspeak

This part of the testing checked that the system could recognise any textspeak entered by the user. It will read the message until an abbreviation is found, it should then expand the abbreviation to include what it means, before continuing reading the rest of the message. For example, the text “That was funny LOL” would become “That was funny LOL<Laughing out loud>”.

Form	**Input**	**Expected Result**	**As Expected?**
SendTweet	“Hi M8, send me the file ASAP. TY.”	Message should save as: “Hi M8<Mate>, send me the file ASAP<As soon as possible>. TY<Thank you>.”	Yes.
SendSMS	“NP. Sent you the file. NRN.”	Message should save as: “NP<No problem>. Sent you the file. NRN<No response/reply necessary>.”	Yes.

Trending and Mentions

This part of the testing checks that the hashtags entered in tweets are recognised and stored in separate files. For example, the tweet: "The #new @NapierBank system is #NBMFS", will create a Trending file containing '#new' and '#NBMFS', and a Mentions file containing '@NapierBank'. Due to these only being available in tweets, this only applies to the 'SendTweet' form.

Input	Expected Result	As Expected?
"Why do @Java #developers wear glasses? They can't C#"	'@Java' added to Mentions list, and '#developers' added to Trending list.	Yes.

Quarantined URLs

Similar to the Textspeak tests, these tests check for any URLs that have been posted via Email or Significant Incident Report. It then replaces these URL's with the text "<URL Quarantined>" in the file. The message is searched until any URLs are found to be replaced.

Form	Input	Expected Result	As Expected?
SendSIR	"Visit https://www.google.com"	Message will save as: "Visit <URL Quarantined>"	Yes.
SendSIR	"My website is http://www.dodgylink.com"	Message will save as: "My website is <URL Quarantined>"	Yes.
SendSIR	"Have a look on www.website.com"	Message will save as: "Have a look on <URL Quarantined>"	Yes. First attempt did not catch the URL, however since updating the regular expression it now works with these URLs.
SendEmail	"Go to https://www.worldwideweb.com"	Message will save as: "Go to <URL Quarantined>"	Yes.
SendEmail	"Check out http://www.virushere.com"	Message will save as: "Check out <URL Quarantined>"	Yes.
SendEmail	"Look here: www.notavirus.com	Message will save as: "Look here: <URL Quarantined>"	Yes.

Version Control Plan

There are a few types of version control systems that can be used, such as a local version control system, which works by keeping patch sets and recreates the file's contents exactly at any point, and a centralised version control system where files are stored on somebody's local computer and are not accessible to other team members working on the same files. For this project, an appropriate version control plan to support the development iteration and collaboration between team members would be a distributed version control system. This type of system stores the entire history of files on every local machine and synchronises the local changes made by a user to the server when required, so that changes can be shared with the rest of a team providing the team with a good collaborating environment.

Evolution Strategy

Software will always continue to change, and technology will continue to advance. These are facts, and facts that will affect any program. Because of this, an effective evolution strategy is required so that the system developed can cope with any changes that may render it in any a lesser product than before, i.e. the new advancements make the system slower, or it may produce errors that mean the system can no longer run at all. In addition to this, the requirements will also change. The user may want or need something new, and it is a good idea to already have an evolution strategy in place to make sure that any changes can be made as simply as possible. The evolution process can be seen as a cycle, where we start with a working system, then a required change is identified, followed by proposals of a possible change, if acceptable this leads to the software being 'evolved' or updated, taking us back to the start of the system where we have a system that fits the requirements. A predicted evolution for this project would be to allow different types of messages to be sent and stored from other social media platforms, i.e. Facebook posts, LinkedIn messages etc. This particular system could also be evolved by making it easier to view individual messages, rather than the current system of all messages of one type showing on the same page. It could also have a more sophisticated system for expanding textspeak, as at the moment it does not cater for people who may typing in all upper case, meaning input such as "PULL" would become "PUL<Upload>L". With the code being commented effectively, any changes to be made can be done so with less hassle, as these comments provide a brief description of what the code does. This means that if someone other than the programmer who originally wrote the code wishes to make any modifications, this can be done in less time with the help of being able to see easily where changes should be made.