



# Schriftliche Ausarbeitung

Erstellung einer Applikation zur Verwaltung von  
Todos, Events und Notizen

Erstellt von den **AngryNerds**:

Fabia Schmid

Florian Rath

Jan Beilfuß

Joscha Nassenstein

Robin Menzel

Ruthild Gilles

Sertan Cetin

Yannick Rüttgers

Prüfer:

Prof. Dr. Seifert

Eingereicht am:

07.02.2019

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>IV</b>
<b>Listingverzeichnis</b>	<b>V</b>
<b>1 Das Team: Angry Nerds</b>	<b>1</b>
<b>2 Ziele des Projektes (Fabia Schmid)</b>	<b>2</b>
<b>3 Projektplanung</b>	<b>4</b>
3.1 Beschreibung des Funktionsumfangs (Florian Rath) . . . . .	4
3.2 Projektlaufplan (Fabia Schmid) . . . . .	4
3.3 Planung der Software . . . . .	9
3.3.1 Planung des Mock-Ups (Robin Menzel) . . . . .	9
3.3.2 Planung der Datenstruktur und Schnittstellen (Ruthild Gilles) .	15
3.3.3 Planung der Activities und Layouts (Florian Rath) . . . . .	19
3.3.4 Planung der Navigation zwischen den Activities (Yannick Rüttgers)	19
3.4 Geplante Aufgabenverteilung im Team (Fabia Schmid) . . . . .	19
<b>4 Beschreibung des Projektverlaufs</b>	<b>21</b>
4.1 Tatsächliche Aufgabenverteilung im Team (Fabia Schmid) . . . . .	21
4.2 Teammeetingprotokolle . . . . .	22
4.3 Projekttagebücher aller Teammitglieder . . . . .	22
4.3.1 Projekttagebuch Jan Beilfuß . . . . .	22
4.3.2 Projekttagebuch Joscha Nassenstein . . . . .	23
4.3.3 Projekttagebuch Fabia Schmid . . . . .	26
4.3.4 Projekttagebuch Florian Rath . . . . .	28
4.3.5 Projekttagebuch Robin Menzel . . . . .	29
4.3.6 Projekttagebuch Ruthild Gilles . . . . .	31
4.3.7 Projekttagebuch Sertan Cetin . . . . .	33
4.3.8 Projekttagebuch Yannick Rüttgers . . . . .	34
4.4 Beschreibung von Problemen . . . . .	35
4.4.1 Probleme von Fabia Schmid . . . . .	35
4.5 Probleme von Florian Rath . . . . .	35
4.5.1 Probleme von Sertan Cetin . . . . .	36
4.6 Probleme von Yannick Rüttgers . . . . .	36

<b>5 Dokumentation der Software</b>	<b>37</b>
5.1 Dokumentation der Paketstruktur (Sertan Cetin) . . . . .	37
5.2 Dokumentation der Activities . . . . .	38
5.2.1 Main Activity . . . . .	38
5.2.2 Todo Activity . . . . .	57
5.3 Dokumentation der Navigation zwischen Activities (Yannick Rüttgers) . . . . .	67
5.4 Dokumentation der Activity-übergreifenden, persistenten Datenhaltung (Jan Beilfuß) . . . . .	67
5.5 Dokumentation der Activity-übergreifenden Klassen (Ruthild Gilles) . . . . .	68
5.5.1 Models-Klassen . . . . .	68
5.5.2 Service-Klassen . . . . .	68
<b>6 Fazit der Teammitglieder</b>	<b>70</b>
6.1 Fazit von Fabia Schmid . . . . .	70
6.2 Fazit von Florian Rath . . . . .	71
6.3 Fazit von Jan Beilfuß . . . . .	72
6.4 Fazit von Joscha Nassenstein . . . . .	72
6.5 Fazit von Robin Menzel . . . . .	72
6.6 Fazit von Ruthild Gilles . . . . .	72
6.7 Fazit von Sertan Cetin . . . . .	73
6.8 Fazit von Yannick Rüttgers . . . . .	74
<b>7 Quellenverzeichnis</b>	<b>76</b>
<b>8 Anhang - Quelltexte</b>	<b>77</b>
8.1 Activities . . . . .	77
8.2 Data . . . . .	77
8.2.1 Service Klassen (Ruthild Gilles) . . . . .	77
8.3 Overview . . . . .	79
8.4 Main Activity . . . . .	79
<b>9 Anhang - Verwendete Tools und Hilfsmittel (Robin Menzel)</b>	<b>80</b>

Ehrenwörtliche Erklärung

## Abbildungsverzeichnis

Abbildung 1: Die Angry Nerds . . . . .	1
Abbildung 2: Projektstrukturplan . . . . .	6
Abbildung 3: GANTT-Diagramm . . . . .	8
Abbildung 4: Übersicht . . . . .	11
Abbildung 5: Event Activity - Unsere Events . . . . .	12
Abbildung 6: Note Activity - Unsere Notizen . . . . .	13
Abbildung 7: Todo Activity - Unsere ToDos . . . . .	14
Abbildung 8: Klassendiagramm . . . . .	16
Abbildung 9: Systemkontextdiagramm . . . . .	17
Abbildung 10: CRUD-Klassen . . . . .	18
Abbildung 11: Paketstruktur . . . . .	37
Abbildung 12: Overview Activity . . . . .	40
Abbildung 13: Landscape Ansicht - Overview Activity . . . . .	41
Abbildung 14: Note Fragments . . . . .	42
Abbildung 15: Todo Fragments . . . . .	43
Abbildung 16: Event Fragments . . . . .	44
Abbildung 17: Overview Activity - Löschen . . . . .	46
Abbildung 18: Overview Activity - Suchen . . . . .	48
Abbildung 19: Usecase Diagramm der Overview . . . . .	50
Abbildung 20: Klassendiagramm . . . . .	52
Abbildung 21: Todo Activity im Landscape-Modus . . . . .	59
Abbildung 22: Todo Activity im Portrait-Modus . . . . .	60
Abbildung 23: Datumeingabe . . . . .	62

## **Listingverzeichnis**

Listing 1: Create Klasse (Ruthild Gilles) . . . . .	77
Listing 2: Read Klasse (Ruthild Gilles) . . . . .	78
Listing 3: Update Klasse (Ruthild Gilles) . . . . .	79
Listing 4: Delete Klasse (Ruthild Gilles) . . . . .	79

## 1 Das Team: Angry Nerds

**Abbildung 1:** Die Angry Nerds



Von links: Florian Rath, Sertan Cetin, Jan Beilfuß, Joscha Nassenstein, Ruthild Gilles, Yannick Rüttgers, Fabia Schmid, Robin Menzel

## 2 Ziele des Projektes (Fabia Schmid)

Das Projekt „TEN Manger“ umfasst die Planung und Entwicklung einer Applikation zur Erstellung und Verwaltung von ToDos, Events und Notes. Diese Applikation stellt die Prüfungsleistung für das Modul „Projekte der Wirtschaftsinformatik“ dar.

Die Umsetzung erfolgt durch das Team „Angry Nerds“, welches sich aus Ruthild Gilles, Fabia Schmid, Jan Beilfuß, Yannick Rüttgers, Robin Menzel, Florian Rath, Joscha Nassenstein und Sertan Cetin zusammensetzt.

Der Zeitrahmen für die Realisierung des Projektes erstreckt sich vom 05.09.2017 bis zum 07.02.2018. Die Organisation, wie die Vereinbarung von Meetings und die konkrete Aufgabenverteilung erfolgen selbstständig innerhalb des Teams.

Die Grundlage für das Projekt bilden sowohl Vorkenntnisse aus vorherigen Vorlesungen von den Teammitgliedern erworben wurden, sowie die Einführung in die Android Programmierung im Rahmen der Vorlesung "Projekte der Wirtschaftsinformatik".

Ziel des Projektes ist die Entwicklung einer Applikation in der Programmiersprache Java. Die Applikation soll auf einem Tablet mit dem Betriebssystem Android laufen und die TEN Verwaltung ermöglichen. Die Verwaltung soll die Erstellung von ToDos, Events und Notes ermöglichen, sowie die Verwaltung dieser. Die Verwaltung soll aus dem Anzeigen, Filtern, Bearbeiten und Löschen bestehen.

Neben der Applikation soll im Rahmen des Projektes noch eine ausführliche Dokumentation angefertigt werden, welche den ganzen Projektverlauf und das Endergebnis dokumentiert und erklärt. Diese muss fristgerecht mit der Applikation abgegeben werden.

Um das Ziel des Projektes zu erreichen wurden verschiedene Termine festgelegt, welche im Folgenden aufgelistet werden:

- 12.09.2018 – Abgabe Projekttagebuch
- 31.10.2018 – Abgabe Projekttagebuch
- 19.12.2018 – Abgabe Projekttagebuch
- 07.02.2019 – Abgabe Dokumentation per Mail
- 09.02.2019 – Vorstellung der Applikation und Abgabe der Applikation

Das Projekt kann nur als erfolgreich bezeichnet werden, wenn alle Termine fristgerecht erfüllt wurden.

## 3 Projektplanung

### 3.1 Beschreibung des Funktionsumfangs (Florian Rath)

Im Rahmen des Projektes „TEN-Manager“ müssen einige Funktionen umgesetzt werden. Die App soll die Verwaltung von Aufgaben (Todos), Ereignissen (Events) und Notizen (Notes) ermöglichen, diese Einträge werden als TENs bezeichnet.

Dem Benutzer sollen für alle TENs die Grundfunktionen Erfassung, Veränderung, Löschung und Betrachtung zur Verfügung stehen. Außerdem soll es dem Benutzer möglich sein die Ansicht einzelner TENs aufzurufen und dort die Eingabefunktion benutzen zu können. Die Funktionen sollen eine einfache und intuitive Bedienbarkeit aufweisen. Die Funktionalität soll durch eine minimale Menge von Interaktionen bestehen und die ganze App soll durch eine ansprechende und übersichtliche Gestaltung der Benutzeroberfläche visualisiert werden. Die App besteht aus vier Hauptansichten, die der Todos, Events und Notes, jede dieser Ansichten, soll wenn sie erstellt wurde in einer kürzeren Version in einer allgemeinen Übersicht dargestellt werden. Mit Todos sind Aufgaben gemeint, die der Benutzer erledigen möchte oder muss, diese sollen einen Titel, eine Beschreibung, einen Zeitraum und die Todos enthalten. Der Fortschritt der Todos soll durch eine Prozentanzeige angezeigt werden. Durch die Events soll der Benutzer zu spezifizierten Zeitpunkten an Ereignisse erinnert werden. Dazu kann der Benutzer einen beschreibenden Text eingeben. Außerdem soll der Benutzer entscheiden können, an welchen Zeitpunkten eine Erinnerung erfolgen soll.

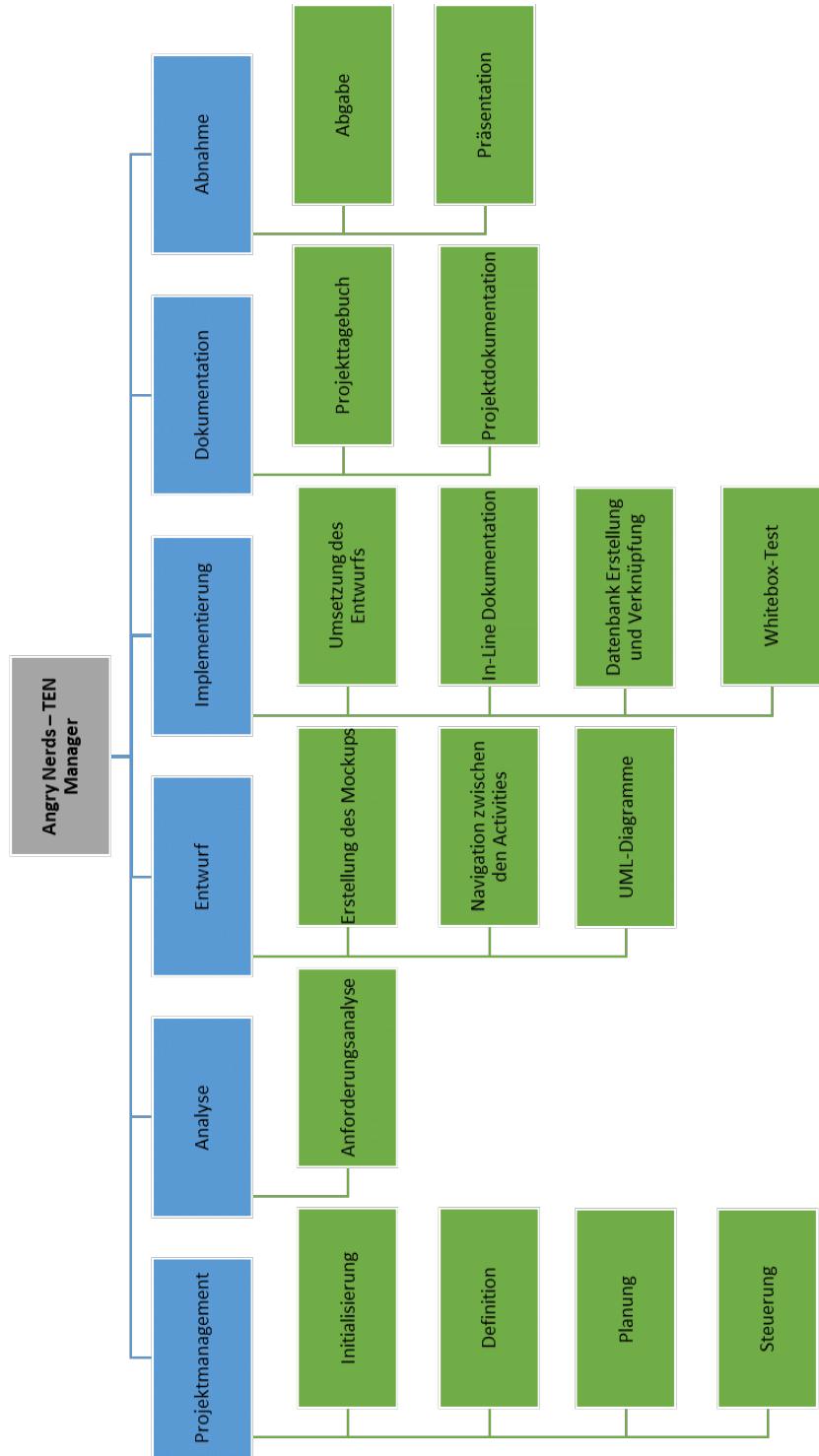
Die Notes ermöglichen es dem Benutzer Informationen zu speichern. Die Informationen können aus einer Notiz, Stichworten und Bildern bestehen. Die Daten der verschiedenen Ansichten sollen in einer Datenbank gespeichert werden, bzw. die Daten sollen dann auch wieder aus der Datenbank geladen werden können.

### 3.2 Projektablaufplan (Fabia Schmid)

Als Projektleiterin wählte ich als Vorgehensmodell für die Entwicklung das erweiterte Wasserfallmodell fest. Dafür sprach die klare Struktur des Modelles und der geringe Managementaufwand. Zusätzlich war die Übersichtlichkeit, sowie die leichte

Verständlichkeit ein klarer Vorteil. Außerdem sprach für das erweiterte Wasserfallmodell, dass es für kleine Projekte mit festgelegten Umfang ausgelegt und geeignet ist.

Auf diesem Vorgehensmodell basierte die Projektstrukturplanung, die Meilensteinplanung und die Zeitplanung, welche in einem GANTT-Diagramm visualisiert wurde.

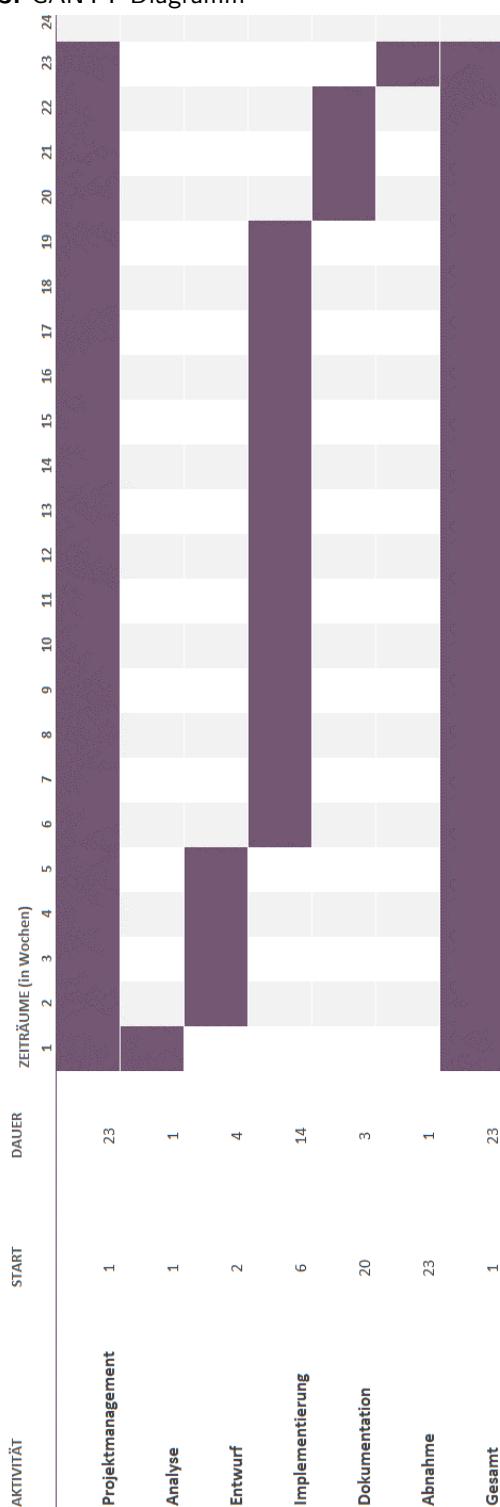
**Abbildung 2:** Projektstrukturplan

**Quelle:** Erstellt von Fabia Schmid

**Tabelle 1:** Meilensteinplan

ID	Meilenstein	Datum
1	Kick-Off gehalten	05.09.2018
2	Vorgehensmodell ausgewählt	12.09.2018
3	Datenbankmodel ausgewählt	14.09.2018
4	Activities verteilt	20.09.2018
5	Mockup fertig	20.09.2018
6	Activities und Layouts fertig	01.12.2018
7	Activities getestet	10.12.2018
8	Activities zusammengeführt	20.12.2018
9	App getestet	05.01.2019
10	Dokumentation fertig	01.02.2019
11	Ausarbeitungsabgabe	07.02.2019
12	Präsentation	09.02.2019

# Angry Nerds - GANTT Diagramm



**Quelle:** Erstellt von Fabia Schmid

### 3.3 Planung der Software

#### 3.3.1 Planung des Mock-Ups (Robin Menzel)

Das Mockup wurde von Beginn der Projektes an erstellt und ist das Ergebnis vieler Änderungen und Versionen. Während beim ersten Meeting zum Aussehen der App bereits Stilrichtung und Präferenzen der Gruppenmitglieder festgehalten wurden, entstand das erste Mock-Up erst einige Zeit später. Neben Handschriftlichen Zeichnungen zu Beginn des Designs, war das Programm Adobe Xd das ausgewählte Werkzeug für das Mock-Up. Adobe Xd war zu Beginn des Projektes erst in einer Beta Version verfügbar, welche für unsere Ansprüche jedoch ausreichte. Es ist ein vektorbasiertes Tool zum Entwerfen und Prototyping der User Experience für webbasierte und mobile Applikationen.

Das Design leitet sich von Googles Designsprache *Material Design* ab, welche besonders durch die materialartigen, kartenähnlichen Flächen und das Flat Design charakterisiert wird. Außerdem ist das Layout und die Farbgestaltung angelehnt an verschiedene Applikationen, wie z.B.

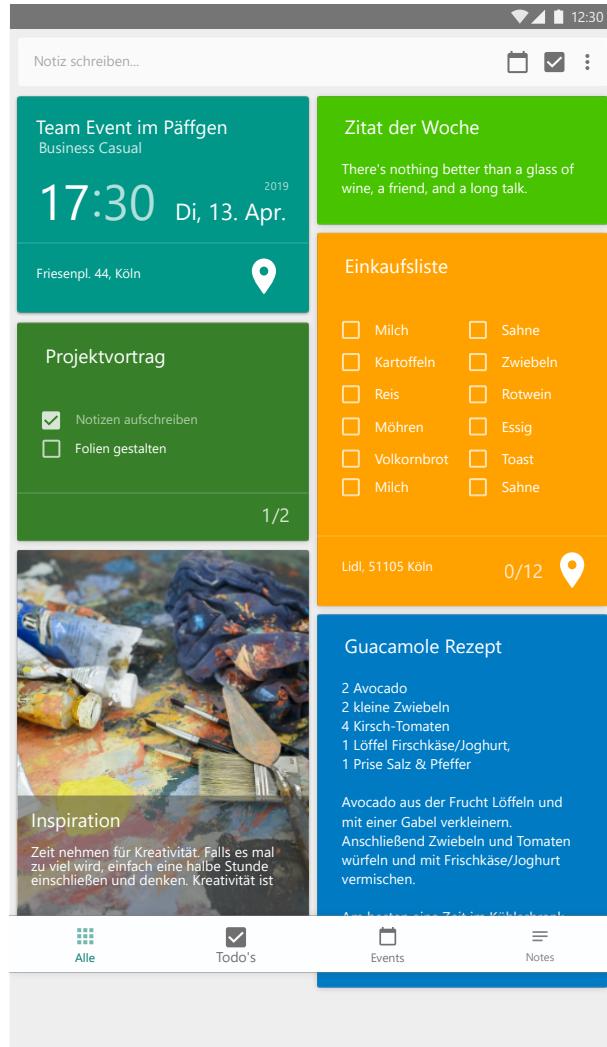
- Google Notizen
- Google Kalender
- Google Fotos
- Wunderlist

Dies liegt vor allem daran, dass in diesen Applikationen das Material Design sehr gut umgesetzt wurde. Aus Google Notizen wurde die Übersichtsseite und die charakteristische, schwebende Appbar übernommen. So sind auch unsere TENs in der Übersichtsseite als Fragmente dargestellt, in denen sich die wichtigsten Informationen schnell ablesen lassen. Die Detail- bzw. Bearbeitungsseiten der TENs wurde angelehnt an die Bearbeitungsseite des Google Kalenders. Hier wurden die Kategorien bzw. Einstellmöglichkeiten durch feine Linien visuell voneinander abgetrennt. Durch die eindeutigen Icons und dem auslassen von Beschriftungen wird der Bildschirm optimal genutzt. Auch Wunderlist schafft es in den Detailansichten von ToDos mit wenigen Hinweisen, eine minimalistische, aber intuitive und vor allem informative Darstellung zu schaffen, von der wir uns inspiriert haben. Bei Google Fotos wurde die untere Navigationsleiste übernommen.

So ist es in unserer App nun möglich mit einem Klick zwischen einer Übersicht über alle TENs, zu einer Übersicht über die unterschiedlichen Kategorien zu wechseln. Dies nutzt auf effektive Weise den Platz auf dem Tablet und bietet die Funktion an einer intuitiven Position an. Außerdem können wir so auf einen so genannten Navigation Drawer verzichten, da dieser die Applikation unnötig aufplustert und unsere Applikation kaum Funktionalitäten anbietet, die in diesem Drawer positioniert werden hätte können.

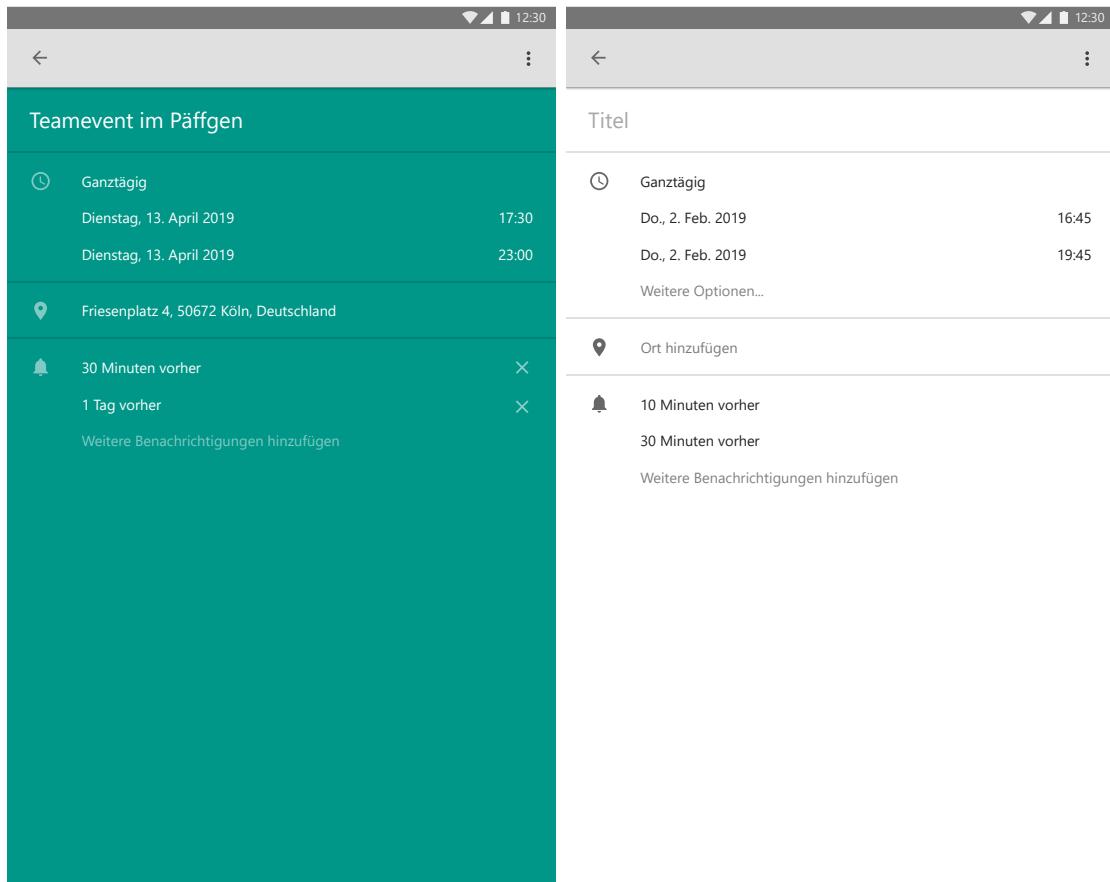
In Adobe Xd lassen sich nicht nur die Oberflächen Designen, sondern auch die User Experience abbilden. So lassen sich Flächen mit einander Verbinden, die nun durch einen Klick geöffnet werden können. Auf diese Art und Weise war es uns möglich Personen aus unserem Umkreis unsere App testen zu lassen. Das Feedback war ausschließlich Positiv. Da wir Elemente aus oft genutzten Apps übernommen haben, stellte die Bedienung unserer App für alle Tester kein Problem dar. Außerdem kam das Feedback, das unsere Oberfläche zur ersten Nutzung einlädt und nicht durch Überladung von Informationen abschreckt.

**Abbildung 4:** Übersicht



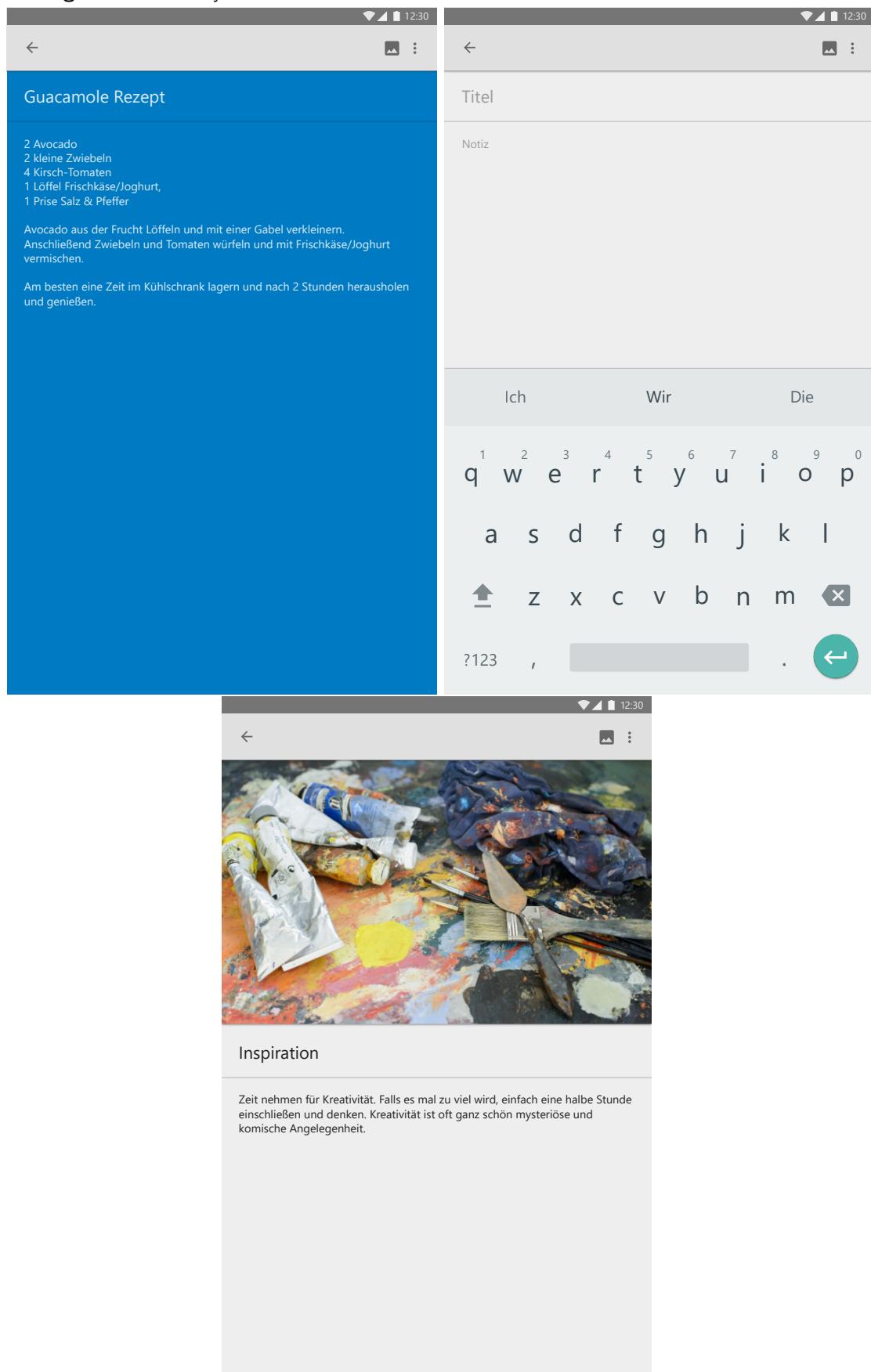
**Quelle:** Mockups - Übersicht über alle TENs

**Abbildung 5:** Event Activity - Unsere Events



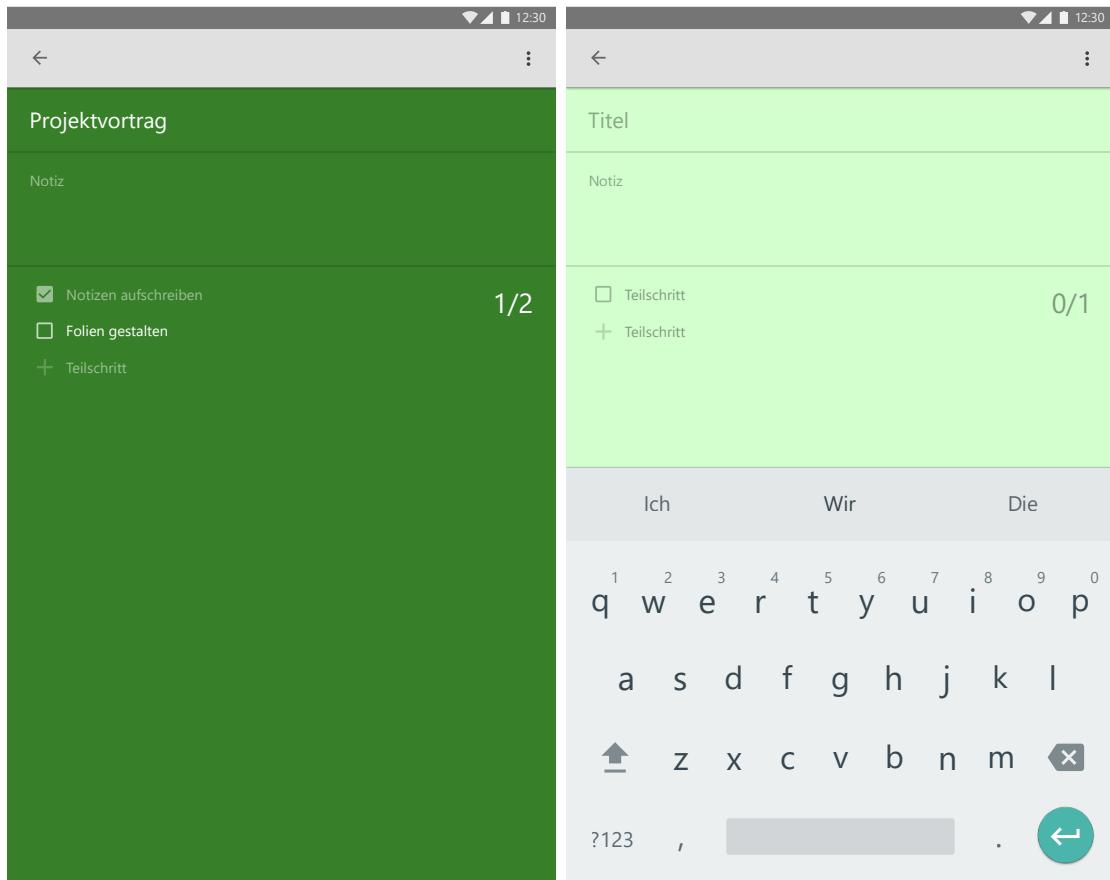
**Quelle:** Mockups - Übersicht über ein vorhandenes und ein neues Event

**Abbildung 6:** Note Activity - Unsere Notizen



**Quelle:** Mockups - Übersicht über ein vorhandene, eine leere und eine Bild-Notiz

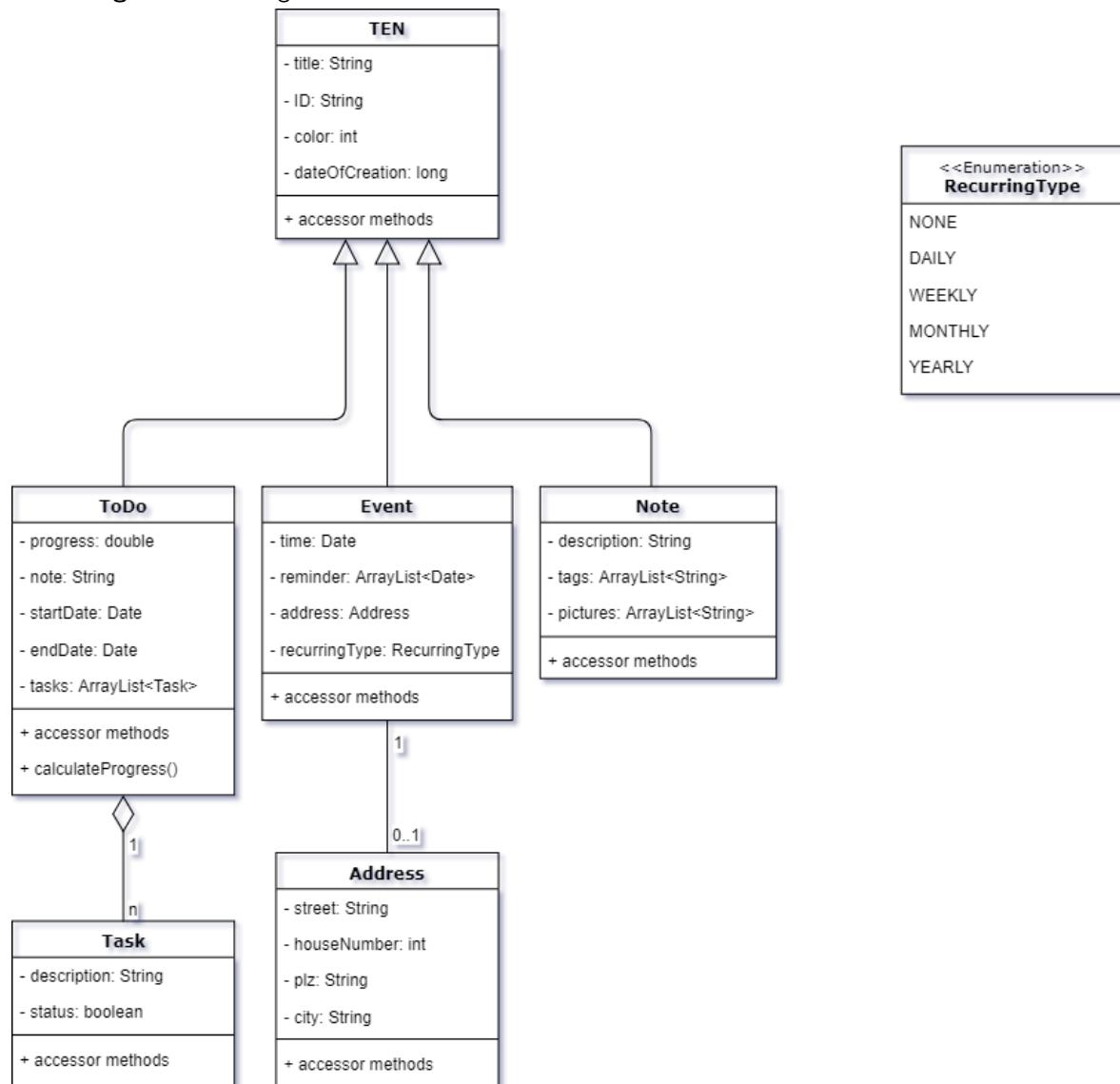
**Abbildung 7:** Todo Activity - Unsere ToDos



**Quelle:** Mockups - Übersicht über vorhandene und neue Todos

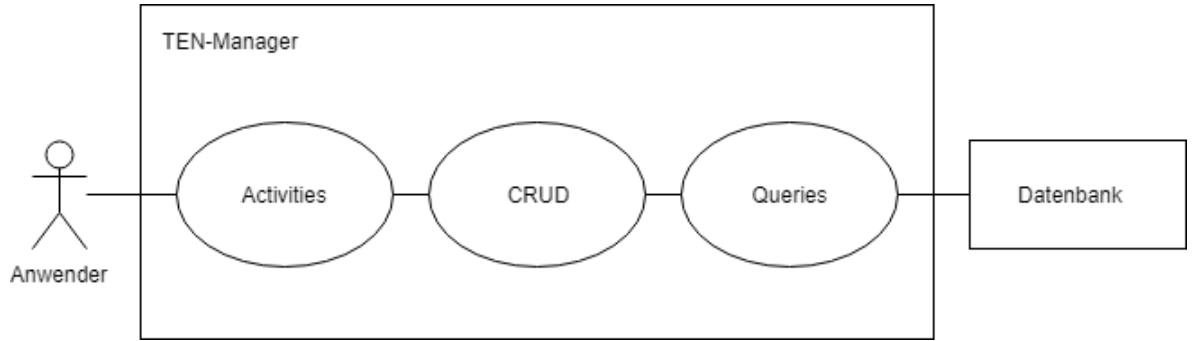
### **3.3.2 Planung der Datenstruktur und Schnittstellen (Ruthild Gilles)**

Die gewünschte Applikation soll das Managen von Todos, Events und Notes vereinfachen. Anhand der Anforderungen an die Applikation überlegte sich das Datenteam, welche Daten beziehungsweise Informationen in der Datenstruktur der Applikation abgebildet werden sollen. Da sowohl Todo-, Event-, als auch Note-Objekte einheitlich aufgebaut sein sollen, wurde sich dazu entschieden, dass die jeweiligen Klassen von einer TEN-Klasse erben. Alle Todo-, Event- und Note-Objekte benötigen eine ID zu eindeutigen Identifikation des Objektes auf der Datenbank und in der Applikation. Außerdem könne die Objekte jeweils einen Titel haben. Zudem sollen die verschiedenen Objekte weitere Informationen enthalten. In folgendem Klassendiagramm sind alle geplanten Attribute der Klassen aufgelistet.

**Abbildung 8:** Klassendiagramm

**Quelle:** Erstellt von Joscha Nassenstein

Zusätzlich zu der Struktur der Daten in Form von Klassen mit entsprechenden Attributten wurde ebenfalls die Struktur der Applikation vom Datenteam definiert. Diese ist in nachfolgender Abbildung in einem Systemkontextdiagramm dargestellt.

**Abbildung 9:** Systemkontextdiagramm

**Quelle:** Erstellt von Ruthild Gilles

Um die Daten auch nach Beendigung der Applikation bei erneutem Starten wieder anzeigen zu können, wurde eine dokumentenbasierte Datenbank an die Applikation angebunden. Auf diese Weise kann eine persistente Datenhaltung erzielt werden. Die einzelnen Activities, welche als Schnittstelle zu den Anwendern dienen, sollen die vom Benutzer eingegebenen Informationen auf der Datenbank speichern können. Dazu sollen Activity-übergreifende Klassen verwendet werden.

Das Datenteam plante die Activity-übergreifenden Klassen und deren Methoden anhand der Anforderungen, der einzelnen Activities. Es sollte möglich sein, einzelne oder auch alle TEN-Objekte von der Datenbank zu erhalten. Auch sollte das Löschen und das Speichern von einzelnen TEN-Objekten möglich sein. Während der Planungsphase wurden hier verschiedene Ansätze in Erwägung gezogen, um diese Anforderungen umzusetzen. Zur Übersichtlichkeit entschied sich das Datenteam letztendlich dafür, einzelne Klassen für jede der vier CRUD-Operationen zu erstellen. Die CRUD-Operationen beinhalten das Erstellen (Create), das Lesen (Read), das Aktualisieren (Update) und das Löschen (Delete) von einzelnen Objekten. Die einzelnen Methoden der CRUD-Klassen sind in folgender Abbildung dargestellt.

**Abbildung 10:** CRUD-Klassen

Create	Read	Update	Delete
<ul style="list-style-type: none"> <li>+ newTodo(): Todo</li> <li>+ newEvent(): Event</li> <li>+ newNote(): Note</li> </ul>	<ul style="list-style-type: none"> <li>+ getAllTENs(): ArrayTENs</li> <li>+ getTodoByID(String): Todo</li> <li>+ getEventByID(String): Event</li> <li>+ getNoteByID(String): Note</li> </ul>	<ul style="list-style-type: none"> <li>+ saveTEN(TEN):</li> </ul>	<ul style="list-style-type: none"> <li>+ deleteTEN(TEN):</li> <li>+ deleteMultipleTENs(ArrayTENs):</li> </ul>

**Quelle:** Erstellt von Ruthild Gilles

Da der Aufwand für die Umsetzung aller geforderten Anforderungen zu Projektstart lediglich grob geschätzt werden konnte, definierte das Datenteam abgesehen von der Schnittstelle zur Datenbank noch einige weitere Schnittstellen. Die Implementierung dieser weiteren Schnittstellen wurde nicht in den Anforderungen gefordert und würde nur bei genug Zeitüberschuss umgesetzt werden. Zu den weiteren optionalen Schnittstellen gehören das Exportieren von Todos, Events und Notes in die Zwischenablage oder auch in andere Applikationen, die auf dem entsprechenden Endgerät installiert sind. Für ein Event soll es die Möglichkeit geben eine Adresse hinzuzufügen. Hier wäre eine weitere optionale Schnittstelle die Verknüpfung mit Google Maps. Auch könnte eine Schnittstelle zu einer anderen Kalender App implementiert werden, in die ein Event exportiert werden könnte.

### 3.3.3 Planung der Activities und Layouts (Florian Rath)

Hier den Text einfach hin kopieren.

### 3.3.4 Planung der Navigation zwischen den Activities (Yannick Rüttgers)

Als Einstiegspunkt für den Nutzer soll eine Übersichtsactivity dienen. Von dieser aus sollen alle weiteren Activities aufgerufen werden können.

Die einzelnen Activities sollen auf zwei Arten aufgerufen werden können. Entweder soll ein neues TEN erstellt werden, oder ein bereits vorhandenes angezeigt werden.

Wenn ein neues TEN erstellt werden soll, soll die zugehörige Activity ohne weitere Parameter aufgerufen werden. Dadurch soll in dieser dann ein leeres TEN erstellt werden, welches dann bearbeitet werden kann.

Soll stattdessen ein bereits bestehendes TEN angezeigt werden, wird der Activity die ID des jeweiligen TEN übergeben, welches dann im weiteren Verlauf von der Activity geladen und angezeigt wird.

Wird aus den jeweiligen TEN-Activities zurücknavigiert, soll wieder die Übersichtsactivity angezeigt werden, in der die getätigten Änderungen angezeigt werden sollten.

## 3.4 Geplante Aufgabenverteilung im Team (Fabia Schmid)

Name	Aufgaben
Ruthild Gilles	Erstellung Service-Klassen, Schreiben eines Protokolls, Projekttagebuch führen, Dokumentation anfertigen
Fabia Schmid	Projektsteuerung und -planung, Erstellung der Layouts für die ActivityOverview, Erstellung der OnClickListener für die ActivityOverview, Projekttagebuch führen, Dokumentation anfertigen

Jan Beilfuß	Datenbankzugriffe, Start-Up-Lade-Routine von Note, Bilderhandling in Note und generell, Note-Applicationlogicstrukturierung, Projekttagebuch führen, Dokumentation anfertigen
Yannick Rüttgers	Erstellung ActivityOverview, Planung der Navigation zwischen Klassen, Erster Latexentwurfprojekttagebuch, Latex-Beauftragter, Projekttagebuch führen, Dokumentation anfertigen
Robin Menzel	Zusammenführung des Quellcodes, Erstellung des Mockups, Erstellung der Event Activity, Projekttagebuch führen, Dokumentation anfertigen
Florian Rath	Aufteilung der Activities, Erstellung Activity ToDo, Projekttagebuch führen, Dokumentation anfertigen
Joscha Nassenstein	Erstellung von Note, Erstellung der TEN-Klassen, Erstellung Datendiagramm, Projekttagebuch führen, Dokumentation anfertigen
Sertan Cetin	Aufteilung der Activities, Erstellung Activity ToDo, Projekttagebuch führen, Dokumentation anfertigen

## 4 Beschreibung des Projektverlaufs

### 4.1 Tatsächliche Aufgabenverteilung im Team (Fabia Schmid)

Name	Aufgaben
Ruthild Gilles	Erstellung Service-Klassen, Schreiben eines Protokolls, Latex-Beauftragte, Projekttagebuch führen, Dokumentation anfertigen
Fabia Schmid	Projektsteuerung und -planung, Erstellung der Layouts für die ActivityOverview, Erstellung der OnClickListener für die ActivityOverview, Projekttagebuch führen, Dokumentation anfertigen
Jan Beilfuß	Datenbankzugriffe, Start-Up-Lade-Routine von Note, Bilderhandling in Note und generell, Note-Applicationlogicstrukturierung, Unterstützung im Umgang mit Git, Mockdatenerstellung, Projekttagebuch führen, Dokumentation anfertigen
Yannick Rüttgers	Erstellung ActivityOverview, Planung der Navigation zwischen Klassen, Erster Latexentwurfprojekttagebuch, Latex-Beauftragter, Projekttagebuch führen, Dokumentation anfertigen

Robin Menzel	Zusammenführung des Quellcodes, Administration der Versionsverwaltung, Planung der Layouts, Hilfe bei der Aufteilung von den Activities, Erstellung des Mockups, Erstellung der Event Activity, Projekttagebuch führen, Dokumentation anfertigen
Florian Rath	Aufteilung der Activities, Erstellung Activity ToDo, Projekttagebuch führen, Dokumentation anfertigen
Joscha Nassenstein	Erstellung von Note, Erstellung der TEN-Klassen, Erstellung Datendiagramm, Projekttagebuch führen, Dokumentation anfertigen
Sertan Cetin	Aufteilung der Activities, Erstellung Activity ToDo, Projekttagebuch führen, Dokumentation anfertigen

## 4.2 Teammeetingprotokolle

Hier den Text einfach hin kopieren.

## 4.3 Projekttagebücher aller Teammitglieder

### 4.3.1 Projekttagebuch Jan Beilfuß

Beschreibung	Dauer	Datum
--------------	-------	-------

#### 4.3.2 Projekttagebuch Joscha Nassenstein

Beschreibung	Dauer	Datum
Aufgabe lesen und verstehen	30 min	05.09.2018
Kick-Off Meeting zur Besprechung der Aufgabe, Verteilung der Rollen	50 min	05.09.2018
Festlegung des Umfangs (Muss/Kann-Kriterien)	50 min	05.09.2018
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Übertragung der Ergebnisse in Microsoft Teams	10 min	05.09.2018
Wahl der LaTeX Distribution	20 min	12.09.2018
Download und Installation von LaTeX	80 min	12.09.2018
Erstellung des Datenmodells	50 min	15.09.2018
Digitalisierung des Datenmodells	20 min	15.09.2018
Team-Meeting	60 min	22.09.2018
Erstellung eines GitHub Accounts	10 min	13.10.2018
Installation und Einrichtung von GIT	30 min	13.10.2018
Absprache zur Vererbungsstruktur innerhalb des Data-Layers	20 min	23.10.2018
Besprechung der Aufgabenverteilung im Data-Team	60 min	26.10.2018
Besprechung der Layouts	30 min	27.10.2018
Pull des aktuellen Stands des Projekts aus dem Repository auf GitHub	10 min	28.10.2018
Implementierung der Klassen TEN sowie der daraus abgeleiteten Klassen ToDo, Event, Note sowie zugehörigen Models	80 min	28.10.2018
Anpassung von colors.xml	10 min	28.10.2018
Veränderung der Art der Übergabe von Farbwerten aus colors.xml in oben genannte Klassen	30 min	28.10.2018
Erstellung von Beispieldaten (ToDos, Events und Notizen)	40 min	30.10.2018
Erstellung des Projekttagebuchs in Latex	30 min	30.10.2018
Update der TEN-Klassen	30 min	17.11.2018
Merge der GIT-Branches	10 min	20.11.2018
Besprechung Data-Team	60 min	20.11.2018

Erstellung Klassendiagramm	20 min	20.11.2018
Besprechung Team Todo und Note	80 min	22.11.2018
Team-Meeting	50 min	04.12.2018
Besprechung der Anforderungen und der Implementierung von Todo und Note	150 min	18.12.2018
Besprechung der Aufteilung zwischen Todo und Note, Übernahme der Note Activity	20 min	21.12.2018
Design der Layouts für die Note Activity	200 min	27.12.2018
Übernahme der Klassenstruktur aus der Vorlesung und Beginn der Implementation	400 min	28.12.2018
Implementierung der Bildanzeige in der Vorschauansicht	250 min	29.12.2018
Implementierung der Bildanzeige auf Vollbildschirm	200 min	29.12.2018
Implementierung der Datenhaltung bei Konfigurationsänderung	80 min	29.12.2018
Implementierung des Bildimports	280 min	30.12.2018
Erstellung der Layouts für die NoteTagActivity	120 min	30.12.2018
Implementierung der NoteTagActivity	320 min	31.12.2018
Implementierung des Austauschs zwischen den Activities	80 min	31.12.2018
Verbesserung des ImageOverlays	60 min	07.01.2019
Verbesserung des Landscape-Layouts	140 min	07.01.2019
Verbesserung der Fehlertoleranz	90 min	08.01.2019
Implementierung der Toolbar	80 min	11.01.2019
Update des Layouts mittels Separatoren	40 min	13.01.2019
Änderung von Klassenattributen auf zentral angelegte Konstanten	40 min	13.01.2019
Umbenennung einiger Variablen zur Erhöhung der Lesbarkeit	60 min	14.01.2019
Implementierung der Bildkorrektur für ein korrekt ausgerichtetes Bild	120 min	16.01.2019
Implementierung der Teilen-Funktion	50 min	17.01.2019
Erweiterung der Suchfunktion auf Stichworte	20 min	18.01.2019
Anpassung des Bildimports an Android API 19	50 min	18.01.2019
Implementierung der Wischgeste für die Bildvorschau	240 min	20.01.2019
Erstellung der Dokumentation	100 min	02.02.2019

## Beschreibung des Projektverlaufs

---

Erstellung des Projekttagebuchs in Latex	50 min	02.02.2019
Erstellung der Dokumentation	160 min	03.02.2019

Summe in Minuten: 4410

### 4.3.3 Projekttagebuch Fabia Schmid

Beschreibung	Dauer	Datum
Aufgabe lesen und verstehen	30 min	05.09.2018
Kick-Off Meeting zur Besprechung der Aufgabe, Verteilung der Rollen	50 min	05.09.2018
Erstellung eines Meilensteinplans	50 min	05.09.2018
Vorstellung des Meilensteinplans und Besprechung der anderen Ergebnisse	40 min	05.09.2018
Aufarbeitung der Abgabetermine und Information der Gruppenteilnehmer	20 min	10.09.2018
Installation von Latex	120 min	10.09.2018
Festlegung des Vorgehensmodells, durch Abwägung von Vor- und Nachteilen der verschiedenen Modelle (Ergebnis: Erweitertes Wasserfallmodell)	40 min	11.09.2018
Teammeeting	60 min	22.09.2018
Github Anmeldung und Einbindung des Projektes	60 min	13.10.2018
Entwicklung des Layouts für das „Event“	70 min	20.10.2018
Entwicklung des Layouts für das „Note“	60 min	21.10.2018
Recherche über Listen, Checkboxen und Verwendung von mehreren Layouts	50 min	21.10.2018
Besprechung der Layouts (Aufbau, etc.)	50 min	27.10.2018
Entwicklung des Layouts für das „ToDo“	40 min	27.10.2018
Erstellung des Projekttagebuch mit Latex	40 min	30.10.2018
Zusammentragung der momentanen Projektstände und Abschätzung, ob die Meilensteine erreicht werden können	30 min	17.11.2018
Neuplanung eines Meilensteins und Abstimmung mit dem Team	20 min	26.11.2018
Teammeeting	50 min	04.12.2018
Koordination der anzufertigen Diagramme und Entwicklungsstand überprüfen	10 min	04.12.2018
Entwicklung des Layouts für das „Image“	20 min	12.12.2018
Entwicklung des Layouts für die Overview	60 min	02.01.2019
Aufteilung der Ausarbeitung	20 min	02.01.2019
Entwicklung des Layouts für das Bedienleisten-Fragment	20 min	04.01.2019

## Beschreibung des Projektverlaufs

---

Meilenstein Umplanung	10 min	15.01.2019
Erinnerung an die Erstellung der Kapitel der Ausarbeitung	10 min	23.01.2019
Dokumentation	240 min	03.02.2019

Summe in Minuten: 1270

#### 4.3.4 Projekttagebuch Florian Rath

Beschreibung	Dauer	Datum
Aufgaben lesen und verstehen	30 min	05.09.2018
Kick-Off Meeting zur Besprechung der Aufgabe, Verteilung der Rollen	50 min	05.09.2018
Erstellung eines Mockups	50 min	05.09.2018
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Installation von LateX	40 min	11.09.2018
Beginn der Planung Navigation zwischen Activities	50 min	11.09.2018
Teammeeting	60 min	22.09.2018
Rollenzuordnung der Activities	60 min	01.10.2018
Bekanntmachung der Zuordnung und Anpassung	30 min	02.10.2018
Installation und Einrichtung von GIT	30 min	13.10.2018
Besprechung der Layouts	30 min	27.10.2018
Beschäftigung mit LateX und Recherche	100 min	29.10.2018
Besprechung Team Todo und Note	80 min	22.11.2018
Team-Meeting	50 min	04.12.2018
Nachbearbeitung Meetingprotokoll und Upload	30 min	04.12.2018
Teilimplementation Todo und Note	150 min	18.12.2018
Beschäftigung mit Datepicker für Todo	180 min	04.01.2019
Implementierung der Colors und Start-/Enddatum	280 min	21.01.2019
Layoutanpassungen	250 min	22.01.2019
Layoutanpassungen und Funktionalitäten für Todo	290 min	26.01.2019
Bug fixes und weitere Funktionalitäten für Todo	310 min	30.01.2019
Dokumentation	210 min	03.02.2019

Summe in Minuten: 2400

#### 4.3.5 Projekttagebuch Robin Menzel

Beschreibung	Dauer	Datum
Aufgabe lesen und verstehen	30 min	05.09.2018
Kick-Off Meeting zur Besprechung der Aufgabe, Verteilung der Rollen	50 min	05.09.2018
Konzeption eines Mockups und den Activities	50 min	05.09.2018
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Installation von Adobe XD für MockUps	20 min	05.09.2018
Installation und Einrichtung von Git	20 min	05.09.2018
Erstellung und Einrichtung eines GitHub Repositorys	40 min	05.09.2018
Erstellung von MockUps in der ersten Version	180 min	06.09.2018
Wahl der LateX Distribution	20 min	10.09.2018
Installation von LateX	120 min	10.09.2018
Projekttagebuch pflegen	10 min	11.09.2018
Weiterentwicklung des MockUps	30 min	15.09.2018
Meeting zur Besprechung der Ergebnisse aus dem Design- und Daten-Team	60 min	22.09.2018
Korrekturen am MockUp und Teilen der Ergebnisse im Microsoft Teams	30 min	23.09.2018
Kommunikation mit Data Team im Bezug auf Objekte und Übergabe an Activities	30 min	23.10.2018
Erstellung des XML-Layouts der Event-Activity	180 min	27.10.2018
Recherche und Design von Farben für die Hintergründe von Activities und Erstellung von res-Files	60 min	28.10.2018
Probleme im Git Repository beheben und einrichten von Branches	30 min	28.10.2018
Erstellung des Projekttagebuchs in Latex	40 min	30.10.2018
Weiterentwicklung des XML-Layouts der Event-Activity	120 min	15.10.2018
Recherche Umsetzung der Datumsauswahl (DatePicker)	60 min	15.10.2018
Git Repository neu aufsetzen (Merger und neu anlegen)	180 min	01.12.2018
Teammeeting	50 min	04.12.2018
Erstellung Notwendiger Klassen für die Event Activity	600 min	10.12.2018
Recherche nach App oder Toolbar für API 19	120 min	12.12.2018

Implementation einer Toolbar für alle TENS	260 min	13.12.2018
Implementation des 3-Punkt-Menüs in der Toolbar	30 min	13.12.2018
Implementation des Öffnens der Activity (Öffnen mit ID, ohne, Übergänge)	90 min	17.12.2018
Erstellung des Projekttagebuchs in Latex	15 min	19.12.2018
Recherche DialogPicker um Zeit und Datum für Events auszuwählen	120 min	23.12.2018
Implementation von Date- und Time-Picker	300 min	27.12.2018
Implementation der dynamischen Reminder in der GUI und dem Auswahldialog	390 min	29.12.2018
Recherche Notification Service und Alarm Manager	90 min	30.12.2019
Implementation von Remindern inkl. Notification Service und Alarm Manager	120 min	31.01.2019
Ausführliches Testen 05.01.2019	120 min	04.01.2019
Anpassungen auf API 19	120 min	05.01.2019
Ausführliches Testen 09.01.2019	90 min	10.01.2019
Anpassungen an dem Alarm Manager um Reminder vor aktueller Zeit zu ignorieren	30 min	10.01.2019
Recherche nach Schnittstellen zu Google Maps	15 min	10.01.2019
Implementation eines Buttons um die Adresse eines Events in Google Maps zu öffnen	30 min 6	10.01.2019
GUI Anpassungen für den Navigations-Button	30 min	11.01.2019
Implementation einer Teilen-Funktionalität für Text-basiertes Event	45 min	13.01.2019
Implementation einer Export-Funktion in andere Kalender Applikationen	30 min	13.01.2019
Implementation der Wiederholungsfunktion (Einmalig, Täglich, ...)	180 min	17.01.2019
Erstellung der Dokumentation 1	120 min	28.01.2019
Erstellung des Projekttagebuchs in Latex	60 min	29.12.2018
Erstellung der Dokumentation 2	140 min	30.01.2019

Summe in Minuten: 4595

#### 4.3.6 Projekttagebuch Ruthild Gilles

Beschreibung	Dauer	Datum
Aufgabe lesen und verstehen	30 min	05.09.2018
Kick-Off Meeting zur Besprechung der Aufgabe, Verteilung der Rollen	50 min	05.09.2018
Festlegung des Umfangs (Muss/Kann Kriterien)	50 min	05.09.2018
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Installation von LaTeX	120 min	10.09.2018
Projekttagebuch ausfüllen	10 min	11.09.2018
Erstellung des Datenmodells	50 min	15.09.2018
Teammeeting	60 min	22.09.2018
Überlegung Aufgabenteilung, Aufgabenvergabe und Erklärung dieser bezogen auf die MainActivity	40 min	09.10.2018
Installation und Einrichtung von GIT	30 min	13.10.2018
Teammeeting Data-Team	60 min	26.10.2018
Klonen des Data-Branches von GIT	120 min	27.10.2018
Deklaration von gettern und settern für Datenobjekte	120 min	28.10.2018
Projekttagebuch ausfüllen	20 min	31.10.2018
Entwicklung von SetterService Klasse	180 min	16.11.2018
Teammeeting Data-Team	100 min	20.11.2018
Entwicklung von Service Klassen	120 min	21.11.2018
Überlegung neuer Struktur für Services	90 min	22.11.2018
Implementierung neuer Service-Struktur	120 min	30.11.2018
Einbindung der Mockdaten	80 min	01.12.2018
Implementierung weiterer Teile neuer Struktur	90 min	02.12.2018
Änderungen an neuer Service-Struktur	40 min	03.12.2018
Teammeeting	50 min	04.12.2018
Entwicklung von Create-Klasse	60 min	04.12.2018
Änderungen an Update-Klasse	40 min	17.12.2018
Projekttagebuch ausfüllen	20 min	18.12.2018
Bugfixing in Update- und Read-Klasse	90 min	02.01.2019
Mockdaten durch Zugriff auf Datenbank ersetzt	60 min	02.01.2019

Ergänzung einer Methode in Delete-Klasse	30 min	04.01.2019
Neue Farben für GUI festlegen	30 min	07.01.2019
Latex - Vorlage anpassen	120 min	14.01.2019
Latex - Struktur für Ausarbeitung erstellen	120 min	18.01.2019
Meinen Teil der Ausarbeitung schreiben	180 min	28.01.2019
Latex - Meetingprotokolle einfügen	180 min	01.02.2019
Latex - Quellcode einfügen	120 min	03.02.2019
Latex - Ausarbeitungen der anderen einfügen	180 min	03.02.2019
Projekttagebuch ausfüllen	20 min	03.02.2019
Latex - Projekttagebücher einfügen	120 min	03.02.2019

Summe in Minuten: 3090

#### 4.3.7 Projekttagebuch Sertan Cetin

Beschreibung	Dauer	Datum
Aufgaben lesen und verstehen	30 min	05.09.2018
Kick-Off Meeting zur Besprechung der Aufgabe, Verteilung der Rollen	50 min	05.09.2018
Erstellung eines Mockups	50 min	05.09.2018
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Installation von LateX	40 min	11.09.2018
Beginn der Planung Navigation zwischen Activities	50 min	11.09.2018
Teammeeting	60 min	22.09.2018
Rollenzuordnung der Activities	60 min	01.10.2018
Bekanntmachung der Zuordnung und Anpassung	30 min	02.10.2018
Installation und Einrichtung von GIT	20 min	13.10.2018
Besprechung der Layouts	30 min	27.10.2018
Beschäftigung mit LateX und Recherche	90 min	29.10.2018
Erstellung des Projekttagebuchs in LateX	20 min	30.10.2018
Besprechung Team Todo und Note	80 min	22.11.2018
Team-Meeting	50 min	04.12.2018
Teilimplementation Todo und Note	150 min	18.12.2018
Vertiefung von androidspezifischen Controls	60 min	19.12.2018
Erstellung des Projekttagebuchs in LateX	20 min	19.12.2018
Layout für Task Todos	130 min	03.01.2019
Implementierung Taskadapter (Schnittstelle zwischen Gui und Data)	200 min	04.01.2019
Emulator Problembehebung (gescheitert)	120 min	21.01.2019
Implementierung Todo Data und Progresstext	200 min	28.01.2019

Summe in Minuten: 1580

#### 4.3.8 Projekttagebuch Yannick Rüttgers

Beschreibung	Dauer	Datum
--------------	-------	-------

## 4.4 Beschreibung von Problemen

### 4.4.1 Probleme von Fabia Schmid

Im Lauf des Projektes konnten zwei Meilensteine nicht fristgerecht erfüllt werden. Einmal die Fertigstellung der Activities und die Fertigstellung der Dokumentation.

Die Fertigstellung der Activities und der Layouts war für den 01.12.2018 geplant, konnte jedoch nicht fristgerecht fertiggestellt werden, da projektfremde Tätigkeiten die Umsetzung verzögerten. Beispielsweise schrieben wir eine Klausur, die bei der Projektplanung noch nicht bekannt war. Als Ergebnis wurde der Meilenstein auf ein späteres Datum geplant und konnte danach fristgerecht erfüllt werden. Diese Verschiebung hatte jedoch keinen negativen Einfluss auf die Fertigstellung des Projektes.

Auch der Meilenstein „Dokumentation fertig“ konnte nicht fristgerecht zum 01.02.2019 erreicht werden. Auch in diesem Fall waren projektfremde Tätigkeiten die Ursache für den Verzug. Jedoch musste der Meilenstein nur um 3 Tage verschoben werden, wodurch das Projektergebnis nicht gefährdet wurde.

Weitere Probleme oder Verzögerungen traten nicht auf und das Projekt konnte erfolgreich am 09.02.2019 vorgestellt werden.

### 4.4.2 Probleme von Florian Rath

Im Zuge der Entwicklung tauchten einige Probleme auf, die unterschiedlichen Ursprüngen entstammten. Zum einen fiel auf, dass die Aufteilung der Entwicklung nach den Activities viel Kommunikation benötigte und einige Klassen redundant erstellt wurden. Die Aufteilung nach Elementen hätte hier sicherlich mehr Sinn gemacht. Dann wären zum Beispiel alle GUI-Klassen von einer Person entwickelt worden. Ein zusätzlicher Faktor für Problem war die Zeit, da die Entwicklung immer mal wieder unterbrochen wurde, kann man als Entwickler öfter aus dem Tritt.

Die Entwicklung mit Android Studio war in manchen Phasen ebenfalls problembehaftet. Der Emulator ließ sich aus unergründlichen Ursachen nicht starten und musste neuinstalliert werden. Nach manchen Abgleichen mit GitHub wies der Code Fehler auf und konnte nicht mehr kompiliert werden, das Problem ließ sich nur mit einem kompletten Download des Projektes bewerkstelligen.

#### **4.4.3 Probleme von Sertan Cetin**

Während der Entwicklung sind bei mir diverse Probleme aufgetreten. Ich habe in meinem privaten Rechner einen AMD Ryzen Prozessor, genauso wie in meinem Laptop. Leider war die Konfiguration mit dem Android-Emulator etwas komplizierter, da Ryzen die Hardwarebeschleunigung von Intel nicht unterstützt. Ich musste lange Recherchearbeit durchführen, bis ich einen Emulator zum laufen bekam. Allerdings stürzte dieser während der Entwicklung aus dem Nichts ab, sodass er bis heute immer noch nicht funktioniert. Ich hatte das Projekt von meinem lokalen Rechner komplett gelöscht und neu heruntergeladen. Nichts brachte die Lösung.

Damit ich trotzdem entwickeln konnte, benutzte ich mein eigenes Android-Handy. Jedoch wäre die Entwicklung mit einem Emulator deutlich entspannter gewesen.

In diesem Punkt betrachte ich Android Studio ein wenig als rückschrittig, wenn man fehlerfreie Tools von Visual Studio gewohnt ist. Bei Visual Studio Fehlern konnte ich im Internet viel schneller eine Lösung finden als bei Android Studio.

#### **4.4.4 Probleme von Yannick Rüttgers**

Die Probleme, denen während des Projektes auftraten, lagen hauptsächlich an der genutzten Entwicklungsumgebung, bzw. der genutzten Software.

Bereits zu Beginn des Moduls funktionierte weder Androidstudio, noch Latex korrekt. Nach dem Kauf eines neuen Laptops funktionierten die Programme zwar soweit, allerdings konnte aufgrund des AMD-Prozessors die Virtualisierung nicht optimal genutzt werden. Dies führte zu langen Wartezeiten und Fehlern beim Starten des Emulators. Durch Nutzung eines externen Smartphones konnte dies schließlich umgangen werden.

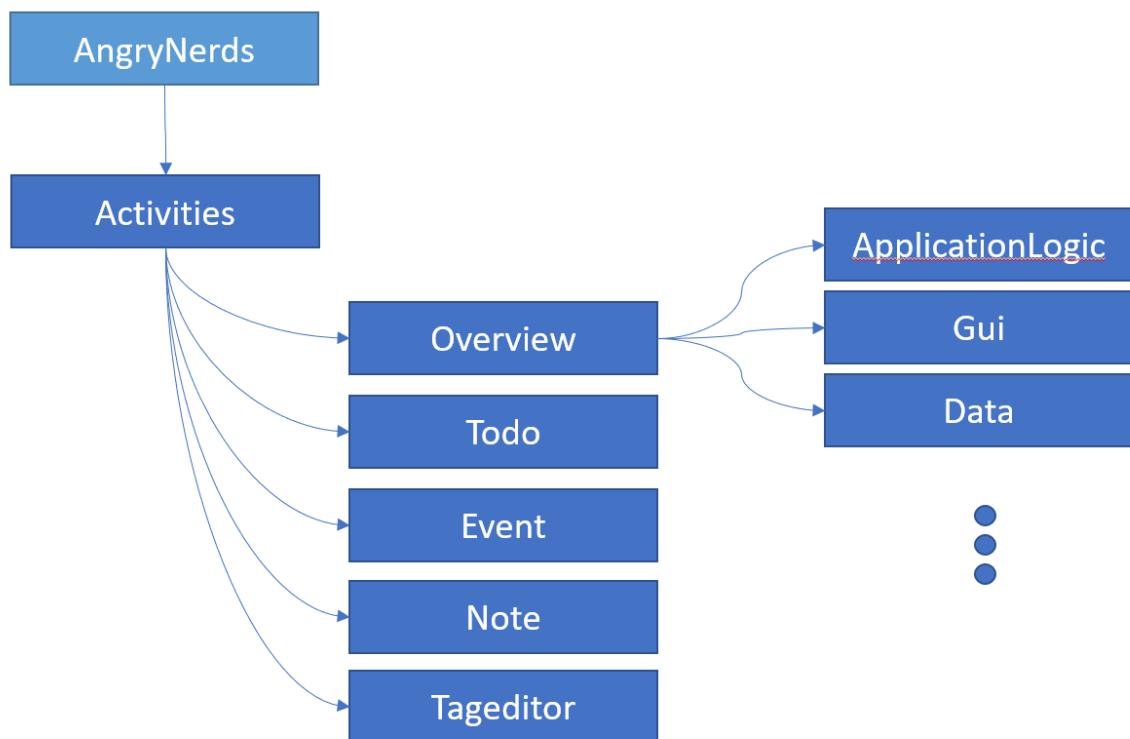
## 5 Dokumentation der Software

### 5.1 Dokumentation der Paketstruktur (Sertan Cetin)

Die entwickelte App ist innerhalb des Paketes com.example.robin.angrynerds-wip in mehrere Klassen unterteilt. Zu jeder Activity gehören mehrere Klassen. Um die Übersichtlichkeit zu steigern, wurden die Klassen und Activities in Ordnern untergebracht. Die gesamte Ordnerstruktur sieht wie folgt aus:

Das Paket beinhaltet insgesamt fünf Activities mit jeweils einer ApplicationLogic, Gui und Data.

**Abbildung 11:** Paketstruktur



**Quelle:** Erstellt von Sertan Cetin

## 5.2 Dokumentation der Activities

### 5.2.1 Main Activity

#### Aufgabe und Funktionen (Yannick Rüttgers)

Die in diesem Kapitel erläuterte Activity Overview dient dem Nutzer als Einstiegspunkt in die Applikation. Von hier aus werden alle weiteren Activities aufgerufen.

Wenn der Nutzer die Applikation startet, sieht er als erstes diese Activity. Diese enthält alle bereits erstellten TENs in Kachelform mit einigen Infos. Die Kacheln sind in einer oder mehreren Spalten, je nach Größe des Displays angeordnet. Diese Kacheln sind scrollbar.

Am unteren Bildschirmrand hat der Nutzer die Möglichkeit, sich nur die Kacheln, die zu einer der verschiedenen TEN-Arten gehören, anzeigen zu lassen. Dazu gibt es vier Buttons, die die jeweiligen TEN-Arten und eine generelle Übersicht über alle TENs darstellen.

Am oberen Bildschirmrand kann der Nutzer die einzelnen TEN-Arten neu erstellen, oder eine Suche öffnen. Auch hier sind wieder Buttons zu finden.

Der Nutzer hat die Möglichkeit, über ein langes gedrückt halten einer Kachel, in den Löschmodus zu gelangen. Hier können dann multiple Kacheln markiert werden und über ein neues Menü am oberen Bildschirmrand gelöscht werden.

Weitere Infos und Screenshots zum Layout der Applikation sind im nachfolgenden Kapitel zu finden.

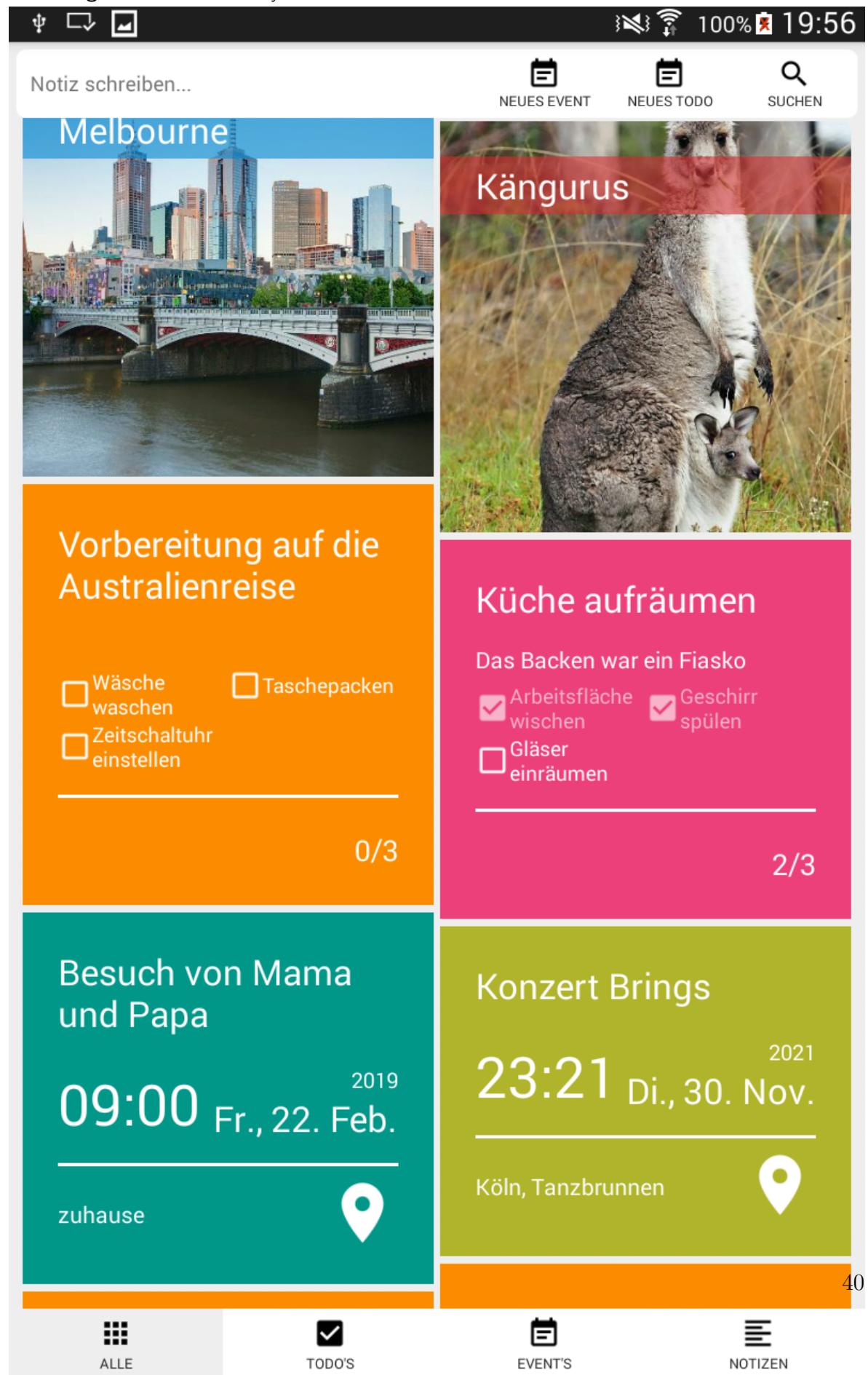
Da die Activity als Einstiegspunkt in die Applikation genutzt wird, müssen von hier aus alle weiteren Activities zu erreichen sein. Die weiteren Activities umfassen die Anzeige und Neuerstellung von TENs. Die Erreichung dieser geschieht über die vorhin genannten Buttons und über die Kacheln.

Die Activity selbst besteht aus einer Oberfläche mit den genannten Buttons. Die einzelnen TENs werden mithilfe von Fragments eingefügt. Für jede TEN-Art gibt es ein eigenes Fragment. Diese Fragments bestehen aus mehreren Klassen. Der genaue Aufbau dieser wird später erläutert. Um mit den Fragments arbeiten zu können, verfügt die Activity über verschiedene Helferklassen, um diesen Prozess zu strukturieren.

### **Layouts (Fabia Schmid)**

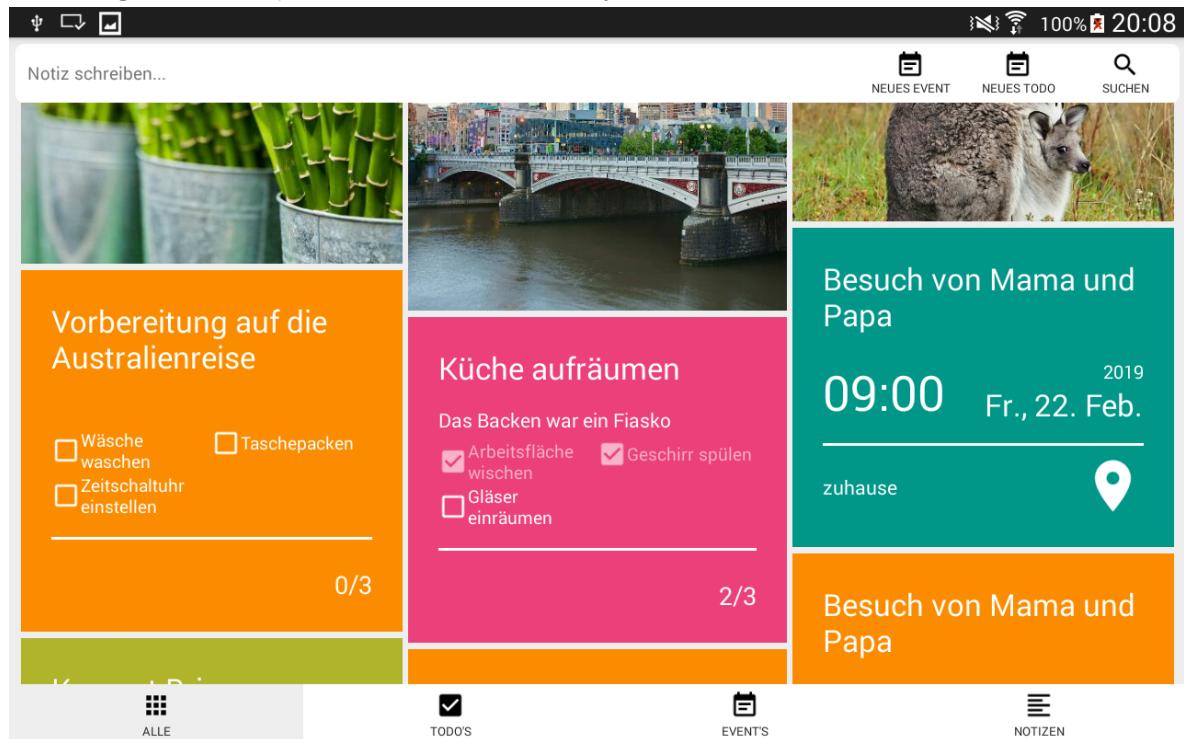
Das Layout der Overview Activity orientiert sich an dem vorher erstellten Mockup und besteht aus drei Grundkomponenten. Es gibt am oberen Bildschirmrand eine Leiste zum Erstellen von Notes, Events und ToDos, sowie ein Button zum Suchen. Unter dieser Leiste befindet sich ein Container, welcher die verschiedenen Fragments der vorhandenen Notes, Events und ToDos beinhaltet. Am unteren Bildschirmrand befindet sich noch eine Leiste mit vier Buttons zum Filtern der Fragments.

Abbildung 12: Overview Activity



Die Activity Overview zeigt normal 2 Spalten mit Fragments, wenn jedoch das Tablet gedreht wird zeigt die Landscape-Ansicht drei Spalten mit Fragments, um den Bildschirm optimal zu nutzen.

**Abbildung 13:** Landscape Ansicht - Overview Activity

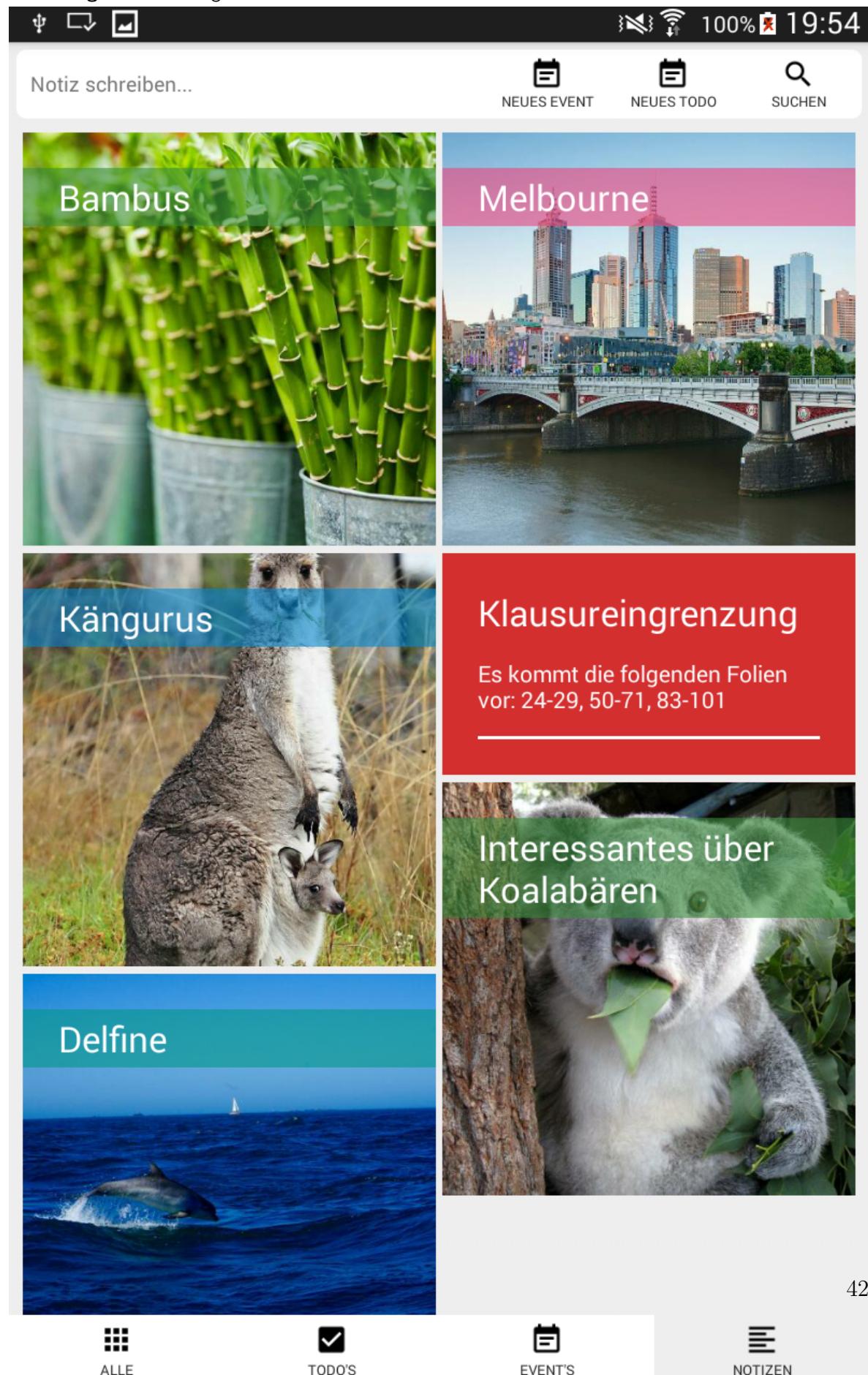


**Quelle:** Screenshot aus der Benutzeroberfläche

Die drei verschiedenen Arten von Fragments sind Notes, ToDo und Event.

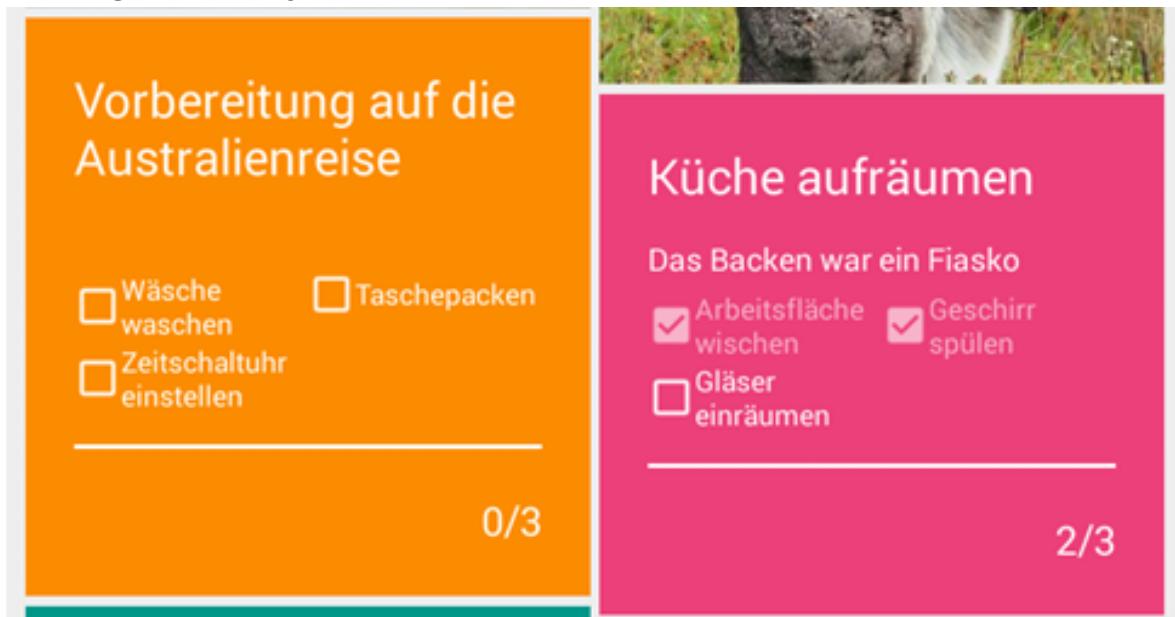
Bei einer Note wird die Überschrift und ein Textauszug angezeigt. Wenn jedoch ein Bild in der Notiz vorhanden ist wird dieses mit der Überschrift in der Overview Activity angezeigt.

Abbildung 14: Note Fragments



Die Fragments für die ToDos, zeugen auch den Titel an und die einzelnen Aufgaben mit Kästchen, die mit einem Hacken gefüllt sind, wenn sie erledigt sind. Zusätzlich wird Angezeigt, wie viele Aufgaben schon erfüllt wurden.

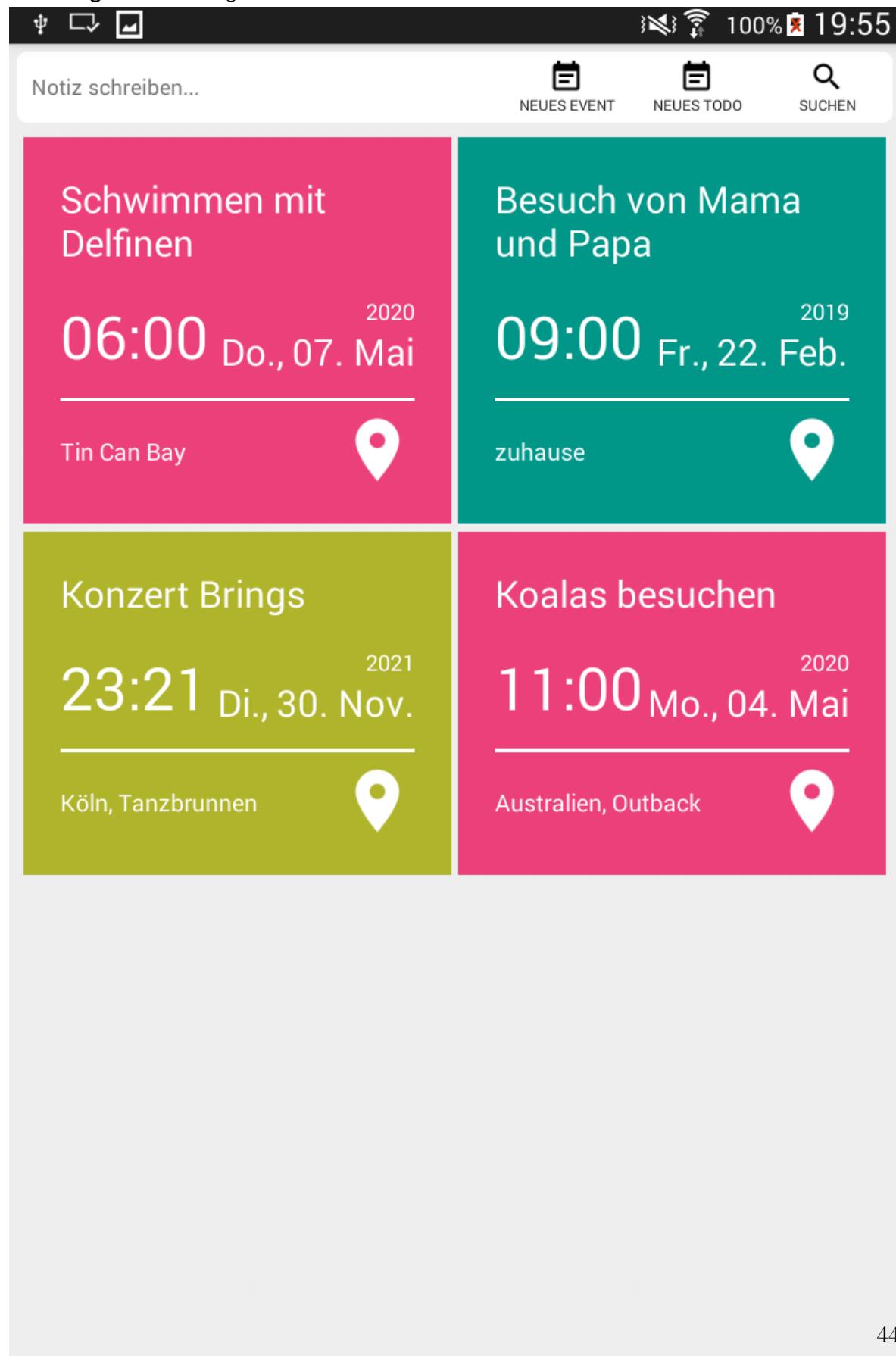
**Abbildung 15:** Todo Fragments



**Quelle:** Screenshot aus der Benutzeroberfläche

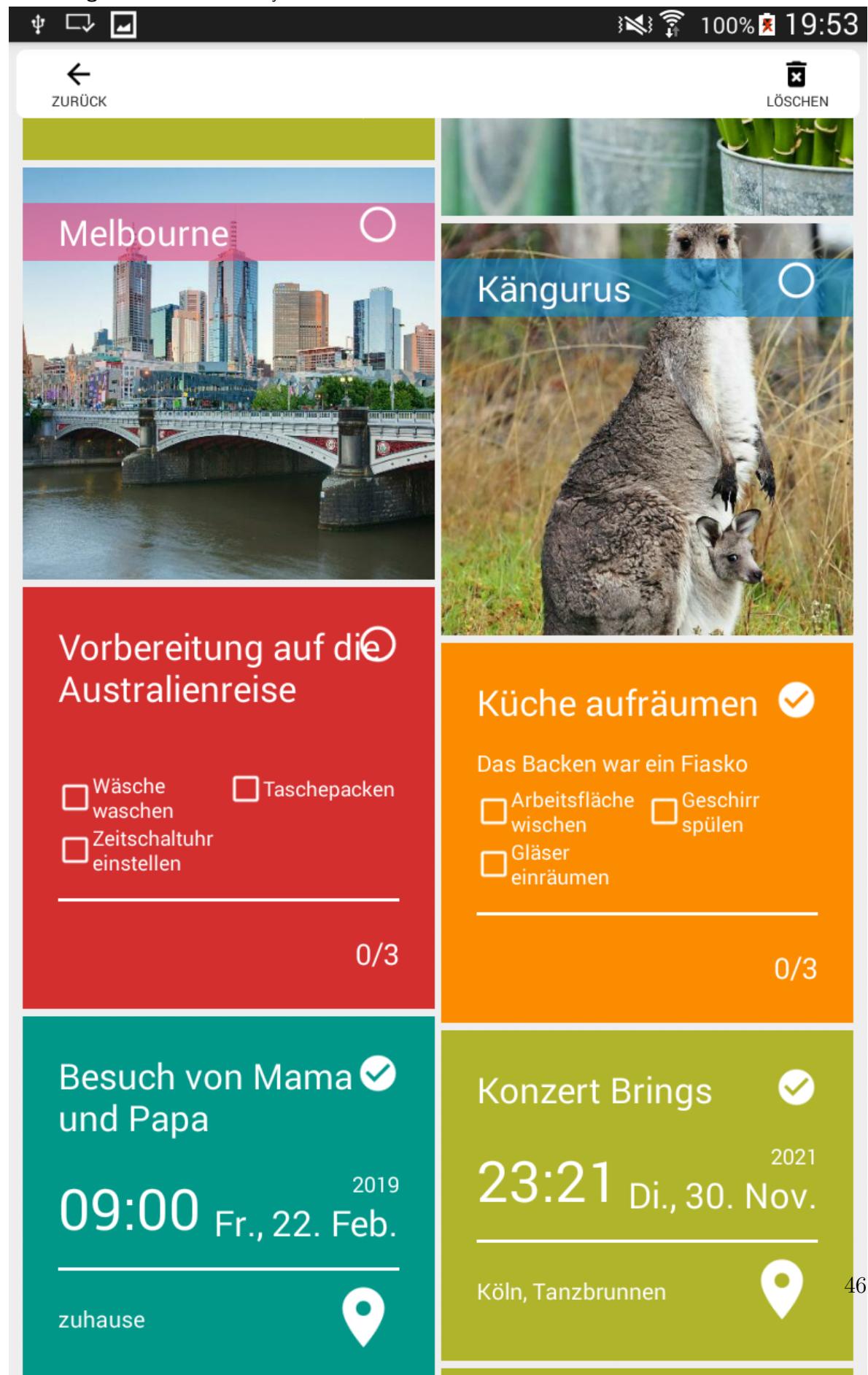
Die Event Fragmentes bestehen aus der Überschrift, dem Datum und der Uhrzeit, sowie aus einem Ort, der angegeben werden kann.

Abbildung 16: Event Fragments



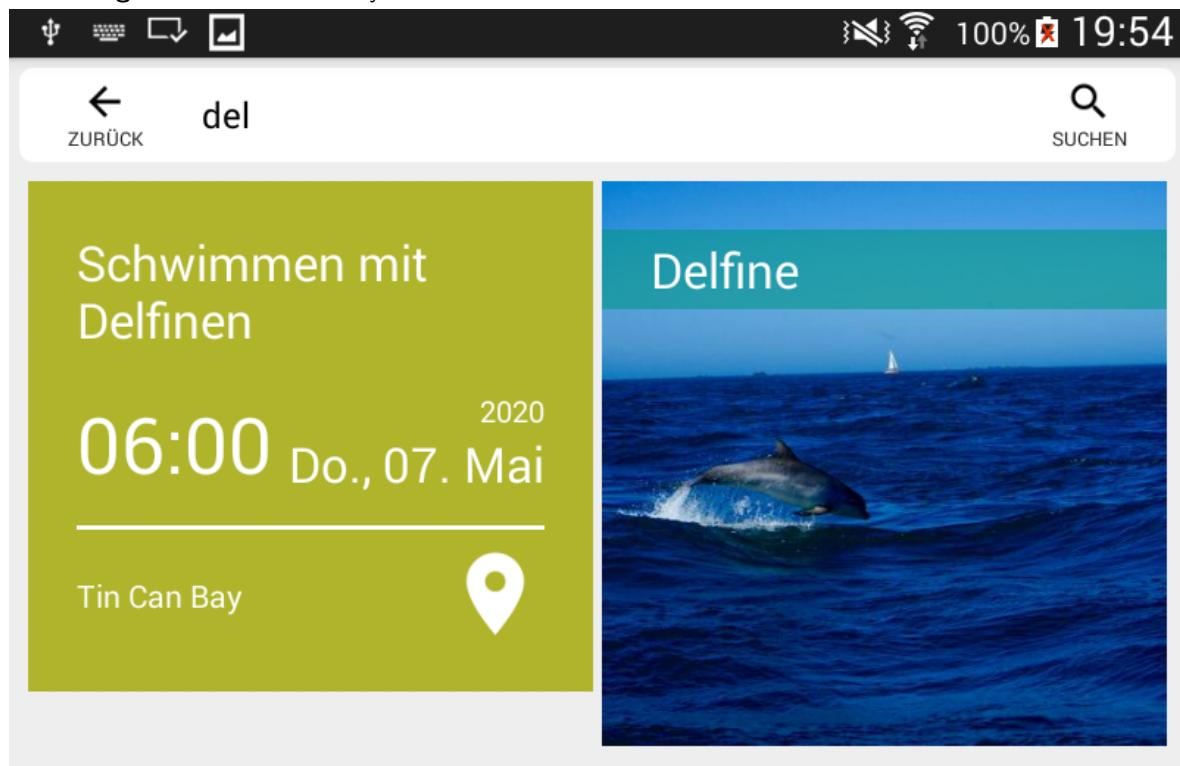
In der Overview Activity kann zusätzlich gesucht und gelöscht werden. Wenn man einen langen Click auf ein Fragment macht, wird die obere Leiste verändert und man bekommt einen Button zum Löschen. Zusätzlich kann in dieser Ansicht nun auf die verschiedenen Fragments geklickt werden, um diese zu markieren und mehrere gleichzeitig zu löschen. Diese Ansicht kann durch den Zurück-Pfeil verlassen werden.

Abbildung 17: Overview Activity - Löschen



Auch bei der Suchfunktion wird die obere Leiste angepasst. Dabei kann nun ein beliebiges Wort eingegeben werden, welches dann in den verschiedenen Einträgen gesucht wird. Die gefundenen Einträge werden daraufhin angezeigt. Um die Suche zu beenden kann der Zurück-Pfeil verwendet werden.

Abbildung 18: Overview Activity - Suchen



### **Use Case (Yannick Rüttgers)**

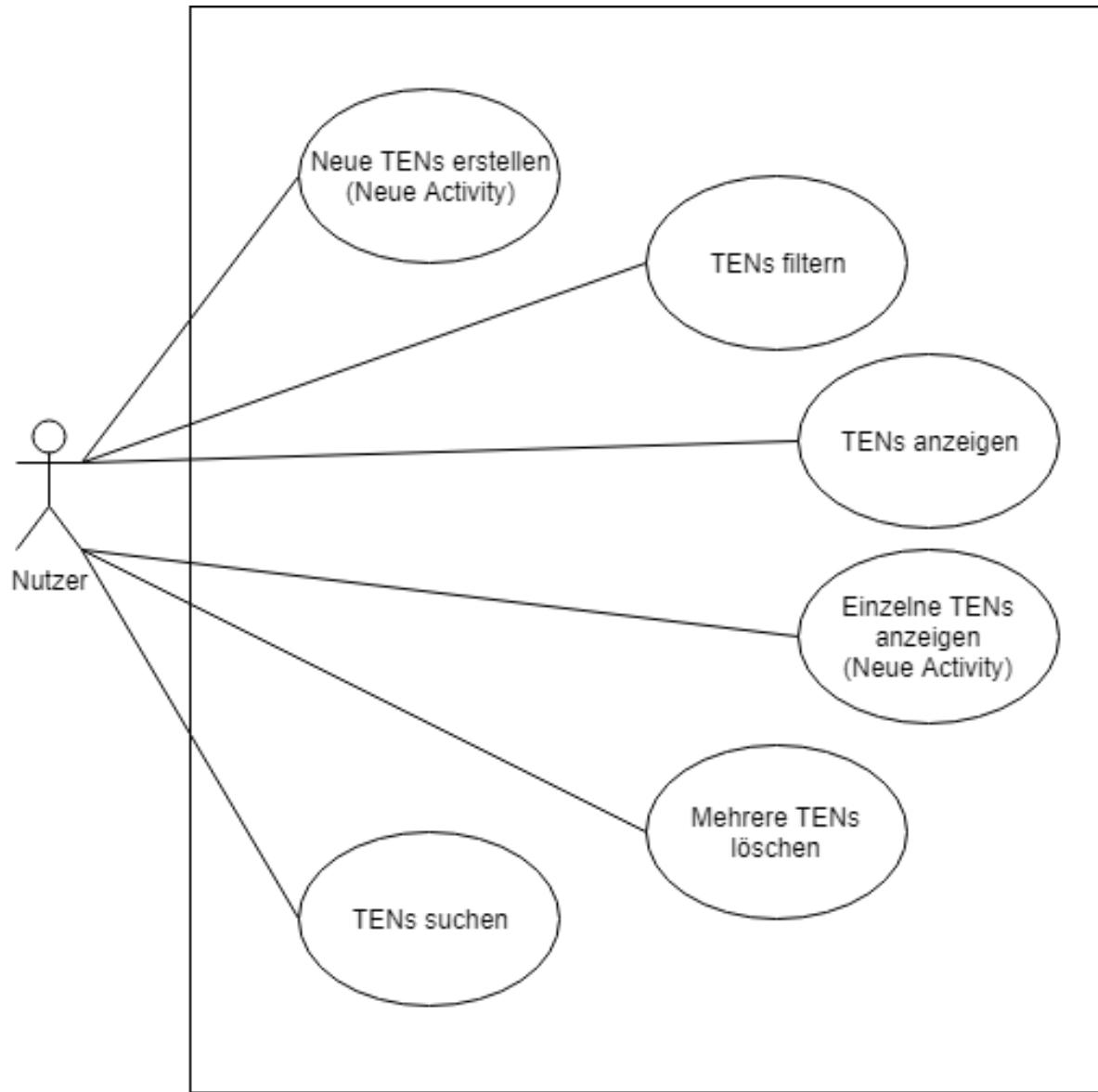
Wie bereits geschrieben, dient diese Activity dem Nutzer als Einstiegspunkt in die Applikation. Hier kann er einige Aktionen durchführen.

Zum einen kann er die Oberfläche der Übersicht selbst beeinflussen. Hierzu gibt es am unteren Bildschirmrand einige Buttons, mit denen er nach den verschiedenen TEN-Arten filtern kann. Zusätzlich kann er über eine Suchfunktion alle TENs durchsuchen.

Aus dieser Activity heraus können vom User auch neue TENs erstellt werden. Hierzu wird er allerdings an weitere Activities weitergeleitet.

Zuletzt ist es dem Nutzer auch möglich, mehrere TENs auf einmal zu löschen. Dies geschieht, wie bereits gesagt, über ein markieren einzelner Kacheln.

Im Folgenden ist das Usecase-Diagramm für die Übersicht zu sehen.

**Abbildung 19:** Usecase Diagramm der Overview

**Quelle:** Erstellt von Yannick Rüttgers

### Datenstruktur und Typen (Yannick Rüttgers)

Die Daten für diese Activity werden hauptsächlich über die Klasse OverviewData verwaltet.

Wenn die Activity initiiert wird, werden zuerst alle nötigen Instanzen erstellt. Hierzu gehört unter anderen eine Instanz der Klasse OverviewData. Diese Klasse ruft in ihrem Konstruktor mithilfe einer statischen Servicemethode alle TENs als Objekte ab. Diese

Objekte werden in einer ArrayList gespeichert. Bei Bedarf werden diese Daten neu geladen.

Um beim Suchen oder Filtern nicht alle Daten neu laden zu müssen, gibt es eine weitere ArrayList, die immer den gefilterten Datenstand enthält. Aus diesen Daten werden die Fragments generiert.

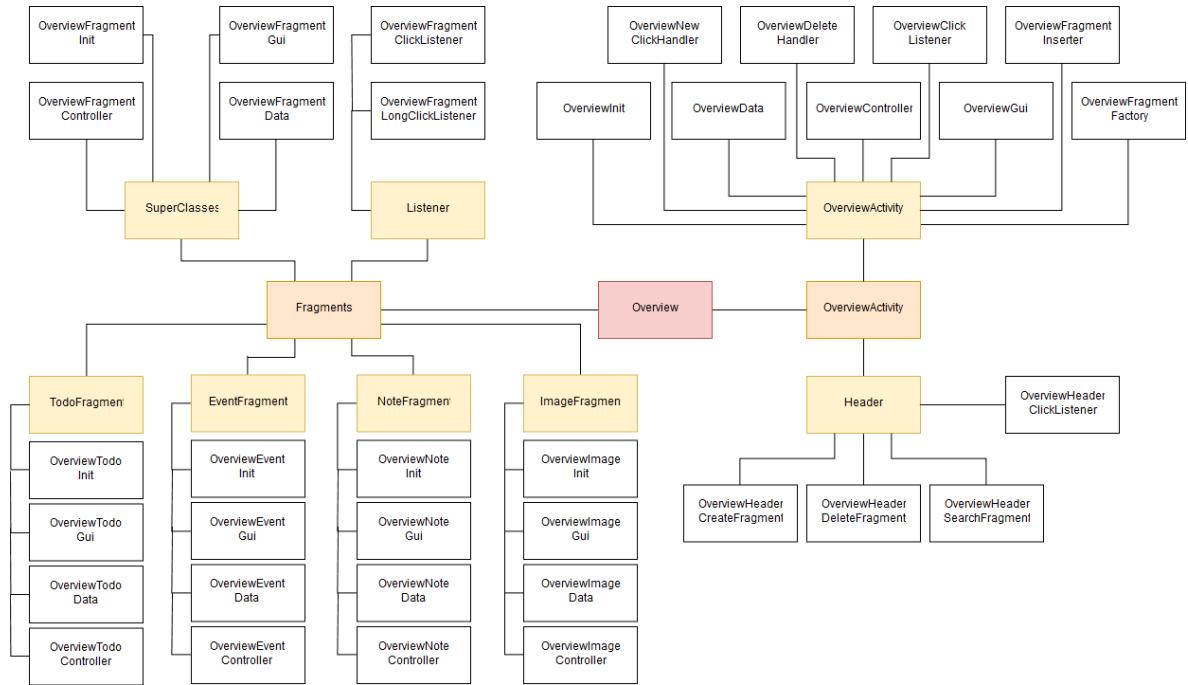
Um die Filterung von Fragments beim Wechsel der Orientierung der Applikation beizubehalten zu können, wird dies ebenfalls im Dataobjekt hinterlegt.

Wenn Fragments generiert werden, bekommen diese die Daten des zugehörigen TEN-Objekts als Bundle mitgegeben. Während der Initialisierung jedes Fragments, werden die Daten im jeweiligen Data Objekt abgelegt. Die Parameter dieser Dataobjekte bilden zum größten Teil die der TEN-Objekte ab. Im Konstruktor der Dataklasse werden den einzelnen Parametern dann die Werte aus dem Bundle zugewiesen. Da in der Activity die TENs nicht bearbeitet werden können, gibt es im Nachhinein keine Möglichkeit den Fragments neue Daten zuzuweisen.

Zusätzlich hält jedes Fragment zwei Informationen über sich. Dies ist zum einen der Löschzustand, über den das Fragment steuern kann, ob es zum Löschen markiert werden kann oder nicht. Zum anderen hält das Fragment die Information, ob es markiert ist, um dies im Falle einer Löschung angeben zu können. Diese beiden Informationen können aus dem Fragment abgerufen werden.

### **Dokumentation des Quelltextes der Activity (Yannick Rüttgers)**

Allgemein Die OverviewActivity besteht aus 35 verschiedenen Klassen. Diese Klassen sind in verschiedene Pakete unterteilt. Zum einen gibt es die Klassen, die unmittelbar zur OverviewActivity an sich gehören. Zu diesen zugehörig sind die Klassen, die für den Header der Activity genutzt werden. Alle weiteren Klassen werden für die Fragments, die die TENs darstellen, genutzt. Hierzu gibt es vier Superklassen und zwei Listener, die in allen Fragments genutzt werden. Jedes der vier verschiedenen Fragments hat nochmal vier weitere Klassen, die von den Superklassen erben. In Folgender Grafik soll dies verdeutlicht werden.

**Abbildung 20:** Klassendiagramm

**Quelle:** Erstellt von Yannick Rüttgers

Die Funktion der Klassen und wichtige Methoden werden im Folgenden erläutert. Activityklassen Zu den Klassen, die unmittelbar zur Activity gehören, zählen neun verschiedene Klassen. Klasse: class OverviewInit extends AppCompatActivity Diese Klasse ist für die Initialisierung der verschiedenen Komponenten der Activity zuständig. Da sie von der Superklasse AppCompatActivity erbt, implementiert sie zusätzlich die Methoden des Activitylifecycles. Wichtige Methoden: -protected void OnCreate(Bundle savedInstanceState) Diese Methode ist eine der Lifecyclemethoden von Activities, sie wird beim Entstehen der Activity aufgerufen. Hier werden die verschiedenen Komponenten der Activity mit den zugehörigen Methoden initialisiert. -public void onConfigurationChanged(Configuration newConfig) Diese Methode wird aufgerufen, wenn sich etwas an der Konfiguration der Applikation ändert, wie in diesem Fall die Ausrichtung des Gerätes. Hier werden der Controller und die Gui neu initialisiert. Dazu werden die Methoden initGui() und initController() aufgerufen. Klasse: class OverviewData Diese Klasse ist für die Datenhaltung innerhalb der Activity verantwortlich. Hier befinden sich auch die zum Sortieren und zum Suchen verwendeten Methoden. Initial werden die Daten einmal geladen, bei Bedarf werden diese über eine Methode neu geladen. Die Daten werden dabei zweimal gehalten: Eine Liste enthält alle TENs, die andere Liste enthält die TENs, die

angezeigt werden sollen. Wichtige Methoden: -public void refresh() Lädt die Daten neu. -public ArrayList<TEN> filterData() Filtert die Daten nach dem aktuell ausgewählten TEN-Typen. -public void search(String pSearchString) Fügt nur die Suchergebnisse in die Liste der TENs ein, die angezeigt werden sollen. Klasse: class OverviewGui Diese Klasse verwaltet die Benutzeroberfläche der Activity. Wichtige Methoden: -public void markButton(Class pClass) Markiert einen der Buttons am unteren Bildschirmrand, je nachdem welcher TEN-Typ ausgewählt ist. -public void showFooter() / hideFooter() Zeigt oder versteckt die Buttons am unteren Bildschirmrand. Klasse: class OverviewController Diese Klasse verwaltet die einzelnen Helferklassen und stellt die Logik der Activity da. Wichtige Methoden: In der Klasse wird die Suche aktiviert, sowie das Löschen von TENs verwaltet. Hierzu werden die Helferklassen OverviewDeleteHandler, OverviewNewClickHandler, OverviewFragmentFactory und OverviewFragmentInserter genutzt. Klasse: class OverviewFragmentFactory Diese Klasse erstellt verschiedene Arten von Fragments, die dann später genutzt werden können. Wichtiger Methoden: -public Fragment createHeader(Create/Delete/Search)Fragment() Diese Methode liefert das gewählte Headerfragment zurück. -public ArrayList<Fragment> createTENFragments(ArrayList<TEN> pTENs) Erstellt eine Liste von Fragments, die aus den TENs erstellt werden. Je nach Art des TENs wird ein unterschiedliches Fragment erstellt. Klasse: class OverviewFragmentInserter Diese Klasse fügt Fragments in die Benutzeroberfläche ein. Hierzu wird der FragmentManager der Activity genutzt. Wichtige Methoden: -public void insertFragment(int pContainerID, Fragment pFragment, String pTag) Fügt ein Fragment in einen gewählten Container ein. Dem Fragment wird ein Tag zugewiesen. -public void replaceFragment(int pContainerID, Fragment pFragment, String pTag) Ersetzt ein Fragment in einem gewählten Container durch ein anderes. Dem Fragment wird ein Tag zugewiesen. -public void insertFragments(int[] pContainerIDs, ArrayList<Fragment> pFragments) Fügt eine beliebige Anzahl Fragments in eine beliebige Anzahl Container an. Dabei werden die Container zyklisch durchlaufen. Klasse: class OverviewClickListener Diese Klasse verwaltet die Klickevents der OverviewActivity. Je nach geklicktem Button wird eine andere Methode des Controllers aufgerufen. Wichtige Methoden: -public void onClick(View view) Ruft je nach angeklickter View die show()-Methode des Controllers mit einem bestimmten Parameter auf. Klasse: class OverviewDeleteHandler Diese Klasse verwaltet die Methoden, die zum Löschen beziehungsweise nicht Löschen von TENs nötig sind. Wichtige Methoden: -public void longClick() Aktiviert den Löschvorgang für alle Fragments. -public void back() Deaktiviert den Löschvorgang für alle Fragments. -public void delete() Sam-

melt alle markierten Fragments und löscht die dazugehörigen TENs. Deaktiviert den Löschvorgang für alle anderen Fragments. Klasse: class OverviewNewClickHandler Verwaltet das Erstellen von neuen TENs. Wichtige Methoden: -public void newTEN(Class pClass) Startet eine neue Activity, je nachdem welche TEN-Art angegeben wurde. Header Die Header der Activity sind wechselnde Fragments, die verschiedene Aufgaben haben. Sie werden am oberen Bildschirmrand ausgetauscht. Ein Fragment dient dem Neuerstellen von TENs, eines das Löschen und eines das Suchen. Die Fragments teilen sich einen ClickListener. Klasse: class OverviewHeaderCreateFragment Dieses Fragment dient dazu, das Erstellen von neuen TENs anzustoßen. Hierzu enthält es drei Buttons. Zusätzlich gibt es einen Button zum Starten der Suche. Klasse: class OverviewHeaderDeleteFragment Dieses Fragment dient dem Löschvorgang. Es gibt einen Button zum Abbrechen des Prozesses und einen Button, um die Löschung durchzuführen. Klasse: class OverviewHeaderSearchFragment Dieses Fragment dient dem Suchvorgang. Es gibt ein Textfeld, in das ein Suchbegriff eingegeben werden kann. Außerdem gibt es einen Button zum Abbrechen des Prozesses und einen, um die Suche durchzuführen. Klasse: class OverviewHeaderClickListener Diese Klasse behandelt die Clickevents der Headerfragments. Je nachdem, welcher Button in einem der Fragments geklickt wurde, wird eine andere Methode im Controller aufgerufen. Wichtige Methoden: -public void onClick(View view) Diese Methode wird ausgeführt, sobald einer der Buttons eines Headers geklickt wird. Je nachdem, welcher Button dies war, wird eine Methode im Controller aufgerufen. Superklassen Die folgenden Klassen dienen als Superklassen für die Fragments. Dies wurde so entwickelt, da alle vier Fragments sich ähnliche Funktionen teilen. Große Teile der Implementierung nutzen außerdem die Polymorphie. Klasse: class OverviewFragmentInit Diese Klasse dient als Superklasse für die Initialklassen der einzelnen Fragments. Hier werden die anderen drei Komponenten der Fragments initialisiert. Außerdem werden zum Start des Fragments mehrere andere Methoden aufgerufen. Wichtige Methoden: -public void onCreateView(Bundle pArguments, View pView) Diese Methode startet, wenn das Fragment erstellt wird, verschiedene Methoden des Controllers. Klasse: class OverviewFragmentData Diese Klasse dient als Superklasse für die Dataklassen der einzelnen Fragments. Sie bildet die Klasse TEN nach. Wichtige Methoden: -public void addData(Bundle pData) Diese Methode pflegt die Daten aus einem Bundle in das Objekt ein. Klasse: class OverviewFragmentGui Diese Klasse dient als Superklasse für die Guiklassen der einzelnen Fragments. Sie kümmert sich um den Markierungsprozess der Fragments für den Löschvorgang. Wichtige Methoden: -public void applyMarked(boolean pMarked) Diese Methode setzt den Zustand des Checkbo-

xicons auf den des übergebenen Zustandes. Klasse: class OverviewFragmentController Diese Klasse dient als Superklasse für die Controllerklassen der einzelnen Fragments. Hauptsächlich wird sich auch hier um den Löschprozess gekümmert, da dieser für alle Fragments gleich ist. Wichtige Methoden: -public void longClicked() Diese Methode startet den Löschprozess. Dazu wird der Controller der Activity, die das Fragment verwaltet aufgerufen, und ihm mitgeteilt, dass der Löschprozess gestartet werden soll. Listener Klasse: class OverviewFragmentClickListener Diese Klasse ist dafür zuständig, dass, wenn ein Fragment angeklickt wird, im Controller dieses Fragments eine Methode ausgeführt wird. So wird eines der Fragments geöffnet. Klasse: class OverviewFragmentLongclickListenerFragment Diese Klasse ist dafür zuständig, dass, wenn ein Fragment lange angeklickt wird, die entsprechende Methode im Controller des Fragments aufgerufen wird. So gelangt der Nutzer in den Löschvorgang. Fragments Die folgenden Kapitel behandeln die vier genutzten Fragments. Alle Fragments erben von den vier zuvor genannten Superklassen, und teilen sich so eine Struktur. Zudem wird, um den Quellcode übersichtlich zu halten, viel Wert auf die Nutzung von Polymorphie gelegt. TodoFragment Die folgenden Klassen implementieren das Fragment, welches die Todos abbildet. Klasse: class OverviewTodoInit Diese Klasse initiiert alle nötigen Komponenten für das Todo-Fragment. Klasse: class OverviewTodoData Diese Klasse stellt die Datenhaltung für das Todo-Fragment dar. Sie bildet die Klasse Todo nach. Wichtige Methoden: - public void addData(Bundle pData) Diese Methode pflegt die Daten aus einem Bundle in das Objekt ein. Klasse: class OverviewTodoGui Diese Klasse verwaltet die GUI für das Todo-Fragment. Hier geschieht auch die Generierung der Checkboxen aus den Todo-Tasks. Wichtige Methoden: -public void addView(View pView) Diese Methode fügt die View des Fragments dem Objekt hinzu. Außerdem werden den Attributen Views zugewiesen. -public void addCheckbox(boolean pStatus, String pDescription) Fügt dem Fragment eine Checkbox hinzu, die aus einem Kästchen und einer Beschreibung besteht. Klasse: class OverviewTodoController Diese Klasse stellt die Logik des Todofragments dar. Hier werden den Feldern der Benutzeroberfläche Werte zugewiesen. Zusätzlich werden aus den Tasks des Todos Checkboxen generiert. -public void applyData() Diese Methode übergibt die Attribute des Dataobjekts an das Guiobjekt, damit die Views aktualisiert werden können. -public void generateCheckboxes() Diese Methode generiert aus der Taskliste des Todos mithilfe der addCheckbox-Methode des Guiobjekts Checkboxen. EventFragment Die folgenden Klassen implementieren das Fragment, welches die Events abbildet. Klasse: class OverviewEventInit Diese Klasse initiiert alle nötigen Komponenten für das Event-Fragment. Klasse: class OverviewEventData Diese Klasse

stellt die Datenhaltung für das Event-Fragment dar. Sie bildet die Klasse Event nach. Hier wird das Kalenderobjekt in seine Einzelteile zerlegt. Wichtige Methoden: -public void addData(Bundle pData) Diese Methode pflegt die Daten aus einem Bundle in das Objekt ein. Klasse: class OverviewEventGui Diese Klasse verwaltet die GUI für das Event-Fragment. Wichtige Methoden: -public void addView(View pView) Diese Methode fügt die View des Fragments dem Objekt hinzu. Außerdem werden den Attributten Views zugewiesen. Klasse: class OverviewEventController Diese Klasse stellt die Logik des Eventfragments dar. Hier werden den Feldern der Benutzeroberfläche Werte zugewiesen. -public void applyData() Diese Methode übergibt die Attribute des Dataobjekts an das Guiobjekt, damit die Views aktualisiert werden können. NoteFragment Die folgenden Klassen implementieren das Fragment, welches die Notes abbildet, die keine Bilder haben. Klasse: class OverviewNoteInit Diese Klasse initiiert alle nötigen Komponenten für das Note-Fragment. Klasse: class OverviewNoteData Diese Klasse stellt die Datenhaltung für das Note-Fragment dar. Sie bildet die Klasse Note nach. Wichtige Methoden: -public void addData(Bundle pData) Diese Methode pflegt die Daten aus einem Bundle in das Objekt ein. Klasse: class OverviewNoteGui Diese Klasse verwaltet die GUI für das Note-Fragment. Wichtige Methoden: -public void addView(View pView) Diese Methode fügt die View des Fragments dem Objekt hinzu. Außerdem werden den Attributten Views zugewiesen. Klasse: class OverviewNoteController Diese Klasse stellt die Logik des Note-Fragments dar. Hier werden den Feldern der Benutzeroberfläche Werte zugewiesen. -public void applyData() Diese Methode übergibt die Attribute des Dataobjekts an das Guiobjekt, damit die Views aktualisiert werden können. ImageFragment Die folgenden Klassen implementieren das Fragment, welches die Notes abbildet, in denen Bilder gespeichert wurden. Klasse: class OverviewImageInit Diese Klasse initiiert alle nötigen Komponenten für das Image-Fragment. Klasse: class OverviewImageData Diese Klasse stellt die Datenhaltung für das Image-Fragment dar. Hier wird das Previewimage für die Notiz geladen. Wichtige Methoden: -public void addData(Bundle pData) Diese Methode pflegt die Daten aus einem Bundle in das Objekt ein und läd das Previewimage. Klasse: class OverviewImageGui Diese Klasse verwaltet die GUI für das Image-Fragment. Wichtige Methoden: -public void addView(View pView) Diese Methode fügt die View des Fragments dem Objekt hinzu. Außerdem werden den Attributten Views zugewiesen. Klasse: class OverviewImageController Diese Klasse stellt die Logik des Image-Fragments dar. Hier werden den Feldern der Benutzeroberfläche Werte zugewiesen. -public void applyData() Diese Methode übergibt die Attribute des Dataobjekts an das Guiobjekt, damit die Views aktualisiert werden können.

### 5.2.2 Todo Activity

#### Aufgabe und Funktionen (Florian Rath)

Die Activity Todo dient dem Nutzer dazu, seine Aufgaben zu organisieren. Wenn er eine neue Todo erstellt hat, hat er die Möglichkeit einen Titel und eine Beschreibung einzugeben. Diese Eingabemöglichkeiten sind jedoch optional und hindern den Nutzer nicht daran, die Todo wieder zu verlassen und somit abzuspeichern. Der Titel für eine Todo könnte beispielsweise „Einkaufsliste“ lauten, während die Beschreibung nähere Informationen wie zum Beispiel „Für die Geburtstagsparty von meiner Tochter“ beinhalten kann. So hat der Benutzer die Möglichkeit mehrere Einkaufslisten anzulegen und sie mit einer Beschreibung voneinander differenzieren.

Neben dem Titel und der Beschreibung können auch ein Start- und Enddatum festgelegt werden. Dies ist hilfreich, wenn für eine Todo bestimmte Fristen vorhanden sind. Geht es in der Todo z.B. um eine Prüfungsvorbereitung, kann der Benutzer auswählen, wann er spätestens mit dem Lernen anfangen muss und bis wann er spätestens mit dem Lernen Zeit hat. Diese Funktionalität ist ebenfalls optional. Man kann also auch Todos verwenden, ohne ein bestimmtes Start- und Enddatum anzugeben. Die App zeigt dann immer jeweils das aktuelle Datum in den Eingabefeldern an.

Die Hauptfunktionalität der Todo-Activity ist das Anlegen von sogenannten Tasks.- Eine Task besteht im Prinzip aus einem Text, wie z.B. „Luftballons“, und einem Status (boolean-Wert). Der Status gibt an, ob ein Task erledigt ist oder nicht. Dies geschieht über eine Checkbox, die neben jedem Task vorhanden ist.

Unterstrichen wird die Gesamtfunktionalität mit einer Fortschrittsanzeige. Ein prozentualer Wert gibt dabei an, wie viele Tasks als erledigt markiert sind. Wurden z.B. 5 von 10 Tasks als erledigt markiert, steht in der App 50 Prozent. Sind es hingegen 1 von 10 markierte, dann nur noch 10 Prozent. Diese Funktionalität dient dazu, dem Benutzer einen Fortschritt über seine Todo zu geben. Anhand des Fortschritts kann er sich nämlich organisieren, ob er in den genannten Fristen liegt. Liegt man kurz wenige Tage vor einer Prüfung gerade bei 10 Prozent, so könnte es langsam brenzlig für den Benutzer werden. Auch in einer sehr langen Liste von Tasks könnte dieser Wert nützlich sein. Da die Todo-Activity darauf ausgelegt ist, theoretisch endlos viele Tasks speichern zu können (so viele, wie ein ArrayList speichern kann), können sehr lange Listen entstehen. Unter hunderten von Tasks könnte es schnell passieren, eine unerledigte Aufgabe zu übersehen

und die Todo verfrüht abzuschließen. Die App macht keine Obergrenze für Tasks, um dem User keine Arbeits- bzw. Organisationsweise vorzuschreiben.

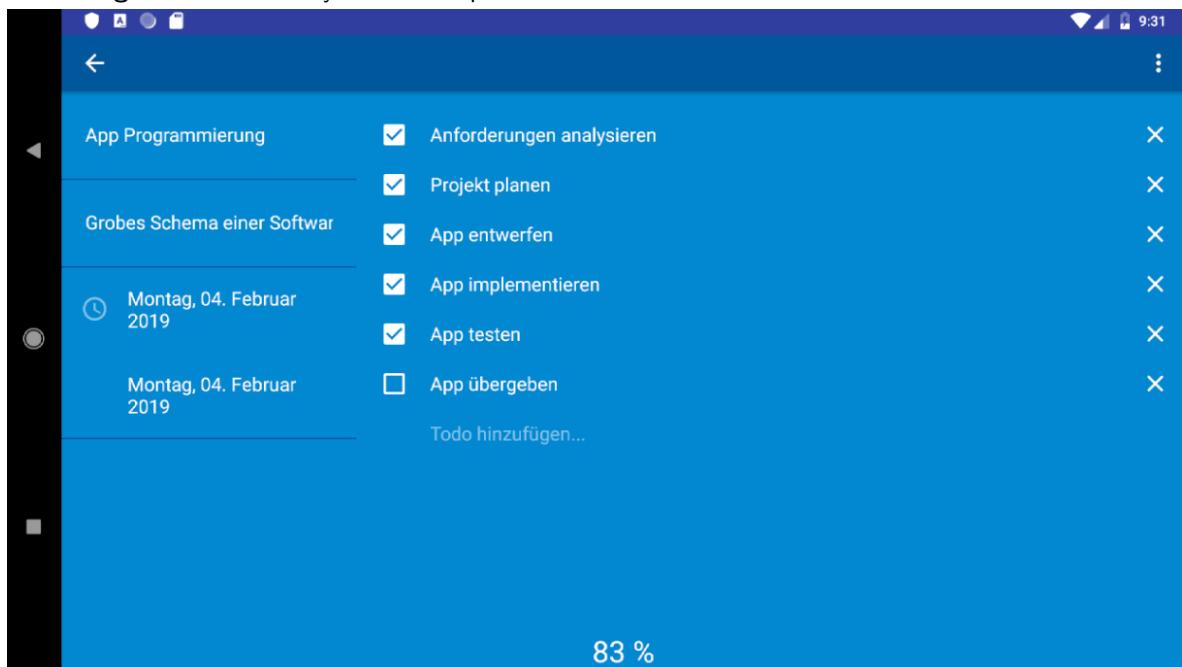
Der User hat auch die Möglichkeit eine Todo zu löschen.

### **Layout, Screenshots (Sertan Cetin)**

Um eine maximale Bedienbarkeit zu gewährleisten, verfügt diese Activity insgesamt genau über zwei Layouts. Während das eine Layout die Benutzersteuerelement im Porträtmodus darstellt, dient das andere Layout dazu, die Elemente im Landscape-Modus darzustellen. Der Unterschied zwischen den beiden Layouts liegt in der Anordnung der Elemente. Im Porträtmodus belegt jedes Element eine Zeile auf dem Bildschirm. Sie sind also untereinander angeordnet. Im Landscape-Modus ist die Ansicht in zwei Spalten geteilt. In der linken Spalte sind der Titel, Beschreibung, Start- und Enddatum untereinander angeordnet. In der rechten Spalte befinden sich die einzelnen Aufgaben. Die Breite der linken Spalte hat einen festen Wert, nämlich 640px. Die Prozentanzeige befindet sich unabhängig von den beiden Spalten immer mittig am unteren Bildschirmrand.

Oben ist eine Toolbar vorhanden, mit einem Zurück-Button, um zur Hauptübersicht zu gelangen, und einem Menü, in welchem die Todo gelöscht werden kann.

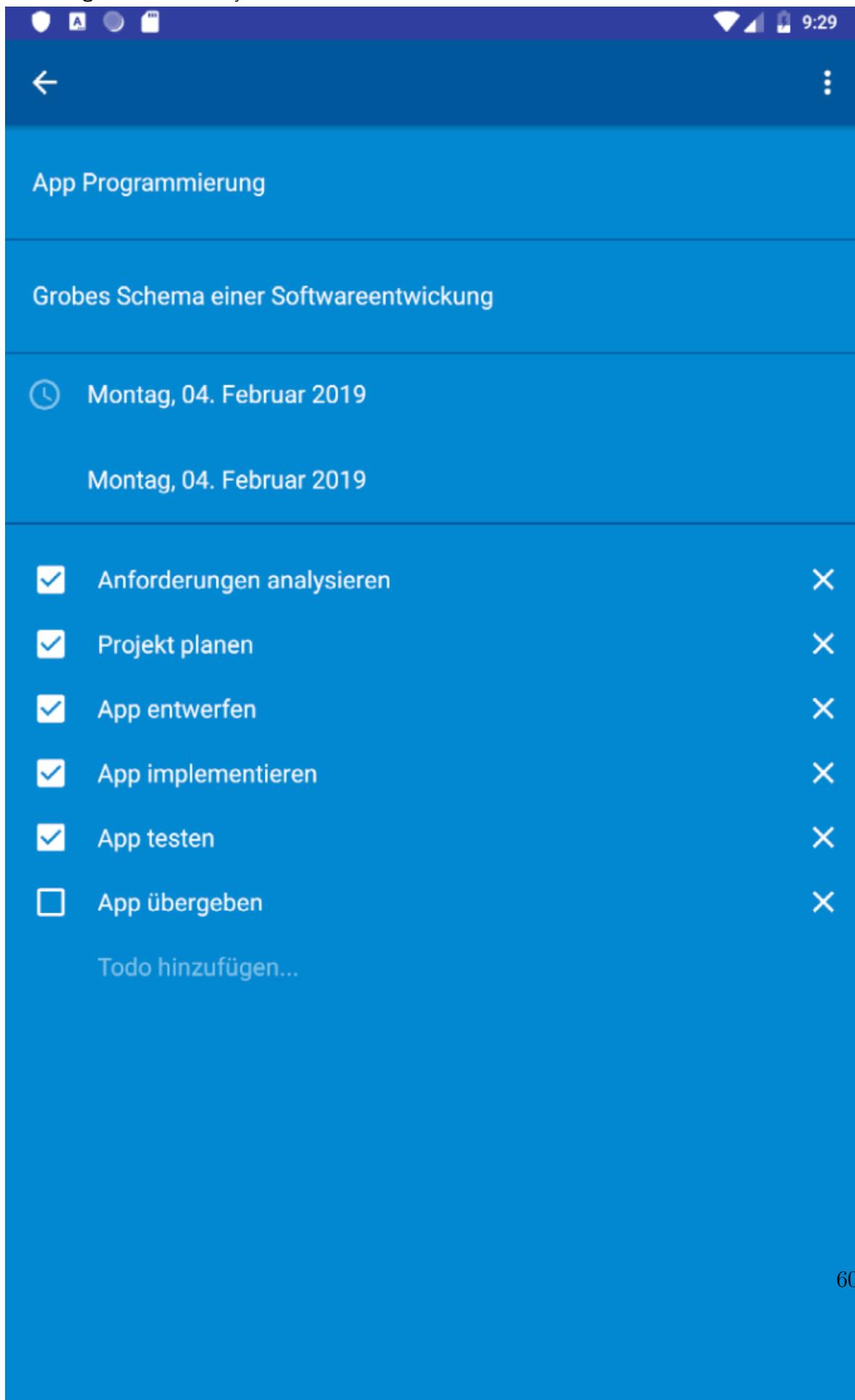
Nachfolgender Screenshot zeigt den zuvor beschriebenen Landscape-Modus der Todo-Activity:

**Abbildung 21:** Todo Activity im Landscape-Modus

**Quelle:** Screenshot aus der Benutzeroberfläche

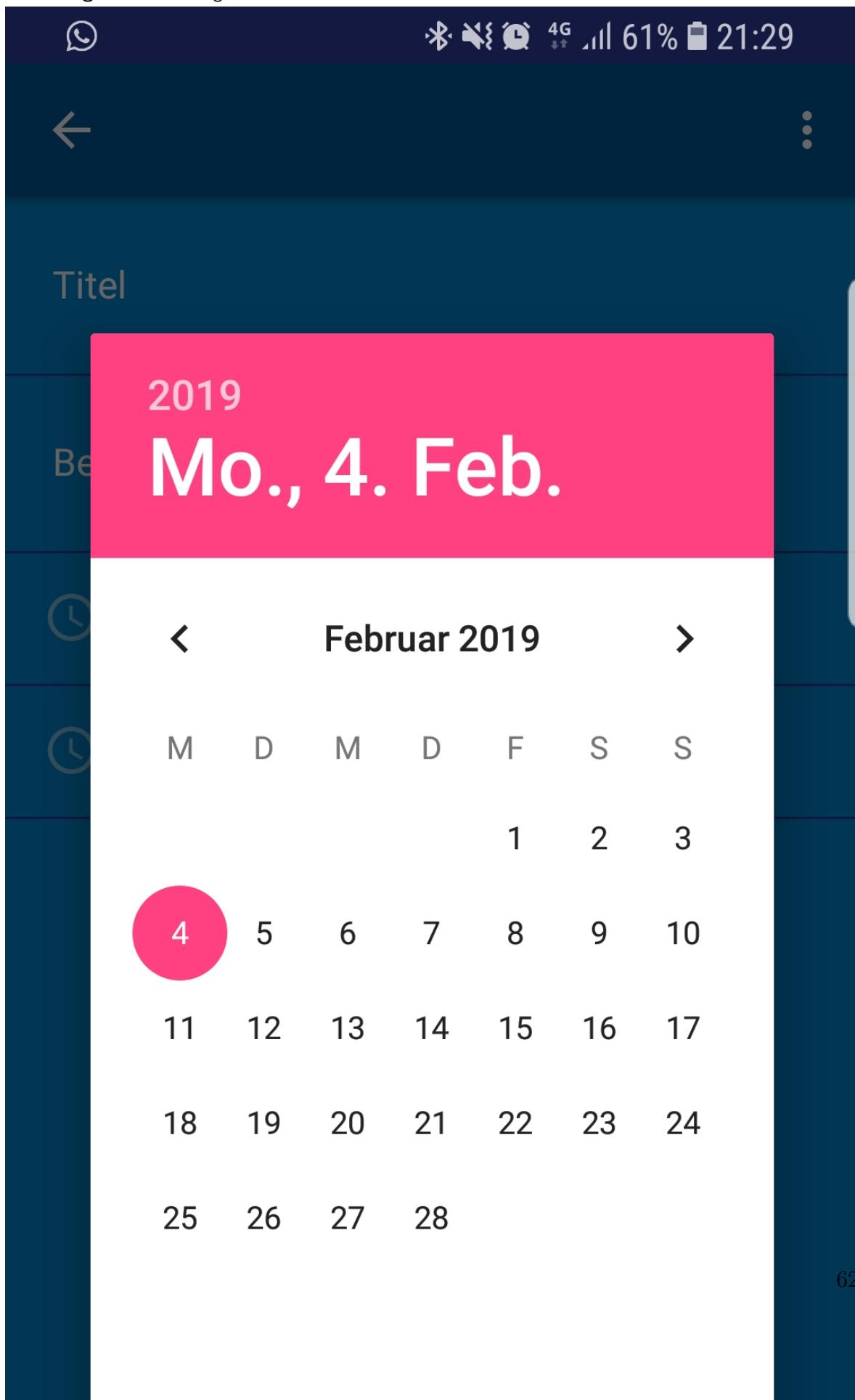
Beim Wechsel der beiden Ansicht-Modi werden die Daten jeweils in das andere Layout übernommen.

Abbildung 22: Todo Activity im Portrait-Modus



Nachdem auf eines der Eingabefelder für Daten getippt wurde, initialisiert die Activity einen Datepicker und zeigt dem Benutzer ein gewohntes Auswahlmenü für ein gewünschtes Datum an. Dies hat den Vorteil, dass der Benutzer durch Erfahrungen in anderen Apps schnell und einfach mit der Datumeingabe zurechtfindet. Wie im Screenshot zu sehen ist, kann der Benutzer mit „Abbrechen“ die Datumsauswahl abbrechen, sodass das Eingabefenster geschlossen wird. Mit dem „Ok“-Button hat der Benutzer alternativ die Möglichkeit, seine Eingabe zu übernehmen. Hierbei liest die Activity das eingegebene Datum aus und stellt es formatiert dar. Es wird außerdem indem Todo-Objekt zwischengespeichert.

Abbildung 23: Datumeingabe



## Use Case (Florian Rath)

Die Todo-Activity kann man über zwei Wege erreichen. Beide Wege erfolgen aus der Übersichts-Activity.

Der User kann entweder eine neue Todo erstellen, sodass in der Todo-Activity eine nicht vorhandene Todo-ID übergeben wird, oder er kann auf eine bereits vorhandene Todo tippen. Die ID dieser Todo wird dann aus der Datenbank geladen und in der Activity dargestellt. Die Todo-Activity deckt die Funktionalitäten des Todo-TEN's ab.

## Datenstrukturen und -typen (Sertan Cetin)

Um die gewünschten Anforderungen zu erfüllen und dabei sowohl eine hohe Wartbarkeit als auch Modularität zu gewährleisten, wurden innerhalb der Todo-Activity mehrere Klassen implementiert. Nachfolgend werden diese Klassen vorgestellt.

Wenn die Todo-Activity aus der Overview-Activity gestartet wird, wird eine Instanz der Klasse Init, die die Schnittstelle AppCompatActivity implementiert, erstellt. In dieser Klasse werden weitere Klassen, die im Rahmen des Projektes angelegt wurden, erstellt. Bei diesen Klassen handelt es sich um Gui, TodoApplicationLogic und Data. Auf diese wird im Nachfolgenden näher eingegangen.

Die Gui-Klasse ist die Schnittstelle zum Layout. Hier werden alle Layout-Elemente in Variablen bzw. typgleichen Objekten gespeichert. So wird z.B. das Eingabefeld für den Titel in einer Variablen vom Datentyp EditText gespeichert. Indem jedes Element aus dem Layout in einer Variablen hinterlegt wird, erhält man die Möglichkeit diese vom Java-Code aus auszulesen bzw. zu manipulieren. Diese Klasse besitzt einen Konstruktor, in welchem die eigentliche Erstellung der genannten Objekte geschieht. Über Getter- und Setter-Methoden stellt die Gui-Klasse den anderen Klassen die Schnittstelle zum Layout zur Verfügung.

Die Data-Klasse speichert das Todo und stellt die Schnittstelle zwischen der Datenbank und der Applikationslogik dar. Sie ist somit für das Speichern, Löschen und Ändern verantwortlich. Im Konstruktor dieser Klasse wird außerdem das Todo-Objekt initialisiert. Beim Initialisierungsvorgang der Data-Klasse wird eine ID übergeben. Ist diese ID nicht vorhanden, wird sie ein neues Todo in der Datenbank an. Andernfalls wird das Todo geladen.

In der TodoApplicationLogic-Klasse findet die gesamte Geschäftslogik statt. Sie verbindet im Wesentlichen die Data mit der Gui. Sie ist unter anderem für die Interaktion zuständig.

### Dokumentation des Quelltextes der Activity

In diesem Kapitel wird der Quelltext der Todo-Activity vorgestellt und erläutert. Dabei wird Klasse für Klasse, Methode für Methode vorgegangen. Methoden der Init-Klasse (Sertan Soner Cetin) Die Init Klasse verfügt insgesamt über 10 Methoden. Da die Klasse die Schnittstelle AppCompatActivity implementiert, sind die Methoden onCreate, onCreateOptionsMenu, onSaveInstanceState, onActivityResult, onBackPressed und onConfigurationChanged vorhanden. Daneben sind noch Methoden wie initData, initGUI und initApplicationLogic vorhanden. Private void initApplication() Diese Methode initialisiert das Objekt des Typs TodoApplicationLogic. Diese ApplicationLogic erhält die GUI- und Data-Objekte Private void initData(String todoId) Diese Methode initialisiert das Data-Objekt des Datentyps Data. Das Objekt erhält eine Instanz zur Activity und eine Todo-ID vom Typ String, welche dieser Methode ebenfalls übergeben wird. Private void initGui() Diese Methode initialisiert das Gui-Objekt. Dieses Objekt erhält eine Instanz zu der Activity. Public void onCreate(Bundle savedInstanceState) Diese Methode wird beim Erstellvorgang der Init-Klasse aufgerufen. Sie ruft die zuvor beschriebenen Methoden auf. Der Parameter savedInstanceState wird der Oberklasse übergeben. Methoden der Gui-Klasse (Sertan Soner Cetin) Die Gui-Klasse verfügt über einen Konstruktor, der im vorangegangenen Kapitel beschrieben wurde. Außerdem verfügt sie über diverse Getter- und Setter-Methoden. Es werden einige exemplarisch dargestellt. Public void setFocusableInTouchmode(boolean value) Diese Methode stellt den TouchModus des Layouts aus. Über den Parameter value kann angegeben werden, ob sich die Tastatur beim Start des Layouts aufklappen soll oder nicht. (true für nicht ausklappen, false für ausklappen) Public EditText getmTitle() Diese Getter-Methode liefert das Objekt mTitle des Datentyps EditText zurück. Public void setmTitle(string pTitle) Diese Setter-Methode schreibt in das Textfeld des Todo-Layouts den übergebenen Parameter mit dem Namen pTitle des Datentyps String. Public EditText getmText() Diese Getter-Methode liefert das Objekt mText des Datentyps EditText zurück. Das Objekt entspricht dem Layout-Element für die Beschreibung des Todos. Public void setColor(int color, int darkColor) An diese Methode werden zwei Farbwerte als Integer übergeben. Der erste Farbwert entspricht einem helleren Farbton, z.B. Hellblau. Der zweite wäre ein dunklerer Blauton, z.B. Dunkelblau. Das Layout bekommt die helle Farbe als Hintergrundfarbe, während die Toolbar oder Trenn-

linien zwischen den einzelnen Elementen die dunklere Farbe erhalten. Methoden der Data-Klasse (Sertan Soner Cetin) Diese Klasse verfügt über einen Konstruktor und über Getter- und Setter-Methoden. Durch die Schnittstelle zur Datenbank sind noch Methoden für das Löschen und Ändern der Todo in der Datenbank vorhanden. Public void deleteTodo() Löscht die Todo aus der Datenbank anhand seiner ID. Public void updateTodo() Ändert die Todo in der Datenbank und aktualisiert somit alle Eigenschaften. Public void setTitle(string title) Ändert den Titel der Todo. Der Parameter title des Datentyps String ist der neue Titel. Public boolean getmIsNew() Gibt an, ob die Todo neuerstellt wurde oder beim Start der Activity bereits vorhanden war. Dieser Wert wird außerhalb benötigt, um festzulegen, ob sich die Tastatur beim Activity-Start aufklappen soll oder nicht. Methoden der TodoApplicationLogic (Florian Rath) Die TodoApplicationLogic stellt die größte Klasse innerhalb der Todo-Activity dar. Dies liegt unter anderem daran, dass die meisten Verantwortlichkeiten hier liegen. Sie ist Dreh und Angelpunkt der gesamten Infrastruktur, da sie die Data bzw. Datenbank mit der Gui bzw. dem Layout verbindet und alles steuert. Private void initGui() Diese Methode stellt die Gui ein und ruft eine andere Methode namens dataToGui auf. Private void dataToGui() Diese Methode schreibt die Todo-Eigenschaften auf die einzelnen Layout-Elemente. So wird z.B. der Todo-Titel in das Eingabefeld für den Titel geschrieben oder die Farbe des Todos als Hintergrundfarbe festgelegt. Private void initListener() Da innerhalb der Applikationslogik die Listener für die einzelnen Events z.B. ClickEvent oder TouchEvent verwaltet werden, wurde diese Methode angelegt, um alle Listener an einer zentralen Stelle zu initialisieren. Hier werden der ClickListener, TouchListener und CheckedChangeListener initialisiert und bei den einzelnen Layout-Elementen registriert. Public void receiveDate(Date date) Diese Methode wurde für den Datepicker benötigt. Sie empfängt quasi das Datum aus dem Eingabefeld. Der Parameter date wird dann an das Eingabefeld übergeben. Public String formatDate(Date date) Diese Methode liefert einen String zurück. Der Parameter date wird in eine leserliche Formatierung gebracht. Die Formatierung des Datums ist z.B. „Dienstag, 13. Januar 2019“. Public void showDatePickerDialog(View v) Diese Methode initialisiert das DatePickerFragment. Dazu wird der Parameter v benötigt, um die beiden Start- und Enddatum-Felder unterscheiden zu können, da beide denselben DatePicker verwenden. Public void returnToOverview() Diese Methode leitet den Benutzer wieder zur Overview-Activity zurück. Bei diesem Vorgang wird außerdem die UpdateTodo-Methode aufgerufen, die weiter unten beschrieben ist. Public void onMenuItemClick(MenuItem item) Bei dieser Methode handelt es sich um einen Event-Handler. Diese wird ausgeführt, wenn ein

Menü-Item angeklickt wird. Da nur ein Menüpunkt vorhanden ist, ist es eindeutig. An dieser Stelle wird die externe Methode des Data-Objekts deleteTodo aufgerufen. Außerdem wird die zuvor beschriebene returnToOverview-Methode aufgerufen. Public void createList() Diese Methode erzeugt eine Liste bzw. initialisiert den TaskAdapter. Der TaskAdapter wird benötigt, um aus einer Liste von Tasks in eine ListView-Elementliste zu verwandeln. Dieser Adapter wird dem GUI-Element ListView zugewiesen. Private void addTask() Diese Methode fügt Tasks-Liste, welche in der GUI angezeigt wird, ein Element hinzu. Public void updateProgress() Die updateProgress-Methode holt sich aus dem Todo-Objekt den Anteil der erledigten Aufgaben. Dieser Wert wird in eine Prozentzahl umgewandelt. Der Prozentwert wird dem entsprechenden Layout-Element zugewiesen. Private void onEditTextClicked() Bei dieser Methode handelt es sich um einen Event-Handler. Dieser wird ausgeführt, wenn auf ein EditText-Feld getippt wurde. Dabei wird ein neues EditText-Feld erzeugt, um eine weitere Aufgabe eingeben zu können. Public void onActivityReturned(int requestCode, int resultCode, Intent data) Diese Methode deckt den Fall ab, falls in die Activity zurückgekehrt wird. Private void onDelteButtonClicked(int id) Es wird dieser Methode eine ID übergeben. Diese ID entspricht der Aufgabe, bei der der Lösch-Button geklickt wurde. Anhand dieser ID wird der Task aus der Liste entfernt. Private void onTextChanged(String s, View mView) Bei dieser Methode handelt es sich um einen Text-Changed Event-Handler. Der Parameter s steht für den Text und mView für das Element. Handelt es sich um den Titel, wird die setTitle-Methode des Data-Objekts aufgerufen. Handelt es sich um die Beschreibung, wird die setText-Methode desselben Objekts aufgerufen. Public void onOkButtonClicked() Wenn der OK-Buttons des Datumeingabefelds geklickt wird, wird die Methode UpdateTodo aufgerufen. Public void onBackPressed() Wird der Zurück-Button aus der Toolbar gedrückt, wird die UpdateTodo-Methode aufgerufen. So wird beim Verlassen der Activity die Todo in der Datenbank gespeichert. Private void addInputTaskField() Fügt der Task-Liste ein weiteres Element hinzu. Außerdem wird der TaskAdapter benachrichtigt, um die GUI zu aktualisieren. Hierdurch wird auch der prozentuale Anteil der erledigten Aufgaben beeinflusst, weshalb die Methode updateProgress aufgerufen wird. Public ArrayList<Task> getmTasks() Liefert die ArrayList mit Task-Objekten zurück. Private int getTasksItemCount() Liefert eine Zahl zurück, die angibt, wie viele Elemente in der Task-Liste gespeichert sind. Public Click-Listener getClickListener() Gibt das ClickListener-Objekt zurück. Public TouchListener getTouchListener() Gibt das TouchListener-Objekt zurück. Public CheckedChangeList-tener getmCheckedChangeListener() Gibt das CheckedChangeListener-Objekt zurück.

Public void UpdateTodo() Diese Methode speichert zuerst das Todo-Objekt aus der Data-Klasse in einer lokalen Todo-Methodenvariable. Von diesem werden der Titel und Beschreibung gesetzt. Das Start- und Enddatum werden ebenfalls aktualisiert. Am Ende der Methode wird das Todo in der Datenbank gespeichert bzw. aktualisiert. Public void onConfigurationChanged(Gui pGui) Bei dieser Methode handelt es sich um einen Event-Handler. Diese initialisiert die Gui und Listener erneut. Dazu werden die beschriebenen Methoden initGui und initListener aufgerufen.

### **5.3 Dokumentation der Navigation zwischen Activities (Yannick Rüttgers)**

Der Einstiegspunkt für den Nutzer ist die OverviewActivity. Von dieser aus können alle weiteren Activities aufgerufen werden, wie in der Planungsphase geplant.

Die Activities werden auf zwei Arten aufgerufen. Entweder geschieht dies, um ein neues TEN zu erstellen, oder um ein bereits vorhandenes aufzurufen.

Soll ein neues TEN erstellt werden, geschieht dies über einen der drei Buttons am oberen Bildschirmrand. Je nachdem welche TEN-Art angeklickt wurde, wird eine dazugehörige Activity erstellt und ohne Mitgabe weiterer Parameter aufgerufen.

Wenn hingegen ein bereits bestehendes TEN angezeigt werden soll, wird wieder die dazugehörige Activity erstellt. Diesmal wird der Activity allerdings als Parameter die ID des TENs weitergeben. Das Laden dessen übernimmt dann die jeweilige Activity.

Wenn aus den einzelnen TEN-Activities zurücknavigiert wird, gelangt der Nutzer zurück in die Übersichtsactivity. Die TENs innerhalb dieser werden neu geladen, um eventuelle Veränderungen anzeigen zu können.

### **5.4 Dokumentation der Activity-übergreifenden, persistenten Datenhaltung (Jan Beilfuß)**

Hier den Text einfach hin kopieren.

## 5.5 Dokumentation der Activity-übergreifenden Klassen (Ruthild Gilles)

Zu den Activity-übergreifenden Klassen gehören abgesehen von den Query-Klassen für die persistente Datenhaltung auch die Service-Klassen und die Models-Klassen. Die Models-Klassen enthalten die TEN-Klasse und die einzelnen Todo-, Event- und Note-Klassen. Hier sind auch weitere Util-Klassen untergeordnet. In den Service-Klassen sind Methoden enthalten, die die Schnittstelle zwischen Datenbank und Activities darstellen. Im Folgendem wird genauer auf die einzelnen Klassen eingegangen.

### 5.5.1 Models-Klassen

### 5.5.2 Service-Klassen

Damit die Activities die Daten der TEN-Objekte auf der dokumentenbasierten Datenbank speichern können, wurden Service Klassen implementiert. Hierzu gehörten hauptsächlich Klassen zum Erzeugen neuer TEN-Objekte (Create), zum Erhalten bereits gespeicherter TEN-Objekte von der Datenbank (Read), zum Speichern veränderter TEN-Objekte (Update) und zum Löschen von TEN-Objekten von der Datenbank (Delete). Diese sogenannte CRUD-Operationen wurden in den entsprechenden Klassen teilweise für alle drei verschiedenen Objekttypen einzeln eingefügt, teilweise aber auch für TEN-Objekte im Allgemeinen. Dank Polymorphie können die jeweils einzelnen Objekttypen ebenfalls an die entsprechenden Methoden übergeben werden.

Diese Struktur wurde während der Planungsphase vom Datenteam überlegt und während der Implementierung angepasst. Wie auch schon in der Planungsphase definiert enthält die Create-Klasse Methoden, die ein neues leeres Todo, Event oder Note zurückgeben. Diese Methode ruft den Konstruktor der jeweiligen TEN-Klasse auf. Eine Interaktion mit der Datenbank ist hier noch nicht nötig.

Die Read-Klasse hingegen muss auf die Datenbank zugreifen, um entweder alle TEN-Objekte an die Main-Activity in einem ListArray zu übergeben oder aber ein spezielles Todo, Event oder Note, welches von den einzelnen Todo-, Event- oder Note-Activities aufgerufen werden kann. Damit das gewünschte TEN-Objekt in der Datenbank gefunden werden kann, benötigen die Read-Methoden die ID des gewünschten Objektes. Dieses wird als String beim Aufruf der jeweiligen Methode übergeben.

Die Update-Klasse dient zum Speichern von Änderungen an Todo-, Event- und Note-Objekten. Dazu überprüft die Methode, der ein TEN-Objekt übergeben wurde, ob dieses bereits auf der Datenbank existiert. Ist dies der Fall, wird eine Methode zum Ausführen des Update-Befehls auf der Datenbank aufgerufen. Ist dies nicht der Fall, wird eine Methode zum Ausführen des Insert-Befehls auf der Datenbank aufgerufen. Da die Todo-, Event- und Note-Klassen von der TEN-Klasse erben, kann hier Polymorphie angewandt werden. Es ist nur eine Methode zum Speichern notwendig.

Für das Löschen von TEN-Objekten sind in der Delete-Klasse zwei Methode vorhanden. Die eine löscht nur ein übergebenes TEN-Objekt aus der Datenbank, indem es eine entsprechende Methode aus einer der Repository-Klassen aufruft und dieser die ID des TEN-Objektes übergibt. Sollen jedoch mehrere TEN-Objekte auf einmal gelöscht werden, kann der zweiten Methode eine Array List, die mehrere zu löschen Objekte enthält, übergeben werden. Dabei müssen die Objekte nicht alle von dem gleichen Datentyp sein, sondern können Todo-, Event- und Note-Objekte enthalten. Die Methode iteriert durch die übergebene Array List und ruft für jeden Eintrag die Methode zum Löschen eines Objektes auf.

## 6 Fazit der Teammitglieder

### 6.1 Fazit von Fabia Schmid

Mein Fazit wird sich in ein generelles Fazit zu der Applikation, ein Fazit bezüglich der Gruppe und ein Fazit zu meiner eigenen Arbeit unterteilen.

Die Applikation, welche von uns, dem Team Angry Nerds, entwickelt wurde, entspricht den Vorgeben, die von dem Professor vorgegeben wurden und erweitert diese mit verschiedenen Funktionen. Zusätzlich ist die Oberfläche farbenfroh und benutzerfreundlich. Zusätzlich achteten wir auf eine einfache und intuitive Bedienung, wodurch auch eine hohe User-Akzeptanz erwartet wird.

Somit kann in Bezug auf die Applikation, das Fazit gezogen werden, dass die Applikation erfolgreich und sehr zufriedenstellend entwickelt wurde.

Auch das Fazit bezüglich der Gruppenarbeit fällt sehr positiv aus. Alle Gruppenmitglieder waren stets motiviert und haben sich an allen Schritten beteiligt. Bei Problemen probierten alle zu helfen und eine Lösung zu finden. Zusätzlich waren alle Teammitglieder sehr zuverlässig und zielorientiert. Durch die durchgängige Motivation war das ganze Team auf einer Wellenlänge und konnte gut zusammenarbeiten und das Projekt erfolgreich abschließen. Die Kommunikation wurde hier sehr vereinfacht, durch die Nutzung von Microsoft Teams und einer WhatsApp-Gruppe, in der schnell kleinere Fragen beantwortet werden konnten.

Die Projektsteuerung, welche meine Hauptaufgabe war, verlief auch erfolgreich. Die geplanten Aktivitäten konnten bis auf zwei Meilensteine fristgerecht erfüllt werden. Die beiden Meilensteine konnte ich jedoch durch verschieben trotzdem abschließen, wodurch das Projektziel nicht gefährdet wurde. Zusätzlich sorgte ich durch regelmäßige Erinnerungen für die Erfüllung der Aufgaben und koordinierte verschiedene Aufgaben im Team erfolgreich. Neben der Koordination, wozu auch die Zeitplanung des Projektes gehörte, erstellte ich in der Overview Activity alle Layouts und programmierte vier Klassen. Auch diese Aufgabe vollendete ich erfolgreich. Die Layouts passen zu den vorher festgelegten Vorgaben im Mockup und die Funktionen funktionieren wie geplant.

Somit kann auch in Bezug auf meine Arbeit, ein positives Fazit gezogen werden. Ich erfüllte die Aufgabe der Projektleiterin zufriedenstellend und beendete auch die programmatischen Aufgaben erfolgreich.

## 6.2 Fazit von Florian Rath

Die Entwicklung der TEN-App hat durch umfassende Projekttätigkeiten viele Aspekte unseres Studiums aufgegriffen und diese in dem praktischen Anteil, bei der Entwicklung der App, verdeutlicht. Durch dieses Projekt habe ich viele neue Erfahrungen gesammelt und habe einen tieferen Einblick in das Projektmanagement erhalten. Die praktische Anwendung hat mir dabei geholfen die Theorie besser zu verinnerlichen.

Während der Umsetzung sind mir eigene Planungsfehler aufgefallen, die ich in einem nächsten Projekt nicht mehr machen würde, z.B. die Aufteilung der Entwicklungspakete. Das Problem habe ich schon in dem Kapitel “Beschreibung von Problemen” genauer erläutert. Natürlich sind auch einige unvorhergesehene Probleme aufgetreten, die jedoch durch gute Teamarbeit bewältigt wurden. Die Kommunikation nahm dabei eine Schlüsselrolle ein und lief in unserer Gruppe, über Office Teams, Whatsapp und teilweise wurde Programmiersessions über Discord gehalten. Teams bietet die Möglichkeit sich in einem Chat auszutauschen und Dateien für jeden abzulegen, Whatsapp diente vor allem der schnellen und direkten Kommunikation. Discord ist eine Software für Sprachchats mit mehreren Leuten, in der wir uns bei Programmierproblemen austauschten. Die Tools wurden von unserer Gruppe gut genutzt und es fand ein produktiver Austausch statt. Jedoch haben gleichzeitige Diskussionen bei Teams und Whatsapp der Kommunikation ein wenig geschadet, allerdings lässt sich so etwas im Eifer des Gefechts nur schwer vermeiden. Diese Problematik ist auch eher selten aufgetreten. Ich finde Kommunikation in einem Team sehr wichtig und bin sehr zufrieden mit der Art und dem Ablauf der Angry Nerds. Die App wurde durch kontinuierliche Verbesserungsvorschläge auf ein immer höheres Niveau getrieben.

Leider lag der Großteil der Projektphase in einer ungünstigen Zeit. Es mussten viele Klausuren und Ausarbeitungen geschrieben. Zu dem kam unsere IHK-Abschlussprüfung hinzu. Das machte den zeitlichen Rahmen für das Projekt sehr eng. Trotzdem war es eine interessante Zeit, wenn auchfordernde Zeit, in der ich viel gelernt habe, z.B. den Umgang

mit LaTeX, GitHub oder Android Studio. Abschließend lässt sich sagen, dass das Projekt erfolgreich abgeschlossen und fristgerecht fertiggestellt wurde.

### **6.3 Fazit von Jan Beilfuß**

Fazit

### **6.4 Fazit von Joscha Nassenstein**

Fazit

### **6.5 Fazit von Robin Menzel**

Fazit

### **6.6 Fazit von Ruthild Gilles**

Das Modul WIP enthält nicht nur die Programmierung einer Applikation, sondern für die erfolgreiche Implementierung wurde auch Projektmanagement benötigt. Diese Kombination finde ich realitätsnäher als es in anderen Modulen der Fall ist. Ein Projekt von vorne bis hinten zusammen in einem Team durchzuführen hat mir viele neue Erkenntnisse und Erfahrungen gebracht.

Zusätzlich wurden während des Projektes allerdings auch einige andere Fähigkeiten gefragt, welche ich noch nicht beherrschte. Dazu gehörte die Versionierung, welche wir mit Git Hub realisiert haben. Zu Beginn war es für mich eine Herausforderung die Funktionsweise von Git zu verstehen. Nach einiger Einarbeitungszeit beherrschte ich jedoch die Grundfunktionen, sodass ich diese Fähigkeit jetzt auch in meinem restlichen Leben einsetze kann. Abgesehen von der Versionierung, welche nicht für das Projekt gefordert war, stand mir die Aufgabe zu, mich mit dem Textverarbeitungsprogramm von Latex zu beschäftigen. Da uns hier ebenfalls jegliche Einweisung fehlte, benötigte ich auch hier zusätzliche Zeit zum Erlernen der benötigten Grundkenntnisse für Latex.

Jedoch werde ich auch diese neuen Kenntnisse außerhalb von dem Modul einsetzen können.

In unserem Projektteam gab es einige sehr gute Entwickler und ich fand es eine Herausforderung mit dieser Leistung mitzuhalten. Wenn ich entwickle benötige ich deutlich mehr Zeit, um mich in die Logik hinein zu denken. Deswegen kam es dazu, dass die Implementierung von Logik von anderen Teammitgliedern übernommen wurden, da es für diese ein höherer Zeitaufwand gewesen wäre, mir die Logik der umgebenden Klassen und Methoden zu erklären, als es eben selbst schnell zu machen.

Ich war dem Datenteam zugeordnet und zu Beginn hatten wir Schwierigkeiten mit der Planung der persistenten Datenhaltung. Die Schnittstelle zwischen den Activities und den Datenbank-Klassen, die von mir entwickelt wurden, konnten wir zu Beginn nicht im Detail planen, da wir nicht genau wussten, welche Anforderungen die Activities an die Datenbank stellen würden. Dies lag an einer nicht ausgereiften Kommunikation zu Beginn des Projektes. Aus diesem Grund erstellte ich Klassen und musste diese später ändern und anpassen. Es war mehr ein Ausprobieren als ein strukturiert geplantes Entwickeln. Bei einem nächsten Projekt würde ich besonders zu Beginn, häufiger Teammeetings einplanen um gemeinsam das Grundgerüst der Applikation zu planen.

Zusätzlich habe ich den Zeitaufwand für die Fertigstellung der Ausarbeitung in Latex unterschätzt. Besonders da im Zeitraum von November bis Februar ausgesprochen viele Prüfungsleistungen und auch der Abschluss unserer Ausbildung anstanden, war es für alle Teammitglieder eine Herausforderung ihren Teil der Ausarbeitung bis zur Deadline fertig zu stellen. Letztendlich hat es jedoch noch alles geklappt.

## 6.7 Fazit von Sertan Cetin

Das Projekt hatte als Ziel, eine App innerhalb einer so großen Gruppe zu entwickeln. Dies war eine völlig neue Erfahrung für mich. Auch, wenn ich in der Programmierung allgemein etwas erfahren bin, war dieses Projekt eine große Bereicherung für mich, weil jeder etwas programmieren sollte und sich die App aus mehreren Einzelteilen zu einem Ganzen zusammensetzte. Bisher hatte ich nämlich immer im Alleingang irgendwelche Software entwickelt.

Auch wenn wir in der Berufsschule bereits in Gruppen etwas programmiert haben, ist es nicht vergleichbar. Dort haben wir mehr oder weniger immer am selben PC zusammengearbeitet und die Gruppen waren auch mit zwei bis drei Personen auch eher überschaubar. Da wir während des TEN-Projektes so viele waren, mussten wir z.B. GIT verwenden. Hier konnte ich von den Vorerfahrungen einiger aus der Gruppe stark profitieren. Ich lernte von ihnen, wofür Branches gut sind und wie man allgemein ein GIT-Repository am besten verwalten sollte.

Dass wir so viele Gruppenmitglieder waren, hatte natürlich auch Auswirkung auf die Kommunikation. Wir nutzten als Microsoft Teams und parallel dazu WhatsApp. Ich empfand die Kommunikation eher als schwierig. Wenn man mal einige Stunden offline war und genau dann in der WhatsApp-Gruppe viel diskutiert wurde, hatte ich mehrere hundert ungelesene Nachrichten in der Gruppe. Die Stärke von Teams war, dass man benachrichtigt wurde, wenn man in einem Chat erwähnt wurde. So konnte ich wichtige Informationen immer mitbekommen. Die Ergebnisse der WhatsApp-Diskussionen waren zumindest auch in Teams.

Zu Beginn des Projektes hatte ich mich geärgert, dass uns ein API-Level vorgegeben wurde. Neuere API-Level hätten die Programmierung von vielen Dingen sehr viel einfacher gemacht. Aber rückblickend bin ich froh, weil man gelernt hat, wie manche Mechanismen im Hintergrund passieren.

Ich fand es gut, dass wir für das gesamte Projekt mehrere Monate Zeit hatten. Auch wenn in diese Zeit sehr viele andere Verpflichtungen fielen, konnte man dies durch eine gute Planung kompensieren.

Als Abschluss lässt sich sagen, dass ich durch das Projekt so viel gelernt habe, dass wir als Team durch unser neues Wissen für ein ähnliches Projekt mit dem gleichen Umfang insgesamt viel weniger Zeit brauchen würden. Vor allem, weil man die meisten Klassen erneut verwenden könnte.

## 6.8 Fazit von Yannick Rüttgers

Das Projekt verlief für mich sehr zufriedenstellend. In diesem Fazit möchte ich auf die drei Punkte Applikation, Team und meine persönliche Entwicklung eingehen.

Mit dem Endergebnis des Projektes, der TEN-Manager App, bin ich sehr zufrieden. Mir gefällt sowohl das Design als auch die Nutzbarkeit des Produkts. Auch der Code, der hinter der Applikation steckt, ist ordentlich strukturiert und nutzt die Paradigmen der Objektorientierung sehr gut.

Die Arbeit im Team Angry Nerds hat mir größtenteils Freude bereitet. Bis auf einzelne Konflikte, die allerdings immer schnell gelöst werden können, funktionierte die Zusammenarbeit im Team sehr gut. Es wurde sich an die meisten Absprachen gehalten, und auf die Ergebnisse der anderen Projektbeteiligten konnte sich verlassen werden.

Meine persönliche Entwicklung würde ich auch als positiv beurteilen. Neben dem Erlernen der Programmierung von Android-Apps, habe ich mich sehr auf das Nutzen der Paradigmen der Objektorientierung, maßgeblich Polymorphie und Vererbung konzentriert. Der so entwickelte Programmcode wirkt sehr aufgeräumt, und bereits vorhandene Programmteile konnten oft wiederverwendet werden.

## 7 Quellenverzeichnis

Quellen, die während der Entwicklung herbeigezogen wurden:

<https://code.tutsplus.com/tutorials/android-essentials-adding-events-to-the-users-calendar--mobile-8363>  
<https://developer.android.com/guide/topics/ui/controls/pickers>  
<https://developer.android.com/reference/android/widget/DatePicker>  
<https://developer.android.com/training/appbar/setting-up>  
<https://developer.android.com/training/sharing/shareaction>  
<https://developers.google.com/maps/documentation/android-sdk/start>  
<https://guides.codepath.com/android/using-the-app-toolbar>  
<https://nnish.com/2014/12/16/scheduled-notifications-in-android-using-alarm-manager/>  
<https://stackoverflow.com/questions/35648913/how-to-set-menu-to-toolbar-in-android>  
[https://www.tutorialspoint.com/android/android\\_datepicker\\_control.htm](https://www.tutorialspoint.com/android/android_datepicker_control.htm)  
<http://www.vogella.com/tutorials/AndroidActionBar/article.html>

## 8 Anhang - Quelltexte

### 8.1 Activities

### 8.2 Data

#### 8.2.1 Service Klassen (Ruthild Gilles)

**Listing 1:** Create Klasse (Ruthild Gilles)

```
public class Create {  
    /* Ruthild Gilles  
     * Class Create contains methods to create new empty TEN objects.  
     * This class only exists to give a consistent form to the CRUD methods.  
     */  
  
    public static Todo newTodo() {  
        return new Todo();  
    }  
  
    public static Event newEvent() {  
        return new Event();  
    }  
  
    public static Note newNote() {  
        return new Note();  
    }  
}
```

**Listing 2:** Read Klasse (Ruthild Gilles)

```

public class Read {
    /* Ruthild Gilles
    Class Read contains methods to get all or one specific TEN object.
    */

    /**
     *-----*
     * Method to get all TEN objects in an arraylist
     *-----*/
    public static ArrayList<TEN> getAllTENs() {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        ArrayList<TEN> allTEN;
        allTEN = databaseRepository.getAllTENs();
        Log.i("Mainfix", "Number Of TENs: " + allTEN.size());
        for (TEN ten : allTEN) {
            Log.i("Mainfix", "ID: " + ten.getID() + ", Titel: " + ten.getTitle());
        }
        return allTEN;
    }

    /**
     *-----*
     * Methods to get one TEN object by ID
     *-----*/
    public static Todo getTodoByID(String id) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        Todo todo = databaseRepository.getTodoByID(id);
        return todo;
    }

    public static Event getEventByID(String id) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        Event event = databaseRepository.getEventByID(id);
        return event;
    }

    public static Note getNoteByID(String id) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        Note note = databaseRepository.getNoteByID(id);
        return note;
    }

    public static int[] getColors(String tenID) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        int[] colors = databaseRepository.getTENColors(tenID);
        return colors;
    }
}

```

**Listing 3:** Update Klasse (Ruthild Gilles)

```

public class Update {
    /* Ruthild Gilles
       Class Update contains methods to save information on a changed or newly created TEN object */
    /**
     *-----*
     * Methods for saving a TEN object
     *-----*/
    public static void saveTEN(TEN newTen) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        if (newTen.getID() == null) {
            databaseRepository.insertTEN(newTen);
        } else databaseRepository.updateTEN(newTen);
    }
}

```

**Listing 4:** Delete Klasse (Ruthild Gilles)

```

public class Delete {
    /* Ruthild Gilles
       Class Delete contains methods to delete the given TEN object.*/
    public static void deleteTEN(String tenID) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        databaseRepository.deleteTEN(tenID);
    }

    public static void deleteMultipleTENs(ArrayList<String> tenIDs) {
        DatabaseRepository databaseRepository = new DatabaseRepository();
        for (String tenID : tenIDs) {
            databaseRepository.deleteTEN(tenID);
        }
    }
}

```

## 8.3 Overview

## 8.4 Main Activity

## 9 Anhang - Verwendete Tools und Hilfsmittel (Robin Menzel)

Tool / Programm	Einsatz
Adobe Xd	Erstellung des Mock-Ups
Android Studio	Entwicklungsumgebung (IDE)
Draw.io	Erstellung der UML-Diagramme
Excel	Durchführung der Aufwandsschätzung
GitHub	Versionsverwaltung des Quellcodes
LaTeX	Dokumentation des Projektes (Struktur)
OneDrive	Dateiablage
OneNote	Protokollierung, Notizen und Absprachen
PowerPoint	Erstellung der Planungsdokumente
Word	Erstellung von Dokumenten

## **Ehrenwörtliche Erklärung**

Hiermit erkläre ich, dass ich die vorliegende Schriftliche Ausarbeitung selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

---

Ort, Datum

---

Unterschrift