# Analysis

Euston Leisure, an association of sport centres in the city of Euston, are looking for a software system, Euston Leisure Messaging (ELM), to be developed.

This system will provide the following service:

Validate and categorise incoming messages into the following formats:

* SMS
* Emails
  + Standard Emails
  + Significant Incident Report Emails
    - Generate a SIR list
* Tweets
  + Generate a trending list of Hashtags
  + Generate a Mentions list

The display and process of these messages is to be done using a suitable Graphical User Interface (GUI)

## Requirements Document

The following Requirement Specification design was chosen because it provides detailed analysis of the customer’s needs and describes how they will be attained. It Is in a clear, readable format for validation of implementation regarding the requirements.

It also provides clear guidelines on the requirements needed to develop the system, what the functional/ non- functional requirements are. This also allows validation test to be written during the implementation process.

This design also allows for maintenance to be done easily as the relationship between each of its parts is described and can be easily understood.

### Purpose

The Purpose of this document is to detail the requirements of the project. The project Requirement Specification will contain the following headlines: Project brief, Scope, Resources, Security, System overview, functional requirements and non-functional requirements.

Project brief will provide a description of the client brief previously supplied.

Scope will give a description of the size of the project

Resource gives a description of what resources have been sourced to be most suitable for this project

System overview contains an overview of what the system will do. And display system requirements

Functional requirements will detail what both the view model and the business model will entail. It will also detail what development software will be used.

Non-functional requirements will detail the design of each interface.

### Project Brief

Euston Leisure are looking for a service, Euston Leisure Messaging (ELM), to be developed that will validate and categorise incoming messages to Euston Leisure in the following forms:

* SMS text messages
* Emails – standard Emails & Serious Incident Report Emails
* Tweets

They are also looking for lists to be produced of the following:

* Trending List
* Mentions List
* SIR list

### Scope

The system will provide the user the ability to load messages from a file and view the messages processed by the system.

An Agile approach will be taken throughout development – this ensures any updates/changes to this system will be implemented quickly.

It is to be used by relevant member of staffs only. Therefore, the user Interface (UI) must be of a simple, non-complex design that does not require an extended period of training to use successfully.

This project is due to be installed by 24th November 2017. Project start date is 12th October. That gives a total of 6 weeks to complete before installation.

### Resources

The project will use the following development creation programs:

* Microsoft Visual studio – to develop the view model and provide key binding to the business model
* Programming code – C# will be used to write the coding of the software
* Microsoft Visio – to create all analysis & design UML diagrams
* Microsoft Office – to write up reports/documents of information required

### User Requirements

The following services will be provided to the user:

* Input messages individually
* Have the system automatically filter & process messages
* Search through all messages processed
* Display lists of:
  + Trends
  + Mentions
  + SIR’s

### System Overview

The project will consist of a front-end interface that will consist of numerous user controls and a single window

Microsoft Visual studio will be used to create the view model for the system. The User Interface will be developed with a Windows Presentation form application.

The JSON/txt files are automatically generated by the system. No need to pre-create them.

The computer system used to deploy this developed software must:

* Have a keyboard
* Have a Monitor
* Have a Mouse
* Be running on Windows 7 service pack 1 or later
* Minimum 4 GB RAM
* Minimum 20 GB HDD space - 5400 RPM hard drive
* 1.6 GHz or faster processor
* 600 MB of available hard disk space (language pack)
* DirectX 10 - capable video card running at 1024 x 768 or higher display resolution
* **Able to run .NET version** 3.5, 4.0, or 4.5

### System architecture

The System will consist of the following modules:

* Views – This module contains all the user controls for the UI that the user will interact with.
* ViewModels – This module contains the classes that provide the content for most buttons/Textblocks. It also contains the button controls and methods for processing messages.
* Models – This module contains the classes that are used to create objects of messages/lists to be saved/loaded from Json/txt files.
* Filter – this module contains the methods for automatic filtering of messages from the txt file input
* Database – this module contains the classes designed to write/load the objects to JSON/txt files
* Commands – this module contains the relay command class to load the user-controls into the relevant content controls.

All the above modules are re-used repeatedly throughout the system. Depending on what the user decides to do: Load messages from a txt file/ input messages/view messages/view lists.

## System Requirement Specification

### Functional Requirements

The computer system will provide the following:

* Message select window
* Input page for each of the following:
  + SMS Input
  + Tweet Input
  + Standard Email Input
  + Significant Incident Report Email Input
* Display Lists of the Following:
  + SIR List
  + Mentions List
  + Trending List
* Must be able to automatically identify the message type & process accordingly
* Ability to write to Json Files
* Ability to read from text/Json files.

Message Select Window will:

* Provide the user with buttons to navigate to one of the following pages:
  + SMS input page
  + Standard Email input page
  + SIR Email page
  + Tweet input page
* Provide a button to navigate to display Lists page
* Provide a button to exit the application

SMS Input Page

* Provide text boxes to Input Message ID, Sender and body of message
* Message ID is ”S” followed by nine numbers
* Sender is an international code followed by the phone/mobile number
* Body is limited to 140 characters
* Check the body of text for any txt abbreviations and provide full meaning in brackets beside abbreviation
* Write the Message to a JSON file
* An insert button to store messages

Tweet Input Page

* Provide text boxes to Input Message ID, Sender and body of message
* Message ID is ”T” followed by nine numbers
* Sender is limited to 15 characters
* Body is limited to 140 characters
* Check the body of text for any txt abbreviations and provide full meaning in brackets beside abbreviation
* Write the Message to a JSON file
* An insert button to store messages
* Write the sender to the mentions list
* Search the body of tweet for any words beginning with ‘#’ and save them to the trend list

Standard Email Input Page

* Provide text boxes to Input Message ID, Sender, Subject and body of message
* Message ID is ”E” followed by nine numbers
* Sender comprises of an Email address that is validated
* Subject is 20 characters long
* Body is limited to 1024 characters
* Check the body for any websites, and replace the URL with < URL Quarantined>
* Process all messages appropriate to its type
* An insert button to store messages
* Store any URL’s found into a Quarantine List

Serious Incident Report Emails

* Provide text boxes to Input Message ID, Sender, Subject and body of message
* Message ID is ”E” followed by nine numbers
* Sender comprises of an Email address that is validated
* Subject Consists of SIR followed by a date
* Body is limited to 1024 characters
* The first line of body consists of the Sport Centre Code: 66-666-99
* The second line of body consists of Nature of Incident and one of the following:
  + Theft of Properties
  + Staff Attack
  + Device Damage
  + Raid
  + Customer Attack
  + Staff Abuse
  + Bomb Threat Terrorism
  + Suspicious Incident
  + Sport Injury
  + Personal Info Leak
* Check the body for any websites, and replace the URL with < URL Quarantined>
* Process all messages appropriate to its type
* An insert button to store messages
* Store any URL’s found into a Quarantine List
* Store Sport Centre Code and Nature of incident into the SIR List

Message Display Page will:

* Allow the user to view each of the following message types:
  + SMS
  + Tweet
  + Standard Email
  + SIR Email
* Two textboxes provided for viewing:
  + One textbox for Message ID
  + The other displays sender, subject and body
* Buttons provided to cycle forwards/backwards through the messages

List Display Page

* Allow the user to view the selected list of one of the following:
  + SIR List
  + Trending List
  + Mention List
* Provides two textboxes:
  + The first will display the Trend/Centre code/Mention
  + The second will display the number of occurrences of the Trend/mention or the Nature of Incident if the SIR List

Output files

ALL output in every JSON file will be of the correct JSON format.

### Non-functional Requirements

* All messages must be strings composed of ASCII characters
* Each input/view page will be loaded in the subsequent Content Control of the current page showing
* Each Page will be a different colour to Show it has been shown
* Each page will provide clear details of which box(es) information is to be inserted into
* Font size for all displayed/input text will be 16
* Textblock displaying: “Euston Leisure Message Filtering System” will always be shown, regardless of option chosen
* Textblock on each window to display use
* Textblocks provided on each page that describes what each textBox displays
* Text in Buttons must provide simple description of what the button(s) do.

### Performance

The software will perform as expected. It will also have the capability for any future expansion that the rugby club may decide upon. The Software will also be robust and not prone to errors from incorrect usage. As a result, this will reduce the amount of maintenance required too.

### Glossary

Agile

Term used to describe an iterative approach to software development – gaining enough information to progress on a project. After a certain time – a few weeks generally, the work developed is then reviewed and changes made to improve. This is based on further information gained/ further analysis of what is required

Visual Studio

Visual Studio is a complete set of development tools for building ASP.NET Web applications, XML Web Services, desktop applications, and mobile applications. Visual Basic, Visual C#, and Visual C++ all use the same integrated development environment (IDE), which enables tool sharing and eases the creation of mixed-language solutions. In addition, these languages use the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP Web applications and XML Web Services.

C#

C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. You can use C# to create Windows client applications, XML Web services, distributed components, client-server applications, database applications, and much, much more. Visual C# provides an advanced code editor, convenient user interface designers, integrated debugger, and many other tools to make it easier to develop applications based on the C# language and the .NET Framework.

Microsoft Office  
Suite of products developed by Microsoft Corporation that includes Microsoft Word, Excel, Access, Publisher, PowerPoint, and Outlook. Each program serves a different purpose and is compatible with other programs included in the package. The suite of programs is compatible with both the Windows and Macintosh operating system. Microsoft Office is the most common form of software used in the western world.

Requirement Specifications

Software requirements specification establishes the basis for an agreement between customers and contractors or suppliers (in market-driven projects, these roles may be played by the marketing and development divisions) on what the software product is to do as well as what it is not expected to do. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Used appropriately, software requirements specifications can help prevent software project failure

View Model

Name given to the Graphical User Interface (GUI) that the User interacts with to perform the required purpose of the developed software.

Functional Requirements

A **functional requirement** defines a function of a system or its component. A function is described as a set of inputs, the behaviour, and outputs.

Non-Functional Requirements

A **non-functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours.

UML Diagram

Unified Modelling Language (UML) diagrams are used to display the various stages of designing a software project. Each diagram representing a different aspect of the design phase.

Visio is use to create these diagrams, easily and professionally.

## Use-Case Model



# Design

## Class Diagram

For the reading messages from a file only

# Implementation

The MVVM style of implementation was used for the prototype. This allows for the modularity developed and can allow for future modifications/evolution to the system. As most of the content/command for Textblocks/textboxes and buttons is stored in the ViewModels classes as properties. Allows for easy manipulation of these properties.

A modular approach was taken. This was achieved by splitting up the prototype by adding the following folders, which represent the different modules:

* Views Folder – Contains the User-controls for the prototype. These are the basic “windows” that are displayed to the user, depending on which button they have pressed – all contain a different background colour to show where they appear when called in each user control content control.
* ViewModels Folder – contains the classes that display text/ button Content/Commands for the corresponding User-Control
* Models Folder – contains the classes were input entered is stored as an object. Input is either via user or read from a file
* Database Folder – contains the classes that read/write to txt files. This is where the JSON Serialize/de-serialize occurs
* Commands folder – contains the Relay Command class that allows pages to be loaded (as an event) into the relevant User-control content control. This class inherits the ICommand class

MainWindow.xaml was not added to a folder it was left out with them.

# Test Document

The following test document was created because it is a representation of the testing done during development that helped validate that the implementation matched the requirements. This also simplified debugging as issues/problems with coding were spotted easily, within each unit, and fixed quickly.

Also, some documentation is cut down as this is a small application, extensive, full testing is not needed. Nor the documentation.

## Test Plan

### Objectives & Scope

User controls for user input will be tested In the following manner:

* Message select window:
  + All buttons will be tested to ensure they display the correct page or exit application successfully
* SMS input page:
  + Validation of each input will be tested
* Email input page:
  + Validation of each input will be tested
* Tweet input page:
  + Validation of each input will be tested
* Display lists Page:
  + All buttons will be tested to ensure they display the correct page
  + All inserted records/lists etc will be checked to ensure they have all been processed correctly
* Display Messages Page:
  + All Buttons tested to ensure they show each message processed correctly

Message Process Automation will be tested by:

* Json/txt files:
  + Will be checked to ensure the written records are in JSON format and display the correct information, depending on what they are expected to store

### Testing Approaches

Unit testing

will be conducted, in Visual studio, to test each type of message validation/processing is correct.

White box & black box testing

will be used for manual input:

* White box testing will be done to ensure input is accepted and output is stored correctly in Json File(s)
* Black box testing will be used to ensure further validation is done, as per functional requirements – character limits are adhered too, Quarantine/SIR/Trend & Mention lists contain ALL respective available data from input.

Parameter interface

Will be the chosen interface testing. As data is passed form one procedure to the next. This interface testing will be conducted by manual input of each message type in the interface created and checking assumed validation by message boxes displaying failed validation/JSON file(s) Written on success.

#### System Testing

Will take the form of Alpha Beta testing. Whereby a member of the development team will be selected to perform a set of tests for the system. Both manual & automatic input.

### Environmental Needs

Visual Studio 2017 (VS) will be used throughout testing. Test cases will be created, using VS Testing facilities.

All testing will be conducted on only one desktop computer, with relevant security access to this machine.

The desktop computer will have the necessary hardware to run the latest version of Visual Studio:

* Windows 7 SP1 (with latest Windows Updates) or Newer OS
* 1.8 GHz or faster processor. Dual-core or better recommended
* 4 GB of RAM recommended
* Hard disk space: 40GB
* Video card that supports a minimum display resolution of 720p (1280 by 720);

### Staffing/Training

Due to the relatively small size of the system, only one person will be needed for testing. They will be fully versed in all appropriate testing requirements, needing little to no training needed.

### Schedule

All testing will be done in 2 days. Commencing 20/11/17 and ending end of 21/11/2017.

### Risks & Solutions

A potential risk is the implementation being corrupted. As a protection to this, the solution used for testing will be a copy.

## Test Cases

### Testing each message type

Each message type will be tested by having the system read from a file of messages and process each accordingly. This will be accomplished by using automated Unit testing facility of Visual Studio. As each message type is tested here. This will also satisfy System Testing.

#### SMS

##### Input

* S987654321,+87852458714,LOL! it's spongebob ofcourse :)
* S123456789,+4474813025,Who Live’s in a pineapple under the sea?
* S7412589638,+2244588965,This is how we do it
* C745896321,+019632547,MUSM xx

##### Expected output

* Saved to Messages JSON file with LOL! Showing its unabbreviated form: Laugh out Loud
* Saved to Messages JSON file
* Fails to save as message ID contains 11 characters
* Fails to save as message ID Begins with C Not S

#### Tweet

##### Input

* T147852369,@JimmyChunga, OMG Amazing last night
* T123456789,@Thundercats, Mumm-ra is beaten again #Lion-oh
* T987654321,@TripleH, I am the game! #Rules D/L #Rules #Rules
* T9632587418,@Laughs, NFM OO n00b #let'sdothis #Rules #Rules #YES!
* B785147236,@example,LOL #Laughs

##### Expected output

* Saved to Messages JSON file with OMG showing its unabbreviated form: Oh My God and @JimmyChunga being save to MentionsList JSON file
* Saved to Messages JSON file with #Lion-oh being saved to TrendingList JSON File, with count =1, and @Thundercats being save to MentionsList JSON file
* Saved to Messages JSON file with D/L showing its unabbreviated form Download, #Rules being saved to TrendingList JSON File, with count =3, and @TripleH being save to MentionsList JSON file
* Fails due to Message ID being 11 characters
* Fails due to Message ID beginning with B

#### Email

##### Input

* E123456789,test@google.com,testingSubject,The body is here
* E987654321,Jimmy@chunga.com,SIR 18/11/2017,Staff Attack A bee stung Stacy
* E987654321,Jimmy@chunga.com,subject!!,testing body <http://www.loser.co.uk>
* E987654321,JimmyChunga,subject!!,testing body http://www.loser.co.uk
* G7854,testingemail@Hogwarts.ac.uk,no subject, www.hogwarts.com

##### Expected output

* Passes validation and saves to Messages JSON file
* Passes validation and saves to Messages JSON file with subject (SIR 18/11/2017) and centre code (66-666-99) to the SIRList JSON file
* Passes validation and saves to Messages JSON file. The website ( <http://www.loser.co.uk>) is replaced with <Quarantined> and the website is written to QuarantineList JSON file
* Fails validation as email is invalid
* Fails validation as Message ID is only 5 characters and G is not valid

## Test Procedures

The test Procedures for the Unit testing is:

* Write all the input (see above) to the methods in unit testing
* Check the outcome is correct via
* Load up solution in Visual Studio 2017
* Cycle through each message displayed to ensure the above expected output is correct
* Check the all list (Trending, Mentions, Quarantine and SIR) all have the expected information in them

## Test Report

In summary, testing was a complete successful. All expected outcomes were achieved, and no unexpected errors appeared. Being a relatively small application, extensive testing was not needed. Such as using IEEE standard 829-1983.

# Version Control

GitHub will be used to create the version control. Each iteration/addition to any representation of the software system will be added to the repository with a note detailing what was done.

Link here <https://github.com/Jalektor/Mitchell\_Jonathan\_set09102.git>

# Evolution Strategy

1. Instead of writing to a file in JSON format. Storing in a database, with encryption, would be a more suitable option. Stored internally within the system itself.
2. The system should only require minimal maintenance. As a result, maintenance costs would be kept low.
3. Adding additional security in the form of a login page? Stored within database
4. Potentially move the system to be web-based.
5. Increase input types to include Facebook?
6. Integrating anti-virus/spam filtering software to check if URL’s in websites are genuine or not – reduces number of quarantined items.