Eine Email-Client-App entwickeln

Noah Vogt & Simon Hammer

Geschrieben im Jahr 2021

Inhaltsverzeichnis

1 Vorwort **3**

2 Einleitung **3**
2.1 Ideenfindung . . . 3
2.2 Ziel der Arbeit . . . 3

3 Konzept der Arbeit **3**
3.1 Funktionsweise . . . 3
3.1.1 Vergleich mit Konkurrenz . . . 3
3.2 Quellcode Modell . . . 3
3.3 Philosophie (suckless) . . . 4
3.3.1 Hintergründe, Technologisch, UNIX, KISS . . . 4
3.4 Lizensierung . . . 4
3.4.1 Hintergründe, philosophisch, technologisch . . . 5

4 Arbeitsprozess **5**
4.1 Hardware . . . 5
4.1.1 Smartphones . . . 5
4.1.2 PC / Laptop . . . 5
4.2 Software . . . 5
4.2.1 Programme . . . 5
4.2.1.1 Version Control System . . . 5
4.2.1.1.1 Git . . . 5
4.2.1.1.2 Github . . . 5
4.2.1.2 IDE . . . 6
4.2.1.3 Texteditor . . . 6
4.2.1.4 Android Emulator(s) . . . 6
4.2.1.5 Compiler . . . 6
4.2.1.6 Installer / Packaging . . . 6
4.2.1.6.1 signing keys . . . 6
4.2.1.6.2 Android Debugging Bridge . . . 6
4.2.1.7 Open Source Programme . . . 6
4.2.2 Librarys . . . 6
4.2.2.1 RecyclerViewer . . . 6
4.3 Recherche Tools / Quellen . . . 7
4.3.1 Internet . . . 7
4.3.2 Bücher, Artikel . . . 7
4.4 Dokumentation(-stools) . . . 7
4.4.1 Latex . . . 7
4.4.2 Pandoc . . . 7

5 Programmstruktur **7**
5.1 Sicherheit / Security (Features) . . . 7
5.1.1 PRIVATE MODE, Sandbox . . . 7
5.2 Code Kompaktheit . . . 7
5.2.1 Maintaining . . . 7
5.2.2 less bug/error-prone . . . 7

6 Umsetzung **7**
6.1 neutraler Bericht . . . 7
6.2 Beispiele aus der Umsetzung . . . 7
6.2.1 Bugs . . . 7
6.2.2 Probleme / Hiccups . . . 7
6.2.3 Kommunikation . . . 8

7 Einschätzung / Schlussfolgerung **8**
7.1 Fremdeinschätzung . . . 8
7.2 Selbsteinschätzung . . . 8
7.2.1 was lief gut . . . 8
7.2.2 was lief schlecht . . . 8
7.2.3 was würden wir gleich machen . . . 8
7.2.4 was würden wir anders machen . . . 8
7.2.5 abschliessende persönliche Schlussfolgerung . . . 8

8 Danksagung **8**

1 Vorwort

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

2 Einleitung

2.1 Ideenfindung

Simon schrib du, du heschd idee gha

2.2 Ziel der Arbeit

was, wie und warum [machen wir das?]

3 Konzept der Arbeit

3.1 Funktionsweise

Unsere App soll die Basisfunktionen eines klassischen Email Clients erfüllen. Dazu gehören das lesen und schreiben von Emails, öffen und Anfügen von Anlagen, die Setzung einer Email-Signatur und das Erstellen von Entwürfen.

3.1.1 Vergleich mit Konkurrenz

Disclamier/Note: Da bei dieser App einen Wert auf Sicherheit und Endnutzer-Freiheit gesetzt wird, wird sie dementsprechen 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.

Wenn wir die meisten anderen quelloffenen, noch maintainten Open Source Email Clients anschauen fällt sofort auf, dass diese unglaublich überladen (bloated) sind. Selbst wenn man im Internet nach einem möglichst simplen Email Client für Android sucht, stösst man dabei meist auf Apps wie k-9 Mail, welche hunderttausende Zeilen Source Code besitzen.

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

3.2 Quellcode Modell

Um ein Programm zu programmieren schreibt man menschenlesbare Instruktionen in ein oder mehrere Textdatein, die dann übersetzt werden in Sprache welche die Computerhardware interpretieren und ausführen kann. 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 ja ausspionieren oder andere bösartige Sachen im Hintergrund machen - oder

einfach das Programm verändern will braucht man umbedingt 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 vorallem ist nicht mehr er 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.

Doch auch bei Open Source Software gibt es verschiedene Lizenzmodelle welche man wählen kann. Diese lassen sich aber ziemlich 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 anyones 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 juridictional jargon is much harder to understand.

Copyleft Licsenses: 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 prupose, study, change the source code and redistrubute the changes. But the trick comes with their restricitve 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 juridictional jargon in the license which makes it significantely 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 Developpers pride themselvers 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 maintaing the evergrowing codebase, more and more bugs occure, 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 unmaintanable in similar way as mentionened above. This way of programming is a lot more difficult, but the struggle is most of the time worth it. This coding philosophy also insentivises (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

3.4 Lizensierung

The differences of different source code models and their licenses have been already discussed in this paper (?). The reason the GNU General Public License Version 3 (short: GPL v3) was chosen because it is one of the most popular and strongest copyleft licenses that suits the application.
The license comes from the Free Software Foundation and their Project to write a fully free software operation system (the GNU operating system)

3.4.1 Hintergründe, philosophisch, technologisch

4 Arbeitsprozess

4.1 Hardware

4.1.1 Smartphones

We used different Models of Smartphones running different variants of the android operating system to install and test our app. It was also used to compare it and benchmark it against our competition (the already existing mail clients on android).

4.1.2 PC / Laptop

The usage of computers was to the actual work of creating the software, documentation and the text of this paper (?).

4.2 Software

Aufgrund dessen, dass ein umfassendes Programm entstehen soll, wird auch Gebrauch von einigen anderen Programmen, Biblitheken und sonsitgen 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 developping 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 mutliple devices and for different developpers.

It also allow for additional advanced features like branching a repository, which means developping multiple features seperately 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 repositorys local auf Computer oder Hardware geteilt werden, mit GitHub könne die repositorys 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 intersting 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 Developpment 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 ar often buggy, resource hungry and slow, like our choice of IDE: Android Studio.

4.2.1.3 Texteditor

vim

4.2.1.4 Android Emulator(s)

4.2.1.5 Compiler

gradle, maven

4.2.1.6 Installer / Packaging

erklären

4.2.1.6.1 signing keys

4.2.1.6.2 Android Debugging Bridge

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

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

duckduckgo, arch wiki, stackoverflow

4.3.2 Bücher, Artikel

4.4 Dokumentation(-stools)

4.4.1 Latex

4.4.2 Pandoc

5 Programmstruktur

5.1 Sicherheit / Security (Features)

5.1.1 PRIVATE MODE, Sandbox

5.2 Code Kompaktheit

5.2.1 Maintaining

5.2.2 less bug/error-prone

6 Umsetzung

6.1 neutraler Bericht

verweis texdiary

6.2 Beispiele aus der Umsetzung

6.2.1 Bugs

erklärung woher, warum falsch, wie gelöst

6.2.2 Probleme / Hiccups

Gründe

6.2.3 Kommunikation

7 Einschätzung / Schlussfolgerung

7.1 Fremdeinschätzung

7.2 Selbsteinschätzung

7.2.1 was lief gut

7.2.2 was lief schlecht

7.2.3 was würden wir gleich machen

7.2.4 was würden wir anders machen

7.2.5 abschliessende persönliche Schlussfolgerung

8 Danksagung