Eine Email-Client-App entwickeln

Noah Vogt & Simon Hammer

Geschrieben im Jahr 2021

Inhaltsverzeichnis

1 Vorwort

Ich (Simon) habe lange Zeit mich darüber aufgeregt, dass es keine Edubs-Mail-App gibt. Ich musste die Browserversion als Link auf meinem Smartphone Bildschirm einfügen und ich habe mich sehr darüber genervt, dass ich mich jedes mal beim öffnen der Browserversion neu anmelden musste. Als dann die Zeit kam mich für eine Maturarbeit zu entscheiden, wollte ich zuerst einen Edubs-Client erstellen, welcher für Schüler gut geeignet ist. Jedoch kannte ich bis dort nur Python und wollte mich mit meiner Idee mit Noah unterhalten, da ich wusste, dass er sicher auch etwas programmieren wird, weil er sich schon seit längerem mit Computern und der Programmiersprachen befasste. Er war der Meinung, ich könnte dies doch gleich für jede Art von Email machen. Da ich mich noch nicht gut mit dem Programmieren auskannte, fragte ich ihn, ob er diese Maturarbeit denn mit mir machen würde. Zu meiner Überraschung war er von der Idee überzeugt und wir einigten uns darauf, einen Email-Client zu erstellen, welcher zwar nicht für Schüler optimiert wurde, aber für den allgemeinen Gebrauch verbessert wurde.

2 Einleitung

Das Kommunikationsmittel Email ist auch nach seinem fünfzigjährigen Jubiläum noch rege im Alltagsgebrauch vieler Leute in den meisten (industrialisierten) Ländern. Es könnte angenommen werden, dass sich auch in den letzten zehn Jahren - nämlich seit dem Aufkommen der massentauglichen Smartphones - die Emailsoftware verbessert hat. Doch für das Smartphone war es für uns, die Ersteller dieser Maturarbeit, schwer einen guten Email Client zu finden. Diese Maturarbeit ist unser Teil, daran etwas zu ändern, wenigstens für uns.

2.1 Ziel der Arbeit

was, wie und warum [machen wir das?]

3 Konzept der Arbeit

3.1 Funktionsweise

3.1.1 Vergleich mit Konkurrenz

Disclamier/Note: Da bei dieser App einen Wert auf Sicherheit und Endnutzer-Freiheit gesetzt wird, wird sie dementsprechend nur mit Apps verglichen, welche auch Freie Software sind, nach Definition [der FSF??]. Somit fallen jegliche Proprietäre Produkte raus, da sie unseren Grundanforderungen eines Email Clients nicht entsprechen.

Unsere App soll die Basisfunktionen eines klassischen Email Clients erfüllen. Dazu gehören das lesen und schreiben von Emails, öffnen und Anfügen von Anlagen, die Setzung einer Email-Signatur und das Erstellen von Entwürfen.
Wenn wir die meisten anderen quelloffenen, noch maintainten Open Source Email Clients anschauen fällt sofort auf, dass diese unglaublich überladen (bloated) sind. Selbst wenn im Internet nach einem möglichst simplen Email Client für Android gesucht wird, stösst man dabei meist auf Apps wie k-9 Mail, welche hunderttausende Zeilen Source Code besitzen.

Im Unterschied zur Konkurrenz soll diese App so programmiert werden, dass sie alle nötigen Grundfunktionen für einen Email Client auf dem Smartphone beinhaltet, aber schneller starten soll als die Apps der Konkurrenz, weniger Speicherplatz und Resourcen verbrauchen soll und nicht mit unnötigen Funktionen überladen sein.

3.2 Quellcode Modell

Um ein Programm zu programmieren, werden menschenlesbare Instruktionen in ein oder mehrere Textdateien geschrieben, die dann in Sprache, welche die Computerhardware interpretieren und ausführen kann, übersetzt. Bei diesem Prozess geht die Lesbarkeit für den Menschen grösstenteils verloren. Um ein Programm ausführen können braucht der Nutzer also keinen Zugang zum Quellcode. Doch wenn er wissen will, was das Programm macht - es könnte ihn ausspionieren oder andere bösartige Sachen im Hintergrund machen - oder einfach das Programm verändern will, braucht er unbedingt Zugang zum Source Code.

Durch die zunehmende Kommerzialisierung und Massentauglichkeit der Computer haben sich aber monetäre und andere Anreize gebildet, den Source Code dem Nutzer nicht mehr zur Verfügung zu stellen. Das hat für den Nutzer verschiedene praktische Folgen. Aber vor allem ist er nicht mehr in Kontrolle seines Computers, Betriebssystems und Programms, sondern der Besitzer oder Copyrightholder des Programms. Das ein Programm einen Besitzer haben kann, ist ziemlich absurd und nur durch die Veränderungen der westlichen Copyright- Gesetze- und Kultur der letzten 150 Jahre [möglich?].

Doch auch bei Open Source Software gibt es verschiedene Lizenzmodelle welche gewählt werden können. Diese lassen sich gut ein zwei Kategorien unterteilen:

Permissive Licenses: These give you the right to run the program for ANY purpose, study the source code, change it and redistribute the changes. Their names come from the fact, that they are “permissive” when it comes to their few restrictions: They do not put of a lot of restrictions on the distribution of the source code, and often allow the software to be forked under ANY terms. This means it is possible to make your fork of a permissive-licensed program proprietary.

This means they do not protect their code from being taken and used in other projects, even proprietary, without any giving back, which is a definite drawback and the reason people insult these licenses as “cuck licenses” as you “write proprietary code for free for big tech without increasing anyone’s computer usage freedom”. A positive point of these licenses however is their simplicity: The MIT license for example only consists of 20 SLOC, while a Copyleft license as the GPL v3 uses more than 600 SLOC and its jurisdictional jargon is much harder to understand.

Copyleft Licenses: This is an obvious play on the word “Copyright”, as it tries to invert its effects. As with permissive licenses you can run the program for ANY purpose, study, change the source code and redistribute the changes. But the trick comes with their restrictive redistribution terms: You can ONLY redistribute the program if your version provides the same user freedoms. This is usually done in only allowing redistributing using the same license.

To provide such protection of the user freedoms in future forks, there needs to be a lot of jurisdictional jargon in the license which makes it significantly longer and harder to understand for the average reader. But most of the people in Free Software who care about user freedom say this is the premiere license for your free software projects, as it ensures that freedoms have to be granted forever and stopping proprietary code maker profiting from their written free code.

3.3 Philosophie (suckless)

(Nowadays) a lot of Open Source Developers pride themselves in writing software with much features that cater to the non-technical enduser. This results in having a large codebase, which gets bigger and bigger with every release. This makes it harder to maintaining the ever growing codebase, more and more bugs occurs, security and (most importantly) performance struggles under these conditions. This degrades the quality of software technology as it is the mainstream narrative to “save time and money”.

This is where the “suckless philosophy” comes in place: It aims at making software with simplicity in mind: Less source lines of code to not render the project unmaintainable in similar way as mentioned above. This way of programming is a lot more difficult, but the struggle is most of the time worth it. This coding philosophy also incentives (quality) code rewrites - which happens a lot less with bloated software counterparts - that gives the user more alternatives to choose from.

3.3.1 Hintergründe, Technologisch, UNIX, KISS

¡¡¡¡¡¡¡ HEAD

3.4 Lizensierung

Die Unterschiede zwischen verschiedenen Source-Code Modellen und deren lizenzirrung wurden bereits besprochen. Der Grund weshalb die GNU General Public License Version 3 (kurz: GPL v3) ausgewählt wurde, ist dass sie eine der bekanntesten und stärksten copyleft Lizenz ist.

3.4.1 Hintergründe, philosophisch, technologisch

4 Arbeitsprozess

4.1 Hardware

4.1.1 Smartphones

Es wurden verschiedene Smartphone Modelle genutzt um die App auf verschiedenen Androidversionen testen zu können. Sie wurden auch genutzt um unsere App mit der Konkurrenz zu vergleichen und die Konkurrenz zu begutachten.

4.1.2 PC / Laptop

Der Gebrauch von Computern wurden gemacht um die Software, die Dokumentation und dieses Maturschreiben zu erstellen.

4.2 Software

Aufgrund dessen, dass ein umfassendes Programm entstehen soll, wird auch Gebrauch von einigen anderen Programmen, Bibliotheken und sonstigen Tools gemacht. In den nachfolgenden Seiten wird beschrieben welche Programme genutzt werden, wieso diese ausgewählt wurden und wie der Umgang mit Ihnen war.

4.2.1 Programme

4.2.1.1 Version Control System

For developing software above a certain level of complexity, it helps to use a so called VCS (Version Control System). This is a software/system that manages the versioning of software. This is useful so you don’t have to for example create MANUALLY a new folder for every new version and copy the code with the new changes in there. This would be not very user friendly, wasteful of disk space and overall very inefficient. These systems also make it easy to synchronise projects to a remote server to allow for easy collaboration on multiple devices and for different developers.

It also allow for additional advanced features like branching a repository, which means developing multiple features separately from each other to then later merge the changes together, when each of the features are working as expected.

4.2.1.1.1 Git Git und GitHub sind wohl die wichtigsten Programme die genutzt wurden. Sie sind Systeme, welche Fileordner (repository) verwalten können und sie für mehrere Computer zur Verfügung stellen, wobei sie sehr viele praktische Funktionen mit sich bringen. Mit Git können repositories local auf Computer oder Hardware geteilt werden, mit GitHub könne die repositories auch über das Internet geteilt werden. Der einfachheitshalber wird nicht zwischen Git und GitHub unterschieden.

4.2.1.1.2 Github This is our remote git server. But is is more than just a git server, it is a popular platform with over 50 million (software) repositories that people search to find interesting software. So we can use this to make our software more searched and popular (but this is not considered of great importance).

4.2.1.2 IDE

IDE stands for "Integrated Development Environment". This means a program where the coder edits, debugs the code. It provides more features than just editing text, but also advanced autocompletion features and warnings/suggestions that the programmer should do something differently because of various reasons of bad programming practice. These programs have a lot of great features, but are often buggy, resource hungry and slow, like our choice of IDE: Android Studio.

4.2.1.3 Texteditor

Text editors to have less "advanced" coding-related features, so they are not optimal for big projects with lots of files and graphical content.

4.2.1.4 Android Emulator(s)

An emulator is a program that simulates an device/operating system so that you can run inside another operation system. We use an Android Emulator inside of our GNU/Linux running on our computer, so that we do can test our app without always having to transfer it to an actual phone running android natively.

4.2.1.5 Compiler

gradle, maven

4.2.1.6 Installer / Packaging

When we compiled our app using gradle, we get an .apk file as its output. APK stands for Android Package, which is the Package Format used for installing most of the software in Android. However, there are to sorts of apk that are compiled: a debug apk and a release apk. The debug version has extra features compiled in that help auditing bugs and new features not ready for using in the app available to users. The release Version has the current version number (?) added to it and is what you need to distribute of your working version to your users.

4.2.1.6.1 signing keys Nearly all Android Systems are configured to only allow application that are signed with a key from the developers. This is done for various reasons like security, compatibility checks and other (TODO: search more on this). So before we can install our app on 99% of Smartphones, we first have to sign it with our cryptographic key we have created. Of course, this is only needed for actual releases and not for using debugging version e.g. in your android emulator.

4.2.1.6.2 Android Debugging Bridge This is a software tool to interact with android devices that are connected to a computer via usb. With it you can do useful thinks like screenshots, screen recordings, transferring files to and from your computer and installing / removing packages.

4.2.1.7 Open Source Programme

Beim Programmieren kann es sehr Hilfreich sein Programme zu haben welche, ähnliche Funktionen haben wie das Programm welches entstehen soll. Solche Vorlagen können beliebig getestet und verändert werden. Simon hatte zu beginn Schwierigkeiten Java zu nutzen, um Programme zu schreiben, da er noch nicht viel Erfahrung mit dem Programmieren hatte. Um sich mit der Art der Sprache und des Programmierens vertieft auseinander zu setzen, begann er Email-Apps, welche Open Source waren, genauer zu Betrachten. Im folgenden Text werden wir diese Programme aufführen und zeigen für was wir sie gebraucht haben.

4.2.2 Librarys

(bedeutung im glossar erklärt)

4.2.2.1 RecyclerViewer

Der Recyclerviewer ist ein Behälter in welchen Daten gepackt werden. Er wird dem Layout hinzugefügt und hat eine grossen Vorteil gegenüber Listen. Eine Liste wird einmalig erstellt und komplett generiert. Das heisst es gibt Behälter, welche existieren, aber nicht auf dem Bildschirm angezeigt werden. Diese Behälter brauchen aber dennoch Speicherplatz, sind aber sinnlos. Hingegen der Recyclerviewer generiert nur so viele Behälter wie auf dem Bildschirm angezeigt werden können. Die Behälter, welche beim Scrollen am Bildschirm ende ankommen werden sogar wieder verwendet, mit neuen Information gespeist und am anderen Ende des Bildschirms angezeigt.
Bild von RecycleView
Ich (Simon) habe viel Zeit in einem template für den Recyclerviewer verbracht, da mir viele Internetseiten nicht helfen konnten. Es waren die ersten Schritte um richtig zu verstehen wie eine App funktioniert und wie sie Aufgebaut ist. Ich habe mich davor schon informiert jedoch wurde es mir dort richtig klar. Ich habe gelernt wie Behälter über einen Key von Java Files aufgerufen werden und mit Daten gespeist werden. Ebenso habe ich verstanden, weshalb in Programmen, basierend auf Java, viele Klassen erstellt erstellt werden müssen, weil jeweils nur einmal die Variablen, Funktionen und Konstruktor einer andere Klasse implementiert werden können. Das heisst, wenn eine Klasse zwei Funktionen aus zwei unterschiedlichen Klassen verwendet werden soll, muss eine der beiden Klassen die andere Klasse implementieren.

4.3 Recherche Tools / Quellen

4.3.1 Internet

In the world wide web we use search engines like DDG (duckduckgo.com), the Arch Linux Wiki, Wikipedia and more programming-related websites like SOV (stackoverlow.com).

4.3.2 Bücher, Artikel

We used different books and articles physically and digitally sometimes for the programming/coding process, but mostly for the text of this paper.

4.4 Dokumentation(-stools)

4.4.1 Latex

To write more formatting-demanding papers like in this paper it was made use of the typesetting engine LaTeX. This is a highly extensible and efficient way for getting your paper to look just like you want it. Also professionals use it to format a big majority of published books since $\tilde{2}0$ years up to this day(TODO: look up on facts/statistics for this).

4.4.2 Pandoc

This is program to convert from one document format to another. This is useful to write a document in format with very simple syntax (e.g. markdown) to convert it to a format that allows much better formatting (e.g. LaTeX) without having to use the more sophisticated syntax of the latter format. This is often used to write documentation or short texts with the minimal amount of time and effort to do so.

5 Programmstruktur

5.1 Sicherheit / Security (Features)

Considering that on 99% of consumer smartphones the users do not even have root access (but which the "owner" of the smartphone software do have), people that are concerned with privacy or security would not use their important mailboxes on their smartphone anyway, but rather on their better secured computer running a Free Software Operating System etc.

Also taking in mind that the email protocol was written in a time with a still very limited access to computers, privacy and security where not in mind of its creators at the time is quite understandable. Why it still lasted to this day in this state is compromised on one side by networking effects and on the other side that simply most people neither know to technical detail nor care.

Nowadays it is possible to encrypt to content (body) of an email, but not the metadata, here the specification of the email protocol is to blame. This and just the fact that is was not BUILT as a secure or private way of messaging, makes emails not useful for these kinds of conversation. If you want to send your friends new plans on overtaking the world and establishing a global catholic monarchy without wanting anyone to know, your are a fool if you use email for that(granted, this is an over exaggerated example, but I hope the idea is clear).

So the conclusion for our application project is to not bloat up our app with hard-to-use security functions that just bloat the app and codebase unnecessarily. However we use sane security-focused default now further explained.

5.1.1 PRIVATE MODE, Sandbox

When storing user settings (and data?), you can choose between different permission modes in android. We choose PRIVATE MODE, which means that apps with user permissions can not see its content. But Google, root users, and apps with root access can easily bypass this restriction of the android permission system.

To prevent this type of exploitation we would have to encrypt ALL sensitive user data with an encryption key. But when there is a keylogger installed, even this is not safe. So considering that there is not a single android phone free from proprietary operating system and firmware code(when connecting to the internet via WiFi), this feature idea has been rejected as out of place and to make our codebase simpler.

5.2 Code Kompaktheit

This is something that we have absolutely achieved, we used so much less SLOC than any of the competing email clients on android.(TODO: compare use of libraries, and if we used more or less than the other apps) This is also very crucial to stand with our initial goal and the suckless philosophy. While we used $\tilde{4}$000 SLOC, other apps use a whopping 300'000 SLOC.

5.2.1 Maintaining

Our codebase should be very easy to maintain: The main things you would have to do is check if the dependencies are up to date and if they can be considered deprecated.

Even if this project gets abandoned by their original developers - or as some people call that: when it get orphaned - and if someone finds it and likes it, he can easily read through it and understand the code without too much effort needed of understanding the codebase. Would the same be the case for bloated programs like thunderbird? I don't think so either.

5.2.2 less bug/error-prone

This is thanks to the suckless nature of our coding philosophy, but also our execution. It could be the case however that bad practices may have been used as the initial project owner did not have a lot of experience of java or android coding.

6 Umsetzung

6.1 neutraler Bericht

verweis texdiary

6.2 Beispiele aus der Umsetzung

6.2.1 Bugs

Erklärung woher, warum falsch, wie gelöst GUI stuff -¿ onliner commit

6.2.2 Probleme / Hiccups

RecyclerViewer, shit java IMAP libraries

6.2.3 Kommunikation

simon fragen, war Kommunikation zwischen uns gut?

7 Einschätzung / Schlussfolgerung

7.1 Fremdeinschätzung

7.2 Selbsteinschätzung

7.2.1 was lief gut

komm., vcs, java syntax (allg.)

7.2.2 was lief schlecht

******* java shit tier libraries

7.2.3 was würden wir gleich machen

As long as java is not deprecated in the future for android programming, we would still use it as it is the most "native" way of programming. The new java clone/fork of google, Kotlin, would be a worse choice in our eyes, as your are even more dependent on google code, which you are already by using android.

7.2.4 was würden wir anders machen

As java is much slower than language that can be compiled into native binaries we would try to use more C or C++ libraries to improve speed and portability.
use more c libs for speed and less java (that's always a good thing)

7.2.5 abschliessende persönliche Schlussfolgerung

8 Danksagung