Entwicklung einer Android Food Tracking App in der Programmiersprache Kotlin

Johannes Franz & Normen Krug

Wintersemester 2017/2018

Vorgelegt bei Prof. Dr. Michael Stepping

Inhaltsverzeichnis

1 Einleitung

Ziel ist es eine App zu entwickeln, die einen Rückschluss von sich zugenommener Nahrung auf Symptome zu ermöglichen. Dabei helfen die in der App eingetragenen Datenpunkte und Fotos diese z.B. mit einer allergischen Reaktion zu verknüpfen. Zielgruppe sind Menschen, die wegen Erkrankungen aus ihren zu sich zugenommenen Speisen und Getränken Rückschlüsse auf ihr Wohlbefinden treffen wollen, um zukünftig solche Speisen zu meiden.

2 Motivation

Durch eigenen Bedarf motiviert entstand die Idee zu dieser App. Um aus wiederkehrenden Mahlzeiten eine Unverträglichkeit abzuleiten, stellt die App entsprechende Funktionen bereit.

3 Kotlin als Programmiersprache

Für die App wurde die Programmiersprache Kotlin verwendet, welche auf der Google IO im Mai 2017 als offizielle Sprache für Android eingeführt wurde. Kotlin hat einen besser lesbaren Syntax als Java. Wie bei Java wird auch bei Kotlin der Bytecode für die Java Virtual Machine übersetzt. Insgesamt ist Kotlin der Programmiersprache Swift 3 bzw 4 syntaktisch sehr nahe, was den Umgang als Entwickler mobiler Anwendungen zusätzlich erleichtert.

Kotlin Code Beispiel:

```
package hello
import kotlin.collections.* // line comment

/**
 * Doc comment here for ‘SomeClass‘
 * @see Iterator#next()
 */
@Deprecated("Deprecated class")
private class MyClass<out T : Iterable<T>>(var prop1 : Int) {
    fun foo(nullable : String?, r : Runnable, f : () -> Int,
        fl : FunctionLike, dyn: dynamic) {
        println("length\nis ${nullable?.length} \e")
        val ints = java.util.ArrayList<Int?>(2)
        ints[0] = 102 + f() + fl()
        val myFun = { -> "" };
        var ref = ints.size
        ints.lastIndex + globalCounter
        ints.forEach lit@ {
            if (it == null) return@lit
            println(it + ref)
        }
        dyn.dynamicCall()
        dyn.dynamicProp = 5
    }

    val test = """hello
                  world
                  kotlin"""
    override fun hashCode(): Int {
        return super.hashCode() * 31
    }
}
```

```
fun Int?.bar() {
    if (this != null) {
        println(message = toString())
    }
    else {
        println(this.toString())
    }
}
var globalCounter : Int = 5
    get = field
abstract class Abstract {
}
object Obj
enum class E { A, B }
interface FunctionLike {
    operator fun invoke() = 1
}
```

4 Entwurfsphase

In der Entwurfsphase wurde keine Zeit verschwendet und somit wurde der Fokus auf Handskizzen gelegt. Der Zwischenschritt diese Skizzen in einem professionellen Entwurf nachzubauen sparte Zeit, so dass sich direkt mit der Programmierung beschäftigt werden konnte.

4.1 Main Screen

Abbildung 4.1: Main Screen

4.2 Hinzufügen eines Eintrags

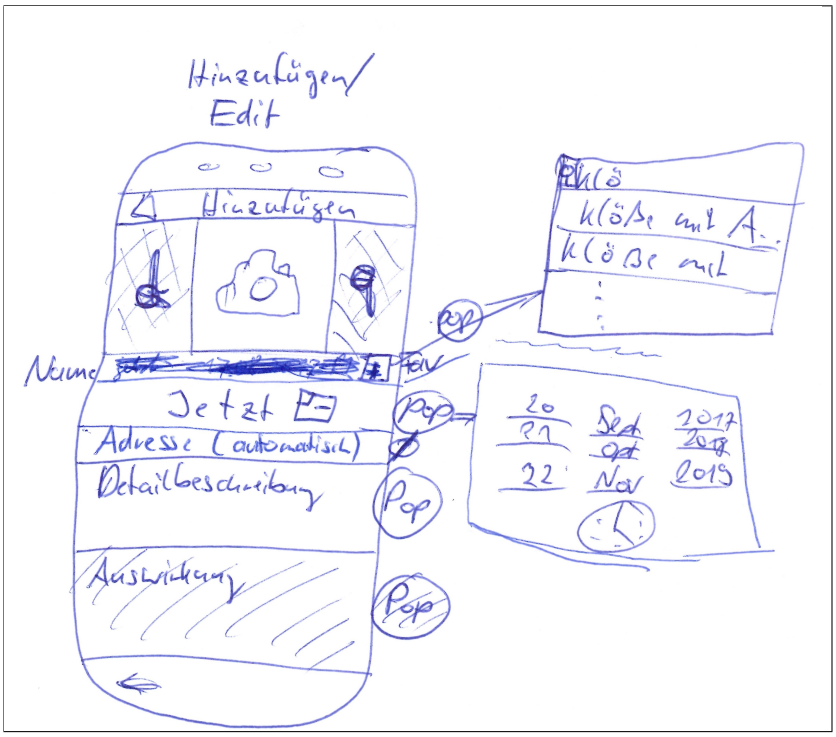

Abbildung 4.2: Hinzufügen eines Eintrags

4.3 Popover im Main Screen

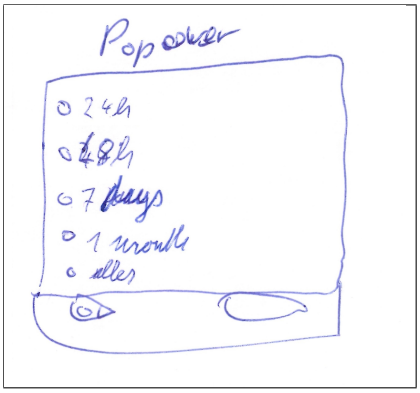

Abbildung 4.3: Popover Screen

4.4 Settings

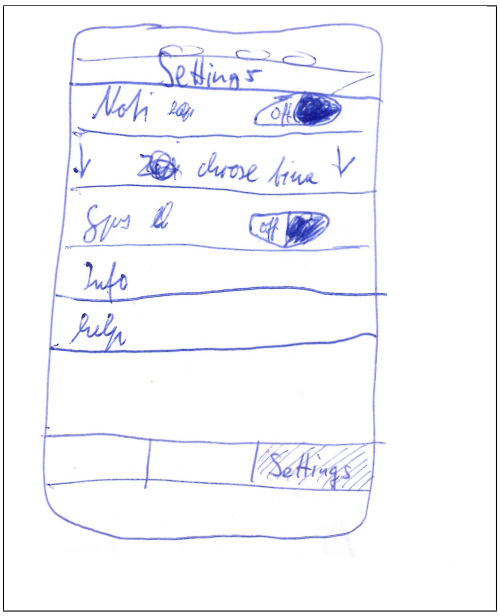

Abbildung 4.4: Settings Screen

4.5 Navigation Drawer Menü

Der auch unter "Hamburger Menü"bekannte Navigation Drawer war einer der ersten Ideen für die App. Viele Funktionen später als Tab Bar bzw. als Icons in der oberen rechten Ecke umgesetzt wurden, sind diese Buttons an die genannten Stellen hin verschoben worden. Im Sinne der Anwenderfreundlichkeit wurde sich am Ende gegen den Navigation Drawer entschieden.

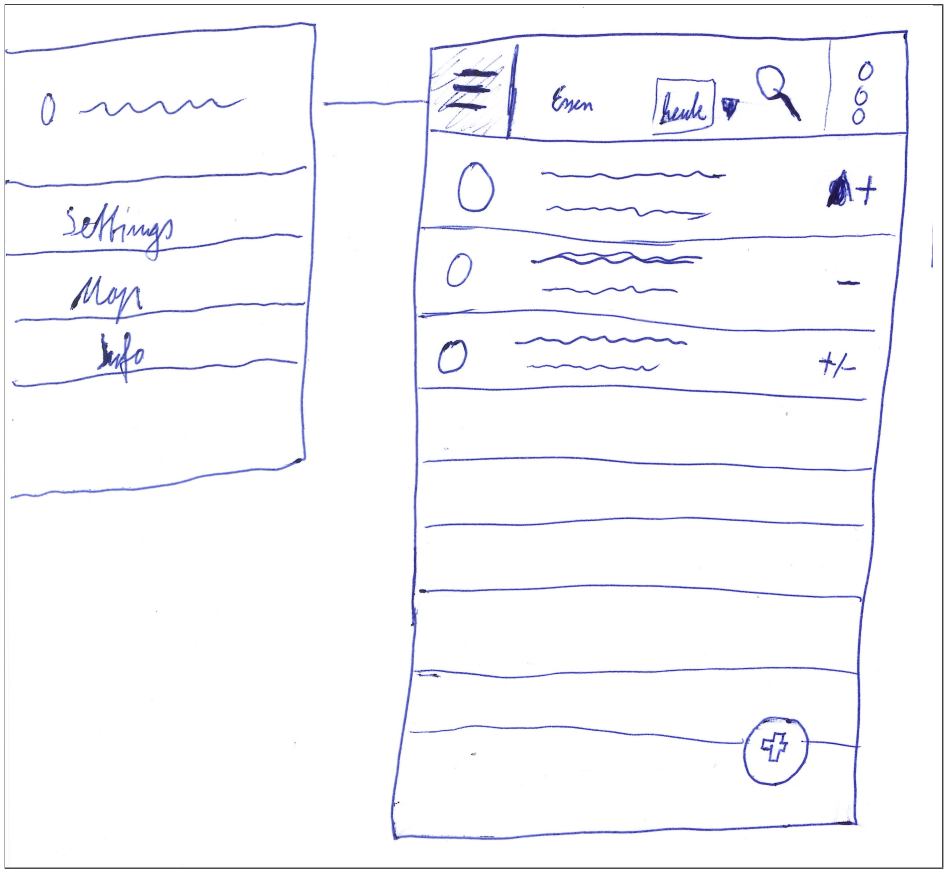

Abbildung 4.5: Navigation Drawer Menü Screen

4.6 Karten Screen

Der Karten Screen zeigt eine Google Map. Darin repräsentieren Pins die Standorte an denen Essens Einträge hinzugefügt wurden.

Abbildung 4.6: Karten Screen

5 Umsetzung

5.1 Datenbank

Für die Speicherung der Daten wurde sich für eine SQLite Datenbank entschieden. Das hat mehrere Vorteile. Zum einen lässt sich der Inhalt dieser Datenbank sehr einfach exportieren, um auf einem anderen Gerät wieder eingespielt werden zu können, zum anderen bietet eine Datenbankstruktur viele Möglichkeiten zur Datenanalyse mit schnellem Zugriff ohne wie bisher mit Files arbeiten zu müssen.

5.1.1 Room

Room wurde als Datenbank Abstraktionsschicht verwendet und bietet einfachen Datenbankzugriff ohne komplexe SQL Befehle. `https://developer.android.com/topic/libraries/architecture/room.html`

5.1.2 Google Maps

Für die Karte ist die Google Maps API eingesetzt worden. Darin wird für jeden Eintrag ein Pin in der Karte dargestellt. Der API Key wurde dankensweise von Normen bereitgestellt.

6 Herausforderungen

In jedem Projekt treten bei ansteigender Komplexität Probleme auf. Diese sind in diesem Abschnitt vermerkt.

- Aufwändige Integration von Dagger 2 die letztendlich verworfen werden musste. Mangelnde Dokumentation. Probleme beim Integrieren von bereits vorhandenen Projekten.

7 Lessons learned

Dem Parkinsonschen Gesetz folgend wurde die Features während der vorhandenen Zeit umgesetzt. Da während der Appentwicklung die Stundenplan App der Hochschule betreut wurde, entstand gelegentlich ein Interessen- und Ressourcenkonflikt. Letztendlich wurden die Änderungen an der Schnittstelle leicht gewichtet behandelt, da diese einen Mehrwert für viele Hundert Studenten auf Android und iOS Seite mit sich brachten.

8 Weitere Arbeiten

In diesem Projektabschnitt konnten alle gesteckten Ziele erreicht werden. Da solch ein junges Projekt noch viele Möglichkeiten beinhaltet Funktionen zu verbessern und neue Funktionen einzuführen werden hier mögliche Punkte zur Anregung aufgelistet.

Abbildungsverzeichnis