Pigeon - E-mail Client Application

Requirements and Work Plan

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

A Software Engineering
Report

PIGEON
BACK TO BASICS

October 7, 2022

1 Preface

This report is intended for clients of the Pigeon e-mail system, more specifically Associate Professor, Lukas Esterle. The report is also composed for the sake of the team behind in order to review requirements for the system later on. The report was revised last on December 20, 2022.

2 Introduction

The Pigeon e-mail system is a program in which users can connect to their own e-mail and read and send e-mails from their own account. Modern mail systems provide numerous functionalities, which, at the end, makes the whole system hard to manage. With our mail-system we wish to bring everything back to basics and focus on the core aspects that an e-mail system should provide. That is, sending and receiving e-mails.

3 Product vision

3.1 User stories

The following user stories are based on conversations with our client, Lukas Esterle, as well as lecture slides and will give a clear vision of how the core e-mail client functions from the user's perspective.

3.1.1 Logging into the client

When you open the client, a window with an e-mail- and password field will be visible above a "Login"- and an "Exit" button and a "Remember me" box.

If you select *Exit*, the window will close and you will exit the client.

If you input either your email or password incorrectly and try to log in, a window with an "OK" button will appear displaying an error message. By selecting "OK" you will exit the error-window and be able to try logging in again.

If you input your email and password correctly and select *Login* or simply press the **Enter key** on your keyboard you will be granted access to the Mail Client. A new window opens and your 'Inbox' and other folders such as 'Spam', 'Trash' etc. will appear. Above these there is a "Refresh"-button and below is a "Create new e-mail" button.

If you have selected *Remember me* before logging in, the Mail Client will remember your account when reopening the client, if you have not logged out again prior to this, such that you will not need to insert email and password again until you log out yourself.

3.1.2 Reading your e-mail

After logging in, an overview of all folders such as your 'Inbox', 'Spam', 'Trash' etc. are visible.

If you choose to enter one of these folders, you double click on the desired folder which will make the contents visible besides the folders overview. Here the "header"[1] of the e-mails will be visible sorted from most recently received from the top down to least recently received. E-mails that have not yet been read are highlighted with red.

[1]The header includes the sender, subject and the date of when the e-mail was received.

If you choose to double click on a certain ”header” the contents of the e-mail will appear in the same window such that you can see the contents and the list of folders to the left of this. Above this is a ”Forward”-button and a ”Reply”-button.

If you choose to exit the current e-mail or want to read another, you click on the ”back”-button below the e-mail’s content.

3.1.3 Create new e-mail

After logging in to the client there will always be a ”Create new e-mail”-button visible.

If you choose to click on this button, a new window will appear with three smaller text boxes representing who the e-mail is to, who to *Cc* the e-mail to and the subject of the e-mail. Below these is a larger text box in which the contents in the e-mail you wish to send will be written. Below the larger text box three buttons are visible: the ”back”-, ”save”- and the ”send”-button.

If you choose to click the ”back”-button a small window will appear saying if you wish to save the current e-mail in drafts in which you can choose either ”yes” or ”no”.
If you choose to click ”yes” the current e-mail will be saved under the ”drafts”-folder.
If you choose to click ”no” the current e-mail will be lost and you will return to the folders overview.

If you choose to click the ”send”-button the e-mail client will evaluate if at least the ”To”-text box and the ”subject” box is filled and the contents of the e-mail is not empty.
If the three aforementioned criteria are not satisfied: a small window with an error message will appear.
If the criteria are satisfied: the e-mail will successfully be sent to the recipient and you will return to the folders overview.

3.1.4 Forward an e-mail

When you have chosen an e-mail as described in the user-story ”Reading your e-mail”, you can choose to forward it. The forward button is located at top right corner, next to the reply button. When clicking the forward button a new window will open. This window will look very similar to the ”Create new e-mail” window, and behaves in exactly the same way. The only visual difference is that in the input-text box, a copy of the message you have chosen to forward will be inserted underneath
- - - - - - Forwarded message (received xx/yy/zzzz) - - - - - -
Above this you can choose to add some additional text.

3.1.5 Reply to an e-mail

When you have chosen an e-mail as described in ”Reading your e-mail”, you can choose to reply to it. The reply button is located at top right corner, next to the forward button. When clicking the reply button a new window will open. This window will look very similar to the ”Create new e-mail” window, and behaves in exactly the same way. The only difference is that the ”To”-box and the subject will already be filled with the sender’s e-mail address and ”Re: *”Original e-mail’s subject”*” respectively and in the input-text box, a copy of the message you have chosen to reply to will be inserted underneath
- - - - - - Replied message (received xx/yy/zzzz) - - - - - -
Above this you can add your reply.

3.1.6 Attachments

While you are writing an e-mail you have the option to attach a file to the e-mail. To the right of the textbox an "Attach" button is visible.

If you select *Attach*, your file explorer will open and you can select a file you would like to attach to your e-mail.

If you try to forward an e-mail with an attachment, a window will appear asking you if you wish to include the attachment from which you can press either "Yes" or "No".

4 Project work plan

Since this project is rather small, we have chosen to work with agile methods since this suits our needs the best. Our project will be structured by means of the tool *Jira*, which we have set up to fit our work structure, which is heavily inspired by the Scrum technique. This report marks the **planning phase**, in which we establish the general objectives for the project.

In Jira we are going to organize our different **sprint cycles** which will, as a rule, have a duration of 1 week going from Friday to Friday. At the end of each *sprint* we will have a *stand-up meeting*, in which we will brief each other on our respective tasks that we have or are about to finish. At the end of each sprint cycle, we will discuss the coming sprint cycle and delegate tasks among each other.

We have decided to keep the roles of the company dynamic. In this way, every team member will be able to get their hands on every aspect of the project and thus get the most exhaustive outcome of the course. This will be the case when it comes to the technical development of the e-mail client. That is, both the front-end and back-end of the coding work. But we will also keep the roles dynamic in the testing phase and when writing the reports.

Asger has been appointed ScrumMaster though we are planning on working closely together throughout the project. As ScrumMaster, Asger has the responsibility to make sure that the sprint cycles are respected and that each team member has a task for the coming cycle he is responsible for. Some tasks/issues will however stand 'unassigned', which means that it is something we will work on together.

The agility in the project also allows us to not have a strict and thorough time plan for the whole project, but rather evaluate each week and set the goal for the following week. For this reason, we have changed minor things in our requirements since the beginning, but listed below are the updated requirements.

The following figure is our time plan (Jira roadmap) at the time this report was revised last. As mentioned before, due to the agility in the project every single sprint, tasks etc. had not been scheduled far in advance, but rather only the deadlines for the reports had been clearly stated as epics[2] which were constantly updated with tasks as the project went forth.

[2] An epic is a large body of work that can be broken down into a number of smaller stories, or sometimes referred to as "Issues" in Jira.

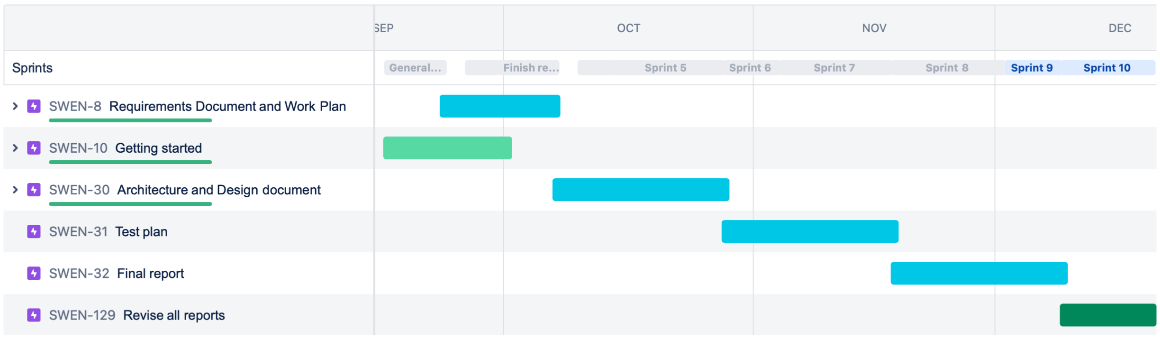

Figure 1: Current Jira roadmap

5 Requirements

In this section we are going to list the different requirements. The user requirements are deduced from the user stories in section 3.1 as well as based on interviews with our client, Lukas Esterle. In this section, each requirement will get a unique identifier which enables us to differ between them when evaluating later in the **project closure phase**.

Note that mandatory requirements, which are requirements the costumer are guaranteed, are identified by an ”M” as the last letter in the unique identifier, and that optional requirements, which are to be seen as ´nice-to-have’ features, are identified by an ”O” as the last letter in the unique identifier.

5.1 User requirements

5.1.1 Functional requirements

This section describes the functionality of the program. The way the program is to behave in specific situations. The functional requirements are those that ensure the client has the functionality needed in the program as described.
The mandatory requirements are what we as Pigeon and the client has agreed upon to be the minimum requirements of the product. We will focus on implementing these first then move on to the optional requirements.

UFM.(1) The user shall be able to log in to the client using an e-mail and its corresponding password. As a default we would like to implement gmail login features.

UFM.(2) The user shall be able to save login credentials, allowing fast login.

UFM.(3) The user shall be able to open their e-mail folders.

UFM.(4) The user should be able to read some message in a specific folder by doubleclicking on its name.

UFM.(5) One shall be able to create a new e-mail to a desired recipient or possibly more with a subject and text-body.

UFM.(6) The user shall be able to send an e-mail as described in UFM.(5).

UFM.(7) Error messages should appear if the e-mail cannot be sent somehow.

UFM.(8) The user should be able to forward a message.

UFM.(9) The user shall manually be able to fetch e-mails from the server by the click of a button.

UFM.(10) The user shall be able to see messages in a specific folder by clicking.

UFM.(11) The user should be able to reply to an e-mail.

UFM.(12) The user should be able to mark an e-mail as unread or read as the user sees fit.

UFO.(1) Be able to use different mail providers.

UFO.(2) Provide an auto respond to new e-mails.

UFO.(3) The user should be able to attach files as attachments in outgoing e-mails.

UFO.(4) The user should be able to search emails for sender, email or subject.

UFO.(5) The user should be able to save an unsent message as a draft.

UFO.(6) The user should be able to choose to write some message that acts as an automatic response when the user wants to.

UFO.(7) The user should be able to create new folders for e-mails. If UFO.(1) is implemented, this should also work for different mail providers.

UFO.(8) The user should be able to cancel very long operations. It follows that the user should have an indication of the duration of very long instructions.

UFO.(9) The user should be able to see old emails fast and without internet connection, if they choose to in settings panel.

UFO.(10) The user should be able to disable that their e-mails are stored locally.

UFO.(11) The user should be able to delete a folder if created as described in UFO.(7). If UFO.(1) is implemented, this should also work for different mail providers.

UFO.(12) The user should be able to clear the cache, such that all information on e-mails that are stored locally are cleared.

UFO.(13) The user should be able to move messages to any folder he/she has created themself.

5.1.2 Non-functional requirements

The non-functional requirements are the more behind the scenes requirements. A user expects to work fast and reliably, but this is not necessarily a function of the program more of a constraint on the system design.

UNM.(1) The e-mail client should be responsive within 1 second (long instructions should be clearly indicated for the user).

UNM.(2) If the user recieves an e-mail, the e-mail client should automatically update.

UNM.(3) The e-mail client should be able to fetch all the user's emails.

UNM.(4) The user should be able to close the program at will.

UNM.(5) If the user deletes an e-mail or moves the e-mail to another folder, the e-mail client should automatically update.

UNO.(1) The user should be able to blacklist specific senders or words detecting unwanted e-mails.

UNO.(2) E-mails that are encoded with HTML or css should be displayed accordingly.

5.2 System requirements

In this section we will list the system requirements. That is, what is needed to implement the user requirements stated above. When possible, we will reference the unique identifiers given to each user requirement.
For many of the optional user requirements we yet lack the knowledge that are needed to provide useful description of associated system requirements. This will prompt us to make further investigations if they are to implemented. If so, this will be evident in the final report where we evaluate the requirements stated in this report.

SR.(1) The e-mail client will be implemented as a Windows Forms application written in the language C# and will consist of multiple forms. The architecture will be elaborated in the next report on the project.

SR.(2) UFM.(1) is implemented using the NuGet package *Mailkit*. With this package, we will create an object, *client*, of type SmtpClient, which will try to connect to the server with the provided credentials. If the client is able to authenticate, the Mailbox will appear. If not, a try-catch statement will catch the error and a *MessageBox* will appear writing the exception that caused the error.

SR.(3) UFM.(2) will be implemented by saving the login credential locally on the owners device storage. Thus fetching it when opening the program, if the user has requested to save it. Note: this is not a safe storage of credentials, but client didn't mind.

SR.(4) UFM.(3) will also be implemented using MailKit. This time an object of type ImapClient will be created, which will *asynchronously* fetch the different folders from the client's e-mail server. These folders will be saved in a list. Thus we simply use the indexes of the different e-mails or folders to access them. The folders will be displayed in the data grid view to the left, and what ever folder is clicked is shown in the mails data grid view.

SR.(5) UFM.(9) When pressing a button the user can manually call the function that fetches emails. This would otherwise only run occasionally.

SR.(6) UFM.(4) will, once again, be implemented using MailKit and a object of type ImapClient. When doubleclicking the respective folder, the value of this field will be used to *open* this folder and thus accessing the messages inside it. This will once again be done asynchronously. This will only be done once when the user first uses the application. When fetched once, the e-mails will be saved in list of lists, the first with the folders and the second with the messages. Thus we simply use the indexes of the different e-mails or folders to access them. The values of the messages in the folder will be added to a listbox. If a message has no subject, the text ´no subject' will be displayed instead.

SR.(7) UFM.(5) and UFM.(6) will be implemented using MimeKit. When creating a new e-mail a new form will appear with different textboxes and a rich textbox for the body of the text. When pressing 'send', an object of type MimeMessage will be created. The content of the textboxes will be set as the value of the corresponding attributes of the MimeMessage. Then a new SmtpClient will establish a connection to the e-mail server and the message will be sent using MailKit.

SR.(8) UFM.(7) If one of the textboxes described above is empty, a MessageBox will appear stating the error. If it is the recipient field that is empty, this message box can only be closed. For the other textboxes, the user can choose to proceed sending the message anyways by clicking 'yes' or 'no'. If MimeKit cannot send the message due to other problems, the exception will be caught and displayed using a try-catch statement, where the try-part is the attempt to send the message.

SR.(9) UFM.(8) is implemented by sending the relevant attributes of the MimeMessage object to a new instance of the newEmail form described in SR.(7) as well as a type key indicating that this is a message to be forwarded. Then the value of the textboxes will be updated to contain the corresponding attributes of the MimeMessage providing a layout as decribed in the Forward e-mail user story.

SR.(10) UFM.(10) This is simply implemented by fetching the correct indexes of the folder and message and then displaying the message as found in the list.

SR.(11) UFM.(11) This is implemented exactly as UFM.(8) but with a different flag and different textboxes filled.

SR.(12) UFM.(12) This is implemented by fetching the correct message in the list as in UFM.(10) and then altering the flags of the message such that it now contains the flag 'Seen' if it didn't before and vice versa. Afterwards we make the corresponding action on the server.

SR.(13) UFO.(2) Automatically send a reply to every new email. Call the send email function with a set reply specified in the settings panel, saved locally. This will likely only work if the program is running.

SR.(14) UFO.(3) For this, we open a new file dialog, where the user can choose some desired file. We save the full filename and display and appropriate short filename in the UI. For adding the attachment to the message to be sent, we make an instance of a 'BodyBuilder' from the MimeKit library. With this we can simply add the full path of the attachment, which is then included in the message.

SR.(15) UFO.(4) For this we traverse through the list of lists that contain the messages and add all the messages matching the search query to a temporary list which is then displayed in the UI.

SR.(16) UFO.(5) When closing a non-empty new e-mail form, the user gets the choice whether or not to save the mail as draft (through a messagebox). If the user chooses 'yes', we append the current MimeMessage to the draft-folder *on the server*. After a few moments, the backgroundworker listening for changes will see this message and add it locally[3].

[3]Note that this gives a small delay in the UI. An alternative implementation is to find the next UniqueId that will be assigned to a message in the draft folder and then create a new msg to add locally. We have, however, not had time to make this implementation.

SR.(17) UFO.(7) By filling a textbox and pressing a button a new folder is created on server with the name of the text of the textbox. The backgroundworker sees this new folder and adds it locally. This operation is quite slow but it takes some time for the server to actually create the new folder.

SR.(18) UFO.(9) We save the emails loaded in earlier sessions to the local disc of the user. Then when the user opens the mail client it will look first in local storage and if they are not there it will fetch them from the email server. For offline viewing we try to ping google.com[4], and if this is unsuccessful, we assume that the user does not have internet-connection. If chosen so, we simply show the e-mails stored locally and disable all functionality that needs interaction with the server.

SR.(19) UFO.(10) For this we simply store in properties whether or not the local storage functionality is enabled (can be disabled with a checkbox in settings). If *disabled* we delete the json-files when the mailbox-form is closed.

SR.(20) UFO.(11) It is only possible to delete folders that the user has created themselves, since gmail does not allow otherwise. If an appropriate folder is marked, and the delete button pressed, the folder is first deleted locally and afterwards the same action is made on server.

SR.(21) UFO.(12) This is implemented in the exact same way as when the user disabled local storage. When deleting the files, we restart the application, and the retrieval of folders and messages begin.

SR.(22) UFO.(13) A dropdown menu containing all existing folders with the exception of a few[5] is visible as well as a 'move message' button. When selecting a folder from the dropdown and clicking the button, the message is first moved locally (using the list of lists structure) and then on the server by using MailKit functions.

SR.(23) UNM.(1) When instructions known to take a long time are to be executed, a message is displayed in the right hand corner indicating what the email client is working on, integrating the changes whenever it is ready.

SR.(24) UNM.(2) and UNM.(5) The client automatically runs a thread in the background that checks the server for any updates and shows these whenever they are received.

SR.(25) UNM.(3) There is no limit on how many emails the user can fetch from the email server. This also helps when needing to search old emails.

SR.(26) UNO.(1) We have made a separate form that can be spawned from settings. Here the user can add e-mails and words to be blocked. These are saved to json-files in a similar manner as for e-mails and folders. When the backgroundworker registers a new message in checks if it is contained in the blacklist. If so, the message is moved to spam first locally then on the server.

SR.(27) The user has a windows computer, able to execute the program.

SR.(28) The user has an existing gmail account and has generated a valid code for use in third party programs.

[4]We have chosen this since google is virtually *never* offline. There can, however, be exceptions if google's servers are down for instance.

[5]It does not make sense to move a message to draft, nor to sent e-mails or trash since these functionalites are implemented for themselves.

SR.(29) UFO.(12) The e-mail clients shall store the information of all e-mails in json files. So when the user clears the cache all the json files should be deleted.