Projektbericht Android App “FroYou”

Julian Schoemaker
Student, 11117045
Gummersbach, Deutschland
Julian.schoemaker@smail.th-koeln.de

Sebastian Faust
Student, 11118719
Gummersbach, Deutschland
Sebastian.Faust@smail.th-koeln.de

Dario Giuseppe Lazzara
Student, 11119142
Gummersbach, Deutschland
dario_giuseppe.lazzara@smail.th-koeln.de

ABSTRAKT
Eine Dokumentation des Design Prozesses, der Design Entscheidungen und der Architektur der App „FroYou“ die im Rahmen des Modules Mobile Computing an der TH Köln im Sommersemester 2018 entwickelt wurde. FroYou ist eine App, die in Kooperation mit dem Süßigkeiten-Geschäft „Süße Ecke“ im Forum Gummersbach entwickelt wurde und dem Kunden bei der Zusammenstellung und Bestellung eines Frozen Yogurts unterstützt.

Autoren Kennwörter
Mobile Computing; Android; TH-Köln; App; Frozen Yogurt;

ACM Klassifikations - Kennwörter
Mobile Computing; Mobile phones; Publish-subscribe / event-based architectures; Design;

EINLEITUNG
Der Bestellvorgang eines Frozen Yogurts ist mit viel Arbeit und Zeitaufwand sowohl für den Verkäufer als auch für den Kunden verbunden. Der Kunde selbst muss sich die Zeit nehmen sich sowohl über einzelne Zutaten zu informieren als auch Gedanken zu machen welche Zutaten in welcher Kombination gut zusammenpassen. Da es eine große Menge an Zutaten gibt, ist dieser Prozess sehr zeitaufwändig. Zudem gibt es keine Möglichkeit aus ausgewählten Rezepten zu wählen, um einen Frozen Yogurt öfter zu kaufen. Zeitgebundene Kunden haben so nicht die Möglichkeit einen Frozen Yogurt schnell zu bestellen.

Unsere Idee basiert darauf, die Zusammenstellung eines Frozen Yogurts in erster Linie zu beschleunigen, aber auch für den Kunden praktischer, einfacher und interaktiver zu gestalten. Der Kunde soll die Möglichkeit haben schon lange im Voraus seinen Yogurt zusammenzustellen und nicht erst bei der Bestellung vor Ort. Sowohl Jung als auch Alt soll spielerisch leicht einen Frozen Yogurt ganz nach eigenen Vorstellungen kreieren können, sodass am Ende der Verkäufer lediglich die Bestellung aufnimmt und den Joghurt für den Kunden erstellt. Das Prinzip des Zusammenstellens des Yogurts soll baukastenartig erfolgen, sodass keinerlei Fragen zur Reihenfolge oder Art der Zutaten aufkommen. Außerdem sollen Kunden die Möglichkeit haben ihre eigenen Kreationen in der App anonym zu veröffentlichen, sodass auch andere Kunden Zugriff auf die Rezepte haben und diese bei Belieben nachbestellen können. Falls ein Kunde keine Zeit oder keine Idee für eine Joghurtkreation hat, soll dieser nicht abgeschreckt sein von der Aufgabe sich selbst einen Joghurt zusammenstellen zu müssen. Er soll die Option haben sich bequem ein bereits kreierten Yogurt aussuchen zu können.

Besonders wichtig bei der App war es uns diese nicht komplett ohne Zusammenhang zum Forum Gummersbach zu entwickeln. Wir wollten in Zusammenarbeit mit einer Filiale eine App entwickeln, sodass wir uns so realitätsnah wie möglich an die Entwicklung begeben konnten. Unsere App sollte am Ende der Entwicklung einen Sinn erfüllen und nutzbar sein und nicht nur den Zweck eines studentischen Projektes erfüllen. Wir haben uns mit dem Süßigkeiten-Geschäft „Süße Ecke“ im Forum in Gummersbach in Verbindung gesetzt, welches Frozen Yogurts verkauft. Sie haben sich dazu bereiterklärt uns jegliche Ressourcen wie z.B. Preise, Größen der Joghurts oder Zutaten bereitzustellen. Da wir selbst Kunden des Geschäfts „Süße Ecke“ sind, war es für uns eine zusätzliche Motivation den Bestellprozess über eine App zu optimieren. Außerdem sehen wir in der App weiteres Potenzial, um darauf aufbauend Funktionalitäten zu implementieren.

Bei der Umsetzung haben wir uns mit den Anforderungen des Inhabers als Kooperations-Partner befasst. Ihm war es wichtig eine möglichst einfache Kommunikation bei der Bestellung zwischen Kunde und Mitarbeiter herzustellen.

DESIGN PROZESS

Zunächst beschreiben wir den Design Prozess der App. Unser erster Schwerpunkt war es, die nötigen Funktionalitäten und das damit zusammenhängende User Interface zu konzipieren, um für die spätere Umsetzung einen möglichst guten Eindruck über den Aufbau zu haben.

Relevante Activities finden

Im ersten Schritt haben wir uns einen Überblick über die nötigen Activities geschaffen. Mit einem einfachen Vorentwurf der Seitenstruktur konnten wir grob einschätzen welche Reihenfolge die Activities haben müssen.

Wireframes

Nachdem wir die relevanten Activities ausfindig gemacht hatten, haben wir begonnen Wireframes zu erstellen, auf denen wir die Funktionalitäten der einzelnen Activities gesammelt haben. Da wir von Beginn an unseren Fokus darauf gelegt haben, den Bestellprozess so einfach wie möglich zu gestalten, haben wir bereits bei den Wireframes Feedback beim Inhaber und der Filialleiterin des Kooperationspartners eingeholt. So konnten wir im weiteren Design Prozess sowohl die bereits gemachten Erfahrungen der Mitarbeiter bei der Aufnahme von Bestellungen, als auch unsere Erfahrungen als Kunden mit einbringen.

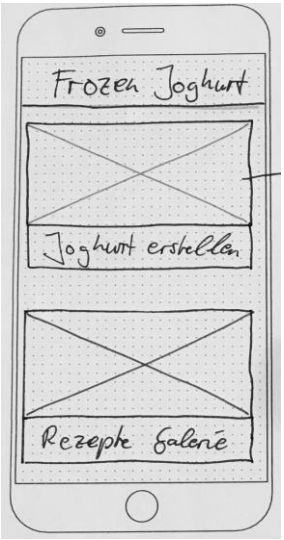

Abbildung 1. Wireframe der Main Activity

Erstellung eines Prototypen

Bevor wir in die Implementierung in XML Layout Dateien übergegangen sind, haben wir die, nach dem Feedback korrigierten, Wireframes in einen klickbaren Prototypen überführt. Dies haben wir mit der frei verfügbaren UX-Design Software Adobe XD umgesetzt. Adobe XD bietet viele Funktionen, die auf die Entwicklung von Interfaces für Android Apps ausgelegt sind. Zum Beispiel bietet es die Möglichkeit Icons, Hintergründe und Grafiken direkt für Android Studio zu exportieren. Zudem kann man über eine App von Adobe XD für Smartphones den erstellten Prototyp direkt auf dem Gerät testen. Wir konnten somit das User Interface schon vor der Implementierung in Bezug auf User Flow, Farbwirkung, Lesbarkeit und Größe von Layoutelementen optimieren.

Durch die Farbgebung und den Einsatz von Icons haben wir einen verspielten und freundlichen Look gewählt, der zur Domäne passt. Außerdem kennzeichnet es unsere App aus und ist eine Art Alleinstellungsmerkmal, dass uns von anderen Apps unterscheidet.

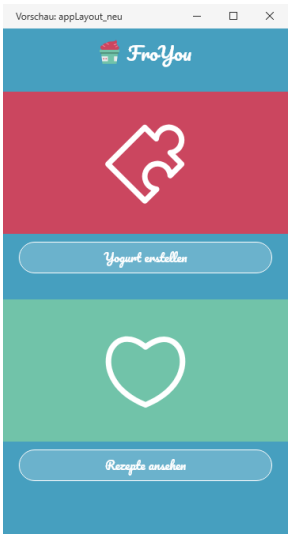

Abbildung 2. Prototype der Main Activity in Adobe XD

Aufbau der XML Layout Dateien

Durch unsere umfangreiche Konzeptionsphase konnten wir nun die Activities unkompliziert in XML Layout Dateien aufbauen. Dadurch, dass wir bereits einen Überblick über die Struktur der Activities hatten, war es für uns einfacher mit der bisher unbekannten Layouterstellung über XML Dateien zu arbeiten.

Der Grundaufbau basiert auf dem Layoutkonzept „Constraint-Layout“, dass Abhängigkeiten von Elementen zueinander spezifizieren kann und man ein responsives Verhalten sicherstellen kann. Unsere App ist sowohl für normale Smartphones, als auch für Tablets ausgelegt. Wir haben uns dabei auf den Portrait-Modus beschränkt, da wir keinen Vorteil in einer Ausweitung auf Landscape gesehen haben. Auch das Testen der App auf Smartphones, bei denen die komplette System-Anzeige auf dem Gerät vergrößert wurde, wurde abgesichert. Hier mussten wir bei einigen Activities nachträglich noch „Scrollviews“ einbauen, damit Elemente nicht außerhalb des Viewports unerreichbar sind.

FUNKTIONALITÄT

Im Folgenden werden wir die Hauptfunktionalität der App beschrieben. Wir werden auf jeden Screen eingehen und die Funktionen der Activity aufzeigen. Die Titel der Activities sind in Englisch benannt, da wir in Bezug auf Klassennamen konsistent mit unserem Code bleiben wollten.

Main Activity

Die Main Activity stellt die Startseite der App dar. Der Nutzer kann wählen, ob er einen neuen Yogurt erstellen oder auf bestehende Rezepte zugreifen will. Zudem bietet die Main Activity den Zugriff auf ein Untermenü, dass die Verlinkungen zur App Information, den Einstellungen und dem Impressum aufweist.

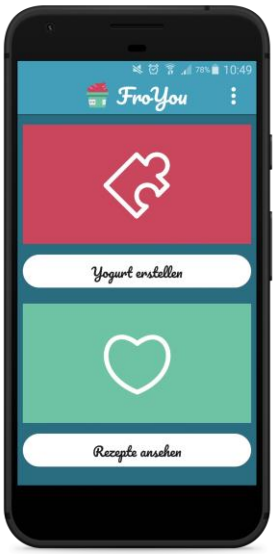

Abbildung 3. Main Activity

Order Choose Pricing

Hier kann ein Nutzer die Größe seines Joghurt auswählen. Im Geschäft „Süße Ecke“ richtet sich der Preis nach der Anzahl von Hauptzutaten und nicht nach Saucen oder Toppings. Die Größe bestimmt also nur wie viele Hauptzutaten der Bestellung hinzugefügt werden können. Eine Hauptzutat ist entweder Joghurt oder Softeis.

Abbildung 4. Order Choose Pricing Activity

Order Process

In dieser Activity kann ein Nutzer nach dem Baukasten-Prinzip einen Joghurt zusammenstellen. Er hat die Auswahl zwischen den Hauptzutaten, Toppings und Saucen. Elemente in der Liste können bewegt, verändert und wieder entfernt werden. Die Reihenfolge entscheidet später über die Schichtung der Zutaten im Joghurt. Der Preis verändert sich dynamisch an Hand der Anzahl der Hauptzutaten. Der genaue interne Aufbau dieser Activity wird im Abschnitt „Architektur: In der App - Model-View-Controller-Pattern des Order Process“ ausführlich beschrieben.

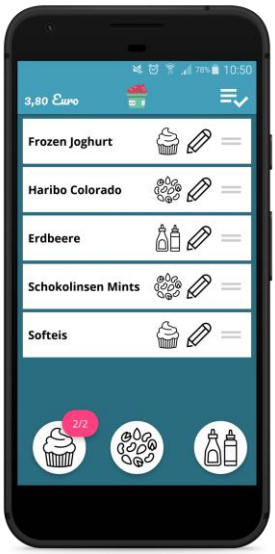

Abbildung 5. Order Process Activity

Order Finalize
Auf diese Activity gelangt der Nutzer, nachdem er seine Kreation abgeschlossen hat. Die fertige Zusammenstellung der Zutaten wird aufgelistet und es wird die Möglichkeit gegeben einen QR code zu generieren oder das Rezept zu teilen.

QR-Code Generator
Diese Activity erreicht man über einen Button von der Order Finalize Activity. Mit der Liste der Zutaten und dem Preis des Frozen Yogurts wird ein QR-Code generiert, der diese Informationen als Plain Text aufweist. Somit kann ein Mitarbeiter den Code mit einem beliebigen QR-Code Scanner vom Gerät des Kunden aus der Entfernung scannen. Die Aufnahme der Bestellung kann also unkompliziert und schnell an den Mitarbeiter zur Bearbeitung weitergegeben werden.

Order Share
Die Order Share Activity bietet die Möglichkeit ein Foto vom gekauften Frozen Yogurt zu machen, es mit Namen und Beschreibung auszustatten und ihn dann zusammen mit der Liste der Zutaten zu veröffentlichen.

Abbildung 6. Order Share Activity

Recipe Gallery
In dieser Activity werden alle veröffentlichten Rezepte in einer scrollbaren Liste angezeigt. Hier findet man nur den Namen der Kreation und das Foto. Eines dieser Rezept kann ausgewählt werden um in die Detailansicht zu gelangen.

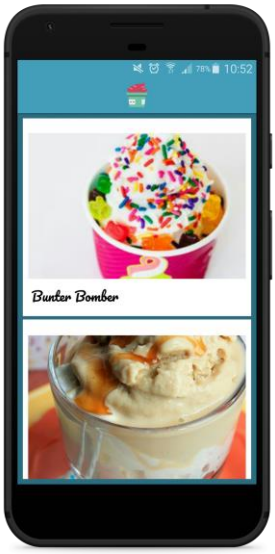

Abbildung 7. Recipe Gallery Activity

Recipe Detail
Auf der Rezept-Detailseite findet man alle Informationen, die im Order Share Schritt veröffentlicht wurden. Zudem kann auch hier, analog zur Order Finalize Activity, direkt ein QR-Code von der Kreation erstellt werden, um ihn zu bestellen.

Menu Settings
Hier kann ein User Auswählen ob er/sie Push-Notifikation erhalten möchte oder nicht.

Sprache
Ursprünglich war es geplant die Umstellung der Sprache in dieser Activity zu ermöglichen. Dies stellte sich allerdings als kompliziert heraus.

Deswegen bestimmen wir die Sprache nun anhand der Systemsprache des Endgeräts. Alle in der App verwendeten Texte sind als Strings in Englisch und Deutsch übersetzt und können nach Belieben aus zwei unterschiedlichen XML Dateien ausgelesen werden. Hierbei wurde der Standard zur Benennung von Ressource-Files mit Sprachcodes genutzt.

Menu Imprint
Das Impressum ist eine Möglichkeit auf unsere Datenschutzerklärung zu kommen. Außerdem Listen wir hier die Autoren der Icons auf, die wir für unsere App verwendet haben.

Menu Information
Hier wird in einem kurzen Text erklärt wie FroYou entstanden ist und was die Grundfunktionalitäten sind.

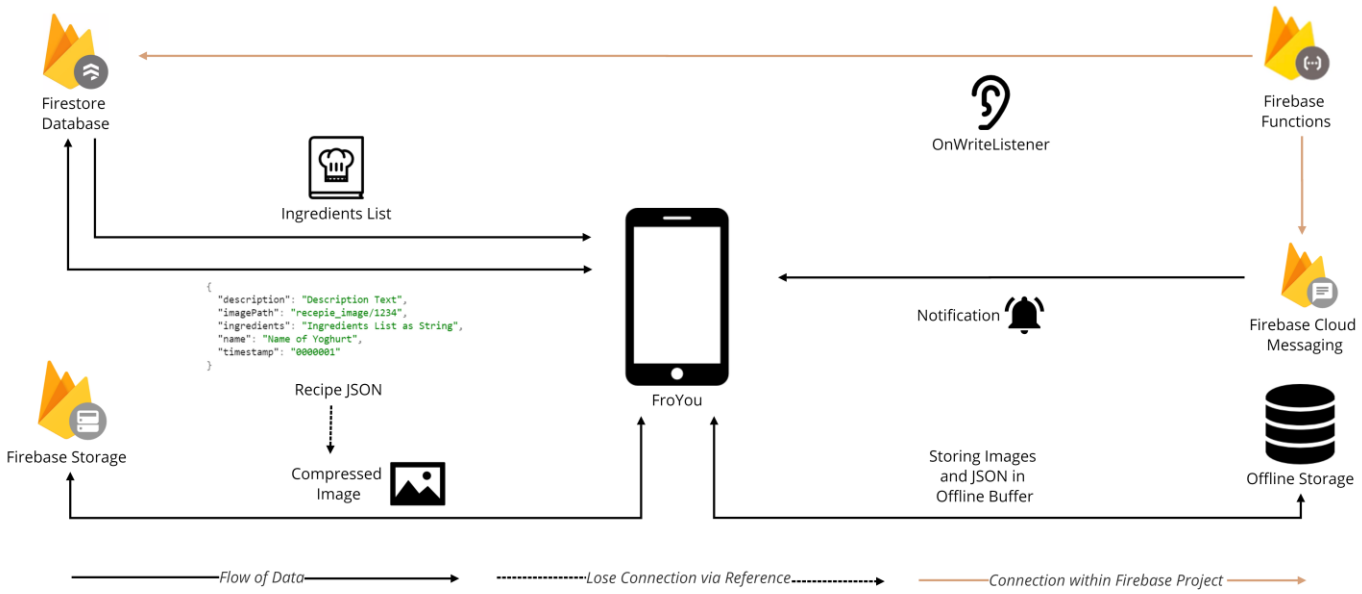

Abbildung 8. Grafik der Architektur außerhalb der App.

Architektur: Außerhalb der App

Eine der Hauptanforderungen des Modules war es, Kommunikation zwischen Usern der App zu realisieren. Um dies umzusetzen, haben wir Googles Firebase Services verwendet. Firebase ist ein Produkt von Google, welches drauf ausgelegt ist die Kommunikation zwischen Anwendungen über das Internet zu realisieren. Firebase besteht aus einer Vielzahl an Web Services, die alle für einen spezifischen Aspekt der Kommunikation ausgelegt sind. Darauf, wie und warum wir diese Service verwendet haben, werden wir im Folgenden eingehen.

Aufbau eines auf Firebase basierendem Systems

Um Firebase nutzen zu können muss als erstes ein „Firebase Projekt“ angelegt werden. Dieses Projekt kann dann mit beliebig vielen von Firebase unterstützen Anwendungen gekoppelt werden (zum Beispiel Android Apps, Node.js Servern, uvm.). Die einzelnen Firebase Services können entweder online in der Firebase Konsole bearbeitet werden, oder von mit dem Projekt gekoppelten Systemen gesteuert werden. Firebase Services können mit gekoppelte Systeme entweder über einen generierten Token oder über eine Publish-Subscribe Service kommunizieren.

Firebase Service	Funktion
Firestore	JSON basierte Online Datenbank, optimiert zum Speichern von textuellen Daten.
Storage	Binär codierte Online Datenbank, Ermöglicht das Speichern von Dateien jegliches Datentyps.
Functions	Ein Node.js Backend mit direktem Zugriff auf andere Firebase Services des Projektes. Es ermöglicht die Erstellung von Triggern, die bei bestimmten Events in anderen Services ausgelöst werden und dann eine Funktion aufrufen.
Cloud Messaging	Ein Dienst zur Realisierung von Publish-Subscribe / 1 zu n / n zu n – Messaging Services.

Tabelle 1. Kurze Erklärung der verwendeten Firebase Services

Authentifizierung und Sicherheit

Ein für uns sehr wichtiges Ziel war es, dass User unsere App nutzen können, ohne sich ein Konto zu erstellen oder sich in irgendeiner Form registrieren zu müssen. Zum einen erhöht dies die Zugänglichkeit unsere App, zum anderen müssen wir uns so nicht über das Handlen von kritischen Daten wie Passwörter und Usernamen Gedanken machen. Da zu keinem Zeitpunkt kritische Daten in der Firebase Datenbank liegen, haben wir uns dafür entschieden die Lese-Berechtigung global freizuschalten. Das Schreibrecht haben wir an die Bedingung gebunden, dass der Zugriff aus einer Instanz der FroYou App kommen muss, was verhindert, dass unsere Datenbank von außen mit fehlerhaften Daten gefüllt werden kann. An dieser Stelle ist es auch noch anzumerken, dass die Firebase API intern bei jeder Transaktion eine End-to-End Encryption einrichtet, was das Abfangen der verschickten Pakete zwecklos macht. Bei unserem Projekt gibt es keinen Anwendungsfall bei der einen spezifischen Instanz von FroYou angesprochen

werden muss, deshalb verwenden wir für die Kommunikation von unserem Server zu der App ausschließlich Publish – Subscribe. Dies ermöglicht es uns komplett ohne das Generieren von Tokens auszukommen.

Zutaten Liste

Im Gespräch mit dem Partner „Süße Ecke“ wurde klar, dass die zur Verfügung stehenden Zutaten, für die Erstellung von Joghurts, jeden Tag anders aussehen kann. Deswegen haben wir uns dafür entschieden diese Liste online in Firestore zu speichern, sodass sie auch noch nach dem Veröffentlichen der App dynamisch angepasst werden kann ohne ein App Update machen zu müssen. Eine JSON-Struktur bietet sich für das Speichern einer Zutat sehr gut an, da jede Zutat in unsere Domäne aus drei Eigenschaften besteht. Zusätzlich speichern wir die Liste auch noch auf dem Endgerät des Users, sodass die App auch offline weiterhin verwendet werden kann. Beim Aufruf der „Order Process“ Activity wird überprüft, ob Änderungen in der Datenbank vorliegen und die Offline-Liste gegebenenfalls aktualisiert werden muss. Das Laden führen wir über einen „Background Service“ aus, der bei diesem potentiell länger dauernden Prozess der UI Thread nicht blockiert. Background Services sind in Android eine Möglichkeit langwierige Prozesse im Hintergrund auszuführen. Die Firestore Android API regelt die offline Speicherung und das Erstellen von Background Services intern. Beim Ansprechen der von der API gestellten Schnittstelle, wird intern überprüft, ob die Daten neu heruntergeladen werden müssen oder sie aus dem offline Speicher gelesen werden können. Die offline gespeicherten Daten bleiben auch bei einem Neustart des Telefons vorhanden.

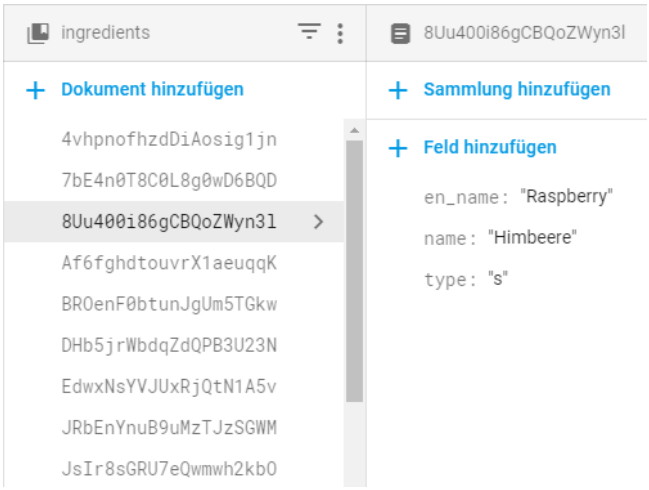

Abbildung 9. Ausschnitt aus der Firestore Konsole. Ein Einblick in den Aufbau unserer Zutaten Datenbank.

Jeder Zutat ist einer von drei Typen zugewiesen: Topping, Sauce oder Main. „Main“ sind die Hauptzutaten Softeis oder Joghurt. Beim Herunterladen der Liste werden die Zutaten in diese drei Typen aufgeteilt. Wir speichern zudem zu jeder Zutat noch den englischen Namen und entscheiden je nach Systemsprache des Gerätes bzw. Sprache der App welcher der beiden angezeigt wird.

Rezepte

Unsere App ermöglicht den Usern das Austauschen von Frozen Yogurt Rezepten. Ein Rezept besteht in unserer Domäne aus: Bild, Namen, Beschreibungstext, einer Liste von Zutaten und einem Zeitstempel. Der Ersteller des Rezeptes wird nicht genannt und bleibt somit anonym.

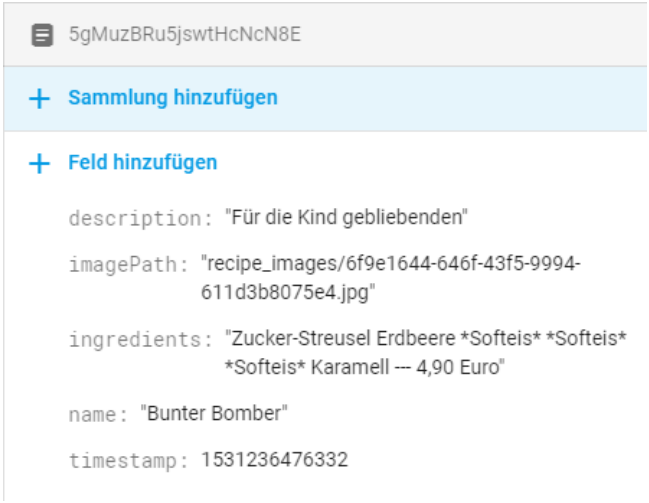

Abbildung 10. Ausschnitt aus der Firestore Konsole. Ein Einblick in den Aufbau unserer Rezept Datenbank.

Wir haben uns dafür entschieden für das Speichern der Rezepte zwei Firebase Services zu kombinieren. Wir verwenden Firestore für den textuellen Teil, da dieser Service genau für JSON Text Objekte optimiert ist und wir verwenden Firebase Storage für das Speichern der Bilder. Wir verbinden die beiden Hälften der Rezepte mittels einer URI auf das Bild, die wir in der Firestore Struktur speichern. Nach dem Laden des Textes verwenden wir diese URI um das Bild zu laden. Mehr hierzu ist in dem Abschnitt „Architektur: In der App - Firebase UI und Glide“ zu finden.

Notification

Wir haben uns zum Ziel gesetzt das Senden von Notifications asynchron von Userinteraction zu realisieren. Notifications sollten auch dann auf einem Gerät ankommen können, wenn ein anderer Nutzer eine bestimmte Aktion ausführt. Bei der Suche nach so einer Aktion sind wir auf die Firebase-Listener gestoßen. Eine Komponente von Firebase Functions, die es ermöglicht auf Änderungen, in einer der im Projekt verwendeten Datenbanken, zu reagieren.

```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

exports.pushNotification =
    functions.firestore.document('recipes/{rec}')
    .onCreate((snap, context) => {
        const recipe = snap.data();

        const payLoad = {
            notification:{
                title: 'New Recipe was Added!',
                body:   'Check out: ' + recipe.name,
                sound: "default"
            }
        };

        return admin.messaging().sendToTopic(
            "PUSH_CHANNEL",
            payLoad);
    });
```

Abbildung 11. Node.js Backend zur Versendung von Notifications bei einer Änderung in Firestore.

Das Backend wird ausgeführt, wenn ein Rezept der Datenbank hinzugefügt wird. Dann wird über Firebase Cloud Messaging eine Nachricht auf das Topic „PUSH_CHANNEL“ veröffentlicht. Beim ersten Starten der App wird dieses Topic in jeder Instanz von FroYou abonniert.

```
public void onFirstLaunch(){
    SharedPreferences prefs =
            PreferenceManager
                    .getDefaultSharedPreferences(
                            getBaseContext()
                    );

    boolean previouslyStarted =
            prefs.getBoolean( s: "FIRST_START",  b: false);

    if(!previouslyStarted) {
        SharedPreferences.Editor edit = prefs.edit();
        edit.putBoolean( s: "FIRST_START", Boolean.TRUE);
        edit.commit();
        FirebaseMessaging
                .getInstance()
                .subscribeToTopic( s: "PUSH_CHANNEL");
    }
}
```

Abbildung 12. Subscribe der Notifikation Topic beim ersten Starten der App.

Wir sichern ab, dass nur einmal abonniert wird, indem wir uns eine Variable in den Shared Preferences anlegen. Das Empfangen der Nachrichten von Firebase Messaging funktioniert über die Implementierung eines „Broadcastreceivers“. Ein Broadcastreceiver ist eine der Hauptkomponenten von Android. Sie dient zur asynchronen Reaktion auf systemweite Events. In einem Broadcastreceiver kann eine Funktion definiert werden, die bei einem bestimmten Event ausgeführt werden soll. Dies geschieht auch bei Inaktivität der App. In FroYou wird also beim Empfangen einer Nachricht das Topic „PUSH_CHANNEL“ eine Notification auf dem mobilen Gerät angezeigt.

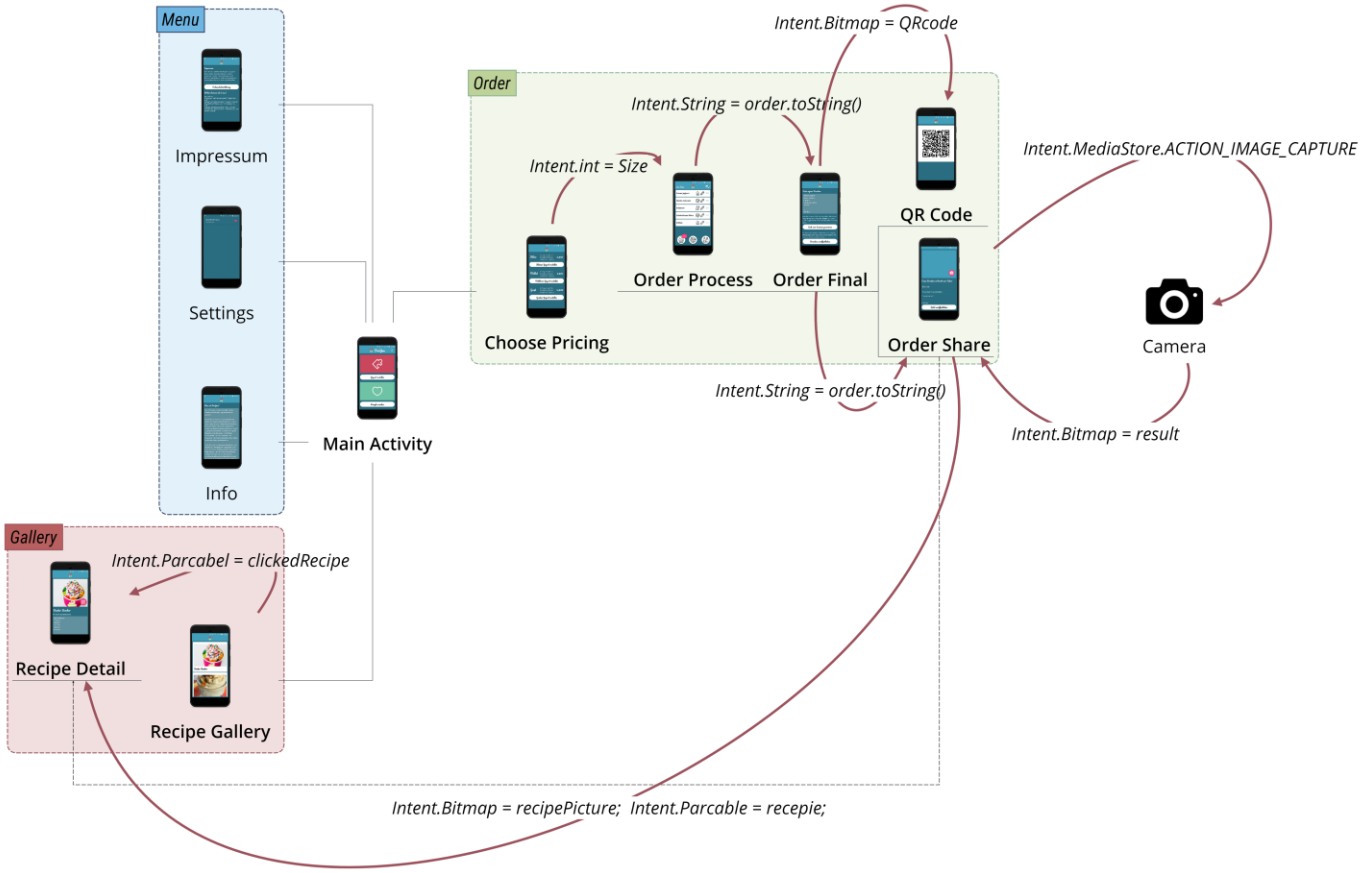

Abbildung 13. Grafik der internen Intents die zwischen Activities und Komponenten verschickt werden. Zudem sind die drei Haupt Segmente von FroYou hier zu sehen.

ARCHITEKTUR: IN DER APP

Im folgenden Abschnitt werden wir beschreiben, wie die interne Architektur von FroYou aufgebaut ist. Wir werden auf einigen Komponenten und externe Bibliotheken eingehen und beschrieben wie wir sie verwendet haben und an welchen Stellen Schwierigkeiten aufgetreten sind.

Der grundlegende Aufbau von FroYou

FroYou ist in drei Segmente aufgeteilt, die jeweils einen bestimmten Funktionskontext in der App abbilden. Die Activity Gruppe „Order“ befasst sich mit dem Erstellen und Bestellen von Frozen Yogurts. Die „Gallery“ befasst sich mit der Auflistung von veröffentlichten Rezepten und bildet somit die „soziale Komponente“ von FroYou. Unter „Menu“ fassen wir generelle Informationen über die App, sowie die Einstellungen zusammen. Somit also alle Elemente, die nicht direkt für die Nutzung der Hauptfeatures relevant sind.

Intents

In Android ist ein Intent das Übergeben einer Aufforderung von einer Komponente an eine Andere. Bei dieser Aufforderung können beliebige Parameter mitgegeben werden. Jeder Intent der innerhalb unserer App verwendet wird ist in Abbildung 13 zu sehen. An dieser Stelle wollen wir noch einmal auf die, unseres Erachtens nach, interessanteste Implementierungen von Intents in unserer App eingehen.

Nachdem ein Rezept in der „Order Share Activity“ geteilt wurde, wird ein Intent an die „Recipe Detail Activity“ geschickt. An diesen Intent wird sowohl das Bitmap des gerade erstellten Fotos, als auch die Details des Rezeptes angehängt. Beim Laden der „Detail“ Activity wird überprüft ob der Aufruf durch die „Order Share“ Activity erfolgt ist. In diesem Fall werden die Rezeptdaten nicht aus Firebase ausgelesen, sondern aus dem Intent genommen. Für den User scheint es somit als wäre das Teilen so gut wie ohne Zeitverzögerung geschehen. In Realität werden die Daten aber noch im Hintergrund hochgeladen. Im Weiteren überschreiben wir den Android-nativen „Back-Button“ in den „Gallery“ Activities um somit das Zurückspringen in die „Order Share“ Activity zu verhindern. Wir wollten somit ausschließen, dass ein Rezept mehrfach veröffentlicht werden kann.

Kamera

Für die Implementierung der Kamera verwenden wir einen Android-nativen Intent Service. Zudem implementieren wir in der Activity die „On Activity Result“ Methode, die ausgeführt wird, sobald der Service abgeschlossen ist. In dem Intent wird ein komprimiertes Bild (Thumbnail) als Bitmap beigefügt, was wir dann auslesen und verwenden

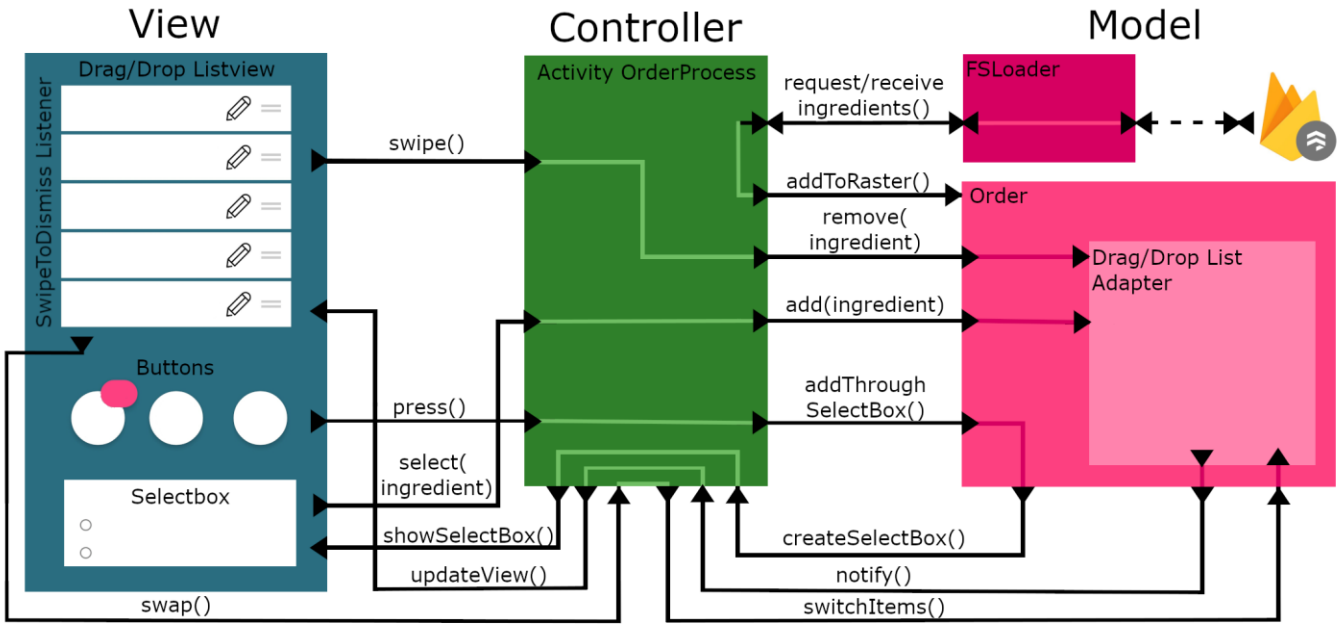

Abbildung 14. Darstellung des Model View Controller – Patterns in der „Order Process Activity“.

können. Die Komprimierung bietet sich in diesem Kontext sehr gut an, da wir uns somit keine Gedanken über das Verschicken von großen Datenpaketen in mobilen Netzen machen müssen.

Model - View - Controller - Pattern des Order Process

Im Folgenden werden wir einen Ausschnitt der Architektur genauer Betrachten und an Hand dieses Ausschnittes eine Implementierung des Model-View-Controller–Patterns darlegen.

Order

Die Order Klasse ist in unserer Architektur Teil des Models. Somit ist sie für die Anwendungslogik des Order Process verantwortlich. Im Grunde ist sie eine Art Warenkorb, dem Zutaten eines Joghurts hinzugefügt werden können und aus dem Zutaten heraus gelöscht werden können. Jede Order hat eine Größe, die bei jedem Bearbeiten von Zutaten überprüft wird. Der Orderklasse kann entweder direkt eine Zutat hinzugefügt werden oder sie generiert ein Select Menü, aus dem Zutaten ausgewählt werden können. Die Order bekommt zudem beim Erstellen der Activity eine Liste aller derzeitig verfügbaren Zutaten, die sie dann für das Erstellen der Select Menüs verwendet. Zusätzlich ist die Order Klasse ein Wrapper, der intern den Adapter unseres Listviews verwaltet und dafür sorgt, dass keine fehlerhaften Daten in dem Listview ankommen. Zudem kann aus der Order auch noch eine textuelle Repräsentation der Bestellung generiert werden, die dann über Intents an andere Activities weitergegeben wird.

Das Ziel dieser Klasse war es, die Logik der Preisberechnung, das Handlen von Ausnahmefällen beim Hinzufügen von Zutaten und das Anzeigen von Select Menüs aus der Activity Klasse auszulagern. In der Activity Klasse wird nur „add()“, „remove()“ oder „addThroughSelectBox()“ aufgerufen. Die restliche Logik und das Hinzufügen der Zutaten in den Listview regelt die Order Klasse.

Wir haben uns dagegen entschieden die Order Klasse „parcelable“ zu machen, was bedeuten würde, dass wir eine Order bei einem Intent mitgeben könnten. Da sie intern sehr viel Anwendungslogik verwaltet, die lediglich im "Order Process“ benötigt wird, haben wir darauf verzichtet.

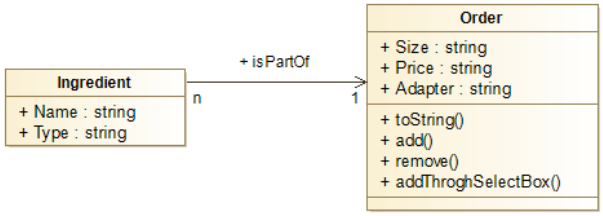

Abbildung 15. Klassen Diagramm: Ingredient und Order.

Das Hinzufügen einer Zutat

Wir verwenden für das Hinzufügen von Zutaten Android-native „Floating Action Buttons“. Diese Buttons haben eine grafische Repräsentation im View, die mit einer logischen Repräsentation im Controller gekoppelt ist. Wenn ein Button auf dem Bildschirm gedrückt wird, wird das vom View registriert. Diese Information wird an den Controller geschickt, der dann auf der Order „addThroughSelectBox()“ aufruft. Die Order erstellt eine TextBox aus den verfügbaren Zutaten des entsprechenden Typens und schickt diese über den Controller an den View zurück. Der User wählt eine Zutat aus und diese wird dann

über den Controller an die Order gegeben und hinzugefügt. Die Order fügt die Zutat dem Adapter hinzu, der dann den Listview aktualisiert.

Drag and Drop List

Eine „Darg and Drop List“ ist ein Konzept für eine Liste mit direkter Benutzerinteraktion. Die Reihenfolge von Elementen (in unserem Fall Zutaten) in einer Liste kann durch längeres gedrückt halten und bewegen von Elementen verändert werden.

Dieses Konzept bietet sich in unserer App sehr für das Erstellen von Joghurts an, da wir es dem User ermöglichen wollten die Reihenfolge seiner Zutaten sehr einfach und intuitiv verändern zu können. Für die Implementierung dieser Features haben wir in weiten Stücken eine Implementierung von Takaiwa [1] verwendet. In neueren Versionen von Android wird für das Erstellen eines Drag and Drop Recycler Views eine native Hilfsklasse angeboten. Wir haben unsere Entwicklung jedoch auf der Minimum SDK 21 begonnen und wollten die Anforderungen mitten im Entwicklungsprozess nicht weiter erhöhen. Deswegen haben wir uns dafür entschieden, diese Implementierung zu nutzen. Takaiwas Implementierung war von allen Bibliotheken, die diese SDK unterstützen, am besten dokumentiert und zeigte im Vergleich die beste Performance. Jedoch sind wir bei der Umsetzung auf mehrere Probleme gestoßen. Die originale Version unterstütze nur eine statische Größe, also mussten wir die Möglichkeit Objekte hinzuzufügen selbst implementieren. Dies stellte sich als nicht trivial heraus. Einige Stellen von Takaiwas Implementierung verwenden eine Hashmap zur Adressierung von Listen-Objekten. Eine Hashmap ist ein Standard Datentyp in Java. Es handelt sich um eine Sammlung von Daten, die aus einem Key und aus einem Value bestehen. Man kann über einen Key aus der Hashmap einen zugewiesen Value auslesen. Takaiwas Hashmap verwendet die Objekt-Referenzen der Listeneinträge als Key. Wenn nun zum Beispiel zwei Mal Hauptzutat hinzugefügt wurde, überschreiben sich diese beide Referenzen in der Hashmap und konnten somit von Teilen des Codes nicht mehr adressiert werden. In anderen Teilen des Codes, in denen die Hashmap nicht verwendet wurde, trat dieser Fehler allerdings nicht auf. Somit wurden die Elemente weiterhin richtig angezeigt, aber konnten nicht bewegt werden. Rückblickend war die Lösung dieses Problems nicht weiter schwierig. Beim Hinzufügen eines Objektes in die Liste haben wir anstelle einer Referenz, eine geklonte Instanz des Listen Eintrages hinzugefügt und dies löste das Problem. Diese Lösung ausfindig zu machen hat sehr viel Arbeit in Anspruch genommen.

Im Grunde besteht die Drag and Drop List aus zwei Komponenten: dem „Listview“ und dem „Adapter“.

Der Listview regelt die Anzeige der Elemente auf dem Bildschirm. Zudem reagiert der Listview auf bestimmte Touch Events. Wenn ein Listen Element länger gedrückt wird, ist es in dem Listview nicht mehr zusehen und der User bewegt ein grafisches Element über den Screen, was genau so aussieht wie der Eintrag. Der Listview reagiert auf dieses grafische Element, in dem es an gegebenen stellen scheinbare Lücken bildet, in Realität ist die Reihenfolge der Einträge aber weiterhin gleich. Erst beim Loslassen des grafischen Objektes wird ein „Swap Event“ an den Adapter geschickt, der dann die Liste neu ordnet.

Der Adapter hat intern eine Liste von allen Elementen der eigentlichen Liste. Zusätzlich wird hier geregelt, wie die einzelnen Listen „Fragmente“ angezeigt und mit Daten befüllt werden. Ein Fragment ist in Android ein kleines wiederverwendbares grafisches Element. An dieser Stelle verwenden wir es für die Elemente in unserer Liste. Beim Hinzufügen und Entfernen von Elementen sorgen wir im Adapter dafür das sowohl die interne Liste als auch die Hashmap korrekt bleibt.

Swipe to Dismiss

„Swipe to Dismiss“ ist ein weitverbreitetes Konzept für Listen auf mobilen Endgeräten. Das Entfernen von Einträgen aus einer Liste geschieht, indem der Listeneintrag horizontal aus dem Bildschirm gewischt wird.

Wir fanden dieses Konzept sehr ansprechend, da es eine sehr einfache Möglichkeit bietet Objekte zu löschen, ohne einen zusätzlichen Knopf zu verwenden. Um das Feature Swipe to Dismiss zu realisieren, verwenden wir eine Bibliothek von Ramees [2]. Diese Bibliothek war wesentlich einfacher einzubauen als der Drag and Drop - Listview. Rammes Bibliothek stellt einen Listener zu Verfügung, der eine beliebige Liste angefügt werden kann. Der Listener implementiert ein Interface in dem dann definiert wird, was passieren soll, wenn eine Listen-Objekt zur Seite des Bildschirmes bewegt wird.

In unserem Fall senden wir bei dem Wischen eines Elementes eine Aufforderung über den Controller an die Order, die dann den Eintrag aus dem Adapter löscht.

FS Ingredients Loader

Dies ist eine von uns selbst entwickelte Klasse. Sie kann verwendet werden um asynchron Daten von Firestore auszulesen. Sie implementiert ein Interface, welches nach dem Laden der vollständigen Liste aufgerufen wird. Zudem castet sie beim Runterladen der Daten, die Texte direkt in eine „Ingredients Objekt“. Dies bedeutet, dass für jede Zutat in der Datenbank ein Objekt der Klasse Ingredient angelegt wird.

Der Loader wird beim Start der Activity aktiviert. Während des Ladens wird ein Ladebalken angezeigt. In diesem Zeitraum sind die Buttons blockiert und der User muss einige Sekunden warten. Wenn der Loader fertig ist und alle Zutaten aus der Datenbank ausgelesen hat, schickt er die fertige Liste aller Ingredients Objekte über den Controller an die Order. Der Ladebalken verschwindet und die Buttons werden wieder frei gegeben.

Der Loader interagiert direkt mit der Firebase API und wie zuvor erwähnt wird an dieser Stelle von der API intern entschieden, ob die Daten runtergeladen werden müssen oder aus dem Offline-Buffer gelesen werden.

Im Loader wird auch entschieden ob die englischen oder deutschen Zutatennamen geladen werden sollen.

```
@Override
void dowithEachDocument(QueryDocumentSnapshot document) {

    if( Locale.getDefault().getDisplayLanguage().equals(
            Locale.ENGLISH.getDisplayLanguage())){

        Ingredient ingredient =  new Ingredient(
                (String) document.get("en_name"),
                ((String) document.get("type")).charAt(0));

        result.add(ingredient);

    }else {
        Ingredient ingredient =  new Ingredient(
                (String) document.get("name"),
                ((String) document.get("type")).charAt(0));

        result.add(ingredient);
    }


}
```

Abbildung 16. FS Ingredients Loader. Das Casten der Firestore Daten in Ingredients Objekte.

Firebase UI und Glide

Für die Darstellung der „Recipe Gallery“ Activity haben wir zwei Komponente aus der Firebase UI [3] Bibliothek verwendet. Die Bibliothek enthält eine Reihe an Grafik Elementen, die für den Gebrauch mit Firebase Services optimiert wurden. Für das Anzeigen der Liste verwenden wir einen „Firestore Recycler Adapter“. Dieser Adapter erleichtert das Laden von Daten aus Firestore in einen „Recycler View“. Ein Recycler View ist im Grunde ein „Listview“ der performanter ist, die im Android Framework nachträglich hinzugefügt wurde. Der Adapter kann ohne Probleme mit dem Android-nativen Recycler View gekoppelt werden. Beim Erstellen des Firestore Adapter wird ein Query Objekt verwendet, welches angibt von wo die Daten aus Firestore ausgelesen werden sollen.

```
Query query = FirebaseFirestore.getInstance()
        .collection( collectionPath: "recipes")
        .orderBy( field: "timestamp", Query.Direction.DESCENDING)
        .limit(50);

FirestoreRecyclerOptions<Recipe> options =
        new FirestoreRecyclerOptions.Builder<Recipe>()
        .setQuery(query, Recipe.class)
        .build();
```

Abbildung 17. Setzen der Query Anweisungen bei einem Firestore Adapter.

Zudem wird bei diesem Adapter festgelegt in welche Klasse die ausgelesenen Daten gecastet werden sollen. Wenn die Namen der Parameter des Konstruktors mit den Namen der Felder in der Datenbank übereinstimmen geschieht der Cast automatisch.

Nachdem wir die textuellen Daten der Rezepte im Adapter geladen haben, verwenden wir „Gilde“: Glide ist eine Bibliothek, die von Firebase UI implementiert wird, um die Bilddaten aus dem Storage zu laden.

```
if(recipe.getImagePath() != null){
    StorageReference recImageRef =
            storageRef.child(recipe.getImagePath());

    Glide.with(context)
            .load(recImageRef)
            .into(image);
}
```

Abbildung 18. Glide Implementierung im Recipe View Holder. Rezept Bilddaten werden in den Recycler View geladen.

Bei der Firebase UI Implementierung von Glide muss lediglich die Referenz des Bildes von Firebase Storage und der Imageview als Ziel angegeben werden. Glide startet einen Service, der die Daten im Hintergrund herunterlädt.

Shared Prefrences

„Shared Preferences“ in Android ist im Prinzip eine Sammlung aus Key-Value Paaren, die auf einem Endgerät gespeichert werden und auch noch nach dem Neustart des Mobiltelefons vorhanden sind.

Wir verwenden Shared Preferences für unsere „Menu Settings“ Activity. Hier verwenden wir Android-native „Preference Fragments“, die eine einfache Möglichkeit bieten Daten in die Key–Value Hashmap zu schreiben. Ein Nutzer kann hier entscheiden, ob er Notifications erhalten will. Beim Herunterladen der Nachrichten von Firebase Cloudmessaging lesen wir dann diesen Wert wieder aus und entscheiden so ob die Notification angezeigt werden soll oder der Nutzer sie deaktiviert hat.

QR-Codes

Sowohl auf der „Order Finalize“-, als auch auf der „Recipe Detail“ Activity kann über einen Button ein QR-Code generiert werden. Hier nutzen wir Teile der Bibliothek zxing [4], die verschiedene Funktionen zum Erstellen von Barcodes beinhaltet. In den beiden aufrufenden Activities wird ein MultiFormatWriter genutzt, dem man die Zutatenliste in Form eines Strings mitgibt und zusätzlich die Art des Barcodes (in unserem Fall QR-Code) und die Größe angibt. Daraus wird dann eine Bitmap mittels BarcodeEncoder generiert, welche mittels eines Intents an eine Activity übergeben wird, die ausschließlich für die Anzeige des QR-Codes zuständig ist. In dem Code befindet sich nur der Text der Zutaten und der Preis, wodurch er mit jeder beliebigen Barcode Scanner App scannbar ist.

FAZIT

Reflexion

Was haben wir gelernt?
Interessant war für uns der Unterschied zwischen der bisher bekannten Java-Programmierung, die wir in den bisherigen Semestern kennenlernen konnten, und der Programmierung für Android. Auch der Unterschied in der Implementierung von Interfaces im Gegensatz zum (Responsive) Webdesign war ein interessanter Aspekt.

Was war uns besonders wichtig?
Uns war bei der Umsetzung der App besonders wichtig, dass die Vorbereitung zur Bestellung von Frozen Yogurts möglichst unkompliziert und schnell zu tätigen ist. Aus diesem Grund haben wir schon in den Anfängen viel Zeit in die Konzeption gesteckt, damit wir später möglichst wenig Zeit mit Korrekturen verbringen müssen. Außerdem war ein Schwerpunkt, dass die App auch auf älteren und verschieden großen Android Geräten gut nutzbar ist.

Was lief gut?
Sehr zufrieden sind wir damit, dass wir all unsere Basis-Ideen aus der Konzeption, die wir zu Beginn des Projektes aufgebaut hatten, umsetzen konnten. Sowohl die Einbindung eines Sensors, als auch der soziale Teil der App konnte nach unserer Vorstellung umgesetzt werden und bringt gute Features für FroYou mit sich. Auch das Interface konnte so umgesetzt werden, wie im Prototypen angedacht.

Was lief nicht so gut?
Die Implementierung des Drag and Drop war sehr zeitaufwändig. Auch die Probleme mit der Verstellbarkeit der Sprache in den Settings konnten wir nicht umsetzen.

Welches Feedback haben wir erhalten?
Von vielen Seiten haben wir gutes Feedback für die App erhalten. Vor allem aus der Poster Session konnten wir viel Positives mitnehmen. Diese Session hat uns aber auch geholfen Dinge noch ein weiteres Mal zu überdenken und zu optimieren, da Personen zum ersten Mal die App genutzt haben und einen anderen Blickwinkel hatten.

Offene Punkte

Ideen aus Vorlesung und Feedback
Aus der Vorlesung und unserer Poster Session haben wir Ideen zur Erweiterung und Verbesserung sammeln können. Eine Idee wäre die Übermittelung von Bestellungen über NFC an den Mitarbeiter. Somit könnte man einen weiteren Sensor einbinden.

Außerdem könnte man die QR-Code Funktionalität dahingehend erweitern, dass man eine Art „pay per scan“ umsetzt, mit der wir als Entwickler der App beteiligt werden, wenn ein Frozen Yogurt über uns kreiert und gekauft wurde. Auch die Einbindung von Werbung wäre zur Monetarisierung möglich.

Ein größerer Umbau würde es jedoch mit sich bringen, wenn wir die App für verschiedene Frozen Yogurt Läden verfügbar machen. Von einem Mitarbeiter des „moxd lab“ kam zudem die Idee, dass man das Baukastenprinzip selbst für viele weitere Zusammenstellungen nutzen könnte (z.B. Subway Bestellungen).

Ideen aus Gespräch mit „Süße Ecke“
Ende Juli haben wir dem Inhaber und der Filialleiterin unseres Kooperationspartners unsere fertige App vorgestellt. In diesem Gespräch konnten wir weitere Verbesserungsmöglichkeit ausfindig machen.

Eine Möglichkeit wäre es, seine Rezepte nicht nur öffentlich speichern, sondern auch lokal auf dem Gerät festhalten zu können. Dadurch wäre es möglich auch für mehrere Personen Bestellungen vorzubereiten (z.B. für Familien).

Als weitere Idee könnte man ein kleines Webinterface entwerfen, auf der die „Süße Ecke“ die vorhandenen Zutaten einpflegen und warten kann, sodass sie keinen Zugriff auf die komplette Firebase benötigen.

REFERENZEN

1. Takeaway, DraggableListViewSample
   Github Repossitory (2016)
   geladen am 04. August 2018 von
   https://github.com/takaiwa/DraggableListViewSample/
2. Ramees, SwipeDismissListViewTouchListener
   Blog Eintrag (Datum Unbekannt)
   geladen am 04. August 2018 von
   http://codesfor.in/android-swipe-to-delete-listview/
3. Firebase, Firebase UI-Android
   Github Repository (2018)
   geladen am 04. August 2018 von
   https://github.com/firebase/FirebaseUI-Android/
4. Zxing, Barcode Scanning Library
   Github Repository (2018)
   geladen am 09. August 2018 von
   https://github.com/zxing/zxing

Team Mitglied	Projekt Bericht	App	Poster
Sebastian Faust	54%	50%	20%
Julian Schoemaker	44%	47%	80%
Dario Giuseppe Lazzara	2%	3%	0%

Tabelle 2. Arbeitsmatrix