

TEXT MAIL – ON DEMAND EMAIL TO SMS SERVICE

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Wijeyaratne Mudiyanseelage Pubudu Ruwanmini Wijeyaratne

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department: Computer Science
Major Program: Software Engineering

May 2012

Fargo, North Dakota

North Dakota State University

Graduate School

Title

TEXT MAIL – ON DEMAND EMAIL TO SMS SERVICE

By

PUBUDU WIJEYARATNE

The Supervisory Committee certifies that this *disquisition* complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Juan Li _____

Chair (typed)

Chair (signature)

Dr. Kendall Nygard _____

Dr. Tariq King _____

Dr. Peter Oduor _____

Approved by Department Chair:

June 28, 2012

Dr. Kenneth Magel

Date

Signature

ABSTRACT

Short Messaging Service (SMS)-based mobile information services have become increasingly common around the world, especially among users with low-end mobile devices. This paper presents the design and implementation of TextMail, an SMS service which allows users to check emails via a text message where email service can be accessed only when the user wants it.

With the development of wireless technology, more people are using their smart phones to access the Internet. However, lots of users, especially in developing countries, cannot afford a smart phone and are still using the low-end cell phones. The aim of this project is to develop an SMS system to enable low-end cell phones to access internet services. In particular, a service called TextMail, allowing users to check emails through text messages even if they do not have Internet access, is provided.

ACKNOWLEDGEMENTS

The completion of this project would not have been possible without the help and support of many people.

First of all, I would like to thank my advisor, Dr. Juan Jen Li, for imparting great knowledge, assisting and believing in me, and giving me hands-on experience and this opportunity. Your guidance has given me the strength to get through many trials in completing this paper.

Second, I would like to thank the members of my supervisory committee for their useful feedback.

I would like to thank the following people for valuable discussions: Prof. M. J. S Wijeyaratne; Mrs. Senani Wijeyaratne; and, especially, Ms. Anjali Perera, who has supported me as a friend, who has been a theoretical advisor and who has been a constructive critic.

Further, gratitude is also extended to the faculty members of the Computer Science Department at North Dakota State University for their assistance.

Last but not least, I would like to thank my parents and my sister. I would not have reached my goals without their limitless support.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
LIST OF TABLES.....	vii
LIST OF FIGURES.....	viii
1. INTRODUCTION.....	1
1.1. Problem Statement.....	1
1.2. Proposed Solution.....	2
1.3. Aim and Contribution.....	3
1.4. Organization of the Paper.....	4
2. BACKGROUND AND RELATED WORK.....	5
2.1. Background.....	5
2.1.1. Methodology Choice.....	5
2.1.2. Database Selection.....	6
2.1.3. Scripting Language.....	7
2.1.4. SMS Communication.....	8
2.2. Related Work.....	11
2.2.1. Google SMS Alerts.....	12
2.2.2. SMS Banking.....	12
3. DESIGN AND DEVELOPMENT.....	14
3.1. System Overview.....	14

3.2.	System Components.....	14
3.2.1.	Mobile Phone	15
3.2.2.	Pure Text API	16
3.2.3.	Call-Back URL	16
3.2.4.	Database Server	16
3.2.5.	TextMail Processor	17
3.3.	Using the TextMail Service	20
3.4.	System Design	21
3.4.1.	Use Case Diagram.....	21
3.4.2.	Sequence Diagram	23
3.5.	Implementation	24
3.5.1.	Validation.....	24
3.5.2.	Checking Email with TextMail.....	26
3.5.3.	Accessing Email with POP and RFC Commands.....	27
3.5.4.	Handling the Length of the Message	30
3.5.5.	Security	31
4.	RESULTS AND ANALYSIS.....	33
4.1.	Project Aim	33
4.2.	Usability Evaluation.....	33
4.3.	Responses of the TextMail Service to the SMS Commands	39
5.	CONCLUSION AND FUTURE WORK	41
6.	REFERENCES	42

LIST OF TABLES

<u>Table</u>	<u>Page</u>
2-1. Wells Fargo SMS commands.....	13
3-1. RFC commands.....	20
3-2. Textmail SMS commands.....	21
4-1. Question list.....	35
4-2. Answer list.....	37

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
3-1. Message passing.....	15
3-2. Use case diagram.....	22
3-3. Sequence diagram.....	23
4-1. Overall analysis.....	38
4-2. Analysis by each group.....	38
4-3. SMS join command example.....	39
4-4. Checking email with Textmail.....	40

1. INTRODUCTION

As time passes, technology is growing and moving faster. The most important and common part of technology in our life is mobile-phone technology. We take a mobile phone with us in everywhere that we go and use it on a daily basis. It is part and parcel of our daily life. Mobile phones have been around for quite some time, but as time go on, mobile phones continuously gain many features. Mobile phones started as simple devices that had only numbers, and most people used them for emergencies only. Now, cell phones have many additional features, such as text messaging, checking email, taking pictures, accessing the web, using as a calculator, etc. The mobile phone is an exceptionally useful tool that advances personal communication beyond all expectations.

Even though most mobile phones have the capability to access emails through the internet, only one-third of American adults (35%) [4] own a Smart phone. The adaptation of email for the rest 75% [4] is not completely straightforward. With a smart phone, email users are able to use a simple email interface to enter the email message and other required fields, whereas with low-end phones, none of these options is possible. In this project, an SMS service that can be used to check emails by sending a text message is developed. The SMS service bridges the gap between low-end cell phones that do not have an internet connection and the need for checking emails anytime or anywhere, even when Internet access is not available and the computing device is not available.

1.1. Problem Statement

There is a need to check emails anytime or anywhere, even when the Internet access and the computing device are not available. An SMS service that enables access to email from low-

end cell phones is a need of the day. Currently, there are email service providers such as Gmail that provide “Push”-based email notification through SMS, where new messages are sent by the server to the mobile device no matter if the user wants them or not. This push-based notification needs the email provider’s support, and it may frequently send emails which bother the user. Therefore, it is necessary to provide an SMS service where email can be accessed only when the user wants it. Such a service should allow the retrieval of emails (or only the email subjects) from the email accounts only when the person needs it. It should also allow anybody to send/reply to emails using the mobile phone through SMS.

1.2. Proposed Solution

To address the above problem, it is proposed to develop an application that can be used to check emails via SMS text messages. This application is named TextMail.

Any mobile phone with SMS capability can be used to access the TextMail service. TextMail uses a pull approach to deliver emails to a user’s mobile phone by allowing the user to access emails only when needed. We have come up with short and easy-to-remember SMS commands that can be used to access various functionalities of the TextMail service. TextMail is supposed to be capable of performing the following functions using SMS commands: read the number of unread emails in the inbox, get a list of unread messages, read the subject or body of a single email, and send an email.

TextMail is a distributed system with six major components working together to cater to its users. These components are further discussed in Section 3.2.

1.3. Aim and Contribution

The primary aim of this project is to design and develop a distributed system which enables email on a non-internet capable phone through SMS text messages.

The steps to achieve the aim of the project are as follows: investigate available SMS APIs currently available; select the most appropriate API using different system-development methodologies to evaluate which is most suitable for this SMS application; formulate an appropriate development process, design, develop, and deploy the SMS service; and, finally, evaluate the final software in relation to the project aim.

From the various SMS APIs that were researched, it was decided that the best option in terms of efficiency and security would be the PureText API.

The development methodology used was the Rapid Application Development (RAD). This selection was made because of the advantage of producing a product quickly with an iterative approach, where emerging requirements can be included at later stages and where the original requirements can be refined as the project continues.

C# was the most appropriate development language because Visual C#.Net was compatible with the MS SQL Server which is one of the best tools to use as data storage. As a result, Visual C#.Net was the best tool to interact with the database.

The Microsoft SQL Server was the DBMS selection because it surpassed other presently available Database Management Systems due to the advantage of a large capacity and high retrieval speed due to the fact that the database would need to store large amounts of data and that the system should be able to retrieve data fast.

ASP.NET was chosen as the preferred server-side scripting/programming technology. The reason for this choice was primarily related to the excellent .NET class library which gives developers the ability to quickly and easily write powerful applications.

Apart from the above analysis, this paper also contains screen designs, use cases, and a sequence diagram which was used to design the system. The layout of this paper is based on the objectives, and hopefully after reading this paper, the reader has a basic understanding of the TextMail service as well as how web and mobile technologies can be used to achieve system objectives.

1.4. Organization of the Paper

This paper is organized as follows.

Chapter 2 provides the Background of the topic and also provides a discussion about Related Work. It also describes the APIs that are used in this project.

Chapter 3 provides information about system architecture, design diagrams, and an explanation of some useful codes.

Chapter 4 provides the Results and Analysis for the application. This includes an analysis of a questionnaire that was distributed among a specific group of people.

Chapter 5 contains the limitations and the future enhancements needed in the application.

2. BACKGROUND AND RELATED WORK

2.1. Background

The three key components of this project are SMS Text messaging, PureText SMS API, and TextMail HTTP Handler. Before going further in this paper, relevant background research has been conducted on these three key components.

2.1.1. Methodology Choice

It is important to follow a specific methodology because it allows the project to be broken down into measurable phases and sub-phases which helps the planning and management of the project as well as allowing a continuous evaluation of how the project is progressing. Many methodologies for system development exist, and it is important to consider some of these various approaches to find the one most suitable for the project because different methodologies offer diverse techniques for different phases and place an emphasis on distinct aspects of development. An investigation has been conducted to find the most suitable methodology to develop this among the Waterfall, Agile Unified Process (AUP), and Rapid Application Development (RAD) methodologies.

During the investigation of methodology choices, it soon became apparent that the Waterfall model would not be suitable because it is too limiting in its approach, and it is the opinion of the author that software development cannot be carried out successfully in one iteration. This leaves two appropriate methodology choices, RAD and AUP. RAD has the advantage of producing a product quickly with an iterative approach, where any emerging requirements can be included at later stages and where the original requirements can be refined

as the project continues. A further advantage of the approach in relation to this project is that different parts of the system can be developed and delivered within given time-boxes, which will provide completion targets for different system modules. The final option was AUP, and although this approach would have been suitable, it would have required tailoring for a one-man development because some of the processes were irrelevant to this project. Furthermore, due to the large number of iterations that take place during each AUP phase, there would not have been enough time to complete and document each iteration. Therefore, it was decided that the RAD approach would be used. The usual disadvantages associated with this approach, i.e., cost and time management, were not issues during the project because it was only the developer's time that was required. Careful planning was done to ensure that time was carefully planned so that the project did not overrun.

2.1.2. Database Selection

There are many different types of database languages that use Structured Query Language (SQL) which has become a global standard for writing, creating, editing, and querying databases. There are currently many SQL Database Management Systems (DBMS) which use SQL, and the syntax between all of them is very similar. Some of these different DBMSs include PostgreSQL [14], Microsoft SQL Server (MS SQL Server), SQLite [15], etc., and they all pose different advantages over each other. For this project, the MS SQL Server has been selected to hold the information which will be received from the users. There are a few reasons for this choice, including the fact that the MS SQL Server is a fast DBMS with many capabilities, that it has the potential to hold huge amounts of data, and that it still runs at a very good speed.

Another benefit of SQL servers is the ease of use and maintenance. Because many people are familiar with Microsoft products, using the visual interface of the SQL server is not as intimidating as other programs [5]; most web servers will run the MS SQL Server as their main DBMS because of the above advantages. SQLite is also a very fast DBMS, but it is lacking some features and is less user friendly than the MS SQL Server. Another option would be PostgreSQL, one of the most powerful SQL DBMSs and a feature-heavy package, which does, however, mean that the speed of this package is not as good as the MS SQL Server; therefore, PostgreSQL is generally not available through many of the ISPs which host websites.

2.1.3. Scripting Language

There are many options which present themselves, the most obvious of which include Java, Python, C/C++, C# (.NET framework), and more. The language which has been chosen to develop this program is C#.Net, and this section of the chapter looks at the reasons behind this selection.

There are several reasons why C# might be preferable to other languages which have been mentioned. Some of the main points are highlighted in the not-so-exhaustive list that follows.

2.1.3.1. C# is secure

C# is one of the first programming languages built with security in mind. C# has many different methods which can be used to help the encryption of data among other security-related issues which were considered in the development of this language. This is a distinct advantage for the SMS system.

2.1.3.2. C# is object oriented

This helps the programmer to visualize a real-world scenario. For example, if the programmer wanted to create a person and assign certain attributes to that person, he/she could do just that. This enables other classes in the project to use this object and its attributes.

2.1.3.3. C# easily communicates across a network

C# has many built-in facilities which make programming across a network a lot easier. This is particularly useful for this project because, as noted above, the project is going to apply SMS technologies through the use of an API which will communicate across the internet. Because C# makes it easier to communicate across a network, it should be simpler to send and receive an SMS request to/from a server. Above all, the developer has a lot of experience with C# as opposed to other languages. To learn another language, such as Python or Java, would take a significant amount of time.

2.1.4. SMS Communication

Given that the chosen method of communication for this application is SMS text messages, there will be a need to do some research about how the application will be able to send and receive SMS messages. This section outlines some of these methods and the basic principles behind how they work.

2.1.4.1. SMS Text Messaging

SMS text messaging plays a key role in this project. Therefore, it is essential to define what SMS is, why and how it is being used, and how popular it really is.

Short messaging service (SMS) is a mechanism of delivering short messages over mobile networks. It is a store-and-forward way of transmitting messages to and from mobile devices. The message (text only) from the sending mobile device is stored in a central short message center (SMC) which then forwards it to the destination mobile phone. This means that, when the recipient is not available, the short message is stored and sent later. Each short message can be no longer than 160 characters. These characters can be text (alphanumeric) or binary Non-Text Short messages [16].

SMS technology is supported by 100% of GSM handsets and is, thus, available for anyone who has a GSM handset. SMS was originally developed for person-to-person messaging; however, it has been widely deployed outside this scope. It is now being used to send SMS notifications for new voice mail, email, and fax messages, allowing remote monitoring of services where an SMS notification is sent out notifying the administrator that a server is running out of resources or that a fault has been detected.

Rapid growth of services and systems that are built around SMS has led to the development of SMS APIs which help developers provide their service to intended users in an easy way. There are many SMS APIs available around the world. Below are some of the APIs that we had time to investigate when searching for a suitable SMS API for our TextMail service.

2.1.4.2. FrontlineSMS

FrontlineSMS is a free piece of software used to send text messages from mobile phones. The software turns a laptop and a mobile phone into a central communication hub. Once installed, the program enables users to send and receive text messages with large groups of people through mobile phones [12]. This piece of software requires a person to plug in the

mobile phone to a laptop, and the software then uses the mobile phone as a hub to send and receive text messages. This means that the text messages sent are chargeable by the mobile phone provider, which is undesirable for a university project.

2.1.4.3. Betavine API

The Betavine API is an open API provided by Vodafone to encourage the development of mobile applications. Betavine allows the downloading of the API in several languages, including Java and Python, which makes it very easy if those languages are used to connect to a Vodafone server and to create a request to send an SMS message to a specified location. A number from which the SMS comes can also be specified so that the replies go to a mobile phone, a home phone, or back to a server which will deal with the information sent in a specific way [3].

2.1.4.4. Online API's

Many APIs exist online for sending and receiving text messages. PureText has released an API to do this [2]. There are also many other online APIs which provide a similar service. Zeep Mobile provides an API for sending and receiving text messages. However, this API is only available in the USA at this time, and it is not reliable based on the previous experience of the TextMail service developers.

In conclusion, after exploring many possible options for sending and receiving text messages from the system, it was decided that the best option, in terms of efficiency and security, would be the PureText API because it would allow the project to proceed by creating the system in an evolutionary way.

2.1.4.5. PureText SMS API

PureText provides an API that can be used to communicate with users through SMS [2]. To start using this API, we first had to create an account on the PureText website.

When creating the account, a call-back URL should be provided (an HTTP handler). When a user sends a message to the PureText Mobile phone number, PureText sends that request to the call-back URL. This application uses the following URL as the call-back URL:
<http://mtext.whatsupcolombo.lk/IncomingHandler.aspx>.

Once all the required information has been submitted, PureText provides an API Token and a Secret Key. These keys should be included in our .Net application in order to do the authorization with the PureText API. When the call-back URL receives a request from the PureText API, the code behind that call-back URL file processes the incoming request and sends a reply back to the users. To send this reply, the application sends an HTTP request to the following URL: <https://www.sendandreceivesms.com/api/>. In the header of the HTTP request, the API Token and the Secret Key must be included.

All messages that should be communicated to the SMS application must be sent to the phone number 701-203-0172.

Ex: to register, send "JOIN" to 701-203-0172.

2.2. Related Work

During the research, we found out that Google SMS alerts and SMS banking systems are fairly related to the TextMail service. Even though there are not any pull-based email-checking SMS services, Google is capable of sending push notifications to a user's mobile phone when a

new email arrives in his/her inbox. Most banks in United States allow their customers to do banking operations with SMS commands.

2.2.1. Google SMS Alerts

Google allows configuration of the inbox to send an SMS alert to a phone whenever users receive a new email. This alert is sent to the users every time they receive new mail, no matter what. Some people find it so annoying; consider the comment a user has put on a Google forum.

"I cannot find anywhere in the settings to disable this alert in gmail. To be honest. I dont even remember where I enabled it. Can anyone provide some feedback on how to get this annoying (and expensive because it's a txt message) alert de-activated?" [1]

2.2.2. SMS Banking

SMS banking is one of the most popular ways of communicating with a bank these days. SMS banking services are operated using both push and pull messages. Push messages are those that the bank chooses to send to a customer's mobile phone without the customer initiating a request for the information. Typically, push messages could be either mobile marketing messages or messages alerting an event for the customer's bank account, such as a large withdrawal of funds from the ATM, a large payment using the customer's credit card, etc.

Pull messages are initiated by the customer, using a mobile phone, to obtain information or perform a transaction in the bank account. Examples of pull messages for information include an account balance inquiry or requests for current information (i.e., currency exchange rates and deposit interest rates) published and updated by the bank [9]. Wells Fargo, Bank of America,

U.S. Bank, and Citi Bank are some of the banks that use SMS banking to cater services to their customers. Table 0-1 shows some of the SMS commands used by the Wells Fargo bank [11].

Table 2-1. Wells Fargo SMS commands

Text Command	Detailed Description
BAL:	Receive the balance of your primary account. Example: BAL
BAL ALL:	Receive the balance of all your enrolled accounts.
ACT:	View account activity for your primary account
ATM (+ ZIP):	Get ATM locations near you. Example: ATM (+ 92126)
DUE:	View credit card payment information. Example: DUE
HELP:	Get Wells Fargo contact information.
STOP:	Disable text banking.

3. DESIGN AND DEVELOPMENT

3.1. System Overview

The architecture of the TextMail service consists of five major components: mobile phone, Pure Text SMS API, call-back URL, SMS processor, and database server. A detailed description about these components is presented later in this chapter. Any mobile phone with a text-messaging capability can interact with the TextMail service. All text messages should be sent to the phone number 701-203-0172. All incoming text messages are first received by the PureText SMS API. The PureText API forwards the incoming message to the call-back URL that was provided during the registration process. When a call has been made to the call-back URL, the code behind the Call Back URL saves the text message and its information to a SQL server database. The TextMail handler processes these requests according to the order they are received. Once a message has been processed, the TextMail handler sends the reply message as an http request to the PureText API. The user's phone number and the message to be sent are added to the header of HTTP request. As the final step, the PureText API delivers the message to the user's mobile phone.

Figure 3.1 shows how messages are passed between the system components when a "check email" command is sent to the TextMail service.

3.2. System Components

This section of the paper discusses the tasks for each component of the TextMail service.

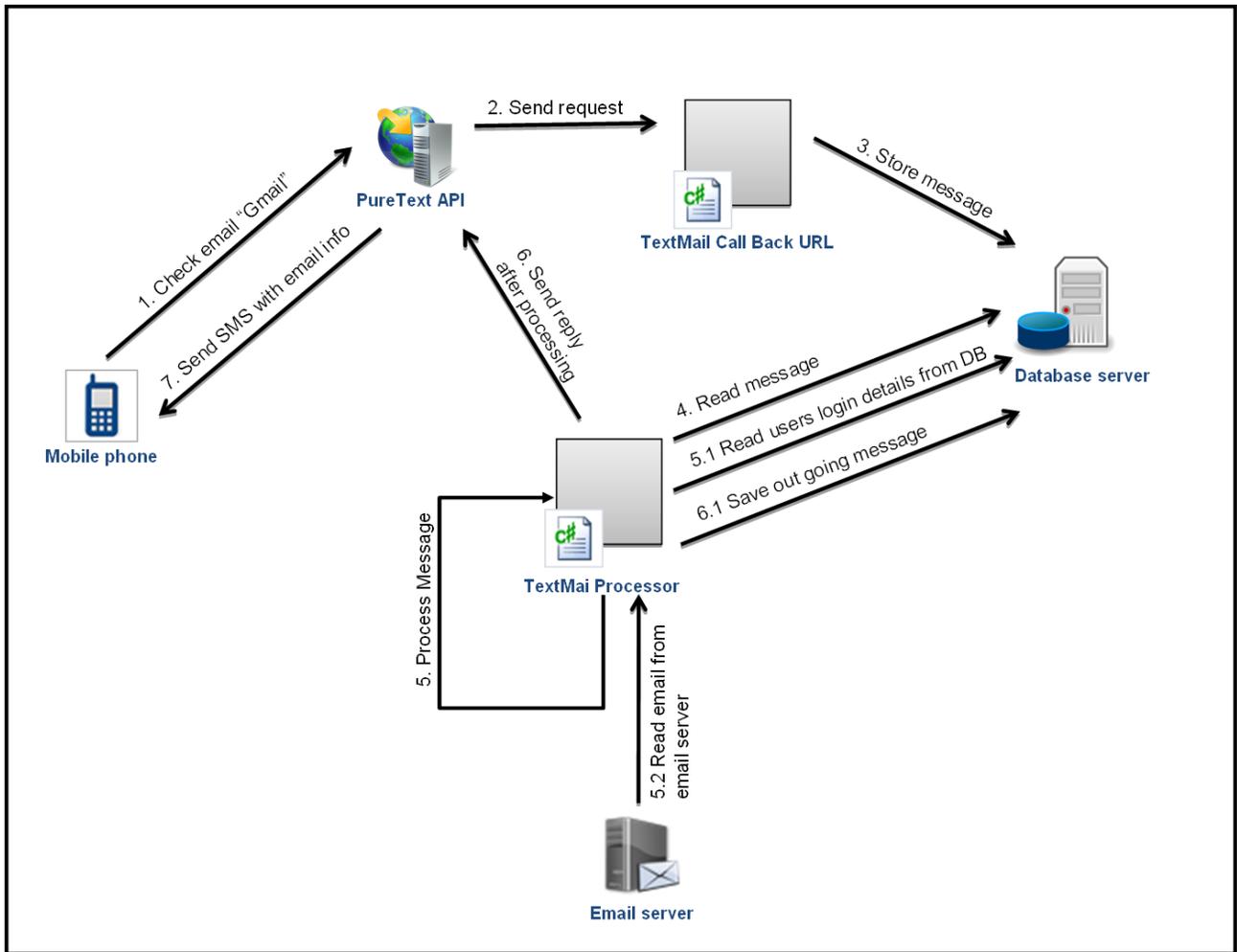


Figure 3-1. Message passing.

3.2.1. Mobile Phone

A mobile phone is the main and only communication interface for users of the TextMail service. A mobile phone is used to send SMS commands to the TextMail service and also to view replies from the service.

3.2.2. Pure Text API

The Pure Text API stands between the mobile phone and the TextMail call-back URL. Users send their SMS commands to the number provided by Pure Text. The API accepts these messages and forwards them to the call-back URL of the TextMail service as an HTTP POST request. From, FromCity, FromState, FromZip, FromCountry, To, ToCity, ToState, ToZip, ToCountry, ApiVersion, Body, and APIToken are passed to the call-back URL as parameters.

3.2.3. Call-Back URL

The call-back URL is an ASP.Net handler (.asmx file) that acts similarly to an ASP.Net web page (.aspx). The call-back URL accepts requests sent by the Pure Text API. The code behind the call-back URL saves most the request information sent by the PureText API to a SQL Server database that resides on a different server than the call-back URL.

Although the Pure Text API sends many parameters, only the ones required for the TextMail service are saved to the database.

3.2.4. Database Server

SQL Server 2008 runs as the database server for the TextMail service. The TextMail database is simple, containing only three tables to store all incoming and outgoing text messages, user phone numbers, and email account details for each user. Simple, stored procedures are created to read and write data to and from the database.

3.2.5. TextMail Processor

The TextMail Processor is the most important and complex component of the TextMail service. It reads the unread messages from the database and processes them. This component is a .Net 4.5 multi-threaded console application. We would like to embrace the word "multi-threaded" because the application does not wait until the processing of one message is done before processing another message. Even though it reads the text messages according to the order they come, the application creates a separate thread for each text message.

The TextMail Processor first validates the sender's phone number. The validation process includes

- Checking whether the account number exists in the database
- Whether the number is activated or deactivated in the database

If the validation process fails, a message is sent back to the number with more information, such as how to register, which SMS codes to send, etc. If the validation process succeeds, the message is processed.

When processing, the message is broken down into pieces to determine the action code. (Detailed descriptions of the action codes are provided in Table 3.1). To determine the action code, the entire message is split from the space character and saved to a string array. To split the message, the application uses the `Split()` function of the .Net framework.

```
string[] messageWords = message.Split(' ');
```

As shown above, the SMS service splits the entire message to the `msgParts` array using the space delimiter. The first element of the array (`msgParts[0]`) is recognized as the action code.

Based on this action code, the rest of the processing is done, and relevant replies are sent.

The piece of code given below describes how the TextMail service differentiates between SMS commands.

```
switch (msgParts[messageParamToLook].ToUpper())
{
    case "JOIN": //Joining service
        break;
    case "ADD": //Adding new email account
        break;
    case "REM": //remove an email account
        break;
    case "CHECK": //checking if new mail in an account
        break;
    case "READ": //read a message
        break;
    case "LIST": //list messages
        break;
    case "SEND": //send an email message
        break;
    case "NEXT": //Pagination
        break;
    case "CODES": //Send all codes to user
        break;
    case "HELP": //send info about a specific code
        break;
    default: // Invalid sms code
        break;
}
```

To read emails pragmatically, the TextMail service uses Post Office Protocol version 3 (POP3) with RFC commands. This section provides information about how TextMail takes advantage of POP3 and RFC.

3.2.5.1. POP3 and RFC

The Post Office Protocol (POP) is designed to allow a workstation (PC) to dynamically access a mail drop on a server host. POP3 is the third version (the latest version) of the Post Office Protocol. In other words, POP3 allows a workstation to retrieve mail that the server is holding for [6]. POP3 transmissions appear as data messages between stations. The messages are either command or reply messages.

There are several different technologies and approaches to build a distributed electronic mail infrastructure. Among them are Post Office Protocol (POP), Distributed Mail System Protocol (DMSP), and Internet Message Access Protocol (IMAP). Of these three, POP is the oldest and, consequently, the best known.

POP3 is not designed to provide extensive manipulation operations for mail on the server; those operations are done by more advanced (and complex) protocols such as IMAP4. POP3 uses TCP as the transport protocol. To access email through POP, it uses RFC commands.

Request For Comments (RFC) is a document that describes the specifications for a recommended technology [7].

Table 3-1 shows the complete list of available RFC commands for POP3 [6].

3.3. Using the TextMail Service

To utilize the TextMail service, users can type any of the following commands (Table 3-2) and send them to the phone number 701-203-0172. To utilize this service, users must enable POP access for the email host.

Table 3-1. RFC commands

Command	Description
USER	Name of user
PASS	Password's users
STAT	Information about messages on the server
RETR	Number of the message to get
DELE	Number of the message to delete
LIST	Number of the message to show
TOP <messageID> <nombredelignes>	Print X lines of the message starting from the beginning (header included)
QUIT	Exit to POP3's server

3.4. System Design

3.4.1. Use Case Diagram

Figure 3-2 shows the use case diagram that was designed for the TextMail service. The use case diagram in Figure 3.2 shows how a mobile phone user (actor) interacts with the TextMail service, including the tasks that can be performed with the TextMail service as a whole. Register, manage email accounts, check email, and send email are the only use cases with which the mobile phone user interacts

Table 3-2. Textmail SMS commands

Code	Description
JOIN	To register your phone number with TextMail service
ADD	ADD <Alias> <email address> <password> To register an email account with the TextMail service. You can add multiple email accounts as long as you give a unique alias for each account. A nickname will be used differentiate your email accounts. We encourage you to use a short nickname for each account.
REM	REM <Alias> To remove an already existing email account
CHECK	CHECK <Alias> To check new messages in a given account, this will give you a list of messages in your inbox
LIST	LIST <Alias> To list all messages in a given account
READ	READ <Message ID> IN <Alias> To read messages in a given account. Message ID will be shown when you type the check-new command.
SEND	SEND <receiver's email address> USING <Alias> <Message> To send a message to an email address. Using the command will recognize which account from which the message should be sent.
NEXT	To receive the continuation of a previous message
CODES	Get all the available SMS commands
HELP	HELP <SMS Command> To get more information about a specific code

The manage email accounts use case includes adding new email accounts to the TextMail service and deleting some existing email accounts from the service. The edit email account option is not given to the users due to the ease of programming, and it also reduces the number of SMS commands that users have to remember in order to utilize the TextMail service.



Figure 3-2. Use case diagram.

The check email use case can also be broken down into sub-use cases. This includes checking how many unread email messages are in the inbox, listing unread emails, and reading a single email.

3.4.2. Sequence Diagram

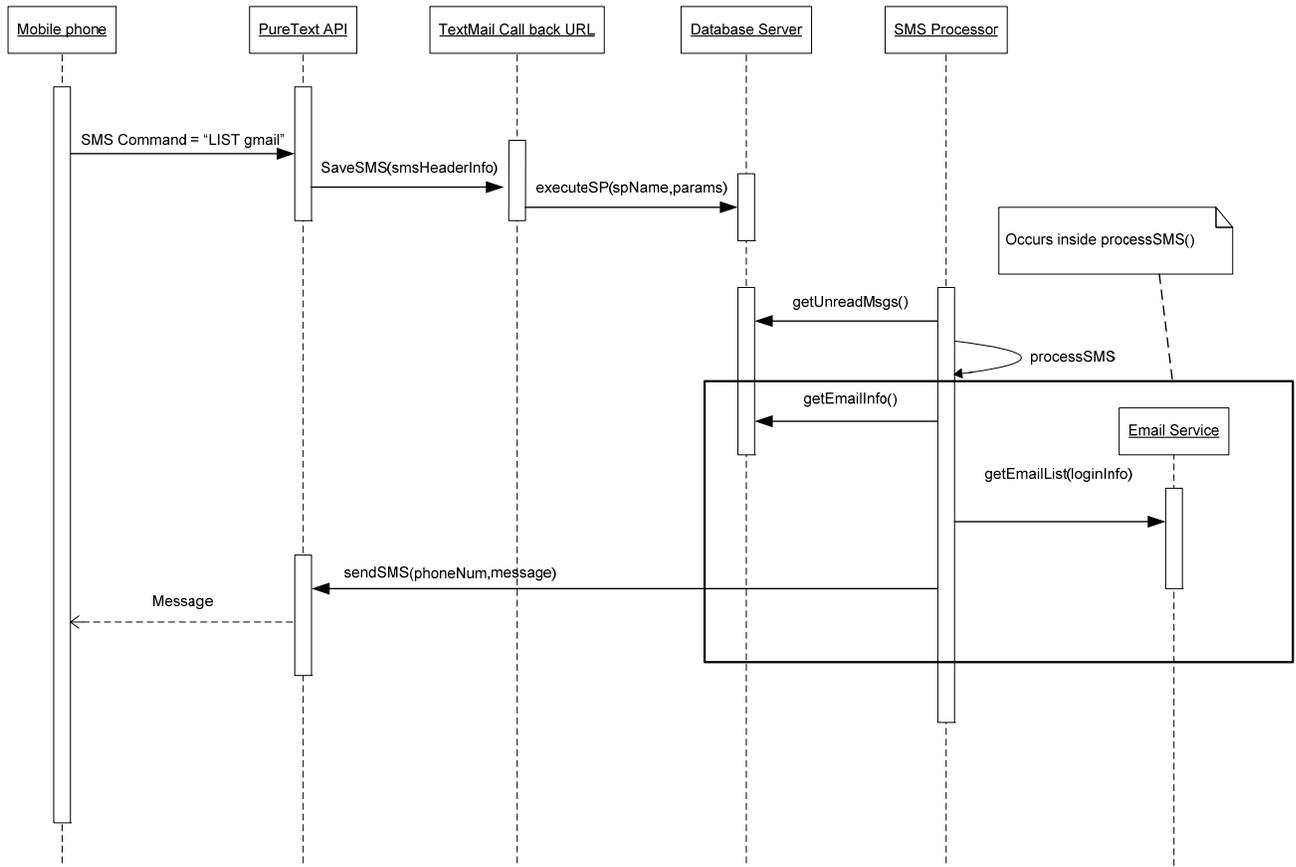


Figure 3-3. Sequence diagram.

Figure 3.3 depicts a UML sequence diagram for the "Check email" use case, taking a system-level approach where the interactions between the actors and the system are shown.

3.5. Implementation

The TextMail service is implemented for only 1100 lines of C# code and uses publicly available parsing libraries. We have not yet conducted detailed benchmarks for our implementation or performed any optimizations. On a 1.8 GHz Core i7 Intel PC with 6 GB of RAM and a 100 Mbps connection, TextMail generally returns results to a user's mobile phone within 10-15 seconds.

3.5.1. Validation

Numerous validation techniques have been implemented to deliver better satisfaction to the users.

3.5.1.1. Validation of phone number

When TextMail receives a message, the first check it does is to make sure that the phone number is registered with the TextMail service. If the phone number is not registered with TextMail, it will not accept all SMS commands from that number. The following piece of code shows how the program handles this situation pragmatically.

```
public string processSMS(string message, string phoneNumber)
{
    user = new clsUser(phoneNumber);
    if (user.isPhoneNumberExists())
    {
    }
    else
    {
```

```

switch (msgParts[messageParamToLook].ToUpper())
{
    case "JOIN": //Joining service
        break;

    case "CODES": //Send all codes to user
        break;

    case "HELP": //send info about a specific code
        break;

    default: // Not registered
        returnVar = cusMsg.getMessage("NOTREG");
        break;
}
}

```

It will only accept the “JOIN,” “CODES,” and “HELP” commands. If the SMS code received is none of the above, the following message is sent back to the phone number: “You are not registered with TextMail! Please type JOIN to join the service.”

3.5.1.2. Validation of SMS codes

TextMail only accepts a certain number of commands. A check has been made to all messages to make sure the SMS code is valid. The valid SMS codes are shown in Table 3.2. If the system receives a command that is outside the given list, it will send a message to the user: “Invalid SMS code. Please type CODES to receive all the available SMS commands.”

3.5.1.3. Validation of email address

Several validations are performed to make sure the email address that the user enters is valid and has not been taken by any other user. A Regular Expression pattern validation is done

to make sure the email address is in the correct format. The following method returns TRUE if the email address follows the correct formatting and FALSE otherwise.

```
public bool isValidEmailAddress(string emailAddress)
{
    string MatchEmailPattern = @"^([\w-]+\.)+[\w-]+|([a-zA-Z]{1}|[\w-]{2,})@" + @"((([0-1]?[0-9]{1,2}|25[0-5]|2[0-4][0-9])\.([0-1]?[0-9]{1,2}|25[0-5]|2[0-4][0-9])\.([0-1]?[0-9]{1,2}|25[0-5]|2[0-4][0-9])\.([0-1]?[0-9]{1,2}|25[0-5]|2[0-4][0-9])){1}|" + @"([a-zA-Z]+[\w-]+\.)+[a-zA-Z]{2,4})$";
    return Regex.IsMatch(emailAddress, MatchEmailPattern);
}
```

If the email address does not follow the correct formatting, the service will send a message back to the user, saying “Invalid Email Address.” If the email address is correct, then the program will check to see if the email address is already taken by another user.

3.5.2. Checking Email with TextMail

Users can add any number of email accounts (e.g., Hotmail, Gmail, or Yahoo) for the TextMail application. The users must provide the login information for the email addresses that they want to access through the TextMail application. The application will collect the following information from the user when adding a new email account:

- Email address
- Email password
- Email prefix (A short nickname for the email account to be used by the system. This should be unique.)

Based on the email address provided, the application will automatically determine to which domain the email address belongs.

When the application detects an action code to check emails, TextMail determines which login information to use based on the email prefix. TextMail queries the database using the email prefix to get this information.

Based on the email domain, an HTTP request is sent to the email server with the required login information. The email server responds if the login is successful or not. If the login is successful, TextMail sends another request, requesting the message information from the email server. Then, the email server replies to the TextMail application with the message information in raw format. TextMail processes the raw message and sends the information back to the user. The communication with the email server and the user continues in this manner.

3.5.3. Accessing Email with POP and RFC Commands

The system will connect to the email host using a `SslStream` with the help of a `TcpClient` object. The host, port number, user name, and password are passed to the SSL stream to do the authentication. If the authentication fails, the mail server sends a response starting with the string “-ERR.” If authentication is successful, it returns “+OK”

The following two methods illustrate how the system connects with the email host. In the `Connect` method, it will just connect to the email server with the host and port number (Ex: `pop.gmail.com` with port 995). After that, it will call the `Login` method by passing the user name and the password.

```
public bool Connect()
```

```

{
    if (Client == null)
        Client = new TcpClient();
    if (!Client.Connected)
        Client.Connect(Host, Port);
    if (IsSecure)
    {
        SslStream secureStream = new SslStream(Client.GetStream());
        secureStream.AuthenticateAsClient(Host);
        ClientStream = secureStream;
        secureStream = null;
    }
    else
        ClientStream = Client.GetStream();
    Writer = new StreamWriter(ClientStream);
    Reader = new StreamReader(ClientStream);
    ReadLine();
    return Login();
}

public bool Login()
{
    bool returnVar = true;
    if (!IsResponseOk(SendCommand("USER " + Email)) ||
        !IsResponseOk(SendCommand("PASS " + Password)))
    {
        returnVar = false;
    }
    return returnVar;
}

```

Once the authentication is successful, the system can take the advantage of using RFC commands. If the user wants to check how many emails he/she has in his/her email, the following method is executed with the RFC command “STAT”:

```
public int GetEmailCount()
{
    int count = 0;
    string response = SendCommand("STAT");
    if (IsResponseOk(response))
    {
        string[] arr = response.Substring(4).Split(' ');
        count = Convert.ToInt32(arr[0]);
    }
    else
        count = -1;
    return count;
}
```

Every time the system receives a response from the email host, it checks whether the response is valid or not. The following method checks it by checking the first few characters of the response.

```
protected static bool IsResponseOk(string response)
{
    if (response.StartsWith("+OK"))
        return true;
    if (response.StartsWith("-ERR"))
        return false;
}
```

```
        throw new Exception("Unidentified server response: " +
response); }
```

3.5.4. Handling the Length of the Message

One of the limitations that SMS has is the length of a message. A message can contain a maximum of 160 characters. Each time a user interacts with the TextMail service, the system replies back to the user. The following question arises: What happens if the reply that is being sent to the user exceeds the 160-character limit?

Inside the program, the developer has set the maximum length of an outgoing message as 103 characters. A check has been made to every outgoing message to see whether it exceeds the maximum-allowed length. If the length is less than 103 characters, the system sends the message without a problem. If the length is greater than 103 characters, the system automatically removes all characters after the 103rd character and appends another string “[Type NEXT for more].” The `adjustMessageLength` method shows how the above description is put into code. Here, `maxMessageLength` is 103 characters.

```
string adjustMessageLength(string text)
{
    if (text.Length > maxMessageLength)
        text = text.Remove(maxMessageLength) + cusMsg.getMessage("NEXT");
    return text;
}
```

By this way, the user receives only 103 characters of the full message, and he/she needs to reply back to the TextMail service with the command “NEXT” if he/she wants to see the rest of the message. The program stores the complete message in a database no matter how long it is.

When TextMail receives the “NEXT” command from a user, the system searches the database for the last sent message for that specific user. The system retrieves that message from the database and removes all the characters before the 103rd character because, the program already sent the first 103 characters in the previous message.

```
string getTheContinuationOfLastMessage(string phoneNumber)
{
    string dataReturnend = "";
    db = new clsDBLayer();
    DataSet ds = db.getData("usp_GetLastSentMessage "+ phoneNumber +');
    foreach (DataRow dr in ds.Tables[0].Rows)
        dataReturnend = dr["Message"].ToString();

    if (dataReturnend.Length > maxMessageLength)
        dataReturnend = dataReturnend.Substring(maxMessageLength);
    db = null;
    return dataReturnend;
}
```

After processing the message, the continuation of the message is sent back to the user.

When sending this message, the same checks will be applied.

3.5.5. Security

Security is an important consideration when developing any software. However, internet-enabled software poses a greater risk because information is being transmitted over an open network. It should be noted that “Software security is a system-wide issue that takes into account both security mechanisms (such as access control) and design for security (such as robust design

that makes software attacks difficult)” [17]. This means that other considerations, such input validation, error handling, quality of coding, and data encapsulation, must be considered as part of the security model.

The system stores email passwords in the database, and it is considered very unsafe to store clear-text passwords in a database in case the database is compromised. Therefore, all passwords are encrypted using the TripleDES hashing algorithm. The TripleDES encryption algorithm is considered to be very secure. It performs three times as much encryption as the standard DES. The method employed here is a "private-key" algorithm that uses one key to encrypt and the same key to decrypt the data.

The Data Encryption Standard (DES) encrypts data using 64-bit or 8-byte blocks and then employs 16 rounds of encryption to every block of data. It uses a key size of 56 bits, 8 bits reduced from 64 bits which serve as parity bits. As mentioned earlier, TripleDES, which performs 3 times the encryption as DES, requires an 168-bit key and encrypts each block three times. That is, each block of data is actually encrypted 48 times, making TripleDES more secure [10].

4. RESULTS AND ANALYSIS

This chapter considers whether the system can be deemed a success. In order to do this, the system that has been created is evaluated against a criterion that examines the original objectives and aims stated in Chapter 1. A usability evaluation is conducted utilizing volunteers to carry out simple, pre-defined tasks to obtain feedback to evaluate the usability of the prototype. Having evaluated the prototype, the overall project management is evaluated, and finally, potential future enhancements are discussed.

4.1. Project Aim

The project aim was to provide a SMS service which allows users to check emails via a text message where the email service can be accessed only when the user wants it. This aim was achieved through the delivery of the TextMail SMS service that allows users to check emails via a text message. To achieve this aim, different system-development methodologies were considered to evaluate which one is most suitable for this project. As a result, the rapid application development was chosen and followed throughout the development. This ensured that discipline was maintained; therefore, deadlines were met, and the project did not overrun.

4.2. Usability Evaluation

Due to the time constraints, it was deemed that targeting 10 people from three different groups for the usability evaluation was reasonable. Three different groups were decided upon because this system can be used by a wide range of groups; therefore, it is important to get a balanced perspective as best can be achieved to fairly evaluate the usability of the system. The three groups were created based on the participants' college. The volunteers from the College of

Science and Mathematics and the College of Engineering and Architecture were considered as having high computing skills. College of Business volunteers were considered as having average computing skills. Finally, College of Agriculture, Food Systems, and Natural Resources; College of Arts, Humanities, and Social Sciences; and College of Pharmacy, Nursing, and Allied Sciences volunteers were considered as having low computing skills. These categories were selected due to the availability of volunteers to take part in the evaluation to evaluate the usability of the system. The volunteers were asked to complete a series of pre-defined tasks with no aid and then to fill out a short questionnaire which was designed using a Likert scale, giving the volunteers five options varying from strongly disagree to strongly agree. The list of pre defined tasks was as follows:

- 1 Send a text to the number 701-203-0172 and follow the instructions (Type CODES to receive all the available SMS commands)
- 2 Add an email account following the ADD command
- 3 Check how many messages you have using the CHECK command
- 4 List all messages in your inbox with the LIST command
- 5 Read an email from your inbox by following the READ command
- 6 Send an email to someone using the SEND command
- 7 Remove the email account you added using the REM command

The Likert scale for the questionnaire is shown in Table 4-1, where 5 is strongly agree and 1 is strongly disagree.

Table 4-1. Question list

#	Question	1	2	3	4	5
1.	Overall I find the system easy to use					
2.	I was able to complete the tasks easily using the system					
3.	Whenever I made a mistake, I find it easy to recover					
4.	The system provides sufficient error handling to help fix a problem					
5.	The system has all the features and functions I expect					
6.	The system response time is acceptable					
7.	The information provided by the system is easy to understand					
8.	The information provided by the system is helpful in completing tasks					
9.	SMS commands are easy to remember and verbiages make perfect sense					
10.	I would happily use this system again					
11.	I would recommend this system to the people I know					

Once all of the volunteers had completed the questionnaires, the marks for each question and user were averaged to determine the average grade for the system. The complete result set is given in Table 4.2. From the results gathered, although the sample size is small, the feedback gained about the usability of the system was positive with volunteers consistently rating most questions as agree or strongly agree. The graph in Figure 4.1 shows that the average score for each answer was around four with some answers being close to five. This suggests that the usability of the system is good with slight room for improvement. The results also show that the system conveys information to the user effectively and that users can become productive with the

system in a reasonable amount of time. From the chart shown in Figure 4.2, it is clear that the different groups found the usability of the system different with the low-computing skills group finding it the least satisfactory. A possible explanation is that volunteers from the College of Science and Mathematics, the College of Engineering and Architecture, and the College of Business tend to be more computer literate than students in other colleges due to technology becoming ever-more present in their daily life. It should be noted that, despite the low-computing skills group finding the usability not as good as the other groups, the mean answer is about 3.8, which is still an encouraging outcome for the evaluation. A possible way to improve this score is to conduct an exercise where feedback can be gathered from the low-computing skills group about how to improve usability and to implement the necessary changes to the system in order to achieve a higher level of usability. By talking with the volunteers after they completed the questionnaire, one problem seemed to arise with the system usability; i.e., volunteers suggested that, rather than showing just the subject of the message, it would be nice to show the entire message body. Another suggestion was to make it possible to read attachments, such as MS Word files, and deliver their content via a text message.

Table 4-2. Answer list

User	Questions										
	1	2	3	4	5	6	7	8	9	10	11
Low computing skills											
1	5	4	5	4	3	4	5	3	2	4	4
2	3	4	4	2	4	5	3	3	3	5	4
3	4	5	4	4	4	3	4	4	2	4	3
4	4	4	4	4	4	4	4	3	2	3	4
5	5	3	4	5	4	2	4	4	2	4	3
6	4	5	4	4	4	5	3	4	2	5	5
7	4	4	3	4	3	4	4	3	4	4	4
8	4	4	4	4	3	5	4	4	4	4	4
9	3	3	3	3	4	3	4	5	3	3	4
10	4	3	4	3	3	4	4	4	2	3	5
Low computing skills	4.00	3.90	3.90	3.70	3.60	3.90	3.90	3.70	2.60	3.90	4.00
Average computing skills											
1	4	5	5	4	3	4	5	4	3	4	4
2	5	4	3	3	3	5	5	5	5	5	4
3	3	5	5	5	4	5	5	5	5	4	3
4	5	5	3	5	5	4	4	3	5	4	5
5	5	5	5	5	5	3	5	5	5	4	5
6	5	5	5	4	5	5	5	3	5	5	5
7	4	4	3	5	3	3	5	3	5	5	3
8	5	4	5	5	3	5	5	3	4	5	5
9	3	5	3	3	4	3	5	3	5	5	4
10	5	5	4	4	5	4	5	4	3	5	3
Average computing skills	4.40	4.70	4.10	4.30	4.00	4.10	4.90	3.80	4.50	4.60	4.10
High Computing Skills											
1	3	5	5	4	4	4	5	4	3	4	4
2	4	4	3	3	3	5	3	5	4	4	4
3	4	5	4	4	3	3	4	4	3	3	5
4	5	4	3	3	4	4	3	4	4	4	5
5	5	5	5	5	4	3	4	4	3	4	4
6	5	5	4	4	5	4	3	4	4	5	5
7	4	5	3	5	3	4	5	5	5	4	4
8	4	4	5	3	3	5	3	3	4	5	5
9	3	3	3	3	4	4	4	4	4	3	4
10	5	4	4	4	5	4	5	4	3	5	5
High Computing Skills	4.20	4.40	3.90	3.80	3.80	4.00	3.90	4.10	3.70	4.10	4.50
Total Mean	4.20	4.33	3.97	3.93	3.80	4.00	4.23	3.87	3.60	4.20	4.20

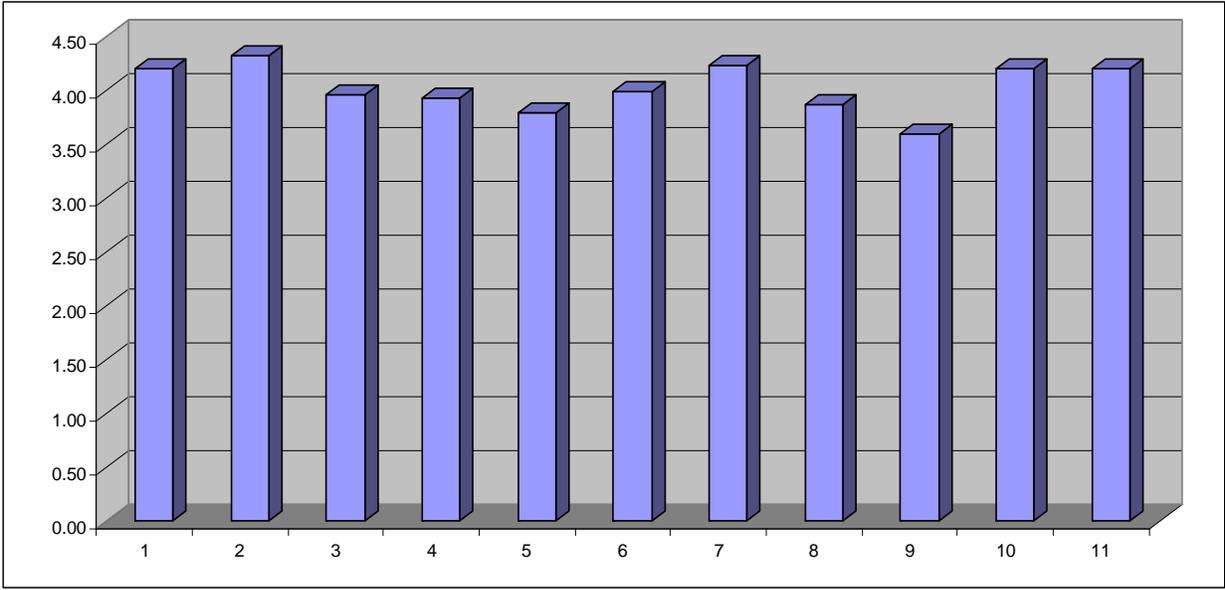


Figure 4-1. Overall analysis.

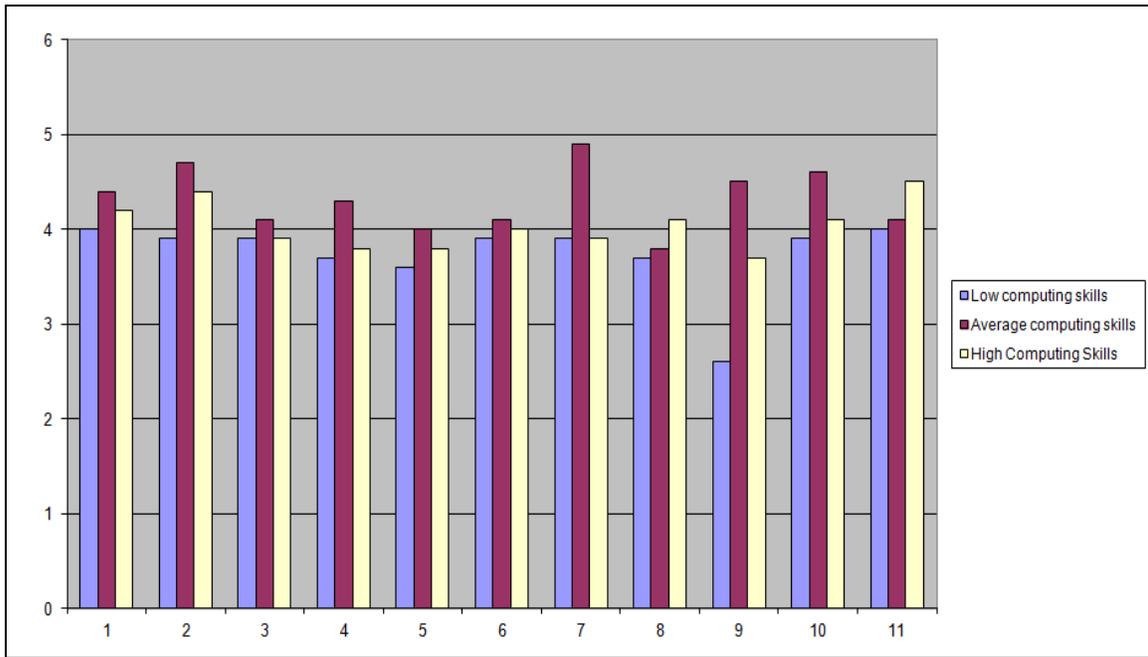


Figure 4-2. Analysis by each group.

4.3. Responses of the TextMail Service to the SMS Commands

The following screenshots (Figures 4-3 and 4-4) are proof that the system responds correctly to the SMS commands that have been sent by a user.

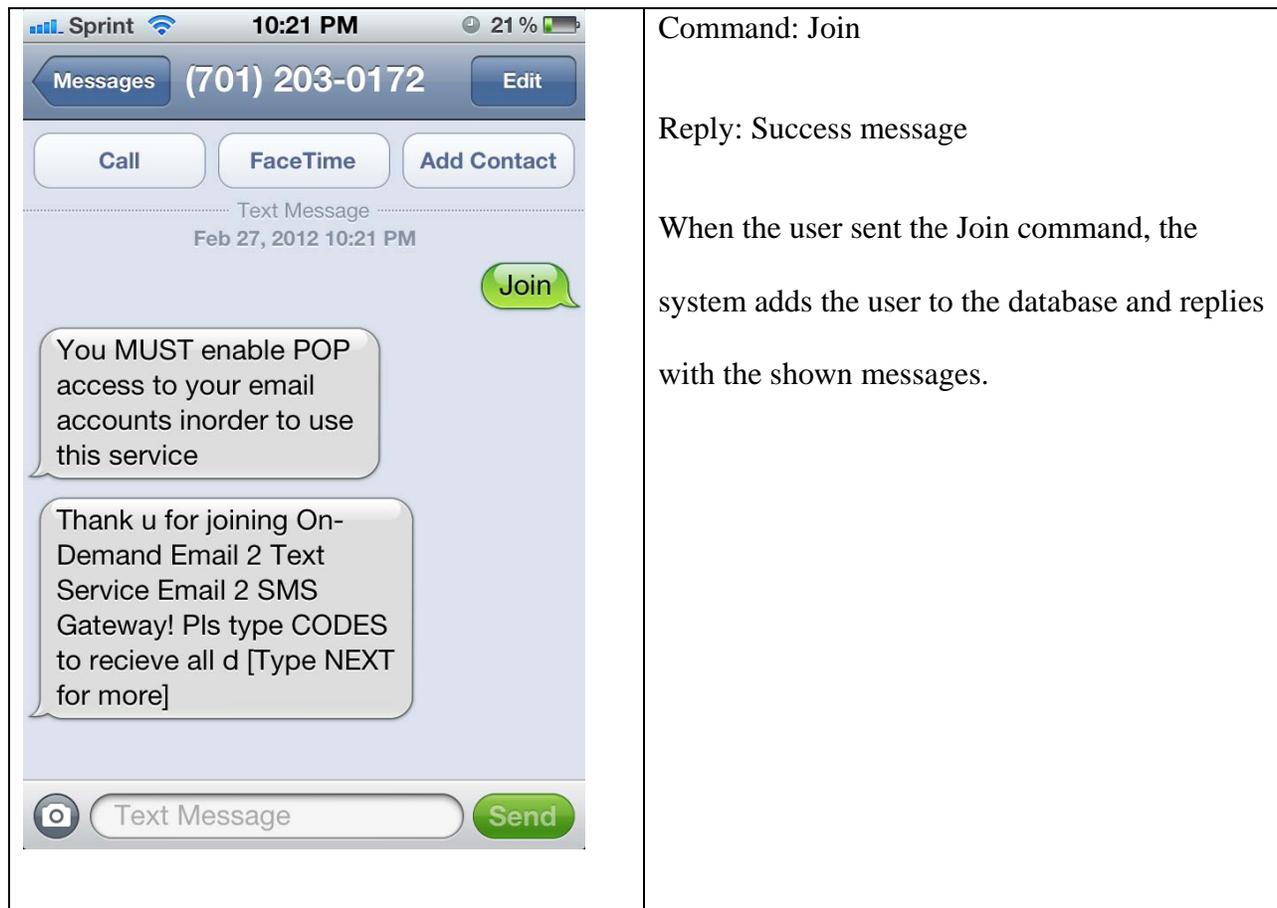


Figure 4-3. SMS join command example.



Command: Check Gmail

Reply: 272 messages

The service will check a user's email to see how many messages are there and will reply back to the user with the number of emails.

Command: LIST gmail

Reply: List of emails in the inbox (Just the FROM field)

The service will list the emails in the user's inbox. Users can send a message "Next" to get the continuation of the list.

Figure 4-4. Checking email with Textmail.

5. CONCLUSION AND FUTURE WORK

The aim of the project was to provide a solution that allows users to check their emails on a "Pull" basis via SMS text messages from a non-internet capable mobile phone. This was achieved with a system that was built and tested. The evaluation proved that the project was a success while leaving the potential for the system to be developed further. The future of the system remains unclear. However, it is possible that the system will be deployed as a commercial application available through mobile service providers such as Verizon, Sprint, etc.

One of the limitations of the TextMail service is that it does not provide the body of the email message due to the constraints of the POP3 protocol. There is no RFC command, which provides the entire body of the message. This limitation will be overcome by further developing the application using a different, newer technology such as IMAP4. IMAP4 was not used in this project due to the developers' lack of experience with the IMAP4 protocol and also due to the lack of documentation available on the internet.

The system only works when the POP access is enabled in the email host. Based on the feedback received from the questionnaire, some non-computer literate people found it hard to enable POP access for their email accounts. Further developing the system with the IMAP4 protocol will overcome this issue.

When users send the "LIST" command to the TextMail service, it will list the emails on the email inbox by showing only the name of the person who sent the email. The system could have sent the subject of the email along with the person who sent it, but due to the SMS character limit, the system will only send the name of the person who sent the email.

6. REFERENCES

1. Google. (2009, January 27). Retrieved February 17, 2012, from Google Mobile Help Forum:
<http://www.google.com/support/forum/p/Google%20Mobile/thread?tid=1537d9b951b3f763&hl=en>
2. PureText. (2011). Retrieved August 4, 2012, from PureText:
<http://www.sendandreceivesms.com/>
3. Betavine. (2011). Retrieved February 4, 2012, from Vodafone:
<http://developer.vodafone.com/>
4. Focus on Smartphone Owners. (2011, August 15). Retrieved January 28, 2012, from Pew Internet: <http://pewinternet.org/Reports/2011/Cell-Phones/Section-2/Focus-on-Smartphone-Owners.aspx>
5. Martin, A. (2008). The Advantages of SQL Server 2008. Retrieved January 20, 2012, from E How: http://www.ehow.com/list_5993393_advantages-sql-server-2008.html
6. POP & POP3. (2006). Retrieved February 14, 2012, from Javvin:
<http://www.javvin.com/protocolPOP3.html>
7. RFC Definition from PC Magazine Encyclopedia. (2011). Retrieved December 28, 2011, from PC MAG:
http://www.pcmag.com/encyclopedia_term/0,2542,t=RFC&i=50510,00.asp

8. Send SMS via Outlook. (2011). Retrieved January 19, 2012, from Microsoft:
<http://www.microsoft.com>
9. SMS Banking. (2011, December 16). Retrieved January 23, 2012, from Wikipedia:
http://en.wikipedia.org/wiki/SMS_banking
10. TripleDES Encryption in .NET. (2006, December 3). Retrieved January 23, 2012, from Code Maverick: <http://codemaverick.blogspot.com/2007/01/tripledes-encryption-in-net.html>
11. Wells Fargo Text/SMS Banking–Commands. (2011). Retrieved January 23, 2012, from Banking 123: <http://www.banking123.com/wells-fargo-textsms-banking-commands/380/>
12. What Can FrontlineSMS Be Used For. (2009). Retrieved December 28, 2011, from FrontlineSMS: <http://www.frontlinesms.com/user-resources/help/what-can-frontlinesms-be-used-for/>
13. Zeep Mobile. (2010). Retrieved September 9, 2011, from <http://www.zeepmobile.com/>
14. PostgreSQL. (2009). Retrieved December 28, 2011, from PostgreSQL:
<http://www.postgresql.org/>
15. SQLite. (2009). Retrieved December 28, 2011, from PostgreSQL: <http://www.sqlite.org/>
16. Gupta, P. (2010). The Short Message Service: What, How and Where. Retrieved January 20, 2012, from Wireless Dev Net:
<http://www.wirelessdevnet.com/channels/sms/features/sms.html>

17. McGraw, G. Software Security. IEEE Security and Privacy, 2(2):81-83, March-April 2004.