MyMail: Development of mail user agent

Aleksandr Goidin

BSc (Hons) Computer Science

19th March 2021

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date: 19th March 2021

Signed: Aleksandr Goidin

Abstract

E-mails are one of the oldest and popular types of Internet communication. Despite its popularity, not everyone knows how e-mail system architecture works. This project aims to build an e-mail client in Java programming language that will support transport and access protocols. The first interest is that MyMail must use self-implemented protocols, instead of prepopulated libraries for Java. Secondly, the interface of the client should be modern looking and interactive. Design decisions and ideas are taken from the research of popular e-mail clients on the market. The project concluded with a working mail user agent and great looking design. MyMail has simple functionality and could be improved by supporting additional features.

Table of content

List of abbreviations

GUI – Graphical User Interface

IANA – Internet Assigned Numbers Authority

IETF – Internet Engineering Task Force

IMAP – Internet Message Access Protocol

MDA – Mail Delivery Agent

MIME – Multipurpose Internet Mail Extensions

MTA – Mail Transfer Agent

MUA – Mail User Agent

POP – Post Office Protocol

RFC – Request for Comments

SMTP – Simple Mail Transfer Protocol

TCP – Transmission Control Protocol

XML – Extensible Markup Language

1. Introduction

Despite the huge popularity of online messengers in the last decade electronic mails, or e-mails how they are usually named, are still one of the most widely used ways of communicating through the web. E-mails are usually associated with a formal way of communication, especially within the organisation, however, their range of usage includes but not limited to contacting friends, family members, professors, or supervisors, applying for a position, requesting information or transferring media files.

There are many possibilities where an e-mail could be sent from or received, but its journey always starts from an MUA, also referred to as an e-mail client. This includes software applications, such as Outlook Express and Mozilla Thunderbird, or webmail services such as those provided by Yahoo!, Microsoft Outlook.com, and Gmail. This paper will only look at and refer to the first type of MUA. Most of these e-mail clients have great functionality, however, the GUI most of them is old-fashioned and doesn't stand alone with modern time.

To simplify the process of creating an e-mail client, there were created external libraries, like JavaMail API for Java programming language. Users can effortlessly implement e-mail protocols and connect to the mail server within a few minutes. Most of the functionality of sending and receiving e-mails is performed using SMTP and IMAP or POP. However, not lots of people know what is actually behind these clients and how the message is transferred from one virtual mailbox to another.

1.1. Motivation

Building MUA and adding a basic functionality might seem like an interesting task. It is an engaging process of understanding how the protocols work and implementing them without the external help of libraries that ease the process. The fact that the project avoids the approach that most other developers use – external libraries, makes it even more inspiring.

In addition to all motivational facts listed above, Terry Gray (2005) states in his article about comparison between IMAP and POP that latter used more often nowadays. In fact, it is easier to implement POP than first protocol, even though IMAP is more usable in nowadays world. Jan Kundrat (2009, p.7) mentioned in the similar project that most of MUAs started only with POP as main client protocol and then adding IMAP few years later. Therefore, it is an interesting challenge to build e-mail client using newer protocol from the very begging and see if it can compete with modern applications.

1.2. Aims

With the problems of a poor GUI in most of the e-mail clients and simplified process of adding basic functionality to them the following aims were set:

1. Understand how the process of transferring e-mail works
2. Successful usage of mailing protocols without the help of external libraries
3. Implement the basic functionality of an MUA to send and receive emails on a local machine
4. Develop an e-mail client in Java with a modern-looking and intuitive GUI

1.3. Document Outline

The remainder of this report is structured to provide material on the design, implementation, and evaluation of the proposed solution. The following chapter looks at the design approach of similar programs and discusses the related project. Chapter 3 describes the e-mail system architecture and design decisions. Chapters 4-6 detail the implementation, testing and evaluation processes. And the final chapter will have the project conclusion along with the personal thoughts of the project author.

2. Background

A number of e-mail users grows daily and results from The Radicati Group Inc. report (2017, p.3) show that there are more than 300 billion e-mails sent and received each day. And as it is known the demand creates supply, and plenty of big technological giants like Google, Microsoft, Mozilla, and others offer their solutions in MUA. Due to the Statista report, some of them are leaders in the number of active users, so it is worth looking at how they look and what they offer. By doing this, it is possible to understand what might attract users and with the help of personal preference try to build a similar-looking GUI.

The goal of this chapter is to analyze existing e-mails clients and identify similar projects. It discusses how do they differ from each other and what can MyMail take from them to fulfil the needs of general functioning and modern looking MUA.

2.1. Similar programs

Gmail by Google

As it was stated in the Introduction (see Chapter 1) some of the MUAs are web-based and Gmail is one of them. It has got to this list because of its huge popularity on the market and over 1.5 billion active users, as Statista says (2021). This might help in making design decisions.

Gmail (see Figure 2.1) has a simple design. Most of the screen is devoted to the list of e-mails or message view, with a minimum of the toolbar and other clutter. It is very intuitive and easy to use, so Gmail is a great example of a user-friendly interface.

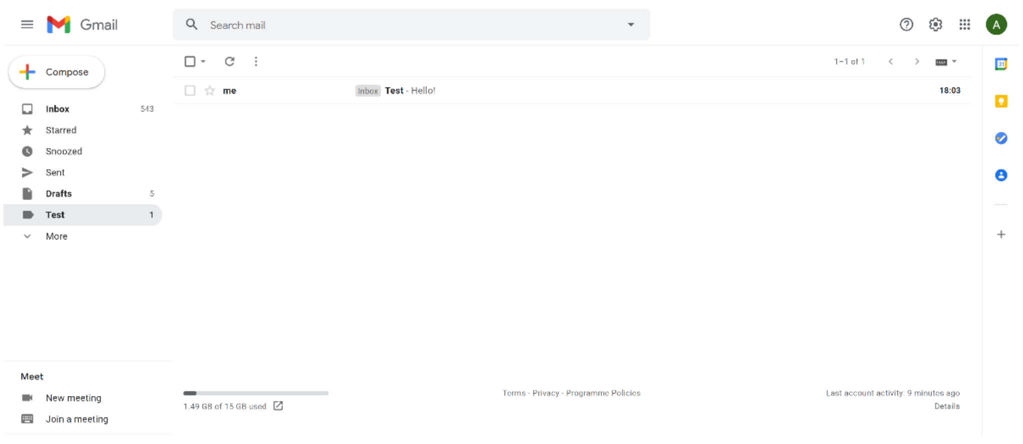
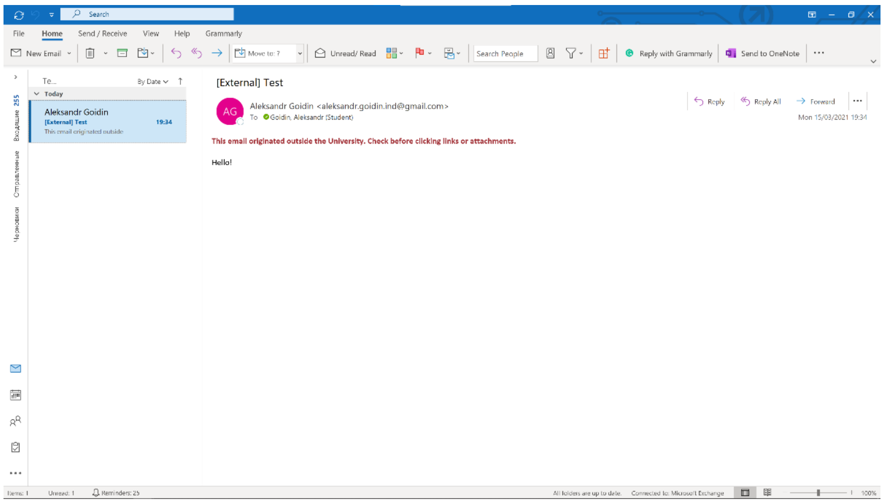


Figure 2.1: Look of Gmail e-mail web client

Outlook by Microsoft

Outlook is another very popular email client. In comparison to Gmail, Outlook offers both – webmail service and desktop application. Outlook (see Figure 2.2) also has a simple design with only two colours. Outlook has a different layout, where the focus is on the content of an e-mail. This helps to read a message and look for other e-mails from the side list at the same time. Unlike Gmail, it is easy to mention that Outlook has many different elements and from the first time it might be quite complicated to navigate and find a way to perform the desired action. On the other hand, Outlook offers a massive range of functionality. Some of them are automatic management of incoming e-mails, customizable e-mail templates, duplicate message deletion and many others. All these features are great and might help to ease reading through a high flow of messages, however, it might be unnecessary and complicated for a casual user.



Aleksandr Goidin
[External] Test 19:34
This email originated outside

[External] Test

Aleksandr Goidin <aleksandr.goidin.ind@gmail.com>
To Goidin, Aleksandr (Student)
Mon 15/03/2021 19:34

This email originated outside the University. Check before clicking links or attachments.

Hello!

Figure 2.2: Look of Outlook desktop e-mail client

Mail by Microsoft

Mail is one more e-mail client application by Microsoft. It is possible to see the similarities between the two products of the same developer. The layout (see Figure 2.3) is message-focused with a side list and a narrow colour scheme. Unlike Outlook, Mail is easy to manage and navigate and from the first look, there are no difficulties in saying what button does what. So, in terms of user-friendliness, it is a great software from where the project software MyMail could take ideas from.

Thunderbird by Mozilla

Thunderbird is well-known and widely used in a past e-mail client and still has 9 million active users. Its interface (see Figure 2.4) looks outdated, however, it is still easy to use.

Although Thunderbird has many features it discourages using it over other solutions on the market.

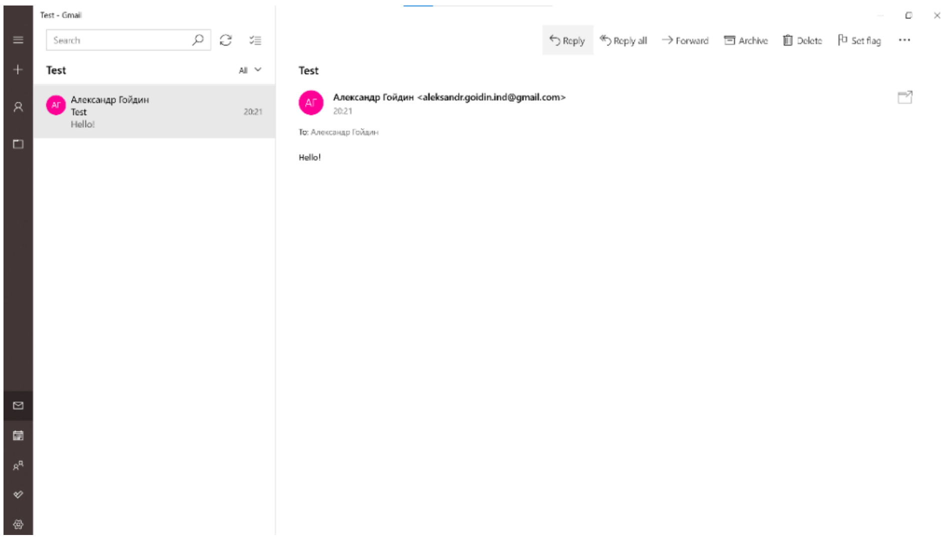


Figure 2.3: Look of Mail desktop e-mail client

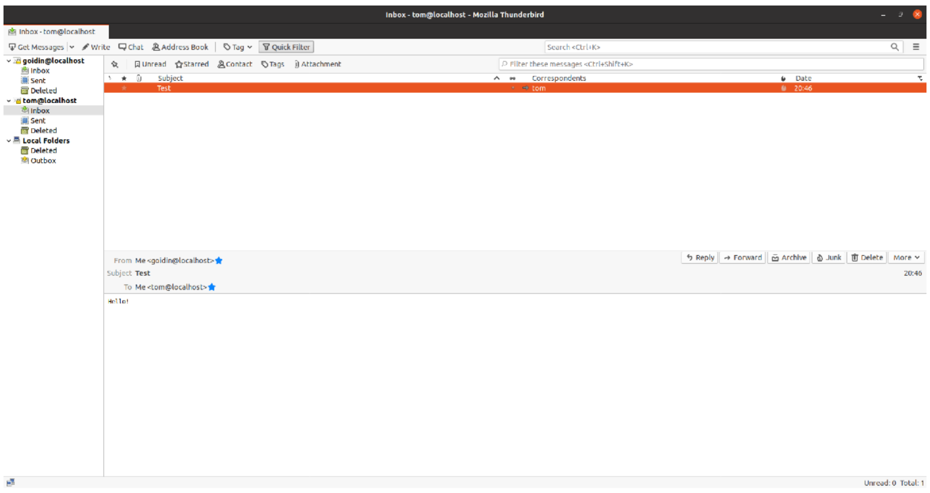
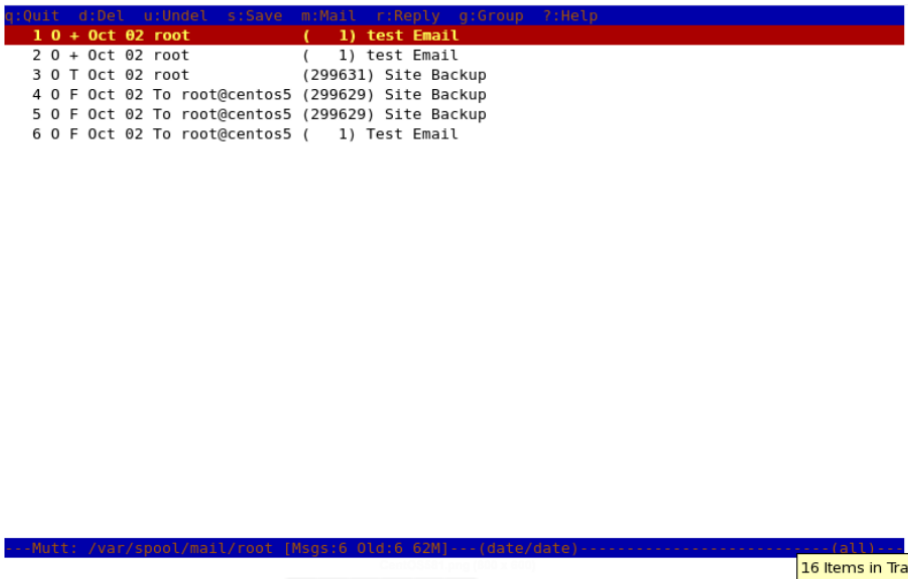


Figure 2.4: Look of Thunderbird desktop e-mail client

2.2. Simple Implementations

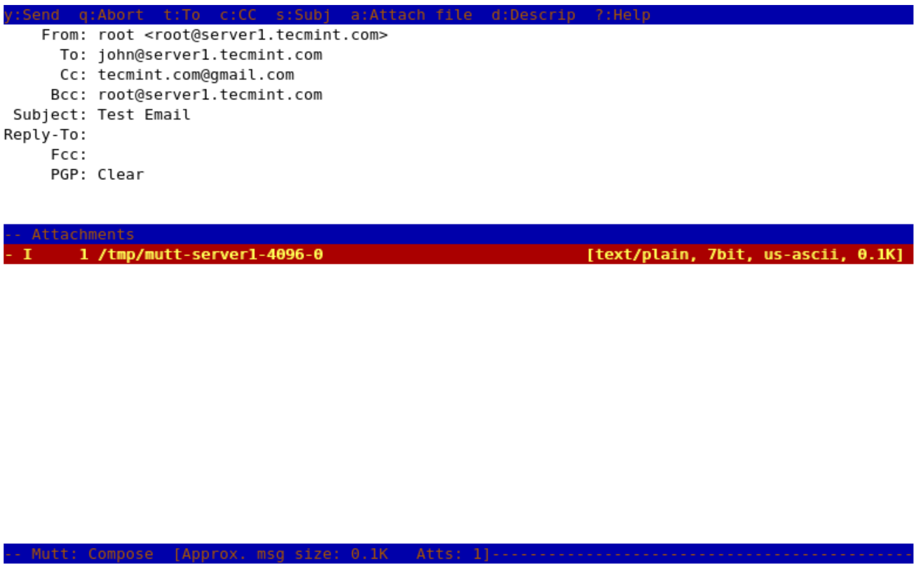
All the MUAs listed above are graphical programs with a huge variety of functions. There are also simple software with a text-based interface. A great and popular example is Mutt, which was first released in 1998 (Elkins, 2021). It is a command-line based e-mail client for UNIX-like systems. It has most of the general features of a modern MUA – view a list of e-mails, send an e-mail with attachments, sort, search, and others. However, it is the least attractive and practical software for a casual user (see Figure 2.5) because the terminal only shows one part of it at a time (see Figure 2.6). In addition to this, it is the hardest in learning and usage as it requires typing commands in the command line to perform some actions. So, Mutt is a great MUA to look up and implement basic functionality just with the addition of a modern GUI.



```
q:Quit  d:Del  u:Undel  s:Save  m:Mail  r:Reply  g:Group  ?:Help
   1 O + Oct 02 root             (    1) test Email
   2 O + Oct 02 root             (    1) test Email
   3 O T Oct 02 root             (299631) Site Backup
   4 O F Oct 02 To root@centos5  (299629) Site Backup
   5 O F Oct 02 To root@centos5  (299629) Site Backup
   6 O F Oct 02 To root@centos5  (    1) Test Email

---Mutt: /var/spool/mail/root [Msgs:6 Old:6 62M]---(date/date)-------------------------(all)---
```

Figure 2.5: Look of Mutt terminal e-mail client (list of e-mails)



```
y:Send  q:Abort  t:To  c:CC  s:Subj  a:Attach file  d:Descrip  ?:Help
    From: root <root@server1.tecmint.com>
      To: john@server1.tecmint.com
      Cc: tecmint.com@gmail.com
     Bcc: root@server1.tecmint.com
 Subject: Test Email
Reply-To:
     Fcc:
     PGP: Clear

-- Attachments
- I     1 /tmp/mutt-server1-4096-0                            [text/plain, 7bit, us-ascii, 0.1K]

-- Mutt: Compose  [Approx. msg size: 0.1K   Atts: 1]------------------------------------------
```

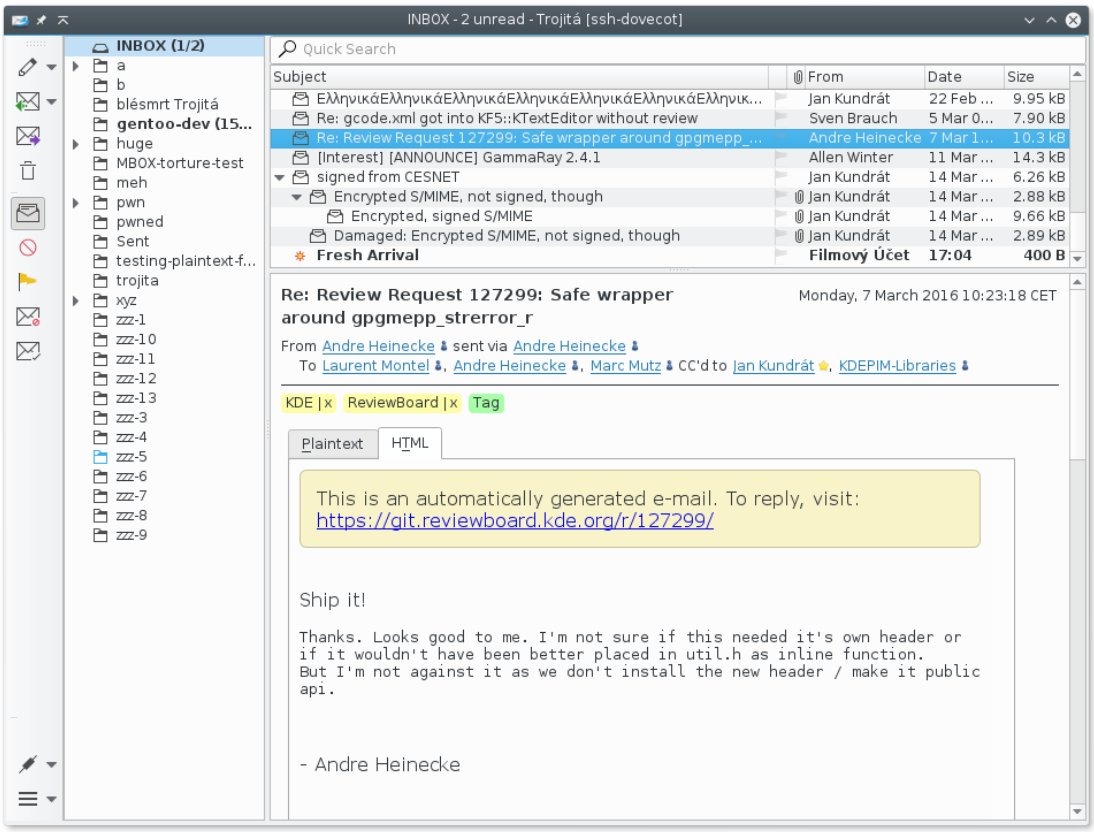
Figure 2.6: Look of Mutt terminal e-mail client (send e-mail)

2.3. Related Project

Trojita was personal development projects and then a thesis topic of Jan Kundrat (2009). As Kundrat stated Trojita was designed to be a stable and high-performing e-mail client supporting IMAP and SMTP protocols. At first release in 2009, it was a great MUA that implemented most of the IMAP features and was able to read Multipurpose Internet Mail Extensions (MIME) messages of all difficulties. Its functionality is great and might even overperform some of the other MUAs. Trojita's success might be a goal for developing of MyMail.

Design and layout are visually similar to Thunderbird by Mozilla (see Figure 2.7) with the list of email and message taking most of the screen. Kundrat documented in the thesis that Thunderbird was an e-mail client of his choice. That might affect on similarities between these two MUAs.

Despite the fact of great implementation and performance, elements in the software look overcrowded and can't compete with modern e-mail clients in design.



Subject	From	Date	Size
ΕλληνικάΕλληνικάΕλληνικάΕλληνικάΕλληνικάΕλληνικάΕλληνικ...	Jan Kundrát	22 Feb ...	9.95 kB
Re: gcode.xml got into KF5::KTextEditor without review	Sven Brauch	5 Mar 0...	7.90 kB
Re: Review Request 127299: Safe wrapper around gpgmepp_...	Andre Heinecke	7 Mar 1...	10.3 kB
[Interest] [ANNOUNCE] GammaRay 2.4.1	Allen Winter	11 Mar ...	14.3 kB
signed from CESNET	Jan Kundrát	14 Mar ...	6.26 kB
Encrypted S/MIME, not signed, though	Jan Kundrát	14 Mar ...	2.88 kB
Encrypted, signed S/MIME	Jan Kundrát	14 Mar ...	9.66 kB
Damaged: Encrypted S/MIME, not signed, though	Jan Kundrát	14 Mar ...	2.89 kB
Fresh Arrival	**Filmový Účet**	**17:04**	**400 B**

Re: Review Request 127299: Safe wrapper around gpgmepp_strerror_r

Monday, 7 March 2016 10:23:18 CET

From Andre Heinecke sent via Andre Heinecke
To Laurent Montel, Andre Heinecke, Marc Mutz CC'd to Jan Kundrát, KDEPIM-Libraries

This is an automatically generated e-mail. To reply, visit: https://git.reviewboard.kde.org/r/127299/

Ship it!

```
Thanks. Looks good to me. I'm not sure if this needed it's own header or
if it wouldn't have been better placed in util.h as inline function.
But I'm not against it as we don't install the new header / make it public
api.
```

- Andre Heinecke

Figure 2.7: Look of Trojita desktop e-mail client

2.4. IMAP over POP

The most important part of MUA takes accessing protocol. That is what manipulates and gives a user access to his/her e-mails. Unlike transferring e-mail where there is only SMTP protocol, e-mail access has a choice of two: IMAP and POP. Dianna and Kevin Mullet (2000) identified POP as the “granddaddy” of mail access protocols . The first version was specified in 1984 in RFC918 (Reynolds, 1984). Even the latest version POP3 (i.e. POP version 3) has only one advantage over IMAP – simplicity. The range of possibilities is narrow, and users are only able to download messages and then decide whether they want to delete or keep them on the server (Heinlein & Hartleben, 2008, p.23).

Nowadays users usually have more than one device with an e-mail account on it, so synchronisation is required to make the process smooth and comfortable. IMAP was designed to solve this problem. A number of studies highlight some of the main advantages of using IMAP over POP3 (Mullet, 2000, Heinlein & Hartleben, 2008, p.28, Lamle, 2009, p.184):

- sorting messages into different folders
- flagging all messages individually
- searching messages for text strings from the server
- downloading only part of the message

These are beneficial features that make MUA very powerful, however, most developers still use POP3 because of its simplicity.

2.5. Summary of research

All researched MUAs have an individual approach to design choices. While Gmail and Mail are minimalistic, others look like “professional tools” filled with lots of features. And in most cases, it is true. Simplicity makes the software available for casual users and Gmail is a great example of it, as it has more users than all other MUAs together.

Both MUAs from Microsoft, Mail and Outlook, have a great layout and give a clear view of the main features by prioritising the mail content section size over others. And at the same time, it is easy to find and navigate through different sections.

Design and implementation choices gathered from the research are described in details in the following chapters.

3. Design

Having looked at the current e-mail client and discussing their strong and weak sides from the usability perspective, this chapter will describe the required information for system implementation. It will briefly explain how the e-mail system work and what is required to consider while developing an MUA.

3.1. E-mail System Architecture

To start with the design of the MUA, it is essential to analyse what are the components of e-mail systems and how messages are transmitted from one client to another. Dianna Mullet & Kevin Mullet (2000) gave a great description of e-mail architecture saying that in general, it consists of agents, mailboxes, and standards.

Agents

Agents are programs that handle messages. There are three main types of agents: MUA, MTA and MDA.

MUA or e-mail client is just an interface between the user and the MTA. MUA fetches messages from mailboxes and sends new messages to MTA.

MTAs or e-mail servers are bridge endpoints in transferring messages from sender server to recipient server. The role of MTA might be described as pushing a message forward through the e-mail chain.

MDA receives a message from MTA and delivers it to the recipient's local mailbox. Some of the software have MDA and MTA bundled together.

Mailbox

Mailbox plays the role of storage of the mail system which contains messages that the user receives. When a user wants to read an email, MUA fetches them from the mailbox.

Standards

Standards are defined specifications of technology or methodology and documented in RFCs by IETF.

3.1.1. Mail transmission process

Suppose a user Bob with the e-mail address bob@mymail.co.uk composes a message using MUA and wants to send it to the user Alice with the address alice@example.com. For Alice to receive the message it should be delivered to Alice's server. As it is impossible for Bob's MUA to send a message to Alice's MTA "directly", the message firstly reaches mymail.co.uk MTA and then transmitted to Alice's SMTP server example.com. All these communications are done via SMTP mail delivery protocol. Once Alice enters his e-mail client, MUA fetches messages from the mailbox via IMAP or POP3 protocol and she can then read them. Figure 3.1 illustrates the mail delivery process.

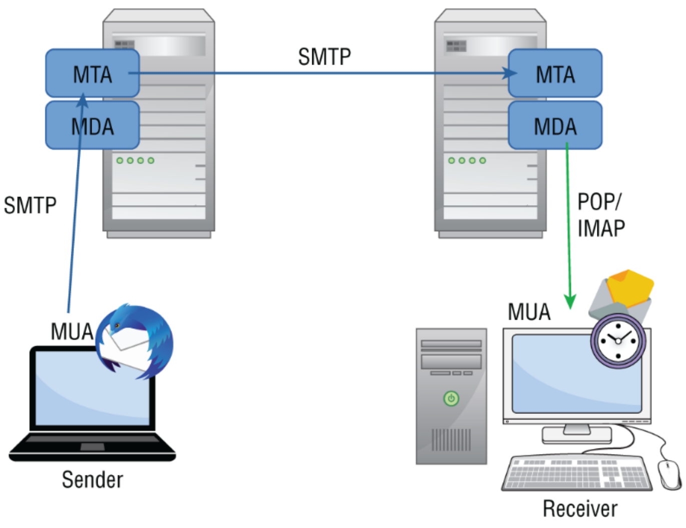
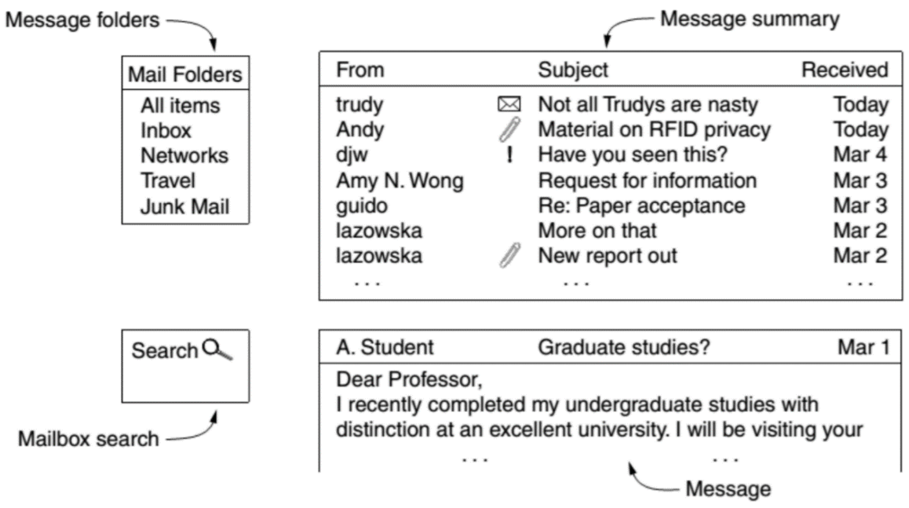


Figure 3.1: Mail transmission process

3.2. Mail user agent

In similar programs (see Chapter 2.1) there were shown a few examples of modern and old e-mail clients. Although they are all different in functionality, design and layout, they have some parts that are common for all of them. Figure 3.2 illustrates all these elements. Message folder shows the existing folders in the user's mailbox, message summary contains the list of messages with the main information about it: sender name/address, message subject and date once it was received; there is often a search bar to manipulate the list, and finally the message content itself.

These are the essential elements that MyMail client should also have to be fully functional MUA.



From		Subject	Received
trudy	✉	Not all Trudys are nasty	Today
Andy	📎	Material on RFID privacy	Today
djw	!	Have you seen this?	Mar 4
Amy N. Wong		Request for information	Mar 3
guido		Re: Paper acceptance	Mar 3
lazowska		More on that	Mar 2
lazowska	📎	New report out	Mar 2
. . .		. . .	. . .

A. Student Graduate studies? Mar 1

Dear Professor,
I recently completed my undergraduate studies with distinction at an excellent university. I will be visiting your
. . . . . .

Figure 3.2: Common elements of all MUAs

3.2.1. MyMail client layout

After the research and comparison of the most popular MUAs on the market (see Chapter 2.1, 2.2, 2.3) the layout and design of Mail by Microsoft is the most enjoyable, modern, and simple looking. Therefore, the layout of MyMail was designed in the same way. Figure 3.3 describes the idea of how elements are positioned in the software. As Mail client, the main focus in MyMail is on message content. This palette is taking most of the user's screen and is responsible for showing the most valuable information: read/send a message, new user log-in. Left wide palette with a search bar on top aim is to list e-mails from the mailbox. The side panel is the menu to manipulate the state of the software. There are not many elements that result in quick and easy usage of the e-mail client.

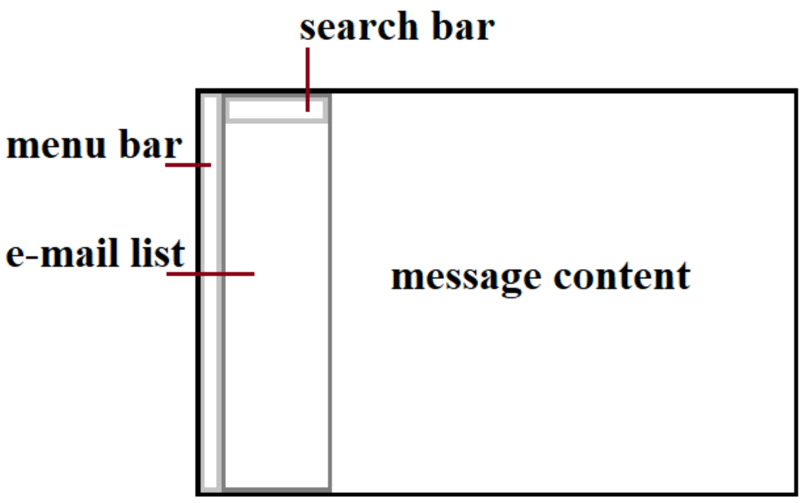


Figure 3.3: MyMail elements layout design

3.3. Message format

The standard format for Internet text messages was defined by David H. Crocker (1982) in RFC822. It defines plain-text messages that consist of an envelope and content (see Figure 3.4). The envelope is constructed by MTA and contains information used by SMTP servers to perform communication between servers. The content itself has a header and a body separated by a single blank line. The first one handles all information about it: sender and recipient address, subject, date etc. The body part is what most users consider being e-mail itself – message text.

This format was designed decades ago, and it had some limitations in what data could be transferred. Some of them were described by Freed & Borenstein (1996, p.3):

- multiple objects in a single message are restricted
- single part message-bodies only
- textual bodies only
- content limited to 1000 characters or less
- use of 7bit US-ASCII characters only

The latest defined message format is RFC5322 has an extended version of header fields with the support of MIME that can transfer non-ASCII text and attachments.

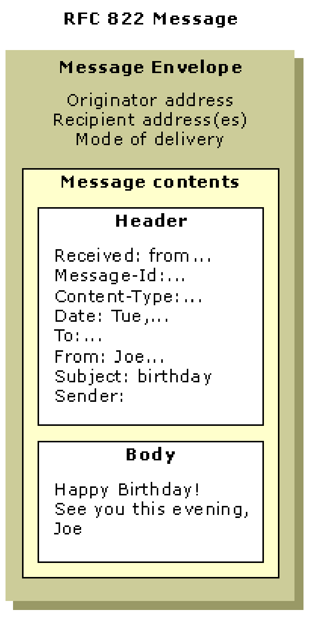


Figure 3.4: RFC822 message format

For an MTA to transmit e-mail, it looks up for the content, which is parsed by an MUA, and specifically header fields. Each header field should be started from a new line and have a unified format: a header field name, semicolon sign, space and value. Table 3.1 lists header fields that normally exist in the header part of the message and used by MTA. Table 3.2 shows some optional fields that can be used by MUA or user.

Header	Meaning
To:	E-mail address(es) of primary recipient(s)
Cc:	E-mail address(es) of secondary recipient(s)
Bcc:	E-mail address(es) for blind carbon copies
From:	Person or people who created the message
Sender:	E-mail address of the actual sender
Received:	Line added by each transfer agent along the route
Return-Path:	Can be used to identify a path back to the sender

Table 3.1: RFC 5322 header fields related to message transport

Header	Meaning
Date:	The date and time the message was sent
Reply-To:	E-mail address to which replies should be sent
Message-Id:	Unique number for referencing this message later
In-Reply-To:	Message-Id of the message to which this is a reply
References:	Other relevant Message-Ids
Keywords:	User-chosen keywords
Subject:	Short summary of the message for the one-line display

Table 3.2: RFC 5322 other header fields

3.4. MIME

To extend an old SMTP e-mail from 7bit ASCII text to multimedia content, a set of MIME standards was defined (Freed & Borenstein, 1996, Moore, 1996). Firstly, MIME standards introduced a structure of message content body part and defined encoding rules. It allows messages to have structured information and data inside them of any length and type. A great example, that most nowadays users might be aware of, are embedded images inside of marketing e-mails.

A certain difficulty is that an MUA must be capable of defining body structure and different types of data that it contains. If some type of data is not specified in MUA, it would require manual encoding from the user to understand it. MIME messages have additional header fields that provide extra information about the message body and data it handles (see Table 3.3).

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

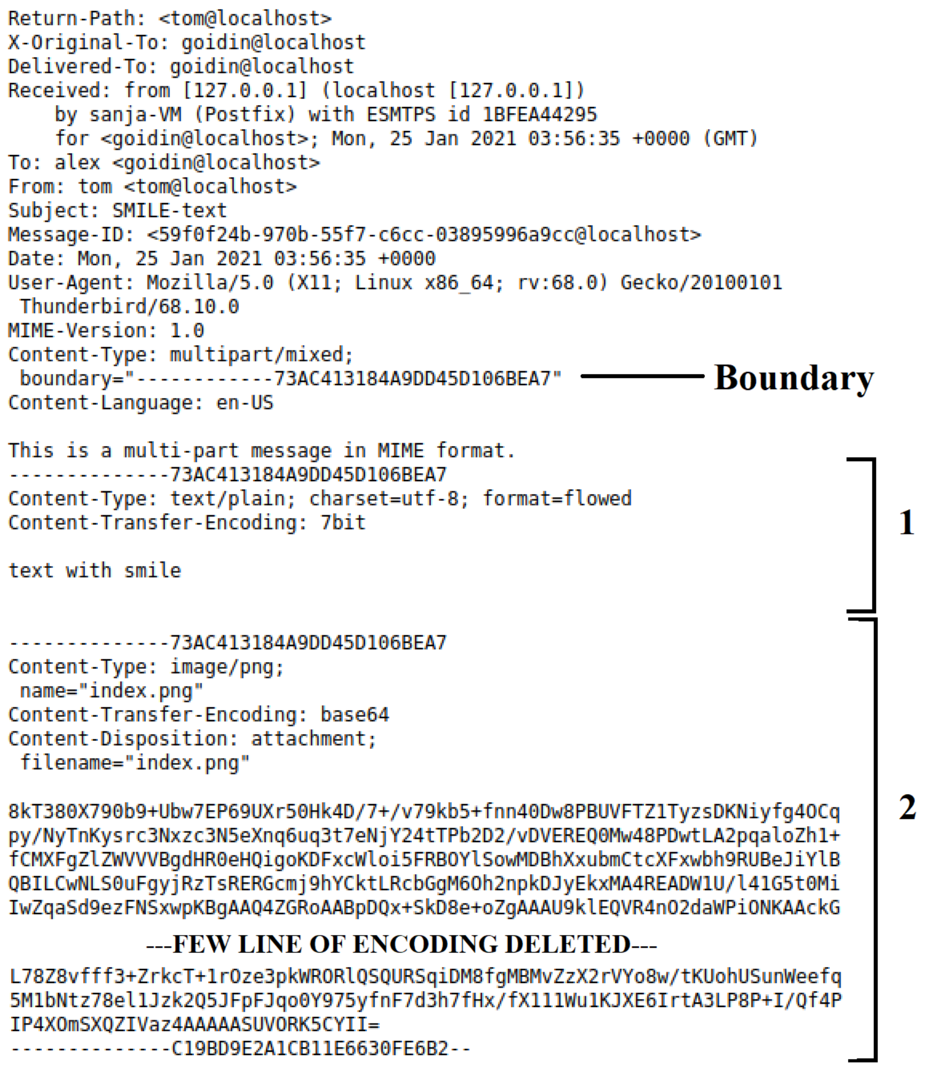
Table 3.3: MIME message headers

The content-type defines how MUA should display a message. It consists of a type, forward slash and followed by subtype. Then, a previously defined RFC822 text message would have a type of *text/plain*. RFC2045 defines 5 main discrete types of content: text, image, audio, video and application (Freed & Borenstein, 1996, p.12). Subtypes are extensions tokens that are registered with IANA (Freed & Klensin & Postel, 1996). There are more than a hundred combinations of pairs type-subtype and only a few e-mail clients support most of them.

The content-type *multipart/mixed* defines that message contains more than one type of data in the message body. MUA must use specified boundaries, or in other words message body parts separators, to navigate through different attachments. Boundaries are defined after all *Content-type: multipart/…* header fields. RFC2046 instructs that each new section must start with a delimiter - two hyphen characters followed by boundary parameter value and a line break. Then section might have additions attributes describing the attachment. A blank line after delimiter indicates the start of attachment itself. The last section closes with a delimiter followed by two hyphen characters.

For MUA to correctly parse the data, message headers should define the encoding mechanism. So, MUA should be able to decode the data based on the value provided. Freed & Borenstein (1996, p.16) recommends having at least two necessary mechanisms: quoted-printable and base64.

Figure 3.5 shows an example of two-part MIME *multipart/mixed* message including *plain text* and *.png* attachment.



```
Return-Path: <tom@localhost>
X-Original-To: goidin@localhost
Delivered-To: goidin@localhost
Received: from [127.0.0.1] (localhost [127.0.0.1])
    by sanja-VM (Postfix) with ESMTPS id 1BFEA44295
    for <goidin@localhost>; Mon, 25 Jan 2021 03:56:35 +0000 (GMT)
To: alex <goidin@localhost>
From: tom <tom@localhost>
Subject: SMILE-text
Message-ID: <59f0f24b-970b-55f7-c6cc-03895996a9cc@localhost>
Date: Mon, 25 Jan 2021 03:56:35 +0000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101
 Thunderbird/68.10.0
MIME-Version: 1.0
Content-Type: multipart/mixed;
 boundary="------------73AC413184A9DD45D106BEA7"
Content-Language: en-US

This is a multi-part message in MIME format.
--------------73AC413184A9DD45D106BEA7
Content-Type: text/plain; charset=utf-8; format=flowed
Content-Transfer-Encoding: 7bit

text with smile


--------------73AC413184A9DD45D106BEA7
Content-Type: image/png;
 name="index.png"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
 filename="index.png"

8kT380X790b9+Ubw7EP69UXr50Hk4D/7+/v79kb5+fnn40Dw8PBUVFTZ1TyzsDKNiyfg40Cq
py/NyTnKysrc3Nxzc3N5eXnq6uq3t7eNjY24tTPb2D2/vDVEREQ0Mw48PDwtLA2pqaloZh1+
fCMXFgZlZWVVVBgdHR0eHQigoKDFxcWloi5FRBOYlSowMDBhXxubmCtcXFxwbh9RUBeJiYlB
QBILCwNLS0uFgyjRzTsRERGcmj9hYCktLRcbGgM6Oh2npkDJyEkxMA4READW1U/l41G5t0Mi
IwZqaSd9ezFNSxwpKBgAAQ4ZGRoAABpDQx+SkD8e+oZgAAAU9klEQVR4nO2daWPiONKAAckG
```

---FEW LINE OF ENCODING DELETED---

```
L78Z8vfff3+ZrkcT+1rOze3pkWRORlQSQURSqiDM8fgMBMvZzX2rVYo8w/tKUohUSunWeefq
5M1bNtz78el1Jzk2Q5JFpFJqo0Y975yfnF7d3h7fHx/fX111Wu1KJXE6IrtA3LP8P+I/Qf4P
IP4XOmSXQZIVaz4AAAAASUVORK5CYII=
--------------C19BD9E2A1CB11E6630FE6B2--
```

Figure 3.5: MIME multipart message format

3.5. E-mail transmission

Movement of e-mail between systems is performed by SMTP. It was firstly defined in RFC821 (Postel J. B., 1982). The protocol has two main functions – e-mail submission and delivery across servers until it reaches destination. For the purpose of this project, only first function will be discussed.

Communication between MUA and SMTP server is established by TCP connection to port 25. Client sends commands to the server and listens to the responses. The main commands are listed in Table 3.4.

Command	**Meaning**
HELO <host name>	Initiate the SMTP session conversation
MAIL FROM <sender address>	Initiate an e-mail transfer
RCPT TO <recipient address>	Specify the recipient
DATA	Asks for permission to transfer the e-mail data
QUIT	Terminate the SMTP connection
NOOP	Check whether the server can respond
RSET	Reset the SMTP connection to the initial state

Table 3.4: Main SMTP commands

To send e-mail to multiple addresses *RCPT TO* command should be called multiple times – once per each new recipient address. After successful response of SMTP server for DATA command, an MUA should send all desired message content like header fields, MIME body structure of just plain text finished by line containing only period. Below is an example of successful communication between e-mail client (C) and SMTP server (S):

C: HELO myserver.com
S: 250 myserver.com

C: MAIL FROM:<user_one@myserver.com>
S: 250 OK

C: RCPT TO:< user_two@myserver.com >
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>

C: “Subject: SMTP connection test”
C: “Hello, user two!”
C: .
S: 250 OK

C: QUIT
S: 221 myserver.com Service closing transmission channel

3.6. E-mail access

The last step in the message transmission process, as shown in Figure 3.1 (page 16), is message access by an e-mail client. From research comparing two existing protocols (see Chapter 2.4) it was clear that IMAP has more functionality and gives a better experience to a user. And it is understandable as IMAP was designed to overbeat the POP.

The latest version IMAP4 (i.e. IMAP version 4) was defined in RFC3501 (Crispin, 2003). Heinlein and Hartleben (2008, p.28) described some of the main features IMAP offers:

- support for subfolders in mailboxes
- full management, sorting and searching of e-mail
- e-mails always remain on the IMAP server, the client keeps only copies
- an e-mail client can retrieve parts of a message when it is necessary, instead of the whole message
- mailboxes are synchronised by the server and can be assessed from multiple clients in parallel
- clients may have both offline and online mode of operation

In the dialogue between client and server, the latter listens to port 143 waiting for the commands and sending back appropriate responses. A command generated by a client must always start with a tag, which might be any random character combination. An IMAP server then responds with a tagged reply to ensure that it is the response to the required command. Successful command completion is followed by an *OK* server reply and if the server failed it would reply with *NO*. *BAD* reply means that server can’t figure out what the client wants, which in most cases is a result of misspelling or wrong command formatting.

Communication always starts with user authentication. The server checks whether it has a user with the provided username and a matching password. This process is similar to what Gmail users have to do before accessing their e-mails. Once the user logged in and accessed there are numerous available commands (see Table 3.5). Users can select, create, delete and rename folders (folders in this context are a more common word for mailboxes as it is used in IMAP4 specification). Selecting stage allows to fetch messages, sort and search by different criteria, mark with flags and it is only a tiny part of full capabilities offered to a user.

Flagging with the combination of sorting is a very powerful feature that most e-mail clients use. Each message could be assigned with one or more flags. For example, all seen messages would have *\Seen* flag, or with the support of an e-mail client, a user could add “importance” to the message with the *\Flagged* flag and then sort the list to show only important messages. Moreover, IMAP allows to add own flags, so with the capable MUA, the user might create *Project_Related* flag stamp all desired messages with it.

Command	Description
LOGIN	Log on to server
SELECT	Select a folder
EXAMINE	Select a read-only folder
CREATE	Create a folder
DELETE	Delete a folder
RENAME	Rename a folder
SUBSCRIBE	Add folder to active set
UNSUBSCRIBE	Remove folder from active set
LIST	List the available folders
APPEND	Add a message to a folder
CHECK	Get a checkpoint of a folder
FETCH	Get messages from a folder
SEARCH	Find messages in a folder
STORE	Alter message flags
COPY	Make a copy of a message in a folder
EXPUNGE	Remove messages flagged for deletion
CLOSE	Remove flagged messages and close folder
LOGOUT	Log out and close connection

Table 8.5: Mostly used IMAP4 commands

In a selected folder *FETCH* command retrieves a specified message. IMAP supports retrieving the envelope, the whole message or even a specific header field of the MIME body section. Below is an example of an e-mail client (C) asking IMAP server (S) to retrieve the subject field of the first message:

```
C: tag1 10 FETCH (BODY[HEADER.FIELDS (SUBJECT)]
S: * 1 FETCH (BODY[HEADER.FIELDS (SUBJECT)] {18}
S: Subject: Smile
S: )
S: tag1 OK Fetch completed (0.001 + 0.000 secs).
```

3.7. Summary

This chapter looked at GUI design decision and various moments that should be kept in mind while implementing the main features of an e-mail client. Implementation of these features will be described in the following chapter.

4. Implementation

Having looked at mail system architecture and design decisions for GUI and requirements for successful communication between e-mail client and servers. This chapter will discuss how different parts of the project were handled and implemented.

4.1. Development Environment

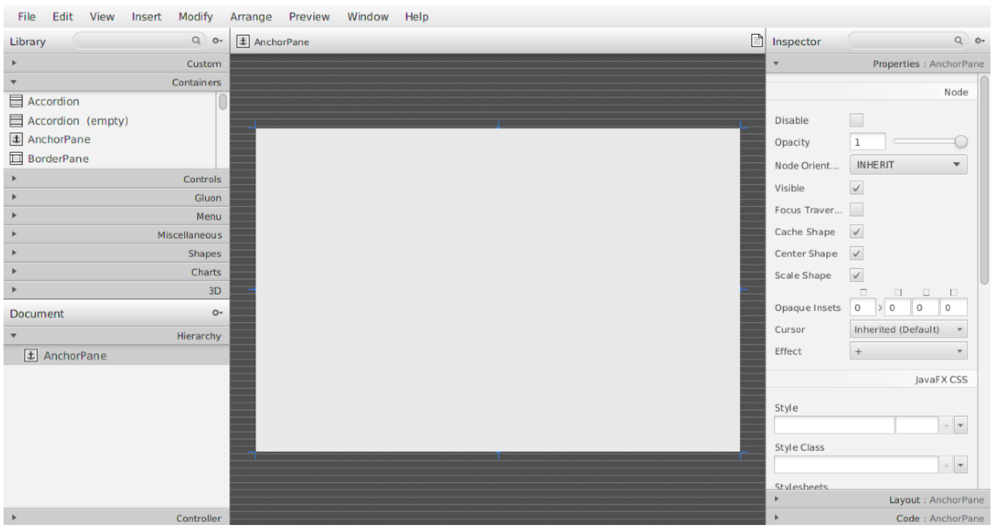
Since the development of an MUA requires an installed SMTP and IMAP server, it was required to use a virtual machine with an installed UNIX-like operating system. Postfix for SMTP and Dovecot for IMAP server is popular software that was chosen because they have great documentation and are easy to install and use (Postfix, Dovecot). For host domain was chosen to set *localhost*.

For programming language, Java was used as it is a popular object-oriented language that greatly fits the needs of this task. It is comfortable for managing different elements in the system and easy to add different features in future development.

For the development of GUI, it was decided to choose the JavaFX software platform over the old Swing widget toolkit. It is great for the development of modern-looking applications with a wide range of functionality. And it is widely used for the development of commercial cross-platform projects.

4.2. GUI

The layout of elements in MyMail was chosen to be similar to Mail desktop app by Microsoft as it was described in chapter 3.2.1. The development of GUI was in parallel with the implementation of functional parts of the system. To construct the visual blocks together was used SceneBuilder platform (see Figure 4.1). It allows placing different containers, choosing desired parameters, assigning them with identifiers and action methods. In most of the cases, there were no requirements to manually write code for placing or editing visual parts of the app.



Disable ☐
Opacity 1
Node Orient... INHERIT
Visible ✓
Focus Traver... ☐
Cache Shape ✓
Center Shape ✓
Scale Shape ✓
Opaque Insets 0 0 0 0
Cursor Inherited (Default)
Effect +

Figure 9.1: SceneBuilder platform for Java GUI development

JavaFX files are written in FXML language and based on XML user interface markup language. It allows having design logic apart from program logic. For the program to interact with the GUI elements controller classes were implemented. They handle identified elements and provide event logic for interactable elements.

Different blocks in MyMail have their FXML file and controller. It makes it easier to control the current state of the program, as JavaFX allows to put Pane (i.e. base container in JavaFX) block from one file inside of element of another. By looking at Figure 4.2, 4.3, 4.4 it is possible to see how one independent block sits inside another.

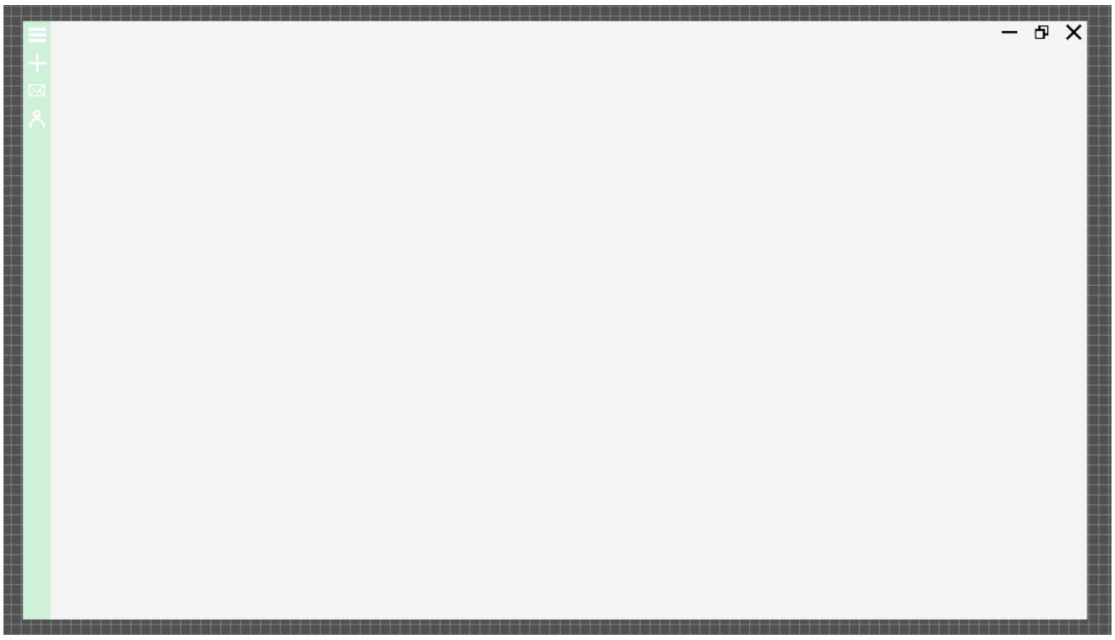


Figure 4.2: Main pane of MyMail app in SceneBuilder editor

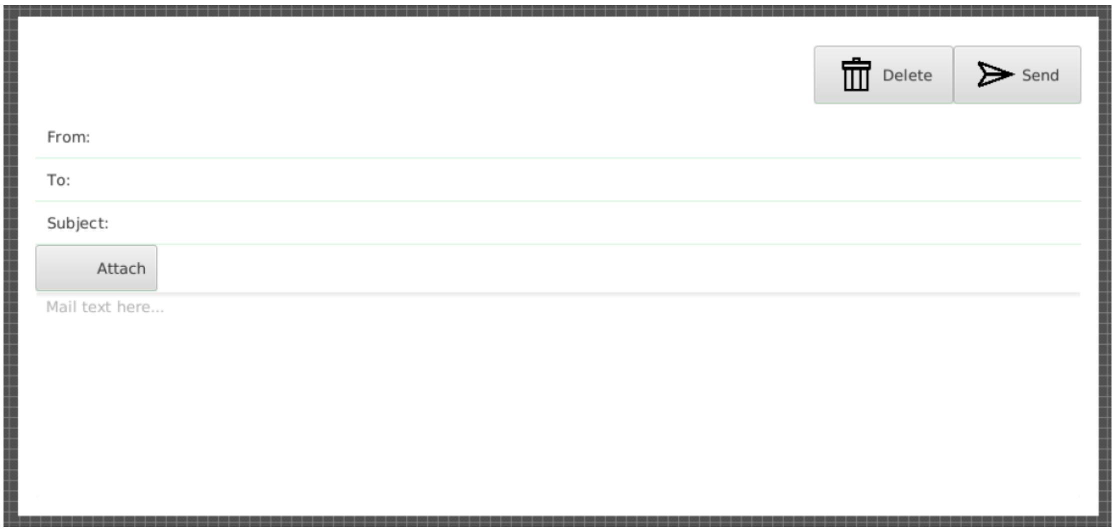


Figure 4.3: New mail pain of MyMail app in SceneBuilder editor

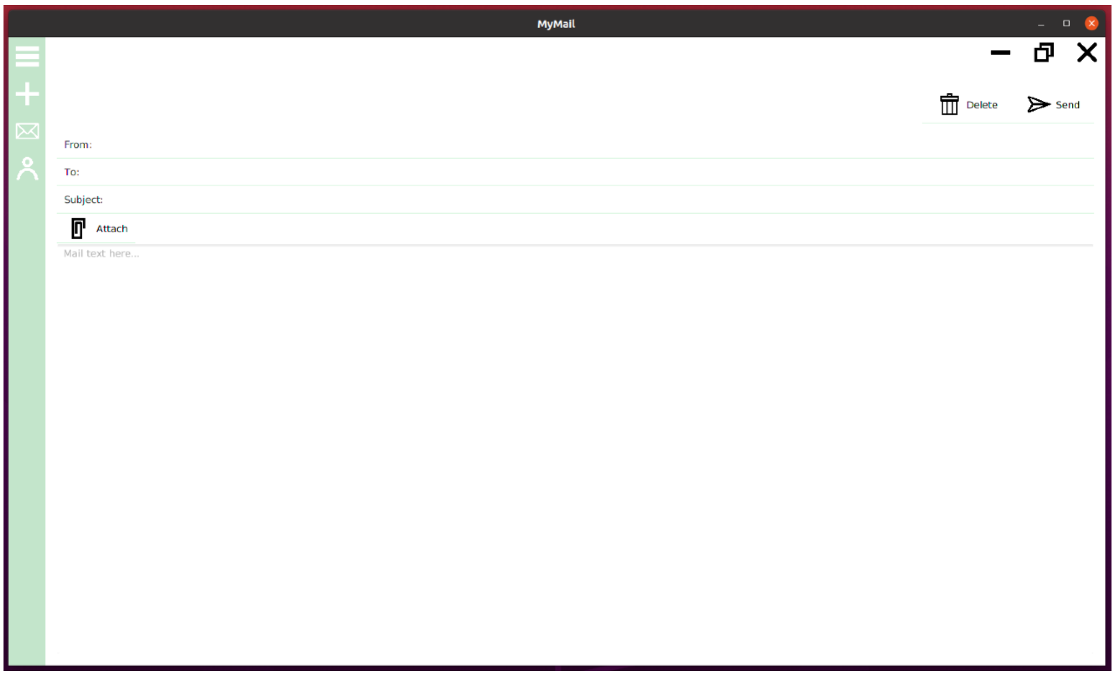


Figure 4.4: MyMail app - new message scene

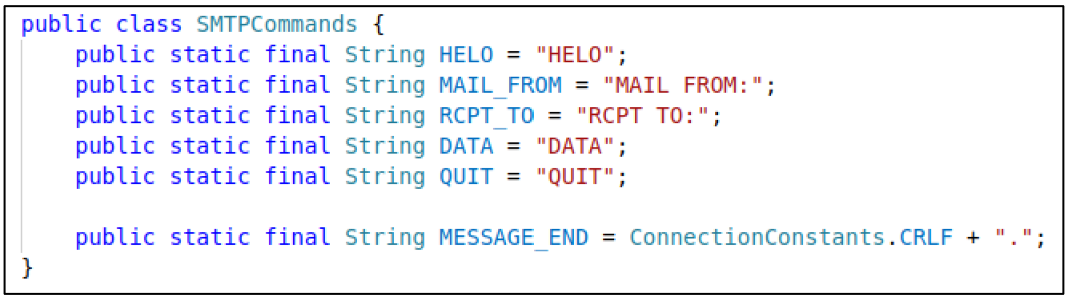
JavaFX also allows applying styles to FXML files and elements. This is process is similar to creating CSS stylesheets while developing webpages in HTML. This allowed to add of different hover effects to clickable elements and made the user interaction more attractive.

Also, the fact that all the graphical elements used in MyMail were made from scratch in Photoshop. This includes icons and background image.

4.3. Send mail

To send a message, the e-mail client initiates TCP/IP connection to port 25 with an SMTP server and starts communication as described in sub-chapter 3.5. Communication with the server, based on commands and user input retrieved from the new message scene. Figure 4.5 shows SMTP commands implemented and used by MyMail and Figure 4.6 shows an example of user input in a new message scene. *Message* class is used to represent message content – header fields and body. Once the message is sent client closes the connection. MyMail specifies the following header fields, apart from basic fields provided by SMTP:

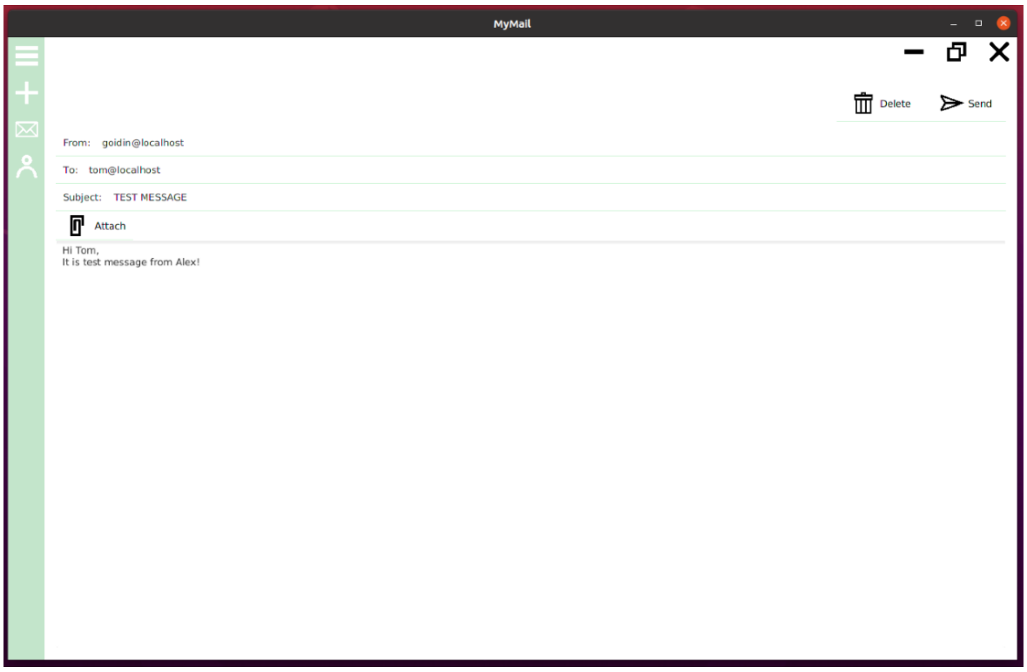
- User-Agent: MyMail/1.0
- Subject: *user-specified subject*
- To: *user-specified recipient address*
- From: *user address*



```java
public class SMTPCommands {
    public static final String HELO = "HELO";
    public static final String MAIL_FROM = "MAIL FROM:";
    public static final String RCPT_TO = "RCPT TO:";
    public static final String DATA = "DATA";
    public static final String QUIT = "QUIT";

    public static final String MESSAGE_END = ConnectionConstants.CRLF + ".";
}
```

Figure 4.5: SMTP commands implemented and used by MyMail



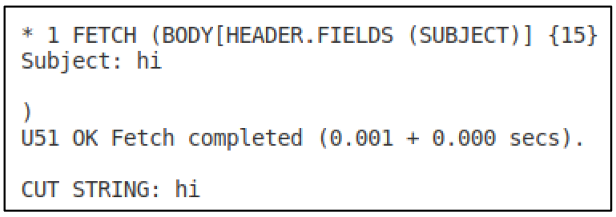
From: goidin@localhost
To: tom@localhost
Subject: TEST MESSAGE

Hi Tom,
It is test message from Alex!

Figure 4.6: Example of user input in new message scene

4.4 Mailbox access

For MUA to interact with IMAP server it first must initiate TCP/IP connection to port 143 and then server listens to next commands from the MyMail. All responses from IMAP server are not in a clear form with only requested information. Required information is a part of response and requires cutting strings to retrieve only useful part. Example of server response for requesting *Subject* field and string cutting is shown on Figure 4.7. The following sub-chapters provide description of implemented functionality based on IMAP commands.



```
* 1 FETCH (BODY[HEADER.FIELDS (SUBJECT)] {15}
Subject: hi

)
U51 OK Fetch completed (0.001 + 0.000 secs).

CUT STRING: hi
```

Figure 4.7: IMAP server response for header field "Subject" and result of string cutting

4.4.1. E-mails listing

Once the *mail list* button pressed MUA sends the command *SELECT INBOX* to go to the default mailbox where all incoming messages are stored. After mailbox selection, MUA retrieves a number of emails using *SEARCH ALL* in the mailbox to create a required number of message list blocks, each representing one e-mail. Then header fields *Subject* and *From* are retrieved individually for each message to add labels to each block. To get header fields MyMail uses the command *FETCH* with the combination of *BODY.PEEK*. It is done to ensure that IMAP doesn't automatically remove \Unseen flag from unseen messages, as it would do with the usual *BODY* command extension.

4.4.2. E-mails sorting and searching

MyMail supports the sorting of messages with standard IMAP flags. Once messages listed, the user can use the drop-down menu at the top section (see Figure 4.8). To retrieve the identification numbers of all e-mail with chosen flags, MyMail sends a *SEARCH* command followed by a chosen flag. To perform searching of e-mails containing typed in text, *TEXT <any text>* is used as a command extension instead of the flag. Then MyMail lists blocks as specified in the previous sub-chapter.

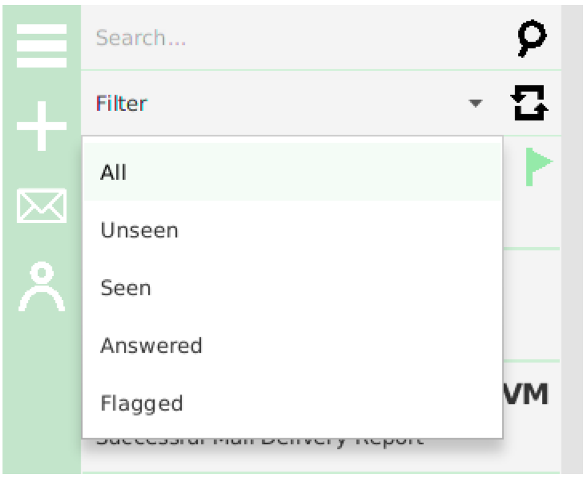


Figure 4.8: MyMail feature of e-mails searching and sorting by flags

4.4.3. E-mails flagging

IMAP server automatically changes only one type of flags – *\Unseen*. Once client fetches message or even part of it, IMAP consider it as "accessed" and deletes the flag. User has an option to set or delete flag *\Flagged* to any message in the list to make it more noticeable or sort messages with higher importance. MyMail send a command *STORE +FLAGS* or *-FLAGS <any flag>* for adding or removing flags, respectively.

4.4.4 Showing e-mail content

Due to the complexity of the MIME message body structure, MyMail can only show plain text message content. Once the user clicks on the preferred message from the list, the client fetches *Subject*, *From* and *Date* header fields as well as text body command extension *BODY[1]*. To ensure that the MIME plain text part will be displayed correctly, *[1]* is used instead of *[TEXT]* or *[RFC822]*. The least two will fetch the whole message body instead of the plain text part if the e-mail will have an attachment. The main screen part appears with all the fetched information to show details to the user.

4.6. Summary

This chapter described the development part of the project. Fully working MUA with rich functionality requires a great knowledge of protocols and even more time for development. Some of the desired features have not been implemented. The main one is sending and reading messages with attachments. Also, some other minor procedures, like account switching, message deletion, reply and forwarding have interface elements. However, they are not functional.

The following chapter will describe testing process of MyMail.

5. Testing

Having looked at the implementation of the interface and basic functionality, this chapter will look at the testing of those. To ensure that the aims and design requirements were met, the testing phase was split into different parts.

5.1. Unit testing

Part of the project included protocols developments. This required most of the components to be worked on independent of each other. So, the process was close to test-driven development. However, the difference is that test units were not automated. Each implemented functionality of the e-mail client was tested manually by ensuring that the required piece of code is running and resulting in the desired outcome.

Two-side communication of MyMail with SMTP and IMAP helped a lot in unit testing. Both these protocols make positive or negative responses, and with reference to documentation (Postel B. J., 1982, p.36, Crispin, 2003, p.63).

5.2. Integration testing

Once enough processes were ready to compose a function, integration tests were performed to ensure that components work and communicate correctly. This mostly included testing the user interface to ensure that FXML files were correctly integrated into the code. For example, that program flawlessly processes user input from text fields and integrates it within SMTP communication.

5.3. Interface usability testing

To ensure that all components of the system front-end behave appropriately manual tests were required. Those included interacting with the system differently. This testing process was done manually and lots of interactions were done in various order to identify potential bugs or, for instance, missing effects. Also, in this stage, additional visual and interaction tests were performed to understand from a user-perspective how easy and intuitive it is to use MyMail e-mail client.

5.4. Overall testing

The last stage included testing the application as a whole. This included methods from interface testing in combination with user input. This phase included sending and receiving emails using Thunderbird MUA to and from the MyMail. These manipulations helped to compare the outcome with fully functioning related software.

5.5. Summary

This chapter discussed testing phases and methods that were used to ensure the smooth and correct workflow of MyMail. The next chapter will include project evaluation.

6. Evaluation

Having looked at the testing methods in previous chapter, this chapter will discuss the results. And whether project requirements that were set at the beginning were met or not.

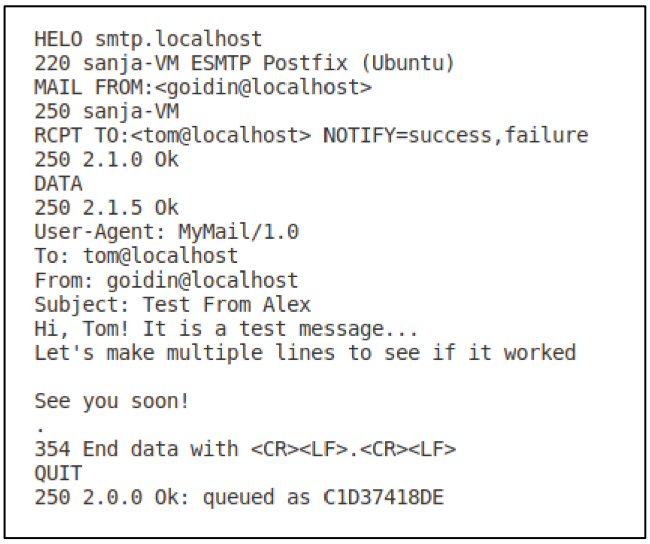
6.1. Use of protocols

The main challenge of the project was to implement two main e-mail transporting protocols. While there were no difficulties in micro problems, lots of them occurred in later stages. The most crucial ones are related to design principles. Because of new knowledge coming every time, the code written in earlier stages was not as extendable and effective as it could be.

The usage of the SMTP protocol follows the standards. MyMail is capable of sending e-mails with the format specified in RFC821. By looking at Figure 6.1, it is possible to see a session with the SMTP server. All replies are successful, and the message delivered to the final MUA, Thunderbird e-mail client, is in the correct format (see Figure 6.2).

The IMAP protocol is more difficult than SMTP and it required more actions to proceed with one request. MyMail has achieved to retrieve different header fields and the body part with plain text. Moreover, the client supports the listing of messages filtered by flags or user text input. Figure 6.3 shows an example of two email lists side by side. The first one shows all e-mails in the mailbox, while the second shows only flagged one.

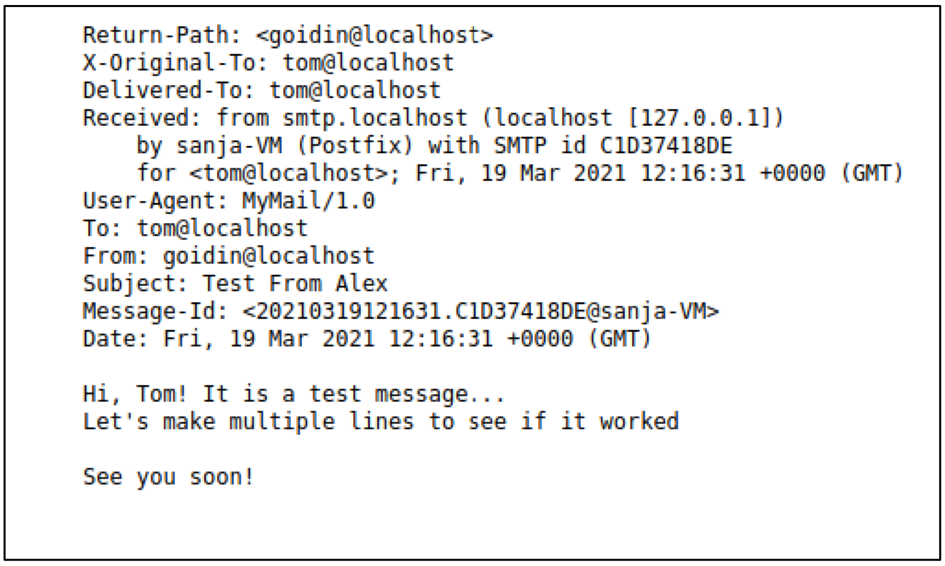
For both sending and receiving MyMail was not able to achieve supporting of MIME messages, in particular sending messages with attachments. The reason for this is the poor design and complex body structure of MIME messages.



```
HELO smtp.localhost
220 sanja-VM ESMTP Postfix (Ubuntu)
MAIL FROM:<goidin@localhost>
250 sanja-VM
RCPT TO:<tom@localhost> NOTIFY=success,failure
250 2.1.0 Ok
DATA
250 2.1.5 Ok
User-Agent: MyMail/1.0
To: tom@localhost
From: goidin@localhost
Subject: Test From Alex
Hi, Tom! It is a test message...
Let's make multiple lines to see if it worked

See you soon!
.
354 End data with <CR><LF>.<CR><LF>
QUIT
250 2.0.0 Ok: queued as C1D37418DE
```

Figure 6.1: Session between MyMail and SMTP server

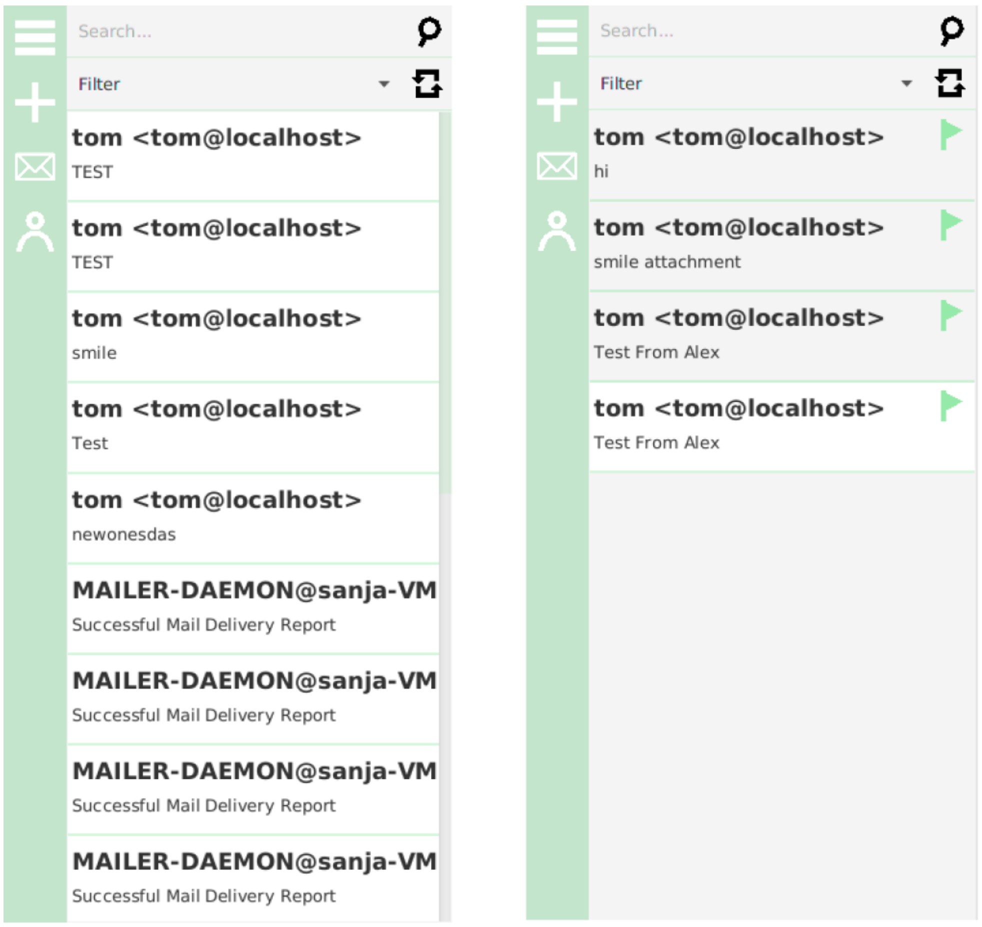


```
Return-Path: <goidin@localhost>
X-Original-To: tom@localhost
Delivered-To: tom@localhost
Received: from smtp.localhost (localhost [127.0.0.1])
    by sanja-VM (Postfix) with SMTP id C1D37418DE
    for <tom@localhost>; Fri, 19 Mar 2021 12:16:31 +0000 (GMT)
User-Agent: MyMail/1.0
To: tom@localhost
From: goidin@localhost
Subject: Test From Alex
Message-Id: <20210319121631.C1D37418DE@sanja-VM>
Date: Fri, 19 Mar 2021 12:16:31 +0000 (GMT)

Hi, Tom! It is a test message...
Let's make multiple lines to see if it worked

See you soon!
```

Figure 6.2: Message content sent by MyMail and received by Thunderbird



tom <tom@localhost>
TEST

tom <tom@localhost>
TEST

tom <tom@localhost>
smile

tom <tom@localhost>
Test

tom <tom@localhost>
newonesdas

MAILER-DAEMON@sanja-VM
Successful Mail Delivery Report

MAILER-DAEMON@sanja-VM
Successful Mail Delivery Report

MAILER-DAEMON@sanja-VM
Successful Mail Delivery Report

MAILER-DAEMON@sanja-VM
Successful Mail Delivery Report

tom <tom@localhost>
hi

tom <tom@localhost>
smile attachment

tom <tom@localhost>
Test From Alex

tom <tom@localhost>
Test From Alex

Figure 6.3: Unsorted list of messages (left) in comparison with sorted list by flag (right)

6.2. GUI

The user interface looks clean. It is capable of being considered “modern-looking”. It supports the idea of simplicity, as other widely used MUAs have, like Gmail or Mail.

It is easy to keep interaction with the app. It has a huge pane for main functions – reading and sending e-mail. The extending sidebar allows to quickly change the state of the app without unnecessary bothering. The light colour scheme and soft lines help to not disturb the eyes for a longer period. Moreover, the GUI was designed the way, that is easy to extend the functionality by adding new tabs or buttons.

6.3. Summary

This chapter discussed which projects requirements were met and what were the difficulties in successful completion of others. The following chapter will sum up the project outcome and provide personal experience of project owner.

7. Conclusion

The final chapter will include the review of whether the projects aims where achieved or not. It will have personal reflections of the project owner talking about experience, difficulties and gained knowledge.

7.1. Review of project aims

The first chapters of this paper clearly defined an aim to understand how the e-mail system works. The problem was that most developers use already existing libraries that do not require detailed knowledge of them. It was required to build an MUA that supports SMTP and IMAP protocols. And the main goal was to allow a user to send and receive messages. Because MyMail supports both functions, but only for text-based messages it would be fair to consider this goal partially achieved.

The second aim was to have a modern-looking and user-friendly interface. MyMail could be considered a modern-looking app. With little disturbing buttons and a correctly chosen layout, it can easily standby with similar products offered on the market. Even by adding extra functional buttons in the future, first-time users could easily navigate through. Therefore, this aim is considered to be fully achieved.

7.2. Personal reflections

As I have stated in the first chapter, a very small number of individuals know how e-mail transportation work. Even, I used to e-mail a few times a daily, I was one of those people. I enjoyed working on this project because it put me on an uncharted topic. This project was challenging at the beginning because it required an unusual method of searching learning material. Instead of common forum reading, in most cases, I had to read old documentations.

I was able to understand the principles behind e-mails transportation. And in fact, implement partially working MUA. Due to the lack of experience, I did not realize the whole depth of the project at the beginning. It was difficult to find the starting point and link two pieces of information. Luckily, with help of my supervisor and personal self-work, I have managed to cope with this pressure.

Moreover, I was able to learn JavaFX as it was required for achieving the project requirement. So, after gaining new skills and knowledge and with the final view of MyMail I think the project is an overall success.

7.3. Future Work

As the goal of the project was fully functional MUA, I would first add support for MIME messages. That would dramatically change the success of the project and the capability of MyMail. I would also work on some small finishes that could fulfil the needs of the user. Some of them are:

- sign-in and sign-up option
- e-mail reply and forwarding
- manipulation of different mailboxes
- encrypted communication

However, I would start by redesigning code logic because it is might be difficult to implement all this with the current state of code. I would create different server parsing, add sessions and more structured way of message bodies.

7.4. Closing remarks

Overall, I am proud of myself. I was able to meet most of the requirements and individually working on research, design and development. Gained skills and knowledge are giving extra motivation to learn more and work on the following projects.

8. References

Oracle (2018) *JavaMail Reference Implementation.* Available at: https://javaee.github.io/javamail/ [Accessed 10 October 2020].

Gray, T. (2005) *Comparing Two Approaches to Remote Mailbox Access: IMAP vs. POP*. Available at: http://staff.washington.edu/gray/papers/imap.vs.pop.brief.html/ [Accessed 12 November 2020].

Kundrat, J. (2009) *IMAP E-mail Client*. BSc. Charles University in Prague.

The Radicati Group, Inc (2017) *Email Statistics Report, 2017-2021*. [docx] Palo Alto: The Radicati Group, Inc. Available at: https://www.radicati.com/wp/wp-content/uploads/2017/01/Email-Statistics-Report-2017-2021-Executive-Summary.pdf [Accessed 10 October 2020].

Statista (2017) *Number of sent and received e-mails per day worldwide from 2017 to 2025.* Available at: https://www.rlf.org.uk/resources/writing-essays/ [Accessed 28 September 2020].

Elkins, M. (2021) *The Mutt E-Mail Client*. Available at: http://www.mutt.org/doc/manual/#intro [Accessed 12 January 2021].

Mullet, D. & Mullet, K. (2000) *Managing IMAP 1st ed.* [Online]. O'Reilly Media, Inc. Available at: https://learning.oreilly.com/library/view/managing-imap/059600012X/?ar

Reynolds J. K. (1984) *POST OFFICE PROTOCOL* . [html] IETF. Available at: https://tools.ietf.org/html/rfc918 [Accessed 12 November 2020].

Heinlein, P. & Hartleben, P. (2008) *Book of IMAP : Building a Mail Server with Courier and Cyrus.* [Online]. No Starch Press, Inc. Available at: https://ebookcentral.proquest.com/lib/lancaster/reader.action?docID=1137520 [Accessed 12 February 2021].

Lammle, T. (2009) *CompTIA Network Deluxe Study Guide (Exam N10-004) 1st ed*. [Online]. Wiley. Available at: https://pdfroom.com/books/comptia-network-deluxe-study-guide/Zavd9L0A5KD [Accessed 12 February 2021].

Crocker D. H. (1984) *STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES*. [html] IETF. Available at: https://tools.ietf.org/html/rfc822 [Accessed 19 November 2020].

Freed N. & Borenstein N. (1996) *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* [html] IETF. Available at: https://tools.ietf.org/html/rfc2045 [Accessed 24 February 2020].

Freed N. & Borenstein N. (1996) *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies* [html] IETF. Available at: https://tools.ietf.org/html/rfc2046 [Accessed 24 February 2020].

Moore K. (1996) *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text* [html] IETF. Available at: https://tools.ietf.org/html/rfc2047 [Accessed 24 February 2020].

Freed N. & Klensin J. & Postel J. (1996) *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures* [html] IETF. Available at: https://tools.ietf.org/html/rfc2048 [Accessed 24 February 2020].

Postel J. B. (1982) *SIMPLE MAIL TRANSFER PROTOCOL* [html] IETF. Available at: https://tools.ietf.org/html/rfc821 [Accessed 21 September 2020].

Postfix (?) *Postfix Documentation*. Available at: http://www.postfix.org/documentation.html [Accessed 10 October 2020].

Dovecot Authors *Dovecot manual*. Available at: https://doc.dovecot.org/

[Accessed 10 October 2020].

Appendix 1

MyMail - Email Application Developing

Aleksandr Goidin

Abstract

The proposed project reported in this paper will aim to create a simple version of MUA (Mail User Agent) using Java Programming Language. The main focus will be on learning Email protocols and implementing them in the program without the usage of external libraries.

Work on the project will begin from reading and understanding the role of standards and key protocols used in mail systems, mainly SMTP and IMAP, respectively, for receiving and sending emails. By the time flow after achieving base goals, user interface with working processes will be implemented and focus shifted towards designing new features: encryption - a secure way of communication and send attachments.

1. Introduction

The history of electronic mails comes from the 1960s, while they only could be sent to the users of the same machine, or a bit later version required both sender and recipient being online [1].

Nowadays, it is hard to imagine our life without using email services. Electronic mails are used for work, study, marketing even keeping friend conversation. Research Group, The Radicati Groups, Inc. released a report which shows that there are a bit more than 400 million active (last accessed within three months) email users and roughly 306.4 billion emails are sent and received daily and this number is predicted to increase by 60 billion by the year 2024 [2]. Despite the popularity of emails, the email itself is similar to an email message sent in the early 1970s.

To simplify the process of creating an email client, there were created external libraries, like JavaMail API for Java programming language [3]. Users can effortlessly implement mail protocols and connect to the mail server within a few minutes. The proposed project will aim to deep learn the standards and key protocols used in email services. This will be achieved by reading publications made by IETF (Internet Engineering Task Force) – organization which develops Internet standards that comprise the TCP (Transmission Control Protocol)/IP (Internet Protocol) protocol suit [4]. The knowledge gained will be used in developing MUA with the functions of sending and receiving emails. A part of the project will focus on creating an appealing GUI (Graphical User Interface) to make software user-friendly and similar looking to the latest client email services. Furthermore, modern email services have options of sending not only text message, but different attachments. As a part of the project, these features will be implemented after the successful completion of the first two parts.

This project proposal will be split into the following sections; the proposed project; background; the programme of work; the resources required; and references. The proposed project will contain overall aims and objectives to be achieved by the end of the project with the help of methodology will be used to retrieve the required information. The background will contain a brief description of related papers. The programme of work will breakdown the project plan and tasks schedule throughout the year with the visual representation using the chart. The section with required resources will describe the resources required to successfully achieve the goals and the last section will contain references to any resources used within this paper.

2. The Proposed Project

2.1 Aims and Objectives

The project aims to create a desktop email service software with a formal design, that will be supporting processes of securely sending, receiving and fetching emails. The development of the application should be done from ground up with limited use of existing libraries which carry out most of the basic mail processing. To succeed in achieving goals the following objectives will be involved:

- Confidence reading of documents of Internet standards as SMTP (Simple Mail Transfer Protocol) for electronic mail transmission, IMAP (Internet Message Access Protocol) for retrieving email messages from a mail server, DNS (Domain Name System) for assigning domain names.
- Researching the most popular desktop email applications to compare the advantages of the user interface to develop a similar GUI.
- Using encryption to get a higher level of message transportation.
- Once main goals achieved, confidence reading of documents to achieve a feature of sending attachments in addition to the basic text.

2.2 Methodology

The software developing approach used will be similar to the waterfall method. How the project will be split up and carried out is described in the Programme of Work (section 4).

For the first learning phase, the study of standards and protocols will be merged with a little coding to consolidate the knowledge of the practice. SMTP protocol will be a starting point of learning. After understanding the specifications of the chosen protocol, IMAP protocol will take the leading place on further studies.

The next stage is the main because it will carry the most important practical part of the proposed project - the development of the main functionalities and testing. To ensure that the proposed project will be successful the following criteria for evaluating will be set;

- The application on the client-side is able to send an email to the server-side.
- The application on the client-side is able to receive an email from the server-side.
- The application can fetch a list of emails from the server.
- The transportation of emails is secured.
- Application has a formal and modern GUI.
- Application is user-friendly and easy to navigate.
- Code is clean, understandable and might be used by others for future developments

3. Background

The papers related to the proposed project are RFCs (Request for Comments), document publications from IETF. RFCs cover lots of different aspects of the Internet and Internet-Connected Systems [5]. The basic email protocols are also part of RFC publications. The first document about SMTP – RFC788 was published at the end of 1981 [6]. It covers the model, procedure and specifications. Later RFCs include innovations, researches, changes that were made by volunteer publishers. The proposed project will include implementing on practice the knowledge gained from reading RFCs regarding the electronical mail standards and protocols.

4. Programme of Work

The project will begin early October 2020, running until March 2021, and it will be broken up into the following stages;

- Project Proposition – This will involve early-stage research on information that might be related to the proposed topic. And later, will be discussed the ideas, aims, methods of the project with the supervisor and documenting them in this paper. This will take around 3 weeks.
- Basic Standards Learning – The focus will be on confidence reading of basic email standards SMTP and IMAP, and testing knowledge by creating a draft application. This will take around 3 weeks.
- Basic Functions Developing – Implementation of gained knowledge and developing main application functionality – sending and receiving emails. Some simple GUI might be created. This will take around 4-5 weeks.
- Testing Phase – Testing the program on local machine and webserver on later phase. This will take around 2-3 weeks.

- Final Paper Writing – Creating notes based on results throughout the project and writing the final thesis. This will take around 6 weeks.
- Focus on Security – Developing security part of the application will be done to support the encrypted way of transportation. This will take around 2 weeks.
- Formal GUI Creation – This will involve researching on the UI of the top email services and creation of the mock-up depending on the information gained, and its implementation. This will take around 2 weeks.
- Extra Features Implementation – This will involve developing new features and introducing them into the software by successful completion of the main part. This will take around 2 weeks.

The overall schedule for the project in 2020-2021 is displayed by Gantt chart in Figure 1 on the following page.

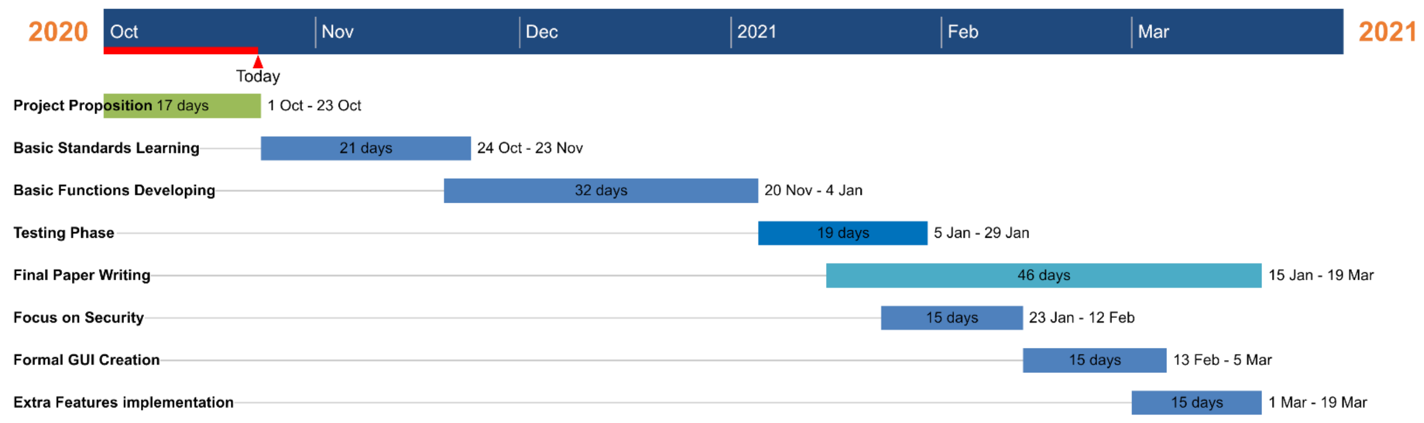


Figure 10: Project Schedule 2020-2021

5. Resources Required

Usage of different email addresses will be required to validate if software works correctly. Moreover, having a few computers with different operating systems will help to test the application, so that it properly works on different types of machines.

For testing phase VM (virtual machine) will be required for connecting to the server.

6. References

1. A brief history of email (Bookyourdata): https://www.bookyourdata.com/email-list-database/a-brief-history-of-email Accessed 1th October 2020.

2. Email Statistics Report, 2019-2023 (The Radicati Group, Inc., 2019): https://www.radicati.com/wp/wp-content/uploads/2018/12/Email-Statistics-Report-2019-2023-Executive-Summary.pdf

Accessed 1th October 2020.

3. JavaMail Reference Implementation (Oracle):
https://javaee.github.io/javamail/
Accessed 3th October 2020.

4. About IETF (IETF):
https://ietf.org/about
Accessed 7th October 2020.

5. Internet Standards: RFCs (IETF):
https://ietf.org/standards/rfcs/
Accessed 10th October 2020.

6. RFC788: Simple Mail Transfer Protocol (Jonathan B. Postel, 1981)
https://www.rfc-editor.org/rfc/pdfrfc/rfc788.txt.pdf
Accessed 8th October 2020.