Final Project Report

Email Client

Skrumpen Interactive

Group Members

Autriche Kabwe Sango	*au590692*
Martin Braas Andreasen	*au668926*
Steffen Torpe Simonsen	*au657869*
William Zoffmann Buchhave	*au676220*

Introduction

Over the course of this report, we will be diving into the features of the product and how they are implemented in more technical detail. Sending and receiving, replying and forwarding, deleting and marking of emails are all features which will be described. Afterwards, there will be a description of the process of developing the product, and the development process the team utilized. Lastly, this paper will highlight any deviations or changes in procedure from the original plan, which the team have deemed necessary.

The goal of this paper is to give the reader an understanding of what the Skrumpen Email Clients promises to do and which needs it will fulfill for the end-user, as well as providing knowledge as to how the technical aspect of the client works. Furthermore, it aims to give insight into the team's development process, and what techniques were utilized during the development.

Product Description

The Skrumpen Email Client is a stand-alone email client, intended for private use. It has similar utility to known email clients such as Gmail and Outlook. Skrumpen Email Client is running locally on the user's machine and is capable of standard email functionality, such as sending emails to any email-address and receiving and viewing any email sent to the email-address used upon login.

The supported features of Skrumpen Email Client are:

- Receiving/viewing emails
- Sending, replying to and forwarding emails
- Deleting emails
- Marking emails as read/unread accordingly

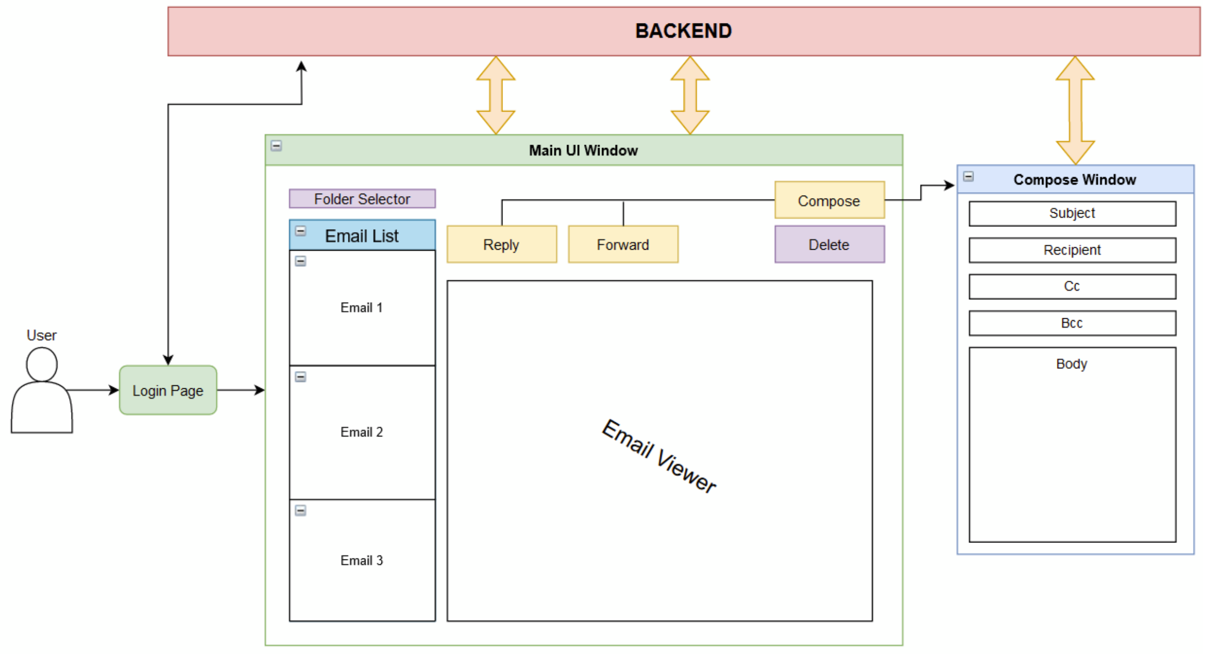
The individual features as well as more detailed descriptions of how they work will be further explored later in the report.

When the user receives the program, the user will need to firstly install the client on the system, then after launching the program the user will be prompted to login using their credentials. The program will then check if the credentials are correct and react accordingly. Then the user will be shown their inbox, from there it is possible to send and receive email. Upon clicking any given email, the email-viewer will display the email so that the user can read and interact with it. Here the user will be able to perform a number of actions, such as forwarding the email to a different email address or reply to the email directly.

To reiterate, the overall workflow of using the program consists of three main stages:

1. Logging in
2. Browsing the inbox - i.e. browsing or reading emails
3. Writing, replying or forwarding emails

Underneath is a chart depicting these three states and how the user interacts with them as well as how they interact with each other and with the backend of the program.



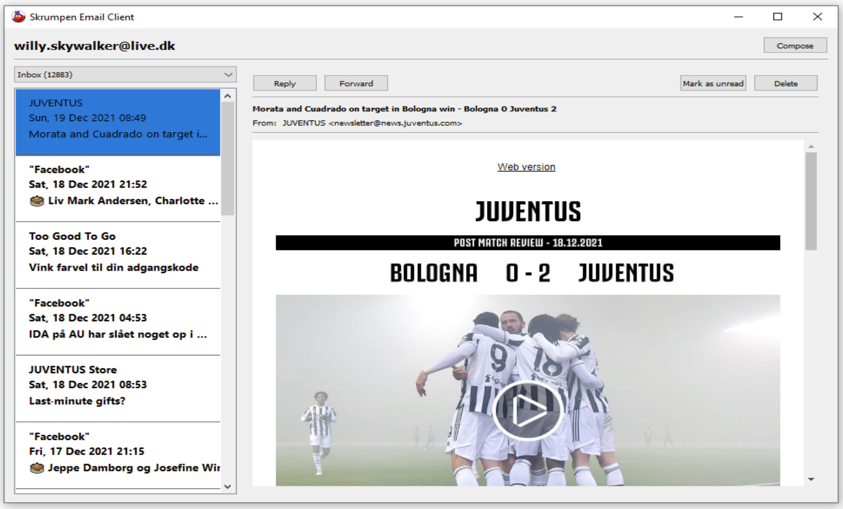
The conceptualized flow and layout of the program

Feature Showcase

1. Receive Email

Skrumpen Email Client is able to receive emails using the IMAP4 protocol. IMAP (Internet Message Access Protocol) differentiates itself from the older POP (Post Office Protocol) by not storing the email in question locally, but instead viewing it directly from the server. This means that the emails can be accessed from any device, instead of being pushed to one device for local storage, which was what the POP protocol did.

Concretely it means that the receive mail functionality of the program prompts an IMAP server, belonging to the email service provider of the user-inputted email address, for the mails in some inbox, usually the general inbox, and then prompts the IMAP server again when accessing more emails or the emails of a specific folder from the drop-down menu in the top left corner.



willy.skywalker@live.dk

JUVENTUS
Sun, 19 Dec 2021 08:49
Morata and Cuadrado on target i...

"Facebook"
Sat, 18 Dec 2021 21:52
Liv Mark Andersen, Charlotte ...

Too Good To Go
Sat, 18 Dec 2021 16:22
Vink farvel til din adgangskode

"Facebook"
Sat, 18 Dec 2021 04:53
IDA på AU har slået noget op i ...

JUVENTUS Store
Sat, 18 Dec 2021 08:53
Last-minute gifts?

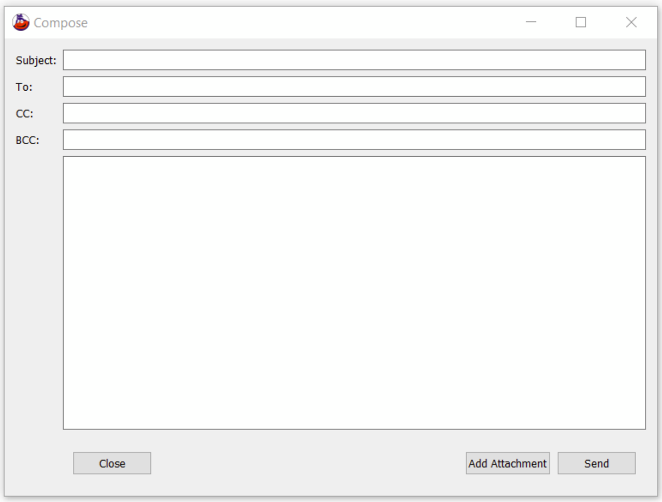
"Facebook"
Fri, 17 Dec 2021 21:15
Jeppe Damborg og Josefine Wir

Morata and Cuadrado on target in Bologna win - Bologna 0 Juventus 2
From: JUVENTUS <newsletter@news.juventus.com>

The Inbox Window and main UI window of the program

Upon receiving the data from the server, the program interprets the attributes of the email and determines whether the email is written in plain text or HTML. Based on this assessment, the email viewer will display the email accordingly. All data is then loaded into the appropriate boxes in the UI.

2. **Send / Forward / Reply Email / Add attachment**
   The client is able to send emails. For finding the functionality within the UI, click ‘Compose’ in the top right corner of the inbox page. The functionality for sending the emails is achieved by the use of “Simple Mail Transfer Protocol”, which is better known as SMTP. By sending a string of data to the SMTP server belonging to the account’s login credentials (For instance Outlook or Gmail), we can contact different email providers SMTP server, and thereby send an email. The strings we send to the SMTP servers are composed of the different input-boxes on the Compose Window, such as Subject, Recipient, CC, Text body and so on. This happens when the ‘Send’ button in the Compose page is pressed.



The Compose Window

‘Forward’ and ‘Reply’ works in the same way as above, the only difference is that some data from the email which are currently selected in the inbox, when the button is pressed, will be forwarded to the Compose Window. When pressing ‘Reply’ the subject and sender attributes of the email currently displayed by the email viewer, will be forwarded to the Compose Window as “RE: (Subject)” and the Sender will be added to as the new recipient. For forward, the entire mail which is selected, will be added to the compose window.

Another useful feature is the ‘Add attachment’ button. This allows the user to add attachments found locally on their computer or network drives and send them with the email.

3. **Mark as Read / Unread**
   In the inbox, emails in the currently selected folder are presented in a list on the left side of the screen. Emails that have not previously been opened are marked as ‘unread’ by setting the font of the sender, subject and first few lines of the body as bold. This works by extracting information from the server associated with the user’s login credentials, about whether or not the email has been read yet. Depending on the status of the email when extracted from the server, the client then displays the mail in either bold or normal font.

   Additionally, the user may choose to press the ‘Mark as unread’ button in the Inbox Window, to mark the email as unread to look at at a later point in time.

4. **Delete Emails**
   In the Inbox Window, the ‘Delete’ button allows users to delete emails permanently from the associated SMTP server. Upon clicking the ‘Delete’ button, the program registers the currently displayed email, and prompts the server to delete it permanently.

5. **Folder Navigation**
   Through the client, users have the ability to choose to load different folders that have been created by or in the user's email service provider’s client. The program loads the header attributes of the folders from the SMTP server upon connecting and displays them in a dropdown menu in the top left.

Process & Development techniques

Regarding the process of developing the email client, we started off by aiming to follow an agile/incremental development process. We chose this instead of a plan-driven development process, as it gave us a bit more flexibility in choosing when to work on what and because it suited our ‘everybody-does-everything’ -policy better than having to plan out everyone's involvement in every development stage.

Choosing an agile approach had both its pro’s and con’s. For one, it revealed itself to be a huge advantage when we decided to re-model the UI, which will be touched upon in greater detail later in this report. However, it also has some logistic implications on the development process, as there weren't necessarily deadlines for when specific processes were supposed to be done, which had its effect on the flow of the development.

Looking back at the development at this point in time, we feel that the pros of choosing an agile approach outweigh the cons, as it put the emphasis on the development team rather than the process - meaning that instead of following some prescriptive process or predestined approach, we were able to freely figure out solutions to problems that occurred on the fly, which gave a tremendous amount of flexibility in the work-process of each team member respectively. This would not have been possible to the same degree, had we chosen to go with a plan-driven development.

Furthermore, since this was a task that none of the team members had previously encountered, we feared that going with a plan-driven process without being confident that we had the necessary skills for developing this product would result in huge pitfalls in the process-planning, since we essentially weren’t equipped to develop a plan that would cover the different aspects of the plan sufficiently.

The most prominent development technique in our work with the client, has been the ‘pair - programming’ technique. This was done by sharing the code repository using VSC’s (Visual Studio Code) LiveShare function, enabling us to work together on the same piece of code on the same computer remotely. We found that this particular strategy suited our group very well, as it was rare for all of us to be working on the code at the same time, people could quite easily reach out to another group member and start working in pairs of two or three on some module in the system.

This also had the added benefit of enabling us to rather quickly identify issues with the code, such as some lines or modules not working correctly, as we were often two or three developers to look through and verify the code that had been written. This process of developing the code together became even more important when we chose to reform the UI, as some team members had a lot more previous knowledge of the PySide library than others.

Software Reuse

For communicating with the SMTP and IMAP protocols, we use python libraries. These libraries have a set of standard functionalities, which allows us to connect, prompt, receive, interpret and send data to different providers SMTP and IMAP Servers. For formatting the data, we receive using the SMTP and IMAP python libraries we utilize the python library called EMAIL, for such things as decoding and so on.

- **SMTP-lib**
  From the SMTPlib python library, we draw the use of different functionality. First off we use the functionality smtplib.SMTP to define the SMTP server and port, which we are trying to connect to, when the send_email method is prompted. We then call the "connect" method of the SMTPlib to connect to the actual server, both when login in, and when sending emails. When writing the emails, we use a mix of functionality from smtplib, and the library called "email" to format the content, in the end formatting it into a string, and calling send, to send the data/string to the SMTP server in question.

- **IMAP-lib**
  From the IMAPlib python library, we once again use some of the same functionality, as the functionality which we got from SMTPlib. We also use the imaplib.IMAP4 functionality, to store and connect to the IMAP server. This means that when the function, in which the imaplib.IMAP4 is called, we establish a connection, which fetches the data from the IMAP server. When we fetch the data, we then once again utilize some of the tools from the email python library, to correctly format the received data into the UI, making it readable. We use a feedback signal loop, which frequently updates the inbox pages UI, to correctly display the mails.

Deviations

Going into this project, we had as previously mentioned we decided to implement an agile development process. Alongside that, we had a few principles in regard to specific aspects of the product itself and the process of development, which we wanted to follow:

- **Incremental Development and Delivery**
  Initially, it was our plan to exploit the benefits that an agile approach offered us, and incrementally develop and deliver the product to the customer - thereby ensuring customer involvement and concurrently specify, develop and validate the product as it progressed. We eventually found it difficult to deliver this process to the customer, as the work on the product would happen irregularly since team members often were preoccupied doing work unrelated to the development of the client. This led to the prioritization of this project falling behind in the queue of deadlines to be met, and ultimately we felt that doing this as a regular occurrence was quite difficult to do.

  We deviated from this plan by instead shifting our focus on meeting the requirements we specified in collaboration with the customer prior to the implementation and programming.

- **Features**

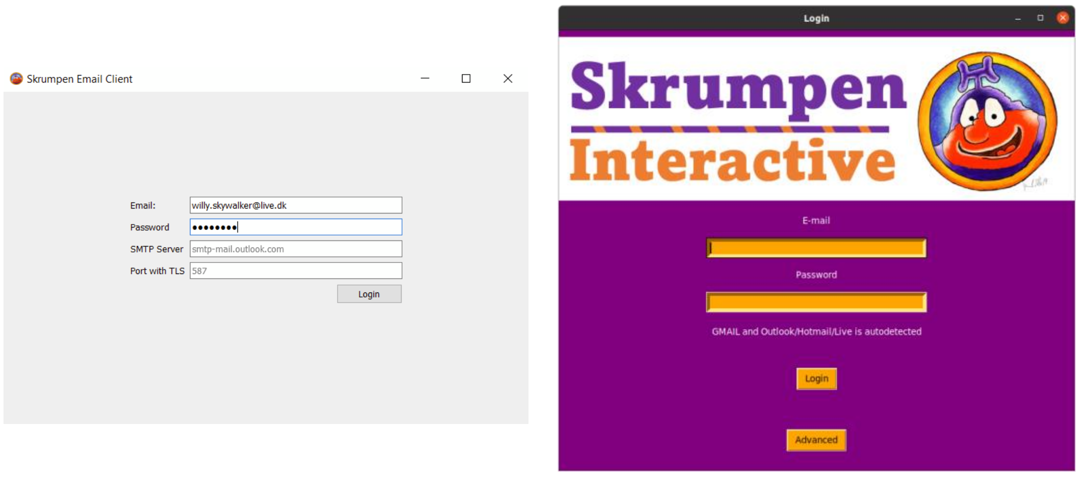
  As discussed above, we had a heightened focus on implementing the basic requirements, them being
  - Sending and receiving emails
  - Forwarding and replying to emails
  - Deleting emails
  - Marking emails as read/unread

  In our requirements specification, we proposed adding additional features
  - Marking emails with flags
  - Organizing emails in user-made folders
  - Create drafts

  These took secondary priority, as we were pressed to implement the minimum requirements first, ultimately resulting in them falling to the wayside. However, the final product does have some additional features which exceeds the scope of the initial minimum requirements:
  - Adding attachments to emails
  - Navigation between folders defined by the email service provider

- **UI - User Interface**

  During the development of the UI, we discovered some issues with the first GUI-tool we used; Tkinter. Tkinter had the advantage of being a part of the Python standard library, and thus the prerequisites for the program wouldn't be as extensive when using Tkinter. However, although Tkinter is quite easy to learn and implement, when aiming for a more modern UI-design, there were some shortcomings. Especially when designing the layout of the inbox-page, we discovered that implementing a design we liked in Tkinter would take a lot of time and involve a few tedious workarounds.



Email: willy.skywalker@live.dk
Password ••••••••
SMTP Server smtp-mail.outlook.com
Port with TLS 587

GMAIL and Outlook/Hotmail/Live is autodetected

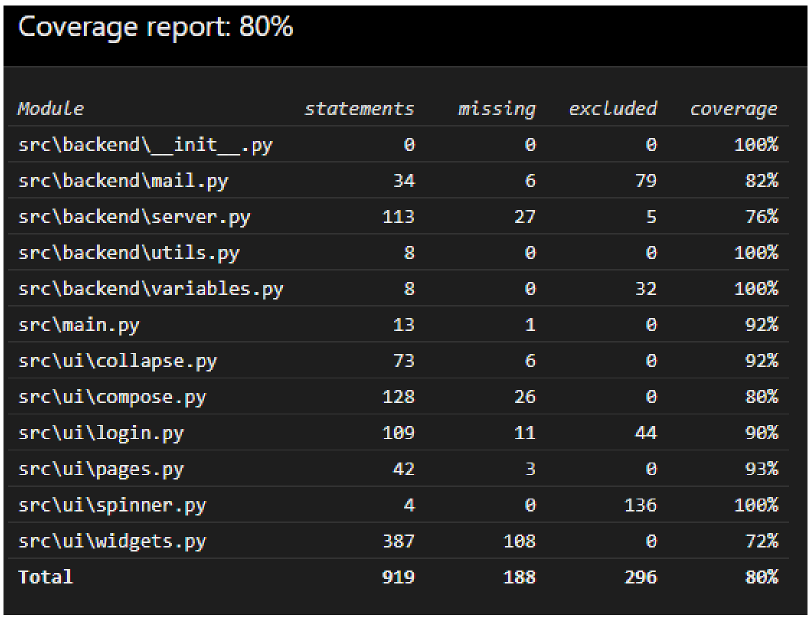
Comparison of the new login-window (left) and the old (right)

These issues, coupled with the fact that the overall look and feel of Tkinter was quite outdated, prompted us to switch to PySide2 for the UI solution.

Even though PySide2 was more capable of designing a modern UI for the application, it was a lot harder than Tkinter to learn and implement. But, since we were committed to delivering a product that looked good and felt nice to operate, we stuck with it.

- **Danish Characters**
  In terms of the different letters in the alphabet, we have met a problem concerning nordic letters such as Æ and Ø. Therefore Skrumpen Email Client is not able to display these symbols unfortunately. The team has concluded that it is due to the libraries which we have implemented and that we utilize to retrieve the emails from the server. In order to fix this problem we would have to implement a rework to the method the program uses to fetch emails from the server side. Unfortunately we realized this was a critical problem too late in the development process and fixing it is not really viable with the time left until the delivery deadline.

- **Tests and Test-Coverage**
  Initially we planned to have a 100% test-coverage of the code - however this proved to be quite a lot of extra work, and in some cases not strictly necessary. Test-coverage on the Skrumpen Email Client has reached 80% at the time of delivery, which means that 80% of all the code written is covered by the tests. For an in-depth look into how we have performed testing on this program we refer to our 'Test Plan'. The overall test results will be included in the program files, where it is possible to see what code is affected by the tests, and which are not. To view specific code, just open the .html files in a browser and it is possible to review the test-coverage that way as well.


Coverage report: 80%

Module	statements	missing	excluded	coverage
src\backend__init__.py	0	0	0	100%
src\backend\mail.py	34	6	79	82%
src\backend\server.py	113	27	5	76%
src\backend\utils.py	8	0	0	100%
src\backend\variables.py	8	0	32	100%
src\main.py	13	1	0	92%
src\ui\collapse.py	73	6	0	92%
src\ui\compose.py	128	26	0	80%
src\ui\login.py	109	11	44	90%
src\ui\pages.py	42	3	0	93%
src\ui\spinner.py	4	0	136	100%
src\ui\widgets.py	387	108	0	72%
Total	**919**	**188**	**296**	**80%**

Table depicting test-coverage of all the program files

As is evident, test-coverage has reached 80% which gives the team at Skrumpen Interactive the confidence to say that critical functionality of the program is tested and working to a satisfactory standard.

Installation Description

The guide for the installation of Skrumpen Email Client can be found in the `README.md` file in the program files, or on the Github-page of the project here.