

# Projektarbeit

Benjamin Wagner, Etienne Foubert

Sommersemester 2021

## Inhaltsverzeichnis

<b>1</b>	<b>Aufgabenstellung</b>	<b>2</b>
<b>2</b>	<b>Entwurfsmuster</b>	<b>2</b>
2.1	Kommandomuster . . . . .	2
2.2	Strategiemuster . . . . .	3
2.3	Model View Controller . . . . .	4
2.4	Implementierungsvererbung . . . . .	4
<b>3</b>	<b>Applikation</b>	<b>5</b>
3.1	Hauptbildschirm . . . . .	6
3.2	Tasks . . . . .	8
<b>4</b>	<b>Fazit</b>	<b>11</b>
4.1	Entwicklung . . . . .	11
4.2	Urheberrecht . . . . .	12

# 1 Aufgabenstellung

Ziel dieser Projektarbeit ist es eine App für Android zu entwickeln, die das Kommando- und das Strategiemuster nutzt, um eine verständliche Beispielimplementierung für die Nutzung in der Software-Engineering-Vorlesung zu liefern.

Unser persönliches Ziel war es, die App zuverlässig und nützlich zu gestalten, damit wir sie im Alltag verwenden können. Wir entschieden uns für eine simple ToDo-App, in der Aufgaben angelegt und abgehakt werden können. Einer der wichtigsten Design-Aspekte war für uns die Gestaltung des „Erledigens“ von Aufgaben durch Wisch-Gesten.

## 2 Entwurfsmuster

Das Kommandomuster sowie das Strategiemuster wurden an jeweils sinnvollen Stellen implementiert und werden im Folgenden beschrieben. Außerdem sind weitere Muster genutzt worden. Zum einen das Model View Controller Pattern, das im Bereich der Entwicklung von Android- und Web-Apps verwendet wird und das Singleton-Pattern, das in der objektorientierten Programmierung überall anzutreffen ist.

### 2.1 Kommandomuster

Durch die Implementierung des Kommando Entwurfsmusters ist es möglich Aktionen wie das Abschließen oder Löschen eines *Tasks* rückgängig zu machen oder erneut auszuführen.

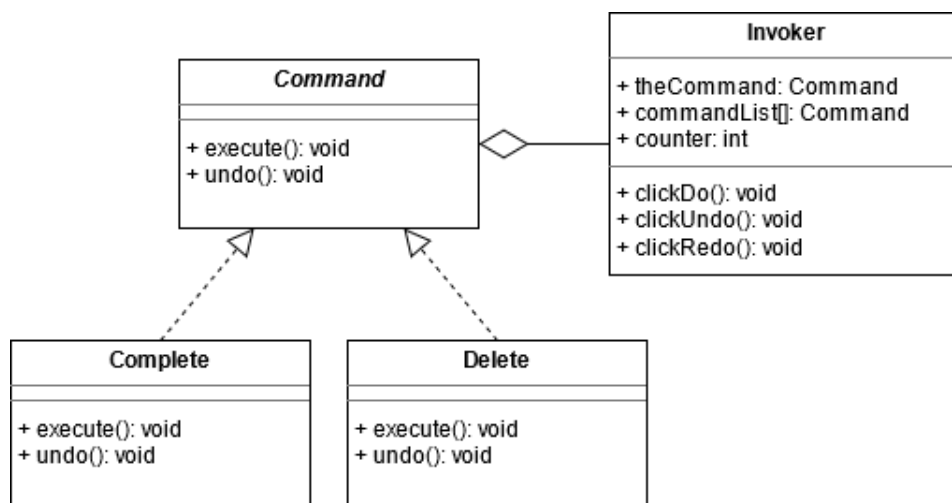


Abbildung 1: UML-Klassendiagramm des Kommandomusters

Realisiert wurde dieses Muster durch vier Klassen. Das Interface *Command* beinhaltet die zwei Methoden `execute()` und `undo()`, die jedes Kommando implementieren muss. In unserem Fall sind das *Complete* und *Delete*, die einen Task von der Tabelle der aktiven Tasks in die Tabelle der abgeschlossenen beziehungsweise gelöschten Tasks verschiebt. Die `undo`-Methoden fallen hier sehr einfach aus, da sie nur den Task wieder in die andere Richtung zwischen den Tabellen bewegen.

Die Übersicht über die ausgeführten Kommandos behält der *Invoker* in der `commandList`. Falls eine Eingabe des Users erfolgt, wird in der *MainActivity* mit `invoker.setCommand()` das entsprechende Kommando gesetzt und anschließend mit `clickDo()` ausgeführt. Bei

Aktivierung des *Undo* oder *Redo*-Knopfes werden analog dazu die Methoden `clickUndo()` oder `clickRedo()` genutzt.

Neben der Liste mit Kommandos besitzt der *Invoker* außerdem einen Zähler, der die aktuelle Position in der Liste der Kommandos festhält. Wird `clickUndo()` ausgeführt, wird der Zähler dekrementiert und `undo()` des letzten Kommandos aufgerufen. Bei `clickRedo()` wird im Gegensatz dazu der Zähler inkrementiert und die `execute()`-Methode des entsprechenden Kommandos ausgeführt.

Neben den genannten Methoden sind auch einige Hilfsmethoden implementiert, zum Beispiel um den Zustand der Undo- und Redo-Knöpfe abzufragen.

## 2.2 Strategiemuster

Das Strategiemuster stellt ein Interface zur Verfügung, welches die Implementierung unterschiedlicher Ausführungen, der selben Aktion ermöglicht. Wir nutzen das Strategiemuster für die Implementierung unterschiedlicher Alarm- und Benachrichtigungsdienste. Diese sind eine Push-Benachrichtigung, ein Vibrationsalarm und das Ausbleiben eines Alarmes.

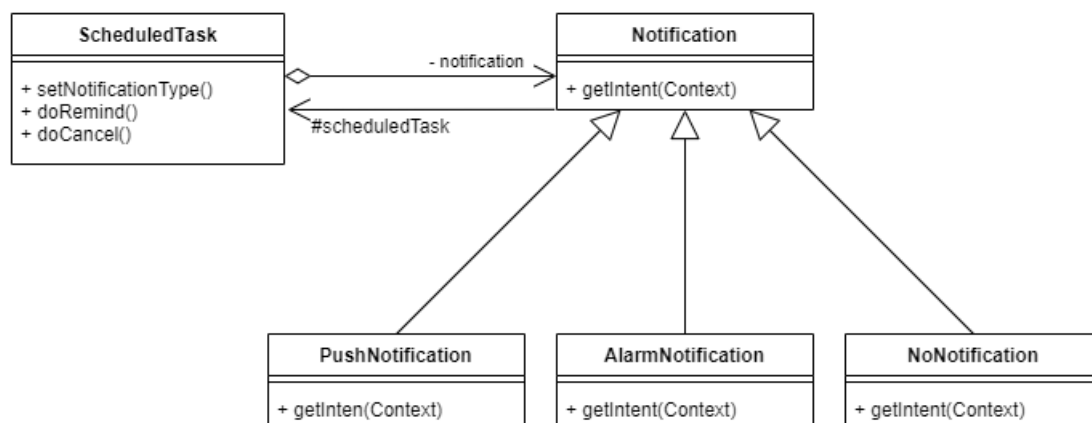


Abbildung 2: UML-Klassendiagramm des Strategiemusters

Das Strategiemuster basiert auf einer Interface-Klasse. Diese gibt vor welche Funktionen die Sub-Klassen implementieren müssen. In unserem Fall ist das die Klasse *Notification*, welche die abstrakte Methode `getIntent()` besitzt. Diese Methode erzeugt je nach Notification-Typ einen Android Intent und startet einen Alarm. Die drei Benachrichtigungsstrategien *PushNotification*, *AlarmNotification* und *NoNotification* implementieren alle das Interface *Notification*, und überschreiben deren Methode.

Wird nun ein Task erstellt, kann diesem eine dieser Strategien in Form eines Objektes der zugehörigen Klasse, hinterlegt werden. Soll einem Task z.B. ein Alarm hinterlegt werden, wird ein Objekt der Klasse *AlarmNotification* erstellt und eine Referenz dazu in dem Task gespeichert. Beim Erstellen des Tasks wird nun die Methode `doRemind()` aufgerufen und der Android-Alarm gesetzt. Beim Löschen des Tasks wird die Funktion `doCancel()` aufgerufen, und der Alarm wird gelöscht. Beide Methoden greifen auf die *Notification* zurück und überschreiben deren Methode `getIntent()`. Vorsicht, Verwechslungsgefahr durch gleichnamige Android-Klasse. Das „Context“-Objekt welches als Funktionsparameter der

`getIntent()` Methode übergeben wird ist im Code auf den Android Context einer Activity zurückzuführen und ist nur indirekt mit dem „Context“ des Strategy-Pattern verwandt, welcher hier in erster Linie durch den „ScheduledTask“ repräsentiert wird. Im Ordner `strategy_pattern` ist ebenfalls ein Beispiel als Konsolen Applikation aufgeführt.

## 2.3 Model View Controller

Der Aufbau der Applikation ist an des MVC-Muster angelehnt, in dem eine Applikation in drei Bereiche aufgetrennt wird. Das Model welches für die Datenspeicherung verantwortlich ist, die View welche das Interface zu dem User bildet, und den Controller welcher die Kommunikation zwischen View und Model realisiert.

Die Daten werden auf einer SQL-Datenbank abgelegt, diese besteht aus drei Tabellen. Jeweils eine für alle aktiven, beendeten und gelöschten Aufgaben. Um die Datenspeicherung und den Datenzugriff einheitlich zu gestalten wurde eine API geschrieben. Diese ermöglicht das Erstellen von den unterschiedlichen drei Aufgabentypen (Normal, Liste, Termin), sowie das Verändern der Einträge. Auf einen bestimmten Eintrag kann per ID zugegriffen werden. Es können aber auch alle aktiven/beendeten oder gelöschten Tasks auf einmal abgerufen werden, sowie die Anzahl als Integer der jeweiligen Einträge in den Tabellen. Außerdem können die Einträge zwischen den Tabellen hin und her geschoben werden, z.b. um einen aktiven Task zu beenden.

Die Android-Activities benutzen diese API als Schnittstelle zu den gespeicherten Daten über die Klasse *DatabaseHelper*. Diese ist zudem aus Performance Gründen als Singleton realisiert. So bleiben Datenspeicherung und Frontend sauber voneinander getrennt.

## 2.4 Implementierungsvererbung

Für die Tasks wurde eine Basisklasse *Task* aufgebaut, von der die drei spezialisierten Klassen erben, wie in Abbildung 3 zu sehen ist. Alle drei Tasks besitzen einen Titel sowie eine ID. Das Feld `notificationType` wird nur von *ScheduledTask* genutzt.

Außerdem im UML-Diagramm zu erkennen ist die Klasse *ListItem*, die Funktionalität für das Abhaken der Elemente in einem *ListTask* bereit hält.

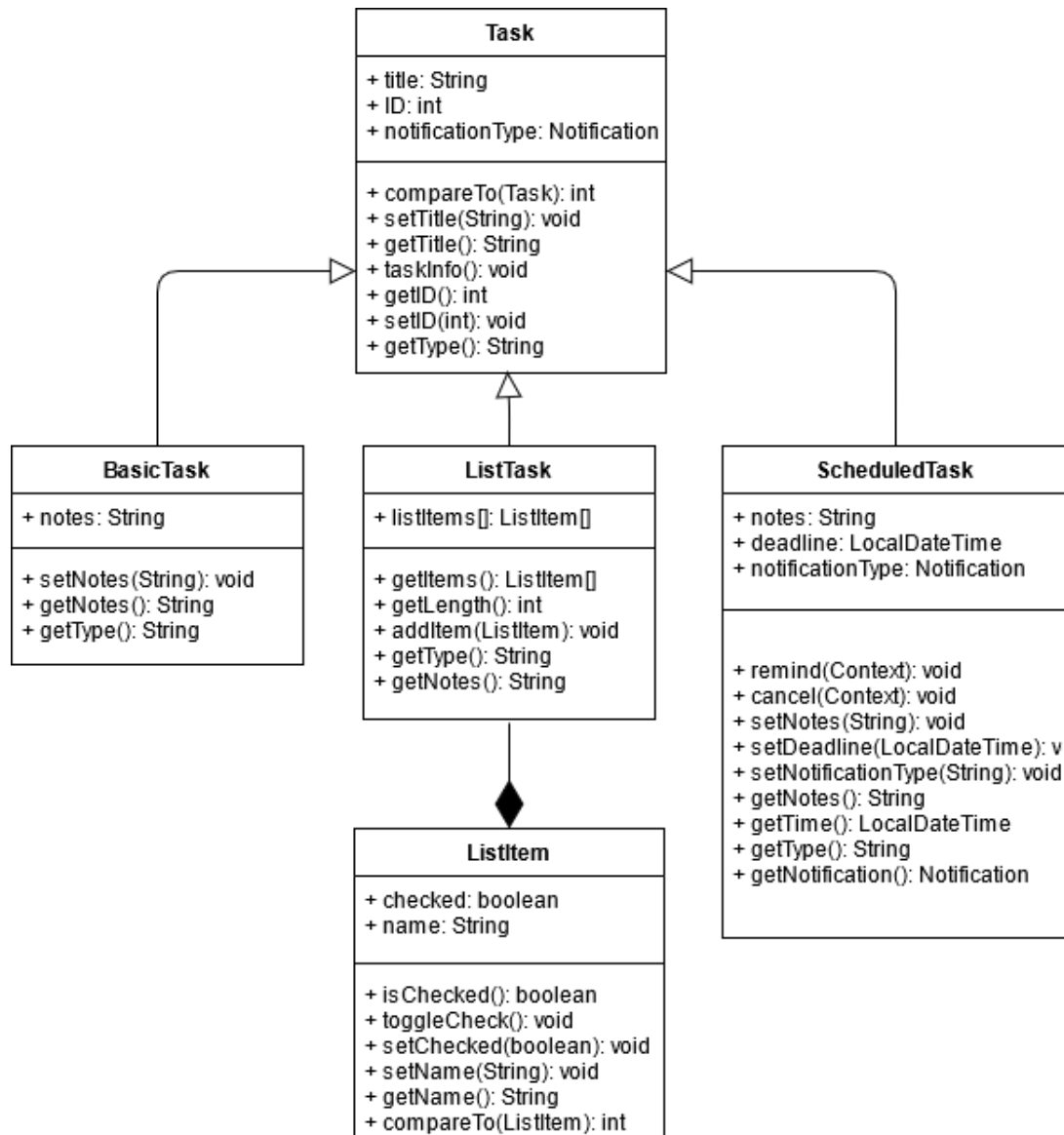


Abbildung 3: UML-Klassendiagramm der Tasks

### 3 Applikation

Im Folgenden werden die Oberflächen der App, sowie die zugrundeliegenden Strukturen im Programmcode erläutert. Dieser Abschnitt dient außerdem als Anleitung zur Nutzung der App.

## 3.1 Hauptbildschirm

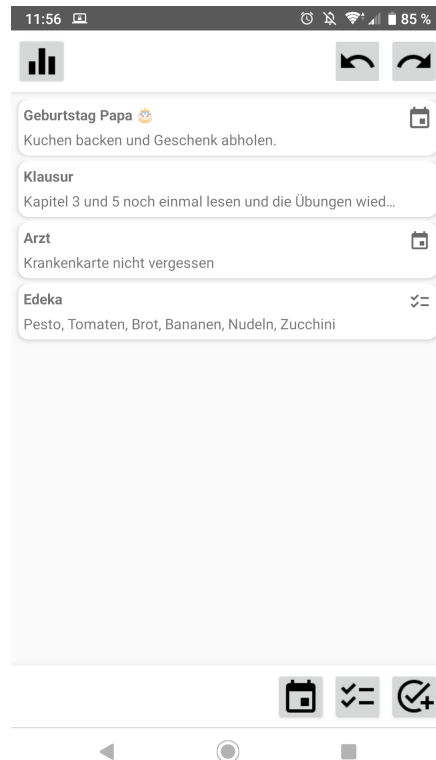


Abbildung 4: Main Screen

Auf dem Hauptbildschirm (vgl. Abbildung 4) wird die Liste der Aktiven Aufgaben angezeigt. Unter dem Titel einer Aufgabe stehen zusätzliche Notizen/Informationen sowie rechts ein Symbol welches den Typ der Aufgabe anzeigt (Standard, Liste, Termin). Am Oberen Bildschirmrand befindet sich der „Statistik“-Knopf welcher zu einer Übersicht über alle bereits erledigten und gelöschten Aufgaben führt. Daneben befinden sich die „Undo“- und „Redo“- Knöpfe, welche es ermöglichen das Löschen/Beenden einer Aufgabe zu widerrufen. Am unteren Bildschirmrand befinden sich Knöpfe welche das Hinzufügen einer der drei Aufgaben-Typen ermöglicht.

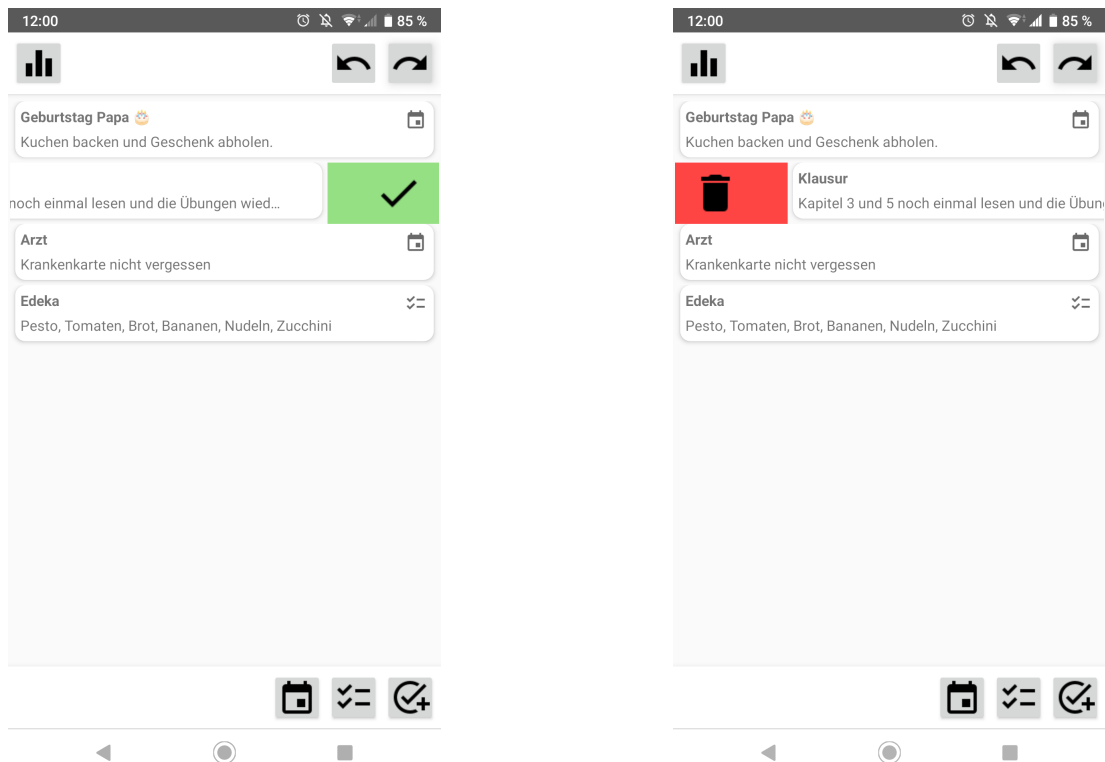


Abbildung 5: Swipe

Die Aufgaben können per Drag-and-Drop nach belieben sortiert werden. Via Swipe-Geste nach links kann eine Aufgabe erledigt oder nach rechts gelöscht (vgl. Abbildung 5) werden.

## 3.2 Tasks

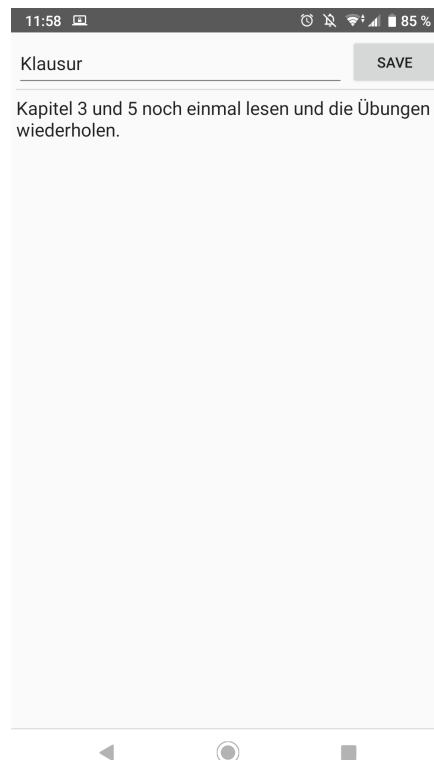


Abbildung 6: Task

Ein Standardtask besteht aus einem Titel und einer Notiz (vgl. Abbildung 6).

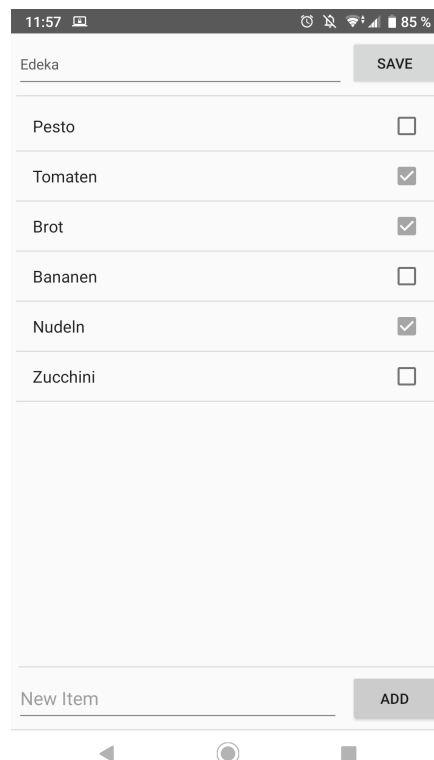


Abbildung 7: List



Eine Liste besteht aus einem Titel und einem oder mehreren Einträgen (vgl. Abbildung 7). Diese können per Knopfdruck auf den erledigt Status gesetzt werden. Erledigte Einträge werden nach einem Neuladen des Screens nach unten sortiert.

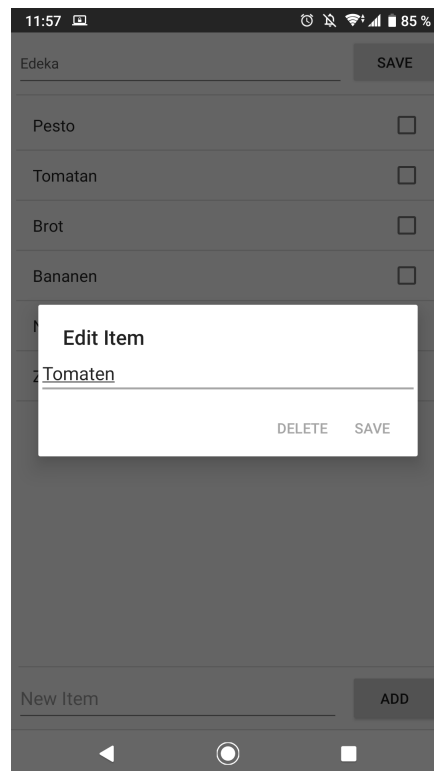


Abbildung 8: Edit List Item

Hält man einen Eintrag gedrückt kann dieser nachträglich noch bearbeitet oder gelöscht werden (vgl. Abbildung 8).

Ein Termin besteht aus einem Titel und einer Notiz (vgl. Abbildung 9). Ebenfalls muss der Termin ein Datum und kann eine Uhrzeit enthalten, zu welcher der User Benachrichtigt werden soll. Hierfür muss eine Benachrichtigungsart ausgewählt werden.

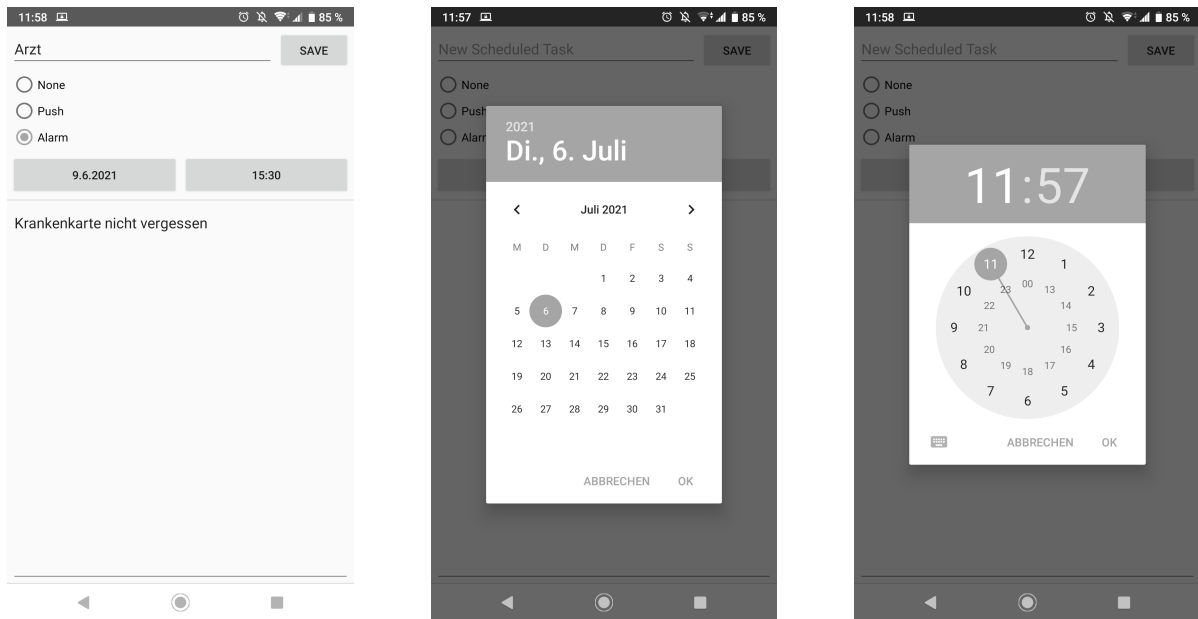


Abbildung 9: Termin

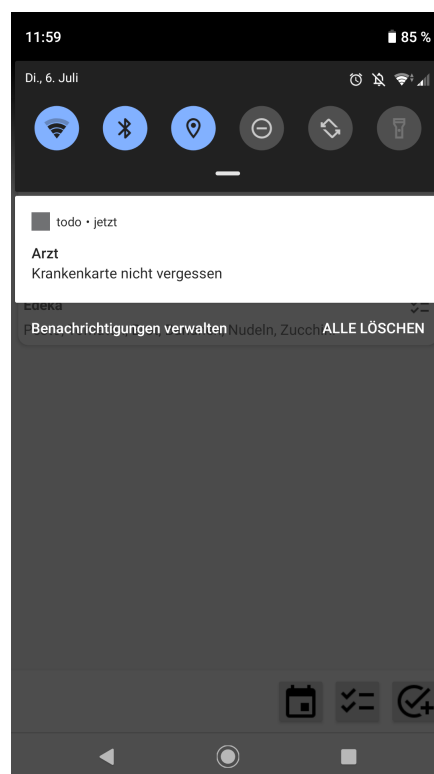
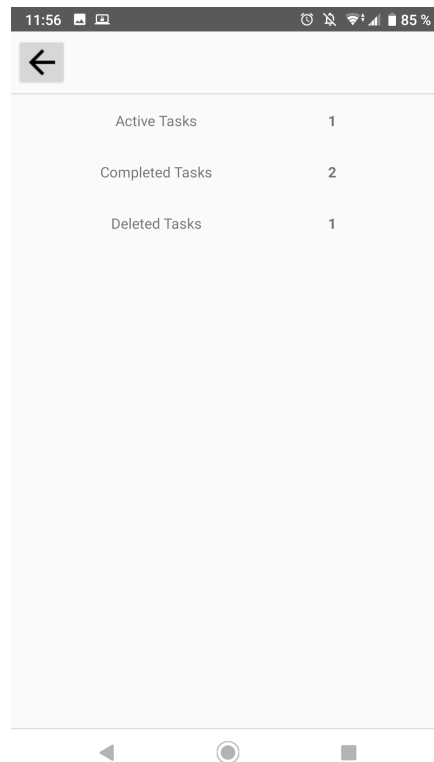


Abbildung 10: Push Notification

Die Push Benachrichtigung erzeugt eine Push-Notification (vgl. Abbildung 10) am oberen Bildschirmrand des Handys (auch bei geschlossener App), der Alarm erzeugt einen

Vibrationsalarm.



Active Tasks	1
Completed Tasks	2
Deleted Tasks	1

Abbildung 11: Overview

Der Statistik-Screen (vgl. Abbildung 11) welcher über den Knopf rechts oben am Hauptscreen erreichbar ist, zeigt wieviele Tasks offen, erledigt und gelöscht worden sind. Mit Klick auf „Completed“ - oder „Deleted“-Tasks wird man auf einen Screen weitergeleitet auf dem diese aufgelistet sind.

## 4 Fazit

Unser Ziel der übersichtlichen Implementierung wurde unserer Einschätzung nach erfüllt, gleichzeitig wurden die Muster an sinnvollen Stellen eingesetzt.

Die persönliche Anforderung an die App, dass sie im Alltag nützlich sein soll, wurde erfüllt, da wir sie beide für Notizen sowie Einkaufslisten nutzen.

### 4.1 Entwicklung

Während der Entwicklung nutzen wir Github als Versionskontrolle. Hier war besonders der Issue-Tracker sehr hilfreich, da wir hier vorhandene Bugs, geplante Features oder GUI-Verbesserungen festhielten, mit eigenen Tags versehen und einem Entwickler zuwiesen. Im Developmentprozess haben wir über 50 Issues angelegt und bearbeitet.

Weiterhin genutzte Software war GIMP für die Erstellung des Logos, Draw.io für die UML-Diagramme und Android Studio für die Entwicklung der eigentlichen App.

## 4.2 Urheberrecht

Diese App wurde entwickelt, um eine Beispielimplementierung für die verwendeten Entwicklungsmuster zu liefern und darf daher für Vorlesungen oder anderen Veranstaltungen im Rahmen von Bildung und Ausbildung genutzt werden.