School of Computing, Napier University

Assessment Brief

1. Module number	**SET09102**
2. Module title	**Software Engineering**
3. Module leader	Zakwan Jaroucheh
4. Tutor with responsibility for this Assessment Student's first point of contact	Zakwan Jaroucheh
5. Assessment	Practical Coursework **Napier Bank Message Filtering Service**
6. Weighting	50% Total
7. Size and/or time limits for assessment	Coursework – 50 hours work
8. Deadline of submission Your attention is drawn to the penalties for late submission	Friday 26 November 2021 15:00
9. Arrangements for submission	Your Coursework must be submitted to the Moodle coursework submission link. Make sure you submit your whole project including the source code, any text output, the video for the prototype demo, and the report in a zipped file. The video demo is required; non-submission of the video demo will result in your coursework to be marked as 0.
10. Assessment Regulations All assessments are subject to the University Regulations.	
11. Requirements for the assessment	See Attached
12. Special instructions	Coursework assistance will be provided during timetabled support sessions from

	Week 8 onwards.
13. Return of work	Formal feedback on coursework will be given within three weeks of submission.
14. Assessment criteria	Your coursework will be marked on Moodle with comments for students to view. The marking criteria are specified in the marking sheets included in the specification attached.

SET09102 – Coursework

This document and the attached scenario covers the coursework submission.

Napier Bank Message Filtering Service

1. Introduction

This coursework comprises 3 parts:

- Software Development Report
- Prototype
- Demonstration

2. Scenario and Requirement

Napier Bank is a medium-sized local bank with many thousands of users. The bank operates from one headquarters and a number of branches. You are required to develop a service, namely Napier Bank Messaging (NBM), which will validate, sanitize and categorise incoming messages to Napier Bank in the form of SMS text messages, emails and Tweets.

2.1 Message Types

The system must deal with three types of message.

All messages are strings of ASCII characters that have a **Message Header** comprising a **Message ID** (Message-type "S","E" or "T" followed by 9 numeric characters, e.g. "E1234567701") followed by the **Body** of the message.

Depending on the message type the **Body** will comprise:

- **SMS messages**
    - SMS message bodies comprise **Sender** in the form of an international telephone phone number followed by the **Message Text** which is a maximum of 140 characters long. The **Message Text** message is simple text but may contain embedded "textspeak abbreviations". Details of the textspeak abbreviations that may be embedded are supplied on Moodle in the form of a CSV file.
- **Email Messages**:
    - Email message bodies comprise **Sender** in the form of a standard email address John Smith john.smith@example.org followed by a 20 character **Subject** followed by the **Message Text** which is a maximum of 1028 characters long. The **Message Text** message is simple text but may contain embedded hyperlinks in the form of standard URLs e.g. http:\\www.anywhere.com. Further detail of email messages is provided in 3.1.2 below.

- **Tweets**
  - Tweet bodies comprise **Sender** in the form of a Twitter ID: "@" followed by a maximum of 15 characters (e.g. @JohnSmith) and the Tweet text which is a maximum of 140 characters long. In addition to ordinary text the Tweet text may contain any of the following:
    - **textspeak** abbreviations (as in SMS above)
    - **hashtags** - strings of characters preceded by a '#' sign that are used to group posts by topic. (such as #BBCClick, #1Donice).
    - **Twitter IDs** as above

3. System Development

You are required to develop a prototype application that will enable the inputting of messages in any of the forms in 2.1 above. The system must detect the message type and output each message in JSON format in a file. You are required to research JSON and identify an appropriate API to allow serialisation in a JSON file. Good places to start are:

https://www.w3schools.com/js/js_json_syntax.asp
https://blog.udemy.com/json-serializer-c-sharp/
https://msdn.microsoft.com/en-us/library/bb410770(v=vs.110).aspx

3.1 Message Processing

Messages must be processed as follows:

3.1.1 **SMS Messages:** Textspeak abbreviations must be expanded to their full form enclosed in "<>", e.g. "Saw your message ROFL can't wait to see you" becomes "Saw your message ROFL <Rolls on the floor laughing> can't wait to see you"

3.1.2 **Email Messages:**

Email messages are of two types: **Standard email messages** and **Significant Incident Reports** that comprise text reports from bank brunch managers concerning incidents of significance that happened during the working day, such as robberies, significant cash shortages, violent incidents. Both types may contain embedded URLs

Standard email messages will contain text. Any URLs contained in messages will be removed and written to a quarantine list and replaced by "<URL Quarantined>" in the body of the message.

Significant Incident Reports will have the **Subject** in the form "SIR dd/mm/yy" and will comprise a message body as above. The message body will begin with the following standard texts on the first two lines:

Sort Code: 99-99-99[1]

Nature of Incident: which will be one of the following (see over):

Theft
Staff Attack
ATM Theft
Raid
Customer Attack
Staff Abuse
Bomb Threat
Terrorism
Suspicious Incident
Intelligence
Cash Loss

Sort Code and Nature of Incident will be written to a SIR list.

Any URLs contained in messages will be removed and written to a quarantine list and replaced by "<URL Quarantined>" in the body of the message.

3.1.3 **Tweets:** Textspeak abbreviations will be expanded (as in SMS messages above). Hashtags will be added to a hashtag list that will count the number of uses of each to produce a trending list. "Mentions", i.e. embedded Twitter IDs will be added to a mentions list.

User Interface

The User Interface (UI) should take the form of a kind of input form(s), e.g. WPF or Java form. For the purposes of testing, messages will be input in the form of the Message Header (in one text box) and a block of Message Body text (in another text box) and redisplayed in appropriate text box(es) processed as specified above. The system must *automatically* identify the message type and process it accordingly.

Ideally the system will also be able to take its input from an input file.

At the end of an input session the system should display the trending list, the list on Mentions and the SIR list.

[1] E.g. 83-19-19

4. Tasks and Submission Requirements

You should complete the following tasks by exercising advanced software development technologies you learnt in this module:

1) Undertake a requirement analysis for NBM. You need to specify the requirement in a Use Case diagram, preferably in USE Case with Soft Goals (NFRs).

2) Produce a class diagram that illustrates the classes required to perform the operations identified in the scenario. Your class diagram should include outline methods and attributes and the relationships among the classes.

3) On the basis of your class diagram develop a WPF application using C# that realises all the functionality specified in Section 2 and 3. Development in other techniques such as Java etc is also acceptable.

4) **Additional Requirement:** Modify your system so that the messages are read from a text file and processed and displayed one-by-one on screen. You can design the structure of this input text file yourself, but it shouldn't be a JSON file.

5) Testing: i) Briefly describe your overall testing strategy for the system. What types of testing will you do, how will you identify test cases? ii) Provide a test plan, which should include Objectives and Scope, Test Items, Tasks and Deliverables, Testing methods, Environmental Needs, possible Tools, Test Schedule, and possible Risks and Solutions. iii) Develop test cases and construct tests to verify that messages are processed correctly for each type of message. Use Visual Studio testing facilities (or equivalence on the platform you have chosen) to conduct your tests where appropriate.

6) Presuming the system is to be developed in agile approach. Propose a plan to use version control to support the development iteration and collaboration among team members.

7) Prepare an evolution strategy for the NBM system. What evolution or maintenance you would predict? What's the maintainability of your system and what are the predicted maintenance costs? What evolution process and methods you plan to use?

You need to submit the following deliverables by the deadline:

S1. Software Development Report. You should describe and justify your requirement specification, system design, implementation, testing and evolution as required in the above task section. Your report should be no less than 8 pages in font size 12 (including diagrams). Feel free to use more pages if you need – we don't set an upper page limit but make sure you only put the relevant material into your report. Zip your report together with your code into one file and submit the file to the Moodle coursework submission link.

S2. Prototype. Please submit your whole project via the Moodle coursework submission link, including the source code, any text output and the report in a zipped file.

S3. Visual Demo. You need to create a short video (5 mins max) of your demo using a screen capture tool. Capture screen activity while recording your audio explaining what your app does. There are many options for this: https://www.techradar.com/uk/news/the-best-free-screen-recorder

Zip everything into one zipped file and submit it via the Coursework Submission link at Moodle. Your pack may include the cw report, source code, any text output, and the video for the prototype demo. Upload it to Moodle under the coursework section.

SET09102 – Coursework Marking Schedule

Element	**Mark**
1. Software Development Report	50
Requirement specification (Complete/mostly correct/does not match scenario/not attempted)	10
Class diagram (Complete/mostly complete/omissions/not attempted)	10
Test plan, test methods, test cases, test outputs and analysis (Complete/mostly correct/does not match scenario/not attempted)	14
Version control plan (Complete/mostly complete/omissions/not attempted)	8
Evolution strategy (Complete/mostly complete/omissions/not attempted)	8
2. Prototype	40
Functionality (Perfect/ reasonable / difficult to follow/not attempted)	25
Look and feel (Overall impression, ease of use)	10
Coding	

(Code format, clarity, comments)	5
3. Visual Demo	10
Total	**100**