

# Pigeon - E-mail Client Application

Final Report

by

**Andreas Kaag Thomsen**  
**Asger Song Høøck Poulsen**  
**Buster Salomon Rasmussen**  
**Niels Viggo Stark Madsen**

A Software Engineering  
Report



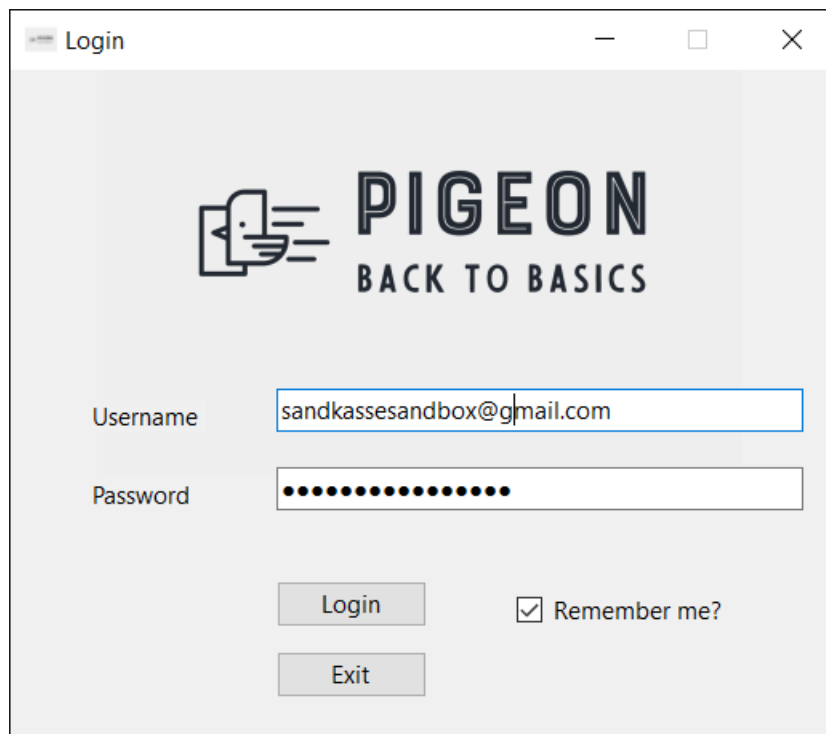
December 20, 2022

# 1 Product description

A list of mandatory requirements were discussed with our customer which are included in **Report 1**. All of these mandatory requirements and some optional requirements have been implemented. The ones that have not been implemented are discussed in section 5.

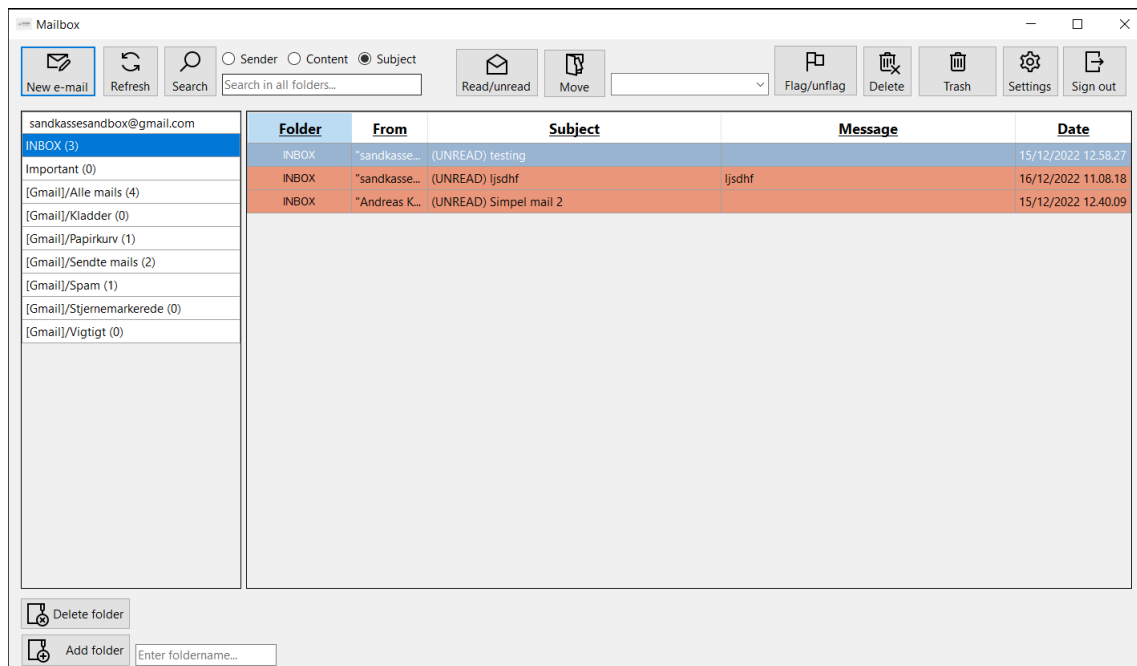
In this way, the Pigeon E-mail client is an application, which focuses on core e-mail functionalities. That is, an user-friendly application that enables the user to see all of their e-mails and interact with them in a familiar way. We have focused on core functionalities such as sending and receiving e-mails as well as draft-functionality, searching, and marking/moving e-mails. Moreover, we have focused on user-friendliness by implementing a settings panel, which enables the user to customize the application to fit their needs. Here, we have given the choice to toggle offline-mode, local storage functionality as well as a choice to clear all data, that is stored locally. The need for this kind of functionality comes from the desire to achieve true responsiveness. For implementing this, we chose to store all of the user's e-mails locally in order to be able to fetch them quickly at login. This also called for the need of concurrency since we wanted to 'listen' for changes made on the server (e.g. receiving e-mails). This increased the complexity of the code significantly and since we are aware that the implementation is not flawless we chose to give the user a choice to disable all this functionality though this will decrease the performance of the application.

In order to make the application more user-friendly, we have built the program of different *forms*. This gives the user the possibility to have several forms open - i.e. read more e-mails at once and/or compose new e-mails meanwhile. We have often referred to these forms in prior reports with the names 'mailbox', 'new email form' and similar. Below are screenshots of the final layout of the different forms.

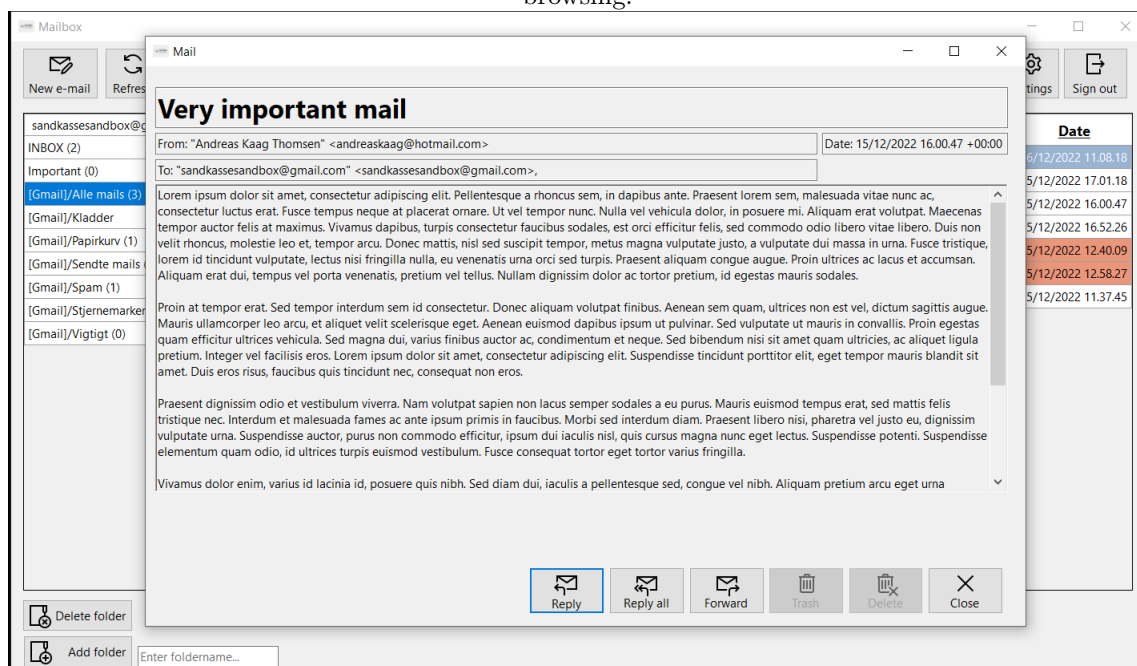


The screenshot shows a login window titled "Login". At the top center is the Pigeon logo, which consists of a stylized pigeon head icon and the text "PIGEON BACK TO BASICS". Below the logo are two input fields: "Username" and "Password". The "Username" field contains the text "sandkassesandbox@gmail.com". The "Password" field is filled with dots, indicating it is masked. Below the password field, there are two buttons: "Login" and "Exit". To the right of the "Login" button is a checked checkbox labeled "Remember me?".

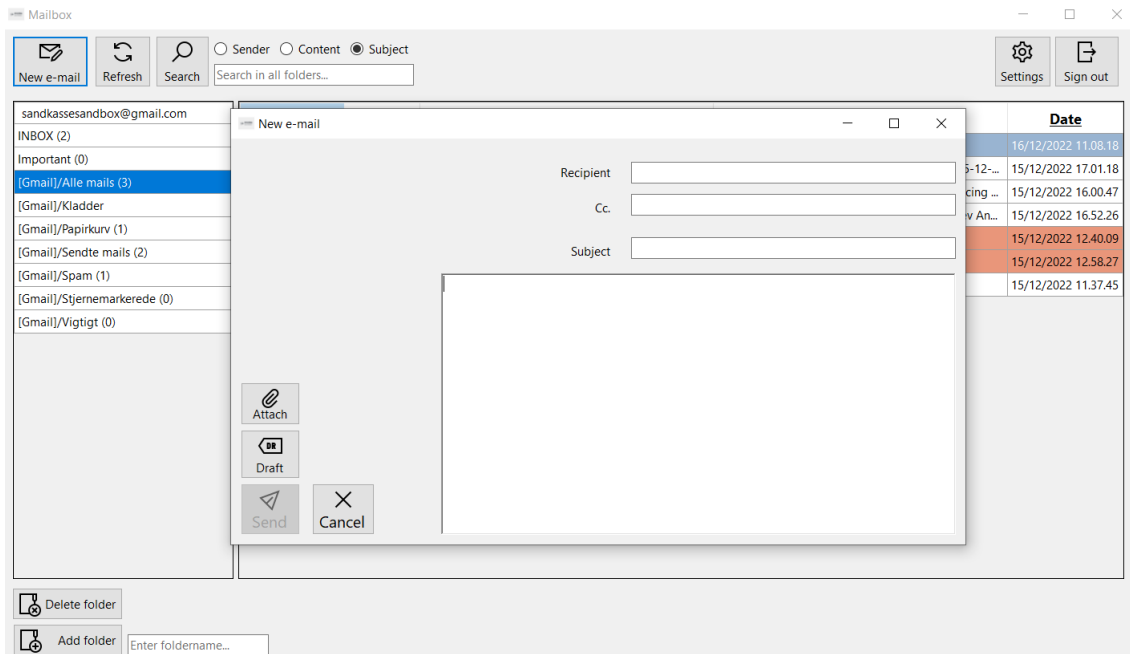
This is the login form.



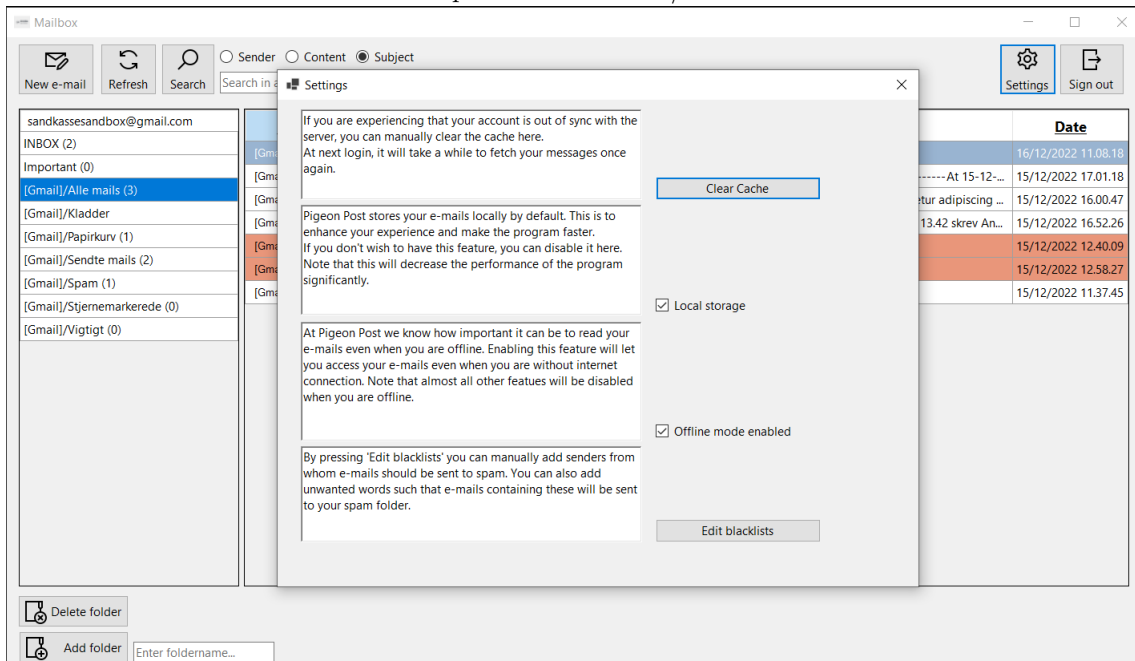
This is the main mailbox window of our mail client where you handle most email operations and browsing.



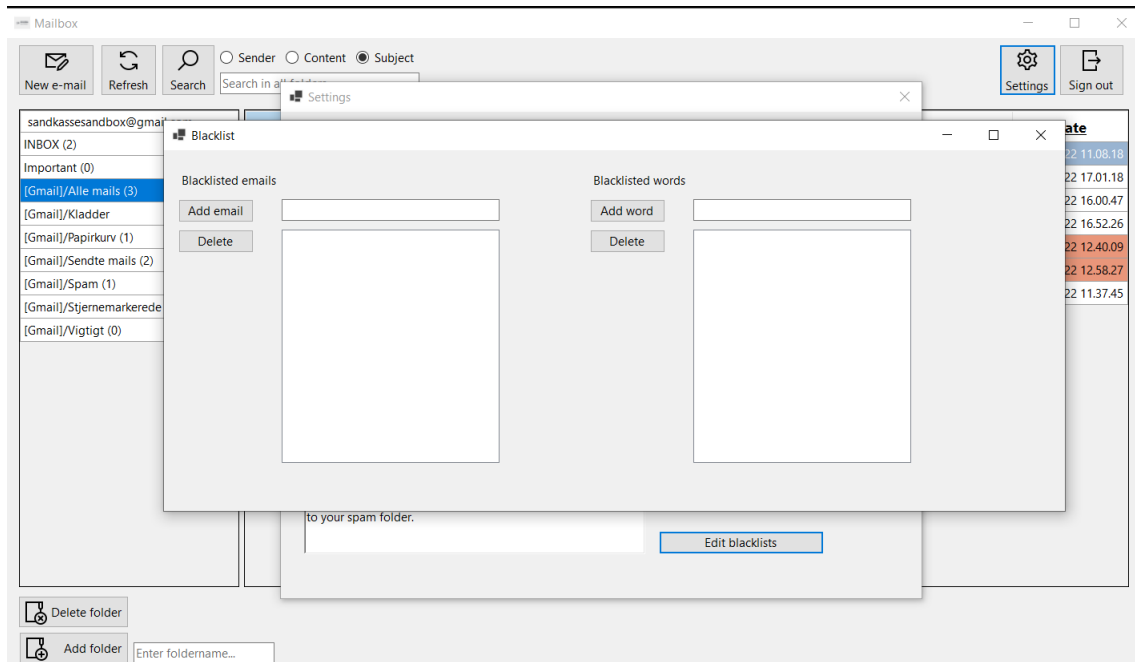
This is how mails are shown when clicked in the mailbox. If the message contains any attachments, these are shown in the bottom left corner where a download button is also available.



This is the compose email window/new e-mail form.



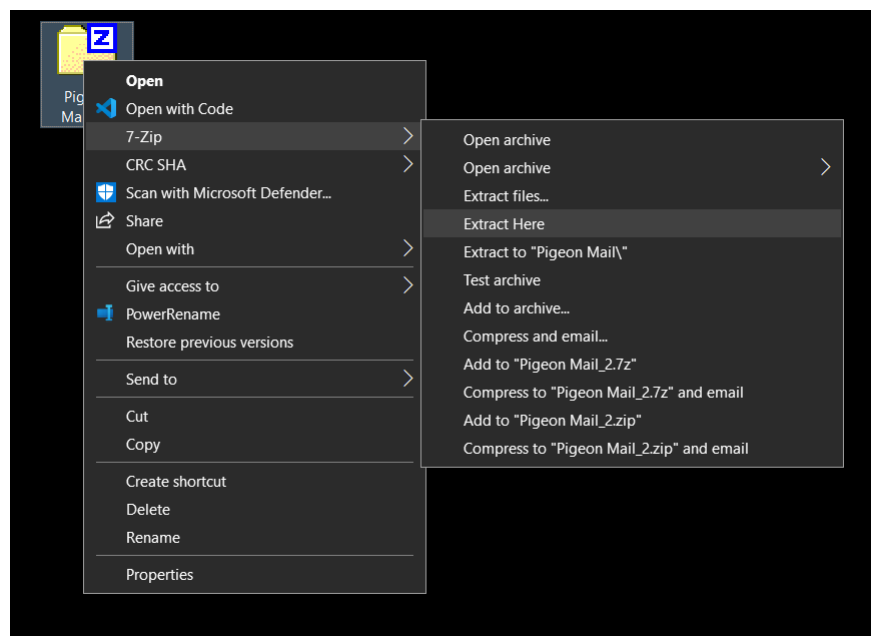
This is the settings window.



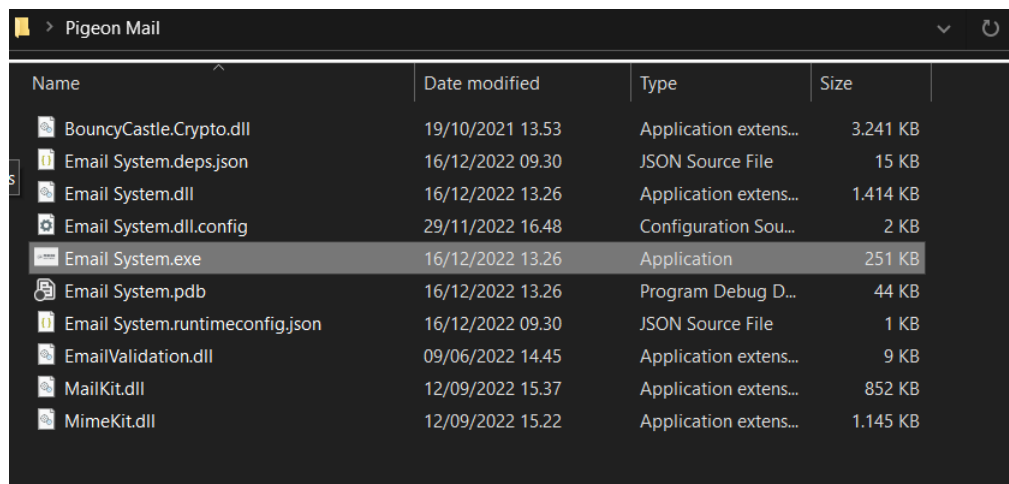
This is the blacklist editing window.

## 2 Installation guide

1. Unzip the provided file.



2. Run the .exe file on a computer running windows.



3. Use your own login for a Gmail account, or use our test email for the purpose, username: sandkassesandbox@gmail.com - password: mexxnwrzbskxwkrp.  
Note that when first starting up the program, the mailbox will appear to 'glitch'. This is simply due to the fact that the mailbox is currently refreshing while fetching your messages from the server.

The source code to the application can be found in our GitHub repository: <https://github.com/Andreas691667/Pigeon-Mail>.

Please refer to the README-file in the repository for finding the source code.

### 3 Individual contributions

As mentioned in **Report 1**, due to the size and the agility of the project, the roles were meant to be kept dynamic such that every team member would get the most exhaustive outcome of the course which was the case in the beginning. But due to different personality types and level of interests and developing skills the team members naturally converged into more fixed roles. The following section describes the main roles of each individual team member and their contribution to the project.

#### Andreas Kaag Thomsen

Andreas has acted as head of development. He has explored different alternatives for server interaction and settled for the best choice. Moreover, he has built the overall structure of the code and has developed the vast majority of the code for the E-mail Client. Unless stated otherwise below, Andreas has implemented all of the requirements from report 1 and also assisted the other team members. When necessary, he has also sought advice and discussed different solution alternatives with the other group members as well as our TA, Morten From Elvebakken. He has actively been using the platform Jira when reporting progress as well as reporting different bugs that some times occurred. He had also encouraged live-coding sessions where all team members discussed the code as it was being written. Since Andreas have had the greatest insight in the source code, he has also

written several parts of the different reports - especially paragraphs that required knowledge about the implementation.

### Asger Song Høøck Poulsen

Asger has acted as Scrum Master throughout the project by preparing for, calling in as well as facilitating the agenda at the stand-up meetings that have been held after every sprint has finished. Furthermore, Asger has been the primary contact between the team and the costumer/teacher whenever the team would have had any questions regarding the needs of the costumer or need any advice from the teacher. This also includes the weekly stand-up meeting that have been held between the teacher and the team.

Since Asger was the only one in the team who works on MacOS, he has mainly acted as an assisting force during the team's live-coding sessions. He has only developed UFM.(12) of the core functionalities by himself.

### Buster Salomon Rasmussen

Buster set up Jira, which was the tool for our agile process. Furthermore he set up confluence, on which the notes of the stand-up meetings have been taken. On the developing side, Buster helped the other developers with solving ad-hoc problems, and he implemented the blacklist functionality, that is UNO.(1). Furthermore he added structure, which included refactoring, commenting and diagramming.

### Niels Viggo Stark Madsen

Viggo has been the primary supporting developer of the e-mail client. Viggo has worked closely together with Andreas during developing phases of the requirements and bug fixing, especially with concurrency problems and other debugging tasks. He has worked a lot on developing a system for synchronizing the locally stored data and the email server data. This has caused some development troubles, because we don't have full control over the servers and cannot predict exactly when and how it does what it does. The data structure developed has a 2 step process for updating the locally stored data. First it updates the UI with the changes made. Then the server is told what changed. When the server is done changing the data it updates with the local storage. This is further described in report 2.

## 4 Software Reuse

The e-mail client is a *Windows forms* application supported by Microsoft's .NET 6.0 Framework<sup>1</sup> written in C#. Since Windows forms is only available on Windows the e-mail client application is not supported by MacOS.

### 4.1 The use of Mimekit

A large amount of our implementation has made use of the NuGet package, Mailkit, which is a subpackage of Mimekit<sup>2</sup>. All interaction with the server is handled with methods from this package. Moreover, we have used the NuGet package EmailValidation, which is made by the same who has

---

<sup>1</sup><https://learn.microsoft.com/en-us/dotnet/fundamentals/>

<sup>2</sup><http://www.mimekit.net/docs/html/Introduction.htm>

made **Mailkit**. We have used this package for validating e-mails both at login and when constructing new e-mails to be sent.

Some code is also copied directly from the **MimeKit**-documentation. When so, this is stated in the source code.

Everything regarding the UI is implemented by ourselves, but whenever we wish to do the same response on the server we use **Mimekit**'s functions to do so. An example of this is when we mark an e-mail as unread we use **Task.IMailFolder.RemoveFlagsAsync()** for the e-mail to be marked on the server.

## 5 Deviations

### 5.1 Time

Almost halfway through the project, all of the mandatory requirements as well as many of the optional requirements were implemented. However, after a prototype was shown to our customer, we were asked to optimize the application by making it concurrent. This caused a major setback in our initial timeplan, since the entire code had to be revised which resulted in that we did not have the time to implement all of our initial optional requirements (further details on this in [5.4](#)).

### 5.2 Process

We have progressed through the entire project, but at times we also had other priorities from other courses. Due to this there has been entire weeks, where we have stalled. But typically this has meant the we would work extra hard on the project the following week. Below is a cumulative flow diagram generated by Jira from December 12, 2022:

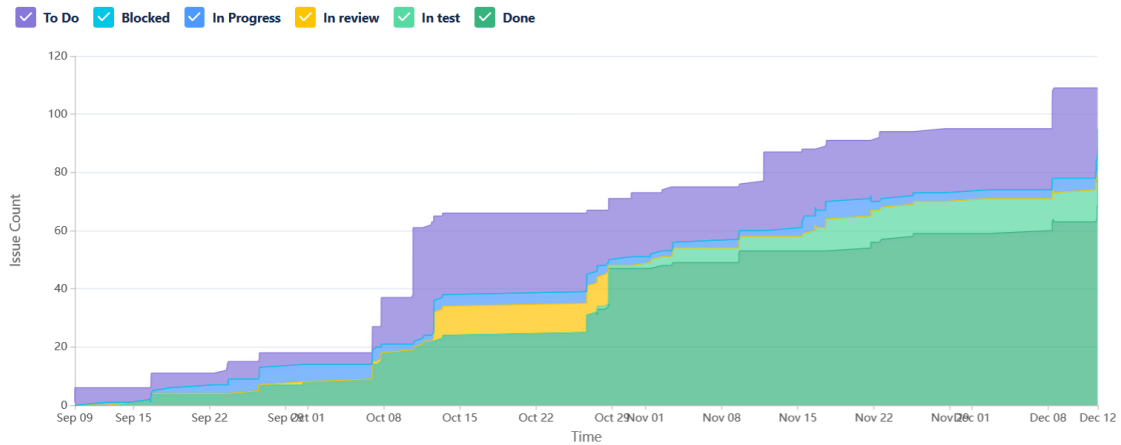


Figure 1: Cumulative flow diagram

Here, it can be seen that we have progressed almost every week with the exception of a few weeks. It is also clear to see from the diagram between the Oct 08 and Oct 15 mark that it was at around this point in time the team was asked to make the e-mail client concurrent. Thus, the drastic rise in "To Do" tasks.



Initially, it was intended that sprint durations were a week long and as a conclusion of the sprint a stand-up meeting would naturally follow, but as can be seen in the roadmap in **Report 1**, some sprints would often last twice as long due to either other courses' deadlines, and thus those would be prioritized for a moment, or the objectives determined at the previous stand-up meeting would be clear and comprehensive enough for the group to agree not to have another stand-up meeting for e.g. another two weeks. Even if a stand-up meeting was not held for a week, there has been a continuous, daily discussion of the project's development inside the group throughout the whole semester.

### 5.3 Tests

All of the initially planned tests for the mandatory requirements were executed. As well as the updated tests added since some requirements have changed. The added tests are

- 3 test cases for the added requirement UFM.(12).
- 1 test case for UFO.(9)
- 2 test cases for the added requirement UFO.(10)
- 1 test case for the added requirement UFO.(11)
- 1 test case for the added requirement UFO.(12)
- 4 test cases for the added requirement UNM.(5)

For further details on results on all the tests see Figure 3. Note that some of the optional requirements were not tested due to the fact that they have not been implemented.

### 5.4 Requirements

As mentioned, the full range of the mandatory requirements described in **Report 1** have been implemented. The following are the optional requirements that have not been implemented or deviated from.

#### UFO.(1)

Due to the fact that many of Mailkit's functionalities dependent on the e-mail provider, we chose to discard this requirement. In this way, some of the functionalities *only* work with g-mail. In the initial implementation we supported different mail-providers such as outlook and yahoo, but as we progressed with our implementation and the complexity increased we chose to abandon this and focus on other things in order to minimize the complexity of the code. If we had had more time, we would focus more on inheritance and in this way support different providers. This would, however, at the moment cause a tremendous restructuring of our entire code and it should have been a priority from the beginning.

## **UFO.(2) & UFO.(6)**

We learned that *all* server interaction relied on internet connection. So in order to detect incoming e-mails the program should be running. This could also be in the background where the user would not see it, but it would still cause the need for the user to have their computer running. This would not make much sense since this functionality is typically needed if the user goes on holiday or similar. Therefore, we chose to discard this optional requirement and focus on other, more relevant ones.

## **UFO.(8)**

After we made the program concurrent and chose to store the messages locally, we almost eliminated the need for this, since there are almost no 'long instructions'. Instead, we have a very responsive UI and provide a logging mechanism to give the user an indication when the changes are also finished on the server. When there is some semi-long operation (e.g. downloading attachments) we indicate this by a loading icon and disabling the current form. These operations do not take very much time and only occur seldom and therefore we do not find the need that the user should be able to cancel the operations.

## **UNM.(2)**

After we made the program concurrent, we always know exactly when there has been made changes on the server and we only need to refresh at this moment. Therefore, instead of a set time interval for refreshing, we only refresh when there is an actual need for it by invoking the refresh-methods from the UI-thread.

## **UNO.(2)**

We have not implemented this optional requirement at all. This was due to our limited time and by agreement with our customer this was not to be prioritized. We did look a bit into it, and it seems that the most straightforward way would be to convert the HTML/css into a png-file and then display this. However, we wish to enable the user to copy the text in the e-mail and forward/reply to it, and this would not function properly, if the whole e-mail was simply a png-file. But as mentioned, our customer explicitly said that this requirement was not to be prioritized.

## **SR.(4) & SR.(6)**

In the implementation of UFM.(3) and UFM.(4) we did not use the dictionary data structure as initially expected. Instead we used a list of lists, the first list with folders and the second with the message structs.

## **SR.(13)**

The implementation of UNM.(1) was changed because the program was made concurrent, by request of our client and this meant that the user is still able to interact with the program even when something is working in the background. Thus, we felt that the user should simply be notified in the right hand corner that the mail client is working on the request made, instead of the cursor changing. The concurrent thread running in the background is always comparing the clients local data to the server side data, thus ensuring synchronisation and quick operation.

SR.(14)

UNM.(2) This has been changed because the program was made concurrent. This means that we can update continuously in the background with no effect on the performance to the main window. Whenever an update is detected on the server it updates it on the UI no need to refresh in intervals.

## 6 A note on software distribution

As mentioned in report 1, we have, as a team, primarily worked within the agile methods. What we failed to mention here, is the use of GitHub. We have, throughout the entire project, used GitHub to share code between the group members. We have had a main branch, which Andreas has primarily worked on as well as branches, in which the other team members could implement features. We did, however, lack sufficient knowledge on how to use these branches properly as well as the use of `.gitignore`-files, which would have helped the process a lot. This caused some trouble when merging branches since our IDE, Visual Studio, generated several build-files that resulted in merge conflicts. Due to this, we sometimes used an excessive amount of time on simple pull/push actions. Yet, GitHub helped very much in distributing the code, and during the entire project, we have updated the repository with the latest code, as shown in the graph below, which depicts the number of commits to GitHub throughout the project:

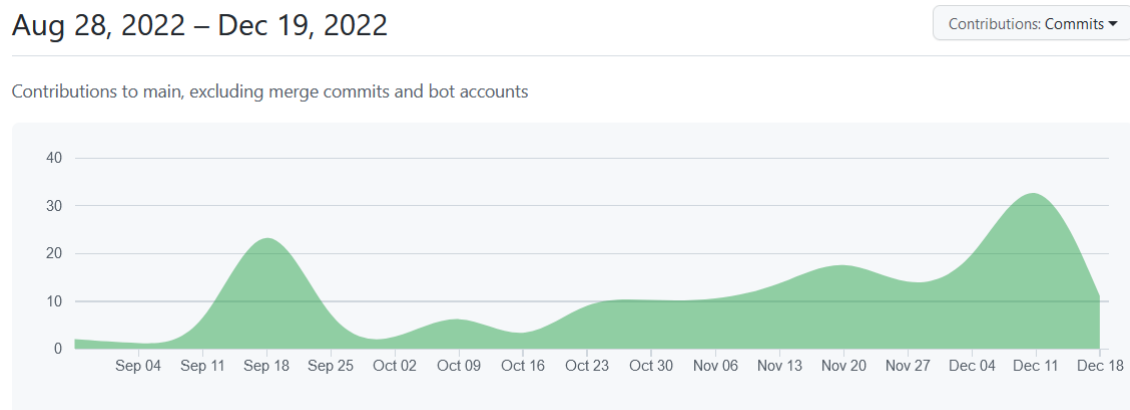


Figure 2: Commits to GitHub throughout the project.

There has been an almost steady rise in the number of commits as time elapsed. This is mostly due to the fact that we became much more familiar with writing in `C#` and thus could implement the requirements much faster towards the end. The rise in end of December is when we truly began performing the tests from report 3, which resulted in a lot of bug-fixing and cleanup in the code. As stated in the installation guide, the repository can be found at [this link](#).



## 7 Appendix

### 7.1 Appendix A - Test results

Requirement	No. of tests	Results	Comments
UFM.(1)	5	All tests passed	
UFM.(2)	2	All tests passed	
UFM.(3)	1	All tests passed	
UFM.(4)	1	All tests passed	
UFM.(5)	1	All tests passed	
UFM.(6)	1	All tests passed	
UFM.(7)	1	All tests passed	
UFM.(8)	1	All tests passed	
UFM.(9)	2	All tests passed	See remark in Report 3
UFM.(10)	1	All tests passed	
UFM.(11)	1	All tests passed	
UFM.(12)	3	All tests passed	
UFO.(1)	1	0 tests passed	Not implemented
UFO.(2)	1	0 tests passed	Not implemented
UFO.(3)	3	All tests passed	
UFO.(4)	3	All tests passed	
UFO.(5)	1	All tests passed	
UFO.(6)	1	0 tests passed	Not implemented
UFO.(7)	1	All tests passed	
UFO.(8)	1	0 tests passed	Not implemented
UFO.(9)	2	All tests passed	
UFO.(10)	2	All tests passed	
UFO.(11)	1	All tests passed	
UFO.(12)	1	All tests passed	
UFO.(13)	1	All tests passed	
UNM.(1)	1	Not tested	Out of scope
UNM.(2)	1	All tests passed	
UNM.(3)	1	All tests passed	
UNM.(4)	1	All tests passed	
UNM.(5)	4	All tests passed	
UNO.(1)	1	All tests passed	
UNO.(2)	1	0 tests passed	Not implemented

Figure 3: Test results of requirements