



Schriftliche Ausarbeitung

Erstellung einer Applikation zur Verwaltung von
Todos, Events und Notizen

Erstellt von den **AngryNerds**:

Fabia Schmid

Florian Rath

Jan Beilfuß

Joscha Nassenstein

Robin Menzel

Ruthild Gilles

Sertan Cetin

Yannick Rüttgers

Prüfer:

Prof. Dr. Seifert

Eingereicht am:

07.02.2019

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Listingverzeichnis	V
1 Das Team: Angry Nerds	1
2 Ziele des Projektes (Fabia Schmid)	2
2.1 Unterkapitelüberschrift	2
2.1.1 Unterunterkapitelüberschrift	2
2.2 Kapitel mit Abbildung	2
3 Projektplanung	3
3.1 Beschreibung des Funktionsumfangs (Florian Rath)	3
3.2 Projektablaufplan (Fabia Schmid)	3
3.3 Planung der Software	4
3.3.1 Planung des Mock-Ups (Robin Menzel)	4
3.3.2 Planung der Datenstruktur und Schnittstellen (Ruthild Gilles) .	8
3.3.3 Planung der Activities und Layouts (Florian Rath)	12
3.3.4 Planung der Navigation zwischen den Activities (Yannick Rüttgers)	12
3.4 Geplante Aufgabenverteilung im Team (Fabia Schmid)	12
4 Beschreibung des Projektverlaufs	13
4.1 Tatsächliche Aufgabenverteilung im Team (Fabia Schmid)	13
4.2 Teammeetingprotokolle	13
4.3 Projekttagebücher aller Teammitglieder	13
4.4 Beschreibung von Problemen	13
4.5 Kapitel mit Abbildung	13
5 Dokumentation der Software	15
5.1 Dokumentation der Paketstruktur (Sertan Cetin)	15
5.2 Dokumentation der Activities	15
5.3 Dokumentation der Navigation zwischen Activities (Yannick Rüttgers)	15
5.4 Dokumentation der Activity-übergreifenden, persistenten Datenhaltung (Jan Beilfuß)	15
5.5 Dokumentation der Activity-übergreifenden Klassen (Ruthild Gilles) .	16

5.5.1	Models-Klassen	16
5.5.2	Service-Klassen	16
5.6	Kapitel mit Abbildung	17
6	Fazit der Teammitglieder	19
6.1	Fazit von Fabia Schmid	19
6.2	Fazit von Florian Rath	19
6.3	Fazit von Jan Beilfuß	19
6.4	Fazit von Joscha Nassenstein	19
6.5	Fazit von Robin Menzel	19
6.6	Fazit von Ruthild Gilles	19
6.7	Fazit von Sertan Cetin	19
6.8	Fazit von Yannick Rüttgers	20
7	Quellenverzeichnis	21
7.1	Unterkapitelüberschrift	21
7.1.1	Unterunterkapitelüberschrift	21
8	Anhang - Quelltexte	22
8.1	Activities	22
8.2	Data	22
8.2.1	Service Klassen (Ruthild Gilles)	22
8.3	Overview	24
8.4	Main Activity	24
9	Anhang - Verwendete Tools und Hilfsmittel (Robin Menzel)	25
Ehrenwörtliche Erklärung		

Abbildungsverzeichnis

Abbildung 1: Die Angry Nerds	1
Abbildung 2: Abbildungsbeschriftung	2
Abbildung 3: Mockups - Übersicht über alle TENs	4
Abbildung 4: Mockups - Übersicht über ein vorhandenes und ein neues Event	5
Abbildung 5: Mockups - Übersicht über ein vorhandene, eine leere und eine Bild-Notiz	6
Abbildung 6: Mockups - Übersicht über vorhandene und neue Todos	7
Abbildung 7: Klassendiagramm	9
Abbildung 8: Systemkontextdiagramm	10
Abbildung 9: CRUD-Klassen	11
Abbildung 10: Abbildungsbeschriftung	12
Abbildung 11: Abbildungsbeschriftung	14
Abbildung 12: Abbildungsbeschriftung	18

Listingverzeichnis

Listing 1: Create Klasse (Ruthild Gilles)	22
Listing 2: Read Klasse (Ruthild Gilles)	23
Listing 3: Update Klasse (Ruthild Gilles)	24
Listing 4: Delete Klasse (Ruthild Gilles)	24

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)

2.1 Unterkapitelüberschrift

Unterkapitel

2.1.1 Unterunterkapitelüberschrift

Unterunterkapitel

2.2 Kapitel mit Abbildung

So kann man Abbildungen einfügen:

Abbildung 2: Abbildungsbeschriftung



Quelle: Quelle

3 Projektplanung

3.1 Beschreibung des Funktionsumfangs (Florian Rath)

Hier den Text einfach hin kopieren.

3.2 Projektablaufplan (Fabia Schmid)

Hier den Text einfach hin kopieren.

3.3 Planung der Software

3.3.1 Planung des Mock-Ups (Robin Menzel)

Übersicht

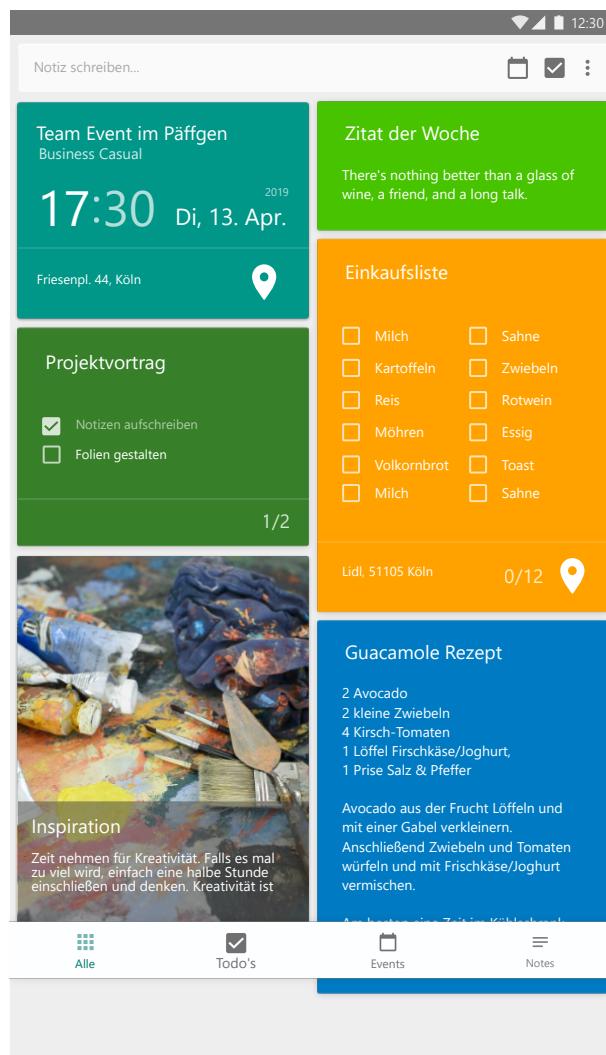


Abbildung 3: Mockups - Übersicht über alle TENs

Event Activity - Unsere Events

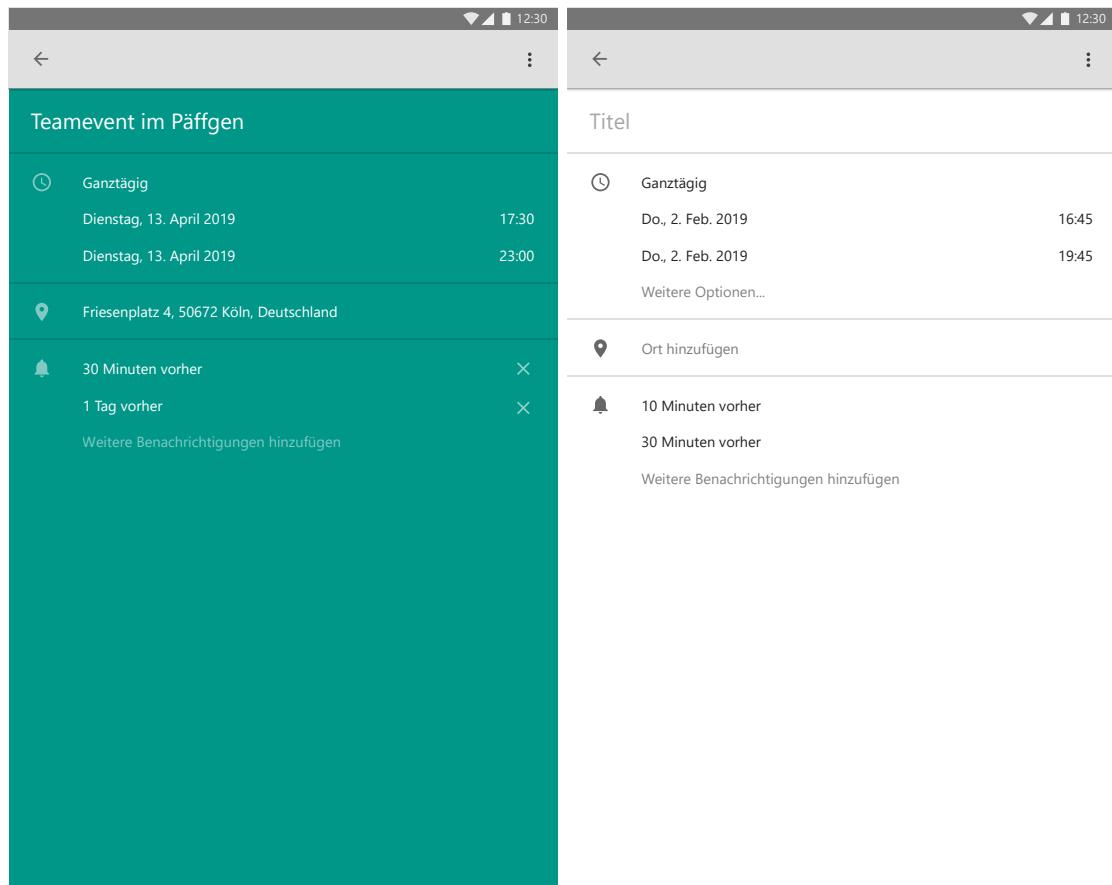


Abbildung 4: Mockups - Übersicht über ein vorhandenes und ein neues Event

Note Activity - Unsere Notizen

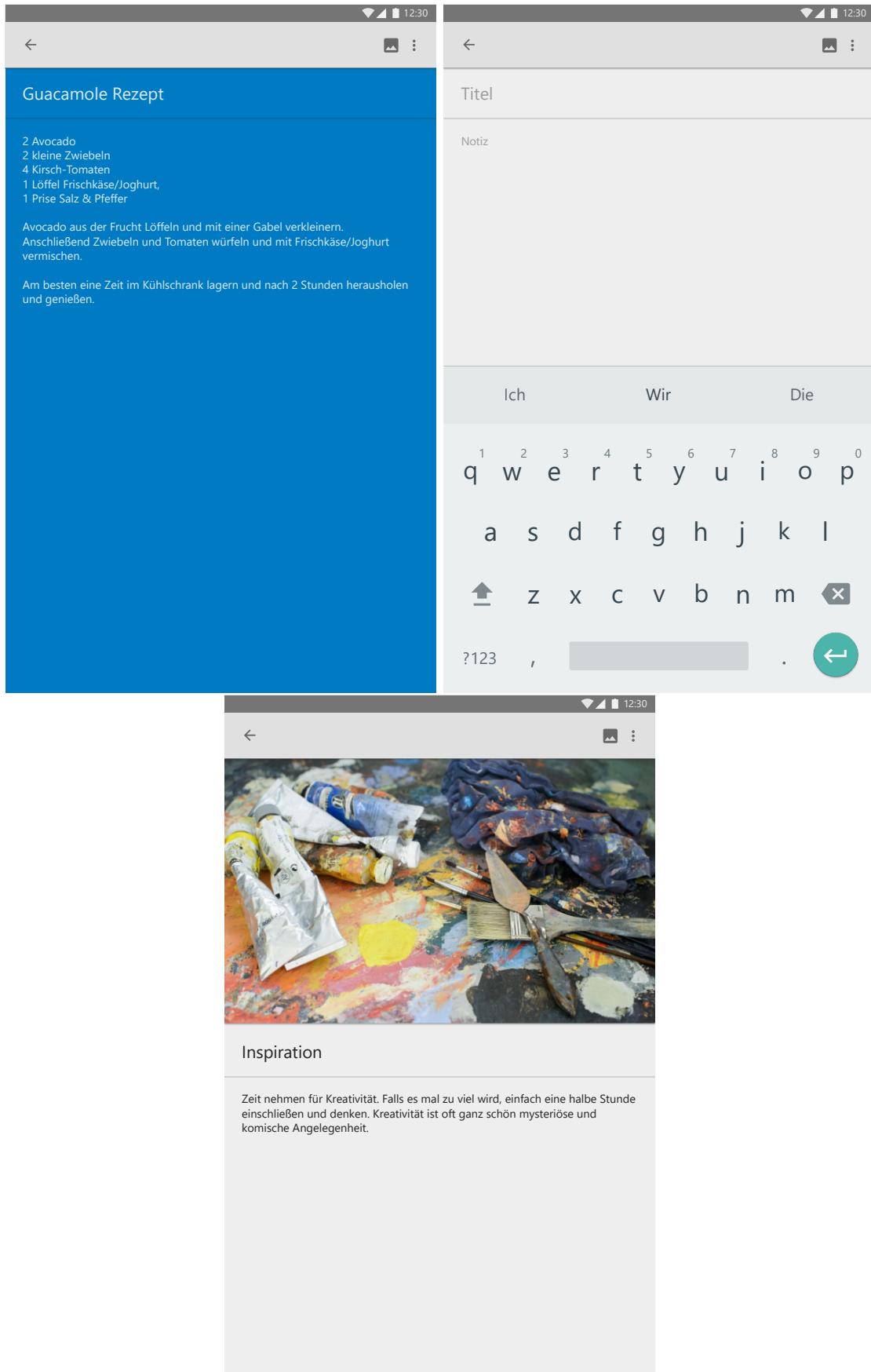


Abbildung 5: Mockups - Übersicht über ein vorhandene, eine leere und eine Bild-Notiz

Todo Activity - Unsere ToDos

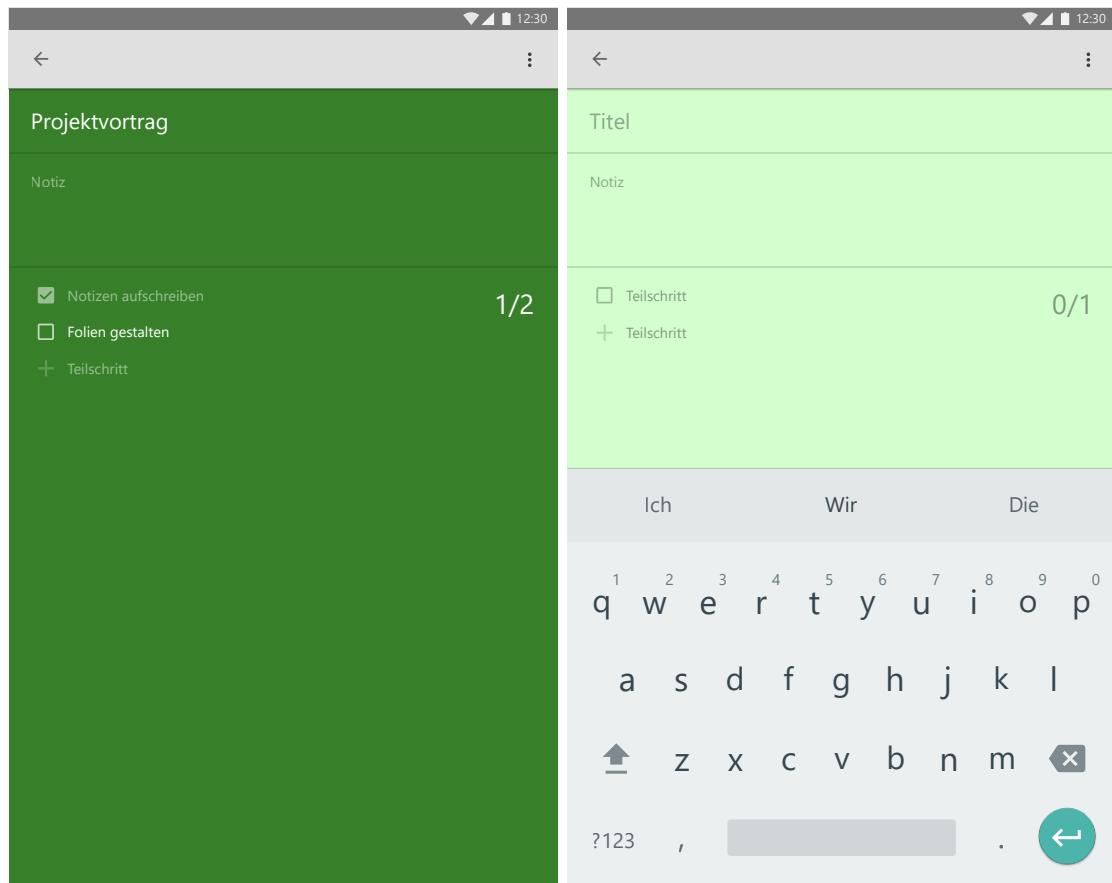
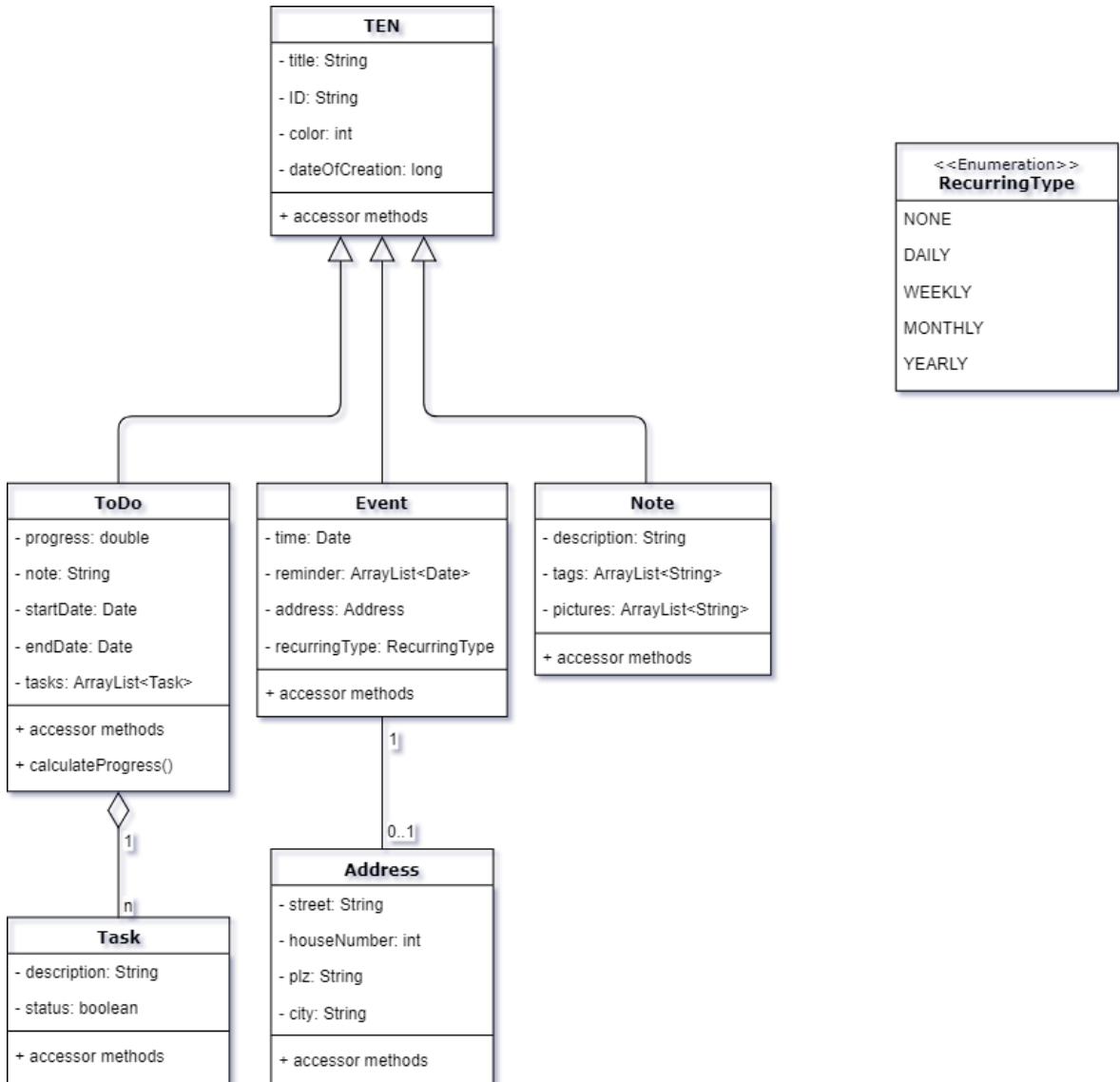


Abbildung 6: Mockups - Übersicht über vorhandene und neue Todos

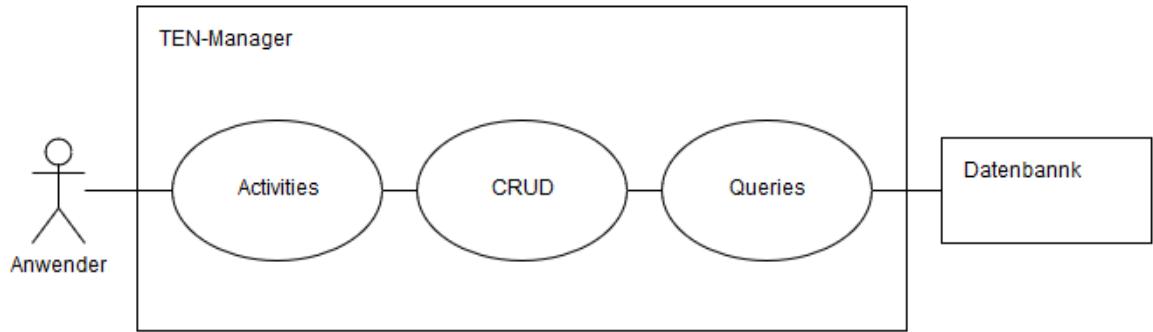
3.3.2 Planung der Datenstruktur und Schnittstellen (Ruthild Gilles)

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

Abbildung 7: 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 8: 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 9: 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)

Hier den Text einfach hin kopieren.

3.3.4 Planung der Navigation zwischen den Activities (Yannick Rüttgers)

Hier den Text einfach hin kopieren.

3.4 Geplante Aufgabenverteilung im Team (Fabia Schmid)

Hier den Text einfach hin kopieren.

So kann man Abbildungen einfügen:

Abbildung 10: Abbildungsbeschriftung



Quelle: <http://dominique-fleury.com/?p=302>

4 Beschreibung des Projektverlaufs

4.1 Tatsächliche Aufgabenverteilung im Team (Fabia Schmid)

Hier den Text einfach hin kopieren.

4.2 Teammeetingprotokolle

Hier den Text einfach hin kopieren.

4.3 Projekttagebücher aller Teammitglieder

Hier den Text einfach hin kopieren.

4.4 Beschreibung von Problemen

Hier den Text einfach hin kopieren.

4.5 Kapitel mit Abbildung

So kann man Abbildungen einfügen:

Abbildung 11: Abbildungsbeschriftung



Quelle: <http://dominique-fleury.com/?p=302>

5 Dokumentation der Software

5.1 Dokumentation der Paketstruktur (Sertan Cetin)

5.2 Dokumentation der Activities

Hier den Text einfach hin kopieren.

5.3 Dokumentation der Navigation zwischen Activities (Yannick Rüttgers)

Hier den Text einfach hin kopieren.

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

Hier den Text einfach hin kopieren.

5.5 Dokumentation der Activity-übergreifenden Klassen (Ruthild Gilles)

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

5.5.1 Models-Klassen

5.5.2 Service-Klassen

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

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

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

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

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

5.6 Kapitel mit Abbildung

So kann man Abbildungen einfügen:

Abbildung 12: Abbildungsbeschriftung



Quelle: Screenshot aus der Benutzeroberfläche

6 Fazit der Teammitglieder

6.1 Fazit von Fabia Schmid

Fazit

6.2 Fazit von Florian Rath

Fazit

6.3 Fazit von Jan Beilfuß

Fazit

6.4 Fazit von Joscha Nassenstein

Fazit

6.5 Fazit von Robin Menzel

Fazit

6.6 Fazit von Ruthild Gilles

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

Zusätzlich wurden während des Projektes allerdings auch einige andere Fähigkeiten gefragt, welche ich noch nicht beherrschte. Dazu gehörte die Versionierung, welche

wir mit Git Hub realisiert haben. Zu Beginn war es für mich eine Herausforderung die Funktionsweise von Git zu verstehen. Nach einiger Einarbeitungszeit beherrschte ich jedoch die Grundfunktionen, sodass ich diese Fähigkeit jetzt auch in meinem restlichen Leben einsetze kann. Abgesehen von der Versionierung, welche nicht für das Projekt gefordert war, stand mir die Aufgabe zu, mich mit dem Textverarbeitungsprogramm von Latex zu beschäftigen. Da uns hier ebenfalls jegliche Einweisung fehlte, benötigte ich auch hier zusätzliche Zeit zum Erlernen der benötigten Grundkenntnisse für Latex. Jedoch werde ich auch diese neuen Kenntnisse außerhalb von dem Modul einsetzen können.

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

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

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

6.7 Fazit von Sertan Cetin

Fazit

6.8 Fazit von Yannick Rüttgers

Fazit

7 Quellenverzeichnis

7.1 Unterkapitelüberschrift

Unterkapitel

7.1.1 Unterunterkapitelüberschrift

Unterunterkapitel

8 Anhang - Quelltexte

8.1 Activities

8.2 Data

8.2.1 Service Klassen (Ruthild Gilles)

Listing 1: Create Klasse (Ruthild Gilles)

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

Listing 2: Read Klasse (Ruthild Gilles)

```

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

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

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

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

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

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

```

Listing 3: Update Klasse (Ruthild Gilles)

```

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

```

Listing 4: Delete Klasse (Ruthild Gilles)

```

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

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

```

8.3 Overview

8.4 Main Activity

9 Anhang - Verwendete Tools und Hilfsmittel (Robin Menzel)

Tool / Programm	Einsatz
Adobe Xd	Erstellung des Mock-Ups
Android Studio	Entwicklungsumgebung (IDE)
Draw.io	Erstellung der UML-Diagramme
Excel	Durchführung der Aufwandsschätzung
GitHub	Versionsverwaltung des Quelcodes
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