



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:

09.02.2019

Inhaltsverzeichnis

Abbildungsverzeichnis	V
Listingverzeichnis	VII
1 Das Team: Angry Nerds	1
2 Ziele des Projektes (Fabia Schmid)	2
3 Projektplanung	4
3.1 Beschreibung des Funktionsumfangs (Florian Rath)	4
3.2 Projektlaufplan (Fabia Schmid)	5
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) .	14
3.3.3 Planung der Activities und Layouts (Florian Rath)	17
3.3.4 Planung der Navigation zwischen den Activities (Yannick Rüttgers)	18
3.4 Geplante Aufgabenverteilung im Team (Fabia Schmid)	19
4 Beschreibung des Projektverlaufs	21
4.1 Tatsächliche Aufgabenverteilung im Team (Fabia Schmid)	21
4.2 Teammeetingprotokolle	23
4.3 Projekttagebücher aller Teammitglieder	29
4.3.1 Projekttagebuch Jan Beilfuß	29
4.3.2 Projekttagebuch Joscha Nassenstein	32
4.3.3 Projekttagebuch Fabia Schmid	35
4.3.4 Projekttagebuch Florian Rath	37
4.3.5 Projekttagebuch Robin Menzel	38
4.3.6 Projekttagebuch Ruthild Gilles	40
4.3.7 Projekttagebuch Sertan Cetin	42
4.3.8 Projekttagebuch Yannick Rüttgers	43
4.4 Beschreibung von Problemen	46
4.4.1 Probleme von Fabia Schmid	46
4.4.2 Probleme von Florian Rath	46
4.4.3 Probleme von Jan Beilfuß	47
4.4.4 Probleme von Joscha Nassenstein	47

4.4.5	Probleme von Ruthild Gilles	48
4.4.6	Probleme von Sertan Cetin	49
4.4.7	Probleme von Yannick Rüttgers	49
5	Dokumentation der Software	51
5.1	Dokumentation der Paketstruktur (Sertan Cetin)	51
5.2	Dokumentation der Activities	54
5.2.1	Main Activity	54
5.2.2	Todo Activity	78
5.2.3	Event Activity	91
5.2.4	Note Activity	101
5.2.5	NoteTag Activity	129
5.3	Dokumentation der Navigation zwischen Activities (Yannick Rüttgers)	136
5.4	Dokumentation der Activity-übergreifenden, persistenten Datenhaltung (Jan Beilfuß)	136
5.4.1	Persistente Datenhaltung	136
5.4.2	Repository	139
5.5	Dokumentation der Activity-übergreifenden Klassen	148
5.5.1	Models-Klassen (Ruthild Gilles)	148
5.5.2	Service-Klassen (Ruthild Gilles)	149
5.5.3	Modules-Klassen	151
6	Fazit der Teammitglieder	153
6.1	Fazit von Fabia Schmid	153
6.2	Fazit von Florian Rath	154
6.3	Fazit von Jan Beilfuß	155
6.4	Fazit von Joscha Nassenstein	156
6.5	Fazit von Robin Menzel	157
6.6	Fazit von Ruthild Gilles	158
6.7	Fazit von Sertan Cetin	159
6.8	Fazit von Yannick Rüttgers	161
7	Quellenverzeichnis	162
7.1	Quellen zur Entwicklung	162
7.2	Quellen zur Entwicklung der Benutzeroberfläche	163
7.3	Bilder in den Mockdaten	164

7.4 Quellen zu Latex	164
8 Anhang - Quelltexte	165
8.1 Activities	165
8.1.1 Event	165
8.1.2 Note	165
8.1.3 Todo	167
8.2 Data	167
8.2.1 Models	167
8.2.2 Repository	175
8.2.3 Services	188
8.3 Modules	192
8.3.1 Image Compression	192
8.3.2 Share	192
8.4 Overview	192
8.4.1 Event Fragment	192
8.4.2 Header	192
8.4.3 Image Fragment	192
8.4.4 Note Fragment	192
8.4.5 Overview Activity	192
8.4.6 Super Classes	192
8.4.7 Todo Fragments	192
9 Anhang - Verwendete Tools und Hilfsmittel (Robin Menzel)	193

Ehrenwörtliche Erklärung

Abbildungsverzeichnis

Abbildung 1: Die Angry Nerds	1
Abbildung 2: Projektstrukturplan	6
Abbildung 3: GANTT-Diagramm	8
Abbildung 4: Mockups - Übersicht über alle TENs	11
Abbildung 5: Mockups - Übersicht über ein vorhandenes und ein neues Event	12
Abbildung 6: Mockups - Übersicht über ein vorhandene, eine leere und eine Bild-Notiz	13
Abbildung 7: Mockups - Übersicht über vorhandene und neue Todos	14
Abbildung 8: Klassendiagramm	15
Abbildung 9: Systemkontextdiagramm	16
Abbildung 10: CRUD-Klassen	17
Abbildung 11: Paketstruktur	52
Abbildung 12: Paketstruktur	53
Abbildung 13: Overview Activity	56
Abbildung 14: Landscape Ansicht - Overview Activity	57
Abbildung 15: Note Fragments	58
Abbildung 16: Todo Fragments	59
Abbildung 17: Event Fragments	60
Abbildung 18: Overview Activity - Löschen	61
Abbildung 19: Overview Activity - Suchen	62
Abbildung 20: Usecase Diagramm der Overview	64
Abbildung 21: Klassendiagramm	66
Abbildung 22: Todo Activity im Landscape-Modus	80
Abbildung 23: Todo Activity im Portrait-Modus	81
Abbildung 24: Datumeingabe	83
Abbildung 25: Screenshot - Event Activity	93
Abbildung 26: Screenshot - Event Activity Erinnerung	94
Abbildung 27: Screenshot - Event Activity Wiederholung	94
Abbildung 28: Screenshot - Event Activity Toolbar	94
Abbildung 29: Use-Case - Event Activity	95
Abbildung 30: Note Portrait-Ansicht	103
Abbildung 31: Note Landscape-Ansicht	104
Abbildung 32: Note Dialog zum Hinzufügen eines Bildes	106

Abbildung 33: Note Bildansicht	107
Abbildung 34: Note Toolbarmenü	108
Abbildung 35: Note Use-Case Diagramm	109
Abbildung 36: Darstellung der Paketstruktur	112
Abbildung 37: Codebeispiel	121
Abbildung 38: NoteTags Stichwortansicht	130
Abbildung 39: NoteTag Use-Case Diagramm	132
Abbildung 40: Dokumentenstruktur	138
Abbildung 41: Paketstruktur in der Repository-Schicht	140
Abbildung 42: Übersicht über Data-Klassen	148

Listingsverzeichnis

Listing 1: Todo (Joscha Nassenstein)	167
Listing 2: Event (Joscha Nassenstein)	168
Listing 3: Note (Joscha Nassenstein)	169
Listing 4: TEN (Joscha Nassenstein)	170
Listing 5: BundleKeys (Jan Beilfuß)	171
Listing 6: Colors (Jan Beilfuß)	172
Listing 7: Image (Jan Beilfuß)	173
Listing 8: MockData (Jan Beilfuß)	173
Listing 9: ReccurringType (Joscha Nassenstein)	173
Listing 10: Tasks (Joscha Nassenstein)	174
Listing 11: DatabaseRepository (Jan Beilfuß)	175
Listing 12: DataContextManager (Jan Beilfuß)	176
Listing 13: RepositoryConstants (Jan Beilfuß)	177
Listing 14: QuerriedTenConverter (Jan Beilfuß)	178
Listing 15: SeparateAttributesConverter (Jan Beilfuß)	179
Listing 16: TensJsonParser (Jan Beilfuß)	180
Listing 17: ImageDeleter (Jan Beilfuß)	181
Listing 18: ImageSaver (Jan Beilfuß)	182
Listing 19: FileRepositroy (Jan Beilfuß)	183
Listing 20: FileSystemConstants (Jan Beilfuß)	183
Listing 21: GetAllTensQuery (Jan Beilfuß)	184
Listing 22: ReadRepository (Jan Beilfuß)	185
Listing 23: DocumentSaver (Jan Beilfuß)	186
Listing 24: WriteRepository (Jan Beilfuß)	187
Listing 25: Create (Ruthild Gilles)	188
Listing 26: Read (Ruthild Gilles)	189
Listing 27: Update (Ruthild Gilles)	190
Listing 28: Delete (Ruthild Gilles)	190
Listing 29: ImageService (Ruthild Gilles)	191

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.

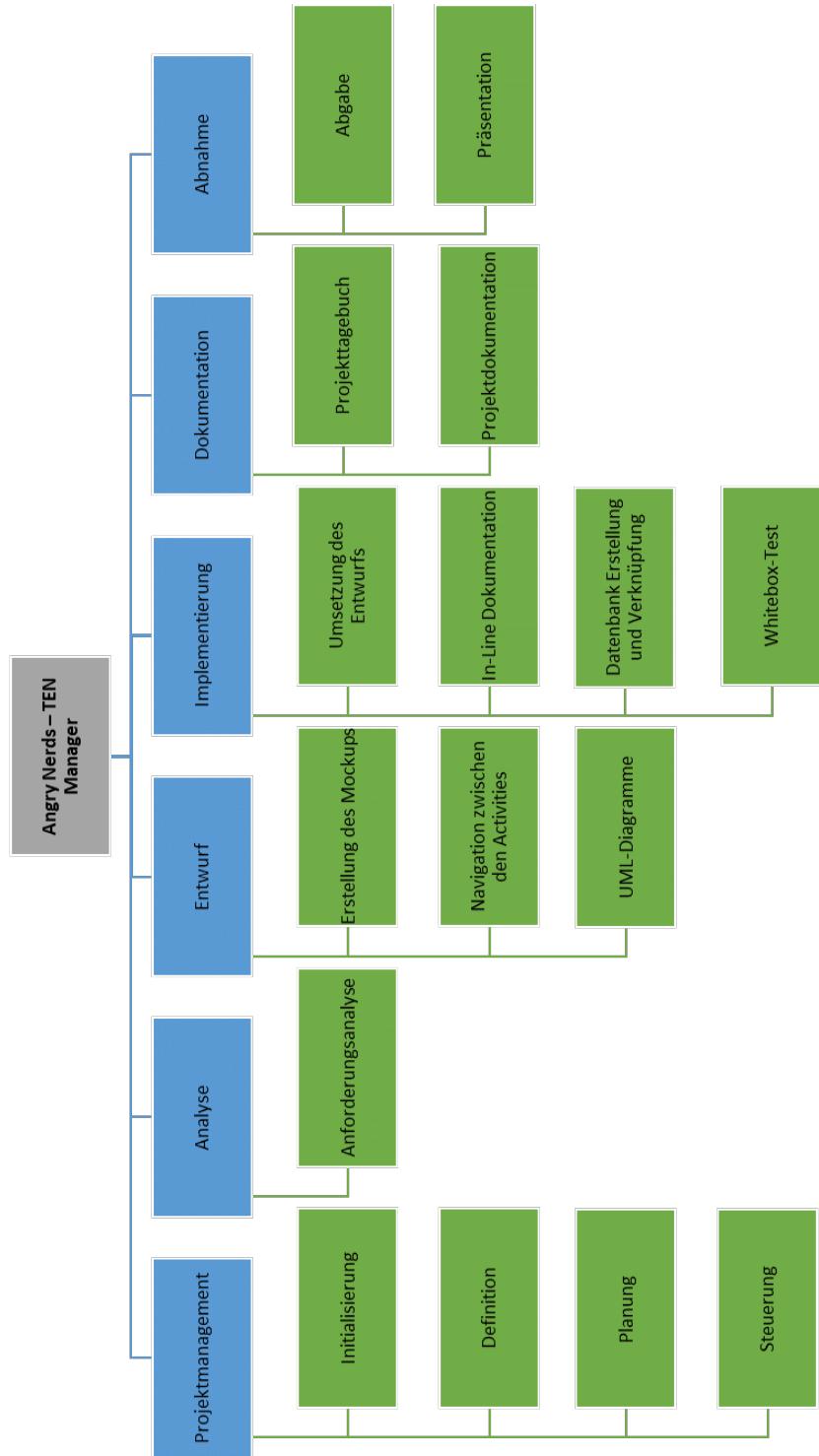
Dem Benutzer sollen für alle TENs die Grundfunktionen Erfassung, Veränderung, Löschung und Betrachtung zur Verfügung stehen. Außerdem soll es dem Benutzer möglich sein die Ansicht einzelner TENs aufzurufen und dort die Eingabefunktion benutzen zu können. Die Funktionen sollen eine einfache und intuitive 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.

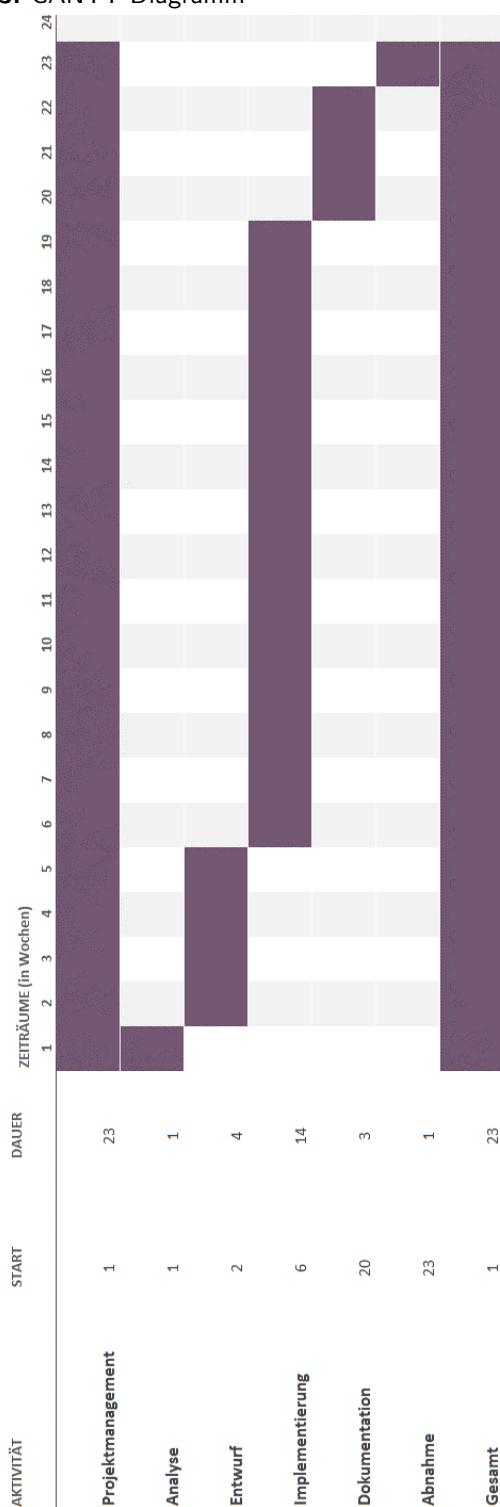
Abbildung 2: Projektstrukturplan

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 Nerd - 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.

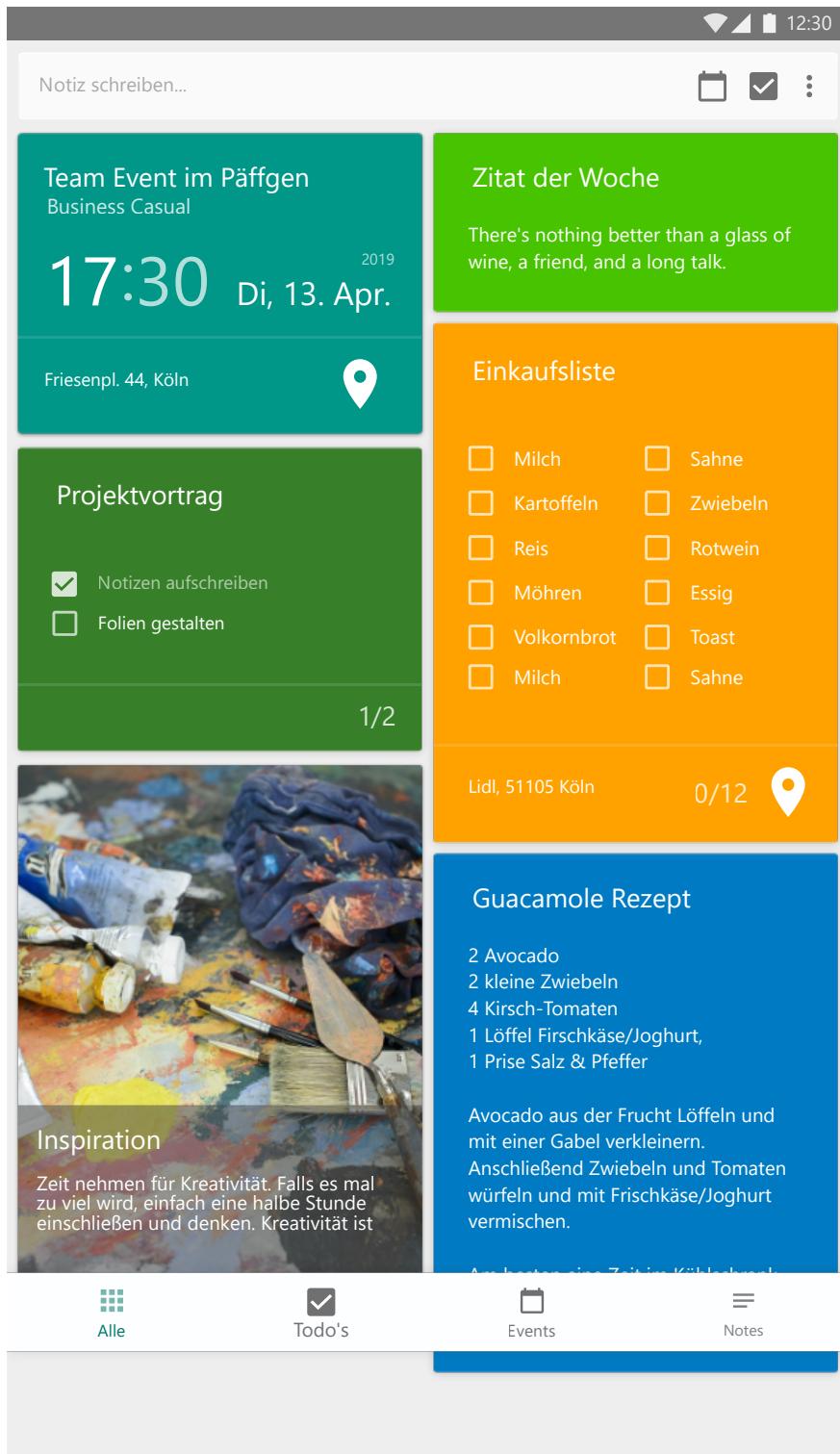
- Google Notizen
- Google Kalender
- Google Fotos
- Wunderlist

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.

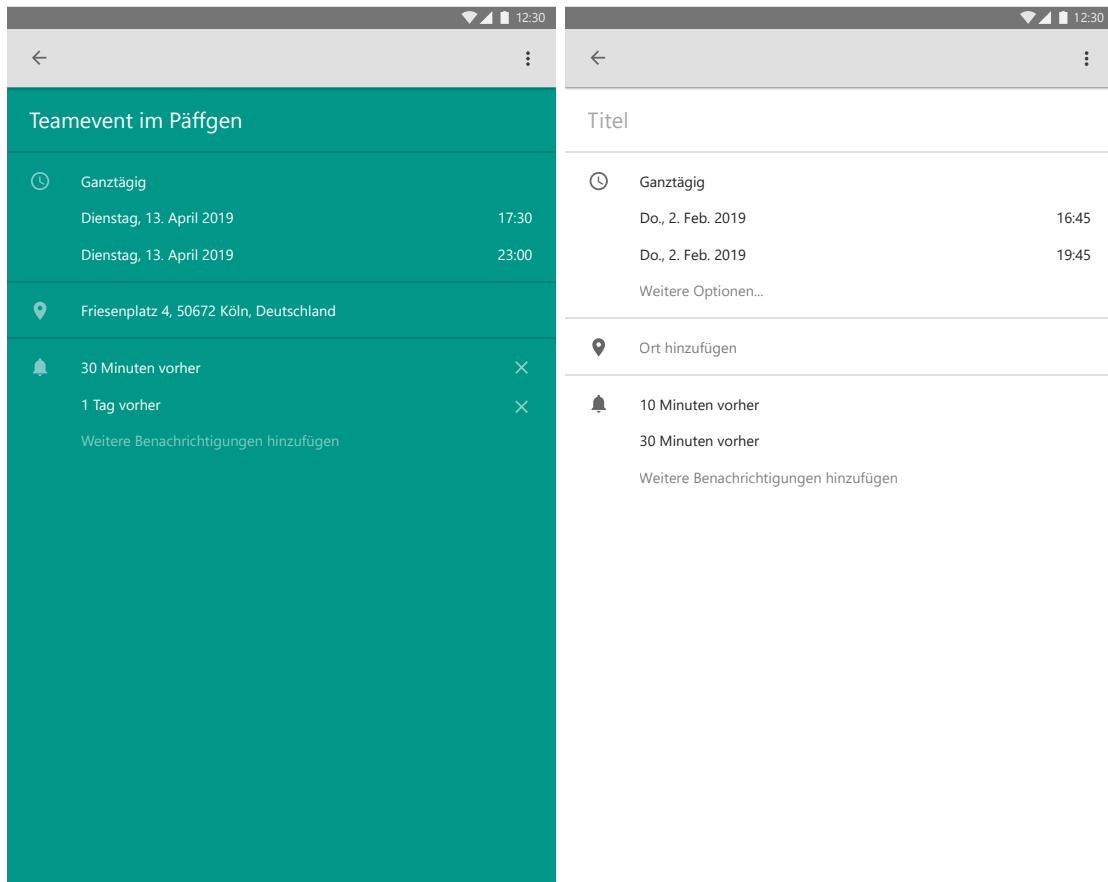
In Adobe Xd lassen sich nicht nur die Oberflächen Designen, sondern auch die User Experience abbilden. So lassen sich Flächen mit einander Verbinden, die nun durch einen Klick geöffnet werden können. Auf diese Art und Weise war es uns möglich Personen aus unserem Umkreis unsere App testen zu lassen. 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: Mockups - Übersicht über alle TENs



Quelle: Erstellt von Robin Menzel

Abbildung 5: Mockups - Übersicht über ein vorhandenes und ein neues Event



Quelle: Erstellt von Robin Menzel

Abbildung 6: Mockups - Übersicht über ein vorhandene, eine leere und eine Bild-Notiz

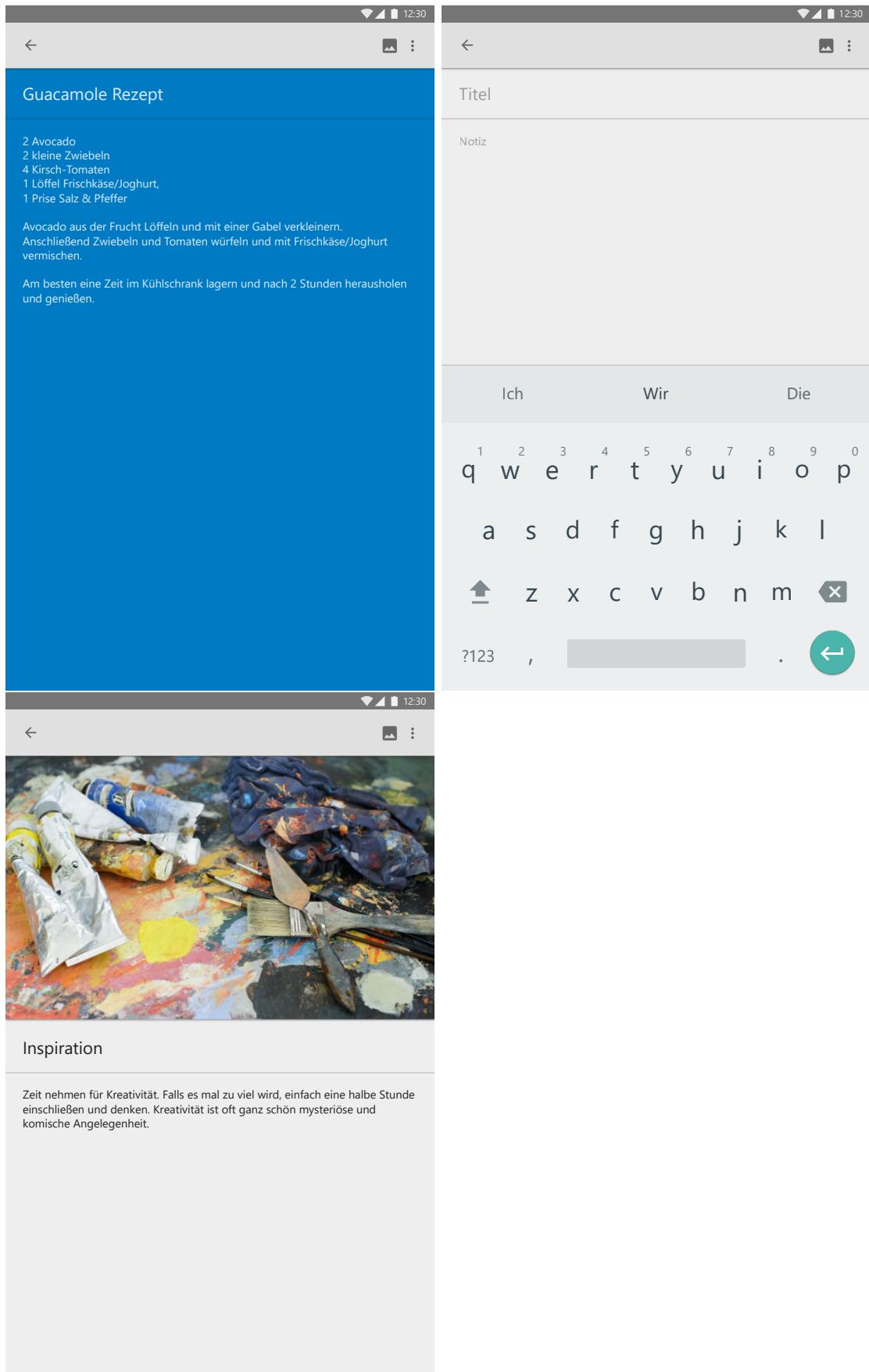
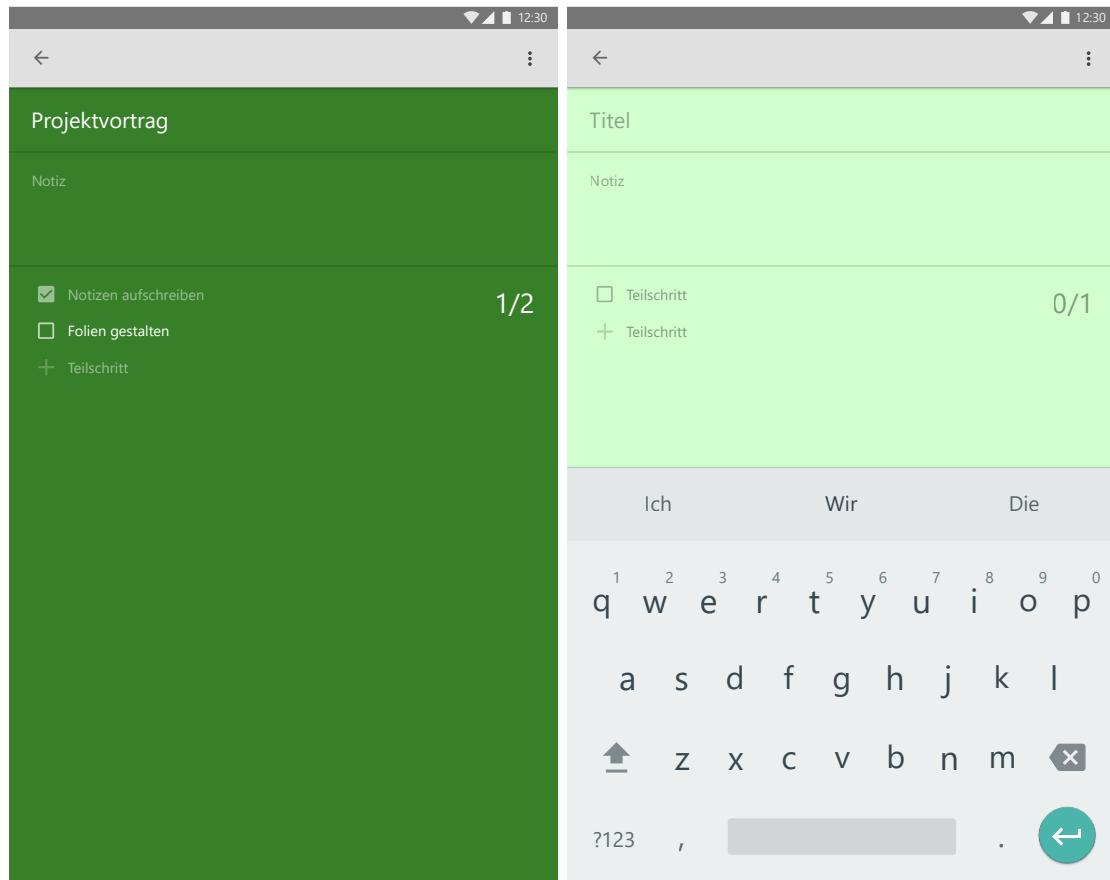
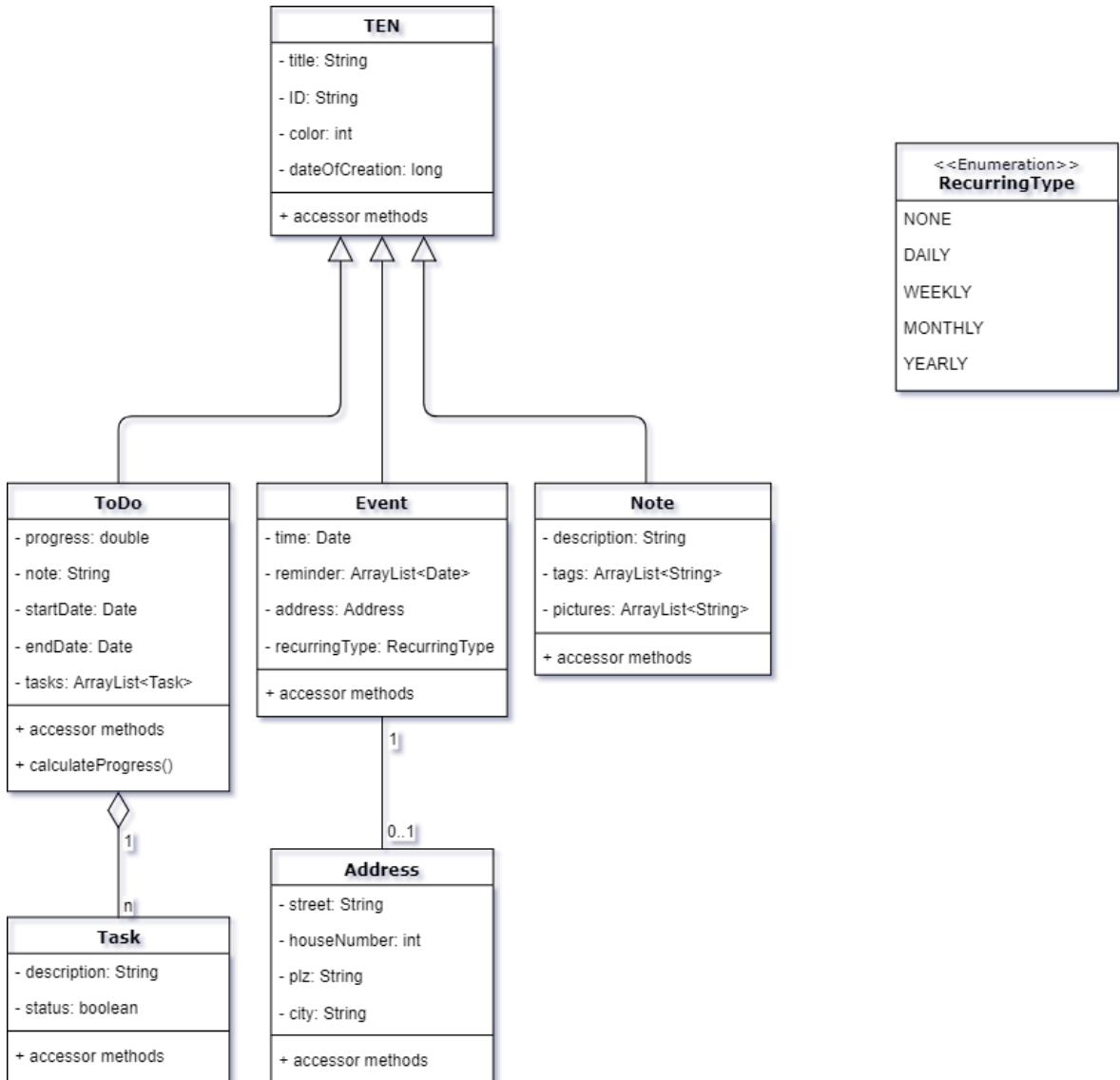


Abbildung 7: Mockups - Übersicht über vorhandene und neue Todos

Quelle: Erstellt von Robin Menzel

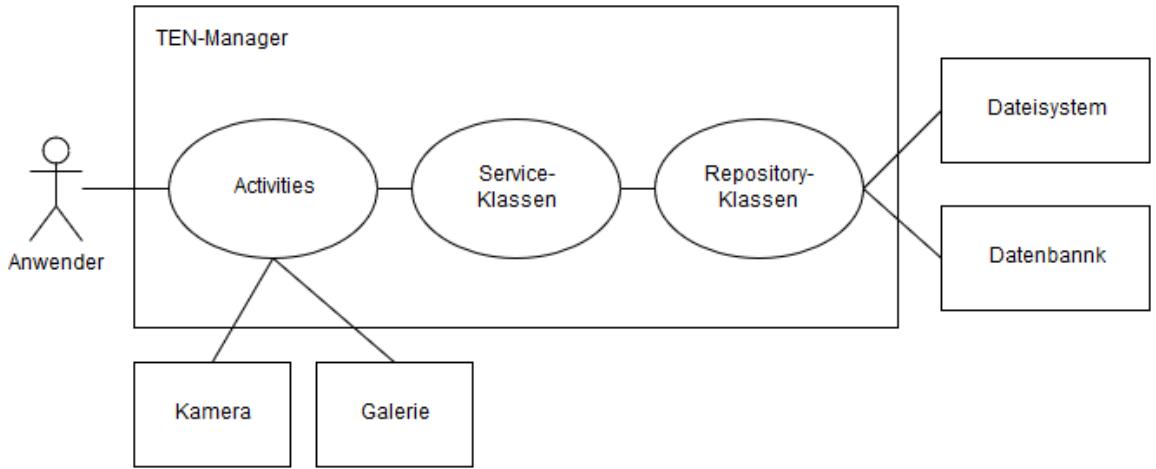
3.3.2 Planung der Datenstruktur und Schnittstellen (Ruthild Gilles)

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
+ newTodo(): Todo + newEvent(): Event + newNote(): Note	+ getAllTENs(): ArrayTENs + getTodoByID(String): Todo + getEventByID(String): Event + getNoteByID(String): Note	+ saveTEN(TEN):	+ deleteTEN(TEN): + deleteMultipleTENs(ArrayTENs):

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)

In dieser Planungsphase wurde mit einer ausführlichen Anforderungsanalyse begonnen, um die Activities und deren Umfang abschätzen zu können. Der teils unterschiedliche Umfang der Activities ist durch die gegebenen Anforderungen zustande gekommen.

Außerdem wurde geprüft ob die Activities Beziehungen aufweisen. Als Activities wurden zuerst das Todo, Event und Note (TEN) identifiziert. Die gesamte Darstellung der TENs sollte in einer Übersicht erfolgen, der Overview. Die Overview bildet eine weitere wichtige Activity, um der App eine übersichtliche Darstellung zu geben.

Nach der Anforderungsanalyse wurden die Activities in Aufgabenpakete zerlegt, um diese auf die Teammitglieder verteilen zu können. Die Layouts können diesen Activities zugewiesen werden und sind daher logisch an diese gebunden. Der Aufbau der Layouts

soll sich an dem erstelltem MockUp orientieren, um der App einen einheitlichen und ansehnlichen Look zu verpassen.

Nachdem die Planung für die Aufteilung der Activities feststand, wurde diese mit den Teammitglieder abgesprochen.

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 zurück navigiert, 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	Einbindung der Datenbank, Erstellung der Event Activity, 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

	Protokoll Kick-Off Meeting Projekt: Angry Nerds - TEN-Manager	Verfasser: Yannick Rüttgers/ Fabia Schmid 5. September 2018 Seite 1 vom 1 UNITY
--	--	--

Datum: 05.09.2018
 Start: 10:20 Uhr
 Ende: 12:40 Uhr

Teilnehmer:

- Fr. Fabia Schmid
- Fr. Ruthild Gilles
- Hr. Robin Menzel
- Hr. Florian Rath
- Hr. Sertan Cetin
- Hr. Jan Beilfuß
- Hr. Joscha Nassenstein
- Hr. Yannick Rüttgers

Verteiler: Teilnehmer

I = Information | M = Maßnahme | E = Entscheidung

Sachstand & Bewertung	I	M	E	Verantwortlicher
Vorbemerkung				
• Es gibt ein Team namens Angry Nerds, Git ist aufgesetzt.	I			Hr. Beilfuß & Hr. Menzel
Top 1 : Das komische Papier durchgehen				
• Die Aufgabenstellung wurde verstanden	I			
• Struktur des Dokumentes wird vom Latexbeauftragten übernommen		E		Hr. Rüttgers
• Gruppenweite Koordination übernimmt Fr. Schmid		E		Fr. Schmid
• Zusammenführung des Programms und Versionsverwaltung übernimmt Hr. Menzel				
• Zusammenführung der Studienarbeit und Latexvorlagen übernimmt Hr. Rüttgers				
• Spezifikation & Dokumentation des Datenmodells übernehmen Fr. Gilles & Hr. Beilfuß				
• Navigation zwischen Activities und Activity-Wechsel-Events übernehmen Hr. Rath & Hr. Cetin				
• Latex-Projekttagbuch wird erstellt & verteilt	E			Hr. Rüttgers
• Als Entwicklungsumgebung wird Android Studio verwendet	E			
• Die App wird für das Samsung Galaxy Note 8 entwickelt + die App muss nicht relativ sein	E			
• Termine werden in Teams eingetragen	E			Fr. Schmid
• Mockup wird durch Hr. Rath, Cetin & Menzel erstellt	E			
• Meilensteinplan wird durch Fr. Schmid & Hr. Rüttgers erstellt	E			
• Aufgabenverteilung am ersten Projekttag:		E		Fr. Schmid
○ 1. Fr. Gilles, Hr. Nassenstein, Hr. Beilfuß				
○ 2. Übersicht Activities & Navigation: Hr. Menzel, Hr. Cetin, Hr. Rath				
○ 3. Teilaufgaben definieren: Fr. Schmid, Hr. Rüttgers				
Top 2 : Aufgaben Bearbeitung				
• Bearbeitung der Aufgaben in kleinen Teams	I			
• Vorstellung der Ergebnisse		E		alle
○ 1. Muss-/ Kann-Kriterien werden von Fr. Gilles, Hr. Nassenstein und Hr. Beilfuß festgelegt und durch das Team erweitert				
○ 2. Mockup-Entwurf wird von Hr. Rath, Hr. Menzel und Hr. Cetin vorgestellt und vom Team diskutiert				
○ 3. Termine / Meilensteine werden von Fr. Schmid vorgestellt				

	Protokoll Meeting Data-Team Projekt: Angry Nerds - TEN-Manager	Verfasser: Joscha Nassenstein 15. September 2018 Seite 1 von 1 Angry Nerds
--	---	---

Datum: 15.09.2018
Start: 11:40 Uhr
Ende: 12:25 Uhr

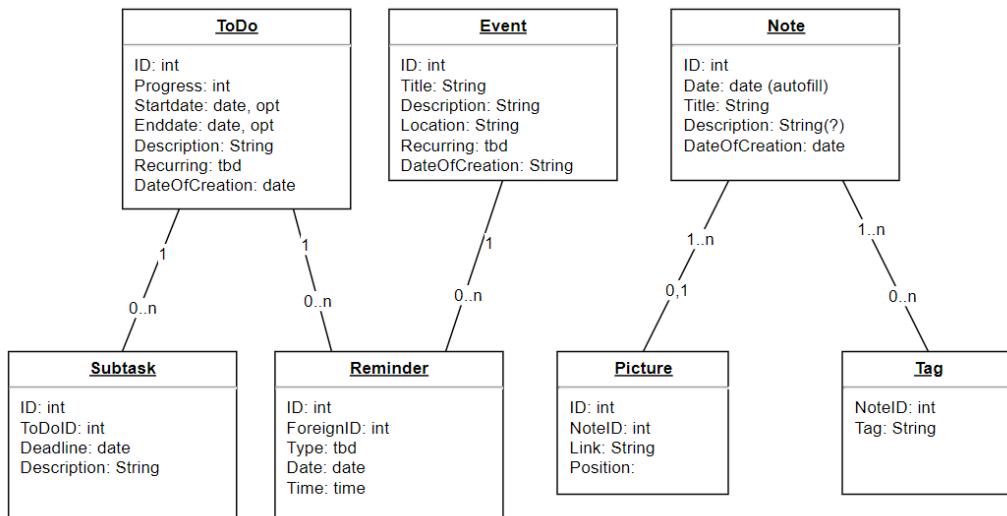
Teilnehmer:

- Fr. Ruthild Gilles
- Hr. Jan Beifuß
- Hr. Joscha Nassenstein

Vertreter: Team Angry Nerds

I = Information | M = Maßnahme | E = Entscheidung

Sachstand & Bewertung	I	M	E	Verantwortlicher
Vorbemerkung				
• Anfallende Daten wurden bereits zum ersten Projekttag festgehalten	I			
Topic 1: Erstellung eines Datenmodells				
• Erstellung der verschiedenen Models für die Daten			E	
• Sammlung von zusätzlichen Attributen				
• Erstellung des Datenmodells		M		Herr Nassenstein



	Protokoll Team-Meeting Projekt: Angry Nerds - TEN-Manager	Verfasser: Ruthild Gilles 22. September 2018 Seite 1 vom 1 Angry Nerds
--	--	---

Datum: 22.09.2018
Start: 10:20 Uhr
Ende: 11:00 Uhr

Teilnehmer:

- Fr. Fabia Schmid
- Fr. Ruthild Gilles
- Hr. Robin Menzel
- Hr. Florian Rath
- Hr. Sertan Cetin
- Hr. Jan Beilfuß
- Hr. Joscha Nassenstein
- Hr. Yannick Rüttgers

Verteiler: Teilnehmer

I = Information | M = Maßnahme | E = Entscheidung

Sachstand & Bewertung	I	M	E	Verantwortlicher
Vorbemerkung				
<ul style="list-style-type: none"> • Mock-Up wurde erstellt von Robin Menzel • Datenmodell wurde erstellt von Joscha Nassenstein, Jan Beilfuß, Ruthild Gilles • Dokumentation in LaTex verzögert sich auf Grund von Hardware-Problemen 	I			
Topic 1: Vorstellung des Mock-Ups				
<ul style="list-style-type: none"> • Das Mock-Up wurde zuvor von Robin Menzel erstellt • Er stellt das Mock-Up den restlichen Teammitgliedern vor • Die Teammitglieder geben Feedback • Auflistung aller Activities anhand des Mock-Ups <ul style="list-style-type: none"> ◦ Hauptseite mit Übersicht über alle TENs ◦ Seite zum Ansehen und Bearbeiten einer Notiz ◦ Seite zum Ansehen und Bearbeiten eines ToDos ◦ Seite zum Ansehen und Bearbeiten eines Events • Entscheidung zur Darstellung der Bilder einer Notiz: oben auf dem Bildschirm; durch seitliches Wischen wird nächstes Bild angezeigt 	I	E	Hr. Menzel	
Topic 2: Vorstellung des Datenmodells				
<ul style="list-style-type: none"> • Das Datenmodell wurde zuvor von Jan Beilfuß, Joscha Nassenstein und Ruthild Gilles erstellt • Hr. Beilfuß stellt das Datenmodell den restlichen Teammitgliedern vor 	I			
Topic 3: Weiteres Vorgehen				
<ul style="list-style-type: none"> • Aufteilung der Activities auf die einzelnen Entwickler erfolgt innerhalb der nächsten Woche durch Florian Rath und Sertan Cetin 	I / E			Hr. Rath und Hr. Cetin

Protokoll Team-Meeting		Verfasser: Jan Beilfuß 26. Oktober 2018 Seite 1 vom 1 Angry Nerds
Projekt: Angry Nerds - TEN-Manager		

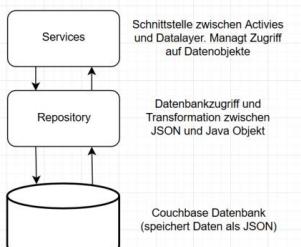
Datum: 26.10.2018
Start: 16:50 Uhr
Ende: 17:40 Uhr

Teilnehmer:

- Fr. Ruthild Gilles
- Hr. Jan Beilfuß
- Hr. Joscha Nassenstein

Verteiler: Teilnehmer

I = Information | M = Maßnahme | E = Entscheidung

Sachstand & Bewertung	I	M	E	Verantwortlicher
Vorbemerkung				
• Hr. Beilfuß hat nach geeignete Datenbanktechnologien recherchiert	I			
Topic 1: Technische Auswahl				
• Die Datenbanktechnologie Couchbase wird den Teammitgliedern durch Herrn Beilfuß vorgestellt	I			
○ Herr Beilfuß wird damit beauftragt, diese im weiteren Projektverlauf zu implementieren		M		Hr. Beilfuß
Topic 2: Struktur der Dataschicht				
• Die Struktur der Datenschicht wurde im Team abgestimmt			E	
		E		
Topic 2: Abstimmung der nächsten Aufgabenpakete				
• Es wurden Aufgabenpakete definiert, die im Anschluss an das Meeting bearbeitet werden:				
○ Entwurf und Implementierung der Java-Objekte für die TENs und benötigten Hilfsklassen		M		Hr. Nassenstein
○ Anforderungsanalyse des Zugriffsbedarf der Activities auf die Datenschicht		M		Fr. Gilles
○ Entwurf und Implementierung der Services (Klassen, die die Schnittstelle zwischen Activities und Data Layer abbilden)		M		Hr. Nassenstein
○ Erstellung eines Set an Mockdaten, um die Entwicklung der Activities zu vereinfachen		M		Hr. Nassenstein
○ Einrichtung der Git-Branches		M		Hr. Beilfuß

	Protokoll DataTeam-Meeting Projekt: Angry Nerds - TEN-Manager	Verfasser: Ruthild Gilles 20. November 2018 Seite 1 vom 1 Angry Nerds
--	--	--

Datum: 20.11.2018
Start: 21:00 Uhr
Ende: 22:40 Uhr

Teilnehmer:

- Joscha Nassenstein
- Ruthild Gilles
- Jan Beifuß

Verteiler: Teilnehmer

I = Information | M = Maßnahme | E = Entscheidung

Sachstand & Bewertung	I	M	E	Verantwortlicher
Vorbemerkung				
• Die einzelnen Teammitglieder des DataTeams haben zuvor die eingeteilten Klassen und Methoden implementiert	I			
Topic 1: Zusammenfassung der entwickelten Methoden und Klassen				
• Die Teammitglieder fassten ihre bisherigen Ergebnisse zusammen	I			
• Die anderen Teammitglieder gaben Feedback		M		
Topic 2: Update der Ergebnisse im Versionsverwaltungstool				
• Alle Branches der Teammitglieder wurden in den Master-Branch gemerged		M		
• Probleme beim Mergen wurden gemeinsam behoben				
Topic 3: Weiteres Vorgehen				
• Aufteilung der weiteren Aufgaben			I / E	Alle
• Abstimmung über Methodennamen				

	Protokoll Team-Meeting Projekt: Angry Nerds - TEN-Manager	Verfasser: Florian Rath 4. Dezember 2018 Seite 1 vom 1 Angry Nerds
--	--	---

Datum: 04.12.2018
Start: 13:00 Uhr
Ende: 13:50 Uhr

Teilnehmer:

- Fr. Fabia Schmid
- Fr. Ruthild Gilles
- Hr. Robin Menzel
- Hr. Florian Rath
- Hr. Sertan Cetin
- Hr. Jan Beifuß
- Hr. Joscha Nassenstein
- Hr. Yannick Rüttgers

Verteiler: Teilnehmer

I = Information | M = Maßnahme | E = Entscheidung

Sachstand & Bewertung	I	M	E	Verantwortlicher
Vorbemerkung				
• Datenbankteam steht kurz vor dem Abschluss	I			
Topic 1: Vorstellung der Datenbankzugriffe				
• Die Klassen und Methoden für den Datenbankzugriff wurden vorgestellt	I			
• Diskussion über den Aufbau der Klassen				
• Um die Modularisierung umzusetzen wird eine Klasse Create erstellt		M		Fr. Gilles
• Die Main-Activity gibt die TEN's als Objekte an die einzelnen Views weiter		E		Team

4.3 Projekttagebücher aller Teammitglieder

4.3.1 Projekttagebuch Jan Beilfuß

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) / Erarbeitung der Eigenschaften von TENs	50 min	05.09.2018
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Installation von LaTeX	60 min	10.09.2018
Erste Überlegungen zum Datenhandling/Datenmodell	20 min	11.09.2018
Erste Recherche Datenbanken in Android	30 min	11.09.2018
Erstellung Datenbankmodell für relationale Datenbank	50 min	15.09.2018
Team-Meeting Projektteam	40 min	22.09.2018
Weitere Recherche Datenbanken mit Android (Dokumentenbasierte Datenbanken)	60 min	22.09.2018
Recherche Couchbase (inkl. Dependency Injection)	60 min	06.10.2018
Planung Data Layer Architektur	20 min	22.10.2018
Anlegen Packagestruktur Struktur (Service/Repository)	10 min	22.10.2018
Teammeeting Data-Team	60 min	26.10.2018
Git: Anlegen Branches für Joscha und Ruthild	10 min	26.10.2018
Git Versuch IDEA-Files aus Repository zu entfernen (gescheitert)	60 min	27.10.2018
Telefonat mit Ruthild: Unterstützung bei der IDEA und Git-Einrichtung	120 min	27.10.2018
Erstellung Protokoll Team-Meeting Data-Team vom 26.10.2018	20 min	30.10.2018
Erstellung des Projekttagebuchs	20 min	30.10.2018
Teammeeting Data-Team	100 min	20.11.2018
Couchbase - Experimente (Nested Arrays und Objects)	120 min	23.11.2018
Couchbase - Einarbeitung Object (De)serialization with Jackson	40 min	13.12.2018

Git Repository neu aufsetzen (Merger und neu anlegen)	180 min	01.12.2018
Erstellung Grafik Softwarearchitektur	30 min	06.12.2018
Couchbase Recherche - Views und Queries	20 min	17.12.2018
Erstellung des Projekttagebuchs	10 min	19.12.2018
Implementierung der ersten schreibenden Zugriffe auf die Datenbank (inklusive Converter)	160 min	07.01.2019
Implementierung der Deserialisierung/Serialisierung	150 min	08.01.2019
Optimierung des Jackson-Einsatzes mit @JsonIgnore	20 min	09.01.2019
Implementierung der Einzelzugriffe auf die Datenbank	100 min	11.01.2019
Implementierung der Query getAllTENS + Testumgebung	70 min	12.01.2019
Optimierung Struktur und Bugfixing Repository	50 min	13.01.2019
Implementierung: Speicherung der Bilder als BLOB in der Datenbank	110 min	15.01.2019
Implementierung asynchrone Ladelogik in Note zum Laden von Bildern	160 min	16.01.2019
Debugging Bilderspeicherung inkl. Evaluation alternativer Speichermöglichkeiten	60 min	17.01.2019
Erste Strukturänderungen in Note: Ladeprozess der Note	120 min	18.01.2019
Unterstützung git bei Todo Leuten	60 min	19.01.2019
Implementierung FileRepository	180 min	19.01.2019
Debugging FileRepository	60 min	20.01.2019
Erneute Anpassung Repositorystruktur	50 min	21.01.2019
Bugfixing und Unterstützung Yannick Rüttgers beim Laden der Bilder	40 min	25.01.2019
Bugfixing getAllTENs Query	20 min	25.01.2019
Festlegung von BundleKeys für TEN-Objekte	10 min	25.01.2019
Unterstützung git bei Todo Leuten	80 min	26.01.2019
Reorganisation Note Data	170 min	26.01.2019
Reorganisation Note ApplicationLogic und Data	210 min	27.01.2019
Umstellung des Bilderhandlings in Note auf den Tablet RAM	110 min	28.01.2019
Weitergehende Speicherplatzoptimierung und damit einhergehendes Bugfixing	130 min	29.01.2019

Beschreibung des Projektverlaufs

Implementierung der Bildkompression bei großen Galerieimporten	60 min	30.01.2019
Tabletbeschaffung	10 min	31.01.2019
Implementierung Configuration Change beim ImageOverlay	80 min	02.02.2019
Testing für Activities auf Tablet	100 min	02.02.2019
Dokumentation des Projektes	120 min	02.02.2019
Testing der App auf dem Tablet	50 min	03.02.2019
Dokumentation des Projektes	480 min	03.02.2019

Summe in Minuten: 4380

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
Projekttagebuch in Latex führen	30 min	19.12.2018
Entwicklung der Layouts für die Overview	130 min	02.01.2019
Aufteilung der Ausarbeitung	30 min	02.01.2019

Anpassung der OverviewActivity-Layouts	70 min	03.01.2019
Implementierung der Funktion OverviewClickListener	80 min	03.01.2019
Implementierung der Funktion OverviewHeaderClickListener	60 min	03.01.2019
Entwicklung des Layouts für das Bedienleisten-Fragment	70 min	04.01.2019
Implementierung der Funktion OverviewFragmentClickListener	100 min	12.01.2019
Implementierung der Funktion OverviewFragmentLongClickListener	70 min	12.01.2019
Meilenstein Umplanung	20 min	15.01.2019
Layoutanpassung Note Image	70 min	18.01.2019
Erinnerung an die Erstellung der Kapitel der Ausarbeitung	10 min	23.01.2019
Layoutanpassungen OverviewActivity	120 min	26.01.2019
Erstellung des GANTT-Diagramms	60 min	01.02.2019
Erstellung der Aufgaben-Tabellen in Latex	110 min	02.02.2019
Anpassung des Event-Layouts	60 min	02.02.2019
Anpassung des ToDo-Layouts	80 min	02.02.2019
Anpassung der Buttons zum Filtern	90 min	03.02.2019
Dokumentation	420 min	03.02.2019
Projekttagebuch in Latex führen	30 min	03.02.2019

Summe in Minuten: 2620

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	240 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	280 min	03.02.2019

Summe in Minuten: 2530

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

Beschreibung des Projektverlaufs

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	240 min	02.02.2019
Latex - Ausarbeitungen der anderen einfügen	240 min	03.02.2019
Projekttagebuch ausfüllen	20 min	03.02.2019
Latex - Projekttagebücher einfügen	120 min	03.02.2019

Summe in Minuten: 3370

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	180 min	03.01.2019
Implementierung Taskadapter (Schnittstelle zwischen Gui und Data)	300 min	04.01.2019
Bugfixes für Todo Tasks Listview	120 min	05.01.2019
Emulator Problembehebung (gescheitert)	120 min	21.01.2019
Implementierung Todo Data und Progresstext	200 min	28.01.2019
Implementierung ShareFunktion Todo	180 min	02.02.2019
Diverse Bugfixes in Todo Funktionen	210 min	03.02.2019
Dokumentation	180 min	03.02.2019

Summe in Minuten: 2720

4.3.8 Projekttagebuch Yannick Rüttgers

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
Besprechung der in Aufgabenteilung entstandenen Ergebnisse	40 min	05.09.2018
Wahl der Latex Distribution	20 min	10.09.2018
Installation von Latex	120 min	10.09.2018
Festlegung des Vorgehensmodells	20 min	11.09.2018
Meeting zur Besprechung der Ergebnisse aus dem Design- und Daten-Team	60 min	22.09.2018
Recherche Technologie "Fragments"	90 min	07.10.2018
Entwicklung einer Testapplikation mit Fragments, Beschäftigung mit dem Lifecycle derer	120 min	08.10.2018
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
Provisorische Startseite fürs Development	60 min	20.10.2018
Kommunikation mit Data Team im Bezug auf Objekte und Übergabe an Activities	30 min	23.10.2018
Besprechung der Layouts	30 min	27.10.2018
Erstellung des Projekttagebuchs in Latex	40 min	29.10.2018
Weitere Beschäftigung mit Latex	90 min	29.10.2018
Teammeeting	50 min	04.12.2018
Beratung bei der Erstellung weiterer Layouts	30 min	12.12.2018
Erstellung von Coding Conventions	50 min	14.12.2018
Hinzufügen der Layouts in das Projekt, Anpassen der Layouts an Coding Conventions	140 min	18.12.2018
Klassenstruktur der Activity Overview planen	200 min	28.12.2018
Erschaffen der Grundstruktur für die Hauptactivity	180 min	28.12.2018
Implementierung des Datenmodells der Activity	50 min	30.12.2018

Implementierung der GUI und des Controllers der Activity	70 min	30.12.2018
Schaffung einer Verbindung zwischen den Schichten	60 min	30.12.2018
Korrektur der Mockdaten	20 min	30.12.2018
Vorbereitung eines Beispielfragments	30 min	31.12.2018
Hinzufügen eines Fragments in die Container der Übersicht	120 min	31.12.2018
Planung der Datenübertragung zwischen Fragment und Activity	110 min	01.01.2019
Implementierung von Bundles in den Datenklassen	60 min	01.01.2019
Implementierung des Notefragments	120 min	02.01.2019
Planung und Entwicklung von Superklassen für die Fragments	160 min	02.01.2019
Bugfixing bezogen auf das Neuerstellen von Fragments bei Verändern der Konfiguration	120 min	03.01.2019
Implementierung von Fragmenthelperklassen	120 min	03.01.2019
Update des OverviewLayouts	60 min	03.01.2019
Planung der Löschfunktionalität	70 min	03.01.2019
Implementation der Löschfunktionalität	120 min	03.01.2019
Implementierung des Markierungsprozesses	40 min	04.01.2019
Implementierung von Helferklassen für den Löschprozess	60 min	04.01.2019
Planung und Implementierung des Layoutwechsels bei Veränderung der Orientierung	80 min	04.01.2019
Implementation Event Fragment	60 min	04.01.2019
Implementierung Todofragment ohne Tasks	60 min	04.01.2019
Implementierung der Tasks	80 min	05.01.2019
Testing der Fragments	60 min	05.01.2019
Bugfixing Löschprozess	80 min	25.01.2019
Planung und Implementierung der Öffnung von TENs	60 min	25.01.2019
Bugfixing Adresse in Event	20 min	25.01.2019
Implementierung des ImageFragments	210 min	25.01.2019
Implementierung einer Highlightfunktionalität für Leiste am unteren Bildschirmrand	20 min	26.01.2019
Anpassen von Bundleids an Konstanten	20 min	26.01.2019

Beschreibung des Projektverlaufs

Bugfixing Scrollview	60 min	26.01.2019
Bugfixing Jahr im Eventfragment	20 min	26.01.2019
Anpassung der Sprache	20 min	26.01.2019
Hinzufügen eines adaptiven Layouts je nach Bildschirmgröße	90 min	27.01.2019
Planung und Implementierung einer Suchfunktion	220 min	03.02.2019
Hinzufügen weiterer Kommentare	50 min	03.02.2019
Bugfixing Todoicon	10 min	03.02.2019
Dokumentation des Projekts	360 min	03.02.2019

Summe in Minuten: 4540

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 Probleme war die Zeit, da die Entwicklung immer mal wieder unterbrochen wurde, kam 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 Jan Beilfuß

Im Projektverlauf traten einige Probleme auf. Während der Arbeit im Datateam waren die Zwischenergebnisse zu Implementierungsbeginn schwierig zu testen, da diese zum Großteil implementiert wurden, bevor man eine Anbindung an die Activities hatte. Die Suche nach möglichen Fehlern wurde hier hauptsächlich über den Logcat durchgeführt. Dennoch fielen noch Fehler bei der Anbindung der Activities entdeckt und konnten demzufolge erst zu dem Zeitpunkt behoben werden.

Weiterhin hat der Wechsel vom privaten Android-Smarhpone auf das FHDW-eigene Tablet Probleme bereitet. Insbesondere die Kombination aus dem geringen Arbeitsspeicher auf dem Tablet und einem nicht funktionierenden Profiler (dieser soll den Arbeitsspeicherverbrauch loggen) in Android Studio hat das Optimieren des Speicherbedarfs in der Note-Activity erschwert.

4.4.4 Probleme von Joscha Nassenstein

Zu Beginn der Entwicklung benötigte es sehr viel Zeit, durch das Implementieren verschiedener Möglichkeiten, beispielsweise für die Stichwortliste, die beste Lösung zu finden. Da ich stets mein eigenes Android-Smartphone, welches zurzeit auf API 29 läuft, für das Testen verwendete, implementierte ich teilweise Lösungen, welche so auf dem Tablet der Hochschule (API 19) nicht in dem Maße funktionierten, wie es auf meinem Gerät der Fall war.

Vor allem im Umgang mit Textfeldern und beim Überschreiben einiger onTouch-Funktionen kam es dabei zu anderen Ergebnissen, als dies vorher der Fall war. Da in dem AndroidStudio-Projekt API-Level 19 bereits voreingestellt war, wurde direkt ein Hinweis angezeigt, falls man eine Funktion nutzen wollte, welche auf dem Level nicht unterstützt wird. Ein weiteres Beispiel ist die Android-eigene Kamerafunktion, welche vor der Android Version Marshmallow (API 23), in welcher spezielle Laufzeit-Berechtigungen eingeführt wurden, eine Berechtigung zum Schreiben und Lesen der Datei mitgegeben werden muss. Auch dies fiel erst auf, als die App auf dem Tablet getestet wurde. Insgesamt mussten nach den Tests einige kleinere Anpassungen vorgenommen werden.

Eine weitere Herausforderung war die Abstimmung der Programmierungsaufgaben, da diese Activity unter zwei Programmierern aufgeteilt wurde. Dies erforderte, falls

in der Zwischenzeit Änderungen dazugekommen waren, ein bisschen Einarbeitungszeit, bevor man weiterprogrammieren konnte. Insgesamt ist das Ergebnis und die Performance jedoch sehr zufriedenstellend und durch die Aufteilung konnten sehr viele Aspekte der Activity optimiert werden. Durch Benennungskonventionen und eine neue Paketstruktur sowie die intensive Abstimmung konnte hier ein gutes Ergebnis entstehen.

4.4.5 Probleme von Ruthild Gilles

Während des Projektes kam es zu einigen Komplikationen. Dies lag zum einen an fehlender Erfahrung beim Projektmanagement und der Entwicklung, aber auch an fehlenden Kenntnissen von Versionierung über Git und Dokumentation in Latex.

Für mich persönlich war es eine Herausforderung zwischen den anderen Teammitgliedern, die deutlich besser bzw. schneller entwickeln können als ich, nicht unterzugehen. Mir wurden aufgrund meiner wenigen Erfahrung in der Programmierung von Java zu Beginn nur wenige Aufgaben bezüglich Entwicklung zugeteilt. Da andere Projektmitglieder weniger implementieren wollten, wurden ihnen Aufgaben im Projektmanagement übertragen, allerdings blieben nicht viele Aufgaben offen, an denen ich mich beteiligen könnte.

Da die Planungsphase zugegebenermaßen doch recht kurz ausfiel, musste ich während der Implementierung meiner Klassen ausprobieren und öfters Änderungen durchführen. Besonders herausfordernd war hierbei auf die möglichen Anforderungen einzugehen, die während der Entwicklung der Activities auftreten würden. Besonders die Verantwortlichen der Activities erwarteten von mir, dass ich Methoden zum getten und setzen von TEN-Objekten implementierte. Allerdings fehlten mir während der ersten Implementation die genauen Anforderungen der Activities, da die jeweiligen Gruppen selbst noch nicht ins Detail geplant hatten, welche Methoden sie benötigen würden. So kam es dazu, dass ich meine wenigen Methoden erneut und erneut implementierte oder Änderungen durchführte.

Ähnlich verlief es auf der Seite der Datenbank. Die Auswahl der Datenbank wurde zwar bereits sehr früh getroffen, jedoch fehlte der verantwortlichen Person zu Beginn das Wissen und auch von dieser Seite hatte ich keine klaren Anforderungen. Deswegen musste ich auch hier häufige Anpassungen vornehmen und es wurde mit der Zeit

auch immer mehr Logik meiner Klassen in die Klassen des Repositories ausgelagert, sodass mein Programmienteil im Endeffekt deutlich kleiner aussieht, als er eigentlich war.

Nachdem ich angemerkt hatte, dass ich Zeit übrig hatte und noch in den Activities helfen könnte, wurde mir jedoch mitgeteilt, dass das Erklären der bisherigen Logik und das Beschreiben der benötigten Funktionalitäten länger dauern würde, als die Implementierung durch bereits eingearbeitete Teammitglieder.

Letztendlich wurde mir die Aufgabe des Latex-Verantwortlichen übertragen und so saß ich die letzte Woche vor Abgabe der Dokumentation an der Einarbeitung, Erstellung und Zusammenstellung der schriftlichen Ausarbeitung in Latex.

4.4.6 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.

4.4.7 Probleme von Yannick Rüttgers

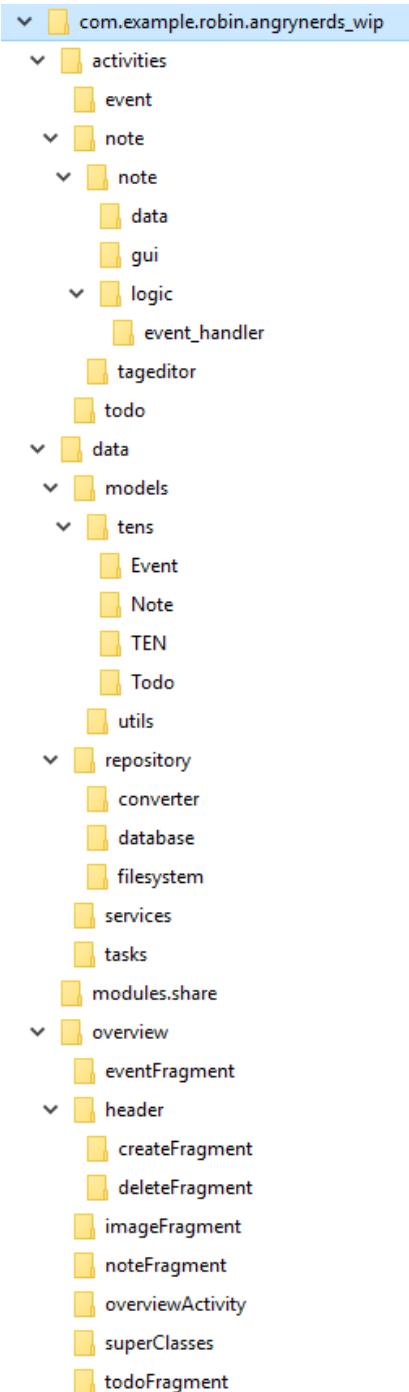
Die Probleme, denen während des Projektes auftraten, lagen hauptsächlich an der genutzten Entwicklungsumgebung, bzw. der genutzten Software.

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)

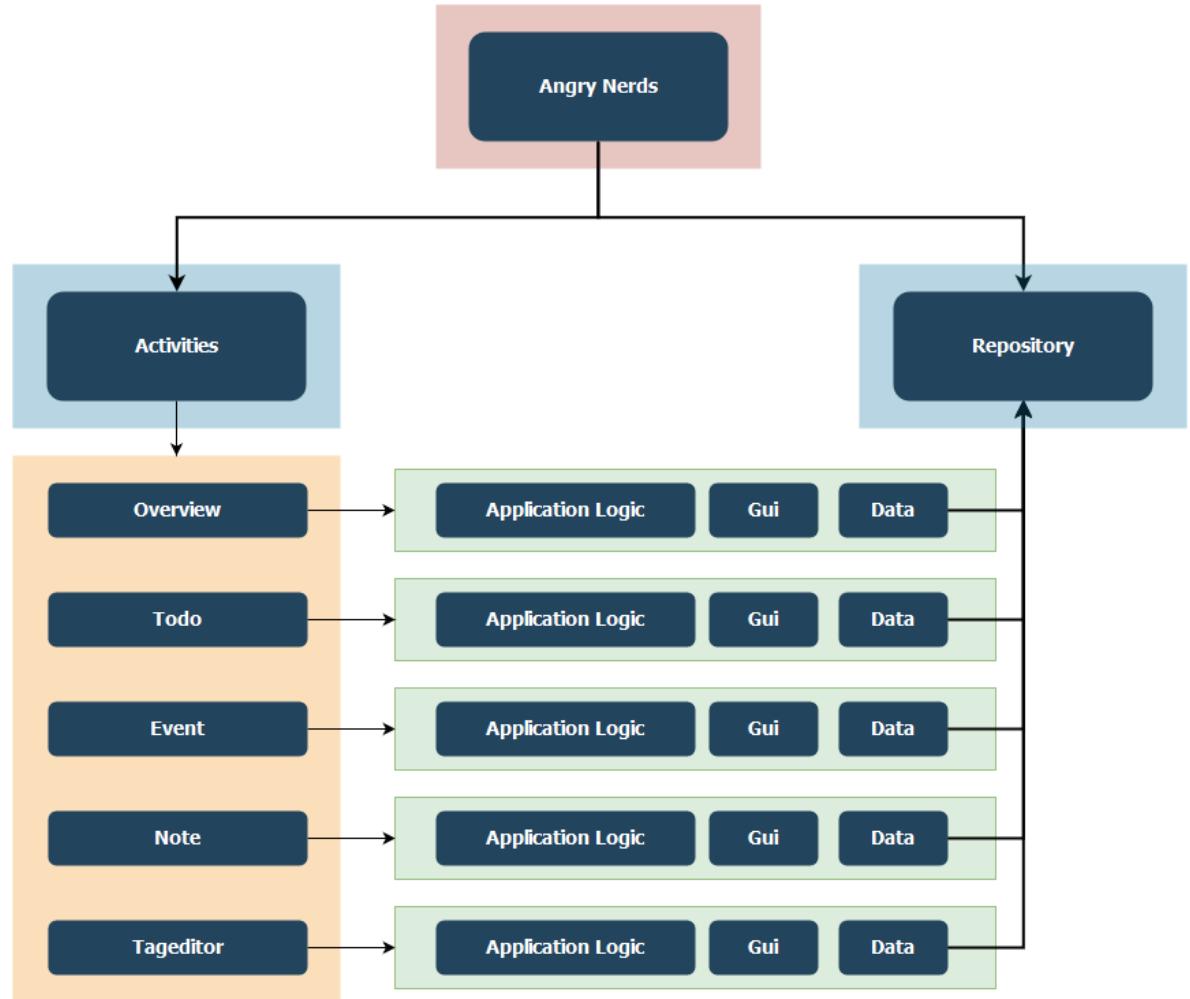
Die entwickelte App ist innerhalb des Paketes com.example.robin.angrynerds-wip in mehrere Klassen unterteilt. Zu jeder Activity gehören mehrere Klassen. Um die Übersichtlichkeit zu steigern, wurden die Klassen und Activities in Ordnern untergebracht. Die gesamte Ordnerstruktur sieht wie folgt aus:

Abbildung 11: Paketstruktur

Quelle: Erstellt von Sertan Cetin

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

Abbildung 12: Paketstruktur



Quelle: Erstellt von Sertan Cetin

5.2 Dokumentation der Activities

5.2.1 Main Activity

5.2.1.1 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.

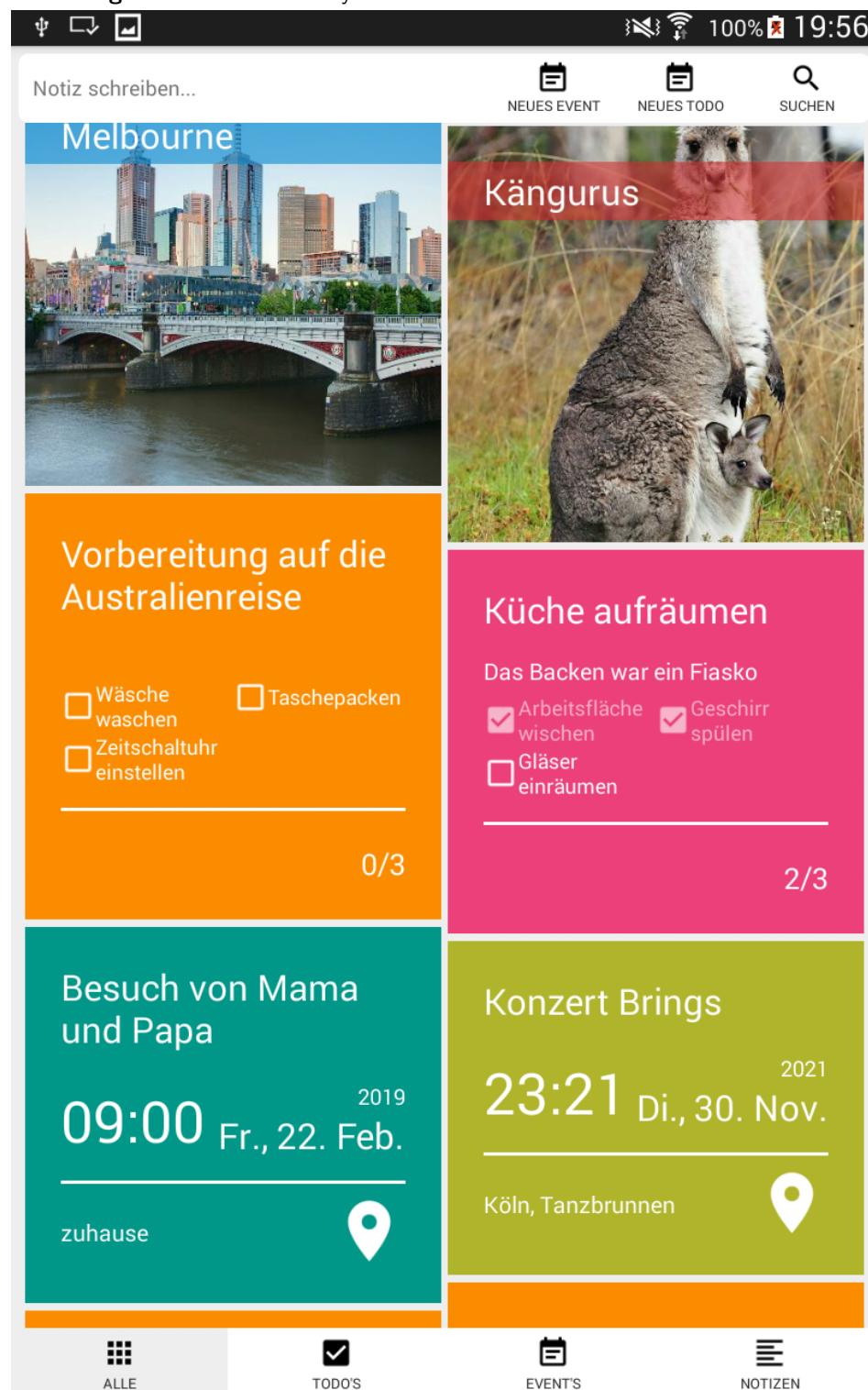
Weitere Infos und Screenshots zum Layout der Applikation sind im nachfolgenden Kapitel zu finden.

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.

5.2.1.2 Layouts, Screenshots (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 (vgl. Abbildung Overview Activity).

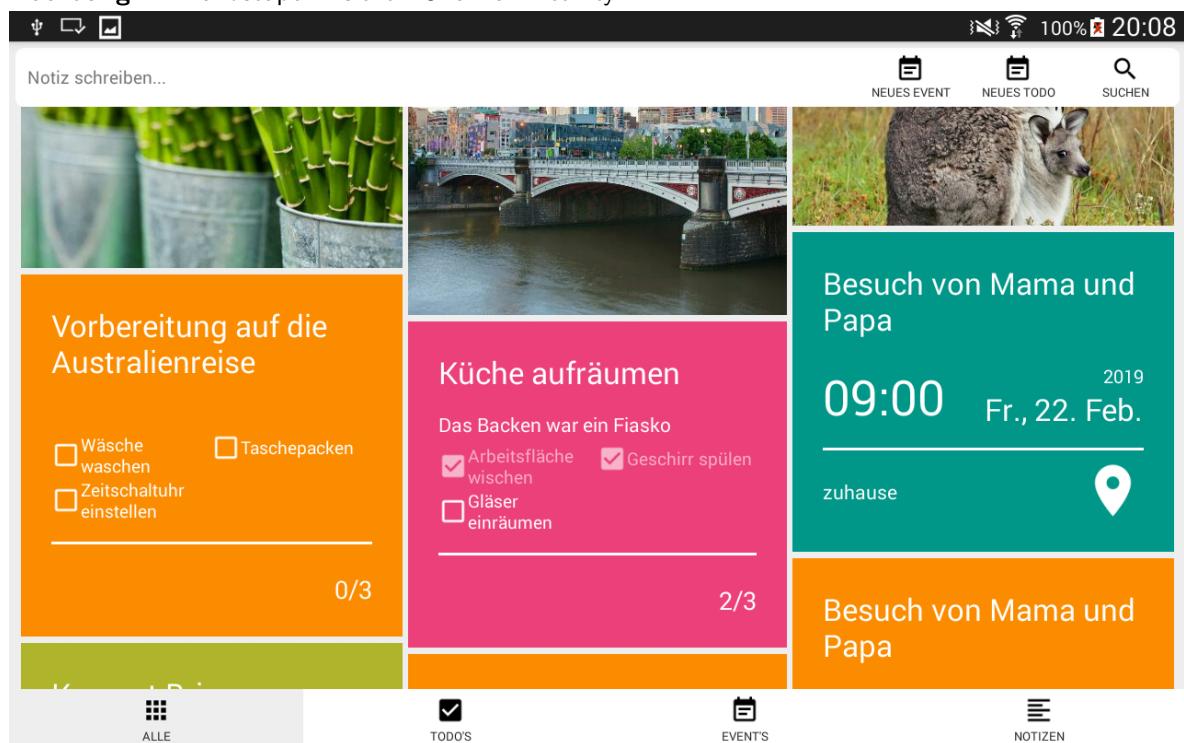
Die obere Leiste ist ein RelativeLayout, welches eine TextView und drei Button beinhaltet. Diese ist ein gesondertes Fragment, damit der Wechsel zu einer anderen Leiste, z.B. bei dem Löschvorgang, vereinfacht wird. Darunter ist eine ScrollView, die es ermöglicht, dass durch die Fragments gescrollt werden kann. In der ScrollView befinden sich LinearLayouts, die mit den Fragments befüllt werden können. Die untere Leiste besteht abschließend aus einem LinearLayout, welches horizontal ausgerichtet ist und vier Buttons beinhaltet, mit denen gefiltert werden kann.

Abbildung 13: Overview Activity

Quelle: Screenshot aus der Benutzeroberfläche

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 (vgl. Abbildung Landscape-Ansicht Overview Activity).

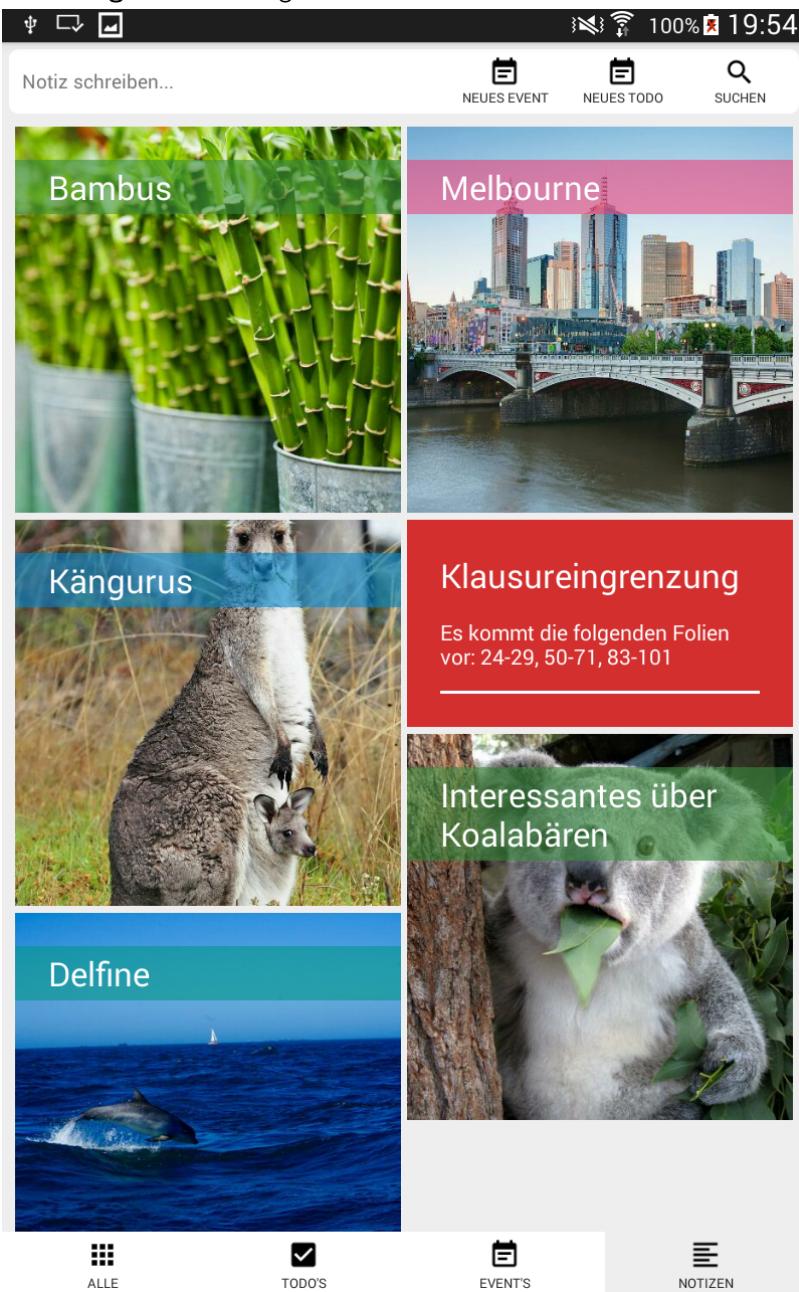
Abbildung 14: Landscape Ansicht - Overview Activity



Quelle: Screenshot aus der Benutzeroberfläche

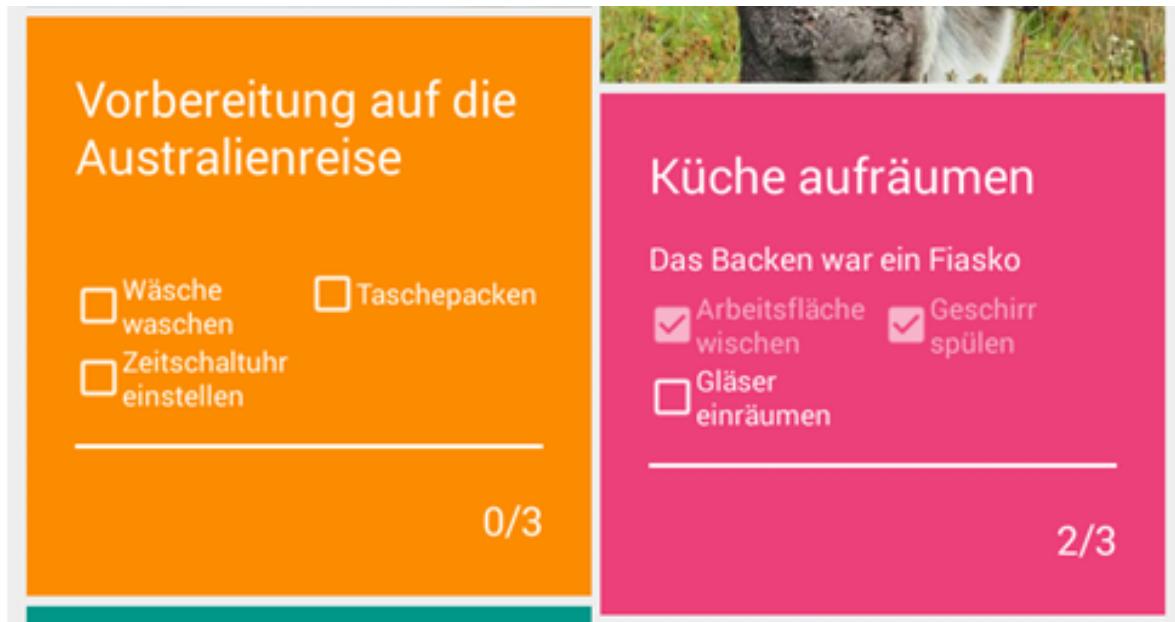
Die drei verschiedenen Arten von Fragments sind Notes, ToDo und Event.

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 (vgl. Abbildung Note Fragments).

Abbildung 15: Note Fragments

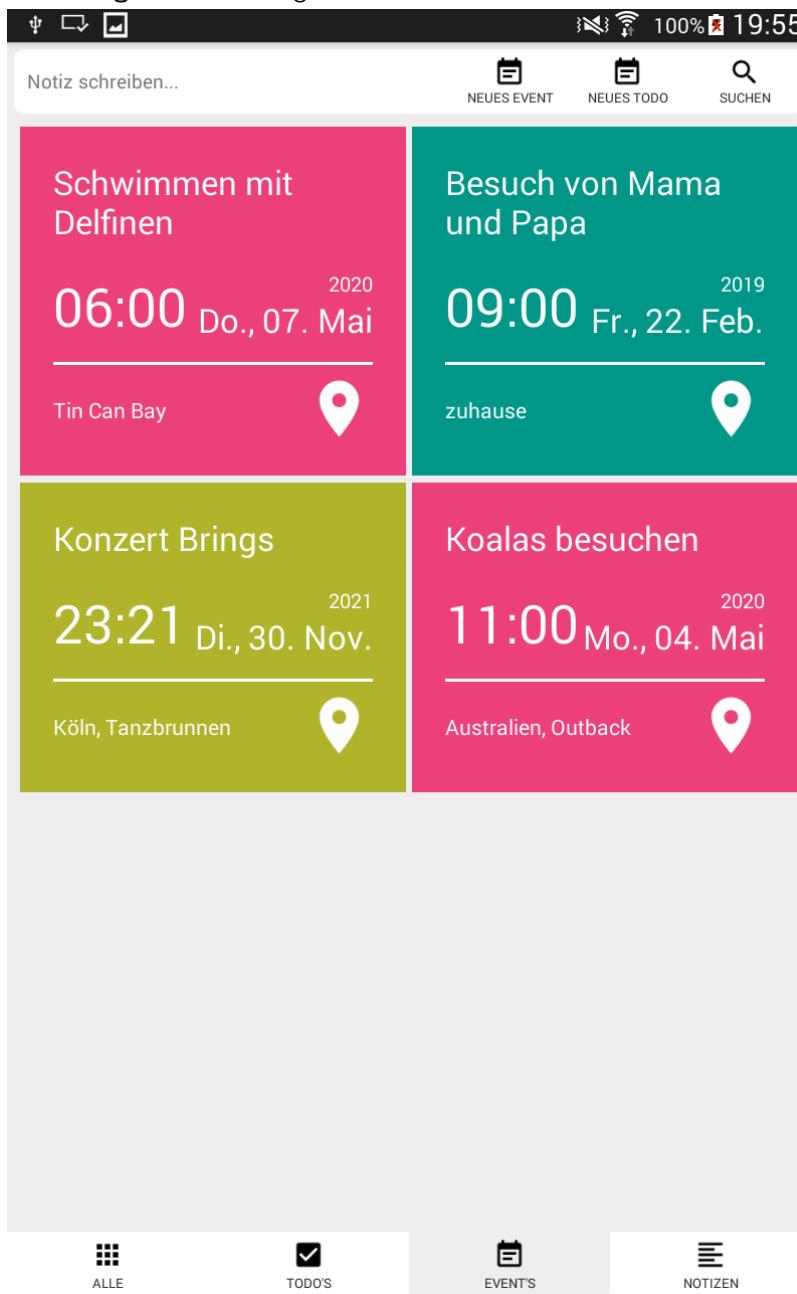
Quelle: Screenshot aus der Benutzeroberfläche

Die Fragments für die ToDos, zeigen 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 (vgl. Abbildung ToDo Fragments).

Abbildung 16: Todo Fragments

Quelle: Screenshot aus der Benutzeroberfläche

Die Event Fragmente bestehen aus der Überschrift, dem Datum und der Uhrzeit, sowie aus einem Ort, der angegeben werden kann (vgl. Abbildung Event Fragments).

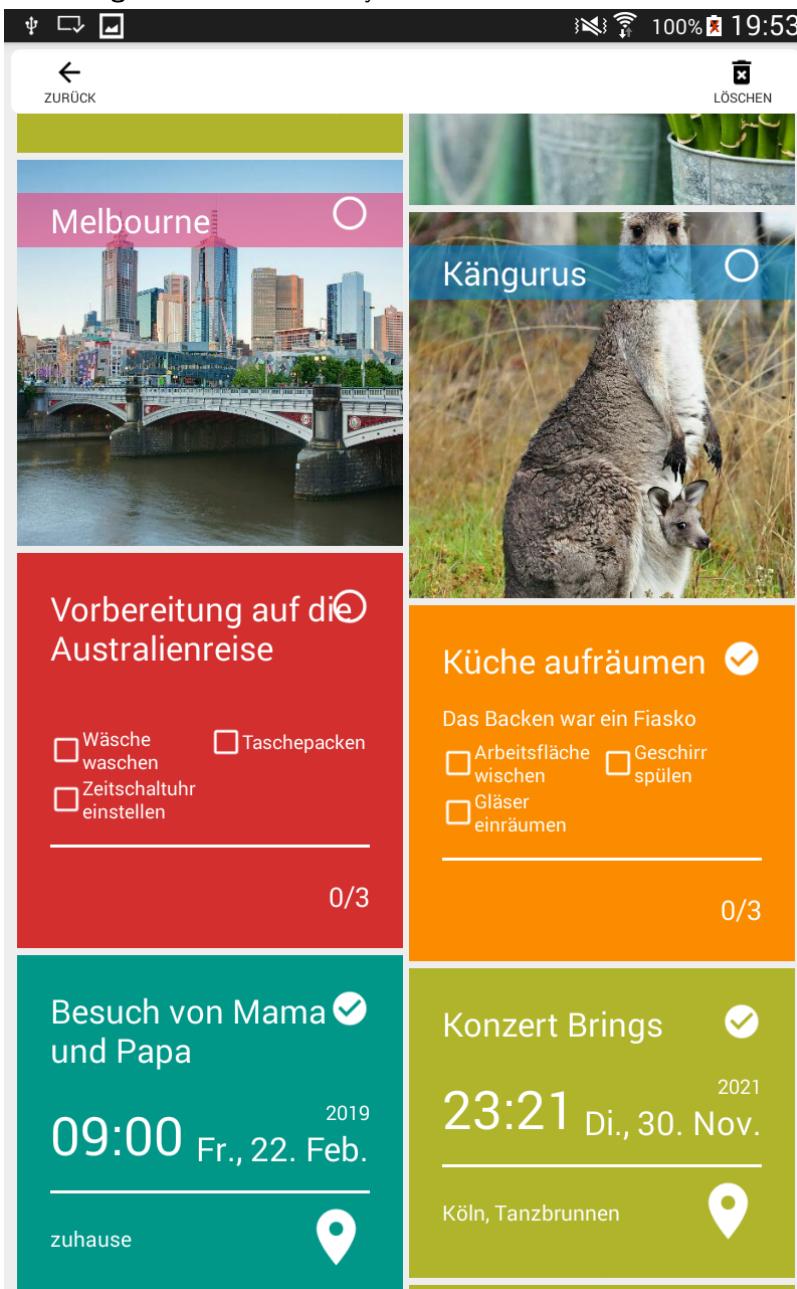
Abbildung 17: Event Fragments

Quelle: Screenshot aus der Benutzeroberfläche

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. Dies ist dadurch möglich, dass die obere Leiste ein Fragment ist. Zusätzlich kann in dieser Ansicht nun auf die verschiedenen Fragments geklickt

werden, um diese zu markieren und mehrere gleichzeitig zu löschen. Markierungen werden mittels einem Hacken gekennzeichnet. Diese Ansicht kann durch den Zurück-Pfeil verlassen werden (vgl. Abbildung Overview Activity Löschen).

Abbildung 18: Overview Activity - Löschen

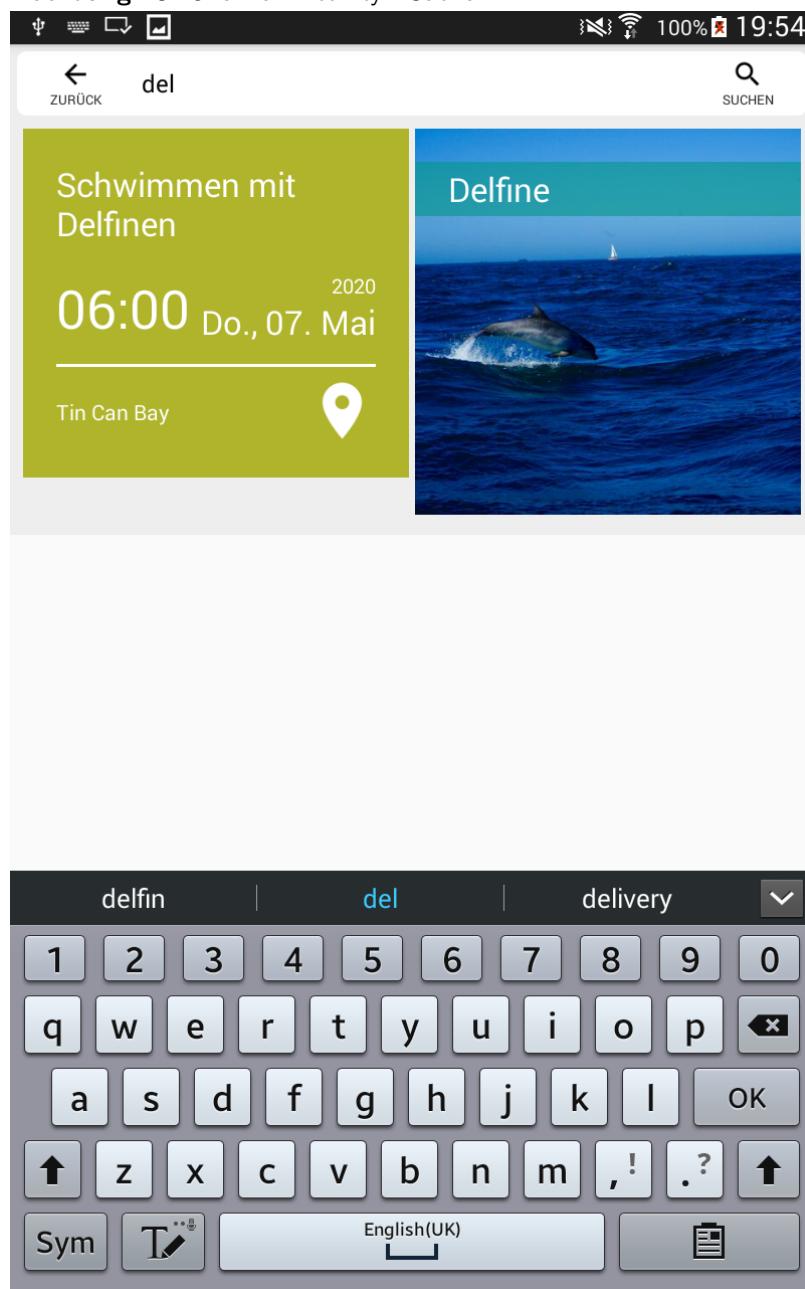


Quelle: Screenshot aus der Benutzeroberfläche

Auch bei der Suchfunktion wird die obere Leiste angepasst. Auch hier möglich durch

das Fragment. Dabei kann nun ein beliebiges Wort eingegeben werden, welches in den verschiedenen Einträgen gesucht wird. Die gefundenen Einträge werden daraufhin angezeigt. Um die Suche zu beenden kann der Zurück-Pfeil verwendet werden (vgl. Abbildung Overview Activity Suchen).

Abbildung 19: Overview Activity - Suchen



Quelle: Screenshot aus der Benutzeroberfläche

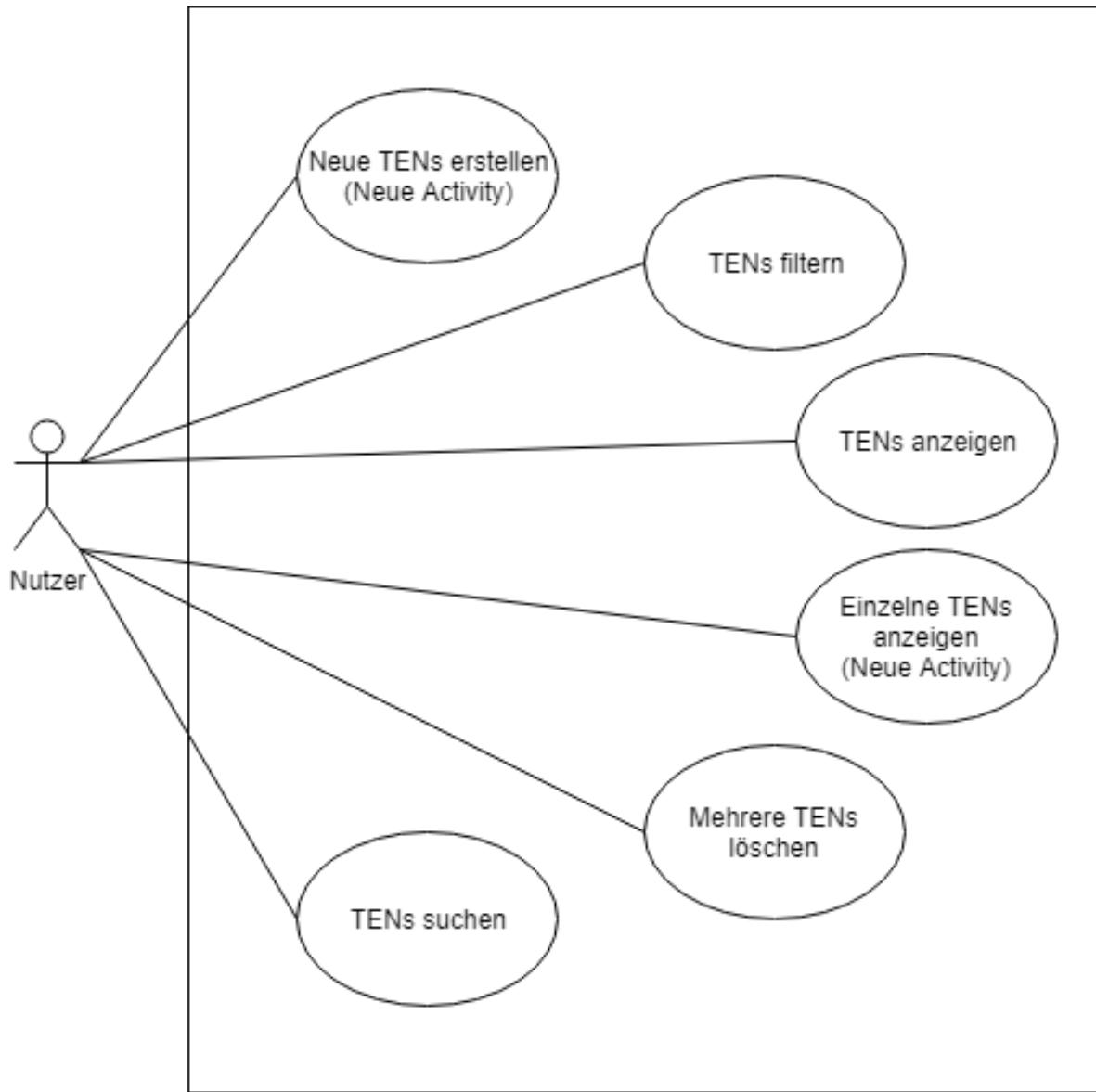
5.2.1.3 Use Case (Yannick Rüttgers) Wie bereits geschrieben, dient diese Activity dem Nutzer als Einstiegspunkt in die Applikation. Hier kann er einige Aktionen durchführen.

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 20: Usecase Diagramm der Overview

Quelle: Erstellt von Yannick Rüttgers

5.2.1.4 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.

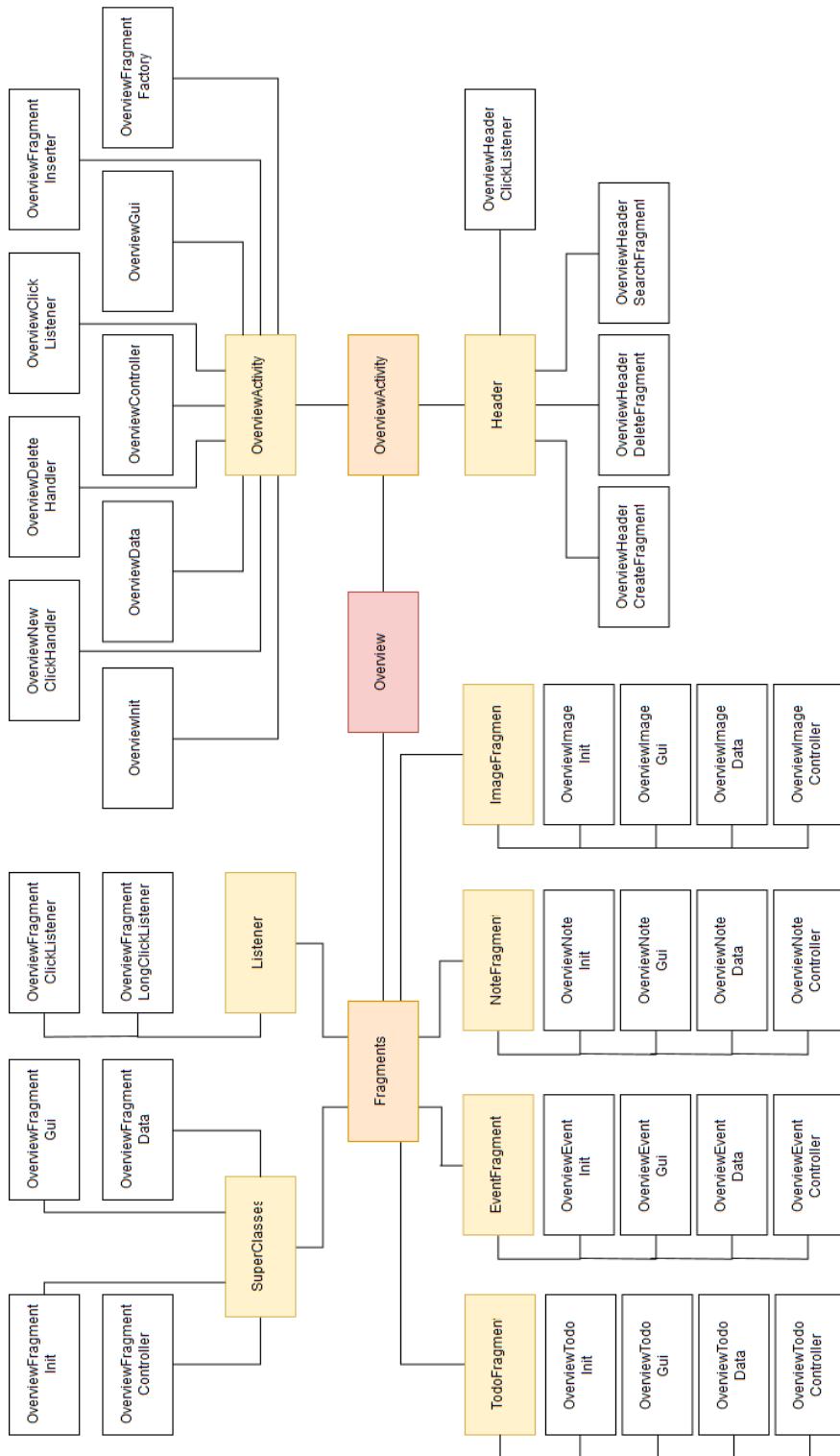
Zusätzlich hält jedes Fragment zwei Informationen über sich. Dies ist zum einen der Löschzustand, über den das Fragment steuern kann, ob es zum Löschen markiert werden kann oder nicht. Zum anderen hält das Fragment die Information, ob es markiert ist, um dies im Falle einer Löschung angeben zu können.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

5.2.1.5 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 21: 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 initiiert.

-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 initiiert. 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()

Läßt 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.

Wichtige 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()

Sammelt 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 Checkboxicons 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 Allgemein 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 Todo-Fragments 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 Attributen 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 Attributen 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 lädt 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 Attributen 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

5.2.2.1 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

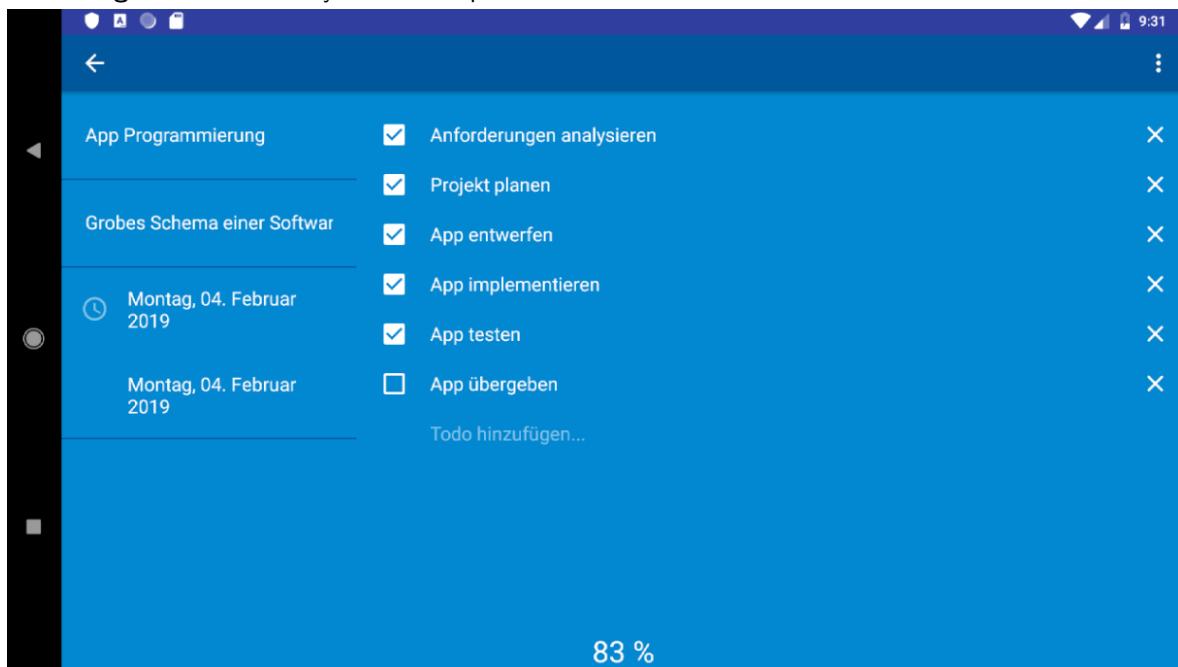
und die Todo verfrüht abzuschließen. Die App macht keine Obergrenze für Tasks, um dem User keine Arbeits- bzw. Organisationsweise vorzuschreiben.

Der User hat auch die Möglichkeit eine Todo zu löschen.

5.2.2.2 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.

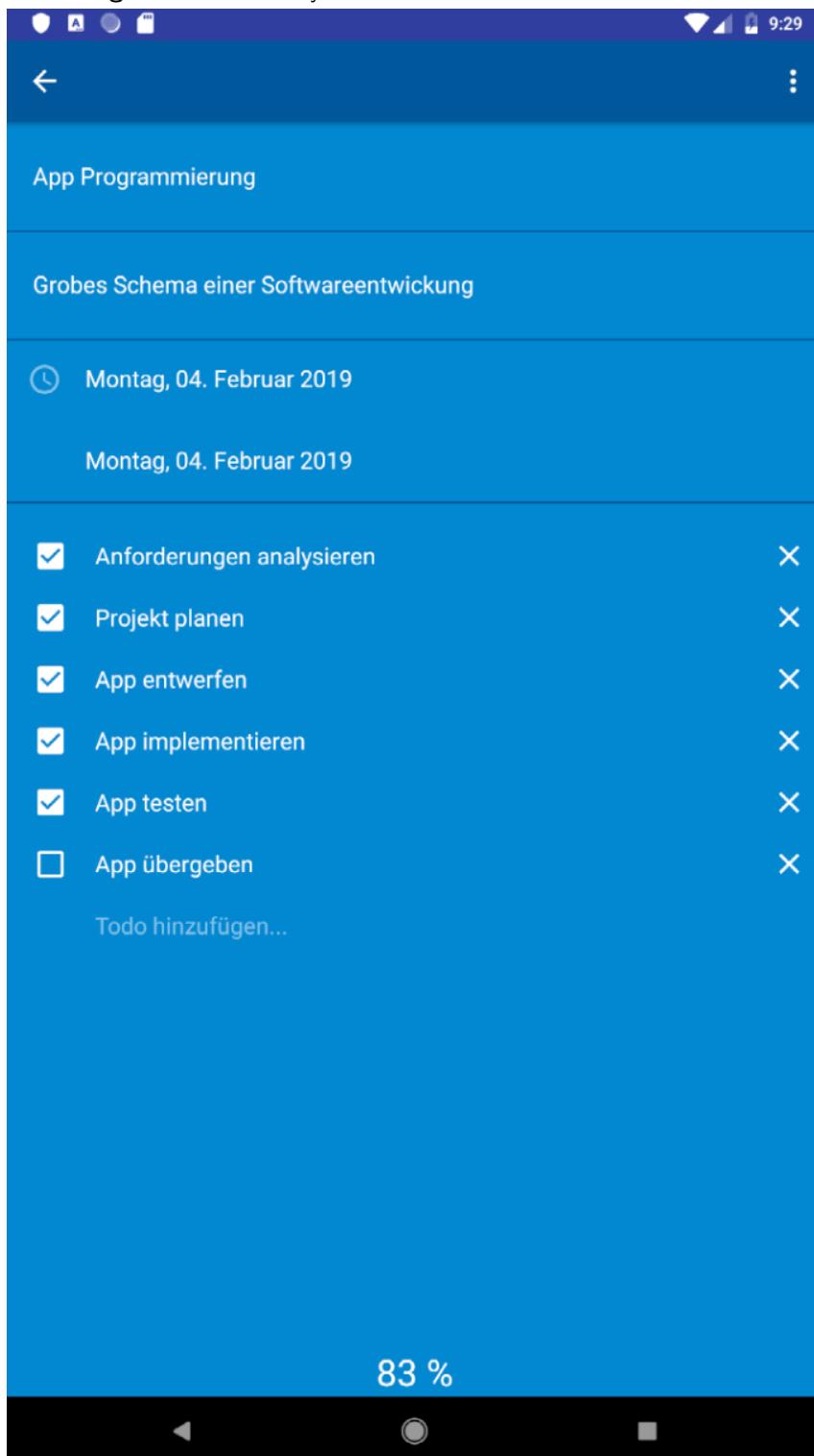
Oben ist eine Toolbar vorhanden, mit einem Zurück-Button, um zur Hauptübersicht zu gelangen, und einem Menü, in welchem die Todo gelöscht werden kann.

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

Abbildung 22: Todo Activity im Landscape-Modus

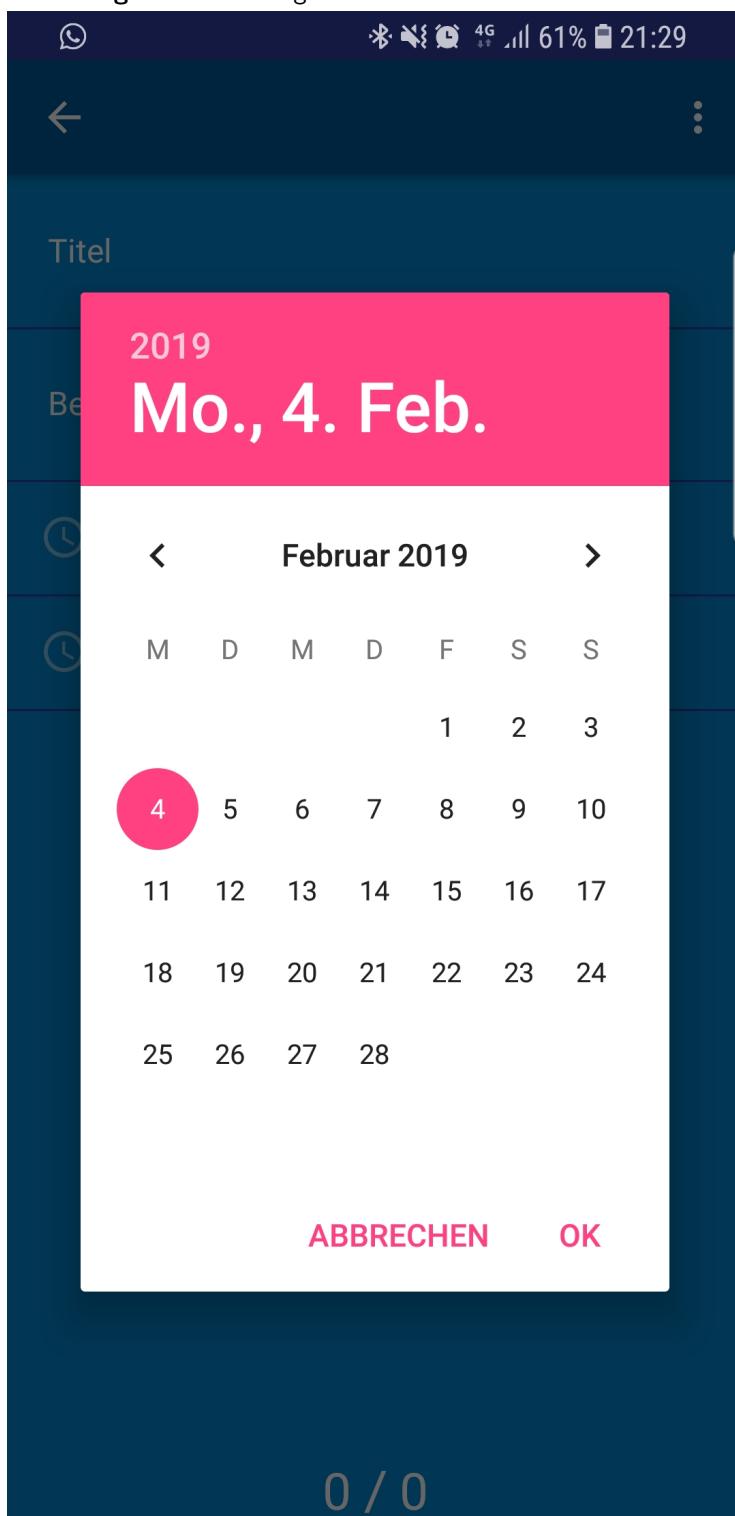
Quelle: Screenshot aus der Benutzeroberfläche

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

Abbildung 23: Todo Activity im Portrait-Modus

Quelle: Screenshot aus der Benutzeroberfläche

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 24: Datumeingabe

Quelle: Screenshot aus der Benutzeroberfläche

5.2.2.3 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.

5.2.2.4 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.

5.2.2.5 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 Trennlinien 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 onActivityResult(int requestCode, int resultCode, Intent data)

Diese Methode deckt den Fall ab, falls in die Activity zurückgekehrt wird.

Private void onDeleteButtonClicked(int id) Es wird dieser Methode eine ID übergeben. Diese ID entspricht der Aufgabe, bei der der Löschen-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 ClickListener getClickListener()

Gibt das ClickListener-Objekt zurück.

Public TouchListener getTouchListener()

Gibt das TouchListener-Objekt zurück.

Public CheckedChangeListener 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.

5.2.3 Event Activity

Dieses Kapitel beschreibt die Funktionsweise der Event-Ansicht innerhalb des TEN-Managers. Diese wird aufgerufen, wenn ein Event aus der Übersichtsseite angeklickt wird oder ein neues Event erstellt wird.

5.2.3.1 Aufgaben und Funktionen (Robin Menzel) Die *Event Activity* hat die Funktion eine Übersicht über ein Event zu geben, Bearbeitungen zu dem Event zu verarbeiten, oder ein neues Event zu erstellen und speichern. Möchte ein User eine neues Event erstellen, ist die Event-Activity bis auf das aktuelle Datum leer. In den Textfeldern stehen halbtransparente Hinweise, wie "Titel eingeben", die dem User Hinweise über den Inhalt geben. Wenn die Activity ein bereits vorhandenes Event anzeigt, kann durch ein Klick auf die jeweilige Konfiguration, eine Änderung durchgeführt werden.

Ganz oben ist eine Toolbar zu finden, welche zum einen eine Navigation zurück zur Übersichtsseite bietet und zum anderen ein Options/Drei-Punkt Menü beinhaltet. Durch einen Klick auf das 3-Punkt Menu der *Event Activity* kann das Event in Form von Text per E-Mail oder Chat-Applikation geteilt werden. Auch ein Export in eine andere Kalender-Applikation wird hier unterstützt. Die letzte Option bietet das Löschen des Events an.

Da ein Event immer ein Datum und eine Zeit hat, kann auch dieses in der Activity ausgewählt werden. Über einen Klick auf das Datum oder die Uhrzeit, öffnet sich ein Dialog Picker, über den sich die Uhrzeit oder das Datum auswählen lässt.

Bestimmte Events wie Geburtstage wiederholen sich in bestimmten Intervallen. Auch diese Events lassen sich in der Activity abbilden, in dem unter der Zeit auf (initialer Wert) Einmalig geklickt wird. Hier steht dem User die Option Einmalig, Täglich, Wöchentlich und Jährlich zur Verfügung. Angezeigt wird immer der Termin, der am nächsten liegt.

Sobald ein User mehr als einen Buchstaben in die Adresszeile des Events eingibt, leuchtet das Navigationsicon auf. Ein klick auf dieses öffnet sich Google Maps und versucht den eingegeben Ort zu finden.

Als letzte Option lassen sich Erinnerungen zu diesem Event einstellen. Dazu hat der User eine Wahl zwischen 1 Stunde vorher, 6 Stunden vorher, 1 Tag vorher und 1 Woche

vorher. Es lassen sich kein, ein, mehrere, oder alle Erinnerungen auswählen. Liegt der Zeitpunkt der Erinnerung in der Zukunft, so wird zu diesem Zeitpunkt eine Push-Benachrichtigung auf dem Gerät erscheinen, welche den Titel und die Uhrzeit des Events enthält.

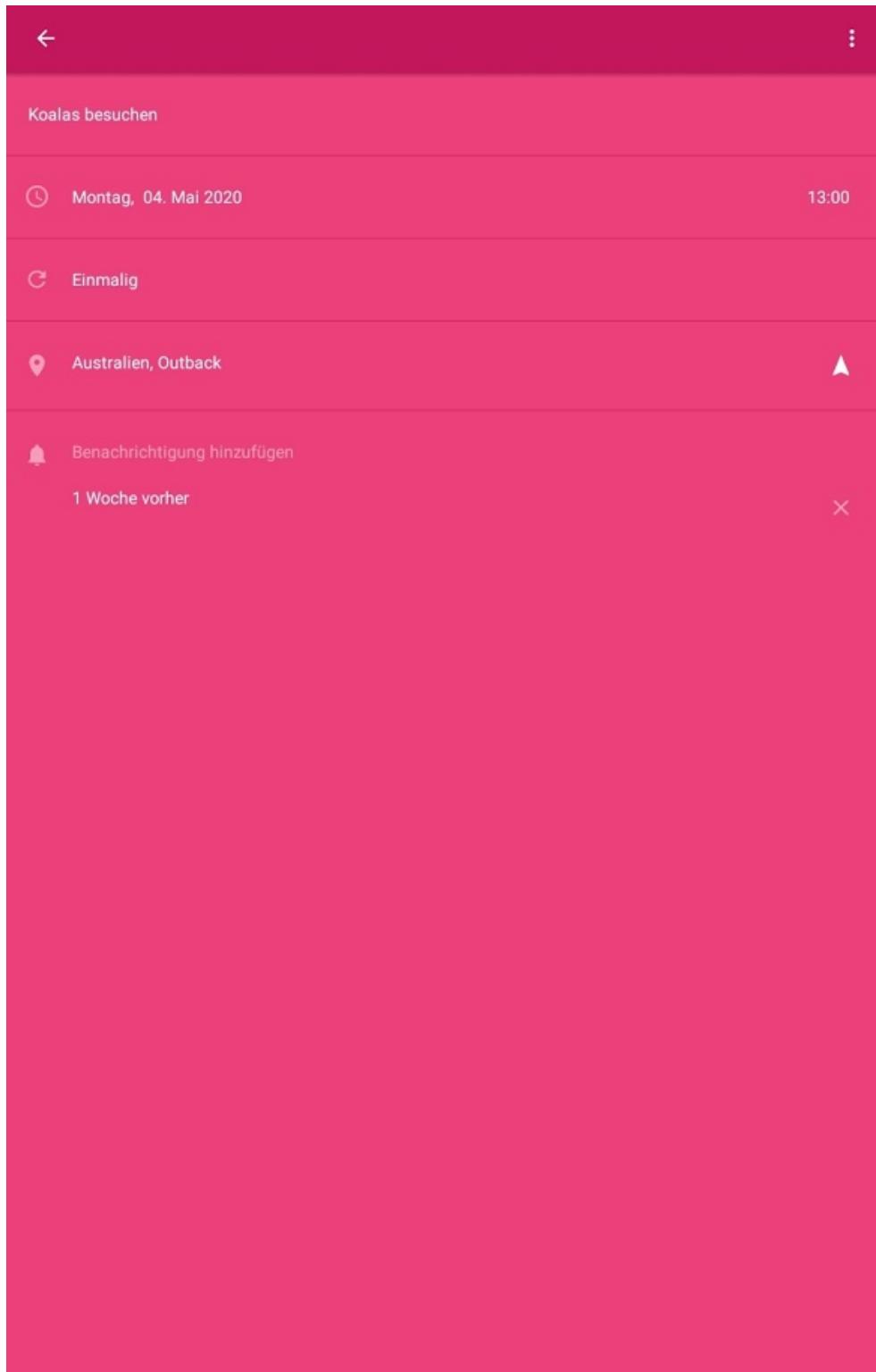
5.2.3.2 Layout (Robin Menzel) Das Layout der Activity setzt sich aus verschiedenen Komponenten zusammen:

- *activity_event.xml*
- *toolbar_ten.xml*

Bei *activity_event.xml* handelt es sich um ein Drawer Layout. Es beinhaltet die Toolbar *toolbar_ten.xml*. Die Inhalte sind in einem Scroll View zusammengefasst.

Die Toolbar *toolbar_ten.xml* ist für alle TENs identisch und beinhaltet ein Drei-Punktmenu mit den Menüpunkten aus *event_menu.xml*

- Event teilen
- Event in den Kalender
- Event löschen

Abbildung 25: Screenshot - Event Activity

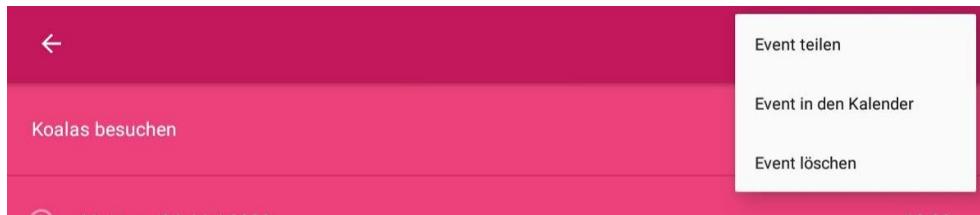
Quelle: Screenshot aus der Benutzeroberfläche

Abbildung 26: Screenshot - Event Activity Erinnerung

Quelle: Screenshot aus der Benutzeroberfläche

Abbildung 27: Screenshot - Event Activity Wiederholung

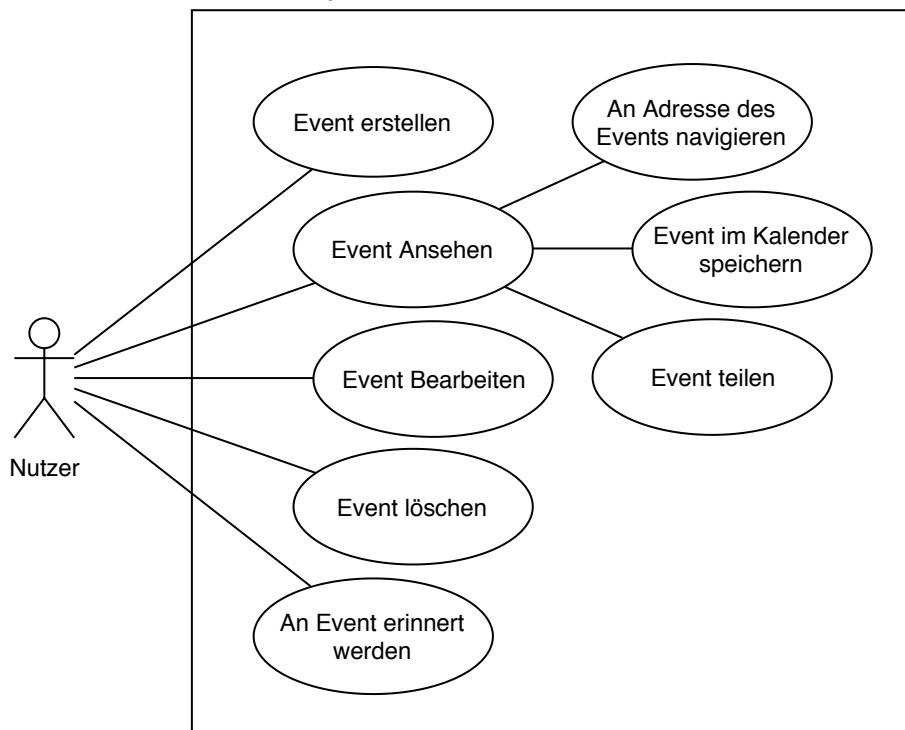
Quelle: Screenshot aus der Benutzeroberfläche

Abbildung 28: Screenshot - Event Activity Toolbar

Quelle: Screenshot aus der Benutzeroberfläche

5.2.3.3 Use-Case (Robin Menzel) Diese Activity dient dem User hauptsächlich zum Anlegen, Bearbeiten, Ansehen und Löschen von einem Event. Außerdem ist die Activity verantwortlich für alle Reminder, die als Push-Benachrichtigung erscheinen und an das jeweilige Event erinnern. In der Activity ist es des Weiteren möglich, über einen Button Google Maps inklusive der Adresse des Events zu öffnen und so zu dem Event zu Navigieren. Außerdem ist es möglich das Event in Form von Text zu teilen (z.B. per E-Mail oder Textnachricht) oder in eine Kalender Applikation zu übertragen.

Abbildung 29: Use-Case - Event Activity



5.2.3.4 Datenstruktur und -typen (Robin Menzel) Die *Event Activity* wird aus der *MainActivity* mittels eines Intents gestartet, welcher falls das Event bereits besteht und kein neues Event angelegt werden soll, die jeweilige ID des Events übergibt, um darüber auf die Datenbank zuzugreifen. Die Klasse *EventActivit* ist aus der Klasse *AppCompatActivity* abgeleitet und überschreibt die für die Funktionen relevanten Methoden dieser Klasse und initialisiert die im Nachfolgenden angesprochenen Verwaltungsklassen *EventData*, *EventGui* und *EventApplicationLogic*.

Die Event-Activity selbst ist in mehrere Pakete aufgeteilt um innerhalb der insgesamt 14 Klassen für eine übersichtliche Struktur zu sorgen und die einzelnen Klassen ihren

übergeordneten Funktionen zuzuordnen. Die Funktionen der einzelnen Klassen und die Klasseninteraktion sind in der Dokumentation des Quelltextes am Ende dieses Kapitels dargestellt.

Logic Paket Das Paket welches die Logik der *Event Activity* beinhaltet, beinhalten alle Listener und Watcher Klassen und die *EventApplicationLogic*. Damit ist sie die zentrale Logik-Klasse der Activity. Von hier aus werden die andren Klassen, welche spezifische Aufgaben übernehmen, initialisiert und angesprochen. Zu den anderen Klassen gehören die Klassen zur Erneuerung der GUI, des automatischen Speichern von Änderungen und den Klassen zur Verwaltung von Erinnerungen.

Data-Paket Das Data-Paket beinhaltet alle Klassen welche für die Datenhaltung zuständig sind. Zentrale Klasse ist hier die *EventData-Klasse* welche für das Laden, aber auch das Speichern von Änderungen bzw. neuen Events zuständig ist. Zudem beinhaltet das Data-Paket eine Reminder Klassse, welche dafür sorgt, dass Reminder zum einen als Zeitpunkt der Erinnerung, aber auch in Text-Form (“1 Stunde vorher”) genutzt werden kann.

Die *Event Activity* interagiert neben der *Overview Activity* auch mit anderen Apps. Entweder durch die ShareAction oder durch die Google Maps Intents.

Gui-Paket Innerhalb des Gui-Paketes subd alle Klassen einsortiert, welche die für die Darstellung der GUI verantwortlich sind. Die EventGui Klasse repräsentiert die Struktur des Layouts und damit alle einzelnen Komponenten, um diese korrekt zu initialisieren und auf einzelne Views zugreifen zu können. Außerdem enthält das Paket die Dialog Picker TimePicker und DatePicker, die für die Auswahl eines Datums bzw. einer Uhrzeit zuständig sind. Zu guter letzt hilft die RecurringTypeHelper dabei die Wiederholungen als Zahlen, aber auch in Textform (“Einmalig”) abbilden zu können.

5.2.3.5 Dokumentation des Quelltextes der Activity (Robin Menzel) Im Nachfolgenden sind die einzelnen Klassen der *Event Activity* mit den jeweiligen wichtigsten Funktionen kurz beschrieben.

EventActivity Diese Klasse ist der zentrale Einstiegspunkt der Note-Activity und ist aus der AppCompatActivity-Klasse abgeleitet. Hier werden einige Methoden dieser Klasse überschrieben und an die neuen Anforderungen angepasst.

onCreate:

Die Activity wird durch einen Intent von der *Overview Activity* gestartet, welchem die ID für ein Event mitgegeben wurde. Diese ID wird ausgelesen und anhand dieser die EventData Klasse initialisiert. Zusätzlich wird die EventGui und die EventApplicationLogic Klasse initialisiert.

onBackPressed:

In dieser Methode wird das normale vorgehen ersetzt durch die Funktion `returnToOverview` der EventApplicationLogic um wieder in die Overview-Klasse zurückzukehren.

onCreateOptionsMenu:

In dieser Methode wird das Menü mit dem Event Menü gefüllt.

LOGIC

EventApplicationLogic Diese Klasse ist die zentrale Logik der Activity.

updateGui

Diese Methode setzt alle Werte aus dem Event-Objekt aus der EventData Klasse in die aktuelle Gui ein.

returnToOverview

Mit Hilfe dieser Methode wird die Activity verlassen und kehrt zurück zur Overview Activity.

setAlarm

Wenn ein Reminder erstellt wird, wird hier der Zeitpunkt zum AlarmManager hinzugefügt (solange der Zeitpunkt in der Zukunft liegt).

updateReminder

Wenn sich die Zeit verändert, müssen auch die Zeitpunkte der Reminder angepasst werden. Dies geschieht in dieser Methode.

onNewReminderClicked

Diese Methode kümmert sich um das Erstellen eines neuen Reminders, solange noch nicht alle vier möglichen Reminder hinzugefügt wurden.

onTextChanged

Sobald der Textwatcher ausgelöst wird, wird diese Funktion aufgerufen um das Event Objekt dynamisch zu speichern.

onCloseReminderClicked

Diese Methode kümmert sich um das Löschen einer Reminders. Dazu gehört auch, diesen aus dem Alarm Manager zu entfernen.

onMenuItemClicked

Diese Methode wird aufgerufen, sobald auf einen der drei Menüpunkte geklicked wurde.

onNavigationIconClicked

Wurde auf das Icon neben der Adresse geklickt, so erstellt diese Funktion einen Google Maps Intent, der die Adresse in Google Maps öffnet.

onRecurrinTypeClicked

Diese Funktion öffnet einen AlertDialog in welchem man den Wiederholungstypen auswählen kann.

listener Klassen Die listener Klassen in dem LOGIC Paket erben alle von View.OnClickListener, text.TextWatcher und Toolbar.OnClickListener. In ihnen wird differenziert welches Element den Listener ausgelöst hat und führt dementsprechend eine Methode aus.

DATA

EventData In dieser Klasse findet die komplette Datenhaltung statt.

updateEvent

In dieser Methode wird des Services “Update” der Datenbank genutzt um ein Event-Objekt zu speichern.

deleteEvent

In dieser Methode wird des Service “Delete” der Datenbank genutzt um ein Event-Objekt zu löschen.

shareEvent

Diese Methode wird Aufgerufen falls der User im ein Event teilen möchte. Von hier aus wird die shareEvent-Methode im ShareModule aufgerufen.

getFormateDate

Diese Methode gibt das Datum des Events in sprach-formattierung zurück, wie z.B. 25. Januar 2019.

exportToCalendar

Diese Methode erstellt aus dem Event ein Intent, welches von allen gängigen Kalender Applikationen unterstützt und geöffnet werden kann.

Reminder Diese Klasse hilft beim Umgang mit Remindern. So müssen z.B. Reminder neu berechnet werden, wenn sich die Zeit des Events verändert hat.

getLabelFromReminder

Diese Methode gerrechnet aus der Differenz zwischen dem Zeitpunkt der Erinnerung und dem Zeitpunkt des Events die Beschriftung (z.B. “1 Stunde vorher”).

getReminderFromLabel

Diese Methode errechnet aus der Beschriftung den richtigen Zeitpunkt einer Erinnerung.

GUI

EventGui Diese Klasse beinhaltet alle Benutzeroberflächenattribute, auf welche so zugegriffen werden kann.

setColor

Die bunte Farbgestaltung ist ein Kernelement unseres Designs. Daher enthält jedes TEN-Objekt auch feste Farbattribute, welche beim Aufbau der Gui dargestellt werden müssen. Diese Methode weißt aus den Farbwerten des Event-Objektes, den Gui Elementen ihre spezifische Farbe zu.

setReminder

Diese Klasse stellt die Reminder, die im Event-Objekt nur als Zeitpunkte dargestellt sind, auf der Oberfläche dar. Dafür interagiert diese mit der Reminder Klasse und passt auch das Auswahl-Menü für weitere Reminder an.

DatePicker und TimePicker Diese Klassen sind fast identisch. Sie erben von der Klasse DialogFragment und überschreiben notwendige Methoden.

onCreateDialog

In dieser Methode wird ein Calendar Objekt erstellt um in ihm später die Ausgewählten Zeiten zu speichern. Falls bereits eine Zeit existiert, wird diese übernommen. Anschließend wird ein DatePickerDialog erstellt.

onDateSet oder *onTimeSet*

Falls eine Zeit ausgewählt wurde, ruft diese Methode, die entsprechenden Methoden in der EventApplicationLogic auf und übergibt die ausgewählten Zeite.

setTime

Da über einen Konstruktor eine vorhandene Zeit eines Events noch nicht übergeben werden kann, muss der Date/Time-Picker zuerst erstellt werden und anschließend kann über diese Methode die bereits vorhandene Zeit eingestellt werden.

RecurringTypeHelper Diese Funktion stellt eine Verbindung zwischen dem Enum Objekt RecurringType und den Beschriftungen der Wiederholungen her.

5.2.4 Note Activity

Im Folgenden wird die Funktionsweise der Notiz-Ansicht innerhalb des TEN-Managers beschrieben, welche bei Erstellung einer neuen Notiz beziehungsweise dem Öffnen einer bereits bestehenden Notiz aufgerufen wird. Da eine Notiz auch Stichworte enthalten kann, besteht für die Bearbeitung dieser zusätzlich eine weitere Activity.

5.2.4.1 Aufgabe und Funktion (Joscha Nassenstein) Die Notiz-Activity soll, wie der Name es schon besagt, dem Nutzer ermöglichen, eigene Notizen anzulegen. Zu den Inhalten einer Notiz zählen dabei ein Titel, eine Beschreibung sowie eine Menge von Bildern und Stichworten. Die Bilder können aus der Notiz heraus aufgenommen beziehungsweise aus der Galerie in die Notiz importiert werden. Die Stichworte können in einer weiteren Ansicht anhand einer dynamischen Liste hinzugefügt, bearbeitet und entfernt werden. Des Weiteren ist es möglich, ein Bild in Großansicht aufzurufen und anhand von Wischgesten zwischen den Bildern zu Wechseln. Allgemein wurde versucht, eine intuitive Bedienung der Notizansicht zu ermöglichen.

5.2.4.2 Layout, Screenshots (Joscha Nassenstein) In der Planung der Notiz-Activity wurde bei der Implementierung der Darstellung der Bilder etwas von der zuvor erstellten MockUp-Ansicht abgewichen, um eine einheitliche Darstellung, aber bei Bildern mit verschiedenen Seitenverhältnissen, zu ermöglichen und diese ohne einen schwarzen Rand darstellen zu können. Zusätzlich sollte der Fokus auf die Notiz selbst gelegt werden, weshalb die Bilder am äußeren Bildschirmrand und etwas kleiner platziert wurden.

Zunächst wurden die XML-Layouts für die beiden Ansichten erstellt. Dabei wurden für die beiden Texteingabefelder jeweils ein EditText-Element verwendet. Zusätzlich wurde für die Stichwörter ein TextView aufgenommen, welches bei einer Usereingabe in Form eines Klicks die neue Activity aufrufen sollte. Für die Bilder wurde ein Horizontaler ScrollView in der Portrait- Ansicht und ein Vertikaler ScrollView in der Landscape-Ansicht eingefügt. Die Elemente wurden zusätzlich von zwei beziehungsweise drei Separatoren unterteilt (Views mit 1dp Höhe).

Des Weiteren wurde ein Layout für das ImageOverlay angelegt, welches ein Bild in Großformat anzeigen sollte. Dies kann in einen rahmenlosen AlertDialog eingefügt werden.

Dieses Layout beinhaltet lediglich einen ImageView.

Die Farben und Texte für die Layouts wurden in jeweils in den Ressourcendateien im values-package angelegt, damit diese zentral verwaltet werden können. Zusätzlich wurde in styles.xml ein Style für das ImageOverlay angelegt, welches den Hintergrund transparent erscheinen lässt. Damit waren die Layouts für diese Activity angelegt.

Im Nachfolgenden sind die verschiedenen Layoutfunktionen anhand von Screenshots dargestellt.

Portrait-Ansicht Die Notizansicht beinhaltet am oberen Bildschirmrand eine Taskleiste, über welche zur Übersicht zurückgekehrt werden kann sowie ein Zugriff auf weitere Optionen erfolgen kann. Darunter werden die beiden Textfelder zur Eingabe des Titels und der Beschreibung angezeigt. Der Titel ist dabei einzeilig, die Beschreibung kann beliebig viele Zeilen umfassen. Falls der Platz zwischen Titel und Bilderansicht nicht ausreicht, wird die Beschreibung scrollbar. Darunter werden die Bilder angezeigt, welche bei entsprechender Anzahl horizontal durchgescrollt werden können. In dieser Ansicht befindet sich stets als rechtestes Element ein Icon, welches zum Hinzufügen von Bildern angeklickt werden kann. Abgeschlossen wird die Ansicht durch eine Auflistung der Stichworte, welche maximal drei Zeilen hoch ist und ab einer Überschreitung dieser Zeilenanzeigen ebenfalls gescrollt werden kann. Beim Klicken auf die Stichworte wird die NoteTagActivity gestartet.

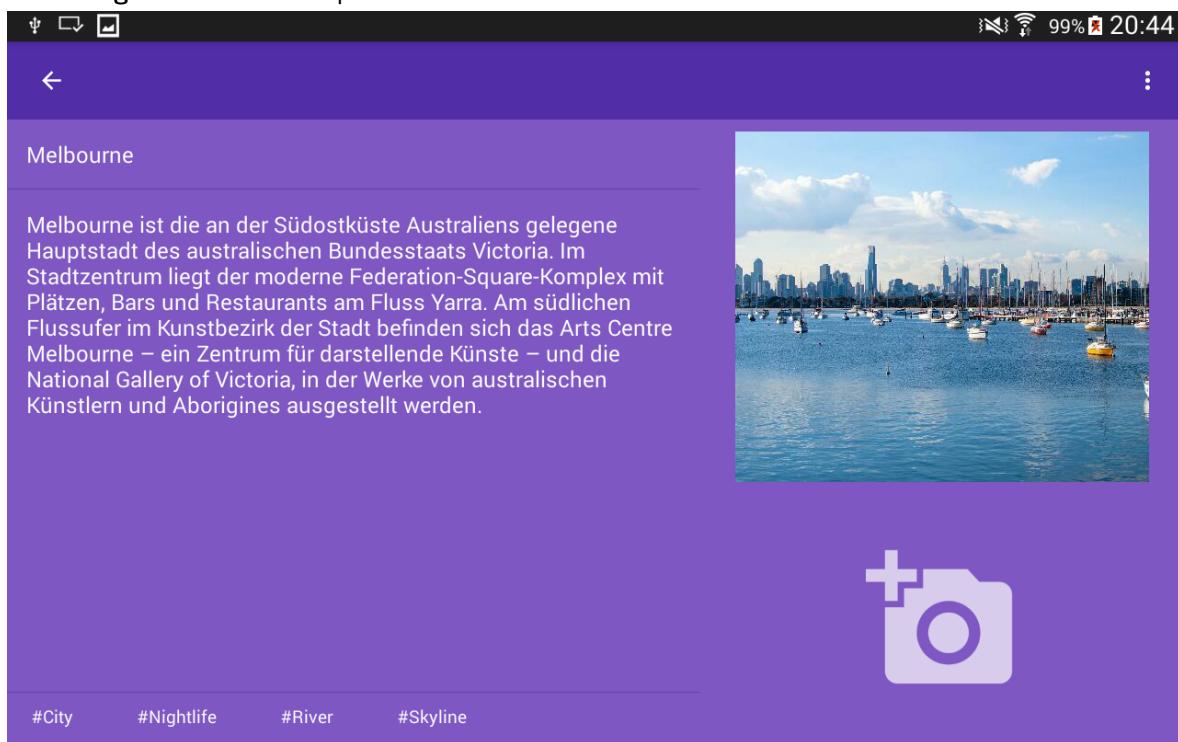
Abbildung 30: Note Portrait-Ansicht

Quelle: Screenshot aus der Benutzeroberfläche

Landscape-Ansicht Wird das Gerät gedreht und damit in die Landscape Ansicht gewechselt, wird die Ansicht etwas anders dargestellt. Primär wird hier die Bilderansicht auf die rechte Seite verschoben, um die Breite des Displays auszunutzen. Von nun an kann hier vertikal gescrollt werden und die Option zum Hinzufügen von neuen Bildern befindet sich an unterster Stelle.

Zusätzlich lässt sich hier erkennen, dass die Ansicht jeweils die Hauptfarbe und die Akzentfarbe aus dem Note-Objekt übernimmt. Die Farbe wird zufällig beim Anlegen einer Notiz aus einer Vorauswahl ausgewählt und im Objekt gespeichert. Hier ist nun als Beispiel eine andere Farbgebung dargestellt.

Abbildung 31: Note Landscape-Ansicht

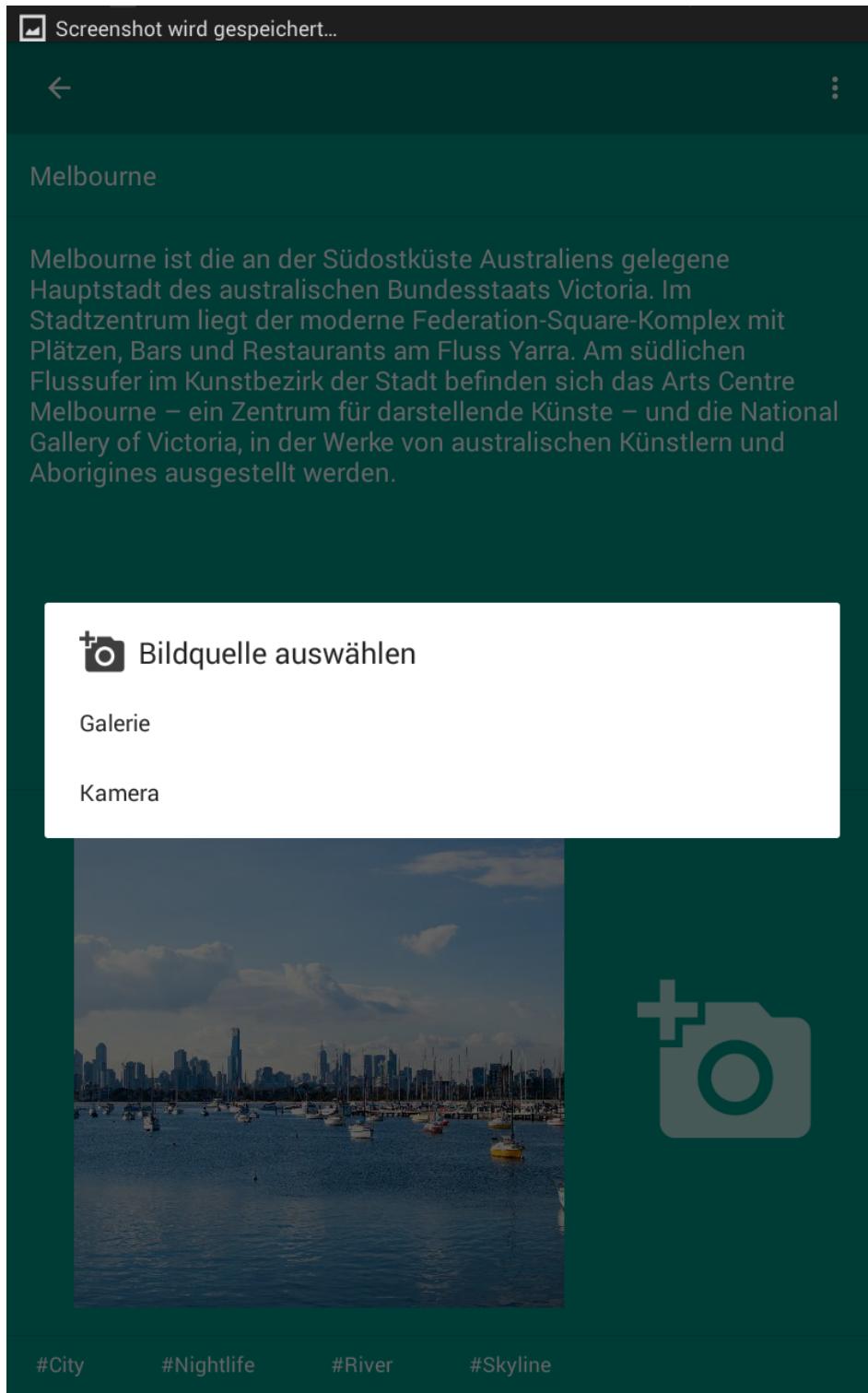


Quelle: Screenshot aus der Benutzeroberfläche

Hinzufügen eines Bildes Falls der Nutzer das Icon zum Hinzufügen eines Bildes anklickt, gibt es zwei verschiedene Möglichkeiten, welche eintreten können. Falls das Gerät eine Kamera besitzt, wird der in der Abbildung dargestellte Dialog aufgerufen, damit der Nutzer sich entscheiden kann, ob ein Bild aus direkt aus der Kamera oder aus der Galerie importiert werden soll. Ist dies nicht der Fall, wird direkt die Androideigene

Galerieimportfunktion gestartet. Für die Kamera wird ebenfalls eine Androideigene Funktion verwendet.

Nach Import des Bildes wird dies in der Bilderzeile angezeigt.

Abbildung 32: Note Dialog zum Hinzufügen eines Bildes

Quelle: Screenshot aus der Benutzeroberfläche

Großansicht eines Bildes Wenn der Benutzer ein Bild auswählt, wird dieses im Großformat angezeigt. Dazu wird ein AlertDialog aufgerufen und mit dem Bild, welches in voller Größe aus der Datenbank geladen wird, gefüllt. Zusätzlich kann durch Wischgesten zwischen den Bildern gewechselt werden. Durch eine horizontale Wischgeste, das Klicken außerhalb des Bildbereiches oder das Betätigen des Zurück-Buttons wird die Ansicht geschlossen.

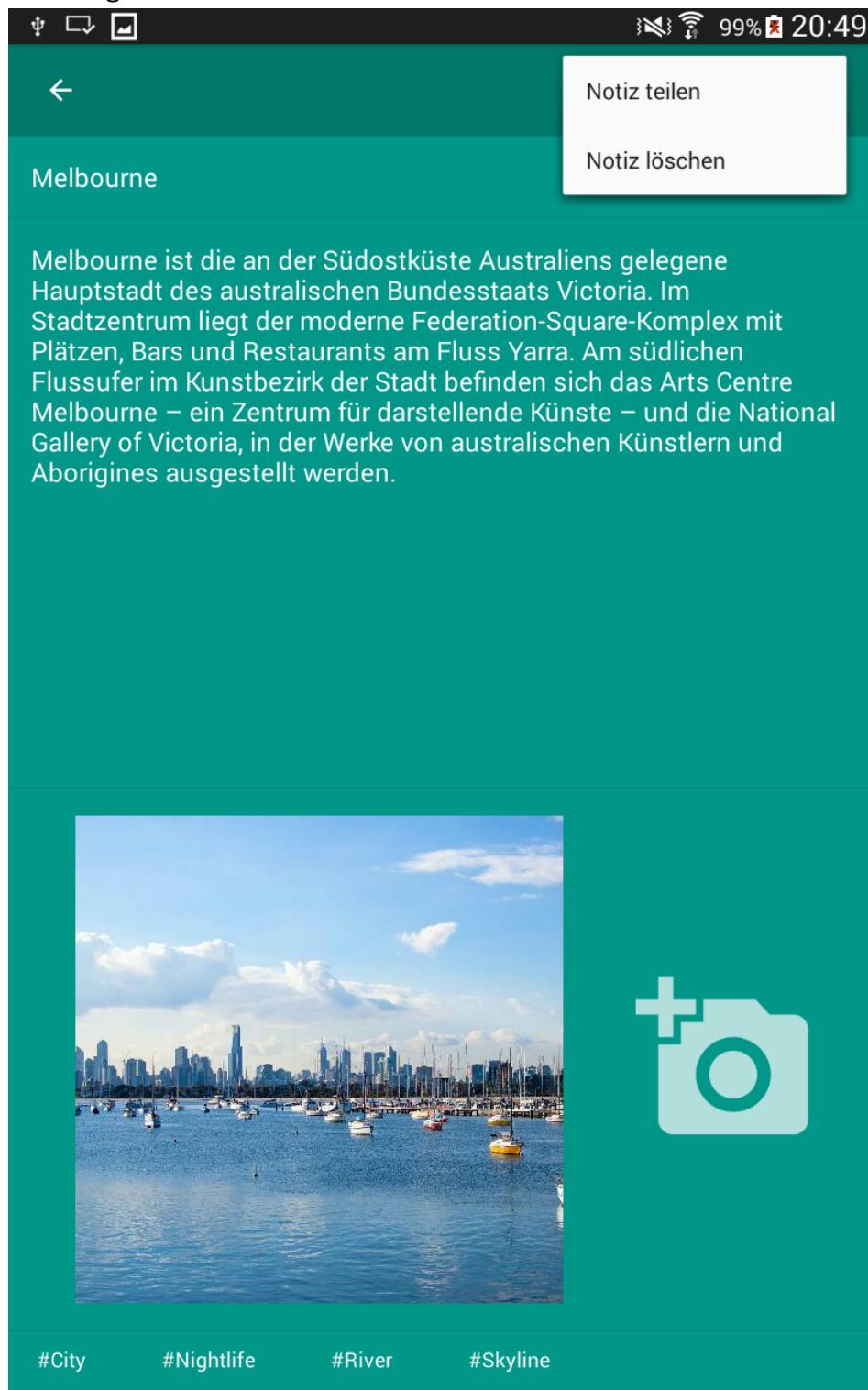
Das Bild füllt dabei den Bildschirm entsprechend eines in den Konstanten der Note-Activity festgelegten Wertes aus (in diesem Falle zu 95 Prozent). Falls das Display gedreht wird und zu dem Zeitpunkt ein Bild in Großansicht angezeigt wird, wird dieses den neuen Bildschirmverhältnissen angepasst.

Abbildung 33: Note Bildansicht



Quelle: Screenshot aus der Benutzeroberfläche

Toolbarmenü Über die Drei Punkte in der Toolbar kann das Menü aufgerufen werden, in welchem der Nutzer die Notiz extern teilen oder die Notiz löschen kann.

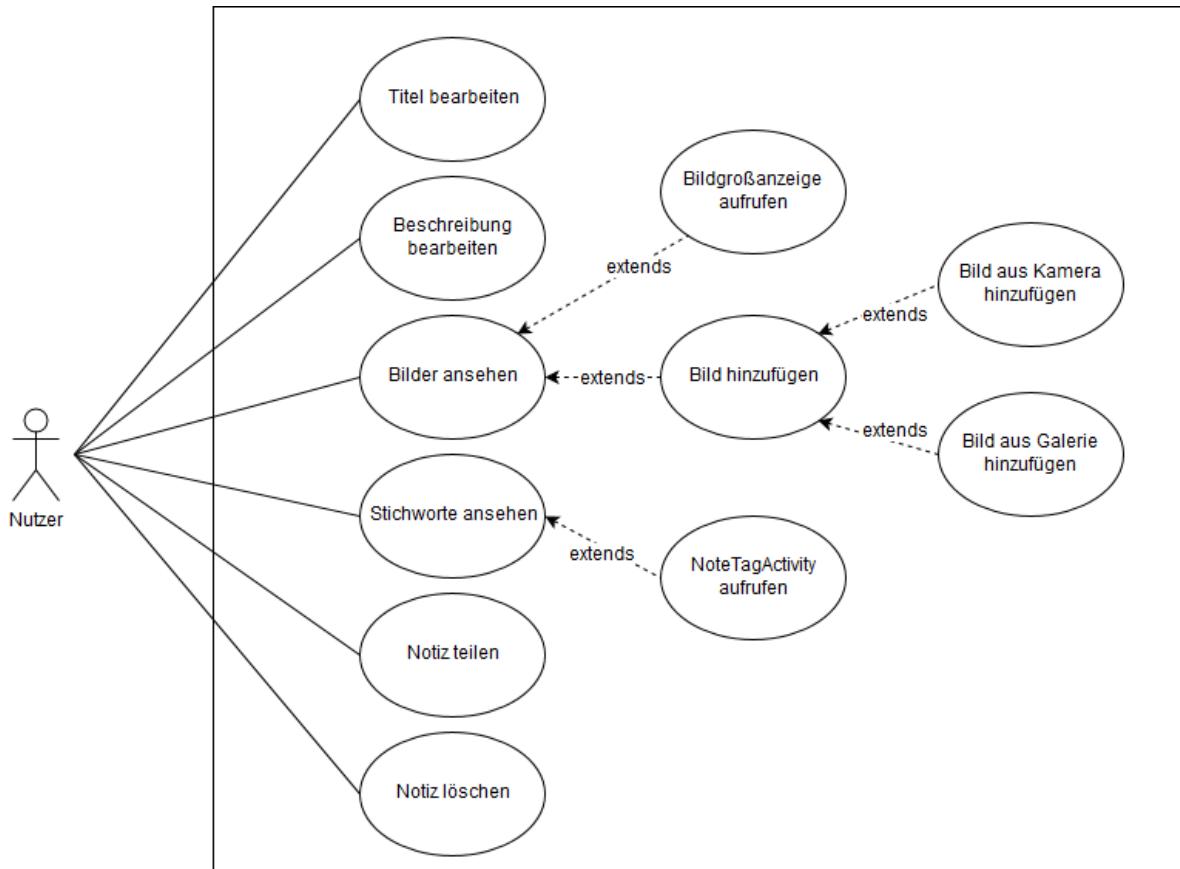
Abbildung 34: Note Toolbarmenü

Quelle: Screenshot aus der Benutzeroberfläche

5.2.4.3 Use Case (Joscha Nassenstein) Der Use-Case dieser Activity besteht darin, sämtliche Attribute einer Notiz einpflegen beziehungsweise anpassen zu können. Der Funktionsumfang wurde bereits in dem vorangegangenen Kapitel Aufgabe und Funktion dargestellt.

Im Folgenden ist ein Use-Case Diagramm abgebildet, welche diese Funktionen zusammenfasst. Auf die Bedingung der extend-Pfeile wurde hierbei zur Erhöhung der Übersichtlichkeit verzichtet, da dies stets die entsprechende Benutzereingabe in Form eines Klicks auf die Fläche darstellt.

Abbildung 35: Note Use-Case Diagramm



Quelle: Erstellt von Joscha Nassenstein

5.2.4.4 Datenstruktur und -typen Die Notiz-Activity wird aus der Overview-Activity mittels eines Intents gestartet, welcher, falls die Notiz bereits besteht und keine neue Notiz angelegt werden soll, die jeweilige ID der Notiz übergibt, damit darüber auf die

Datenbank zugegriffen werden kann. Die Klasse NoteActivity ist aus der Klasse AppCompatActivity abgeleitet und überschreibt die für die Funktionen relevanten Methoden dieser Klasse und initialisiert die im Nachfolgenden angesprochenen Verwaltungsklassen NoteData, NoteGui sowie NoteApplicationLogic.

Restruktrierung der Pakete (Jan Beilfuß) Als bereits ein Großteil des Funktionsumfangs der Notiz-Activity implementiert war, gab es eine NoteData-Klasse und eine NoteApplicationLogic-Klasse. Alle Klassen der Notiz-Activity befanden sich in einem Paket.

Die Klassen NoteData und NoteApplicationLogic enthielten eine Vielzahl von Methoden mit unterschiedlichend fachlichen wie technischen Funktionen. Die Länge der beiden Klassen von 200 bis 300 Zeilen machte diese unübersichtlich. Alle Klassen lagen in einem Paket. Als Folge dessen sinkt die Erweiterbarkeit und der Aufwand bei der Fehlerfindung steigt.

Es wurden folgende Maßnahmen getroffen, um die beschriebene Problemstellung zu lösen. Zu einen wurde im Paket note die drei Pakete data, logic und gui angelegt. Diese wurden weiter in Unterpakete aufgeteilt, sodass garantiert war, dass nicht mehr als sechs bis sieben Klassen im selben Paket liegen. Ebenfalls wurden die Klassen nach ihrer Funktion in die entsprechenden Pakete sortiert.

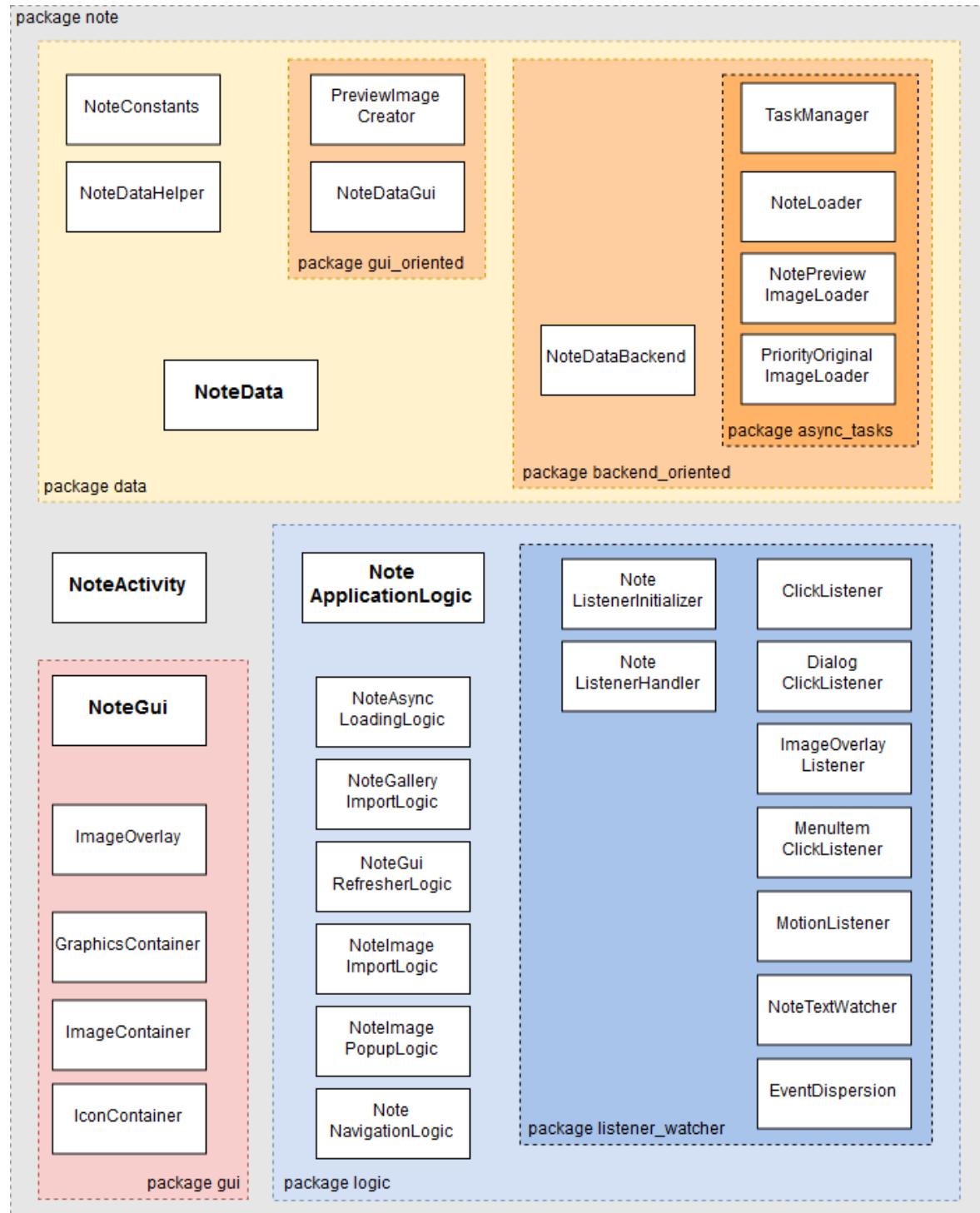
Durch die ergriffenene Maßnahmen konnte bei der Weiterentwicklung der Applikation jeweils ein schneller Einstieg gefunden werden. Weiterhin konnte die gewählte Struktur flexibel für Implementierungen, die aus dem Debuggingprozess hervorgegangen sind, angepasst und erweitert werden.

Data-Paket (Joscha Nassenstein) Das Data-Paket beinhaltet alle Klassen, welche für die Datenhaltung zuständig sind. Diese Funktionen werden zentral von der NoteData-Klasse verwaltet. Das Paket ist in Backend-orientierte und Benutzeroberflächen-orientierte Klassen aufgeteilt, um auch hier für eine Differenzierung zu sorgen. Zu den Backend-orientierten Klassen zählen dabei alle Klassen, welche für das asynchrone Laden der Notiz, insbesondere der Bilder, sorgen, um die Performance dieser Ansicht zu optimieren. Die Benutzeroberflächen-orientierten Datenklassen bestehen einer Klasse, welche für das Importieren der Bilder und das anschließende Bearbeiten, insbesondere in Bezug auf die Vorschaugröße, zuständig sind.

GUI-Paket (Joscha Nassenstein) Innerhalb des GUI-Paket sind alle Klassen eingesortiert, welche Views beinhalten, welche auf der Benutzeroberfläche dargestellt werden. Die NoteGui-Klasse repräsentiert die Struktur des Layouts und damit alle einzelnen Komponenten, um diese korrekt zu initialisieren und auf einzelne Views zugreifen zu können. Des Weiteren ist eine Klasse für die Darstellung eines ausgewählten Bildes in Form eines Overlays sowie Klassen für die Einsortierung der einzelnen Vorschaubilder in das allgemeine Notiz-Layout erstellt worden.

Logic-Paket (Joscha Nassenstein) Das Paket, welches die Logik der Note-Activity beinhaltet, ist mit Abstand das umfangreichste der drei Pakete. Zunächst sind hier alle Listener- und Watcher Klassen beinhaltet, welche bei Klicken eines bestimmten Elements, darunter Views (auch Bilder) oder bestimmte Menüoptionen, sowie bei der Eingabe von Texten in den Titel und die Beschreibung bestimmte Aktionen ausführen. Eine Klasse ist zusätzlich für die Initialisierung dieser zuständig, eine weitere für das Ausführen der damit verbundenen Aktionen.

Die NoteApplicationLogic-Klasse ist die zentrale Logik-Klasse in der Note-Activity. Von hier aus werden die anderen Klassen, welche spezifische Aufgaben übernehmen, initialisiert und angesprochen. Zu den Funktionen letzterer Klassen gehören dabei die Verwaltung der asynchronen Ladelogik, eine Klasse zur Verwaltung der Galerie-Import Funktion, eine Klasse zur Erneuerung der Benutzeroberfläche, eine Klasse für die Detailanzeige von Bildern sowie eine Klasse für die Verwaltung der Navigation zwischen der Note-Activity und den damit verbundenen anderen Activities. Die Note-Activity interagiert neben der Overview-Activity auch mit der NoteTag-Activity zur Verwaltung der Stichwortliste sowie mit den von Android bereitgestellten Kamera- und Galerie-Activities.

Abbildung 36: Darstellung der Paketstruktur

Quelle: Erstellt von Joscha Nassenstein

5.2.4.5 Dokumentation des Quelltextes der Activity Im Nachfolgenden sind die einzelnen Klassen der *Note-Activity* mit den jeweiligen wichtigsten Funktionen kurz beschrieben. In einem Beispiel ist ein Quellcodeausschnitt beigefügt.

Klasse: NoteActivity (Joscha Nassenstein) Diese Klasse ist der zentrale Einstiegs-
punkt der Note-Activity und ist aus der AppCompatActivity-Klasse abgeleitet. Hier
werden einige Methoden letzterer überschrieben und an die eigenen Anforderungen
angepasst.

onCreate:

In dieser Methode wird die im Intent übergebene ID ausgelesen und anhand dessen
die NoteData-Klasse initialisiert. Zusätzlich werden NoteGui, NoteApplicationLogic
und EventDispersion initialisiert. Falls keine ID übergeben wurde, wird die Tasta-
tur eingeblendet und die NoteGui-Klasse mit dem Parameter pNewNote = true ini-
tialisiert, wodurch das Titelfeld zur direkten Eingabe durch den Benutzer fokussiert
wird.

onActivityResult:

An dieser Stelle wird das Ergebnis einer zuvor gestarteten externen Activity abgefangen
und an die NoteApplicationLogic weitergegeben, wo das Ergebnis weiterverarbeitet
werden kann.

onCreateContextMenu:

Diese Methode wird jeweils für die Erstellung des Kontextmenüs innerhalb der Activity
und in der Toolbar überschrieben. Die Methoden unterscheiden sich dabei durch den
Rückgabetyp und den Parameter. Die Erstellung des Kontextmenüs findet letztlich in
der Klasse EventDispersion statt.

onContextItemSelected:

Hier wird ebenfalls die entsprechende Methode in EventDispersion aufgerufen.

onConfigurationChanged:

Diese Methode wird aufgerufen, falls die Konfiguration des Gerätes, beispielsweise die
Ausrichtung, verändert wird. Normalerweise wird bei einer Konfigurationsänderung die
aktuelle Instanz der Activity beendet und eine neue Instanz aufgerufen. Dies wird
jedoch durch die explizite Einstellung im AndroidManifest verhindert, damit hier ein
eigener Ablauf implementiert werden kann. Konkret wird die Datenklasse erhalten, um

weiterhin den aktuellen Stand der Daten zur Verfügung zu stellen. Die Oberfläche der Activity wird hingegen neu erstellt, da sich einige Elemente ändern. Dazu zählt primär der horizontale Scrollview, welcher in der Landscape-Ansicht einem Vertikalen weichen soll. Zusätzlich wird die entsprechende Methode in der NoteApplicationLogic aufgerufen, um weitere Aktionen durchzuführen.

Klasse: NoteConstants (Jan Beilfuß) Diese Klasse enthält alle Konstanten, die in der Programmierung der Notiz-Activity Verwendung finden. Dazu gehören IDs und Requestcodes für den Datenaustausch mit den aus der Notiz-Activity aufgerufenen Intents Gallery, Kamera und NoteTagActivity, aber auch Skalierungsfaktoren und Größen, die in der Gui verwendet werden.

Klasse: NoteDataHelper (Jan Beilfuß) Diese Klasse hat den Zweck Methoden zu bündeln, welche nicht am direkten Datenaustausch mit dem Backend oder Gui zu tun haben. Dazu zählt beispielsweise eine Methode, um zu checken, ob das gehaltene Notiz-Objekt lediglich mit default-Werten gefüllt ist.

isSaveable:

Diese Methode prüft, ob der Titel und die Beschreibung leere Strings sind. Weiterhin wird geprüft, ob die enthaltenen Listen tags und pictures leer sind. Sind alle Strings und Listen leer, wird das Objekt aus der Datenbank gelöscht und false zurückgegeben. Dadurch soll verhindert werden, dass leere Notiz-Kacheln in der Overview-Activity angezeigt werden.

DATA

Klasse: NoteData (Joscha Nassenstein) Die NoteData-Klasse spiegelt die zentrale Datenverwaltungsklasse wieder, aus der die backend- beziehungsweise benutzeroberflächenbezogenen Datenklassen angesprochen werden.

shareNote:

Diese Methode wird aufgerufen, falls der Benutzer im Kontextmenü die Option des Teilens ausgewählt hat. Von hier aus wird die shareNote-Methode im ShareModule aufgerufen.

Klasse: NoteDataBackend (Jan Beilfuß) In dieser Klasse werden alle Datenfunktionen gesteuert, welche mit dem Backend interagieren. Weiterhin wird hier die Liste der Bilder gehalten, welche beim Speicher zu löschen sind. Wenn der Nutzer beispielsweise ein Bild löscht und dann die App ohne speichern schließt, ist das Bild beim nächsten Laden im Vergleich zum direkten Löschen noch vorhanden.

deleteImage:

Diese Methode entfernt das zu löschen Bild aus dem Note-Objekt und fügt dieses der Liste mit den zu löschen Bildern hinzu.

triggerOriginalImageLoad:

Diese Methode ruft im Taskmanager eine Funktion auf, um ein Bild in voller Auflösung im Hintergrund nachzuladen.

setColors:

Diese Methode lädt die Haupt und die Akzentfarbe zu dem Notizobjekt, dessen ID übergeben wurde und setzt diese in dem in NoteData gehaltenen Notizobjekt.

executeSaveRoutine:

Diese Methode veranlasst das persistente Löschen der Bilder. Wenn die Notiz komplett leer ist, wird diese gelöscht, andernfalls wird sie in der Datenbank gespeichert.

finallyDeleteImages:

Diese Methode geht über alle ImageObjekte in der Liste mit den zu löschen Bildern und übergibt diese der Delete-Methode im ImageService. Dadurch werden die Bilder im Dateisystem gelöscht.

deleteNote:

Diese Methode wird aufgerufen, wenn man die Notiz in der Datenbank löschen möchte. Wenn das NoteObjekt existiert, wird dessen ID der Delete-Methode im Delete-Service aufgerufen und die ID übergeben.

loadNote:

Diese Methode wird im Erstellungprozess der Notiz-Activitiy aufgerufen und veranlasst das Laden des Notizobjektes. Dafür wird im Taskmanager die Methode loadNote aufgerufen.

deleteImageFromDisk:

Diese Methode löscht ein einzelnes Bild im Speicher. Sie wird aufgerufen, um die Datei, welche temporär für die Kamera-Activitiy angelegt wurde zu löschen.

saveImage:

Diese Methode speichert ein übergebenes Bild. Sie wird verwendet nachdem eine Bitmap aus der Note oder Kameraactivity geladen und mit einer ID versehen wurde, um diese im Datenordner der Applikation abzulegen.

Klasse: TaskManager (Jan Beilfuß) Diese Klasse verwaltet den Einsatz aller asynchrone Tasks, die zum Laden von Daten aus der Datenbank und dem Dateisystem verwendet werden.

triggerOriginalImagePriorityLoad:

Diese Methode erzeugt ein Objekt der Klasse PriorityOriginalImageLoader. Über dieses Objekt wird das Bild zu dem übergebenen Index in originaler Größe asynchron geladen.

loadPreviewImages:

Diese Methode erzeugt ein Objekt der Klasse NotePreviewImageLoader. Über dieses Objekt wird das Bild zu dem übergebenen Index in Vorschaugröße asynchron geladen.

loadNote:

Diese Methode erzeugt ein Objekt der Klasse NoteLoader. Über dieses Objekt wird das Noteobjekt zu der übergebenen ID asynchron geladen.

Klasse: NotePreviewImageLoader (Jan Beilfuß) Diese Klasse erzeugt einen asynchrone Task der Klasse LoadPreviewImageTask, welche von AsyncTask erbt.

loadPreviewImage:

Diese Methode erzeugt den asynchronen Task und führt diesen aus. In der Methode `textitdoInBackground` wird eine Kopie des Image-Objektes aus dem NoteObjekt angefertigt und in diese das gewünschte Vorschaubild geladen. Nach dem Laden wird in `textitonPostExecute` das Bild aus dem Noteobjekt gelöscht, falls keine Bitmap geladen werden konnte, oder andernfalls das Bild über die Methode `textitaddAsyncPreviewImage` in die Note Activity geladen.

Klasse: PriorityOriginalImageLoader (Jan Beilfuß) Diese Klasse erzeugt einen asynchrone Task der Klasse LoadOriginalImageTask, welche von AsyncTask erbt.

loadOriginalImage:

Diese Methode erzeugt den asynchronen Task und führt diesen aus. In der Methode textitdoInBackground wird das Bild in originaler Größe geladen. Vor dem Laden wird in textitonPreExecute der Ladespinner gestartet. Nach dem Laden wird in textitonPostExecute das Bild aus dem Noteobjekt gelöscht, falls keine Bitmap geladen werden konnte, oder andernfalls das Bild über die Methode textitaddAsyncPreviewImage in die Note Activity geladen, der Ladespinner gestoppt und das Bild als Popup angezeigt.

Klasse: NoteLoader (Jan Beilfuß) Diese Klasse erzeugt einen asynchrone Task der Klasse LoadNoteTask, welche von AsyncTask erbt. *loadNote:*

Diese Methode erzeugt den asynchronen Task und führt diesen aus. In der Methode textitdoInBackground wird das Note-Objekt aus der Datenbank geladen. Nach dem Laden wird in textitonPostExecute das Note-Objekt in NoteData gesetzt und die Methode textitdataToGui ausgeführt, um dieses in der Oberfläche anzuzeigen.

Klasse: NoteDataGui (Jan Beilfuß) In dieser Klasse werden alle Datenfunktionen gebündelt, welche Datenänderungen durch Nutzer-Interaktion mit der Oberfläche erzeugen. Weiterhin wird hier die Liste der Bilder gehalten, welche in der Oberfläche als Vorschaubilder angezeigt werden. *addImageFromGallery:*

Diese Methode bekommt die Bitmap des aus der Gallerie importierten Bildes übergeben. Daraus wird über die Methode textitaddPreviewImageFromOriginal ein Vorschaubild erzeugt und dieses der Liste mit Vorschaubildern hinzugefügt. Weiterhin wird das Originale Bild in den applikationseigenen Ordner abgespeichert. *addPreviewImageFromOriginal:* Diese Methode erzeugt aus einem originalen Bild ein Vorschaubild und fügt dieses der klasseneigenen Liste an Vorschaubildern hinzu. *addImageFromCamera:*

Diese Methode fügt das aus der Kamera-Activity übergebene Bild hinzu. Dazu wird die Methode textitaddImageFromGallery aufgerufen und zusätzlich die zuvor temporär erstellte Datei gelöscht. *getOriginalImage:*

Diese Methode wird aufgerufen, wenn man auf ein Bild klickt, um dieses in originaler Größe zu laden. Sofern dieses nicht in dem Note-Objekt in NoteData gespeichert ist, wird in NoteDataBackend die Methode textittriggerOriginalImageLoad aufgerufen, um

dieses aus dem Dateisystem zu laden. *deleteImage*:

Diese Methode wird aufgerufen, wenn der Nutzer in der Oberfläche ein Bild löschen möchte. Dieses Bild wird aus der Liste an Vorschaubildern gelöscht und die Oberfläche aufgefrischt. *refreshImages*:

Diese Methode ruft in der NoteGuiRefresherLogic die Methode `textitrefreshImages` auf, um die Gui bei Änderungen im Datenbestand zu aktualisieren. *getLatestImage*:

Diese Methode gibt das letzte Bild der Liste an Vorschaubildern zurück. Dies wird verwendet, damit man dieses beim Asynchronen Laden als einzelnes Bild animiert dem Layout hinzufügen kann.

Klasse: PreviewImageCreator (Joscha Nassenstein) Diese Klasse beinhaltet die Methode, um ein Bild auf Vorschaugröße zu skalieren.

getPreviewImage:

Erstellt aus dem Originalbild, welches als Parameter übergeben wird, ein verkleinertes und im Seitenverhältnis angepasstes Bild. In der aktuellen Form wird dabei ein quadratisches Bild in Größe der entsprechenden Konstanten in NoteConstants erstellt.

GUI

Klasse: NoteGui (Joscha Nassenstein) Die NoteGui-Klasse beinhaltet die Benutzeroberflächenelemente als Attribute, wodurch auf diese zugegriffen werden kann. Zusätzlich werden von hier aus die Animationen für das asynchrone Laden der Bilder gesteuert.

initiateOrientationDependentViews:

Die Layouts für den Portrait- und den Landscapemodus beinhalten in der Note-Activity verschiedene Elemente, um eine optimale Darstellung zu ermöglichen. Einerseits gibt in der Portrait-Ansicht einen Separator mehr als in der Landscape-Ansicht, andererseits ist das ScrollView in der Portrait-Ansicht horizontal und in der Landscape-Ansicht vertikal scrollbar. Um diese Views entsprechend der Orientation des Gerätes auszurichten, wird in dieser Methode die aktuelle Orientation abgefragt und entsprechend des Ergebnisses die Variablen initialisiert.

setColors:

Da jede Notiz zwei verschiedene Farben enthält, werden in dieser Methode die Benutzeroberflächenelemente anhand der Farben aus der Notiz angepasst.

formatTags:

In dieser Methode werden die Stichwörter aus der Notiz für die Ansicht formatiert.

Klasse: GraphicsContainer (Joscha Nassenstein) Diese Klasse stellt die Überklasse für die beiden nachfolgenden Klassen dar und beinhaltet ein LinearLayout, in welches ein Vorschaubild beziehungsweise ein Icon eingefügt werden kann. Dieses wird wiederum in den ScrollView der Note-Ansicht eingefügt. Das LinearLayout besitzt eine ID, welche für die Erkennung des zu löschen Bildes bei der entsprechenden Benutzereingabe verwendet wird.

Klasse: ImageContainer (Joscha Nassenstein) Diese Klasse beinhaltet ein ImageView, in welches durch die nachfolgende Methode das entsprechende Vorschaubild eingefügt werden kann. Der ImageView kann von außerhalb der Klasse mithilfe eines Getters abgerufen werden, um in diesem spezifischen Falle eine Animation für das Erscheinen nach dem asynchronen Laden zu ermöglichen. Der ImageView wird im Konstruktor der Klasse ebenfalls für das Kontext-Menü registriert, um das Löschen eines Bildes zu ermöglichen.

initiateView:

Anhand eines übergebenen Bildes wird ein imageView initialisiert und auf das LinearLayout aus der GraphicsContainer-Klasse gesetzt.

Klasse: IconContainer (Joscha Nassenstein) In dieser Klasse wird in der nachfolgenden Methode ebenfalls ein ImageView erstellt, hier jedoch mit einem Icon an Stelle eines Bildes. Der ImageView selbst wird hier nicht gehalten, da dieser nur temporär in der nachfolgenden Methode benötigt wird.

initiateView:

Anhand des im Konstruktors übergebenen Icons wird der ImageView initialisiert und auf das LinearLayout aus der GraphicsContainer-Klasse gesetzt. Die Layout-Parameter werden anhand der aktuellen Orientierung ausgewählt, um eine unnötig breite Erscheinung des Layouts zu vermeiden.

Klasse: ImageOverlay (Joscha Nassenstein) In dieser Klasse wird anhand eines übergebenen Bildes und der aktuellen Displayeigenschaften ein AlertDialog erstellt, welcher das Bild im Großformat anzeigt.

display:

Der bereits erwähnte AlertDialog wird mit dem übergebenen Bild und entsprechend der in der nachfolgenden Methode berechneten Größe erstellt und anschließend angezeigt.

calculateSize:

Hier wird anhand der in die Klasse übergebenen aktuellen Displayeigenschaften, sprich der Displaygröße, das aktuelle Seitenverhältnis ausgerechnet. Dies wird ebenfalls für das Bild durchgeführt. Anschließend wird anhand verschiedener Vergleiche die optimale Bildgröße ermittelt, sodass das Bild in maximaler Größe angezeigt werden kann und der AlertDialog keine Ränder aufweist. Ermittelt wird diese Größe dadurch, dass überprüft wird, ob das Bild bei einer Skalierung zunächst die Breite oder die Höhe des Bildschirms überschreiten würde. Zusätzlich wird ein in NoteConstants vermerkter Füllfaktor berücksichtigt, welcher bestimmt, zu welchem Grad der Bildschirm mit dem Bild ausgefüllt werden soll. Der Quellcode ist im Nachfolgenden dargestellt:

Abbildung 37: Codebeispiel

```
//Calculates size of frame according to display metrics
private boolean calculateSize() {
    int imageWidth = mImage.getWidth();
    int imageHeight = mImage.getHeight();
    double imageAspectRatio = (double) imageWidth / imageHeight;
    double displayAspectRatio = (double) mDisplayWidth / mDisplayHeight;

    //Checks if image is smaller than display * edgeFactor
    if(imageWidth <= mDisplayWidth *NoteConstants.IMAGE_OVERLAY_FILL_FACTOR &&
       imageHeight <= mDisplayHeight *NoteConstants.IMAGE_OVERLAY_FILL_FACTOR){
        return false;
    }
    //Checks if image is a square
    else if (imageWidth == imageHeight) {
        if (mDisplayHeight > mDisplayWidth) {
            mFrameWidth = mDisplayWidth;
            mFrameHeight = mDisplayWidth;
        } else {
            mFrameWidth = mDisplayHeight;
            mFrameHeight = mDisplayHeight;
        }
    }
    //Checks if image exceeds display in width
    else if (imageAspectRatio > displayAspectRatio) {
        mFrameWidth = mDisplayWidth;
        mFrameHeight = (int) ((double) mFrameWidth / imageAspectRatio);
    }
    //Checks if image exceeds display in height
    else {
        mFrameHeight = mDisplayHeight;
        mFrameWidth = (int) ((double) mFrameHeight * imageAspectRatio);
    }
    return true;
}
```

Quelle: Screenshot aus der Entwicklungsumgebung

changeOrientation:

Bei einer Konfigurationsänderung wird in der NoteApplicationLogic stets überprüft, ob aktuell ein ImageOverlay angezeigt wird. Ist dies der Fall, wird diese Methode aufgerufen, um den aktuellen AlertDialog zu verwerfen und entsprechend der neuen Displayeigen-

schaften einen Neuen zu erstellen, damit der Bildschirm wieder maximal ausgefüllt wird.

LOGIC

Klasse: NoteApplicationLogic (Joscha Nassenstein) Diese Klasse stellt die zentrale Stelle für die Logik der Note-Activity dar und verwaltet die anderen Klassen, welche in diesem Paket zu finden sind.

onConfigurationChanged:

Diese Methode wird aus NoteActivity aufgerufen, falls sich die Konfiguration der Geräts geändert hat. Anhand der neu übergebenen Gui-Klasse werden dabei die ClickListener neu gesetzt und die Benutzeroberfläche mit den bestehenden Daten aufgefüllt. Zusätzlich wird die Methode changeConfiguration in der ImagePopupLogic aufgerufen.

Klasse: ClickListener (Joscha Nassenstein) Die Klasse ClickListener implementiert das Interface View.OnClickListener und kann dementsprechend instanziert auf ein View als OnClickListener gesetzt werden.

onClick:

Hier wird die Methode aus der Basisklasse überschrieben und an die eigenen Anforderungen angepasst. Konkret wird überprüft, ob auf die Stichworte oder auf den Zurückpfeil in der Toolbar geklickt wurde, auf die jeweils ein ClickListener gesetzt wurde. Ist dies nicht der Fall, wird anhand der ID überprüft, ob auf ein Bild geklickt wurde und auf welches, um letztlich das Popup mit dem Bild anzuzeigen

Klasse: DialogClickListener (Joscha Nassenstein) Diese Klasse implementiert das Interface DialogInterface.OnClickListener und wird in diesem Falle auf das Menü zur Auswahl der Bildquelle gesetzt.

onClick:

Hier wird ermittelt, ob die Option Galerie oder die Option Kamera ausgewählt wurde und dementsprechend die jeweilige Methode in der Klasse ImageImport aufgerufen.

Klasse: EventDispersion (Joscha Nassenstein) Diese Klasse ist für das Erstellen und das Behandeln des Kontextmenüs zuständig, welches hier auf ein Bild im ScrollView gesetzt wird, falls der Nutzer lange auf dieses drückt.

onCreateContextMenu:

Hier wird das Menü auf den View gesetzt und die ID des Views gespeichert, um bei einer Auswahl der Option „Bild löschen“ die ID für das Bild zu haben, welches gelöscht werden soll.

onContextItemSelected:

Wird ausgewählt, dass das Bild gelöscht werden soll, wird hier die entsprechende Methode in der NoteApplicationLogic anhand der zuvor gespeicherten ID aufgerufen.

Klasse: ImageOverlayListener (Jan Beilfuß) Diese Klasse implementiert das Interface DialogInterface.OnCancelListener. textitonCancel Diese Methode wird aufgerufen, wenn der Dialog, welchem der KlickListener hinzugefügt wurde durch den Nutzer geschlossen wird. Es werden dann zum einen alle Bitmaps, mit Bildern in Originalgröße aus dem Arbeitsspeicher gelöscht und das ImageOverlay ebenfalls auf null gesetzt, um den Speicherverbrauch zu optimieren.

Klasse: MenuItemClickListener (Joscha Nassenstein) Diese Klasse implementiert das Interface Toolbar.OnMenuItemClickListener und ist für das Behandeln der Optionen im Menü der Toolbar zuständig.

onMenuItemClick:

Falls der Nutzer eine Option ausgewählt hat, wird hier die Methode onMenuItemClicked in der Klasse NoteClickHandler aufgerufen und das MenuItem als Parameter übergeben.

Klasse: MotionListener (Joscha Nassenstein) Die Klasse MotionListener implementiert das Interface View.OnTouchListener und wird hier als OnTouchListener auf den ImageView des Popups gesetzt, um Wischgesten erkennen zu können.

onTouch:

Die Methode onTouch überschreibt die Methode der Basisklasse und ermittelt anhand des übermittelten MotionEvent, ob und falls ja, in welche Richtung gewischt wurde.

Die minimale Distanz ist dabei in NoteConstants festgelegt. Anhand der Richtung wird bei Erfüllung dieses Kriteriums die jeweilige Methode in NoteImagePopupLogic aufgerufen. Bei einer horizontalen Wischgeste wird dabei das nächste bzw. vorherige Bild aufgerufen (falls dies existiert) und bei einer vertikalen Wischgeste wird das Overlay geschlossen.

Klasse: NoteListenerHandler (Joscha Nassenstein) Diese Klasse beinhaltet alle Methoden, welche aus den onClick-Methoden der Listener- bzw. Watcher-Klassen aufgerufen werden und bestimmte Aktionen ausführen.

onImageClicked:

Diese Methode wird aufgerufen, falls auf ein Bild beziehungsweise das „Bild hinzufügen“-Icon im ScrollView geklickt wurde. In letzterem Falle wird das Importieren eines Bildes initialisiert. Falls ein Bild angeklickt wurde, wird das Popup zur Vollbildanzeige des Bildes initialisiert.

onTagsClicked:

Falls der TextView, welcher die Stichworte enthält, angeklickt wurde, wird die Methode startTagActivity() in der NoteNavigationLogic aufgerufen, welche die neue Activity zur Verwaltung der Stichwörter startet.

onTextChanged:

Hier wird, je nachdem, ob der Text im Titel oder in der Beschreibung geändert wurde, stets der aktuelle Stand in das Note-Objekt gespeichert.

onMenuItemClicked:

Diese Methode wird aus dem menü der Toolbar heraus aufgerufen und ist je nach Auswahl dafür zuständig, den „Notiz teilen“-Vorgang aus dem ShareModule heraus zu starten beziehungsweise die Notiz zu löschen und zur Übersicht zurückzukehren (mittels der Methode returnToOverview() der NoteNavigationLogic).

Klasse: NoteListenerInitializer (Joscha Nassenstein) Diese Klasse hat den Click-Listener der Note-Activity als Attribut enthalten und ist für dessen Initialisieren zuständig.

initListener:

In dieser Methode wird der ClickListener auf die verschiedenen Views gesetzt. Zu den

Views zählen das LinearLayout, welches im ScrollView die einzelnen Bilder und das Icon beinhaltet, sowie der TextView mit den Stichwörtern und der Zurück-Button der Toolbar. Zusätzlich werden auf die EditText-Views für Titel und Beschreibung jeweils ein TextWatcher gesetzt und der MenuItemClickListener auf das Toolbar-Menü gesetzt.

Klasse: NoteTextWatcher (Joscha Nassenstein) Der NoteTextWatcher implementiert das Interface android.text.TextWatcher und ist zur Speicherung von Eingaben in Titel und Beschreibung zuständig.

onTextChanged:

Diese Methode überschreibt die entsprechende Methode der Basisklasse und ist dafür zuständig, die OnTextChanged-Methode im NoteClickHandler aufzurufen, falls ein Text geändert wurde, und die entsprechende CharSequence und den View, in welchem die Änderung stattgefunden hat, zu übergeben.

Klasse: NoteImageImportLogic (Joscha Nassenstein) In dieser Klasse sind die Funktionen beinhaltet, welche das Importieren von Bildern aus Kamera und Galerie anstoßen.

requestImageSource:

In dieser Methode wird zunächst überprüft, ob das Gerät eine Kamera besitzt. Ist dies nicht der Fall, wird die Methode importImageFromGallery aufgerufen. Falls das Gerät eine Kamera besitzt, wird dem Nutzer ein Dialog angezeigt, welcher abfragt, ob der Benutzer ein Bild aufnehmen oder aus der Gallerie importieren möchte. Der DialogClickListener, welcher auf den AlertDialog gesetzt wird, erfasst die Benutzereingabe.

importImageFromCamera:

In dieser Methode wird ein Intent erstellt, welcher anhand der Methode Activity.startActivityForResult die Android-eigene Kamera-Activity startet. Um die Aufnahme als Datei speichern zu können, wird durch den ImageService, welcher in den Activity-übergreifenden Klassen zu finden ist, eine Bilddatei erstellt (Methode: createImageFile) und die entsprechende Uri durch den Intent an die Kamera-Activity mitgegeben. In Android Version unter Android 6.0 („Marshmallow“) müssen diesem Intent zusätzlich zwei Flags mitgegeben werden, welche das Lesen und Schreiben dieser Uri erlauben und somit eine SecurityException

vermeiden. Die Kamera-Activity wird anhand eines in NoteConstants festgelegten, spezifischen RequestCodes gestartet, um das Ergebnis nach Rückkehr richtig zuordnen zu können.

importImageFromGallery:

Hier wird ein Intent erstellt, in welchem nach Start der Activity ein Bild aus der Gallerie importiert werden kann. Da der Pfad zu dem jeweiligen Bild direkt aus dem Result extrahiert werden kann, muss hier keine Datei erstellt werden. Auch hier wird ein in NoteConstants spezifizierter Requestcode mitgegeben.

Klasse: NoteAsyncLoadingLogic (Jan Beilfuß) Diese Klasse bündelt alle Methoden, welche beim asynchronen Nachladen von Bildern aufgerufen werden, um Veränderungen in der Oberfläche zu bewirken.

addAsyncPreviewImage:

Diese Methode wird aus dem asynchronen Task LoadPreviewImageTask aufgerufen. Sie fügt das Bild der Liste an Vorschaubildern in NoteDataGui hinzu und lässt das Bild animiert in der Gui hinzufügen.

startLoadingSpinner:

Diese Methode ruft in der Gui die Methode textitdisabledAll auf. Dadurch werden die Texteingabefelder und das Tagfeld deaktiviert und reagieren nicht mehr auf Klick-Events. Weiterhin wird der Ladespinner angezeigt. *stopLoadingSpinner:*

Diese Methode ruft in der Gui die Methode textitenableAll auf. Dadurch werden die Texteingabefelder und das Tagfeld wieder aktiviert und reagieren auf Klick-Events. Weiterhin wird der Ladespinner wieder ausgeblendet.

Klasse: NoteGalleryImportLogic (Jan Beilfuß) Diese Klasse bündelt alle Methoden, die den Bilderimport aus der Gallerie managen.

getPath:

Diese Methode ermittelt aus dem Context und der aus der Gallery Intent zurückgegeben Uri den absoluten Pfad. Ein Objekt der Klasse Cursor zeigt auf die entsprechende Datei, und gibt den Pfad dieser zurück. Im Anschluss wird dieses geschlossen und der Pfad zurückgegeben. *importImageFromGallery:*

Diese Methode bildet den gesamten Prozess ab, um ein Bild aus der Gallerie zu Importieren. Beginnend beim Pfinden des Pfads, der Kompression, falls das Bild zu groß ist und die Korrektur der Rotation. Danach wird in NoteData die Methode `textitad-dImageFromGallery` aufgerufen, um das Bild den Daten hinzuzufügen und danach die Oberfläche aktualisiert.

Klasse: NoteGuiRefresherLogic (Joscha Nassenstein) Diese Klasse ist dafür zuständig, die Benutzeroberfläche zu aktualisieren, beispielsweise bei Aufruf der Activity und bei einer Konfigurationsänderung.

dataToGui:

Aus dieser Methode heraus werden die Farben der Benutzeroberfläche gesetzt und anhand des Note-Objekts die Felder Titel, Beschreibung und Stichworte mit Inhalt gefüllt. Anschließend wird die nachfolgende Methode zur Erneuerung der Vorschaubilder aufgerufen.

refreshImages:

In dieser Methode werden zunächst alle Views aus dem LinearLayout, welches im ScrollView der Activity die Bilder beinhaltet, gelöscht. Anschließend werden alle Preview-Images aus NoteData auf das LinearLayout gesetzt und jeweils mit einer ID und einem ClickListener versehen. Als letzter Schritt wird das „Bild hinzufügen“-Icon auf das LinearLayout gesetzt und ebenfalls mit einer ID und einem Clicklistener versehen.

Klasse: ImagePopupLogic (Joscha Nassenstein und Jan Beilfuß) In dieser Klasse wird das ImageOverlay erstellt und gestartet.

openImagePopup (Joscha Nassenstein):

In dieser Methode werden die aktuellen Displayeigenschaften abgefragt und anhand der aktuellen Maßen das ImageOverlay erstellt und gestartet.

changeConfiguration (Jan Beilfuß):

Diese Methode wird ausgeführt wenn man ein Image-Popup geöffnet hat und den Bildschirm öffnet. Über die neue Konfiguration kann man sich die Bildschirmmaße nach der

Drehung in dp ausgeben lassen. Über die Pixeldichte wird Größe auf Pixel hochgerechnet und der Methode `textitchangeOrientation` des `ImageOverlays` mitgegeben. Diese erzeugt das Overlay dann auf die neue Displaygröße angepasst.

displayPreviousImage / displayNextImage :

Falls eine Wischgeste von links nach rechts auf dem `ImagePopup` erkannt wurde, welche die Mindestdistanz in `NoteConstants` übertrifft, wird diese Methode ausgeführt. Hier wird überprüft, ob ein vorheriges Bild existiert. Falls dies der Fall ist, wird die Methode `displayNewImage` mit der neuen ID aufgerufen. In der Methode `displayNextImage` wird der gegenteilige Fall behandelt.

displaynewImage:

Hier wird der aktuelle `AlertDialog` geschlossen, die geladenen Bitmaps zur Speicheroptimierung gelöscht und anschließend anhand der übergebenen ID ein neuer `AlertDialog` aufgerufen.

Klasse: NoteNavigationLogic (Joscha Nassenstein) In dieser Klasse werden alle Interaktionen mit anderen Activities gesteuert.

saveAndReturnToOverview:

Hier wird die Methode `executeSaveRoutine` in `NoteData` ausgeführt und anschließend die `NoteActivity` beendet, wodurch der Nutzer zu der Übersicht zurückkehrt.

onActivityReturned:

In dieser Methode wird anhand des übergebenen Requestcodes ermittelt, aus welcher zuvor gestarteten Activity das Ergebnis stammt. Zu den Activities gehören die Android Kamera- und Galeriefunktion sowie die `TagActivity` zur Verwaltung der Stichwörter. Anschließend wird das Ergebnis in anderen Logik-Klassen weiterverarbeitet. Im Falle der Stichwörter werden diese direkt in das Note-Objekt und die Oberfläche eingefügt.

startTagActivity:

Hier wird der Intent erstellt, welcher für das Starten der `TagActivity` genutzt wird. Dem Intent werden die aktuelle Stichwortliste sowie die Farbe und die Akzentfarbe der Notiz mitgegeben. Anschließend wird die `TagActivity` gestartet.

5.2.5 NoteTag Activity

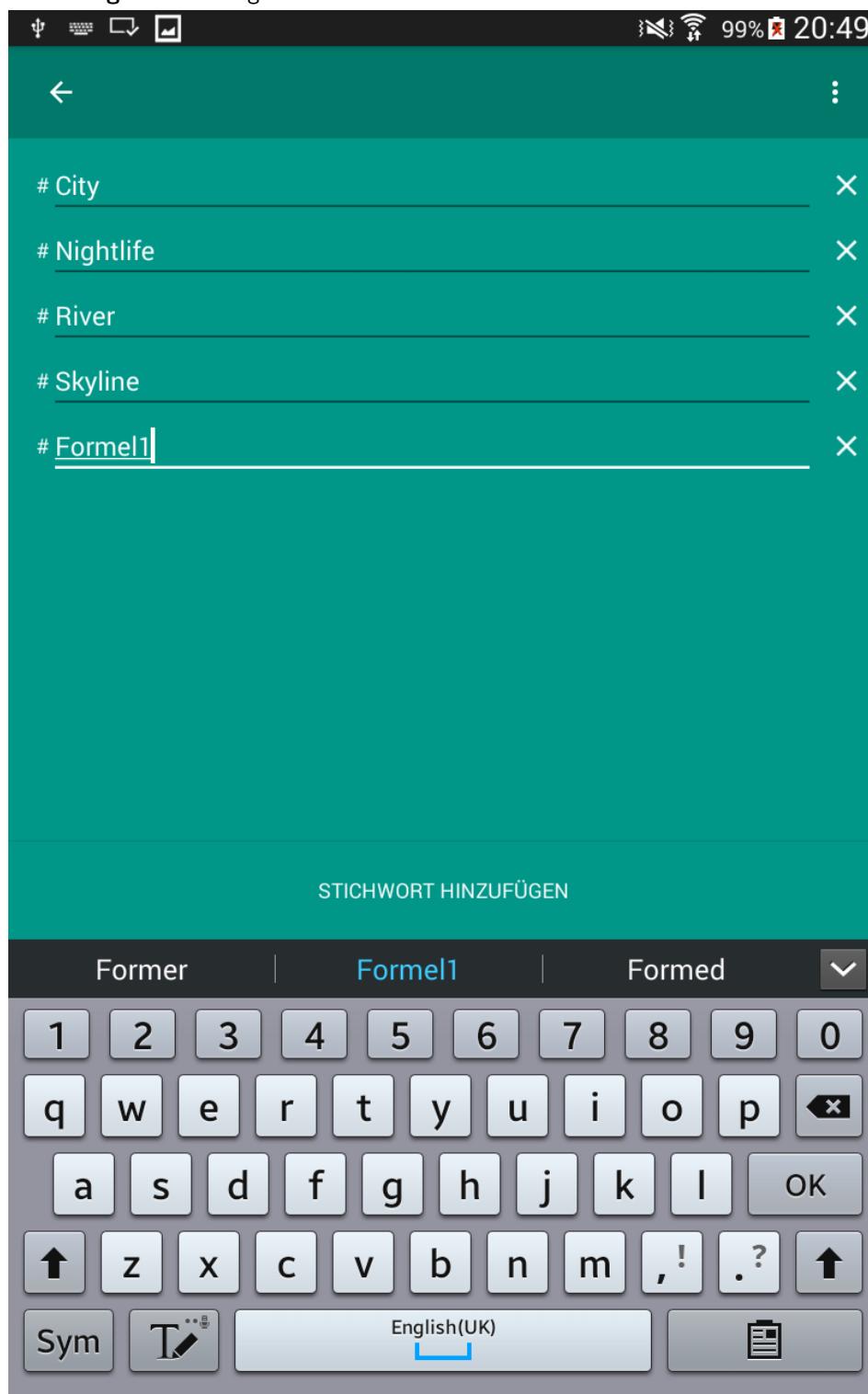
Aus der Notiz-Ansicht heraus kann man sich zwar die Stichwörter zu der jeweiligen Notiz ansehen, diese können jedoch nicht bearbeitet werden. Für die Bearbeitung der Liste steht eine weitere Activity zur Verfügung, dessen Erstellungsprozess im Folgenden beschrieben werden soll.

5.2.5.1 Aufgabe und Funktion (Joscha Nassenstein) Die Aufgabe der Activity ist, dem Nutzer das Bearbeiten der Liste zu ermöglichen. Die Activity wird stets aus der Notiz heraus gestartet und bekommt die aktuelle Liste der Stichwörter sowie die beiden Farben der Notiz übergeben. Beim Beenden der Activity wird die Liste an die Note-Activity zurückgegeben. Bei der Bearbeitung können die einzelnen Stichworte geändert, weitere Stichworte hinzugefügt und Stichworte gelöscht werden. Zusätzlich besteht eine Option, alle Stichwörter direkt zu löschen.

5.2.5.2 Layout, Screenshots (Joscha Nassenstein) Für die NoteTag-Activity war das Erstellen von zwei Layouts nötig. Aufgrund der Einfachheit des Layouts wurde auf eine andere Darstellung je nach Ausrichtung des Geräts verzichtet.

Zunächst wurde hier die allgemeine Ansicht erstellt. Es umfasst die Toolbar, einen ListView zum Anzeigen der Stichworte, einen Separator in Form eines 1dp-hohen Views sowie einen Button zum Hinzufügen weiterer Stichworte.

Für die einzelnen Zeilen des ListViews wurde anschließend ein separates Layout erstellt, welches einen TextView zum Anzeigen der Raute, ein EditText zum Bearbeiten des jeweiligen Stichwortes sowie einen ImageButton umfasst, welcher in Form eines X das Löschen des jeweiligen Stichwortes ermöglicht. Im Folgenden ist das Layout dargestellt.

Abbildung 38: NoteTags Stichwortansicht

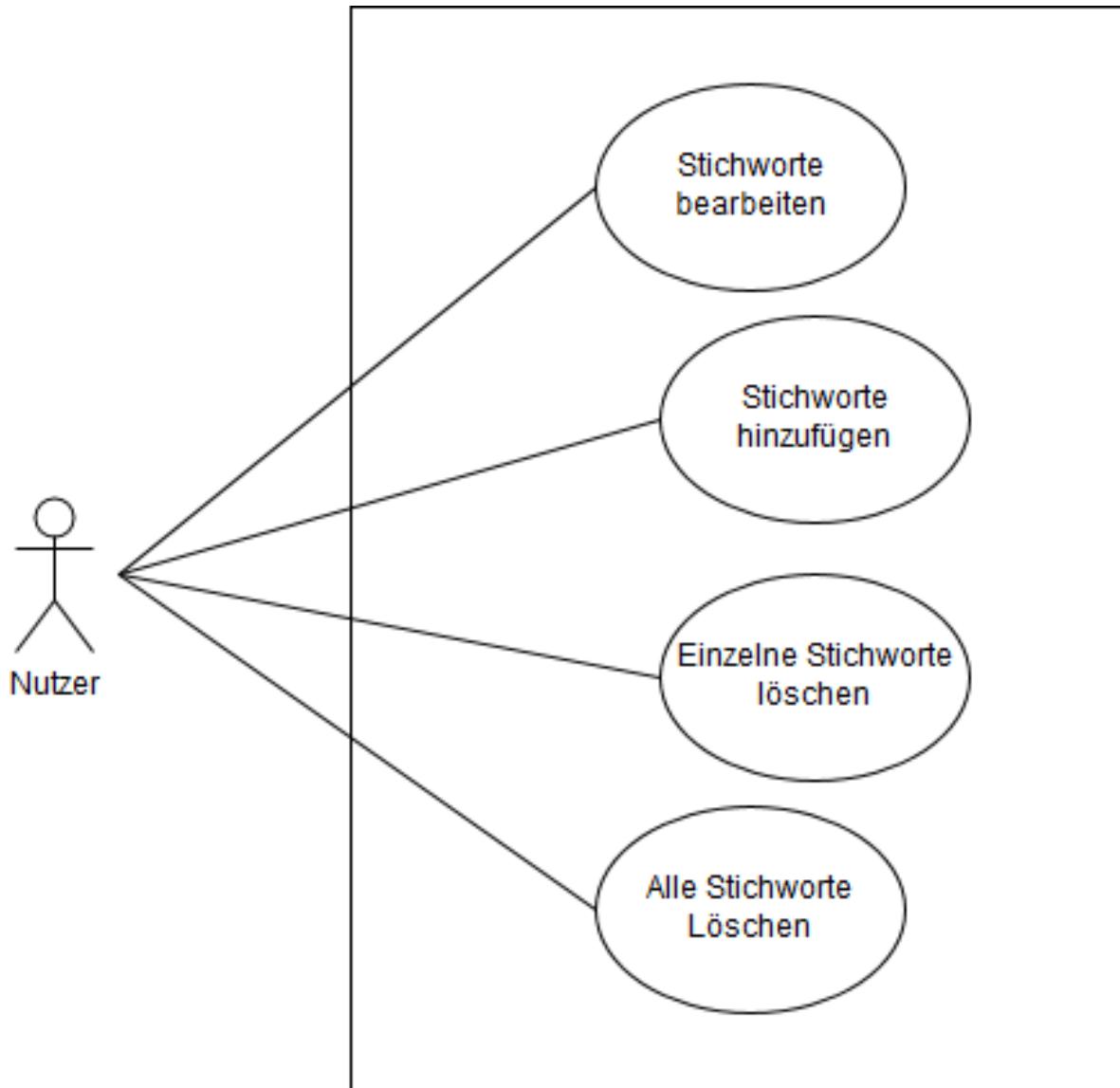
Quelle: Screenshot aus der Benutzeroberfläche

Im oberen Bildschirmbereich ist die Toolbar zu sehen. Darunter findet sich der List-View, welcher dynamisch erweitert werden kann und zu Beginn mit den übergebenen Stichworten initialisiert wird. Auf der rechten Seite können die einzelnen Stichworte gelöscht werden.

Unten befindet sich der Button zum Hinzufügen von Stichworten. Dieser wird dabei stets am unteren Rand des Displays angezeigt und verschiebt sich mit der Tastatur nach oben, falls diese aufgeklappt wird.

Die Farben der Activity werden aus der Notiz, aus welcher der Aufruf erfolgt ist, übernommen. Die Toolbar ermöglicht das Zurückkehren zur Notiz sowie das Löschen aller Stichworte über das Menü.

5.2.5.3 Use-Case (Joscha Nassenstein) Der Use-Case dieser Activity besteht darin, die Stichwortliste umfassend bearbeiten zu können. Die Möglichkeiten wurden dabei bereits in dem vorangegangenen Kapitel Aufgabe und Funktion dargestellt.

Abbildung 39: NoteTag Use-Case Diagramm

Quelle: Erstellt von Joscha Nassenstein

5.2.5.4 Datenstruktur und -typen (Joscha Nassenstein) Da die NoteTag-Activity nur einen sehr kleinen Umfang hat und deshalb nur wenige Klassen besitzt, wurde hier auf eine Unterstrukturierung in Packages verzichtet. Die Struktur orientiert sich an der in der Vorlesung vorgestellten Aufteilung. Alle Klassen mit Ausnahme der NoteTagActivity-Klasse sind dabei package-private, damit von außerhalb des Paketes nicht darauf zugegriffen werden kann.

5.2.5.5 Dokumentation des Quelltextes der Activity (Joscha Nassenstein) Im Nachfolgenden sind die einzelnen Klassen der NoteTag-Activity mit den jeweiligen wichtigsten Funktionen kurz beschrieben.

Klasse: NoteTagActivity Diese Klasse ist der zentrale Einstiegspunkt der NoteTag-Activity und ist aus der AppCompatActivity-Klasse abgeleitet. Hier werden einige Methoden letzterer überschrieben und an die eigenen Anforderungen angepasst.

onCreate:

In dieser Methode werden die über den Intent übergebenen Informationen, darunter die Stichwortliste, die Farbe und die Akzentfarbe, ausgelesen und mithilfe dieser die ApplicationLogic initialisiert. Zusätzlich wird die Gui-Klasse initialisiert.

Klasse: ApplicationLogic Die ApplicationLogic-Klasse beinhaltet die Logik der NoteTag-Activity. Zusätzlich ist hier die ArrayList mit den Stichwörtern beinhaltet, da dies die einzigen Daten sind, welche für die NoteTag-Activity gehalten werden müssen und eine dedizierte Data-Klasse dafür zu viel Overhead produzieren würde.

returnToNoteActivity:

In dieser Methode wird der Intent gebildet, welche als Result für die Activity gesetzt wird, damit dieser bei Abschluss der Activity an die NoteActivity übergeben wird. Zunächst werden dafür alle leeren Felder aus der Stichwortliste gelöscht und anschließend die Liste zum Intent hinzugefügt.

onDeleteButtonClicked:

Hier wird anhand der übergebenen ID das entsprechende Stichwort gelöscht, der Adapter für den ListView benachrichtigt und anschließend die Tastatur ausgeblendet, falls diese eingeblendet war.

onTextChanged:

Hier wird der übergebene Wert anhand der ID des Views, in welchem der Text geändert wurde, in die Stichwortliste eingefügt.

addInputTagField:

In dieser Methode wird ein leerer String an die Stichwortliste angefügt und anschließend der Adapter benachrichtigt, sodass ein neues Feld für die Benutzereingabe erscheint.

onMenuItemClick:

Diese Methode wird aufgerufen, falls der Benutzer im Toolbarmenü die Option „Alle Stichworte löschen“ ausgewählt hat. Hier wird die Liste der Stichworte geleert und anschließend zur Note-Activity zurückgekehrt. Eine weitere Abfrage ist hier nicht implementiert, da für das Löschen aller Stichworte so schon zwei Klicks nötig sind.

Klasse: ClickListener Die Klasse ClickListener implementiert das Interface View.OnClickListener und kann dementsprechend auf ein View gesetzt werden, um die Benutzereingabe zu erkennen und Aktionen durchzuführen.

onClick:

Hier wird überprüft, auf welchem View das Klick-Event ausgelöst wurde. Zur Auswahl stehen dabei der Zurück-Button der Toolbar, der Button, um ein Stichwort hinzuzufügen, sowie die einzelnen Entfernen-Buttons innerhalb des Listviews für jedes Stichwort. Je nach Ort der Ausführung wird die entsprechende Methode innerhalb der ApplicationLogic ausgeführt.

Klasse: Gui Die Gui-Klasse beinhaltet die Benutzeroberflächenelemente als Attribute, wodurch auf diese zugegriffen werden kann.

initiateListView:

In dieser Methode wird der als Parameter übergebene ListViewAdapter auf den ListView gesetzt.

hideKeyboard:

Hier wird das SoftKeyboard ausgeblendet.

Klasse: ListViewAdapter Die Klasse ListViewAdapter erweitert ArrayAdapter<String> und kann dadurch auf einen ListView gesetzt werden.

getView:

In dieser Methode wird eine Zeile des ListViews anhand des in den Layouts spezifizierten Zeilenlayouts initialisiert, die Position in der Liste als IDs auf die Views gesetzt und jeweils ein ClickListener für den Löschen-Button und ein TextWatcher für das Stichwort gesetzt. Anschließend wird der View zurückgegeben.

Klasse: MenuItemClickListener Diese Klasse implementiert das Interface Toolbar.OnMenuItemClickListener und ist dafür zuständig, das Klicken auf ein Menüitem in der Toolbar, hier „Alle Stichwörte löschen“, zu behandeln und dementsprechend hier die Methode in ApplicationLogic aufzurufen.

Klasse: TextWatcher Der NoteTextWatcher implementiert das Interface android.text.TextWatcher und ist zur Speicherung von Eingaben in einem Stichwortfeld zuständig.

onTextChanged:

Diese Methode überschreibt die entsprechende Methode der Basisklasse und ist dafür zuständig, die OnTextChanged-Methode in der ApplicationLogic aufzurufen, wo der Text in der ArrayList gespeichert wird.

afterTextChanged:

In dieser Methode wird verhindert, dass die Texte Leerzeichen enthalten, da Stichworte immer als ein Wort geschrieben werden sollen.

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)

5.4.1 Persistente Datenhaltung

5.4.1.1 Objekte Im Bereich der persistenten Datenspeicherung gab es die Herausforderung, dass es eine breite Auswahl an Speichermöglichkeiten gibt. Es musste zu Beginn evaluiert werden, welche Technologie für das Projekt am geeignetsten war. Zur Auswahl standen Bundles und Datenbanken, welche sich weiter in SQL-Datenbanken und NoSQL-Datenbanken unterteilen lassen. Darüber hinaus war zu Projektbeginn nicht bekannt, welche Datenbankimplementierungen für den Einsatz auf mobilen Geräten geeignet sind. Nach einiger Recherche wurde die Implementierung im Bundle als zu aufwendig und nicht flexibel genug erachtet. In der Folge wurde sich für den Einsatz einer Datenbank entschieden.

Aus dem bisherigen Studienverlauf waren unter den Datenbanken nur SQL-Datenbanken bekannt. Diese unterscheiden sich von NoSQL-Datenbanken, indem man eine klare Datenstruktur, in der Regel in Tabellenform vorgibt. NoSQL-Datenbanken sind beispielsweise dokumenten- oder graphenbasiert. Dokumente sind flexibel und diesen können einfache Felder in Form von Key-Value-Paaren hinzugefügt werden. Im mobilen Umfeld hat man häufig Änderungen am Datenmodell, sodass einem in diesem Bereich eine NoSQL-Datenbank entgegen kommt. Weiterhin hat man bei SQL-Datenbanken einen Aufwand zur Erzeugung und Definition von Tabellen sowie bei deren Änderung. Und da man aus den Anforderungen an die App ableiten konnte, dass keine komplexen Tabellenrechnungen gefordert waren, wurde sich für eine dokumentenbasierte NoSQL-Datenbank entschieden.

Bei der Wahl der spezifischen Datenbanktechnologie entschied man sich am Ende für Couchbase Lite, da diese gut dokumentiert ist, einem eine lokale Datenbank und Tools wie ein Gateway zur Hand gibt, wenn man es langfristig in Erwägung zieht Todos, Events und Notizen über einen Server zu synchronisieren.

Um den Aufwand bei der Umwandlung von Java-Objekten zu Dokumenten zu vereinfachen, wurde das Serialisierungs- und Deserialisierungsframework Jackson verwendet. Dieses wandelt Java Objekte in JSON-Strings um, welche in jeweils einem Dokument abgelegt wurden. Weiterhin bietet es eine Funktionalität, diese wieder zurückzuwandeln. Einige Attribute, darunter die ID des TEN-Objektes, die Hauptfarbe, die Kontrastfarbe und das Erzeugungsdatum wurden hiervon ausgenommen.

Abbildung 40: Dokumentenstruktur

Dokument

```

graph TD
    A["ObjectKey: Objekt in JSON-String serialisiert  
TypeKey: Klasse des Java Objektes (Event/Note/Todo)  
CreationDateKey: Datum, an dem das TEN-Objekt erzeugt wurde  
ColorKey: Primärfarbe des TEN-Objektes  
AccentColorKey: Akzentfarbe des TEN-Objektes"]
    B["id: Dokumenten-ID"]
    A --- B
  
```

Quelle: Erstellt von Jan Beilfuß

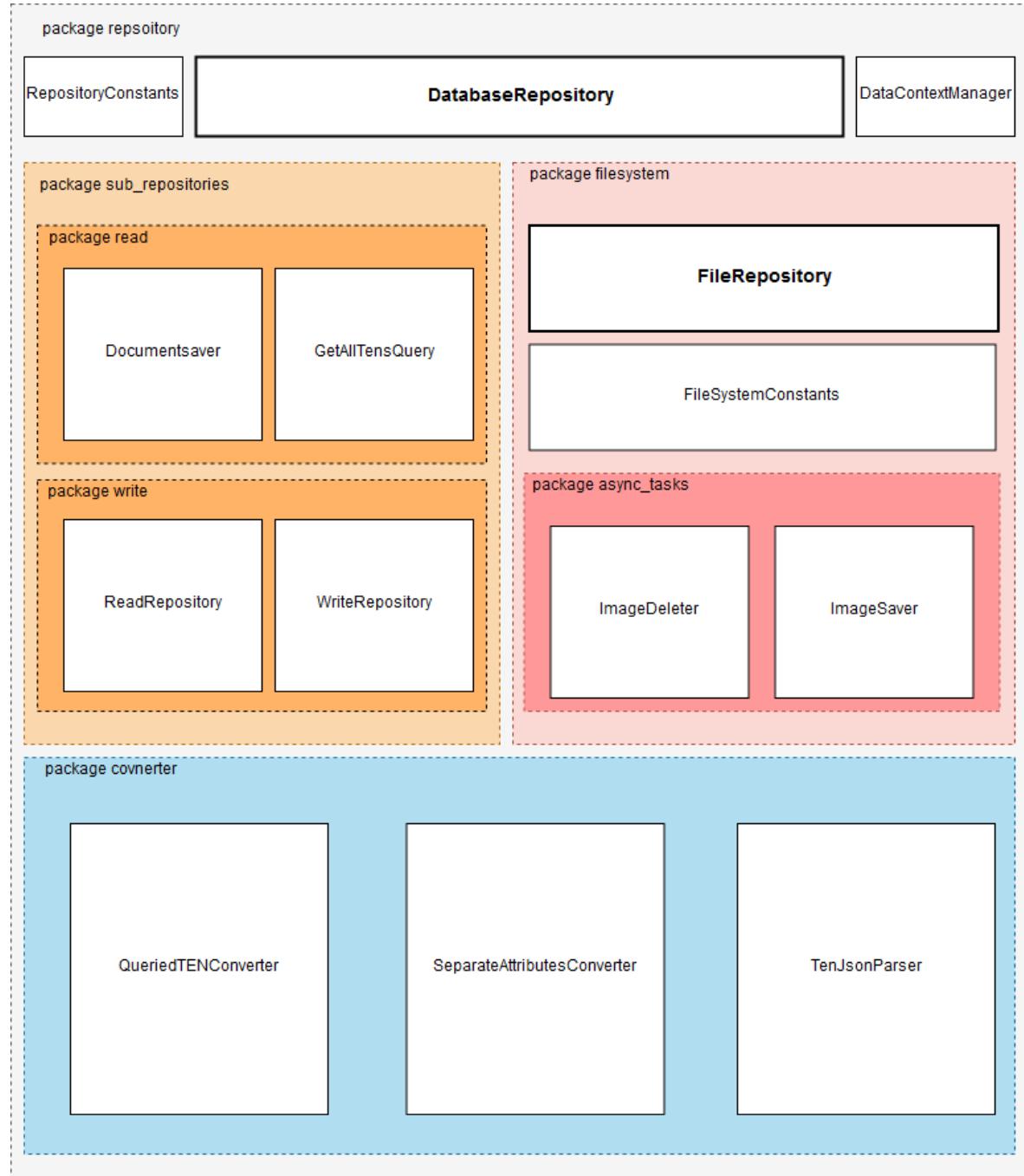
Die ID ist die ID des Dokumentes selber und würde somit doppelt gehalten werden. Die Farben wurden in eigenen Key-Value-Paaren abgelegt, damit man nicht das gesamte TEN-Objekt erzeugen muss, wenn man auf diese zugreifen möchte. Dies fand Einsatz in der Note Activity. Hier war es das Ziel die Performance beim Öffnen zu optimieren. Daher wurden die Farben getrennt von den eigentlichen Notizdaten im Vorraus geladen. Weiterhin wurde im Dokument gespeichert, welche Klasse das speichernde Objekt hat, damit dieses beim Laden vernünftig aus dem JSON-String erzeugt werden kann. Zusätzlich wurden noch die Objektattribute welche bei Datenbankabfragen genutzt wurden, in einem eigenen Key-Value-Pair abgespeichert.

5.4.1.2 Bilder Das Speichern der Bilder hat sich im Vergleich zu den Java-Objekten als aufwendiger erwiesen, da diese mehr Speicherplatz und Rechenleistung benötigen.

Der erste Lösungsansatz war, die Bilder als Binary Large Objects in der Datenbank abzulegen. Dies erwies sich als zu imperfekt und umständlich, sodass entschieden wurde, die Bilder im Dateisystem des Gerätes zu speichern. Weiterhin wurde zwischen den Bildern in hoher Auflösung und Bildern in Vorschaugröße unterschieden. Die Vorschaubilder werden in der Übersicht und Notiz-Activity eingebunden. Aufgrund der stärkeren Kompression und geringeren Auflösung können diese schneller geladen werden. Darüber hinaus ist es auf dem Zielgerät hardwarebedingt nicht möglich, mehrere Bilder in Originalgröße im Arbeitsspeicher zu halten.

5.4.2 Repository

Das Repository bildet in der Applikationsstruktur die unterste Ebene und ist für das Handling der Datenbank und die Bereitstellung der persistent gespeicherten Daten verantwortlich.

Abbildung 41: Paketstruktur in der Repository-Schicht

Quelle: Erstellt von Jan Beilfuß

5.4.2.1 Klassenstruktur

Klasse: DatabaseRepository (Jan Beilfuß) Diese Klasse wird aus der Service-Schicht angesprochen. Sie bündelt Methoden, um Daten aus der Datenbank zu lesen oder zu schreiben. Alle Methoden rufen Methoden in den Klassen WriteRepository oder ReadRepository auf.

getTodoByID:

Diese Methode gibt das zu der ID passende und in der Datenbank abgespeicherte Todoobjekt zurück.

getEventByID :

Diese Methode gibt das zu der ID passende und in der Datenbank abgespeicherte Eventobjekt zurück.

getNoteByID :

Diese Methode gibt das zu der ID passende und in der Datenbank abgespeicherte Noteobjekt zurück.

insertTEN :

Diese Methode speichert ein neues TEN-Objekt in die Datenbank.

updateTEN :

Diese Methode aktualisiert ein bereits bestehendes TEN-Objekt.

getAllTENs :

Diese Methode gibt eine Liste aller in der Datenbank liegenden TEN-Objekte zurück.

deleteTEN :

Diese Methode löscht das in der Datenbank liegende Objekt zu der übergebenen ID.

getTENColors :

Diese Methode gibt die Haupt- und die Akzentfarbe des zu der übergebenen ID passenden TEN-Objektes zurück.

Klasse: DataContextManager (Jan Beilfuß) Diese Klasse managt das Handling des Datenbankenobjektes.

initDatabase :

Diese Methode prüft, ob die Datenbank mit dem übergebenen Datenbanknamen auf

dem Gerät liegt. Falls dies der Fall ist wird diese als Datenbankobjekt in die Applikation geladen. Andernfalls wird eine neue erzeugt.

compactDatabase :

Diese Methode räumt die Datenbank auf. Da gelöschte Dokumente nur mit einem Flag versehen, aber nicht endgültig gelöscht werden, verbrauchen diese weiterhin Speicherplatz.

getNumberOfDocuments :

Diese Methode gibt die Anzahl der Datensätze zurück. Sie findet bei der Arbeit mit Mockdaten ihre Anwendung. Dadurch wird verhindert, dass diese mehrfach in die Datenbank geschrieben werden. *getDatabase :*

Diese Methode gibt das Datenbankobjekt zurück.

Klasse: RepositoryConstants (Jan Beilfuß) Diese Klasse enthält alle Konstanten, die bei der Arbeit mit der Datenbank benötigt werden. Dazu zählen die in den Dokumenten eingesetzten Schlüssel, aber beispielsweise auch der Name der Datenbank.

Klasse: WriteRepository (Jan Beilfuß) Diese Klasse beinhaltet alle Funktionen, die einen schreibenden Zugriff auf die Datenbank haben.

insertTEN :

Diese Methode erzeugt eine neues veränderbares Dokument und setzt in dem übergebenen TEN-Objekt die ID des Dokumentes. Dieses Dokument wird inklusive dem übergebenen TEN-Objekt dem DocumentSaver übergeben.

updateTEN :

Diese Methode lädt das Dokument aus der Datenbank und entfernt den Schreibschutz von diesem. Dieses Dokument wird inklusive dem übergebenen TEN-Objekt dem DocumentSaver übergeben.

deleteTEN :

Diese Methode lädt das zu der übergebenen ID passende Dokument aus der Datenbank. Weiterhin wird deleteNoteImages aufgerufen. Wenn ein Dokument zu der übergebenen ID existiert, wird dieses gelöscht.

deleteNoteImages :

Diese Methode entfernt alle mit dem zu löschen Objekt assoziierten Bilder aus dem persistenten Speicher, sofern das zu löschen Objekt ein Note-Objekt ist.

Klasse: DocumentSaver (Jan Beilfuß) Diese Klasse enthält alle Methoden, welche benötigt werden, um die Attribute eines TEN-Objektes in ein Dokument zu übertragen und dieses in der Datenbank zu speichern.

updateCompleteDocument :

Diese Methode wird aus dem WriteRepository aufgerufen und bildet den ganzen Prozess von der Dokumentenvorbereitung über das Hinzufügen der einzelnen Key-Value-Paare bis hin zum finalen speichern des Dokumentes ab.

saveSeparateAttributes :

Diese Methode legt alle Attribute, welche in der Deserialisierung durch Jackson nicht beachtet werden, als Key-Value-Paare im Dokument ab. Dazu zählt das Erzeugungsdatum des TEN-Objektes, die Farbe und die Akzentfarbe.

setTypeOfDocument :

Diese Methode bestimmt abhängig von der Klasse des übergebenen TEN-Objektes, welcher Typ in dem Dokument gespeichert werden muss. Dies ist notwendig, da bei der Deserialisierung des Json-Strings, die Zielklasse angegeben werden muss.

Klasse: ReadRepository (Jan Beilfuß) Diese Klasse beinhaltet alle Methoden, die einen lesenden Zugriff auf die Datenbank haben.

getTodoByID :

Diese Methode lädt ein einzelnes Dokument aus der Datenbank und wandelt dieses in ein Todo-Objekt um. Dazu wird erst aus dem Json-String das Objekt erzeugt und im Anschluss die separat abgelegten Attribute gesetzt.

getEventByID :

Diese Methode lädt das zu der übergebenen ID passende Dokument aus der Datenbank und wandelt dieses in ein Event-Objekt um. Dazu wird erst aus dem Json-String das Objekt erzeugt und im Anschluss die separat abgelegten Attribute gesetzt.

getNoteByID :

Diese Methode lädt das zu der übergebenen ID passende Dokument aus der Datenbank und wandelt dieses in ein Note-Objekt um. Dazu wird erst aus dem Json-String das Objekt erzeugt und im Anschluss die separat abgelegten Attribute gesetzt.

getAllTENs :

Diese Methode ruft die Query getAllTENs auf und gibt dann das Ergebnis dieser, eine Liste aller in der Datenbank gespeicherten TEN-Objekte, zurück.

getTENColors :

Diese Methode lädt das Dokument zu der übergebenen ID aus der Datenbank und gibt die Haupt- und die Akzentfarbe des erzeugten TEN-Objektes zurück.

Klasse: GetAllTensQuery (Jan Beilfuß) Diese Klasse enthält die Datenbankabfrage, um alle TEN-Objekte aus der Datenbank zu laden.

getAllTENs :

Diese Methode baut eine Datenbankabfrage, um alle TEN-Objekte aus der Datenbank zu laden. Das Einträge des ResultSets werden dem entsprechenden Converter übergeben, dort umgewandelt und dann einer Liste aus TEN-Objekten hinzugefügt, welche am Ende zurückgegeben wird.

Klasse: QueriedTENConverter (Jan Beilfuß) Diese Klasse enthält eine Methode, welche genutzt wird, um die Ergebnisse einer Datenbankabfrage in TEN-Objekte umzuwandeln.

createTENFromResult :

Diese Methode wandelt ein Result, eine CouchbaseLite-Klasse, in ein TEN-Objekt um. Dem Result wird ein Dictionary entnommen und aus diesem der Json-String. Der wird im Anschluss zur Erzeugen des TEN-Objektes verwendet. Wenn dies erfolgreich ist, werden dem TEN-Objekt noch die separat abgelegten Attribute hinzugefügt.

Klasse: SeparateAttributesConverter (Jan Beilfuß) Diese Klasse ist dafür verantwortlich, dass in einem übergebenen TEN-Objekt die nicht im JSON-String enthaltenen Attribute gesetzt werden. Dazu zählt die ID, welche die ID des Dokuments selber ist. Darüber hinaus die Farbe, die Akzentfarbe und das Erzeugungsdatum des TEN-Objektes.

addTENPropertiesFromDocument :

Diese Methode entnimmt die aufgeführten Attribute einem Document, Objekt aus dem Couchbase Lite-Umfeld, und setzt diese im übergebenen TEN-Objekt.

addTENPropertiesFromResult :

Diese Methode entnimmt die aufgeführten Attribute einem Result, Objekt aus dem Couchbase Lite-Umfeld, und setzt diese im übergebenen TEN-Objekt.

Klasse: TenJsonParser (Jan Beilfuß) Diese Klasse enthält Methoden, die aus einem JSON-String die TEN-Objekte erzeugen. Dafür wird das Framework Jackson eingesetzt.

stringToTodo :

Diese Methode erzeugt aus dem übergebenen JSON-String ein Todo-Objekt und gibt dieses zurück.

stringToEvent :

Diese Methode erzeugt aus dem übergebenen JSON-String ein Event-Objekt und gibt dieses zurück.

stringToNote :

Diese Methode erzeugt aus dem übergebenen JSON-String ein Note-Objekt und gibt dieses zurück.

Klasse: FileRepository (Jan Beilfuß) Diese Klasse enthält alle Zugriffe auf das Dateisystem. Sie findet Einsatz in der Verwaltung der Bilder, da diese im Dateisystem abgelegt werden. *saveImagePersistent :*

Diese Methode erzeugt ein Objekt der Klasse ImageSaver und speichert darüber das Bild im Dateisystem.

readImageFromDirectory :

Diese Methode ist dafür verantwortlich, aus einem Image-Objekt mit leerer Bitmap und dem übergebene Ordnernamen, das entsprechende Bild zu laden. Dafür baut man aus einer Grundpfad, dem übergebenen Ordnernamen und der im Image-Objekt enthaltenen ID den Bildpfad zusammen. Im Anschluss wird die zu dem Pfad gehörende Datei zu einer Bitmap dekodiert, ein neues Image-Objekt inklusive der Bitmap erzeugt und dieses zurückgegeben.

deleteImageFromDirectory :

Die Methode deleteImageFromDirectory überladen. Diese nimmt den absoluten Pfad einer Datei an und löscht diese, falls sie existiert. Diese Methode ist synchron, da sie lediglich für einzelne Dateien beim Kameraimport verwendet wird und sich innerhalb eines vernachlässigbaren Bereichs auf die Performance auswirkt.

deleteImageFromDirectory :

Die Methode deleteImageFromDirectory überladen. Diese nimmt ein Image-Objekt an und erzeugt einen ImageDelete. Diese Klasse erzeugt einen asynchronen Task, um das Bild zu löschen. Hier wird asynchron gearbeitet, das es vorkommen kann, dass diese Methode beim schließen der Note-Activity aufgerufen wird, um alle zuvor als zu löschen markierten Bilder zu löschen. Da eine flüssige Navigation gewährleistet werden soll, wird hier der UI-Thread verlassen.

Klasse: FileSystemConstants (Jan Beilfuß) Diese Klasse enthält alle Konstanten, welche im Umgang mit dem Filesystem benötigt werden. Dazu zählen Ordernamen, Dateiendungen, aber auch der Kompressionsfaktor für Bilder.

Klasse: ImageDelete (Jan Beilfuß) Diese Klasse erzeugt einen asynchronen Task, der die Bilder aus dem Dateisystem löscht. Diese Klasse enthält als innere Klasse, die Klasse DeleteImageTask, welche von AsyncTask erbt. *execute :*

Es wird ein asynchroner Task der Klasse DeleteImageTask erzeugt und ausgeführt. Dieser löscht die Bilder zu der im ImageObjekt übergebenen ID. Die dort definierte Methode textitdoInBackground baut die Dateipfade für die beiden Ordner Originale Bilder und Vorschaubilder zusammen, überprüft, ob die Datei existiert und löscht diese, falls sie existiert.

Klasse: ImageSaver (Jan Beilfuß) Diese Klasse erzeugt einen asynchronen Task, der die Bilder im Dateisystem speichert. Diese Klasse enthält als innere Klasse, die Klasse SaveImageTask, welche von AsyncTask erbt.

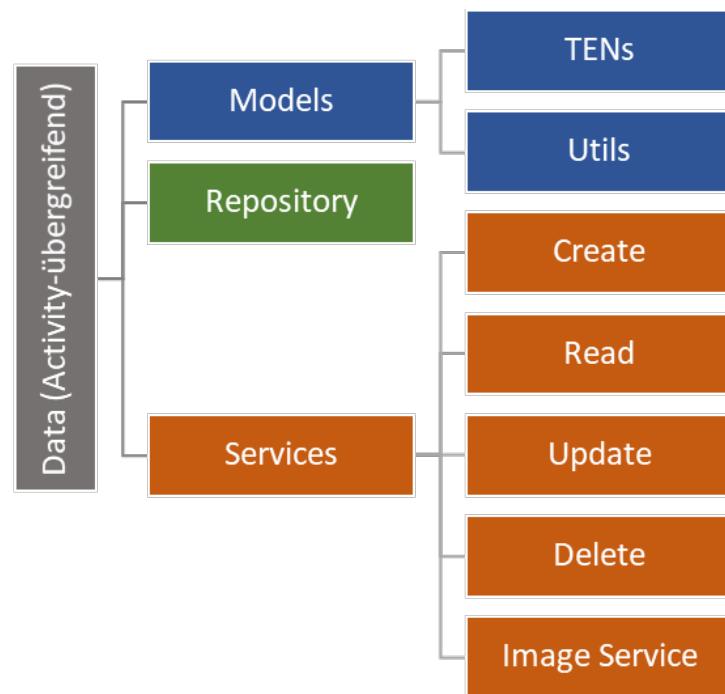
execute :

Es wird ein asynchroner Task der Klasse SaveImageTask erzeugt und ausgeführt. Die dort definierte Methode `textitdoInBackground` baut die Zielpfade für die beiden Ordner Originale Bilder und Vorschaubilder zusammen. Im Anschluss wird geprüft, ob das Bild bereits existiert, um diese nicht bei jedem Speichern zu überschreiben. Wenn die übergebene Bitmap nicht Null ist wird diese über ein Objekt der Klasse `FileOutputStream` dort abgelegt. Für den Ordner der Vorschaubilder wird zuvor ein Vorschaubild erzeugt, und dieses abgespeichert.

5.5 Dokumentation der Activity-übergreifenden Klassen

Zu den Activity-übergreifenden Klassen gehören abgesehen von den Repository-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.

Abbildung 42: Übersicht über Data-Klassen



Quelle: Erstellt von Ruthild Gilles

5.5.1 Models-Klassen (Ruthild Gilles)

Wie schon während der Planungsphase definiert, gibt es Klassen, in denen die Struktur der einzelnen TEN-Objekte definiert ist. Sowohl die Todo-Klasse als auch die Event- und Note-Klasse erben von der TEN-Klasse. In dieser werden für alle TEN-Objekte Attribute wie der Titel, eine ID, ein Erstellungsdatum und die Farbe deklariert. Die Farbe wird dabei bei Initialisierung eines Objektes zufällig aus einer Liste von Farben ausgewählt. Die Farben wurden anhand eines Material-Designs ausgewählt, damit

die graphische Benutzeroberfläche ästhetisch ansprechend ist und sich keine Farben beißen.

Die Klassen für die einzelnen Objekt-Typen enthalten weitere Attribute, die notwendig sind, um alle relevanten Informationen darstellen zu können.

In der Klasse Todo werden zusätzlich noch Attribute für den Fortschritt der Erfüllung des Todos, einer Notiz, eines Start- und Enddatums deklariert. Der wichtigste Teil eines Todo-Objektes ist die Liste an Aufgaben, die abgearbeitet werden sollen. Dazu wurde eine weitere Klasse implementiert, welche die Struktur einer einzelnen Aufgabe angibt. Aufgaben bestehen aus einer Beschreibung und einem binären Status. Objekte dieser Klasse können in einem Objekt der Klasse Todo in einer ArrayList gespeichert werden.

Die Klasse Event erweitert das TEN-Objekt noch um ein Datum inklusive Uhrzeit, eine Adresse und ein Wiederholungstyp. Dieser kann folgende Werte annehmen: None, Daily, Weekly, Monthly oder Yearly. Die Werte sind in einem Enum im Ordner Utils definiert. Zusätzlich können einem Event-Objekt mehrere Erinnerungen hinzugefügt werden. Diese werden ähnlich wie die Aufgaben eines Todo-Objektes in einer ArrayList gesammelt.

Ein Note-Objekt enthält, abgesehen von den Attributen aus der TEN-Klasse, zusätzlich eine Beschreibung, mehrere Tags und Bilder, welche in ArrayListen dem Note-Objekt hinzugefügt werden können. Ein Bild ist ein Objekt der Bild-Klasse und enthält die Attribute ID und Bitmap.

Damit für die Präsentation der Applikation bereits einige Todo-, Event- und Note-Objekte in der Benutzeroberfläche zu sehen sind, gibt es eine weitere Klasse, welche Mockdaten initialisiert. Diese werden auf die Datenbank geschrieben und von dort wie reguläre Daten verwendet.

5.5.2 Service-Klassen (Ruthild Gilles)

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.

In den Service Klassen bestehen zusätzlich zwei Klassen für die Verarbeitung von Bildern.

Die Klasse ImageService stellt dabei Funktionen für die Schnittstelle zur Datenbank zur Verfügung, beispielsweise zum Laden oder Löschen eines Bildes. Zusätzlich kann eine temporäre Bilddatei zur Übergabe an den Kamera-Intent erstellt werden, in welcher bei der Aufnahme die Bilddatei gespeichert wird. Die Klasse ImageCorrectionService stellt Funktionen zur Korrektur der Ausrichtung eines Bildes anhand der mitgelieferten Exif-Daten bereit, da auf einigen Geräten die Bilder sonst nicht korrekt dargestellt wurden. Diese Klassen werden zurzeit lediglich von der Note-Activity genutzt.

5.5.3 Modules-Klassen

5.5.3.1 Paket: modules In diesem Paket werden Funktionen gebündelt, die in mehrere Activities implementiert werden oder das Potenzial dazu haben.

Paket: image

Klasse: ImageToolsModule (Jan Beilfuß) Diese Klasse bündelt die Methoden, welche im Modul ImageTools enthalten sind und dient bei Aufrufen aus den Activities als Single Point Of Contact.

importCompressedImage :

Dieser Methode wird der Pfad zu einer Bilddatei übergeben. Sie ruft die Methode `textitimportCompressedImage` im `ImageCompressionModule` auf, welche die Bitmap zurückgibt, welche auf dem Pfad liegt.

correctImageRotation :

Dieser Methode wird der Pfad zu einer Bilddatei und die dazugehörige Bitmap übergeben. Sie ruft die Methode `textitcorretImageRotation` im `ImageCompressionModule` auf, welche die Bitmap abhängig von dem Bild zu Pfad dreht und zurückgibt.

Klasse: ImageCompressionModule (Jan Beilfuß) Diese Klasse bündelt die Methoden, welche benötigt werden, um Bilder komprimiert aus dem Dateisystem zu importieren.

importCompressedImage:

In den BitmapFactory.Options kann man einen Wert setzen, sodass beim Dekodieren nur die Informationen zu dem Bild eingelesen werden. Dazu zählt unter anderem auch die Höhe und die Breite. Diese wird dann genutzt, um einen Skalierungsfaktor zu berechnen, um die Größe des Bildes auf den Arbeitsspeicher des Zielgerätes anzupassen. Dieser wird beim Dekodieren der Bilddatei berücksichtigt, sodass ein kleineres Bild mit weniger Speicherverbrauch in den Arbeitsspeicher geladen wird.

calculateImportSampleSize:

Diese Methode berechnet den Skalierungsfaktor, welcher benötigt wird, um das Bild von der Originalgröße auf die kleinste Größe oberhalb der Zielgröße zu bekommen. Dabei ist zu beachten, dass der Faktor eine Zweierpotenz ist, da auf diese abgerundet wird. Der Faktor wird schrittweise erhöht. Sobald das Bild kleiner als die Zielgröße ist, wird der Faktor zurückgegeben.

Klasse: ShareModule (Fabia Schmid) Das ShareModule bietet jeweils für Todo, Event und Note eine Funktion, welche angesprochen werden kann, um den jeweiligen Inhalt in einer externen App zu teilen. Dafür wird ein Intent erstellt, welcher einen Benutzerdialog startet, in dem ausgewählt werden kann, in welcher Form beziehungsweise in welcher Applikation der Inhalt geteilt werden soll. Innerhalb des Intents werden anschließend die Felder Betreff und Text mit den entsprechenden Daten aus dem momentan geöffneten TEN-Objekt gefüllt. Die Struktur des Textes entspricht dabei der Struktur eines Todos, eines Events oder einer Notiz. Anschließend wird die Activity zum Anzeigen des Dialoges und dem anschließenden Teilen gestartet.

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 wurden 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 die 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, 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

Ich werde ein kurzes Fazit zum Projekt allgemein, zur Vorbereitung und zu meinem persönlichen Fortschritt im Projekt ziehen.

Die Gestaltung der Prüfungsleistung in Projektform habe ich insgesamt als sehr erfrischend wahrgenommen. Anstatt sich in theoretische Sachverhalte und Methoden zu vertiefen, hatte man hier die Möglichkeit lösungsorientiertes Denken, Kreativität und Problemlösefähigkeit anzuwenden. Dabei hat das Modul in Teilen auch auf bereits absolvierten Modulen wie PUT oder OPR aufgebaut. Ich denke, dass die Teamzusammenstellung in der Projektdurchführung eine wichtige Rolle gespielt hat. Das heterogene Skillset der Teammitglieder zahlte sich aus. Konflikte wurden zumeist konstruktiv diskutiert, haben zum Nachdenken angeregt und oft zur optimalen Lösung geführt. Der Projektablauf war gut strukturiert. Die Layouts wurden grafisch ansprechend gestaltet und den technischen Herausforderungen wurde mit performanceorientierten Lösungen entgegengetreten. Dies spiegelt sich insbesondere im Projektergebnis wider. Die Applikation ist aus meiner Sicht sehr gelungen. Dazu hat aber auch die intensive Kommunikation untereinander und die motivierende und hilfsbereite Atmosphäre beigetragen.

Die Vorbereitung des Projekts hat einem ein grundlegendes Verständnis der Entwicklung einer Applikation für Android vermittelt, jedoch hätte ich mich gefreut, wenn ein bis zwei Veranstaltungen den Themen Applikationsstruktur und Versionsverwaltung mit beispielsweise Git gewidmet worden wären, anstatt sich beispielsweise in technische Lösungen für die Manipulation von Farbwerten zu vertiefen. Insbesondere die Versionsverwaltung war integrativer Bestandteil des Projektes.

Für mich persönlich kann ich sagen, dass die Projektdurchführung mir das mir vorher unbekannte Gebiet der Entwicklung von Android Applikation offen gelegt hat. Darüber hinaus konnte ich durch die Beschäftigung mit Bildern und Bitmaps viel in den Bereichen Code- und Speicheroptimierung lernen. Insgesamt konnte ich die Fähigkeit Problemlösefähigkeit im Projekt immer wieder schulen. Zusätzlich konnte ich viel über die Abstimmung mit anderen Projektmitgliedern bei der Arbeit im Activity-übergreifenden Data-Bereich, aber auch in der erfolgreichen Zusammenarbeit mit Herrn Nassenstein

an der Notiz-Activity, lernen. Jedoch hat einen der hohe zeitliche Aufwand, der mit der Implementierung und dem Herantasten an neue Technologien verbunden war, einen in einigen sehr prüfungsintensiven Zeiträumen an die Grenze gebracht.

6.4 Fazit von Joscha Nassenstein

Das Modul „Projekte der Wirtschaftsinformatik“ und damit das Lernen der Programmierung von Android-Apps hat für mich eine angenehme Abwechslung von den sonst doch sehr theorielastigen Inhalten anderer Vorlesungen geboten. Im Vergleich zum Vorjahr fand ich persönlich die Aufgabenstellung spannender und ich stelle mir die entstandene App auch nützlicher vor, als sie es sonst gewesen wäre.

Das Programmieren der App selbst hat mit einerseits sehr viel Freude bereites, andererseits hat es jedoch auch sehr viel Ressourcen in einem bereits mit vielen Prüfungsleistungen gefüllten Zeitraum gekostet. Während der Start etwas holprig verlief, da zunächst eine Grundlage für die mir zugeteilte Activity geschaffen werden musste, fand ich recht schnell Spaß an der Sache und investierte vor allem in der Zeit zwischen Weihnachten und Neujahr sehr viel Zeit in das Projekt. Der Funktionsumfang wuchs stetig und es musste auch darauf geachtet werden, diesen nicht zu groß werden zu lassen und die Übersichtlichkeit zu wahren.

Während der Entwicklung traten jedoch auch einige Schwierigkeiten auf, da einige Aspekte, welche hier gefordert waren, so kein Inhalt der Vorlesungsveranstaltungen waren. Dazu zählten allgemein die Versionsverwaltung, welche die Zusammenarbeit zwar sehr vereinfachte, von mir persönlich vorher noch nie genutzt worden war, sowie die Anforderung der Notiz-Activity zum Umgang mit Bildern. Letztere brachten eine Speichernutzungs- und Ladezeitproblematik mit sich, weshalb Herr Beilfuß den Umgang mir den Bildern im Back-End letztlich übernommen hat. Die Zusammenarbeit hat letztlich ermöglicht, dass eine sehr umfangreiche und performante Ansicht entstanden ist.

Auch die Arbeit mit dem Rest des Teams habe ich als sehr angenehm empfunden. Der Zusammenhalt war stets spürbar und es wurden viele Verbesserungsvorschläge seitens anderer Teammitglieder eingebracht und im eigenen Aufgabenbereich umgesetzt. Die Zeit wurde zum Ende hin etwas knapp, jedoch konnten wir das Projekt erfolgreich fertigstellen.

Ich persönlich habe aus dem Projekt viele Erfahrungen mit Hinblick auf die Programmierung in Java und für Android, jedoch auch im Projektmanagement und der Zusammenarbeit mitgenommen. Durch die Selbstständigkeit der Arbeit wurde meine Problemlösefähigkeit stets gefordert und ich habe, auch in Zusammenarbeit mit dem Rest des Teams, stets zu einer Lösung finden können.

Als Ergebnis finde ich unsere App sehr gelungen und meine Erwartungen wurden sowohl in Sachen Design, als auch von der Usability und dem Umfang der Funktionen her übertroffen. Aus den genannten Gründen würde ich das Projekt als erfolgreich abgeschlossen bezeichnen.

6.5 Fazit von Robin Menzel

Zu Beginn der Gruppenarbeit war meine Freude über die Aufgabenstellung groß. Einen TEN-Manager kann ich gut gebrauchen - ein Kalorientagebuch, wie die Jahrgänge zuvor nicht. Bei der Größe der Gruppe hingegen war ich sehr gespannt, was das mit 8 Personen wird. Außerdem wollte ich mich schon immer mal mit der Android App Entwicklung auseinander setzen. Aber direkt mit so einem großen Projekt? Was kann ich, fast ohne Vorkenntnisse beitragen?

Unsere Gruppe, bzw. Teile davon haben sich bereits in anderen Projektarbeiten als gute Teams bewiesen. Alle kannten sich gut und konnten sich gegenseitig vom Kenntnisstand und von der Arbeitsmoral her gut einschätzen. So war auch eine Aufgabenverteilung und eine Unterteilung in Sub-Teams sehr einfach umzusetzen. Und so konnte die Arbeit effektiv auf die Gruppe verteilt werden.

Der Start viel uns etwas schwer. Da es bei der Entwicklung nicht nur um das Coding geht, sondern zumindest mal feststehen muss, was überhaupt gecode werden muss, wussten wir nicht, wie wird das am besten Aufteilen. Gerade bei Abhängigkeiten zwischen Arbeitspaketen (Mock-Up erstellen & Aufteilung der Activities) haben wir mehr Zeit benötigt, als notwendig. Dies hat sich in der Implementierung wiederholt, da die Abhängigkeit zwischen den Daten und den unterschiedlichen Activities hier sogar etwas höher ist. Durch guten und offenen Austausch und etwas Flexibilität der Teammitglieder stellte dies aber keine große Herausforderung da.

Innerhalb der Gruppe fand ich den Wissensaustausch und die daraus entstehenden Synergieeffekte bemerkenswert. So war es uns über alle Activities hinweg möglich, mit jedem

Wissensstand seinen Teil beizutragen. Themen wie das Sharing von TENs oder die Toolbar mussten nicht von jedem Sub-Team neu erfunden werden, sondern konnten direkt von Anfang an übernommen werden, wodurch mehr Zeit blieb weitere Herausforderungen zu meistern oder kleine extra Features einzubauen.

Für mich persönlich habe ich unglaublich viel über die App-Entwicklung gelernt und viel Spaß beim Finden von Lösungen gehabt. Meine Lernkurve war konstant stark ansteigend und die Erfahrung am Ende auch die Mühe wert.

Zusammenfassend ist mein Fazit sehr positiv. Mit dem Projektverlauf bin ich sehr zufrieden, wenn man bedenkt das von uns noch niemand ein so großes Software Engineering Projekt von Anfang bis Ende durchgeführt hat oder überhaupt mal eine App entwickelt hat. Die Applikation selbst hat meine Erwartungen übertroffen. Die Muss-Kriterien aus der Aufgabenstellung wurden alle mit etwas Interpretationsspielraum implementiert und viele kleine, nützlichen Extra Features haben es mit in die Applikation geschafft.

Lediglich bei der Dokumentation habe ich bei vielen Dingen nicht ganz den Sinn durchschaut. Wie eigentlich jeder Entwickler dokumentiere auch ich gerne nur das aller nötigste. Dennoch habe ich im Hinblick auf andere Dokumentation bei Software Engineering Projekten, in dieser Dokumentation nicht überall einen Mehrwert entdeckt.

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

7.1 Quellen zur Entwicklung

Scrollbare Bildergalerie

<https://stackoverflow.com/questions/29429556/android-horizontal-scrolling-image-gallery>

Android Benachrichtigung: Toast

<https://developer.android.com/guide/topics/ui/notifiers/toasts>

Adapter für ein eigenes ListView

<https://www.journaldev.com/10416/android-listview-with-custom-adapter-example-tutorial>

Galerie und Kamera Import

<https://developer.android.com/training/camera/photobasics>

Wisch-Geste erkennen und behandeln

<https://stackoverflow.com/questions/6645537/how-to-detect-the-swipe-left-or-right-in-android>

Android MotionEvent

<https://developer.android.com/reference/android/view/MotionEvent>

Image in AlertDialog

<https://stackoverflow.com/questions/6276501/how-to-put-an-image-in-an-alertdialog-android>

Bildkorrektur

<https://www.samiettamawy.com/how-to-fix-the-camera-intent-rotated-image-in-android/>

Komprimierter Import von Bilddateien:

<https://stackoverflow.com/questions/477572/strange-out-of-memory-issue-while-load>

Couchbase

<https://docs.couchbase.com/couchbase-lite/2.1/java.html>

NoSQL vs SQL

https://dbs.uni-leipzig.de/file/seminar_1112_tran_ausarbeitung.pdf

Jackson

<https://stackoverflow.com/questions/2378402/jackson-vs-gson>

<https://www.baeldung.com/jackson-object-mapper-tutorial>

<https://stackoverflow.com/questions/16019834/ignoring-property-when-deserializing>

Ladespinner

<https://developer.android.com/reference/android/widget/ProgressBar>

<https://stackoverflow.com/questions/11752961/how-to-show-a-progress-spinner-in-an>

AsyncTask

<https://developer.android.com/reference/android/os/AsyncTask>

<https://stackoverflow.com/questions/9671546/asynctask-android-example>

Unwandlung dp in px für changeConfiguration des ImageOverlays

<https://stackoverflow.com/questions/4605527/converting-pixels-to-dp>

FileRepository

<https://stackoverflow.com/questions/4181774/show-image-view-from-file-path>

<https://stackoverflow.com/questions/14053338/save-bitmap-in-android-as-jpeg-in-ex>

<https://developer.android.com/reference/java/io/File>

Bilderimport aus der Gallerie

<https://stackoverflow.com/questions/20324155/get-filename-and-filename-of-selected>

7.2 Quellen zur Entwicklung der Benutzeroberfläche

Material Design

<https://material.io/design/color/the-color-system.html>

GUI

<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

<http://www.vogella.com/tutorials/ActionBar/article.html>

7.3 Bilder in den Mockdaten

https://commons.wikimedia.org/wiki/File:Cute_koala.jpg
https://upload.wikimedia.org/wikipedia/commons/4/49/Koala_climbing_tree.jpg
<https://upload.wikimedia.org/wikipedia/commons/3/37/Flying-kangaroo.jpg>
https://upload.wikimedia.org/wikipedia/commons/thumb/0/03/K%C3%A4nguru_am_Strand_%2830719542583%29.jpg/1280px-K%C3%A4nguru_am_Strand_%2830719542583%29.jpg
https://upload.wikimedia.org/wikipedia/commons/thumb/e/e2/Kolm%C3%A5rden_delfin.JPG/1280px-Kolm%C3%A5rden_delfin.JPG
https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/Dolphins_Oman-2.jpg/1280px-Dolphins_Oman-2.jpg
https://upload.wikimedia.org/wikipedia/commons/thumb/8/8b/Bamboo_Richelieu.jpg/1280px-Bamboo_Richelieu.jpg
https://en.wikipedia.org/wiki/Princes_Bridge#/media/File:Eureka_Tower_and_Yarra_River_-_Melbourne.jpg
https://upload.wikimedia.org/wikipedia/commons/thumb/c/c5/Melbourne_skyline_from_St._Kilda.jpg/1280px-Melbourne_skyline_from_St._Kilda.jpg

7.4 Quellen zu Latex

<https://en.wikibooks.org/wiki/LaTeX>
<https://www.namsu.de/Extra/befehle>
<https://www.latex-project.org/>
http://latex.hpfsc.de/content/latex_tipps_und_tricks/links_urls_pdf/

8 Anhang - Quelltexte

Listing 1: Titel (Vorname Nachname)

8.1 Activities

8.1.1 Event

8.1.1.1 Data

8.1.1.2 GUI

8.1.1.3 Logic

Listener

8.1.1.4 Reminder

8.1.2 Note

8.1.2.1 Note

Data

Gui

Logic

8.1.2.2 Notetags

8.1.3 Todo

8.2 Data

8.2.1 Models

Listing 2: Todo (Joscha Nassenstein)

```

@JsonIgnoreProperties(ignoreUnknown = true)
public class Todo extends TEN {
    private double progress;
    private String note;
    private Date startDate;
    private Date endDate;
    private ArrayList<Task> tasks;
    //Constructors
    public Todo(){
        super();
        this.note = "";
        this.startDate = new Date();
        this.endDate = new Date();
        this.tasks = new ArrayList<>();
        tasks.add(new Task());
    }
    public Todo(String title){
        super(title);
        this.startDate = new Date();
        this.endDate = new Date();
        this.tasks = new ArrayList<>();
        tasks.add(new Task());
    }
    public Todo(String title, String note){
        super(title);
        this.note = note;
        this.startDate = new Date();
        this.endDate = new Date();
        this.tasks = new ArrayList<>();
        tasks.add(new Task());
    }
    public Todo(String title, String note, ArrayList<Task> tasks){
        super(title);
        this.note = note;
        this.tasks = tasks;
        this.progress = calculateProgress();
        this.startDate = new Date();
        this.endDate = new Date();
    }
    public Todo(String title, String note, ArrayList<Task> tasks, Date
        ↗ endDate){
        super(title);
        this.note = note;
        this.tasks = tasks;
        this.progress = calculateProgress();
        this.startDate = new Date();
        this.endDate = endDate;
    }
    public Todo(String title, String note, ArrayList<Task> tasks, Date
        ↗ startDate, Date endDate){
        super(title);
        this.note = note;
        this.tasks = tasks;
    }
}

```

Listing 3: Event (Joscha Nassenstein)

```

@JsonIgnoreProperties(ignoreUnknown = true)
public class Event extends TEN {
    private Date time;
    private ArrayList<Date> reminder;
    private String address;
    private RecurringType recurringType;
    //Constructor
    //empty default
    public Event() {
        super();
        this.time = new Date();
        this.reminder = new ArrayList<>();
        this.address = "";
        this.recurringType = RecurringType.NONE;
    }
    //simple for usage
    public Event(String title, Date time, ArrayList<Date> reminder, String
        ↪ address) {
        super(title);
        this.time = time;
        this.reminder = reminder;
        this.address = address;
        this.recurringType = RecurringType.NONE;
    }
    //all Attributes for reconstruction of complete Object
    public Event(String title, String ID, int color, int accentColor, Date
        ↪ dateOfCreation, Date time, ArrayList<Date> reminder, String
        ↪ address, RecurringType recurringType) {
        super(title, ID, color, accentColor, dateOfCreation);
        this.time = time;
        this.reminder = reminder;
        this.address = address;
        this.recurringType = recurringType;
    }
    //Getter and Setter
    public Date getTime() {
        return time;
    }
    public void setTime(Date time) {
        this.time = time;
    }
    public ArrayList<Date> getReminder() {
        return reminder;
    }
    public void setReminder(ArrayList<Date> reminder) {
        this.reminder = reminder;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public RecurringType getRecurringType() {
        return recurringType;
    }
    public void setRecurringType(RecurringType recurringType) {
        this.recurringType = recurringType;
    }
    public Bundle getBundle() {
        Bundle bundle = super.getBundle();
        bundle.putLong(BundleKeys.KEY_EVENT_TIME, time.getTime());
    }
}

```

Listing 4: Note (Joscha Nassenstein)

```

@JsonIgnoreProperties(ignoreUnknown = true)
public class Note extends TEN {
    private String description;
    private ArrayList<String> tags;
    private ArrayList<Image> pictures;
    public int imageIDCounter;
    //Constructors
    public Note() {
        super();
        this.description = "";
        this.tags = new ArrayList<>();
        this.pictures = new ArrayList<>();
        this.imageIDCounter = 0;
    }
    public Note(String title, String description) {
        super(title);
        this.description = description;
        this.tags = new ArrayList<String>();
        this.pictures = new ArrayList<Image>();
    }
    //all Attributes for complete Reconstruction
    public Note(String title, String ID, int color, int accentColor, Date
        ↪ dateOfCreation, String description, ArrayList<String> tags,
        ↪ ArrayList<Image> pictures, int imageIDCounter) {
        super(title, ID, color, accentColor, dateOfCreation);
        this.description = description;
        this.tags = tags;
        this.pictures = pictures;
        this.imageIDCounter = imageIDCounter;
    }
    //Getter and Setter
    public String getDescription() {
        return description;
    }
    public void setDescription(String description) {
        this.description = description;
    }
    public ArrayList<String> getTags() {
        return tags;
    }
    public void setTags(ArrayList<String> tags) {
        this.tags = tags;
    }
    public Image addImage(Bitmap bitmap) {
        this.imageIDCounter++;
        String imageID = this.getId() + FileSystemConstants.IMAGE_CORE_ID +
            ↪ this.imageIDCounter;
        Image image = new Image(imageID, bitmap);
        Log.i("cool", image.getId());
        this.pictures.add(image);
        return image;
    }
    public void addImage(Image pImage) {
        Log.i("NoteRemake", "Pictures Size: " + pictures.size());
        for (int i = 0; i < pictures.size(); i++) {
            if (pImage.getId().equals(pictures.get(i).getId())) {
                pictures.set(i, pImage);
            }
        }
    }
    public ArrayList<Image> getPictures() {
        return pictures;
    }
}

```

Listing 5: TEN (Joscha Nassenstein)

```

public class TEN {
    private String title;
    @JsonIgnore
    private String ID;
    @ColorInt
    @JsonIgnore
    private int color;
    @ColorInt
    @JsonIgnore
    private int accentColor;
    @JsonIgnore
    private Date dateOfCreation;
    //Constructor
    //empty default
    public TEN() {
        this.ID = null;
        this.title = "";
        int colorIndex = Colors.getRandomColorIndex();
        this.color = Colors.COLORS[colorIndex];
        this.accentColor = Colors.DARKER_ACCENT_COLORS[colorIndex];
        this.dateOfCreation = new Date();
    }
    //simple for usage
    public TEN(String title) {
        this.ID = null;
        this.title = title;
        int colorIndex = Colors.getRandomColorIndex();
        this.color = Colors.COLORS[colorIndex];
        this.accentColor = Colors.DARKER_ACCENT_COLORS[colorIndex];
        this.dateOfCreation = new Date();
    }
    //complete Object must be reconstructed
    public TEN(String title, String ID, int color, int accentColor, Date
        ↪ dateOfCreation) {
        this.ID = ID;
        this.title = title;
        this.color = color;
        this.accentColor = accentColor;
        this.dateOfCreation = dateOfCreation;
    }
    public boolean isFound(String pSearchString){
        return title!=null?title.toLowerCase().contains(pSearchString.
            ↪ toLowerCase()):false;
    }
    public Bundle getBundle() {
        Bundle bundle = new Bundle();
        bundle.putString(BundleKeys.KEY_TEN_ID, ID);
        bundle.putString(BundleKeys.KEY_TEN_TITLE, title);
        bundle.putInt(BundleKeys.KEY_TEN_COLOR, color);
        bundle.putInt(BundleKeys.KEY_TEN_ACCENT_COLOR, accentColor);
        bundle.putLong(BundleKeys.KEY_TEN_DATE_OF_CREATION, this.
            ↪ dateOfCreation.getTime());
        return bundle;
    }
    //Getter and Setter
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getID() {
        return ID;
    }
}

```

8.2.1.1 TENs

Listing 6: BundleKeys (Jan Beilfuß)

```
public class BundleKeys {  
    //TEN Keys  
    public static final String KEY_TEN_ID = "TenID";  
    public static final String KEY_TEN_TITLE = "TenTitle";  
    public static final String KEY_TEN_COLOR = "TenColor";  
    public static final String KEY_TEN_ACCENT_COLOR = "TenAccentColor";  
    public static final String KEY_TEN_DATE_OF_CREATION = "  
        ↪ TenDateOfCreation";  
    //Event Keys  
    public static final String KEY_EVENT_TIME = "EventTime";  
    public static final String KEY_EVENT_ADDRESS = "EventAdress";  
    //Note Keys  
    public static final String KEY_NOTE_DESCRIPTION = "NoteDescription";  
    public static final String KEY_NOTE_TAGS = "NoteTags";  
    public static final String KEY_NOTE_PICTURES = "NotePictures";  
    //To Do Keys  
    public static final String KEY_TODO_NOTE = "TodoNote";  
    public static final String KEY_TODO_STATUS = "TodoStatus";  
    public static final String KEY_TODO_DESCRIPTION = "TodoDescription";  
}
```

Listing 7: Colors (Jan Beilfuß)

```
public class Colors {  
    //700er Material Colors  
    public static final int[] COLORS = {  
        0xFF0288D1, //light blue  
        0xFFEF5350, //red  
        0xFFAFB42B, //lime  
        0xFF388E3C, //green  
        0xFFFFB8C00, //Orange  
        0xFF7E57C2, //Deep Purple  
        0xFFEC407A, //Pink  
        0xFF009688, //Teal  
    };  
    //900er Material Colors  
    public static final int[] DARKER_ACCENT_COLORS = {  
        0xFF01579B, //light blue  
        0xFFB71C1C, //red  
        0xFF827717, //lime  
        0xFF1B5E20, //green  
        0xFFEF6C00, //orange  
        0xFF512DA8, //Deep Purple  
        0xFFC2185B, //Pink  
        0xFF00796B, //Teal  
    };  
    public static int getRandomColorIndex() {  
        int index = Colors.COLORS.length;  
        while (index == Colors.COLORS.length) {  
            index = (int) (Math.random() * Colors.COLORS.length);  
        }  
        return index;  
    }  
}
```

Listing 8: Image (Jan Beilfuß)

```

public class Image {
    private String id;
    @JsonIgnore
    private Bitmap bitmap;
    //Constructor
    public Image() {
        this.id = "";
        this.bitmap = null;
    }
    public Image(String id) {
        this.id = id;
        this.bitmap = null;
    }
    public Image(String id, Bitmap bitmap) {
        this.id = id;
        this.bitmap = bitmap;
    }
    public Image (Image image){
        this.id = "" + image.getId();
        if(image.getBitmap()!=null){
            this.bitmap = Bitmap.createBitmap(image.getBitmap());
        }
    }
    //Getter and Setter
    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public Bitmap getBitmap() {
        return bitmap;
    }
    public void setBitmap(Bitmap bitmap) {
        this.bitmap = bitmap;
    }
}

```

Listing 9: MockData (Jan Beilfuß)**Listing 10:** RecurringType (Joscha Nassenstein)

```

public enum RecurringType {
    NONE,DAILY,WEEKLY,MONTHLY,YEARLY
}

```

Listing 11: Tasks (Joscha Nassenstein)

```
public class Task {  
    private String description;  
    private boolean status;  
    //Constructors  
    public Task(){  
        this.description = "";  
        this.status = false;  
    }  
    public Task(String description){  
        this.description = description;  
        this.status = false;  
    }  
    public Task(String description, boolean status){  
        this.description = description;  
        this.status = status;  
    }  
    //Getters and Setters  
    public String getDescription(){return description;}  
    public void setDescription(String description) {this.description =  
        ↪ description;}  
  
    public boolean getStatus(){return status;}  
    public void setStatus(boolean status){this.status = status;}  
}
```

8.2.1.2 Utils

8.2.2 Repository

Listing 12: DatabaseRepository (Jan Beilfuß)

```

/Class that manages All Requests to the Database
//Author: Jan Beilfuss
public class DatabaseRepository {
    ReadRepository mReadRepository;
    WriteRepository mWriteRepository;
    public DatabaseRepository() {
        this.mReadRepository = new ReadRepository();
        this.mWriteRepository = new WriteRepository();
    }
    public Todo getTodoByID(String pId) {
        return mReadRepository.getTodoByID(pId);
    }
    public Event getEventByID(String pId) {
        return mReadRepository.getEventByID(pId);
    }
    public Note getNoteByID(String pId) {
        return mReadRepository.getNoteByID(pId);
    }
    //for new TEN-Objects
    public void insertTEN(TEN pTen) { mWriteRepository.insertTEN(pTen); }
    //for already saved TEN-Objects
    public void updateTEN(TEN pTen) {
        mWriteRepository.updateTEN(pTen);
    }
    public ArrayList<TEN> getAllTENs() {
        return mReadRepository.getAllTENs();
    }
    public boolean deleteTEN(String pTenId) {
        return mWriteRepository.deleteTEN(pTenId);
    }
    //returns main and accent color for given TEN-Object-ID
    public int[] getTENColors(String pTenId) {
        return mReadRepository.getTENColors(pTenId);
    }
}

```

Listing 13: DataContextManager (Jan Beilfuß)

```

//Class that provides the database to the DatabaseRepository-Classes and
    ↪ Functions
//Author: Jan Beilfuss
public class DataContextManager {
    public static Database database = null;
    public static Context context = null;
    //Gets Database from Context if it is not set
    public static void initDatabase(Context pContext) {
        if (DataContextManager.getDatabase() == null) {
            DataContextManager.context = null;
            DataContextManager.context = pContext;
            try {
                DatabaseConfiguration config = new DatabaseConfiguration(
                    ↪ pContext.getApplicationContext());
                DataContextManager.database = new Database(
                    ↪ RepositoryConstants.DATABASENAME, config);
                compactDatabase();
            } catch (CouchbaseLiteException e) {
                Toast.makeText(pContext, "Fehler bei der
                    ↪ Datenbankerstellung", Toast.LENGTH_LONG);
            }
        }
    }
    //Compacts the Database = Cleans all Empty Entries (they are just
        ↪ flagged on Delete)
    public static void compactDatabase() {
        try {
            DataContextManager.getDatabase().compact();
        } catch (CouchbaseLiteException e) {
        }
    }
    //returns the number of Documents in the Database. ONLY needed when
        ↪ using Mockdata
    public static int getNumberOfDocuments() {
        Query query = QueryBuilder.select(SelectResult.expression(Meta.id))
            ↪ .from(DataSource.database(DataContextManager.getDatabase()))
            ↪ ;
        try {
            ResultSet rs = query.execute();
            return rs.allResults().size();
        } catch (CouchbaseLiteException e) {
            return -1;
        }
    }
    public static Database getDatabase() {
        return DataContextManager.database;
    }
}

```

Listing 14: RepositoryConstants (Jan Beilfuß)

```
//Author: Jan Beilfuss
public class RepositoryConstants {
    //database name
    public static final String DATABASENAME = "TENDB";
    //Keys for Key-Value-Pairs in Document
    public static final String OBJECT_KEY = "ObjectKey";
    public static final String TYPE_KEY = "TypeKey";
    public static final String CREATION_DATE_KEY = "CreationDateKey";
    public static final String COLOR_KEY = "ColorKey";
    public static final String ACCENT_COLOR_KEY = "AccentColorKey";
    public static final String COUCHBASE_ID_KEY = "id";
    //Type corresponding to TEN-Classes
    public static final String EVENT_TYPE = "Event";
    public static final String NOTE_TYPE = "Note";
    public static final String TODO_TYPE = "Todo";
}
```

Listing 15: QueriedTenConverter (Jan Beilfuß)

```

//class that manages the Conversion from Query Result to TEN Java Object
//Author: Jan Beilfuss
public class QueriedTenConverter {

    private TenJsonParser mTenJsonParser;
    private SeparateAttributesConverter mSeparateAttributesConverter;

    public QueriedTenConverter() {
        this.mTenJsonParser = new TenJsonParser();
        this.mSeparateAttributesConverter = new
            ↪ SeparateAttributesConverter();
    }

//Method that maps the Dictionary of a Query Result To a TEN-Object
//Object only contains Information from the JSON, but none of the ones
    ↪ stored in the document
public TEN createTENfromResult(Result pResult) {

    Dictionary dictionary = pResult.getDictionary(RepositoryConstants.
        ↪ DATABASENAME);
    String type = dictionary.getString(RepositoryConstants.TYPE_KEY);
    String objectJSON = dictionary.getString(RepositoryConstants.
        ↪ OBJECT_KEY);

    TEN resultTEN = null;
    switch (type) {
        case RepositoryConstants.EVENT_TYPE:
            resultTEN = mTenJsonParser.stringToEvent(objectJSON);
            break;
        case RepositoryConstants.NOTE_TYPE:
            resultTEN = mTenJsonParser.stringToNote(objectJSON);
            break;
        case RepositoryConstants.TODO_TYPE:
            resultTEN = mTenJsonParser.stringToTodo(objectJSON);
            break;
        default:
    }
    if (resultTEN != null) {
        resultTEN = this.mSeparateAttributesConverter.
            ↪ addTENPropertiesFromResult(resultTEN, pResult);
    }
    return resultTEN;
}
}

```

Listing 16: SeparateAttributesConverter (Jan Beilfuß)

```

//Methods that adds all Attributes not stored in the JSON
//Author: Jan Beilfuss
public class SeparateAttributesConverter {

    //Document is the Databasereturn when getting only a single TEN
    public TEN addTENPropertiesFromDocument(TEN pTen, Document pDocument)
        {
            pTen.setID(pDocument.getId());
            pTen.setColor(pDocument.getInt(RepositoryConstants.COLOR_KEY));
            pTen.setAccentColor(pDocument.getInt(RepositoryConstants.
                ACCENT_COLOR_KEY));
            pTen.setDateOfCreation(pDocument.getDate(RepositoryConstants.
                CREATION_DATE_KEY));
            return pTen;
        }

    //Result is the Databasereturn when querying a set of TENs
    public TEN addTENPropertiesFromResult(TEN pTen, Result pResult) {
        Dictionary dictionary = pResult.getDictionary(RepositoryConstants.
            DATABASENAME);
        pTen.setID(pResult.getString(RepositoryConstants.COUCHBASE_ID_KEY))
            ;
        pTen.setColor(dictionary.getInt(RepositoryConstants.COLOR_KEY));
        pTen.setAccentColor(dictionary.getInt(RepositoryConstants.
            ACCENT_COLOR_KEY));
        pTen.setDateOfCreation(dictionary.getDate(RepositoryConstants.
            CREATION_DATE_KEY));

        return pTen;
    }
}

```

Listing 17: TenJsonParser (Jan Beilfuß)

```
//Class, that parses the JSON into a TEN Object
//Author: Jan Beilfuss
public class TenJsonParser {

    private ObjectMapper mObjectMapper;

    public TenJsonParser() {
        this.mObjectMapper = new ObjectMapper();
    }

    //parses To-do
    public Todo stringToTodo(String pJson) {
        try {
            Todo todo = this.mObjectMapper.readValue(pJson, Todo.class);
            return todo;
        } catch (IOException e) {
            return null;
        }
    }

    //parses Event
    public Event stringToEvent(String pJson) {
        try {
            Event event = this.mObjectMapper.readValue(pJson, Event.class);
            return event;
        } catch (IOException e) {
            return null;
        }
    }

    //parses Note
    public Note stringToNote(String pJson) {
        try {
            Note note = this.mObjectMapper.readValue(pJson, Note.class);
            return note;
        } catch (IOException e) {
            return null;
        }
    }
}
```

8.2.2.1 Converter

Listing 18: ImageDeleteer (Jan Beilfuß)

```

public class ImageDeleteer {

    FileRepository mFileRepository;

    //Class that creates async Task to delete Images from the Filesystem
    //Author: Jan Beilfuss
    public ImageDeleteer(FileRepository pFileRepository) {
        this.mFileRepository = pFileRepository;
    }

    public void execute(Image image) {
        DeleteImageTask deleteImageTask = new DeleteImageTask();
        deleteImageTask.execute(image);
    }

    private class DeleteImageTask extends AsyncTask<Image, Integer, Void>
    {
        //Deletes Image in preview and original folder
        @Override
        protected Void doInBackground(Image... images) {
            String[] folders = {FileSystemConstants.IMAGE_ORIGINAL_FOLDER,
                FileSystemConstants.IMAGE_PREVIEW_FOLDER};

            for (String folder : folders) {
                String directoryPath = mFileRepository.getContext().
                    ↪ getExternalFilesDir(Environment.DIRECTORY_PICTURES).
                    ↪ getAbsolutePath() + "/" + folder + "/";
                String filePath = directoryPath + images[0].getId() + ".jpg
                    ↪ ";

                File file = new File(filePath);
                if (file.exists()) {
                    file.delete();
                }
            }
            return null;
        }
    }
}

```

Listing 19: ImageSaver (Jan Beilfuß)

```

public class ImageSaver {

    FileRepository mFileRepository;

    //Class that creates async Task to save Images to the filesystem
    //Author: Jan Beilfuss
    public ImageSaver(FileRepository pFileRepository) {
        this.mFileRepository = pFileRepository;
    }

    public void execute(Image image) throws IOException {
        Image copy = new Image(image);
        image.setBitmap(null);
        SaveImageTask saveImageTask = new SaveImageTask();
        saveImageTask.execute(copy);
    }

    private class SaveImageTask extends AsyncTask<Image, Integer, Void> {

        //saves Image to preview and original folder
        @Override
        protected Void doInBackground(Image... images) {
            PreviewImageCreator previewImageCreator = new
                ↪ PreviewImageCreator();
            Context context = DataContextManager.context;
            String[] folders = {FileSystemConstants.IMAGE_ORIGINAL_FOLDER,
                ↪ FileSystemConstants.IMAGE_PREVIEW_FOLDER};

            for (String folder : folders) {
                String folderPath = Environment.DIRECTORY_PICTURES + "/" +
                    ↪ folder;
                String imageName = images[0].getId() + FileSystemConstants.
                    ↪ JPEG;

                File storageDir = context.getExternalFilesDir(folderPath);
                FileOutputStream fos = null;
                Bitmap bitmap = images[0].getBitmap();
                File image = new File(storageDir, imageName);

                if (!image.exists()) {
                    if (bitmap != null) {
                        if (folder.equals(FileSystemConstants.
                            ↪ IMAGE_PREVIEW_FOLDER)) {
                            bitmap = previewImageCreator.getPreviewImage(
                                ↪ bitmap);
                        }
                        try {
                            fos = new FileOutputStream(image);
                        } catch (FileNotFoundException e) {
                        }

                        bitmap.compress(Bitmap.CompressFormat.JPEG,
                            ↪ FileSystemConstants.COMPRESSION_FACTOR, fos);
                        try {
                            fos.flush();
                            fos.close();
                        } catch (IOException e) {
                        }
                    }
                }
                images[0].setBitmap(null);
            }
        }
    }
}

```

Listing 20: FileRepositroy (Jan Beilfuß)

```

//Class that handles everything regarding persistent Images
//Author: Jan Beilfuss
public class FileRepository {
    public Context getContext() {
        return DataContextManager.context;
    }

    //creates ImageSaver to save Images to the Filesystem
    public void saveImagePersistent(Image pImage) throws IOException {
        ImageSaver imageSaver = new ImageSaver(this);
        imageSaver.execute(pImage);
    }

    //reads and decodes Image from the Filesystem
    public Image readImageFromDirectory(Image image, String directory) {
        Context context = DataContextManager.context;
        String directoryPath = context.getExternalFilesDir(Environment.
            ↪ DIRECTORY_PICTURES).getAbsolutePath() + "/" + directory + "/";
        String filePath = directoryPath + image.getId() +
            ↪ FileSystemConstants.JPG;
        Bitmap bitmap = BitmapFactory.decodeFile(filePath);
        Image result = new Image(image.getId(), bitmap);
        return result;
    }

    //deletes single file at given path
    public void deleteImageFromDirectories(String path) {
        File file = new File(path);
        if(file.exists()) file.delete();
    }

    //creates ImageDeleteer to delete Files from original and preview folder
    public void deleteImageFromDirectories(Image image) {
        ImageDeleteer imageDeleteer = new ImageDeleteer(this);
        imageDeleteer.execute(image);
    }
}

```

Listing 21: FileSystemConstants (Jan Beilfuß)

```

public class FileSystemConstants {
    public static final String JPEG = ".jpg";
    public static final int COMPRESSION_FACTOR = 75;

    public static final String IMAGE_CORE_ID = "imageID";

    public static final String IMAGE_ORIGINAL_FOLDER = "original";
    public static final String IMAGE_PREVIEW_FOLDER = "preview";
}

```

8.2.2.2 Filesystem

Listing 22: GetAllTensQuery (Jan Beilfuß)

```
//Class that contains our currently only Database-Query
//Author: Jan Beilfuss
public class GetAllTensQuery {

    private QueriedTenConverter mQueriedTenConverter;

    public GetAllTensQuery() { this.mQueriedTenConverter = new
        ↪ QueriedTenConverter(); }

    //Creates Query for all Saved TENS and returns converted java objects
    public ArrayList<TEN> getAllTENS() {
        Query query = QueryBuilder.select(
            SelectResult.all(),
            SelectResult.expression(Meta.id))
            .from(DataSource.database(DataContextManager.getDatabase()))
            ↪ )
            .orderBy(Ordering.property(RepositoryConstants.
                ↪ CREATION_DATE_KEY).descending());
        ArrayList<TEN> resultList = new ArrayList<TEN>();

        try {
            ResultSet rs = query.execute();

            List<Result> allResults = rs.allResults();
            for (Result result : allResults) {
                TEN tenObject = this.mQueriedTenConverter.
                    ↪ createTENFromResult(result);
                resultList.add(tenObject);
            }
            return resultList;
        } catch (CouchbaseLiteException e) {
            return null;
        }
    }
}
```

Listing 23: ReadRepository (Jan Beilfuß)

```

//Class, that manages all reading Requests onto the Database
//Author: Jan Beilfuss
public class ReadRepository {
    TenJsonParser mTenJsonParser;
    SeparateAttributesConverter mSeparateAttributesConverter;
    GetAllTensQuery mGetAllTensQuery;

    public ReadRepository () {
        this.mTenJsonParser = new TenJsonParser();
        this.mSeparateAttributesConverter = new
            ↪ SeparateAttributesConverter();
        this.mGetAllTensQuery = new GetAllTensQuery();
    }

    //returns To-Do with given ID
    public Todo getTodoByID(String pId) {
        Document todoDocument = DataContextManager.getDatabase().
            ↪ getDocument(pId);
        String json = todoDocument.getString(RepositoryConstants.
            ↪ OBJECT_KEY);
        Todo finalTodo = this.mTenJsonParser.stringToTodo(json);
        finalTodo = (Todo) this.mSeparateAttributesConverter.
            ↪ addTENPropertiesFromDocument(finalTodo, todoDocument);
        return finalTodo;
    }

    //returns Event with given ID
    public Event getEventByID(String pId) {
        Document eventDocument = DataContextManager.getDatabase().
            ↪ getDocument(pId);
        String json = eventDocument.getString(RepositoryConstants.
            ↪ OBJECT_KEY);
        Event finalEvent = this.mTenJsonParser.stringToEvent(json);
        finalEvent = (Event) this.mSeparateAttributesConverter.
            ↪ addTENPropertiesFromDocument(finalEvent, eventDocument);
        return finalEvent;
    }

    //returns Note with given ID
    public Note getNoteByID(String pId) {
        Document noteDocument = DataContextManager.getDatabase().
            ↪ getDocument(pId);
        String json = noteDocument.getString(RepositoryConstants.
            ↪ OBJECT_KEY);
        Note finalNote = this.mTenJsonParser.stringToNote(json);
        finalNote = (Note) this.mSeparateAttributesConverter.
            ↪ addTENPropertiesFromDocument(finalNote, noteDocument);
        return finalNote;
    }

    //returns all saved TEN-Objects
    public ArrayList<TEN> getAllTENS() {
        return mGetAllTensQuery.getAllTENS();
    }

    //Needed when TEN is loaded asynchronously for setting the Color of
        ↪ the Activity
    public int[] getTENColors(String pTenId) {
        Document document = DataContextManager.getDatabase().getDocument(
            ↪ pTenId);
        int[] colors = new int[2];
        colors[0] = document.getInt(RepositoryConstants.COLOR_KEY);
    }

```

Listing 24: DocumentSaver (Jan Beilfuß)

```

//class that manages the persistent saving process of Tens
//Author: Jan Beilfuss
public class DocumentSaver {

    ObjectMapper mObjectMapper;
    FileRepository mFileRepository;

    public DocumentSaver() {
        this.mObjectMapper = new ObjectMapper();
        this.mFileRepository = new FileRepository();
    }

//manages process of writing a ten to a document and saving this
public boolean updateCompleteDocument(TEN pTen, MutableDocument
    ↪ pMutableTENDocument) {
    pMutableTENDocument = setTypeOfDocument(pMutableTENDocument, pTen);
    ↪
    pMutableTENDocument = saveSeparateAttributes(pMutableTENDocument,
        ↪ pTen);
    try {
        pMutableTENDocument.setString(RepositoryConstants.OBJECT_KEY,
            ↪ this.mObjectMapper.writeValueAsString(pTen));
        try {
            DataContextManager.getDatabase().save(pMutableTENDocument);
            return true;
        } catch (CouchbaseLiteException e) {
            return false;
        }
    } catch (JsonProcessingException e) {
        return false;
    }
}

//saves attributes not saved in the JSON-String
private MutableDocument saveSeparateAttributes(MutableDocument
    ↪ pMutableDocument, TEN pTen){
    pMutableDocument.setDate(RepositoryConstants.CREATION_DATE_KEY,
        ↪ pTen.getDateOfCreation());
    pMutableDocument.setInt(RepositoryConstants.COLOR_KEY, pTen.
        ↪ getColor());
    pMutableDocument.setInt(RepositoryConstants.ACCENT_COLOR_KEY, pTen.
        ↪ getAccentColor());

    return pMutableDocument;
}

//Sets Type depending of the class of the given Object
private MutableDocument setTypeOfDocument(MutableDocument
    ↪ pMutableDocument, TEN pTen) {

    if (pTen.getClass().getName().contains("Event")) {
        pMutableDocument.setString(RepositoryConstants.TYPE_KEY,
            ↪ RepositoryConstants.EVENT_TYPE);

    } else if (pTen.getClass().getName().contains("Note")) {
        pMutableDocument.setString(RepositoryConstants.TYPE_KEY,
            ↪ RepositoryConstants.NOTE_TYPE);

    } else if (pTen.getClass().getName().contains("Todo")) {
        pMutableDocument.setString(RepositoryConstants.TYPE_KEY,
            ↪ RepositoryConstants.TODO_TYPE);
    }
}

```

Listing 25: WriteRepository (Jan Beilfuß)

```

//Class that handles all writing accesses to the database
//Author: Jan Beilfuss
public class WriteRepository {

    private DocumentSaver mDocumentSaver;
    private TenJsonParser mTenJsonParser;
    private FileRepository mFileRepository;

    public WriteRepository(){
        this.mDocumentSaver = new DocumentSaver();
        this.mTenJsonParser = new TenJsonParser();
        this.mFileRepository = new FileRepository();
    }

    //new TEN to the Database
    public void insertTEN(TEN pTen) {
        MutableDocument mutableTENDocument = new MutableDocument();
        pTen.setID(mutableTENDocument.getId());
        this.mDocumentSaver.updateCompleteDocument(pTen,
            ↪ mutableTENDocument);
    }

    //already Existing TEN to the Database
    public void updateTEN(TEN pTen) {
        MutableDocument mutableTENDocument = DataContextManager.
            ↪ getDatabase().getDocument(pTen.getID()).toMutable();
        this.mDocumentSaver.updateCompleteDocument(pTen,
            ↪ mutableTENDocument);
    }

    public boolean deleteTEN(String pTenId) {
        Document document = DataContextManager.getDatabase().getDocument(
            ↪ pTenId);
        deleteNoteImages(document);
        try {
            if(document != null){
                DataContextManager.getDatabase().delete(document);
                return true;
            }
            return false;
        } catch (CouchbaseLiteException e) {
            return false;
        }
    }

    //Side Effects of Deleting a Note
    public void deleteNoteImages(Document pDocument) {
        if (pDocument.getString(RepositoryConstants.TYPE_KEY).equals(
            ↪ RepositoryConstants.NOTE_TYPE)) {
            String json = pDocument.getString(RepositoryConstants.
                ↪ OBJECT_KEY);
            Note note = mTenJsonParser.stringToNote(json);
            for(Image image: note.getPictures()){
                mFileRepository.deleteImageFromDirectories(image.getId());
            }
        }
    }
}

```

8.2.2.3 Sub-Repositories

8.2.3 Services

Listing 26: Create (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 27: Read (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 28: Update (Ruthild Gilles)

```

public class Update {
    /* Ruthild Gilles
       Class Update contains methods to save information on a changed or
       ↪ newly created TEN.
    */
    /**
     *-----*
     * 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 29: Delete (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);
        }
    }
}

```

Listing 30: ImageService (Ruthild Gilles)

```

public class ImageService {

    public static void saveImage(Image image) {
        try {
            FileRepository fileRepository = new FileRepository();
            fileRepository.saveImagePersistent(image);
        } catch (IOException e) {
            Log.e("ImageService", e.getMessage());
        }
    }

    public static Image getImage(Image image) {
        FileRepository fileRepository = new FileRepository();
        Image result = fileRepository.readImageFromDirectory(image,
            ↪ FileSystemConstants.IMAGE_ORIGINAL_FOLDER);
        return result;
    }

    public static Image getPreviewImage(Image image) {
        FileRepository fileRepository = new FileRepository();
        Image result = fileRepository.readImageFromDirectory(image,
            ↪ FileSystemConstants.IMAGE_PREVIEW_FOLDER);
        return result;
    }

    public static void deleteImage(Image image) {
        FileRepository fileRepository = new FileRepository();
        fileRepository.deleteImageFromDirectories(image);
    }

    public static void deleteImage(String path) {
        FileRepository fileRepository = new FileRepository();
        fileRepository.deleteImageFromDirectories(path);
    }

    public static File createImageFile(Activity pActivity) throws
        ↪ IOException {
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss", Locale.
            ↪ GERMANY).format(new Date());
        String imageFileName = "JPEG_" + timeStamp + "_";
        File storageDir = pActivity.getExternalFilesDir(Environment.
            ↪ DIRECTORY_PICTURES);
        return File.createTempFile(imageFileName, ".jpg", storageDir);
    }
}

```

8.3 Modules

8.3.1 Image Compression

8.3.2 Share

8.4 Overview

8.4.1 Event Fragment

8.4.2 Header

8.4.2.1 Create Fragment

8.4.2.2 Delete Fragment

8.4.2.3 Search Fragment

8.4.3 Image Fragment

8.4.4 Note Fragment

8.4.5 Overview Activity

8.4.5.1 Fragment Manager

8.4.6 Super Classes

8.4.7 Todo Fragments

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