FREIE UNIVERSITÄT BERLIN

Entwurf und Implementierung einer auf unterschiedlichen Anwendungsfällen aufbauenden Darstellung eines Posteingangs auf mobilen Endgeräten

Bachelor-Arbeit
zur Erlangung des Grades Bachelor of Science
im Studiengang Informatik

Vorgelegt am 29. Juni 2021 von

Hannes Hattenbach

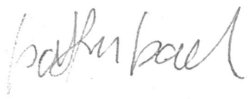
Erstgutachter	**Zweitgutachter**
Prof. Dr.-Ing. Volker Roth	Prof. Dr. Claudia Müller-Birn

Betreuer: Prof. Dr.-Ing. Volker Roth, Oliver Wiese M.Sc.

Fachbereich Mathematik und Informatik

Eidesstattliche Erklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.



Hannes Hattenbach, Berlin, den 29. Juni 2021

Zusammenfassung

Email wird für viele verschiedene Anwendungsfälle wie Reiseplanung, Datei- und Aufgabenmanagement etc. verwendet, diese Verwendung von Email für Zwecke ausserhalb der simplen Kommunikation wurden bereits untersucht. Allerdings ist das User-Interface (UI) der klassischen Inbox diesen verschiedene Anwendungsfällen bzw. Kategorien von Email nicht gewachsen und deshalb wurde für dieses Problem der Begriff der *Typ-Überladung*, also der Überladung der Inbox durch zu viele verschiedene, für die klassische Inbox ungeeignete Typen von Email, eingeführt. Ziel dieser Arbeit ist es, einen Ansatz zum Umgang mit Typ-Überladen Emails zu entwickeln und in einer iOS-App zu implemetieren. Das soll dadurch gewährleistet werden, dass verschiedene Arten bzw. Kategorien von Email mit einem eigens dafür angepassten User-Interface dargestellt werden. So könnten beispielsweise Reiseinformationen anders dargestellt werden als private Chatverläufe. Es wurden verschiede bestehende Ansätze untersucht und potenzielle Nutzer befragt, damit ein neues User-Interface konzipiert und ein Framework zum entwicklerseitigen Erstellen neuer Kategorien in der LetterBox iOS-App umgesetzt werden konnte.

Inhaltsverzeichnis

1 Einleitung und Hintergrund

1.1 Einleitung

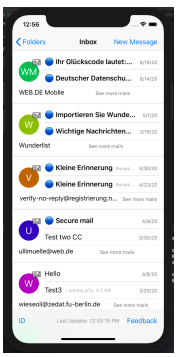
Die Verwendung von Email als Kommunikationsmittel bleibt relevant [13, 40]. Dennoch ist die Nutzeroberfläche der meisten Applikationen noch nach dem gleichen Konzept wie bei der Einführung der Email in den 80er Jahren aufgebaut. Das Problem daran ist, dass bereits mehrfach (siehe 2.1.1) festgestellt wurde, dass das ursprüngliche Konzept den vielen Anwendungen von Email nicht mehr gerecht wird.

Ein möglicher Umgang mit dieser sogenannten Typ-Überladung soll in dieser Arbeit behandelt und umgesetzt werden. Der Hauptansatz ist dabei die Einführung von Email-Kategorien mit eigener Nutzeroberfläche, welche genau für diese Email-Kategorie angepasst ist (siehe 2.1.2).

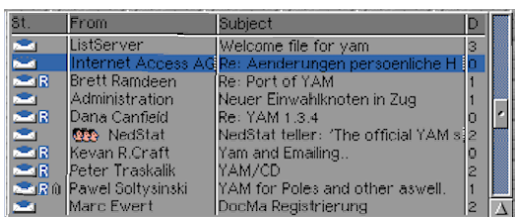
Da mit der wachsenden Nutzung von Smartphones eine Verschiebung der privaten Email-Nutzung zu mobiler Verarbeitung zu beobachten ist[46], wird in dieser Arbeit eine nutzerfreundlichere Oberfläche für Mobilgeräte konzipiert und anschließend rudimentär im Rahmen der iOS-App „LetterBox“ implementiert. Dazu werden vorerst bereits bestehende Arbeiten betrachtet, um verschiedene Ansätze und Hintergründe zu verstehen und mögliche Kategorien herauszufinden.

Da diese Arbeit im Konzept mehr Kategorien vorschlägt, als vorerst implementiert werden, soll es zukünftigen Entwicklern von „LetterBox“ erleichtert werden, weitere Kategorien zu erstellen. Dazu wird neben der Implementierung der Inbox-Hauptoberfläche ein Framework erstellt werden, welches genau dies ermöglicht.

Schließlich wird der Prototyp noch evaluiert, um verschiedene Vorteile und eventuelle Nachteile darzustellen und zukünftige Arbeiten anzuregen.



(a) Momentane Inbox von Letterbox



(b) Inbox aus 1998 [4]

Abbildung 1.1: linear strukturierte Inbox

1.2 Hintergrund

1.2.1 Letterbox

Die iOS-App LetterBox gehört dem Enzevalos-Projekt an. Das Enzevalos-Projekt wurde 2016 von der „AG Sichere Identität“ ins Leben gerufen [24] mit dem Ziel, die schon seit langem bestehenden Möglichkeiten zur Verschlüsselung und Authentifizierung von Emails der Nutzerin nahezubringen.

1.2.2 Swift und SwiftUI

Deklaratives UI-Framework

Beim klassischen, imperativen User-Interface(UI)-Ansatz werden die Interface-Layout-Bausteine durch bestimmte Funktionen (z. B. beim Drücken eines Knopfes) neu geladen, angepasst und anschließend gerendert. Im Gegensatz dazu wird in einem deklarativen UI-Framework die Nutzerschnittstelle durch eine zustandsabhängige Form definiert. Das bedeutet, die Layout-Dateien sind keineswegs statisch, sondern abhängig vom momentanen App-Zustand, dem State, definiert. Das vereinfacht den Entwicklungsprozess, da weniger auf korrektes Nachladen bzw. Neuladen von UI-Bausteinen geachtet werden muss, da die Bausteine bereits zustandsabhängig sind und sich in diesem Sinne ‚selbst nachladen‘.

SwiftUI

Die Letterbox-App verwendete bis zum Softwareprojekt WS 19/20 Apples imperative UI-Framework, UIKit. Im Rahmen dieses Softwareprojektes wurde auf das 2019 von Apple eingeführte, deklarative UI-Framework, SwiftUI, gewechselt. In SwiftUI sind diese Layoutbausteine standardmäßig spezielle Structs, die zu einem sogenantem `View protocol`[1] konform sind.
Ein paar fertige `Views` wie z. B. ein `Button` werden dabei bereits zur Verfügung gestellt. Es lassen sich aber auch neue `Views` durch Ineinanderschachtelung bereits bestehender Views erzeugen. Durch diese Ineinanderschachtelung lassen sich beliebig komplexe UIs bauen, damit wird aber auch das State-Management komplexer. Auch hier kommt einem SwiftUI mit verschieden `Propertywrappern` (siehe 1.2.2) entgegen und ermöglicht eine sehr einfache Verwendung des MVVM[2]-Patterns [20].

Swift

Zur Nutzung des SwiftUI-Frameworks verwendet man Swift, eine Programmiersprache die seit 2014 Objective-C in der Entwicklung von iOS[3]-Apps ablöst. Swift kombiniert dabei verschiedene Paradigmen sowohl aus objektorientierter als auch funktionaler Programmierung [19].

Propertywrapper

Ein Feature, welches Swift unterstützt, sind sogenannte `Propertywrapper`, diese werden wie normale Klassen oder Structs, aber mit dem zusätzlichen Schlüsselwort `@propertyWrapper`, deklariert. Dann können sie im Code mit einem @ vor Variablen geschrieben werden, um diesen Variablen zusätzliche, im Propertywrapper definierte, Logik anzuhängen.

[1] z.Dt. Protokoll ; Swifts’ Variante des Interface-Patterns
[2] **M**odel, **V**iew, **V**iew**m**odel: ein Entwurfsmuster zur Trennung von Datenmodell, des UIs und dem State/Data-Management
[3] auch andere Betriebssysteme wie WatchOS oder MacOS

Ein Beispiel dafür wäre der `@Published`-Wrapper, welcher unter SwiftUI dem ViewModel denjenigen Variablen vorangesetzt wird, die Zustands-relevant sind, d. h. ein Neu-Rendern des UIs erfordern. Das entsprechende Gegenstück dazu wäre der `@ObservedObject`-Wrapper, welcher in einem View verwendet wird, um dieses neu zu rendern, wenn eine `@Published`-Variable des `@ObservedObject` sich verändert hat.

Combine

Umsetzen lassen sich diese beiden Wrapper mithilfe des Combine-Frameworks. Ebenfalls in dem SDK[4] von Apple enthalten, ist dieses Framework dazu vorgesehen, um vor allem in der objektorientierten Programmierung einen Event-getriebenen, asynchronen Datentransfer zu ermöglichen [18]. Genauer beschrieben stellt Combine ein Publisher-Subscriber-Framework zur Verfügung.
In dem eben beschriebenen Beispiel aus `@Published`- und `@ObservedObject`-Wrappern würde der `@Published`-Variable ein Publisher hinzugefügt werden, während das `ObservedObject` einen entsprechenden Subscriber zu diesem Topic erhält. Bei Änderung des publizierten Wertes würde der Subscriber dann ein Neu-Rendern einleiten.

[4] Software Devement Kit, Entwicklungsbaukasten

2 Vergleichbare Arbeiten

2.1 Wissenschaftliche Arbeiten

Die wissenschaftliche Literatur beschäftigt sich bereits mit der Nutzung von Email, auch im Hinblick auf ihren „Overload“ (Überladung), und mit verschiedenen Lösungsansätzen.

2.1.1 Email-Überladung

1996 publizierten Whittaker et al. eine Studie, worin erstmals das Problem der Email-Überladung, also eine Verwendung von Email zu ursprünglich nicht angedachten Zwecken, beschrieben wurde [45]. Dabei wurde in 20 ein- bis zweistündigen, semistrukturierten Interviews untersucht, wie Email verwendet wird und festgestellt, dass Email nicht nur zur Kommunikation, sondern auch zur Aufgaben- und Dateiverwaltung verwendet wird, wozu Email nur bedingt geeignet war. Ebenfalls wurde nach einer Untersuchung, wie mit der Überladung umgegangen wird, festgestellt, dass es kaum mehr möglich ist, jede Email sofort zu bearbeiten. Als Lösungsansatz präsentierten Whittaker et al. die Idee des Threadings[1], um sowohl die Gesamtgröße der Inbox zu reduzieren als auch zu ermöglichen Emails als „requiring action“ zu markieren, um diese nicht aus den Augen zu verlieren [45].
Diese Erkenntnisse wurden häufig zitiert und erweitert [15], unter anderem 2014 von Grevet et al., in deren Arbeit nochmal aus aktuellerer Perspektive auf dasselbe Problem zurückgeschaut wird [17]. In ihrer Arbeit wurde hauptsächlich die Definition von Email-Überladung erweitert und in drei Kategorien unterteilt: Informations-, Status- und Typ-Überladung. Dabei ist die Status-Überladung vergleichbar mit der ursprünglich von Whittaker et al. genannten Überladung. Die Informations-Überladung bezeichnet die ebenfalls bereits von Whittaker et al. festgestellte zu hohe Menge an Information, so dass diese nurnoch schwer lesbar und überschaubar bleibt.
Der „Type-Overload“ (Typ-Überladung) ist die für diese Arbeit interessanteste Art von Überladung, da er in erster Linie in privater Email-Nutzung vorkommt: die verschiedenen Typen von Email, die alle von demselben Interface behandelt werden müssen. So wird als Beispiel für verschiedene Typen aufgeführt: „bills, medical information, clubs, newsletters, personal Emails, and others“ [17].

2.1.2 Bewältigung der Überladung durch kategorisierte Nutzerschnittstellen

Die Grundidee, auf der meine Arbeit beruht, entstammt der Arbeit eines Forschungsteams von Yahoo: Bentley et al. [14]. Das Paper untersucht anhand einiger Nutzerstudien die Nutzung von privater Email und stellt

[1] „An Email thread is an Email message that includes a running list of all the succeeding replies starting with the original Email. The replies are arranged visually near the original message, usually in chronological order from the first reply to the most recent.“ [11]

dabei fest, dass es sich in erster Linie um Business zu Customer (B2C) Email handelt. Dabei spielen Coupons und Werbung neben anderen Kategorien eine große Rolle. Außerdem stellen sie vorwiegende Nutzung auf Smartphones fest und verwenden dieses Wissen, um einen Prototyp namens ‚Cardmail' zu konzipieren. An genau diesem Prototyp möchte ich mit dieser Arbeit ansetzten. Die Hauptidee dieses Prototyps besteht darin, die Typ-Überladung zu bewältigen, indem die klassische Inbox verworfen wird, und durch eine Ansicht über Kategorien ersetzt wird.[2] Diese Kategorien beschäftigen sich dabei mit je einem Typen von Email und sollen ein dafür optimiertes UI darstellen. Zusätzlich zu den Kategorien aus der Arbeit von Grevet et al. werden hier noch Reisen und Coupons/Deals aufgeführt.
Die Kategorien sollten neben einem eigenen Interface auch eine eigene Sortierung besitzen. Nicht nur die fehlende globale Suche, sondern auch eine strikte Trennung der Kategorien bringt allerdings auch ein paar Probleme mit sich, die es zu beheben gilt.

Kategorien

Neben den in 2.1.1 genannten Arbeiten haben sich auch zwei weitere Arbeiten von Yahoo [16, 21] mit möglichen Kategorien beschäftigt. Deren Grundidee basiert darauf, dass nicht jede Nutzerin ihre eigenen Ordner erstellt, sondern, dass es eine Anzahl an nahezu allgemeingültigen Kategorien gibt [16]. Dazu wurden die von den Nutzerinnen erstellten Ordner untersucht. Die meistgenutzten Ordner waren „ebay, accounts, personal, school, saved mail, jobs, amazon, taxes, recipes, business, college, bills, house, facebook, paypal, save, food, linkedin, pictures, surveys, travel, jobs, misc". Insgesammt wurden 100000 Ordner gesammelt und damit ein LDA-Modell[3] trainiert. Mit dessen Hilfe wurden dann die Kategorien „human, career, shopping, travel, finance, social" herausgefiltert [16]. Bei einer höheren Anzahl an Kategorien wurde die Gesamtabdeckung der Emails durch diese Kategorien wieder geringer, da Nischen-Kategorien wie Häkeln, Kochen oder Astrologie extrahiert wurden. Allerdings wurde ebenfalls gezeigt, dass diese Kategorien nur teilweise mit den Ordnern vereinbar sind. So besteht zum Beispiel der Ordner „Mom und Dad" zu 55% aus Emails, die der Kategorie Finanzen zugeordnet wurden.

2.1.3 Aufgabenverwaltung

Weiterhin wurde von Whittaker et al. festgestellt, dass Email entgegen dem ursprünglichen Zweck als Aufgabenverwaltungs-Tool verwendet wird. Das heißt, dass Emails auch mit dem Typen ‚Aufgabenverwaltung' überladen sind. Mit dem Thema Aufgabenverwaltung über Email beschäftigen sich viele andere Arbeiten. So wird angemerkt, dass nicht nur die Verwaltung von eigenen Aufgaben, sondern auch die des Gegenübers behandelt werden müssen. [7, 43] Dabei ist es ebenfalls wichtig, jeder Aufgabe eine Wichtigkeit zu verleihen [7], welche aber beispielsweise per Machine-Learning (ML) zugeordnet werden kann [9, 10].

2.2 Vorhandene Software

Auch aus der Wirtschaft kommen verschiedene Softwareansätze, um eine unübersichtliche, verstopfte Inbox zu umgehen. In den folgenden Abschnitten sollen verschiedene Lösungen genauer betrachtet werden, um herauszufinden, welche Ansätze den Umgang mit Email erleichtern. Unter Umgang ist hier sowohl das

[2] dieselbe Idee schlägt eine Keystroke-Analyse vor [6]

[3] In der Datenwissenschaft wird ein LDA-Modell verwendet, um aus einer großen Sammlung von Dokumenten die latenten Themen (hier die Kategorien/ Themen/ Ordner) zu extrahieren. Siehe [8].

Sortieren und damit schnellere Finden eingegangener Emails als auch das darauffolgende Handeln (Antworten, Notieren von Informationen, etc.) und deren Auslösung gemeint.

Wie in den folgenden Abschnitten zu erkennen ist, sind die Softwarelösungen teilweise sehr unterschiedlich. Gerade deshalb lohnt sich eine Analyse dieser Software, um herauszufinden, welche Ansätze es gibt, und diese anschließend zu sortieren und zu gruppieren. Es wird sich hier auf innovative Software, vor allem im Bereich der mobilen Nutzung, konzentriert und diese kurz eingeführt. Dazu wurden die Funktionalitäten und Merkmale von 15 Email-Clients betrachtet, welche mit ihrem Interface-Design und Funktionsumfang möglichst große Differenzen von der klassischen Inbox aufweisen.

2.2.1 Google Inbox

Eine der Neuerungen kam dabei von Google mit der Einführung von „Inbox“ [31] 2014.

Hier wurden verschiedene Funktionalitäten eingeführt. Eine davon ist die „ToDo“-Funktion. Mit dieser wurde der Nutzerin die Möglichkeit geboten, sich Erinnerungen zu setzen, um beispielsweise auf bestimmte Emails zu antworten, für die man momentan keine Zeit hat (siehe Abbildung 2.1b)

Des Weiteren wurden Emails automatisch in bestimmte Kategorien eingeordnet [4], Dateien wurden direkt in der Inbox angezeigt und es gab einen gesonderten Ort für angepinnte Emails.

2.2.2 Hey Mail

Diese Ideen wurden von Hey Mail [30] aufgenommen, weitergedacht und mit neuem Funktionsumfang erweitert. Hey Mail fokussierte sich dabei auf ein ähnliches Ziel wie diese Arbeit: die Optimierung des Email-Clients im Umfeld der privaten Email-Nutzung. Besonders interessant ist dabei die Einführung von Kategorien mit entsprechend dafür optimiertem UI (siehe Abbildung 2.1a). Bei den verwendeten Kategorien handelt es sich im weitesten Sinne um Abschnitte für neue Absender, News, Belege und Ähnliches, Dateien, „später Antworten“ und einen Abschnitt zum ablenkungsfreien Beantworten von offenen Emails.
Zusätzlich hat Hey Mail noch einige kleine Features, wie z. B. die Möglichkeit Betreffe umzubenennen, Threads zu vereinen oder zu blockieren, implementiert.

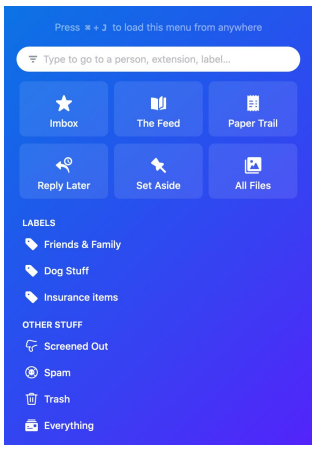
2.2.3 Spike

Einen ganz anderen, messaging- und personenorientierten Ansatz verfolgt Spike mit seiner Applikation [37].
Statt konventionell die Email nach Eintreffzeitpunkt zu sortieren, sortiert Spike nach Konversationspartner, wie es aus Messaging-Apps wie Telegram oder Whatsapp bekannt ist. Auch die Ansicht, in welcher die Email gelesen wird, ist bei Spike stark von Instant-Messagern inspiriert (siehe Abbildung 2.1c).
Spike setzt seinen Fokus also auf persönliche, informelle Interaktion und hat sein UI dementsprechend aufgebaut.

2.2.4 Weitere

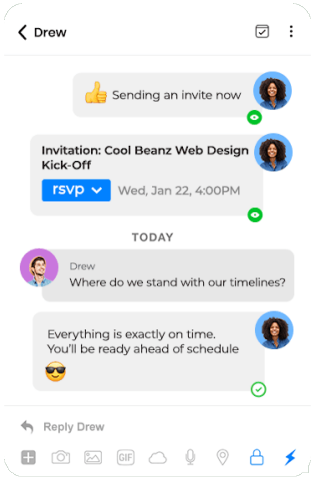
Neben diesen Email-Clients wurden noch weitere innovative (mobile) Email-Clients, wie Spark, Airmail, Plummail, Newton, Polymail, Unibox, Outlook und Thunderbird, berücksichtigt und deren Ideen herausge-

[4] Die automatische Einordnung von Email wurde allerdings, anders als die Idee von Kategorien an sich, von S. Whittakers und C. Sidners Nutzerstudie s[45, p. 283] als tendenziell unerwünscht klassifiziert.



(a) Eine Übersicht über die verschiedenen „Kategorien“ von HeyMail [2]

(b) Erinnerungen für einen bestimmten Zeitpunkt setzten in Inbox by Google [1]



(c) Spike’s Instant-Messaging ähnliche „Email Threads“ [3]

Abbildung 2.1: verschiedene innovative Funktionen bereits bestehender Email-Clients

arbeitet. Darunter befinden sich Ideen, die nicht direkt mit dem Lesen, sondern mit dem generellen Umgang mit Email zu tun haben, wie beispielsweise das geplante Senden einer Email zu einem bestimmten Zeitpunkt [29, 35, 36, 37, 32].
Aber auch viele Ideen, die zur Konzeption eines „Lese- und Finde-“ UI zumindest berücksichtigt werden sollten. Diese lassen sich grob verschiedenen Kategorien zuordnen:

To-Do und To-Read

Viele der Implementationen beschäftigen sich mit der Verwaltung und Bewältigung von Aufgaben.

Unter anderem geht es darum, Emails zu markieren, um diese schneller wiederzufinden. Bekannt ist diese Funktion häufig unter Begriffen wie ‚Star‘ [33, 38] oder ‚Pin‘ [36]. Viele Clients bieten die Funktion eine Email als ‚benötigt Aktion‘ zu markieren [31, 37, 36, 32]

Ebenfalls gibt es oft die Möglichkeit Emails für spätere Bearbeitung zu verschieben, darunter ‚Snooze‘ [32, 29] sowie ‚Zur Seite Legen‘ [30, 32].

Außerdem existieren Funktionen, um sich spätere Erinnerungen für Emails zu setzen [37, 36, 31, 32] oder um sich Notizen direkt an die Email zu schreiben [30] bzw. bestimmte Regionen einer Email hervorzuheben [30, 34].

Private „casual“ Kommunikation

Zwar sind Spike und Unibox [39] wohl die Apps, die von allen untersuchten den Ansatz der personenbezogenen Kommunikation am stärksten umsetzen; neben der Gruppierung nach Kontakten bzw. Gruppen [37] und der Chatansicht gibt es aber noch weitere interessante Ansätze auf diesem Gebiet: die Lesebestätigung

[35, 37, 32] sowie eine spezielle Kontaktansicht mit möglichst vielen Informationen über diesen Kontakt [35][5] und auch der Möglichkeit, Kontakte oder Threads stumm zuschalten / zu blockieren [30, 34, 37] oder für unterschiedliche Kontakte eigene Benachrichtigungstöne zu verwenden [37].

Threads

Darüber hinaus befassen sich viele Apps mit der Gruppierung von Nachrichten zu Threads, wobei die meisten diese nicht nur implemetieren, sondern um Funktionen erweitern. So gibt es Möglichkeiten, um:

- mehrere Threads zu einem zu vereinen [30];
- Threads als beendet zu markieren und dementsprechend nach Beendung dieses Threads keine weiteren Nachrichten mehr von diesem zuzulassen sowie bei Beendung eines Threads durch einen anderen Teilnehmer gesondert benachrichtigt zu werden [34];
- Aktionen auf viele Emails gleichzeitig anwenden [37].

[5] Bei Polymail handelt es sich um ein Client für Unternehmen, die auf dieser Kontaktübersicht bereits Informationen aus LinkedIn etc. angezeigt bekommen

3 Entwurf eines Prototyps

Die vorgenannten Ideen wurden nach ähnlichen und zusammengehörigen Konzepten gruppiert.[1] Die Gruppierungen, die dabei entstanden sind, lassen sich in ein paar Funktionsgruppen unterteilen:
Aufgabenverwaltung (siehe dazu auch 7.1.3) bzw. ToDos sowie ToReads, Threading, Features aus Instant-Messagern als auch die Hervorhebung wichtiger Informationen.
Hauptaugenmerk wurde allerdings auf die verschiedenen Kategorien gesetzt. Hier bleibt die Grundidee, die klassische Inbox durch eine kategorisierte Inbox zu ersetzen, bei der jede Kategorie ihr eigenes UI[2] mit ihrem eigenen Sortierungskonzept bekommt [14]. Vergleiche hierzu auch Absatz 2.1.2. Diese Idee kann, ähnlich wie in einer Arbeit von Whittaker et al. [44], durch die Erstellung unterschiedlicher UIs für unterschiedliche Nutzer erweitert werden. Vorerst lag allerdings der Fokus auf der generellen Erstellung von allgemeingültigen Kategorien.

3.1 Erster Prototyp

Aufgrund dieser Ideen wurde ein erster Prototyp zur weiteren Evaluation konzipiert. Die Grundidee des Prototyps basiert auf dem Prototyp „Cardmail“. Dementsprechend ist auch hier der Hauptfokus darauf gelegt, die Typ-Überladung mit eigenen Nutzer-Oberflächen für jeden ‚Type‘ anzugehen.
Dazu wurden in einem schnellen iterativen Prozess auf Papier mehrere Low-fidelity-Designkonzepte skizziert, welche daraufhin mithilfe von Figma[3] zu einem High-fidelity-Prototyp mit 3 Ebenen (Startansicht, geöffnete Kategorie, geöffnete Email) sowie einigen austauschbaren Elementen umgesetzt wurden (vergleiche hierzu Abbildung C.1).

3.1.1 Startansicht

Die Startansicht ist wohl der wichtigste Grundbaustein meines Konzeptes. Im Prototyp von „Cardmail“ erhält man beim Start nur eine statische Übersicht über alle Kategorien (siehe Abbildung 3.1).

Im Gegensatz dazu wird man in verschiedenen Versionen meiner Prototypen nicht mit einer statischen Übersicht über alle Kategorien begrüßt, sondern einer dynamischen Ansicht aus Previews der zuletzt verwendeten Kategorien (siehe Abbildung 3.2). „Verwendet“ bedeutet in diesem Fall das Eintreffen einer Email. Darüber hinaus lassen sich Kategorien (z. B. Spam oder Gutscheine) ‚stumm‘ schalten. Diese könnten dann am unteren Ende der Hauptansicht bleiben. Andere Sortierungsmöglichkeiten sollten aber zusätzlich einstellbar sein.

[1] Die Gruppierungen und Zusammenhänge können unter C.2 nachvollzogen werden
[2] User-Interface z.Dt. UI
[3] Ein Designertool zum Erstellen von high-fidelity Prototyp (siehe https://www.figma.com/)

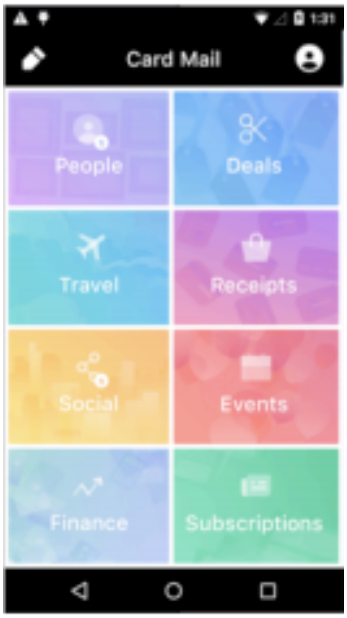


Abbildung 3.1: „Cardmail“ [14]: statische Übersicht ohne Previews

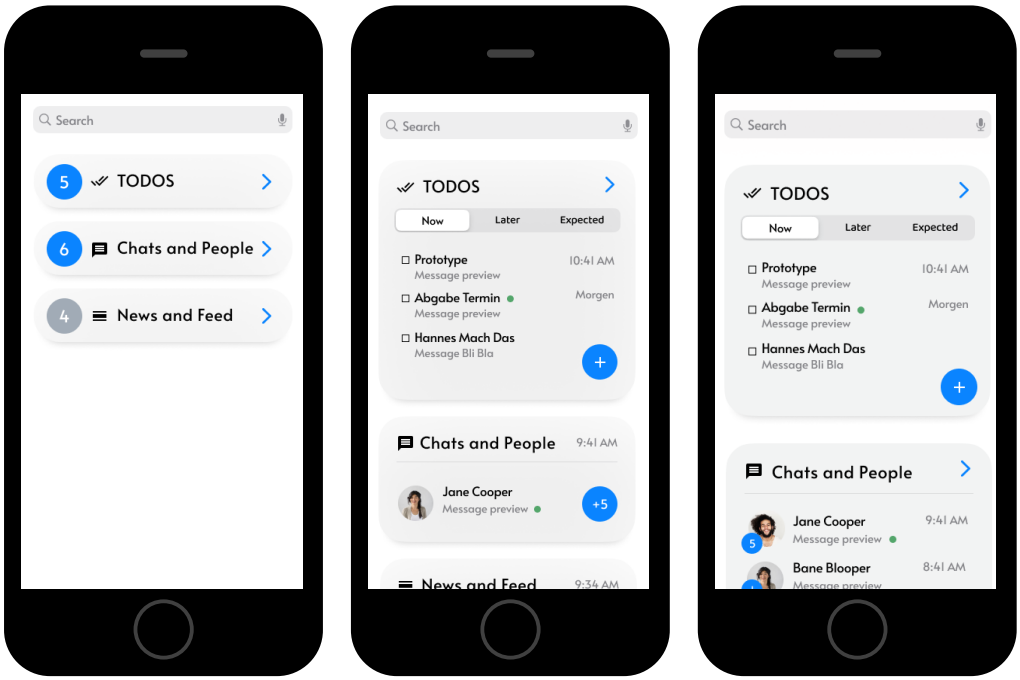
3.1.2 Previews

Da nun einzelne Previews der Kategorien mit den letzten oder wichtigsten Emails, die zu dieser Kategorie eingetroffen sind, angezeigt werden, ist die Größe bzw. die Komplexität der jeweiligen Previews zu überlegen. In Abbildung 3.2b sieht man einen Prototyp mit variablen Größen, wobei eine Kategorie aufgrund ihrer Wichtigkeit hervorgehoben wurde und ein größeres, priorisiertes Preview bekam: die ToDos. Dementsprechend wird im folgenden Absatz dargestellt, wie die Designentscheidungen eines Previews am Beispiel der ToDos aussehen könnte.

ToDos

Bei den ToDos es wichtig, dass eine unmittelbare Übersicht über die Aufgaben mit möglichst wenigen Klicks für die Nutzerin zu erreichen ist, und die Aufgaben nicht in irgendeinem Ordner verschwinden [7]. Genau das wird in den Prototypen durch ein großes Preview über die ToDos am oberen Ende der Inbox gewährleistet. Auch innerhalb der ToDo-Ansicht könnte die Sortierung einstellbar sein. Eine Sortierung nach Wichtigkeit, welche beim Erstellen des ToDos (z. B. durch einen Slider) wählbar sein sollte, wäre eine gute Alternative zum Einstellen und Sortieren nach einer Deadline.
Beim Einstellen eines ToDos könnte sowohl eine Auswahl an Passagen, die ein ToDo sein könnten, als auch deren Wichtigkeit gestützt durch ML vorgeschlagen werden [10, 9]. Dabei könnten auch die jeweiligen Kategorien, denen diese Email zugeordnet ist, berücksichtigt werden [14]. Die konkrete Umsetzung der ToDos ist allerdings nicht Teil dieser Arbeit, dennoch sollten diese Erkenntnisse berücksichtigt werden.

Die Idee, wichtige Passagen einer Email via ML herauszufiltern, lässt sich allerdings auch auf die Previews generell anwenden. So könnten statt der ersten, die wichtigsten Zeilen des Inhaltes einer Email angezeigt werden. Vorerst ist allerdings die Idee, die Gestaltung der Previews den einzelnen Kategorien zu überlas-



(a) keine Previews (b) dynamische, kleine Previews (c) dynamische, große Previews

Abbildung 3.2: Prototypen der Hauptansicht der neuen Inbox

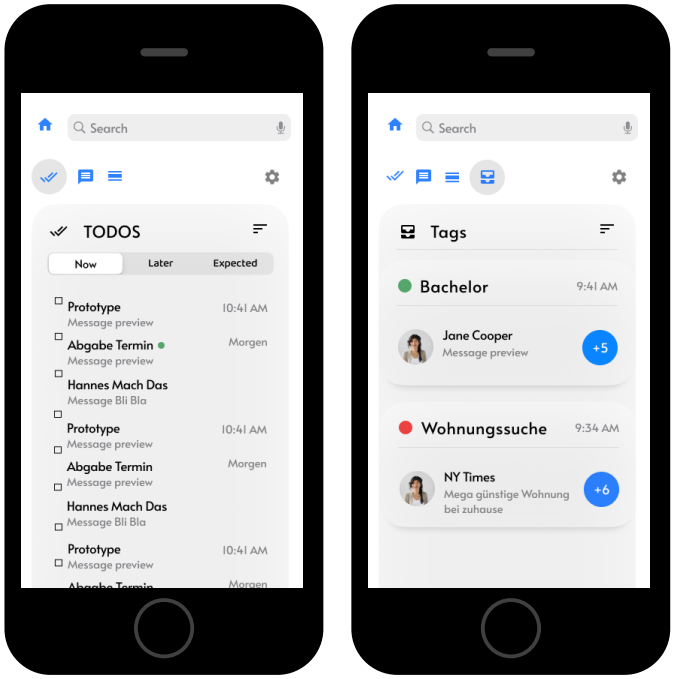
sen.[4]

3.2 Geöffnete Kategorien

Die Previews der Prototypen sind bereits bis auf Scrollbarkeit geringfügig interaktiv. Außerdem sind sie expandierbar, um die volle Ansicht der entsprechenden Kategorie zu öffnen. Die expandierte Ansicht sollte den gleichen Stil wie das Preview anwenden und bestenfalls dorthin animierbar sein, um der Nutzerin den Systemstatus sichtbar zu machen. Am oberen Ende der Ansicht (siehe Abbildung 3.3) befindet sich immer noch die Suchleiste; hier könnte nun aber durch die Nutzerin ausgewählt werden, ob sich die Suche nur auf diese Kategorie oder aber auf die gesamte Inbox beziehen soll. Unmittelbar darunter befindet sich eine Leiste, welche neben der Übersicht über alle Kategorien einerseits ebenfalls den Systemstatus verdeutlicht, indem zu sehen ist, welche Kategorie momentan geöffnet ist, andererseits aber auch die Möglichkeit bietet, schnell zu einer anderen Kategorie zu wechseln.

Die expandierten Kategorien könnten dann beliebig komplex sein; es ergäbe sich allerdings je eine Übersicht über die Kategorie (wie in Abbildung 3.3a) und eine Detailansicht (wie in Abbildung 3.4) zur entsprechenden Email. In den jeweiligen Ansichten könnte auch auf andere Kategorien in Bezug auf diese Email zugegriffen werden. So könnte beispielsweise wie in Abbildung 3.4 eine Hauptansicht mit der Beschreibung, Wichtigkeit und Zuordnung des ToDos zu sehen sein und darunter eine Ansicht mit Nachrichtenpassagen, die diesem ToDo zugehörig angeclippt wurden. Die Clip-Sektion wäre eine weitere denkbare Kategorie, auf die dann auch aus den anderen Kategorien heraus zugegriffen werden könnte. Eine weite-

[4] Das heißt es wird nicht der Hauptinhalt dieser Arbeit sein verschiedene Kategorien zu entwerfen. Stattdessen liegt das Ziel darin zukunftigen Entwicklerinnen das Erstellen von Kategorien zu vereinfachen.



(a) Expandierte ToDo-Übersicht (b) Expandierte Tags-Übersicht

Abbildung 3.3: Expandierte Ansichten

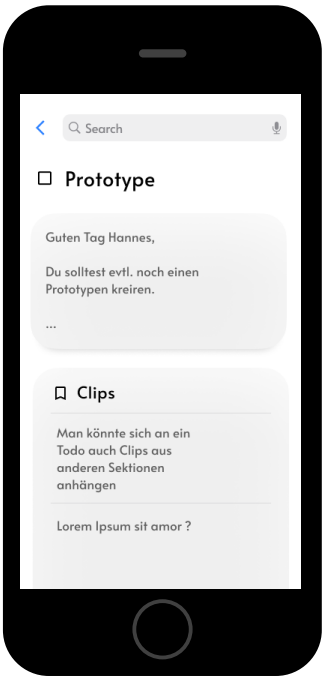


Abbildung 3.4: Expandierte Ansicht eines ToDos

re vorstellbare Kategorie, auf die aus anderen Kategorien zugegriffen werden könnte, wären die Dateien [30, 39, 7].

Weitere denkbare Kategorien, zum Großteil aus bestehenden Arbeiten herausgesucht, sind unter B.1 aufgelistet.

3.2.1 Farbgebung

Bei der Farbgebung wurde darauf geachtet, wenige wichtige Akzente zu setzten, sodass wichtige Nachrichtencounter sowie verschiedene Aktionen hervorgehoben werden. Außerdem wurde die Farbe so gewählt, dass sie sich unter nahezu allen Farbsehschwächen abhebt und selbst unter kompletter Monochromie stumm geschaltete Kategorien noch gut von aktiven unterscheidbar sind. Diese Farbgebung wird eventuell später noch um die Farben Grün für eine sichere Email, Gelb für eine unsichere und Rot für „Gefahren-Aktionen“ hinzukommen. Diese sind im Prototyp noch nicht zu erkennen. Zusätzlich sollten zu den Farben noch die Symbole Postkarte und Brief zur Verdeutlichung von mitlesbar bzw. sicher verwendet werden [12].

3.3 Tags, Ordner und Regeln

Schon 1996 wurde von Whittaker et al. festgestellt, dass verschiedene Nutzerinnen unterschiedlich mit Ordnern umgehen. So gibt es diejenigen, die regelmäßig ihre Ordner pflegen und befüllen, als auch die, die nur eine einzige, überfüllte Inbox besitzen [45]. Diese Diskrepanz der Nutzung von zur Verfügung gestellten Funktionen weitete sich seit ihrer Einführung auf Tags sowie Regeln zur automatischen Einordnung aus. So nutzen einige Menschen Tags, andere das, was Gmail automatisch zur Verfügung stellt, und wieder andere erstellen Regeln, um die Emails automatisch den eigens erstellten Ordnern bzw. Kategorien zuzuordnen [5].

Die Kategorien in meinem Prototyp ersetzten nun die klassischen Ordner, die Emails werden automatisch entsprechenden Typen zugeordnet und dementsprechend unterschiedlich angezeigt. Einige schwer kategorisierbare Emails werden dabei übrig bleiben und klassisch dargestellt. Die Kategorien können parallel und unabhängig von der Ordnerstruktur bestehen. Zudem könnte es aber sinnvoll sein, Ordner und Tags zusammenzufassen und den Kategorien überzuordnen, so dass, wie in Abbildung 3.3b zu sehen, eine eigene Kategorie für Tags möglich ist. Bei Auswahl eines Tags wird wieder die Anfangsansicht (Abb. 3.2) angezeigt, diesmal allerdings gefiltert nach Emails, die diesem Tag/Ordner entsprechen. Somit wird das Problem des Unterschiedes zwischen strukturellem Zusammenhang (welcher Kategorie gehört eine Email an) und inhaltlichem Zusammenhang, d. h. welchem Thema diese Email angehört (z. B. Bachelorarbeit oder Ausflug nach Berlin), gelöst. Denn einem Thema können durchaus Emails aus verschieden Kategorien angehören, einer Kategorie aber auch Emails verschiedener Themenbereiche zugeordnet werden. Diese könnten, wie in Apple-Produkten üblich, durch kleine Farbkreise erkennbar sein. So könnte beispielsweise der Tag „Mom und Dad“ zum Großteil aus Emails der Kategorie Finanzen, aber auch aus Emails der persönlichen Kommunikation bestehen [16].

4 Studie zur Evaluation der Typ-Überladung und des Prototypen

Nach der Analyse bestehender Software (siehe Kap. 2.2), sowie bestehender wissenschaftlicher Literatur (siehe Kap. 2.1), stellen sich insbesondere drei Fragen:

- Inwiefern ist die von Whittaker und Sidner festgestellte und von Grevet et al. verfeinerte Typ-Überladung noch immer vertreten, und welche Kategorien von Email gibt es?
- Wie gut bewältigt mein Prototyp das Problem der Typ-Überladung?
- Wie gefällt potenziellen Nutzerinnen das sonstige Erscheinungsbild des Prototyps?

4.1 Methodik und Aufbau

Um diese Fragestellungen zu beantworten, wurden drei einstündige semistrukturierte Interviews zur weiteren qualitativen Datenerhebung durchgeführt. [1]

Hierbei wurde ein semistrukturiertes Interview gewählt, um möglichst qualitative Erkenntnisse mit verschiedenen Details zu erreichen, ohne Gefahr zu laufen, einige wichtige Fragestellungen zu missen.

Das Interview bestand aus drei ineinander übergehenden Teilen. Der erste Teil hatte zum Ziel, die Studienteilnehmenden näher einzuordnen; er enthielt Fragen über demografische Daten sowie Informatikhintergrund und Email-Verwendung. Auf den Email-Hintergrund wurden dann genauer, mit Fokus auf privaten Konsum, eingegangen, um herauszufinden, wie häufig Email verwendet wird, auch in Betracht auf mobile Nutzung, da das Interface in erster Linie für Smartphones konzipiert worden ist.

Im zweiten Teil wird genau auf den Email-Konsum eingegangen. Die Studienteilnehmenden wurden befragt, welche Typen von Email sie erhalten bzw. lesen. Dabei wurden sieben in den bisherigen Arbeiten und Implementationen häufig auftretende Kategorien direkt erfragt, aber auch die Möglichkeit gegeben, weitere Kategorien oder Anwendungsfälle von Email zu nennen.
Anschließend wurde diskutiert, welcher Umgang dieser Typ-Überladung gerecht wird, d. h. inwiefern ein eigenes UI pro Kategorie bei diesem Problem hilft und welche anderen Werkzeuge innerhalb eines Email-Clients hilfreich sind.

Zuletzt wurde eine relativ offene Diskussion über die von mir generierten Prototypen eingeleitet, wobei die Studienteilnehmenden dabei den interaktiven Figma-Prototyp austesteten und mir qualitative Rückmeldung sowohl zum Funktionsumfang als auch zum Erscheinungsbild geben konnten.

[1] Die Struktur des Interviews kann unter B.2 nachvollzogen werden.

Die drei[2] Interviews wurden dabei auf drei verschiedene Arten durchgeführt, eine in Person Vorort, eine per Telefonat mit zeitgleicher Protokollierung, sowie ein aufgenommenes und im Nachhinein protokolliertes Videotelefonat. Dabei wurden Teilnehmende aus verschiedenen, eher jüngeren Altersgruppen gewählt, wobei auch der Hintergrund zur Informatik variierte. So befanden sich unter den Teilnehmenden zwei Studierende sowie ein wissenschaftlicher Mitarbeiter. Zwei Teilnehmende waren aus dem Bereich der Informatik. Alle Teilnehmenden wohnen in Berlin und nahmen unbezahlt an der Befragung teil.

4.2 Erkenntnisse

Durch die Interviews konnten die beschriebenen Fragestellungen beantwortet werden. So wurde eine leichte Verschiebung der verschiedenen Typen innerhalb der Typ-Überladung festgestellt, als auch Rückmeldung zum Prototyp gegeben. Darüber hinaus rückte in allen drei Gesprächen ein weiteres Thema in den Fokus, welches ich ursprünglich nur kurz behandeln wollte: die (automatische) Zuordnung von Email in ihre Kategorien.

4.3 Typ-Überladung

Es gab in meiner Stichprobe nur eine einzige Kategorie, die von allen Teilnehmenden häufig gelesen wird: Accountmanagement[3], eine Kategorie von Email, in welcher neu erstellte Accounts bestätigt werden, Passwordänderungen beantragt werden oder Ähnliches. Dennoch gibt es auch Kategorien, wie Jobverwaltung oder Reisen, die eine relativ hohe Verwendung finden. Darüber hinaus werden von allen Teilnehmenden die Kategorien Belege/Rechnungen sowie Werbung/Deals empfangen, aber nur gelegentlich gelesen bzw. verwendet. Siehe hierzu auch Tabelle 4.1. Es war also ein breites Spektrum all der befragten Kategorien vertreten, was darauf hinweist, dass das Problem der Typ-Überladung aktuell ist.

Vor allem wird von den Befragten beschrieben, dass ihre Inbox für all diese Kategorien nicht geeignet ist und deshalb nach Möglichkeit oft lieber auf andere, für diesen Typus besser geeignete Tools, wie beispielsweise Discord für private Nachrichten, zurückgegriffen wird. Allerdings wurde erwähnt, dass eine bessere Unterstützung der Typen durch ein angepasstes UI von Vorteil wäre und dann entsprechend eventuell häufiger auf Email zurückgegriffen werden würde.

Ein Teilnehmer, P1, der Email eher selten verwendet, nannte als Hauptfunktionen, für die er Email verwendet, das Senden von Dateien vom Smartphone zum PC und das Verifizieren von Accounts etc. Das wären zwei weitere Typen, für die Email ursprünglich nicht angedacht war - womit sie heutzutage überladen ist. Das Dateimanagement könnte man dabei gut in eine Dateikategorie integrieren, während die Erstellung einer weiteren Kategorie mit Verifizierungen bzw. Accountmanagement für die andere Überladung ein denkbarer Ansatz wäre.

4.4 Kategorisierung

Die Meinung über eigene Nutzerschnittstellen für verschiedene Arten bzw. Typen von Email war insgesamt sehr positiv. So betitelte ein weiterer Teilnehmer, P3, diese Idee mit „absolut sinnvoll“. Allerdings wurde

[2] Siehe Kap. 4.6
[3] Diese Kategorie wurde in der Arbeit von Grevet et al. jedoch gar nicht beschrieben.

Kategorie	Empfangen	Gelesen	Bemerkung
Werbung/ Deals	3/3	1/3	Wird von P2 als „Spam“ und „Müll“ bezeichnet
private Nachrichten	1/3	1/3	Zusätzlich P2: „sehr selten, einige Male im Jahr zum [Networking]“
Aufgaben Management	1/3	1/3	Die jüngeren Teilnehmer kaum
Reisen/ Tickets	2/3	2/3	„Erster Anlaufpunkt“
Jobs	3/3	2/3	Diejenigen die nicht auf Jobsuche sind, sagten aus, dass sie dafür Email verwenden würden
Bestätigungen/ Belege/ Rechnungen	3/3	2/3	Gelesen bei Rückgaben
Accountmanagement/ Verifizierungen	3/3	3/3	Hauptverwendungszweck
Sicherheitswarnungen	1/1	1/1	Diese Kategorie kam leider erst im letzten Interview auf

Tabelle 4.1: Verschiedene Kategorien von Emailtypen und deren Verwendung unter den Studienteilnehmenden. Spalte 2 bzw. 3 beinhalten den Anteil der Teilnehmenden, von denen Emails dieser Kategorie regelmäßig empfangen bzw. gelesen werden. Die Kategorien in der zweiten Sektion (unterer Tabellenbereich) wurden erstmalig von den Studienteilnehmenden genannt.

auch Sorge über erhöhte Komplexität, die vorallem für die höheren Altersgruppen eine Herausforderung sein könnte, (von P2) geäußert.

Besondere Aufmerksamkeit sollte aber auf die Kategorisierung gesetzt werden, die hier, mit Erfahrung von Spam-Kategorisierung, schnell Frust entstehen kann, wenn die Email nicht am erwarteten Ort zu sehen ist. So verdeutlichte P2 das Problem der Fehlklassifizierung im Fall von Spam damit, dass er ein False-Positive mit „10-mal“ so schwerwiegend wie ein False-Negative beschrieben hat. Um dieses Problem zu umgehen, wurde vorgeschlagen, als Fallback die klassische Inbox am oberen Ende der Inbox beizubehalten. Dort könnte dann beispielsweise die Email mit einem Verdacht, in welche Kategorien diese gehören könnte, markiert werden, damit sie leicht zugänglich in einer passenden Kategorie gelesen werden kann. Beispielsweise würden verschiedene Knöpfe an einer Email angezeigt werden, die diese Email in einer bestimmten, für diese Email geeigneten Ansicht öffnen. Damit könnten alle Emails zusätzlich an einem Ort vereint bleiben, diese aber dennoch bei Wunsch in einer geeigneten Kategorie gelesen werden.

4.5 Dediziertes Feedback zum Prototyp

Die Ministudie ergab auch einiges an Rückmeldung zum momentanen Stand des Prototyps.

Zum einen scheint es nicht gewünscht zu sein, dass die ToDo-Kategorie, wie in Abschnitt 3.1.2 konzipiert, am oberen Ende angepinnt bleibt, da das eine nicht direkt ersichtliche Inkonsistenz darstellt (P1, P2). Stattdessen ist eine eigene Sortierung erwünscht, bei der einige Kategorien fest angepinnt werden, während die anderen sich nach zuletzt eingetroffener Email sortieren (P1). Darüber hinaus wurde von P1 der Vorschlag genannt einen Toggle einzubauen, der die Pins direkt aktiviert bzw. deaktiviert, um mit einem Klick die neuen Emails nach oben zu bekommen.

Zum anderen wurde von P1 und P2 vorgeschlagen, die Leiste, welche bei expandierten Kategorien unmit-

telbar unter der Suchleiste zu sehen ist (vergleiche Abb. 3.3), auch auf der Startansicht anzuzeigen. Damit sollte erreicht werden, dass auch ohne Scrollen mit einem Klick schnell die gewünschte Kategorie geöffnet werden kann, falls man z. B. die News lesen möchte. Dafür sollten aber nach P1 die Icons etwas detaillierter, ähnlich zu denen von Apps, gestaltet sein. Die verschiedenen Kategorien seien dann etwa wie verschiedene Apps auf Basis von Email vorstellbar[4].

Abgesehen davon, wurde über die Größe und Komplexität die Aussage getroffen, dass verschiedene feste Größen obsolet seien, wenn man einen Griff[5] implementieren würde, mit der sich die Kategorien dynamisch vergrößern ließen (P1, P2).

Die Rückmeldung in Hinblick auf das im Prototyp entwickelte Erscheinungsbild war insgesamt positiv und es gab am Erscheinungsbild durch die drei Teilnehmenden nichts auszusetzen.

4.6 Limitationen

Es bleibt anzumerken, dass diese Studie nur im kleinen Rahmen mit drei Teilnehmenden stattfand. Es gab zwar zahlreiche qualitative Anmerkungen, die wissenschaftliche Aussagekraft, vorallem im quantitativen, statistischen Bereich wie dem momentanen Vorhandensein der Typ-Überladung, ist begrenzt und wäre bei weiterer Forschung erneut in größerem Umfang durchzuführen.

4.7 Fazit

Ein Bedarf an an bestimmte Kategorien speziell angepasste UIs ist durchaus vorhanden. Dabei sollte aber besonderer Fokus auf eine zufriedenstellende automatische Kategorisierung gesetzt werden.

Da in dieser Arbeit allerdings nur das konzeptionelle Framework zum Erstellen dieser Kategorien implementiert wird, wird das korrekte Zuordnen nicht fokussiert. Für die folgende Implementation bleiben aber im Vergleich zum Prototyp einige Änderungen möglich. So wäre die Kategorie-Übersichtsleiste auch in die Startansicht zu bringen sowie eine Fallback-Standard-Kategorie einzubauen. Diese kann dann optional jegliche Email beinhalten, um das alte Interface und seine Vorteile beizubehalten. Siehe hierzu auch Absatz 7.1.4. Des Weiteren wäre eine eigene Sortierung der Kategorien noch umzusetzen sowie eine dynamische Größe der Previews.

[4] siehe hierzu auch 7.1.6

[5] Damit ist eine virtuelle Leiste gemeint, mit der sich durch Zieh-Gesten hier eine Größe verändern lässt.

5 Implementation

Die Konzepte aus Kapitel 3 wurden im Rahmen dieser Arbeit größtenteils in die bereits bestehende LetterBox-App integriert[1].

Das bedeutet, die Ansichten (Views), ähnlich wie sie in Abbildung 3.2c bzw. 3.2b zu sehen sind, wurden mithilfe von SwiftUI (vergleiche Abschnitt 1.2.2) umgesetzt. Dabei stand nicht im Vordergrund, dass die einzelnen Kategorien, wie z. B. ToDos, bereits voll funktionsfähig sind, sondern dass ein Framework erstellt wird, mit dem es zukünftigen Entwicklerinnen von LetterBox erleichtert wird, neue Kategorien zu erstellen und einzubinden.

Das Framework soll dabei sowohl die Funktionalität, die im Hintergrund geschehen muss, um beispielsweise den einzelnen Kategorien ihre Emails zuzuordnen, als auch einen visuellen Rahmen bieten. Der visuelle Rahmen besteht dabei aus dem umgebenden UI sowie einem Grunddesign für die eigenen Kategorien, so dass sich eine Entwicklerin nur noch um die Elemente einer neuen Kategorie kümmern muss, die diese Kategorie von anderen unterscheidet.

Da sich die Implementation relativ nah an den Prototyp halten soll, lassen sich diesem visuellen Rahmen bereits einige Aufgaben zuweisen. Zum einen soll das Navigieren zwischen den einzelnen Kategorien ermöglicht werden, das heißt das Wechseln, Öffnen und Schließen verschiedener Kategorien. Sind alle Kategorien geschlossen, soll eine Übersicht über die existierenden Kategorien gegeben werden – sowohl darüber, welche Kategorien existieren, als auch darüber, welche Aktivität in einer Kategorie gerade vorherrscht und eine Übersicht über den Inhalt der jeweiligen Kategorien bzw. erschienener Emails.

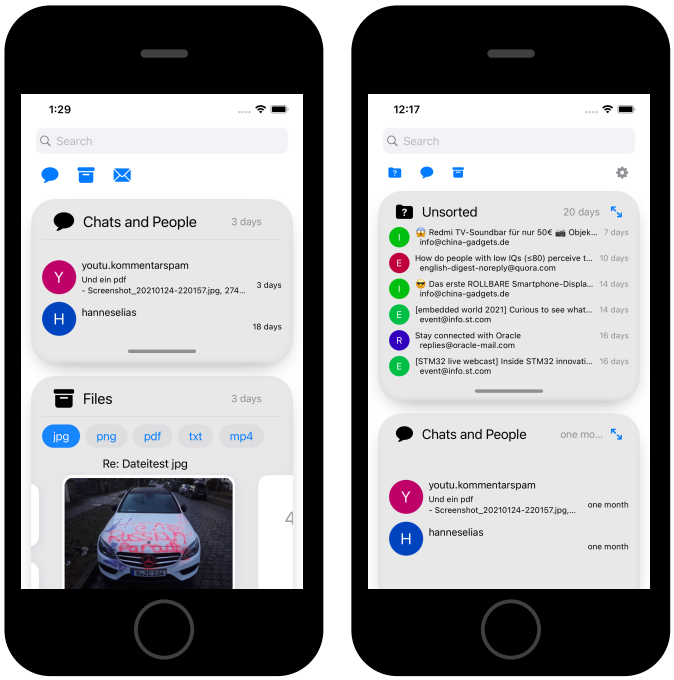
Ist keine Kategorie geöffnet und dementsprechend die eben beschriebene Übersicht zu sehen, wird diese Ansicht im Folgenden die *Startansicht* genannt.

Im Vergleich zum Prototyp (vergleiche Abbildung 3.2) werden an der Startansicht entsprechend den Ergebnissen des Interviews zwei Änderungen vorgenommen: Einerseits werden die Icons der Kategorien auch hier unter der Suchleiste zu sehen sein, andererseits sind die Vorschauen (Previews) der einzelnen Kategorien dynamisch vergrößerbar, weshalb am unteren Ende einer Kategorie ein Griff zum Verändern der Große eines Previews zu sehen ist.

Die geplante finale Startansicht ist in Abbildung 5.1 zu sehen. Wie dort zu erkennen ist, gibt es neben der Möglichkeit der manuellen Auswahl einer Kategorie oder Email eine Suchleiste, über die ein schnelles Finden älterer oder nicht direkt ersichtlicher Emails ermöglicht wird.

Mit folgenden Problemen muss sich innerhalb der Implementation beschäftigt werden:

[1] Siehe hierzu auch in Anhang A



(a) Die neue Übersicht mit Dateivorschau

(b) Die neue Übersicht mit Unsortiert-Kategorie

Abbildung 5.1: Das neue Interface

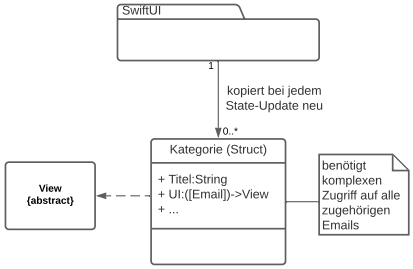
- Wie wird das entwicklerfreundliche Erstellen von Kategorien mit einheitlichem, nutzerfreundlichem Erscheinungsbild ermöglicht?
- Wie werden die Emails einer Kategorie zugeordnet? Dabei sollte sowohl auf Performanz als auch Entwicklerfreundlichkeit gesetzt werden.
- Wie wird die Suchfunktion gewährleistet?
- Wie wird die Navigation innerhalb einer Kategorie, aber auch zwischen verschiedenen Kategorien und dem damit einhergehenden State-Management[2] umgesetzt?
- Wie kann die in den Interviews vorgeschlagene dynamische Größenveränderung umgesetzt werden[3].

5.1 Erster Entwurf

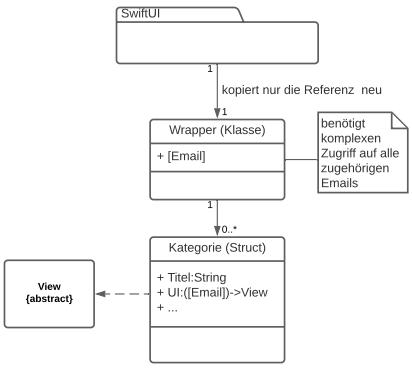
Um möglichst hohe Konsistenz zu erreichen, sollen die Previews bzw. Ansichten der Kategorien denselben Designrichtlinien folgen. Dazu gehört unter anderem, dass die Kategorien von dem gleichen Stil umgeben werden.

Standardmäßig würde man unter SwiftUI (vgl. Abschnitt 1.2.2) dafür einen Struct deklarieren, welcher dann Parameter wie Titel sowie das kategoriespezifische UI übergeben bekommt. Die Übersicht würde dann bei jeder Zustandsänderung (State-Update) eine neue Instanz dieses Structs erstellen und SwiftUI damit die Preview-Übersicht bauen lassen.

[2] z.Dt. Zustandsverwaltung
[3] hier treten einige Schwierigkeiten auf, die in Abschnitt 5.7.1 behandelt werden



(a) Bei jedem State-Update werden die komplexen Kategorien erneut erstellt



(b) Erste Behebung dieses Problems

Abbildung 5.2: nicht vollständig korrekte UML-Diagramme zur Visualisierung

Die Kategorie ist dabei jeweils sehr komplex, da sie vollen Zugriff auf die Emails benötigt. Für jedes Frame- bzw. State-Update eine neue Instanz zu erstellen, wäre ressourcenineffizient (vergleiche Diagramm 5.2a). Da Structs Pass-By-Value sind, würden bei jeder erneuten Übergabe die Dateien innerhalb des Structs kopiert werden.
Um das zu verhindern, würde man weiter oben in der View-Hierarchie ein Klassenobjekt erstellen, welches Pass-By-Reference ist (vergleiche Diagramm 5.2b), und damit den View-Baum heruntergereicht werden kann, ohne bei jedem Update vielfach neu erstellt bzw. kopiert werden zu müssen.

Dieses Objekt könnte dann das speicherintensive Verwalten der Emails und des State-Managements bewerkstelligen.[4] Das birgt aber ein Redundanzproblem, welches im folgenden Absatz genauer behandelt wird.

5.2 Kategorie-Klasse

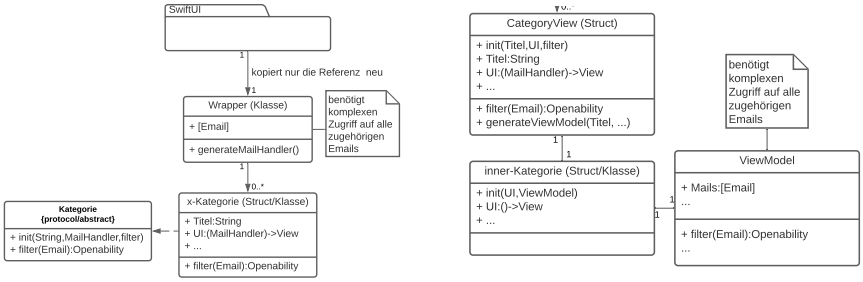
Neben dem UI definiert sich eine Kategorie auch noch darüber, welche Art von Email sie anzeigen bzw. öffnen kann. Bei der Deklaration benötigt ein Kategorie-Objekt also einen Filter, der definiert, welche Emails dieser Kategorie angehören, sowie eine Funktion, die diese Emails übergeben bekommt, und damit das UI dieser Kategorie generiert.

Ein Struct für jede Kategorie zu definieren wäre möglich. Diese Structs würden dann alle ein `Protokoll` implementieren, sodass ihnen ein ‚MailHandler' übergeben werden kann, der dann die bereits beschriebene harte Arbeit der Emailverwaltung für diese Kategorie übernimmt. Dieses folgt bei jeder Kategorie demselben Schema und sollte nicht erneut implementiert werden müssen. Da Structs allerdings aufgrund ihrer Natur keine Vererbung unterstützen, wäre eine Möglichkeit, dass die Kategorien jeweils ihre eigene `Klasse` definieren, die wiederum von einer allgemeinen, abstrakten Kategorien-Klasse erben. In ihrer Initialisierung würden sie dann der `Superklasse` die Informationen zur Verfügung stellen, welche benötigt werden, um dann die Datenverwaltung durchzuführen, so dass diese Verwaltung nicht für jede Kategorie neu implementiert werden muss.

Diese beiden Ansätze sind grob in Grafik 5.3a dargestellt,

Dennoch besteht ein geringer Overhead, da für jede Klasse diese Init-Funktion geschrieben werden muss.

[4] in dem MVVM-Pattern wäre dieses Klassenobjekt das ViewModel für die gesammte kategorisierte Inbox



(a) jede x-Kategorie müsste hier mitsammt aller Funktionen neu eingebaut werden

(b) einzelner Struct, der sich intern um weitere initialisierung kümmert

Abbildung 5.3: nicht vollständig korrekte UML-Diagramme zur Visualisierung

Außerdem sollten für SwiftUI-Views Structs verwendet werden, da diese deutlich kompakter sind, und das Updaten durch State-Changes vereinfachen.

Deshalb wurde ein `CategoryView`-Struct erstellt, welcher die Parameter einer Kategorie, d. h. Name, Filter und ein `Closure`[5] zum Bauen des UIs bekommt. Dieser Struct kümmert sich dann intern, transparent um jegliche benötigte Initialisierung. So muss der Entwickler nichts weiter tun, als eine neue Instanz des CategoryViews mit Namen, Filter und innerem UI zu erstellen (vergleiche Diagramm 5.3b).

Da dieser Struct aber aufgrund seiner Pass-by-Value-Natur ständig neu erstellt wird, wäre es ressourceneffizient, jedes Mal die zum Darstellen der Kategorie benötigten Emails neu fetchen zu müssen.

5.2.1 ViewModel Provider

Dementsprechend ruft der `CategoryView`-Struct einen Provider-`Singleton` auf, welcher nun überprüft, ob ein ViewModel, welches sich um das State- und Emailmanagement einer Kategorie kümmert, für diese Kategorie bereits besteht, ggf. ein neues erstellt und dieses zurückgibt. So wird einerseits vermieden, dass das ViewModel bei kleinen Änderungen im UI neu erstellt werden muss, andererseits vereint es aber auch alle ViewModels der Kategorien an einem Ort.

5.3 Kategorisierung

Wie bereits erwähnt, definiert sich eine Kategorie mitunter darüber, welche Art von Email sie anzeigen bzw. öffnen kann. Das Objekt (ViewModel), das die Kategorien verwaltet, muss also wissen, welche Arten von Email einer Kategorie zugeordnet werden – nicht nur, um sie ihr dann zur Verfügung zu stellen, sondern auch, um die Reihenfolge der Kategorien untereinander zu bestimmen. Diese wird nämlich, wenn nicht durch die Nutzerin anders eingestellt, dadurch bestimmt, welche Kategorie zuletzt eine Email erhalten hat, d. h. welche Kategorie die primäre Kategorie für diese Email darstellt.

Eine Möglichkeit zur Zuordnung von Email zu einer Kategorie wäre eine binäre Entscheidung, ob eine Kategorie eine Email öffnen möchte oder nicht. Das wäre dann leicht über eine Funktion mit einem Wahrheitswert (Boolean) als Rückgabewert umsetzbar [16].

Angenommen, eine Email mit Reisedetails und einem Ticket in Form eines PDF-Anhangs erreicht

[5] eine anonyme Funktion

die Nutzerin [6] . Nun würden sowohl die Reisekategorie als auch die Dateienkategorie diese Email als positiv bzw. als „Ich möchte diese Email öffnen“ klassifizieren.

Jetzt stellt sich die Frage, welcher Kategorie diese Email zugeordnet wird und damit nach unserer Sortierung ans obere Ende der Inbox gelangt. Beide Kategorien gleichhoch zu priorisieren und zu platzieren ist nicht möglich, denn eine Nebeneinanderplatzierung zweier Kategorie-Previews lässt sich aufgrund des Platzmangels auf mobilen Geräten ausschließen. Außerdem ist es gar nicht sicher, ob es bei zwei Kategorien bleibt, da tendenziell beliebig viele Kategorien erstellt werden könnten und diese Email als positiv einstufen. Eine Möglichkeit dieses Problem zu bekämpfen wäre, dass die Kategorisierungsfunktionen als Rückgabewert keinen Boolean, sondern eine Zahl (z. B. ein Integer) haben könnte, welche als Priorität interpretiert werden könnte. Diese Zahl könnte dann bestimmen, welche Kategorie weiter oben angezeigt wird. Darüber hinaus würde eine 0 bedeuten, dass diese Email von dieser Kategorie nicht geöffnet werden möchte bzw. kann. Dieser Ansatz ist jedoch für die Entwicklerin , die die Kategorien erstellt, relativ unüberschaubar, schließt gleiche Zahlen auch nicht aus (obwohl die Wahrscheinlichkeit vernachlässigbar wird), und vorallem wäre es wahrscheinlich unerwünschtes Verhalten, wenn beispielsweise die Spamkategorie und die Dateikategorie ganz oben, als wichtigste aktuelle Kategorien zu sehen sind, nur weil eine eine Email als Spam und ‚enthält eine Datei‘ klassifiziert wurde.

Ein Kompromiss aus beiden Ansätzen besteht darin, als Rückgabewert des Filters ein `Enum`[7] mit den Werten *cannotOpen* , *canOpen* und *shouldOpen* zu definieren. Das UI einer Kategorie bekommt dann alle Emails mit *canOpen* überreicht, wird allerdings nur an das obere Ende der Übersicht gesetzt, wenn eine Email als *shouldOpen* klassifiziert wird. Nun bleiben diejenigen Kategorien, die eine Email zwar öffnen können, aber nicht die primäre Kategorie sind, unten, und nur die relevanten Kategorien kommen nach oben. Trotzdem kann es noch vorkommen, dass mehrere Kategorien eine Email als *shouldOpen* klassifizieren und damit oben einordnen wollen. Dieser Fall sollte nun aber relativ selten eintreten und die ‚obere‘ Kategorie wird dann vereinfachend darüber bestimmt, welche Kategorie in der sortierten Liste, in der sie definiert wurden, früher genannt wird.

In Anbetracht des Sicherheitsaspektes von Letterbox könnte eine Sicherheitswarnung-Kategorie, die z. B. Phishing erkennt, eine interessante Erweiterung ein. Diese müsste dann das *canOpen* der anderen Kategorien überschreiben, sodass nur noch die Phishing- bzw. Sicherheitswarnung-Kategorie diese Phishing-Email öffnet und sie in den anderen Kategorien nicht mehr angezeigt wird. Dafür wurde der Enum um ein *mustOpen* erweitert. Einer Kategorie wird vom Emailmanagement-Objekt also nur diejenige Email überreicht, die sie selber mindestens als *canOpen* einstuft und keine andere als *mustOpen*.

5.3.1 Datenbank

Dieser Zuordnungs-Filter kann komplex sein, z. B. wenn eine Kategorie später ML verwendet, um eine Email zu klassifizieren. Dementsprechend ist es äußerst ungeeignet, einen Filter nur über eine Datenbankabfrage zu definieren. Die Datenbankabfrage bringt allerdings den Vorteil, dass sie im Vergleich zum Filtern für jede einzelne Email zur Laufzeit deutlich performanter ist. Zusätzlich existiert das Problem, dass, wie bereits beschrieben, eine Kategorie eine Email nicht unbedingt öffnen kann, nur weil ihr eigener Filter sie als *canOpen* klassifiziert. Das bedeutet, dass entweder die Emails zentral verwaltet werden müssen und ein

[6] Aus Gründen der Lesbarkeit wird in dieser Arbeit das generische Femininum verwendet, selbstverständlich sind damit alle Geschlechter gemeint.

[7] z.Dt. Aufzählungstyp. Also eine Sammlung verschiedener Aufzählbarer Werte

MailHandler auf die zu sortierende Email jeweils jeden Filter anwendet [8], um die Email einzuordnen und im Falle eines *mustOpen* die Öffnungsmöglichkeiten der anderen Kategorien überschreibt. Anschließend würde dieser MailHandler damit einzelne Handler für die jeweiligen Kategorien erstellen und diesen übereichen. Dies müsste ebenfalls bei jedem Neustart der App ausgeführt werden und wäre damit ineffizient.

Stattdessen wurde eine neue Kategorie-Tabelle in der Datenbank angelegt, die abspeichert, welche Email von welcher Kategorie wie stark [9] geöffnet werden kann. Um nun herauszufinden, welche Emails eine Kategorie öffnen kann, muss nur eine Datenbankabfrage gestellt werden: Welche Email hat mindestens ein *canOpen* in der Kategorie-Tabelle für die entsprechende Kategorie und kein *mustOpen* einer anderen Kategorie.

Allerdings muss diese Tabelle vollständig sein, sonst können Emails fehlen oder fehlerhaft dargestellt werden, obwohl eine andere Kategorie sie als *mustOpen* klassifiziert hätte. Deshalb wird bei Start der App der Laufzeitfilter einmal für alle Kategorien auf die nicht zugeordneten Emails angewendet und diese zugeordnet und in der Tabelle hinterlegt. Das geht jedoch deutlich schneller, als jedes Mal die kompletten Emails so zuzuordnen.

5.4 Statemanagement

Das State-Management der App wird mithilfe des MVVM-Pattern gehandhabt.

Innerhalb der neuen, kategorisierten Inbox entsteht dennoch, zumindest mit der entwicklerfreundlichen Implementation, wie sie im Rahmen dieser Arbeit durchgeführt wurde, ein Problem. Denn für die Ausgangsansicht (siehe Abbildung 3.2) ist es wichtig, dass das Viewmodel ein Neu-Rendern des Views auslöst, wenn eine Kategorie sich auf aktiv[10] setzt. Denn ist eine Kategorie aktiv, wird diese im Vollbild statt im Preview angezeigt, während alle anderen Kategorien nicht zu sehen sind.

Allerdings sind viele weitere Zustandsänderungen der einzelnen Kategorien für die Ausgangsansicht nicht von Relevanz. Dementsprechend ist es kein guter Ansatz, ausschließlich die Wrapper `@Published` und `@ObservedObject`, wie sie in Abschnitt 1.2.2 beschrieben sind, zu verwenden. Denn üblicherweise wird das Viewmodel in dem dazugehörigen View durch ein `@ObservedObject` markiert und überträgt damit alle publizierten Änderungen an das View, dadurch würde also hier die Ausgangsansicht bei jeder kleinsten Zustandsänderung innerhalb einer Kategorie neu gerendert/berechnet werden.

Nur die `IsActive`-Werte der Kategorien zu publizieren, ist allerdings schwer, da das Subscriben einzelner Eigenschaften mit dem Standardansatz nicht vorgesehen ist. Dass das Viewmodel einer einzelnen Kategorie schon nur die `IsActive`-Werte publiziert, ist ebenfalls kein guter Ansatz, da dadurch die Views der Kategorien ebenfalls nicht mehr neu gerendert werden, obwohl sich eine render-relevante Eigenschaft ihres Viewmodels ändert.

Ein möglicher Lösungsansatz für dieses Problem wäre, für jede Kategorie zwei Viewmodels zu besitzen: eines, welches nur die für die Kategorie selbst relevanten Eigenschaften publiziert, und eines, das die Eigenschaften publiziert, welche sowohl für die Kategorie als auch die Ausgangsansicht relevant sind. Dieser Ansatz ist leider sehr unsauber, da nun auch noch auf die Logiksynchronisation zwischen den beiden Modellen geachtet werden muss.

[8] Laufzeit in $\mathcal{O}(n^2)$, mit n: Anzahl der Emails
[9] Damit sind die Werte *cannotOpen* , *canOpen* und *shouldOpen* sowie *mustOpen* gemeint
[10] D. h. die Kategorie setzt ihre boolesche Eigenschaft `isActive` auf wahr

Stattdessen basiert der verwendete Lösungsansatz auf der Implementation neuer Propertywrapper mithilfe des Combine-Frameworks (siehe dazu Abschnitt 1.2.2). Der neue `@PPublished` Wrapper erweitert den standardmäßigen `@Published` Wrapper um eine für diesen Anwendungsfall kritische Eigenschaft: statt, dass nur publiziert wird, dass sich das Objekt geändert hat, kann beim Deklarieren einer `@PPublished` Variable eine Wichtigkeit bzw. ein Grund für diese Variable mit angegeben werden. Der Subcriber sieht dann bei einer Änderung nicht nur, dass die Änderung geschehen ist, sondern auch aus welchem Grund bzw. mit welcher Wichtigkeit. Somit ist es dem Viewmodel der Ausgangsansicht möglich, die Ausgangsansicht nur neu berechnen bzw. rendern zu lassen, wenn eines der Kategorie-Viewmodels publiziert, dass es sich mit dem Grund `"For HomeModel"` ändert.

5.5 Navigation

Apple stellt in dem SwiftUI-Framework zwei Structs, `NavigationView` und `NavigationLink` zur Verfügung, mit denen die Navigation zwischen verschiedenen Ansichten umgesetzt werden kann. Das NavigationView wird dabei beliebig (oft als Root) in dem Viewbaum eingebaut. Dort funktioniert es dann als eine Art Anker, der beim Aktivieren eines NavigationLinks durch den in diesem Link spezifizierten Inhalt ersetzt wird.

Da sowohl in der Ausgangsansicht mit den Previews als auch in der geöffneten Ansicht einer Kategorie einige Elemente, wie z. B. die Suchleiste, beibehalten werden, bietet es sich leider nicht an, den standardmäßigen SwiftUI-Navigationsstapel als Root zu verwenden, da dieser dann jeweils die ganze Ansicht durch eine neue ersetzt und damit die Suchleiste etc. bei jeder Navigation verschwinden würden.

Das gilt so nur, wenn der Viewanker für die Navigation als Vollbild bzw. Root gesetzt wurde; es ist auch möglich, ihn erst unterhalb der Suchleiste etc. zu setzten, um diese dann auch bei Verwendung der nativen Navigation beizubehalten. Jedoch ist es dann nicht mehr möglich, innerhalb einer Kategorie eine Vollbildansicht zu öffnen, in der keine Suchleiste zu sehen ist. Das liegt daran, dass beim Pushen einer neuen Ansicht auf den Navigationsstapel diese dort angezeigt wird, wo der in der View-Hierarchie nächstgelegene Anker gesetzt ist. Das bedeutet, dass es nicht möglich ist, einen zweiten Anker als Vollbild zu setzten und dann dynamisch zu entscheiden, auf welchen Stack bzw. zu welchem Anker eine neue Ansicht gepusht werden soll.

Diese Funktion wird aber benötigt, damit sowohl Navigation innerhalb einer Kategorie als auch über die Kategorie hinaus stattfinden kann. Zur Illustration ein Beispiel:

> Die Nutzerin bekommt innerhalb der Chatkategorie eine Übersicht über die Kontakte, mit denen sie zuletzt geschrieben hat, und deren letzte Nachrichten. Es wird ein Chat ausgewählt. Dieser wird dann innerhalb der Chatkategorie angezeigt, das heißt Suchleiste sowie die restliche Kategorieansicht bleiben erhalten. Nun wird allerdings von der Nutzerin ausgewählt, dass sie zu diesem Kontakt einen neuen Schlüssel importieren möchte. Die Schlüsselimport-Ansicht ist von der Kategorie unabhängig und wird daher in einer neuen Vollbildansicht geöffnet.

Um dieses etwas komplexere Navigationsmanagement zu bewältigen, wurde das `TagNavigationView` implementiert. Dieses ermöglicht beim Setzen eines neuen Ankers einen Tag festzulegen. Dieser Tag kann dann beim Pushen eines neuen Views verwendet werden, um festzulegen, auf welchen Anker die neue Ansicht gesetzt wird. [11]

[11] Dadurch entsteht eine Art Baumstruktur, bei der jeder neue Anker als Knoten mit 2 Kindern betrachtet werden kann. Dabei

Nach dem momentanen Stand ist es allerdings nur möglich, Views auf einen Tag zu pushen, der innerhalb der Vorfahrenlinie des momentan geöffneten Views liegt. Dadurch ist die Struktur eher als eine Liste bzw. ein Stack mit verschiedenen Eintrittspunkten zu verstehen. Dennoch wird das Ziel erreicht, mehrere Navigationviews erfolgreich inneinander zu verschachteln, ohne dabei die Möglichkeit zu verlieren, jedes einzelne davon gewählt zu verwenden.

Der beschriebene Beispielanwendungsfall ließe sich also lösen, indem jede Kategorieansicht einen eigenen Tag bekommt, und außerdem ein Vollbildnavigationview-Anker gesetzt wird. Das Öffnen eines Chats pusht nun also auf den Tag der Kategorie und das Öffnen der Schlüsselimport-Ansicht auf den Vollbildtag.

Umgesetzt ist es momentan so, dass der Vollbildstack das klassische `NavigationView` ohne Tag verwendet, und einer Kategorie bei ihrer Initialisierung automatisch ein Tag zugewiesen wird, welches dann einfach durch Verwenden eines `TagNavigationLinks` statt des klassischen `NavigationLinks` innerhalb der Kategorie angesprochen wird.

5.6 Suche

Neben eigenem Design und Email-Filter kann eine Kategorie zusätzlich noch eigene Suchfilter bestimmen. In der Startansicht ist eine klassische Suchleiste mit den Filtern *Betreff*, *Sender*, *Body* und *Alle* vorzufinden. Wird eine Kategorie geöffnet, so bleibt die Suchleiste am oberen Ende bestehen, die wählbaren Filter können nun aber durch die Kategorie definiert sein. So gibt es in der beispielhaften Dateienkategorie zusätzlich einen Filter für Dateinamen. Wird nun beispielsweise der Begriff „Thesis“ eingegeben, so werden ihr nach Auswahl des Filters *Dateiname* nur diejenigen Emails angezeigt, die eine Datei beinhalten, dessen Name „Thesis“ beinhaltet.

Dazu wurde ein `Search` Struct definiert, der alle für die Suche nötigen Informationen wie Suchtext, aktiver Filter, verfügbare Filter etc. beinhaltet. Die Suche selbst wird von dem `HomeModel` gehandhabt, das die Suchfilter der aktiven Kategorie bereitstellt und bei Updates der Suche diese an die einzelnen Kategorien weiterpropagiert.

5.7 UI

Mithilfe der beschriebenen Implemenationen konnte dann das UI mit Funktion versetzt werden. Das Design dieses UIs wurde möglichst nah am Prototyp gehalten, wobei einige der im Interview vorgeschlagenen Verbesserungen ebenfalls vorgenommen wurden. So existiert nun eine änderbare Sortierung und die Kategorieleiste bleibt permanent.

Wie mit SwiftUI Standard, wurden die einzelnen Designelemente dabei durch verschiedene `Structs`, die dem `View` Protokoll konformieren, realisiert. Die einzelnen Bauteile wurden dabei möglichst elementar gehalten, um eine hohe Wiederverwendung der Bauteile zu ermöglichen.

5.7.1 Größe der Previews

Eines dieser Bauteile ist die `DragResizableCard`. Wie der Name bereits suggeriert, ist es ein Card Element, also ein Element, was meist für eine moderne Snapshot-artige Vorschau verwendet wird [23].

bezeichne nun der linke Branch den jeweiligen Hauptbranch und der rechte den abgezweigten. Wird nun ein neues View auf Tag x gepusht, dann wird dieses View am linkesten Kind des rechten Branches angehangen.

Die Besonderheit hierbei ist, dass am unteren Ende ein virtueller Griff, vom Design ähnlich zum „Home Indicator“ am unteren Ende des iPhones, angebracht ist, mit dem sich die Karte vergrößern bzw. verkleinern lässt. Damit lässt sich kurzfristig mehr Inhalt innerhalb des Karte anzeigen. Das wird in dem neuen Interface für die Previews der einzelnen Kategorien verwendet. Diese Idee wurde in den Interviews vorgeschlagen und hat den Vorteil, dass dadurch schnellerer Überblick über eine Kategorie gewonnen werden kann, ohne diese tatsächlich zu öffnen.

Der Struct `DragResizableCard` ist dabei recht simpel: Er bekommt ein View übergeben und gibt ein View zurück. Das zurückgegebene View wiederum besteht aus einem `CardView`, welches mit dem übergebenem View und dem virtuellen Griff populiert wird. Ein weiterer Parameter, der der `DragResizableCard` übergeben werden kann, ist die Größe, eine mit dem Propertywrapper `@Binding` umgebene Variable, welche die Größe der Karte beinhaltet und diese mit dem ParentView synchron hält.

Allerdings gab es mit dem Verwalten der Größe in der tatsächlichen Implementation der neuen Inbox einige Probleme.
Eines davon war, dass die Gestenerkennung des Frameworks keine Drag-Gesten, also diese, die zum vergrößernden Ziehen eines Kategorie-Previews innerhalb der DragResizableCard nötig wären, erkennt, wenn diese innerhalb eines `ScrollViews` liegen. Warum dieses Verhalten so stattfindet ist nicht ganz klar, da normalerweise Kinder eines Views für die Gestenverwendung priorisiert werden. Nach Ausprobieren verschiedener Ansätze aus `Stackoverflow.com`, vorallem in Bezug auf die Priorisierung, habe ich sehr unzuverlässiges Verhalten erreicht. Behoben werden konnte das Problem durch ein Vergrößern der Touchsensitiven Fläche um den Griff herum.
Ein zweites Problem hing damit zusammen, dass die momentane Größe des Previews als Eigenschaft einer Kategorie weitergereicht wurde, damit diese abhängig von der Größe unterschiedliche Darstellungsformen annehmen kann, um sich möglichst responsiv zu verhalten. Wird nun durch die Nutzerin die Größe des Previews verändert, so ändert sich eine Eigenschaft dieser Kategorie, und das entsprechende ViewModel publiziert, dass es sich geändert hat. Diese Änderung wurde von dem ViewModel der Startansicht, dem `HomeModel` beobachtet, welches daraufhin die Standardansicht neu rendern lässt. Das erneute Berechnen der Standardansicht bewirkt, dass die Kategorie-Interfaces neu berechnet werden und damit die Größe der Previews zurückgesetzt wird. Ein Kreisschluss. Dadurch verändert sich die Previewgröße nicht in die gewünschte Richtung, sondern flackert nur. Dieses Problem wurde mithilfe der in Abschnitt 5.4 beschriebenen Auftrennung in verschiedene `@Published`-Wrapper behoben.

6 Evaluation

Das neue Interface wurde abschließend evaluiert. Aufgrund der aktuellen Kontaktbeschränkungen wurden Evaluationstechniken ohne Nutzerteilnahme eingesetzt.

6.1 Nielson Heuristika

Eine der Evaluationstechniken ohne Nutzerteilnahme ist die sogenannte Nielson Heuristik [25]. Mithilfe der zehn von Jakob Nielson gestellten Heuristiken ist es mir möglich, das Interface statisch zu inspizieren. Neben diesen Heuristiken gibt es weitere Prinzipien, die sich per Inspektion untersuchen lassen, zum Beispiel von Bruce Tognazzini, einem der Leiter der Niels-Norman-Group [41] (siehe [42]). Aufgrund der Kompaktheit und Anerkennung wurde sich hier für die Nielson Heuristik entschieden. Die hauseigenen Richtlinien von Apple[1] sind diesen sehr ähnlich.

6.1.1 Sichtbarkeit des Systemstatus

Der Systemstatus ändert sich mit dem neuen Interface vor allem in dem Aspekt, ob und welche Kategorie gerade geöffnet ist. Dies wird dadurch signalisiert, dass die gewählte Kategorie nun (leider nach momentanen Implementierungstand noch nicht stetig animiert) expandiert wird, bis der Bildschirm gefüllt ist, und der Griff zum Vergrößern bzw. Verkleinern der Ansicht verschwindet. Da auch durch manuelles Vergrößern der Kategorie prinzipiell dieselbe Ansicht erreicht werden könnte, ohne diese Kategorie zu aktivieren, wird darüber hinaus das Icon der aktivierten Kategorie unter der Suchleiste durch einen grauen Hintergrund hervorgehoben.

Ein paar standardmäßige Elemente, wie das Hervorheben der Suchleiste beim Suchen oder Sättigungsanpassung beim Drücken eines Knopfes, sind hier ebenfalls vorhanden.

6.1.2 Vergleichbarkeit zwischen System und echter Welt

Momentan gibt es erst drei Kategorien. Diese sind unsortierte Email, ähnlich zur klassischen Inbox, sowie die beiden Kategorien Dateien und Chats. Jede Kategorie hat dabei ihr eigenes Icon. Die Chat-Kategorie wird durch eine Sprechblase verdeutlicht, die für unverbindliches Kommunizieren steht, die Datei-Kategorie wird durch eine Aktenbox dargestellt und die unsortierten Emails durch einen Ordner mit Fragezeichen drauf.

[1] siehe https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

6.1.3 Nutzerkontrolle und -freiheit

Die Nutzerfreiheit stand besonders im Fokus: So ist es jederzeit möglich, eine Kategorie zu maximieren oder wieder zum Preview zu verkleinern – sowohl über die Leiste oben als auch über die Mini- bzw. Maximierungsknöpfe in der Ecke der Kategoriekarten. Auch mit geöffneter Kategorie ist nicht nur die schnelle Navigation zur Startansicht, sondern auch die innere Navigation gut möglich; so kommt man schnell zur vorherigen Ansicht oder ganz zurück. Eine Vorwärtsnavigation ist momentan aufgrund des Platzmangels nicht implementiert, ließe sich aber im Rahmen einer Portierung auf iPadOS o.Ä. schnell umsetzten.

6.1.4 Konsistenz und Standards

Das Interface wurde in Anlehnung an Apples eigenes Design gestaltet. Daher sind Standards wie Farben, Icons und Navigationselemente vertraut. Das neue Kategorie-Interface-Framework ist ebenfalls für möglichst hohe Konsistenz innerhalb der Applikation zuständig, da es den Entwickler stark dazu motiviert, dieselben Kategoriebausteine zu verwenden. So wird gewährleistet, dass alle Kategorie-Karten bis auf ihren tatsächlichen Inhalt ähnlich aussehen.

6.1.5 Fehler-Prophylaxe

Durch ausreichend Abstand zwischen einzelnen klickbaren Elementen[2] werden sogenannte Slips, ein versehentliches Danebenklicken, vermieden. Ansonsten ist das Interface simpel genug, um wenige Fehler durch Missinterpretation anzunehmen, diese sollte aber ggf. genauer mithilfe weiterer Personen evaluiert werden.

6.1.6 Recognition vs. Recall

Zum Wiederfinden einer Kategorie ist es nicht nötig, sich die Position einer Kategorie zu merken oder sogar scrollend nach ihr zu suchen. Denn in der oberen Leiste sind alle Kategorien schnell anhand ihrer Icons wiederzuerkennen.

6.1.7 Flexibilität und Nutzbarkeitseffizienz

Die Individualisierbarkeit des Interfaces ist kein Fokus gewesen, ließe sich dementsprechend noch weiter ausbauen. Momentan existiert ein Darkmode (siehe Abbildung 6.1b), der allerdings mehr mit dem individuellem Geschmack als der individuellen Effizienzerhöhung zu tun hat. Zusätzlich besteht die Möglichkeit, die Unsortiert-Kategorie so einzustellen, dass nicht nur die Email hier landet, die zu keiner anderen Kategorien passt, sondern auch die, die kategorisiert werden konnte. Nun werden alle Emails zusätzlich in einer gemeinsamen Kategorie beibehalten, wie es im Interview vorgeschlagen wurde. Außerdem ist die Sortierungsweise zwischen den Kategorien, das heißt welche Kategorien in der Übersicht weiter oben dargestellt werden, individualisierbar. Momentan existiert zwar nur die Möglichkeit für „zuletzt eingetroffen“ und „statisch“, es ist aber durchaus denkbar, Kategorien anpinbar zu machen, um so einen weiter individualisierbaren Hybriden aus statischer und dynamischer Sortierung zu schaffen.

Zusätzlich gibt es mehrere Möglichkeiten, eine Kategorie zu öffnen bzw. zu schließen: einerseits über die Leiste und anderseits über das Symbol innerhalb der Kategorie. Hier kann die Nutzerin also ihre priorisierte Variante verwenden.

[2] In dem Momentanen Implementationsstand sind die Einträge der Unsortiert-Kategorie eventuell etwas nah beieinander, das könnte in einer quantitativen Evaluation mit Nutzerin herausgefunden werden.

6.1.8 Ästhetik und Minimalismus

Die Ästhetik ist subjektiv, aber den Interviewteilnehmenden gefiel das Design. Es existieren außerdem keine sinnlosen Designelemente. Man könnte allerdings die Karten, die die Kategorien umgeben, entfernen. Das würde jedoch in der Vorschau problematisch werden, da dann die unterschiedlichen Kategorien nicht mehr gut voneinander zu trennen sind. Es wäre zu überlegen, ob man bei einer maximierten Kategorie diese immer noch auf einer Karte darstellen möchte. Für eine Entfernung der Karte spräche, dass dadurch wenig, anderweitig verschwendeter Platz frei wird, da weniger irrelevante Elemente dargestellt werden. Andererseits wird damit die Konsistenz genommen, da nicht mehr konsequent Kategorien auf einer Karte dargestellt sind.

6.1.9 Erholung und Diagnose von Fehlern

Das Hauptproblem, das dieser Heuristik noch nicht ganz gerecht wird, entsteht dadurch, dass man momentan relativ schnell (auch durch einen Slip) eine Kategorie schließen oder wechseln kann. Das ist auch gewünscht. Aber das Problem ist, dass die Kategorie ihren Zustand verloren hat, wenn zur Erholung aus dem Fehler die Kategorie schnell wieder geöffnet wird. Das heißt, dass z. B. die vorherige Navigation einer Nutzerin , die tief in eine Kategorie navigiert hat und wieder zurückkehren möchte, umsonst war und sie wieder in der Anfangsansicht der Kategorie landet.

Um das zu beheben, müssten Änderungen in dem Framework vorgenommen werden, sodass der Zustand der Kategorien in bestimmten Momenten nicht volatil ist.

6.1.10 Hilfe und Dokumentation

Momentan existiert noch keine Hilfe oder Dokumentation zu dem neuen Interface. Aufgrund der geringen Komplexität des neuen Interfaces scheint ein kurzes Onboarding hier eine angemessene Wahl zu sein, dieses wurde aber noch nicht im Rahmen dieser Arbeit umgesetzt.

6.2 Cognitive Walkthrough und Vergleich zu alten Interfaces

Ein weiteres Modell zur Evaluation ohne Nutzerteilnahme ist das sogenannte **G**oals-**O**perators-**M**ethods-**S**election (GOMS) Modell [22]. Dieses Modell basiert allerdings auf der präzisen Messung bestimmter Operationen zur quantitativen Evaluation und ist daher im Rahmen dieser Arbeit nicht verwendet worden. Jedoch lassen sich einige Konzepte nachahmen und die einzelnen benötigten Operatoren für exemplarische Ziele (Goals) im Bereich der Email zwischen dem neuen, kategorisierten und dem alten, klassischen Interface vergleichen.

Ein Beispielsziel wäre das Antworten auf eine vor einigen Tagen von einer Freundin eingetroffene kurze Nachricht. Bei beiden Interfacetypen setzt sich dieses Ziel aus den drei groben Methoden Nachricht finden, Nachricht öffnen und Nachricht beantworten zusammen.

6.2.1 Nachricht finden

Klassische Inbox

Eine Methode, die zur Nachricht führen würde, wäre simpel nach der Nachricht zu scrollen. Das würde allerdings eine Weile dauern. Deshalb wird die Methode per Suchleiste zu suchen selektiert. Hierzu wird

(a) Das Optionsmenü (b) aktivierter Darkmode

Abbildung 6.1: Das neue Interface

zuerst auf die Suchleiste geklickt und, da der Inhalt der Nachricht nicht bekannt ist, der Name des Senders eingegeben. Nachdem die Suche durchgeführt wurde, wird wieder (aufgrund vermutlich kleiner Trefferliste nur kurz) gescrollt und die Email kann nun geöffnet werden.

Kategorisierte Inbox

In einem kategorisierten Interface – sei es das hier konzipierte oder die Kategorien aus Google Inbox – sind die Operationen etwas anders. Hier wird zuerst die Kategorie gewählt. Dabei fallen in Googles Inbox die drei Operationen Seitenpanel Öffnen, zur Kategorie Scrollen, Auswählen an. In meinem Interface hingegen kann direkt die Kategorie in der oberen Leiste gewählt werden. Falls es viele Kategorien gibt, ist eventuell auch ein kurzes Scrollen notwendig. Dennoch gibt es hier ein Speed-Up von mindestens einer Operation.

Nachdem die Kategorie gewählt wurde, kann zur entsprechenden Nachricht gescrollt werden. Diese Operation würde in einer klassich kategorisierten Inbox [3] eventuell eine Weile dauern, sodass es wahrscheinlich der bessere Weg gewesen wäre, klassisch manuell zu suchen. In manchen Implementationen sind aber vorher dennoch die drei Operationen zum Öffnen der Kategorie notwendig, da die Suche nicht kategorieübergreifend ist.
In dem hier konzipierten Interface hingegen sind in der gewählten Chat-Kategorie die Nachrichten nach Nutzer gruppiert, sodass nur bis zu dem gesuchten Gesprächspartner gescrollt werden muss. Da anzunehmen ist, dass die Anzahl der Nachrichten die der Gesprächspartner (vorallem im Bereich der Chat-Nachrichten) um ein Vielfaches übersteigt, sollte der Scrollvorgang dem der klassisch-kategorisierten Inbox stark überlegen sein und das manuelle Suchen mit Suchbegriff ineffezienter sein.

[3] mit einer klassisch kategorisierten Inbox sind die Implementationen gemeint, die zwar eine Kategorisierung der Email mit sich bringt, jedoch die Kategorien selber eher wie automatische Ordner funktionieren und die Emails innerhalb der Kategorie genauso dargestellt werden, wie in der klassischen Inbox, d.h. ohne kategoriespezifisches User-Interface

Sollte allerdings das manuelle Suchen gewünscht sein, so ist dies aufgrund der kategorieübergreifenden Suche auf der Startseite direkt möglich, wobei mit nur einem zusätzlichem Klick nach der korrekten Kategorie gefiltert werden kann.

Das neue Interface ist den klassischen also überlegen, da es zusätzlich zu bereits bestehenden Methoden noch (in diesem Beispiel schnellere) Alternativmethoden bietet.

6.2.2 Nachricht öffnen

In der klassischen sowie in der klassisch kategorisierten Inbox wird nun einfach auf die gefundene Email getippt. Ähnlich ist es in dem neuen Interface: Hier wird einfach auf den gefundenen Kontakt getippt und der Thread der zuletzt ausgetauschten Chat-Nachrichten ist für erhöhten Kontext zu sehen.

6.2.3 Nachricht beantworten

Das Beantworten von Nachrichten läuft klassischerweise so, dass man auf einen Reply-Button klickt, um dann in einer Antworten-Ansicht zu landen, in der jeglicher Kontext fehlt und daher der Cognitive-Load stärker beansprucht werden muss. Nach Verfassen der Nachricht wird diese durch Tippen auf einen Versende-Knopf abgeschickt.

Der Prozess ist in dem neuen Interface – bis auf einen wichtigen Punkt – sehr ähnlich. Denn hier wird, sobald der Kontakt im vorherigen Schritt ausgewählt wurde, am unteren Ende der Ansicht sofort eine Antwortleiste[4] eingeblendet, wie sie sich in Instant-Messagern bewährt hat. So kann direkt die entsprechende Nachricht mit Blick auf den Kontext verfasst werden.

6.3 Email spezifisch

„Flow: As people are working on other tasks, they want to keep up with the flow of incoming messages as they arrive. Triage: After people are away from their Email for a period of time, they need to catch up and deal with all the Email that accumulated while they were away. Task management: People often use Email to remind them what they need to do, and to help them get tasks done. Archive: People store Email so they can refer to it later. Retrieve: After archiving messages, people need a method of retrieving messages.“ [43]

Das sind vier Email-spezifische Eigenschaften, die in einem Email Client gut umgesetzt sein sollten.

6.3.1 Flow

Wie in einer klassischen Inbox sollten neue Nachrichten am oberen Ende der Inbox auftauchen bzw. je nach gewählter Sortierung die entsprechende Kategorie mit nach oben ziehen. Ebenfalls sinnvoll sind Push-Benachrichtigungen, so dass eine Nutzerin , egal womit sie sich gerade beschäftigt, direkt sieht, welche Kategorie gerade welche Email empfangen hat. (Diese sind aber aktuell noch nicht implementiert.)

6.3.2 Triage

Eine Kategorie kann simpel den von meinem Framework zur Verfügung gestellten Unread-Counter aktivieren, welcher dann in der Ecke einer Kategorie anzeigt, wie viele ungelesene Emails vorhanden sind.

[4] siehe dazu Abbildung C.3

Eine weitere Verbesserungidee wäre, dass der Unread-Counter auch bei den kleinen Kategorie-Icons in der Leiste erscheint, damit auf einen Blick sofort erfasst werden kann, in welcher Kategorie ungelesene oder neue Nachrichten vorhanden sind.

Eine weitere Idee besteht darin, zwischen neu und ungelesen zu differenzieren, und dafür zwei beispielsweise verschiedenfarbige Counter zur Verfügung zu stellen. Das ist momentan jedoch nicht vorgesehen.

6.3.3 Archive und Retrieve

Aktuell ist noch keine Archivierungs- oder Dearchivierungsfunktionalität integriert. Auch hier wäre eine weitere Kategorie denkbar, die beispielsweise verschiedene Archive zur Verfügung stellt. Dann könnte zu den Optionen (siehe Abbildung 6.1a) ein neuer Archivierungsknopf hinzugefügt werden, der eine Email einem Archiv zuordnet.

6.4 Joy of Use

Da das neue Interface in erster Linie im Hinblick auf den Konsum von Email im privaten Bereich (d.h. nicht im Arbeitsumfeld) konzipiert wurde, ist neben den klassischen Evaluationsansätzen zur Bewertung der Nutzbarkeit auch die sogenannte Joy of Use interessant. „Joy-of-Use bezeichnet das positive, subjektive Empfinden einer Person, das im Zusammenhang mit der Benutzung eines Softwareproduktes auftritt“ [27]. Die Ästhetik hatte bei den Interviewteilnehmenden einen positiven Eindruck hinterlassen. Das subjektive Empfinden könnte Gegenstand zukünftiger Untersuchungen sein. Mein Ansatz war es, mit einer nutzerfreundlichen Oberfläche die Funktionalität und damit die Bereitschaft zur Verwendung von Email zu erhöhen.

7 Schlusswort und Ausblick

Zusammenfassend wurden in dieser Arbeit 3 Ziele erreicht. Nachdem das Problem der Typ-Überladung analysiert wurde, wurden verschiedene innovative mobile Emailclients untersucht, um deren Merkmale und Umgang mit Typ-Überladung zu erkennen. Dabei stellte sich heraus, dass es unter anderem für verschiedene Typen unterschiedliche Clients gibt; so kann man z. B. Unibox verwenden, wenn man einen Hauptfokus auf Instant-Messaging-ähnlichen Umgang legen möchte. Ebenfalls existieren einige Clients, die bereits die Kategorisierung der verschiedenen Typen vornehmen. Es gibt allerdings nur sehr wenige Ansätze, die sich mit jeweils eigenen, speziell typspezifisch designten Interfaces beschäftigen.

Deshalb wurde im Rahmen dieser Arbeit die Grundidee, die aus einem Forschungsteam von Yahoo [14] stammt, genau betrachtet und ein Prototyp erstellt, der genau diesen Anforderungen der Überladung durch verschiedene Kategorien innerhalb von Email gerecht werden soll.

Anschließend wurden die Erkenntnisse bereits bestehender Arbeiten zur Typ-Überladung nochmal mit eigenen Interviews untersucht. Innerhalb der Interviews wurden zudem die designten Prototypen vorgestellt und von potenziellen Nutzern evaluiert.

Mithilfe des daraus entstandenen Feedbacks wurde ein Kategorieframework zur Einführung verschiedener Kategorien in ein im Rahmen des Letterbox-Projektes erstellten UI entwickelt. Dieses Framework enthält bereits das umgebende UI zur Navigation zwischen den Kategorien sowie zwei Beispielskategorien. Das Framework wurde mit dem Ziel entwickelt, sowohl eine möglichste gute, konsistente Nutzererfahrung zu gewährleisten, als auch das einfache Hinzufügen neuer Kategorien durch eine Entwicklerin zu ermöglichen.

So wird die Funktionalität der klassischen, unkategorisierten Inbox nicht eingeschränkt, sondern um Funktionalität erweitert. Das Navigieren zwischen den Kategorien ist mit nur je einem Klick möglich. Ähnlich einfach ist das Erstellen neuer Kategorien: Eine Entwicklerin muss dafür lediglich einen vom Framework zur Verfügung gestellten Struct mit Kategoriename etc. parametrisieren und einer Liste hinzufügen. Um das Zustandsmanagement etc. kümmert sich das Framework automatisch. Auch das grundlegende UI und die Darstellung der Kategorien ist bereits implementiert; lediglich das kategoriespezifische UI muss mithilfe von SwiftUI erstellt werden und der Kategorie neben Namen und Filter übergeben werden.

7.1 Ausblick

7.1.1 Kategorisierung

Ein großes Problem bei der Darstellung von kategorie-sortierter Email ist es, die Emails den korrekten Kategorien zuzuordnen. Da Emails statisch sind, speichert das Framework die Zuordnung einer Email persistent, was es ermöglicht, rechenintensive Zuordnungsalgorithmen zu verwenden, da diese nur jeweils einmal pro

Email ausgeführt werden müssen. In den beiden beispielhaften Kategorien findet auch nur ein sehr simpler Filtervorgang statt: Die Dateikategorie überprüft, ob die Email Anhänge hat, während die Chat-Kategorie nur Nachrichten mit unter 200 Zeichen als *.shouldOpen* deklariert. Das sind zwei sehr produktionsferne Filtermethoden, die so wahrscheinlich nur zu Frust unter den Nutzerinnen führen würden.

Eine Lösungsmöglichkeit wäre das Erstellen komplexerer Filtermethoden mit regulären Ausdrücken und komplexeren Abfragen, dies hat allerdings einen hohen Entwicklungsaufwand.

7.1.2 Maschinelles Lernen

Ein besserer Lösungsansatz wäre das Einsetzen von maschinellem Lernen (ML). Einen Ansatz dazu findet man beispielsweise in der Arbeit von Koren et al. [21]. Maschinelle Lernmodelle ließen sich allerdings nicht nur zur Kategorisierung der Emails, sondern auch für das Herausfiltern wichtiger Informationen [14] für die Emailpreviews verwenden. Hier liegt ein großes Potenzial, das zukünftig weiterverfolgt werden könnte.

Ebenfalls könnte ML zur Extrahierung der Aufgaben aus Emails innerhalb einer ToDo-Kategorie verwendet werden.

7.1.3 Aufgabenverwaltung

Die bisherigen Ideen zur Aufgabenverwaltung beschäftigen sich vor allem mit dem Markieren von Emails als ToDo. Allerdings wird dieser Gedanke erweitert um eine Differenzierung in eigene und fremde ToDos [7, 43] sowie eine Zuordnung der Wichtigkeit eines ToDos [7, 10]. Sowohl die Zuordnung einer Wichtigkeit als auch das Herausfinden, was die ToDos in einer Email sind, kann dabei ML-gestützt geschehen [10]. Viele Arbeiten beschäftigen sich ebenfalls damit Erinnerungen für bestimmte Emails zu erstellen, um diese später genauer zu Lesen [32, 37, 37, 31, 7, 10].
Damit ergeben sich auch zwei Kategorien: ToDo und ToRead.

Es existeren bereits Forschungsarbeiten zur Verwaltung von ToDos innerhalb von Email, in Letterbox implementiert ist davon momentan noch nichts. Hier wären die bestehenden Ideen, wovon einige bereits unter C.2 gruppiert sind, genauer zu untersuchen und bei Bedarf in Letterbox zu integrieren.

7.1.4 UI

Auch an dem UI selbst gibt es noch Entwicklungsbedarf.

Kritik

Es gab zwei Kritikpunkte, die in meinen Interviews deutlich wurden.

Einer davon war, dass die klassische Inbox fehlen könnte. Dem wurde bereits durch Hinzufügen der klassichen Inbox begegnet. Wird diese allerdings verwendet, so verschwinden darin alle Vorteile der Kategoriesierung, da alle dort geöffneten Emails klassisch dargestellt werden. Ein besserer Ansatz wäre es, bei Verwendung der klassischen Inbox die Emails automatisch mit einem Tag zu versehen, zu welcher Kategorie sie zugeordnet wurden. Ein Auswählen des Tags öffnet diese Email dann mit dem UI der entsprechenden Kategorie. So kann eine Nutzerin alle Emails an einem Ort gesammelt haben und dennoch die Vorteile der Kategorisierung mit eigenem UI nutzen.

Der zweite Kritikpunkt war, dass ein neues, komplexeres Interface, vorallem für ältere Leute, zu kompliziert sein könnte. Da diese aber durchaus zur Zielgruppe gehören, könnte das Hinzufügen eines ständig

sichtbaren Hilfebuttons mit entsprechender Hilfe bzw. Dokumentation als auch ein kurzes Onboarding ein Lösungsansatz sein.

Sind zu einem späteren Zeitpunkt viele Kategorien vorhanden, wäre es außerdem sinnvoll, der Nutzerin im Onboarding die Möglichkeit zu geben, verschiedene Kategorien zur Verwendung auszuwählen.[1]

7.1.5 Skalierung auf verschiedene Größenverhältnisse

Zudem ist zu berücksichtigen, dass das UI auf besonders kleinen Geräten, wie einem iPod, zuviel Platz verbraucht. Hier sollte dann eventuell auf einige Features, wie die permanente Kategorieleiste, verzichten werden. Das Problem existiert aber zunehmend auch in die andere Richtung: Da Smartphones zunehmend größer werden [28],ist es schwer, das obere Ende des Bildschirms zu erreichen. Samsung präsentierte hier eine gute Lösung, indem der obere Teil des Bildschirms in erster Linie zum Darstellen von Inhalten verwendet wird, während die Interaktion primär unten, im leichter erreichbaren Bereich, stattfindet [26].

Die zwei Hauptelemente zum Finden von Email in meinem Prototyp, die Suchleiste und die Kategorieleiste, befinden sich momentan am oberen Ende. Es kann durchaus sinnvoll sein, sie auf großen iPhones oder sogar generell nach unten zu verlegen. Die Kategorieleiste ließe sich außerdem als eine `TabBar` interpretieren, welche sich nach Apples Guidelines am unteren Ende befinden sollte. Für iPadOS und MacOS könnte ein Desktop-ähnlicheres UI entwickelt werden, in dem beispielsweise in verschiedenen Spalten die Kategorieübersicht neben der momentan geöffneten Nachricht dargestellt werden.

7.1.6 Weitere Kategorien

In meiner Implementation wurde neben zwei Beispielskategorien vor allem ein Framework zum Erstellen neuer Kategorien gebaut. Weitere Kategorien, die noch implementiert werden könnten, sind unter B.1 zu sehen.

Würde man die hier vorgestellte Idee der eigenen Interfaces für verschiedene Kategorien bzw. Typen von Email ins Extreme ziehen, so ließe sich Email als eine Art menschenlesbares Transportprotokoll interpretieren, auf dessen Basis verschiedene Anwendungen, die Kategorien, laufen. So könnten Emails in Clients, die diese Funktionalität unterstützen, durch die entsprechenden Kategorien bzw. Apps geöffnet werden, während in klassichen Clients die Nachricht dennoch lesbar ist. Das könnte Gegenstand weiterer Forschung und Entwicklung sein. [2]

[1] Diese sollten allerdings auch über Einstellungen veränderbar sein

[2] Allerdings ist grundsätzlich die Zukunft von Email zu untersuchen. Denn der Hauptvorteil von Email im Vergleich zu Instant-Messengern oder ähnlichen Tools ist die Interkompatibilität zwischen verschiedenen Clients aufgrund der Verwendung eines einheitlichen Protokolls. Dieser könnte durch aufstrebende Alternativen wie `https://matrix.org/` schwinden.

A Implementation

Die vollständige Implementation kann unter https://git.imp.fu-berlin.de/enzevalos/enzevalos_iphone/-/commit/4fa31f89726bd08ff363173183e11575c2d4b1b4 nachvollzogen werden.

Ebenfalls wurde im Rahmen dieser Arbeit das unter Absatz 5.5 beschriebene Navigationframework als eigene Bibliothek ausgelagert. Diese ist unter https://github.com/HannesGitH/TagNavVi zu finden.

B Anhang

B.1 Kategorien

Weitere denkbare Kategorien:

- Eine Kategorie zum späteren Lesen von Emails: zu-Lesen [43, 10, 36, 30, 29, 32]. Diese könnte eventuell auch ein Unterteil von ToDo sein. So könnte die ToDo-Kategorie unterteilt werden in dringend zu tun, später zu tun, zu lesen, auf Reaktion des Kommunikationspartner wartend [43, 7].
- Eine Kategorie zum Verwalten der Zuordnungen von Emails. Hier könnten später verschiedene Zuordnungsparameter durch die Nutzerin angepasst werden, sowie nicht zuordbare Emails wie in einer Klassischen Inbox angezeigt werden und manuell zugeordnet werden können. Da das ML-gestützte Zuordnen allerdings nicht Teil dieser Arbeit ist, wurde der erste Teil vorerst wegfallen.
- Die Kategorien News [17, 36, 30], Reisen [14], Werbung [31] bzw. Coupons und Angebote [14, 17] sowie Jobs [17], Rechnungen [30, 17] und Medizinisches [17], und Wichtiges [30, 29, 37].
- Eine Kategorie die Speziell zum Antworten und Konzentrierten Lesen von Emails optimiert ist [30].
- Eine „Unsicher“-Kategorie, welche sich speziell mit Phishing etc. beschäftigen könnte.
- Eine Clip-Kategorie in welcher Emails oder einzelne Passagen angezeigt werden, die vorher angeklippt (bzw. angepinnt/favoritisiert) wurden [33, 36, 29].
- Zur unterstützung von ‚Tags‘ wäre auch eine Tag-Kategorie sinnvoll, welche eine Übersicht über die Tags gibt (siehe Abbildung 3.3b). Und bei Auswahl eines Tags erneut die Inbox anzeigt, allerdings nun nach diesem Tag gefiltert.

B.2 Interview

B.2.1 Kurze Einführung

- wer bin ich?
- Ziel meiner Bachelorarbeit
- gerne ehrliches Feedback
- darf ich das Interview aufnehmen? (nur Ton wäre OK)

B.2.2 Allg. Fragen

- kurze Info über dich:
  - Alter
  - Geschlecht
  - Informatikhintergrund
- nutzt du Email eher privat oder für die Arbeit?
- folgende Fragen beziehen sich auf den privaten (nicht Arbeitsumfeld) Bereich
- Nutzt du Email täglich?
- vorallem am Handy, oder am PC?
  - wieso?/wieso nicht?
- eher senden oder eher lesen?
- was sind die Arten von Email die du erhältst und welche davon ließt du?
  - Accountverwaltung
  - Werbung / Deals
  - private Nachrichten
  - Aufgaben Management
  - Reisen/ Tickets
  - Jobs
  - Bestätigungen/ Belege/ Rechnungen
  - weitere
- wofür Nutzt du deine Email noch?
  - Dateien/ Aufgaben/ Notizen an sich selber senden
  - etc

B.2.3 Spezifischere Fragen

- sinnvoll diese Arten einem eigenem UI zuzuordenen? (z. B. Tickets anders darstellen als private Nachrichtenverläufe)
  - wieso, wieso nicht?
  - vollautomatisch?
- Welche Kategorien verdienen ihr eigenes UI/ welche würdest du Nutzen?
- Welche Email Features besonders sinnvoll/ oft genutzt?
  - Threading
  - Ordner
  - Tags/Flags
  - Stars/Pins

  - weitere Ideen

B.2.4 Prototyp vorführen

- Stell dir vor das wäre deine Inbox.. (nur Prototyp, d.h. Emailinhalte etc. nicht berücksichtigen)
- was denkst du von diesen Designs?
  - Favoriten? Wieso?
  - Vorschau von Kategorien, welche Infos sollten angezeigt werden?
- was weglassen, was hinzufügen?
- allgemeine Änderungsvorschläge

C | Sammlung

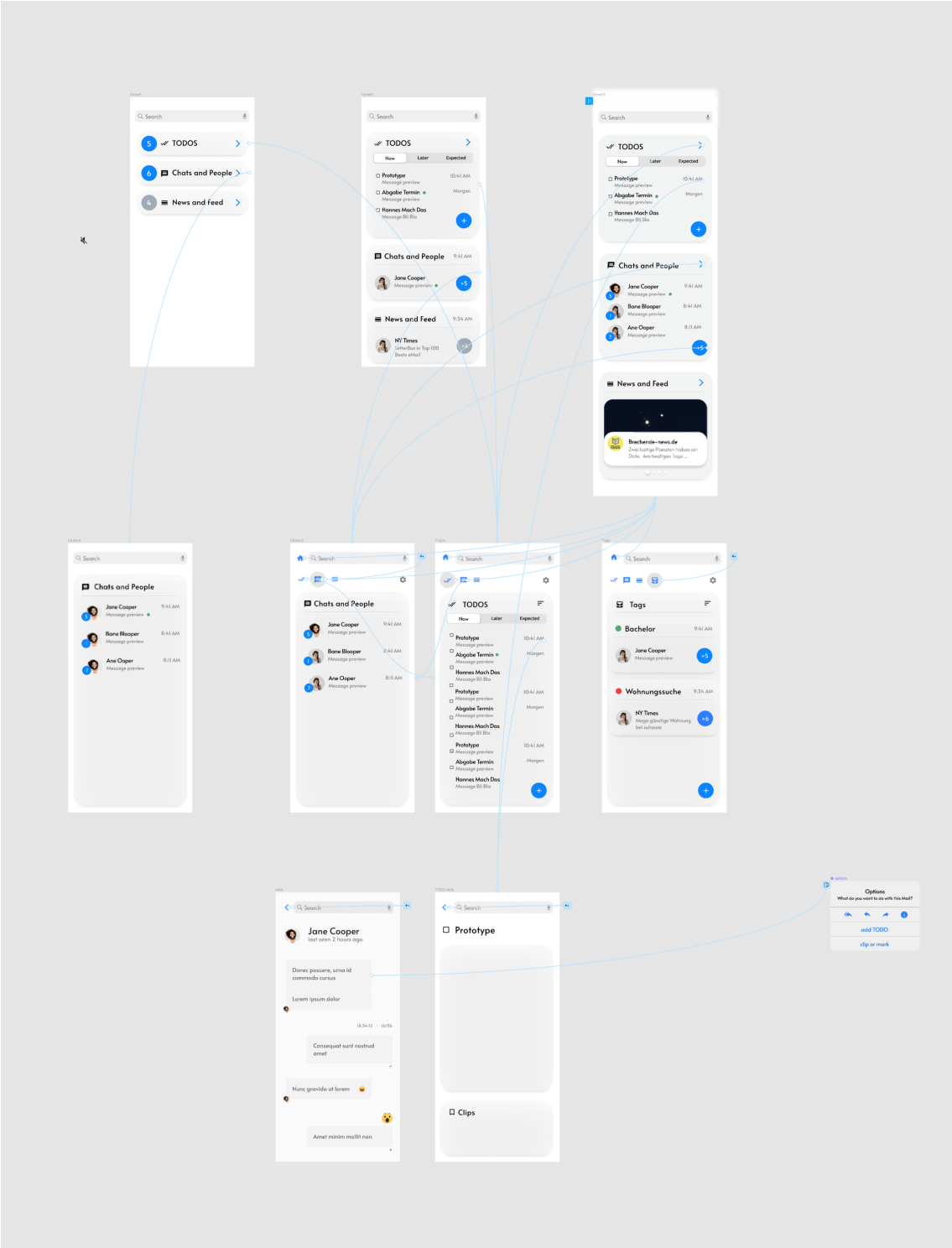


Abbildung C.1: High-fidelity Prototyp der neuen kategorisierten inbox in Figma

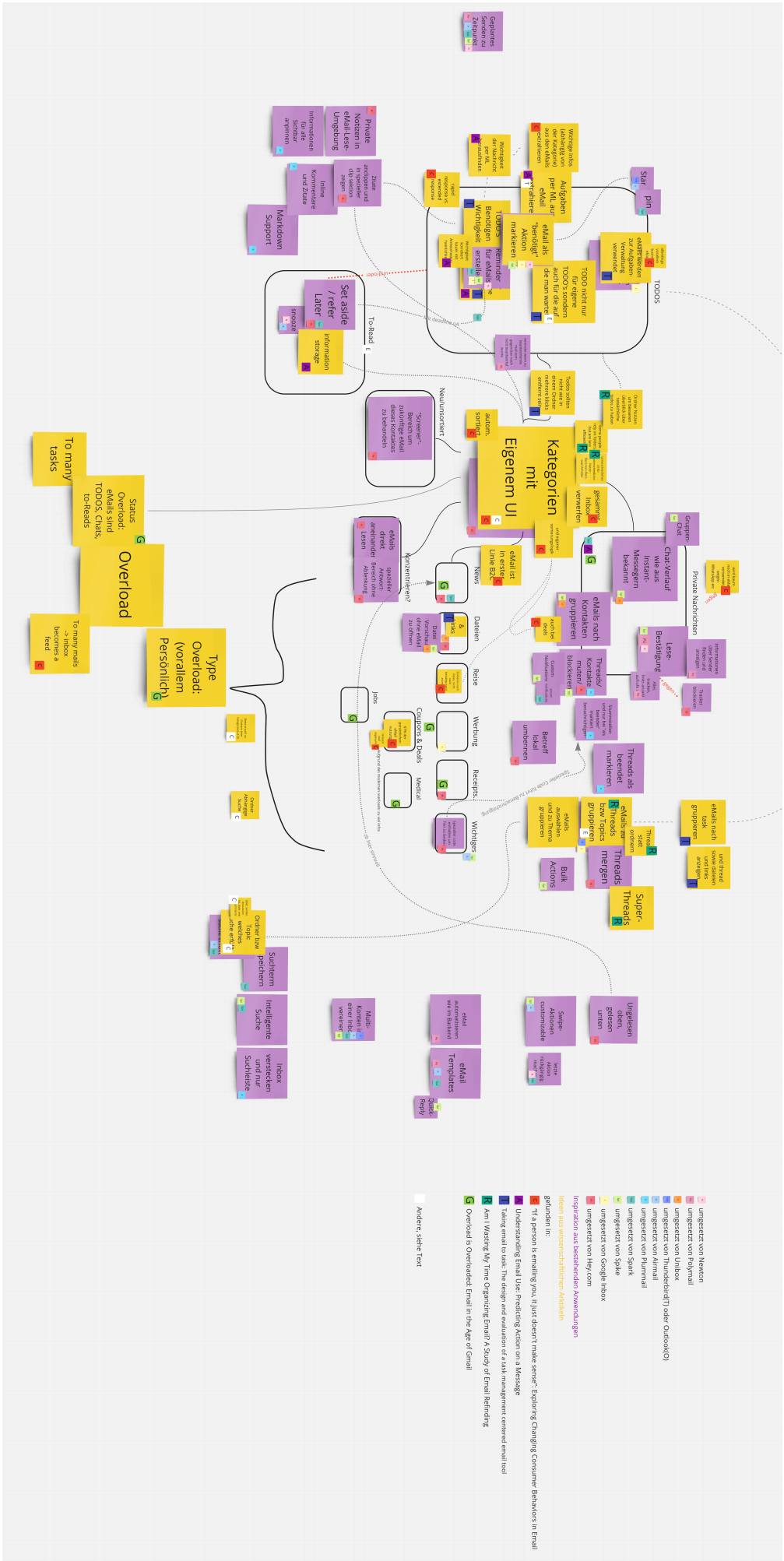


Abbildung C.2: Gruppierung der Ideen

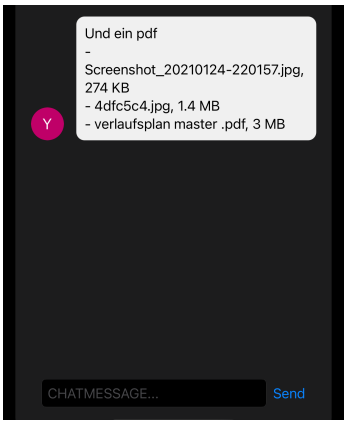


Abbildung C.3: Eine Beispielhafte Chat-Ansicht mit Antwort-Eingabezeile am unteren Ende

Literaturverzeichnis

[1] Inbox by gmail unterstützt endlich das iphone x. https://www.appgefahren.de/wp-content/uploads/2018/07/inbox-by-gmail.jpg, 2018. Eingesehen am 17.10.2020.

[2] Hey mail menu. https://hey.com/assets/home/preview-go.webp, 2020. Eingesehen am 12.11.2020.

[3] Spike read receipts. https://www.spikenow.com/wp-content/uploads/04_Read-Recipts_static-1.png, 2020. Eingesehen am 12.11.2020.

[4] Yam.ch. https://web.archive.org/web/19980630085235/http://www.yam.ch/future1.html, 1998. Eingesehen am 09.10.2020.

[5] Marcelo G Armentano and Analía A Amandi. Enhancing the experience of users regarding the email classification task using labels. *Knowledge-based systems*, 71:227–237, 2014.

[6] Olle Bälter. Keystroke level analysis of email message organization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, page 105–112, New York, NY, USA, 2000. Association for Computing Machinery.

[7] Victoria Bellotti, Nicolas Ducheneaut, Mark Howard, and Ian Smith. Taking email to task: The design and evaluation of a task management centered email tool. *Palo Alto Research Center*, 2003.

[8] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, March 2003.

[9] Simon Corston-Oliver, Eric Ringger, Michael Gamon, and Richard Campbell. Task-focused summarization of email. Microsoft Research https://www.aclweb.org/anthology/W04-1008.pdf.

[10] Laura A. Dabbish, Robert E. Kraut, Susan Fussell, and Sara Kiesler. Understanding email use: Predicting action on a message. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, page 691–700, New York, NY, USA, 2005. Association for Computing Machinery.

[11] Email thread. https://www.techopedia.com/definition/1503/Email-thread, 2017.

[12] Konstantin Deichmann. Entwurf und implementierung nutzerzentrierter ansätze zur verbreitung von ende-zu-ende-verschlüsselung. Master's thesis, FU Berlin, 2018.

[13] Eurostat. Personen, die das internet zum senden/empfangen von e-mails genutzt haben. https://ec.europa.eu/eurostat/tgm/table.do?tab=table&init=1&plugin=1&language=de&pcode=tin00094, 2019. Eingesehen am 26.06.2020.

[14] Nazanin Andalibi Frank Bentley, Nediyana Daskalova. „if a person is emailing you, it just doesn't make sense": Exploring changing consumer behaviors in email. https://dl.acm.org/doi/pdf/10.1145/3025453.3025613, 2017.

[15] Google. Email overload. https://scholar.google.com/scholar?cites=12425354786116165928&as_sdt=2005&sciodt=0,5&hl=de, 2021.

[16] Mihajlo Grbovic, Guy Halawi, Zohar Karnin, and Yoelle Maarek. How many folders do you really need? classifying email into a handful of categories. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, page 869–878, New York, NY, USA, 2014. Association for Computing Machinery.

[17] Catherine Grevet, David Choi, Debra Kumar, and Eric Gilbert. Overload is overloaded: Email in the age of gmail. *School of Interactive Computing, Google*, 2014.

[18] Apple Inc. Combine. https://developer.apple.com/documentation/combine/, 2021. Eingesehen am 21.02.2021.

[19] Apple Inc. Swift. https://swift.org/, 2021. Eingesehen am 03.03.2021.

[20] Apple Inc. Swiftui. https://developer.apple.com/documentation/swiftui/ und https://developer.apple.com/xcode/swiftui/, 2021. Eingesehen am 21.02.2021.

[21] Yehuda Koren, Edo Liberty, Yoelle Maarek, and Roman Sandler. Automatically tagging email by leveraging other users' folders. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, page 913–921, New York, NY, USA, 2011. Association for Computing Machinery.

[22] Prof. James A. Landay. Action analysis. https://hci.stanford.edu/courses/cs147/2014/au/lectures/16-action-analysis-automated-tools.pdf, 2014.

[23] Page Laubheimer. Cards: Ui-component definition. https://www.nngroup.com/articles/cards-component/, 2016.

[24] Project enzevalos. https://www.inf.fu-berlin.de/groups/ag-si/enzevalos.html. Eingesehen am 16.10.2020.

[25] Jakob Nielson. 10 usability heuristics for user interface design. https://www.nngroup.com/articles/ten-usability-heuristics/, 1994.

[26] Samsung's one ui: a singular smartphone experience. https://news.samsung.com/global/samsungs-one-ui-a-singular-smartphone-experience, 2019.

[27] Inga Elisabeth Reeps. Joy-of-use – eine neue qualität für interaktive produkte. http://kops.uni-konstanz.de/bitstream/handle/123456789/6320/Masterarbeit_IngaReeps.pdf?sequence=1&isAllowed=y, 2004.

[28] Evolution of the mobile phone. https://www.tigermobiles.com/evolution/#seventhPhone, 2019.

[29] Airmail. https://airmailapp.com/#ios. Eingesehen am 9.10.2020.

[30] Hey mail. https://hey.com/how-it-works/. Eingesehen am 14.10.2020.

[31] Google inbox. https://www.youtube.com/watch?v=ayFAQ2OoJaA. Eingesehen am 14.10.2020.

[32] Newton mail. https://newtonhq.com/. Eingesehen am 9.10.2020.

[33] Microsoft outlook. https://outlook.live.com/owa/. Eingesehen am 9.10.2020.

[34] plum mail. https://plummail.co. Eingesehen am 04.11.2020.

[35] Polymail. https://polymail.io/product. Eingesehen am 9.10.2020.

[36] Spark. https://sparkmailapp.com/. Eingesehen am 14.10.2020.

[37] Spike. https://www.spikenow.com/. Eingesehen am 14.10.2020.

[38] Mozilla thunderbird. https://www.thunderbird.net/de/features/. Eingesehen am 9.10.2020.

[39] Unibox. https://www.uniboxapp.com/#people. Eingesehen am 9.10.2020.

[40] INC. THE RADICATI GROUP˙ Email statistics report, 2019-2023. Technical report, The Radicati Group, 2019.

[41] Bruce Tognazzini. about bruce tognazzini. https://asktog.com/atc/about-bruce-tognazzini/.

[42] Bruce Tognazzini. about bruce tognazzini. https://asktog.com/atc/principles-of-interaction-design/.

[43] Gina Danielle Venolia, Laura Dabbish, JJ Cadiz, and Anoop Gupta. Supporting email workflow. Technical report, Microsoft Research, 2001.

[44] Steve Whittaker, Tara Matthews, Julian Cerruti, Hernan Badenes, and John Tang. Am i wasting my time organizing email? a study of email refinding. *IBM Research*, 2011.

[45] Steve Whittaker and Candace Sidner. Email overload: Exploring personal information management of email. *Lotus Development Corp.*, 1996.

[46] Martin Wilhelm. Mobile e-mail-nutzung auf hohem niveau. https://newsroom.web.de/2018/08/31/mobile-e-mail-nutzung-auf-hohem-niveau/, 2018. Eingesehen am 26.06.2020.