

# U N I K A S S E L V E R S I T Ä T

CODECAMP CONTEXT AWARENESS 2

FACHGEBIET COMMUNICATION TECHNOLOGY

---

## Dokumentation

---

*Autoren:*

Julian

Kevin

Steffen

*Betreuer:*

Lars

2. September 2022

# Inhaltsverzeichnis

Abkürzungsverzeichnis	ii
<b>1 Einleitung</b>	<b>1</b>
<b>2 Grundlegende Architektur</b>	<b>2</b>
<b>3 Bibliotheken</b>	<b>4</b>
3.1 Accompanist . . . . .	4
3.2 Coil . . . . .	4
3.3 Dagger/Hilt . . . . .	4
3.4 FuzzyWuzzy . . . . .	6
3.5 Jetpack Compose . . . . .	6
3.6 Moshi . . . . .	7
3.7 MPAndroidChart . . . . .	7
3.8 OKHttp . . . . .	8
3.9 Retrofit . . . . .	9
3.10 Room . . . . .	10
3.11 ZXing . . . . .	11
<b>4 Features</b>	<b>12</b>
4.1 Authentifizierung . . . . .	12
4.2 Applikations- und Navigationsleiste . . . . .	13
4.3 Getränkeübersicht . . . . .	14
4.4 Transaktionsverlauf und Statistiken . . . . .	17
4.5 Nutzerübersicht . . . . .	18
4.6 Profilübersicht . . . . .	19
4.7 NFC-Scan . . . . .	21
4.8 Bekannte Bugs . . . . .	21
<b>5 Anleitung</b>	<b>21</b>

# Abkürzungsverzeichnis

**API** Application Programming Interface

**DAO** Data Access Object

**DI** Dependency Injection

**HTTP** Hypertext Transfer Protocol

**JSON** JavaScript Object Notation

**JWT** JSON Web Token

**MVVM** Model-view-viewmodel

**REST** Representational State Transfer

**UI** User Interface

**XML** Extensible Markup Language

**SDK** Software Development Kit

# 1 Einleitung

Im Rahmen des Moduls Java CodeCamp Context Awareness 2 sollte eine Android Applikation zur Getränkebuchung in Kotlin implementiert werden. Die Getränkebuchung fand dabei nur virtuell über die Anbindung eines entsprechend vorgegebenen Server-Backends statt. Als Anforderungen wurden folgende Funktionalitäten definiert, die die Applikation bereitstellen sollte:

- Nutzer Login mit JSON Web Token (JWT) als Authentifizierung mit automatischem Token-Refresh und der sicheren Speicherung der Zugangsdaten.
- Separater Adminbereich zur Verwaltung anderer Nutzer und Getränke und dem Hinzufügen von Guthaben für einzelne Nutzer
- Übersicht aller kaufbaren Getränke, sowie der Möglichkeit, diese zu kaufen.
- Visuelle Darstellung in Form von Diagrammen verschiedener Statistiken wie beispielsweise der Anzahl verschiedener Items, die ein Nutzer gekauft hat oder dem zeitlichen Verlauf des Guthabens.
- Errungenschaften für verschiedene Meilensteine wie 100 gekauften Getränken.
- Individualisierung wie beispielsweise ein Empfehlungssystem, abhängig von den Gewohnheiten und bisherigen Bestellungen des Nutzers.

Diese Dokumentation soll eine Zusammenfassung über die komplette Projektphase von Idee bis Umsetzung bereitstellen und dem Leser einen Überblick über wesentliche Aspekte der App, wie beispielsweise verwendete Technologien oder Entwurfsansätze geben. Abschnitt 2 befasst sich dabei mit der grundlegenden Konzeption der Applikation und erklärt die dabei verwendeten Design-Patterns und Modelle. Anschließend daran gibt Abschnitt 3 einen Überblick über die verwendeten externen Bibliotheken beschreibt deren Funktionalität und erklärt ihre konkrete Anwendung innerhalb der App. Da für die Implementierung und den Entwurf der User Interface (UI) ein relativ neues Framework namens Jetpack Compose verwendet wurde, geht Unterabschnitt 3.5 im Detail auf die Vor- und Nachteile gegenüber des bisherigen Ansatzes ein. In Abschnitt 4 werden die einzelnen Funktionalitäten, welche die Applikation bereitstellt, sowohl jene, die vorausgesetzt waren, als auch eigene Ideen, hinsichtlich Konzept und Umsetzung näher erklärt, sowie deren Nutzung beispielhaft veranschaulicht. Mit Abschnitt 5 gibt diese Dokumentation interessierten Entwicklern einen Leitfaden an die Hand, in dem die Voraussetzungen und Anforderungen erklärt und gezeigt werden, die es benötigt, um die Applikation importieren, kompilieren und ausführen zu können.

## 2 Grundlegende Architektur

Entwurfsmuster vereinfachen bereits seit vielen Jahren die Komplexität von Code. Dabei gibt es viele verschiedene Ansätze. Mit ihrem Buch *Design Patterns: Elements of Reusable Object-Oriented Software* führten Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides 23 klassische Entwurfsmuster der Softwareentwicklung ein [1]. Im Laufe der Jahre sind noch weitere dazugekommen. Die vorliegende Anwendung ist nach dem Entwurfsmuster Model-view-viewmodel (MVVM) aufgebaut und entsprechend strukturiert[2]. Eine grobe Übersicht über dieses Konzept bietet Abbildung 1.

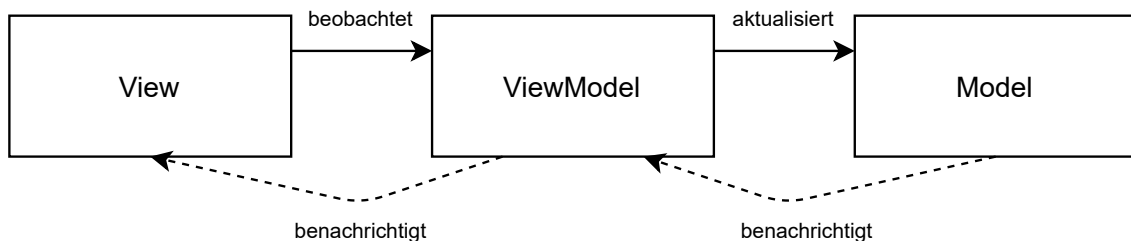


Abbildung 1: MVVM-Pattern

Dabei gibt es wie der Name schon vermuten lässt drei Hauptbestandteile. Die View, das ViewModel und das Model. Die View beschreibt alle grafischen Benutzeroberflächenelemente. In einer typischen Anwendung wären das alle Elemente, welche für Nutzende sichtbar sind. Die View beobachtet dabei permanent das ViewModel. Dieses enthält die Logik hinter der UI und dient als Bindeglied zwischen View und Model. Ein wichtiges Merkmal ist, dass ViewModel keine Informationen über die View enthält. Über Databinding können dann Daten zwischen dem View und ViewModel ausgetauscht werden. Der letzte zu erwähnende Bestandteil ist die Modelschicht. Die Modelschicht ist eine Schicht, welche die nötigen Daten der Applikation enthält. Die Daten werden im MVVM-Pattern vom ViewModel aktualisiert.

Das MVVM-Entwurfsmuster wurde in der vorliegenden Anwendung durch das Repository-Entwurfsmuster erweitert[3, s. 322]. Das Repository kapselt die unterschiedlichen Wege an Daten zu kommen oder diese zu organisieren nochmal ab. Es besitzt eine direkte Referenz zum lokalen Modell, aber auch eine zum remote gespeicherten Modell. Beim lokalen Modell werden die Daten in einer Datenbank gespeichert. Das Remotemodell greift dabei auf einen Webservice zu, welcher die Daten zur Verfügung stellt. Dies könnte zum Beispiel ein Webservice sein, welche mithilfe einer Programmierschnittstelle (eng. Application Programming Interface (API)) Bilder zur Verfügung stellt. Das Repository-Entwurfsmuster ermöglicht es über Schnittstellen und Methoden, wie zum Beispiel `getById(id: String)`, an Daten zu kommen.

Neben diesen verschiedenen Entwurfsmustern wurden auch weniger komplexe Entwurfsmuster verwendet. Das wichtigste dabei ist das Return-Early-Entwurfsmuster, welches tiefgeschachtelten Code vereinfacht und die Gesamtkomplexität des Codes reduziert[4]. Die weniger komplexen Entwurfsmuster werden an dieser Stelle allerdings nicht näher erläutert. Im Projekt wurde allerdings noch ein weiteres Entwurfsmuster verwendet, welches expliziter Erklärung bedarf. Dieses Entwurfsmuster nennt sich Dependency Injection (DI)[5].

Bei DI werden Abhängigkeiten zwischen Objekten erst zur Laufzeit hergestellt. Dabei werden Abhängigkeiten von einem zentralen System verwaltet. Dieses zentrale System nennt sich Injektor, weil es Abhängigkeiten injiziert. Vereinfacht ausgedrückt gibt es bei diesem System einen Client, der einen Service nutzen möchte und einen Injektor, der diesen Service bereitstellt. Der Client verwendet dabei ein Interface, das konkret vom Service implementiert wird. Dies soll anhand eines Beispiels deutlicher werden. Die Klasse **Shop** besitzt eine Beziehung zur Schnittstelle **IBeverageService**. Mithilfe von DI kann nun festgelegt werden, dass die Klasse **CoffeeService** standardmäßig für Abhängigkeiten der Schnittstelle **IBeverageService** verwendet werden soll. Wichtig ist hierbei zu erwähnen, dass **CoffeeService** die Schnittstelle **IBeverageService** implementiert. Die größten Vorteile von DI sind die zentrale Initialisierung der Dienste und die Möglichkeit diese, auch während der Laufzeit, auszutauschen. Somit können Implementierungen auch durch gemockte<sup>1</sup> Implementierungen, im Rahmen von Unit Tests, ausgetauscht werden.

Mit einem ähnlichen Hintergrund ist im Laufe des Projekts eine weitere Idee entstanden. Jedes ViewModel besitzt ein Interface, welches sicherstellt, dass bestimmte Felder oder Methoden implementiert werden. Dabei gibt es für jede View zwei verschiedene ViewModels. Das ursprüngliche ViewModel wird verwendet, um Daten und Funktionalitäten während der Laufzeit bereitzustellen. Das neue ViewModel wird verwendet, um während des Entwicklungsprozesses Previews in der Viewschicht zu generieren. Aus diesem Grund besitzt es den Namen **ViewModelPreview** und implementiert ebenfalls dasselbe Interface. Der Vorteil ist nun, dass Daten in diesem ViewModel zu Vorschauzwecken modelliert werden können. Diese Mockdaten können außerdem auch zum testen verwendet werden.

Allgemein lässt sich sagen, dass mehrere von Android bereitgestellte Klassen verwendet werden. Die hauptsächlichen Klassen, welche die Architektur ausmachen sind **ComponentActivity** und **ViewModel**. Die **ComponentActivity**-Klasse besitzt einen Lebenszyklus, welcher primär dafür gedacht ist die erstellte View zusammen mit den zugehörigen ViewModels einzubinden. Durch bestimmte Funktionen lassen sich diese Aktivitäten wechseln und es können neue Interaktionsmöglichkeiten präsentiert werden. Die Funktionalitäten von ViewModels wurden bereits ausreichend in diesem Kapitel erklärt, weswegen diese hier nicht näher erläutert werden.

---

<sup>1</sup>Mock-Daten sind Platzhalter für echte Daten

## 3 Bibliotheken

In diesem Kapitel sind die im Projekt verwendeten Bibliotheken, lexikographisch sortiert, aufgelistet und beschrieben, sowie Verweise zur vollständigen Dokumentation der einzelnen Bibliotheken angegeben. Die Bibliotheken werden mithilfe des Build-Management-Automatisierungstools Gradle<sup>2</sup> verwaltet. Alle hier aufgelisteten Bibliotheken und deren exakten Versionsnummern sind in der `build.gradle` des Projekts zu finden.

### 3.1 Accompanist

Accompanist<sup>3</sup> ist eine Sammlung von Bibliotheken, um Features von Jetpack Compose (siehe Unterabschnitt 3.5) zu erweitern. Für dieses Projekt wurde dabei die Bibliothek System UI Controller<sup>4</sup> verwendet, um die Farben der Systemleiste anzupassen, also der Leiste, in der beispielsweise der Zurückbutton oder Empfangsstärke angezeigt werden. Dabei übernimmt die Bibliothek auch die Anbindung verschiedener API Versionen, damit dies nicht mehr manuell vom Entwickler übernommen werden muss [6].

### 3.2 Coil

Coil<sup>5</sup> steht für Coroutine Image Loader und ist eine Bibliothek, um Bilder zu laden. In diesem Projekt wird Coil verwendet, um die Profilbilder der Nutzer aus der Datenbank zu laden und anzuzeigen. Coil bietet dabei den Vorteil einer verbesserten Performance durch Caching oder Downsampling. Dabei nutzt Coil sogenannte `ImageLoader` Klassen, die `ImageRequests` ausführen. Innerhalb dieser `ImageRequests` lässt sich dann definieren, von wo der `ImageLoader` die Bilder wie laden soll, also bspw. aus der Datenbank als Bitmap oder von der API als ByteArray [7].

### 3.3 Dagger/Hilt

Hilt<sup>6</sup> ist eine DI-Bibliothek, die auf Dagger<sup>7</sup> aufbaut. DI wurde dabei bereits ausführlich in Abschnitt 2 erläutert. Dagger ermöglicht es, mithilfe von Annotationen DI zu erleichtern und Code zu generieren. Hilt ist im Grunde genommen eine Erweiterung, die neue Annotationen mit sich bringt. Diese sind speziell für die Entwicklung von Android Applikationen ausgelegt.

---

<sup>2</sup><https://gradle.org/>

<sup>3</sup><https://github.com/google/accompanist>

<sup>4</sup><https://github.com/google/accompanist/tree/main/systemuicontroller>

<sup>5</sup><https://github.com/coil-kt/coil>

<sup>6</sup><https://dagger.dev/hilt/>

<sup>7</sup><https://dagger.dev/>

Die Codegenerierung wird im Projekt aktiviert, sofern `@HiltAndroidApp` an eine Application-Klasse annotiert ist. Damit Felder mithilfe der `@Inject` annotiert werden können, muss die Annotation `@AndroidEntryPoint` vorher verwendet werden. Diese kann nur an fünf verschiedenen Stellen definiert werden: Activity, Fragment, View, Service und BroadcastReceiver. ViewModels besitzen eine eigene Annotation `@HiltViewModel`, die es ermöglicht Werte im Konstruktor zu injizieren. Diese werden von einem `@Module` zur Verfügung gestellt. `@InstallIn` definiert den Geltungsbereich dieses Moduls. Wenn dieser auf `SingletonComponent` gesetzt wird, ist das Modul für die ganze Applikation sichtbar. Das geschieht mithilfe der `@Provides`-Annotation.

```
1  @Module
2  @InstallIn(SingletonComponent::class)
3  class AppModule {
4      @Provides
5      @Singleton
6      fun provideCoffeeRepository(): CoffeeRepository = CoffeeRepositoryImpl()
7  }
```

*Listing 1: Annotierte Klasse AppModule*

Listing 1 zeigt beispielhaft ein Modul, welches ein `CoffeeRepository` bereitstellt. Ebenfalls in diesem Listing zu sehen ist die Annotation `@Singleton`. Diese Annotation definiert den Geltungsbereich für diese Methode. In diesem Fall ist dieser, genau wie beim `AppModule`, für die ganze Applikation.

```
1  @HiltViewModel
2  class CoffeeViewModel @Inject constructor(
3      private val coffeeRepository: CoffeeRepository
4  ) : ViewModel() {
5      private fun printCoffeeTypes(){
6          println(coffeeRepository.getCoffeeTypes())
7      }
8  }
```

*Listing 2: Annotierte Klasse CoffeeViewModel*

Listing 2 zeigt das ViewModel `CoffeeViewModel`. Dank der Annotation `@Inject` vor dem Konstruktor, kann das `CoffeeRepository` zur Laufzeit injiziert werden. Die dafür nötigen Abhängigkeiten werden dann mithilfe von DI durch das `AppModule` zur Verfügung gestellt. Im Speziellen bedeutet dies, dass die Funktion aus Listing 1 das `CoffeeRepository` für den Konstruktor in Listing 2 zur Verfügung stellt. Auf



diese Art und Weise können in Listing 2 zur Laufzeit Funktionen des injizierten Konstruktorparameters verwendet werden.

### 3.4 FuzzyWuzzy

Die FuzzyWuzzy-Bibliothek<sup>8</sup> berechnet das Ähnlichkeitsverhältnis zwischen zwei gegebenen Strings. Dabei verwendet der Algorithmus die Levenshtein-Distanz. Die Levenshtein-Distanz bezeichnet die minimale Anzahl, um die erste Zeichenkette in die zweite Zeichenkette zu verwandeln. Folgenderweise ist die Levenshtein-Distanz zwischen dem Wort Kaffee und Bier fünf.

Kaffee  $\xrightarrow{K \rightarrow B}$  Baffee  $\xrightarrow{a \rightarrow i}$  Biffee  $\xrightarrow{f}$  Biffee  $\xrightarrow{f}$  Biee  $\xrightarrow{e \rightarrow r}$  Bier

Diese Distanz wird in der Bibliothek dann weiterhin verwendet, um ein Ähnlichkeitsverhältnis auszurechnen. Basierend auf unterschiedlichen Parametern können unterschiedliche Verhältnisse ausgerechnet werden, wobei ein Verhältnis immer zwischen 0% und 100% liegen kann. Dabei steht 0% für überhaupt keine Übereinstimmung von Buchstaben und 100% bei exakt gleicher Übereinstimmung aller Buchstaben.

Dieses Bibliothek wird im Projekt verwendet, um Sucheingaben für Nutzende zu erleichtern. Trotz Tippfehlern können weiterhin relevante Suchergebnisse vorgeschlagen werden. Dabei ist wichtig einzustellen, wie die Mindestübereinstimmung des Suchbegriffes mit Vorschlägen einer Liste ist. Andernfalls werden möglicherweise zu viele unrelevante Inhalte vorgeschlagen.

### 3.5 Jetpack Compose

Jetpack Compose<sup>9</sup> ist ein Framework von Google, um die Erstellung von UI von Android Applikationen zu vereinfachen und die Menge an Code zu reduzieren. Im Gegensatz zum bisherigen imperativen Ansatz zum UI-Design mit Extensible Markup Language (XML) Layouts, basiert Compose auf einem deklarativen Ansatz. Das bedeutet, dass beim bisherigen imperativen Ansatz das “Wie“ im Mittelpunkt steht. UI Elemente werden entworfen und anschließend gerendert und angezeigt.

Dem gegenüber steht der deklarative Ansatz von Compose, der sich auf das “Was“ konzentriert und es Entwicklern ermöglicht, die UI abhängig von den Daten, die angezeigt werden sollen, zu erzeugen[8]. Vorteil hiervon ist, dass die gesamte Applikation in einer Sprache geschrieben werden kann, im Falle von Compose Kotlin, was gleichzeitig in deutlicher reduzierter Größe der Applikation sowie ihrer Build-Dauer resultiert.

---

<sup>8</sup><https://github.com/xdrop/fuzzywuzzy>

<sup>9</sup><https://developer.android.com/jetpack/compose/documentation>

Ein UI Element unter Verwendung von Compose basiert dabei auf einer Funktion, die definiert, welchen Inhalt das jeweilige Element anzeigen soll. Diese Funktionen werden mit der `@Composable` Annotation deklariert und im folgenden als Composable bezeichnet. Ein Composable besitzt eine initiale Zusammensetzung bzw. Erscheinung, die beim erstmaligen Aufrufen des jeweiligen Fensters berechnet wird. Diese Zusammensetzungen können Zustände, verwalten, wobei nach jeder Zustandsänderung die Zusammensetzung des Composables neu berechnet wird. Ein Zustand innerhalb einer Applikation ist dabei jeglicher Wert, der nicht statisch ist und sich somit über die Zeit ändern kann. Da jedes Composable nichts weiteres als eine Funktion mit der `@Composable` Annotation ist, und diese nur innerhalb anderer Composables aufgerufen werden können, lassen sich redundante Komponentendeklarationen vermeiden und eine bessere Struktur und Übersichtlichkeit herstellen.

Die Entscheidung für Compose war zum Einen in der Motivation zur Verwendung neuester Technologien begründet. Zum Anderen bietet Compose aber auch einige messbare Verbesserungen, wie weniger Code und somit besserer Wartbarkeit und beschleunigter Entwicklung, aber auch eine Verringerung von Speicherplatz [9]

## 3.6 Moshi

Moshi<sup>10</sup> bietet die Möglichkeit unter Android Daten im JavaScript Object Notation (JSON) Format zu entsprechenden Java oder Kotlin Klassen umzuwandeln und umgekehrt Klassen im JSON Format zu serialisieren. Für das Projekt wird Version **1.13.0** der Bibliothek verwendet. Damit Daten von Moshi umgewandelt werden können müssen zunächst Klassen implementiert werden, die als Modell dienen. Den Variablen dieses Modells lassen sich dann über verschiedene Annotationen die entsprechenden Felder innerhalb der JSON Datei zuweisen. Folgende Beispieldekларation veranschaulicht dies:

```
data class Foo(@field: Json(name="jsonField") val id: String)
```

Hierbei wird der Wert des JSON Feldes mit dem Namen **jsonField** dem Wert der String-Variable **id** zugeordnet. Das Projekt nutzt die Bibliothek, um eine einfache und standardisierte Methodik für die Umwandlung zwischen den Modellierungsarten zu erhalten. Einer weiterer Vorteil der Bibliothek ist die Anbindung an Retrofit (siehe Unterabschnitt 3.9) durch Bereitstellung eines entsprechenden Adapters, um die Verarbeitung von API Antworten weiter zu automatisieren [10].

## 3.7 MPAndroidChart

MPAndroidChart<sup>11</sup> ist eine Bibliothek, welche die Erstellung, Verwaltung und das Einbinden von verschiedenen Diagrammen in Android ermöglicht bzw. vereinfacht.

---

<sup>10</sup><https://github.com/square/moshi>

<sup>11</sup><https://github.com/PhilJay/MPAndroidChart>

Die Bibliothek stellt zahlreiche Klassen und Methoden zur Verfügung um verschiedene Arten von Graphen wie beispielsweise Kuchen- oder Balkendiagramme zu erstellen und Daten zu visualisieren. Je nach Diagrammtyp werden dabei unterschiedliche Datenstrukturen, wie beispielsweise Arrays oder Listen von Arrays, für die zugrunde liegenden Datensätze unterstützt, insbesondere Zeitreihen.

Da die Applikation verschiedene Statistiken wie die Anzahl an Bestellungen über einen bestimmten Zeitraum oder die fünf meistbestellten Getränke eines Nutzers anzeigen soll, wurde hierfür auf die MPAndroidChart Bibliothek zurückgegriffen. Grund dafür sind zum Einen die einfache Handhabung und Dokumentation, aber auch die gute Unterstützung zur Visualisierung von Daten in Zeitreihen. Zum Anderen aber auch der Mangel an Alternativen, da diese entweder nicht den selben Umfang besitzen oder, zum Zeitpunkt der Implementierung, Kotlin nicht unterstützen.

### 3.8 OKHttp

OkHttp<sup>12</sup> stellt eine Implementierung eines Hypertext Transfer Protocol (HTTP)-Clients für Android zur Verfügung. Weiterhin übernimmt die Bibliothek die Verwaltung und Steuerung der Verbindung zwischen Server und Client, wie beispielsweise das erneute Herstellen einer Verbindung nach Verbindungsabbrüchen oder Anfragenwiederholungen. Um die Bibliothek zu verwenden, benötigt die Applikation verständlicherweise die Erlaubnis von Android, eine Internetverbindung herzustellen. Wie in Unterabschnitt 3.9 beschrieben, nutzt das Projekt OkHttp als Client und Netzwerkschicht für Retrofit. Das Bereit- und Erstellen eines OkHttpClient mit Hilfe von DI geschieht wie folgt:

---

<sup>12</sup><https://square.github.io/okhttp/>

```

1      @Singleton
2      @Provides
3      fun provideOkHttpClient(
4          loggingInterceptor: HttpLoggingInterceptor,
5          bearerInterceptor: BearerInterceptor,
6      ): OkHttpClient {
7          return OkHttpClient.Builder()
8              .addInterceptor(loggingInterceptor)
9              .addInterceptor(bearerInterceptor)
10             .callTimeout(10, TimeUnit.SECONDS)
11             .connectTimeout(10, TimeUnit.SECONDS)
12             .readTimeout(10, TimeUnit.SECONDS)
13             .writeTimeout(10, TimeUnit.SECONDS)
14             .build()
15     }
16

```

*Listing 3: Funktion zum Bereitstellen des OkHttpClient*

Die Methode `provideOkHttpClient` in Listing 3 erhält dabei als Übergabeparameter zwei Interceptors. Ein Interceptor kann als Mechanismus angesehen werden, mit dem HTTP Aufrufe beispielsweise abgefangen und in der Konsole zu Debugging Zwecken ausgegeben werden, wie im Fall des `loggingInterceptor`, oder, wie im Falle des `bearerInterceptor`, modifiziert und erst aus dann ausgeführt werden können.

In den Zeilen 8 bis 13 in Listing 3 finden zusätzliche Konfiguration, wie das Hinzufügen von den übergebenen Interceptors oder das Definieren von Verbindungstimeouts, statt. Schlussendlich wird mit Zeile 14 der eigentliche Client erstellt und per DI zur Verfügung gestellt[11].

### 3.9 Retrofit

Retrofit<sup>13</sup> ist eine typsichere HTTPClient Bibliothek für Android, die es der Applikation vereinfacht, Representational State Transfer (REST)-APIs zu konsumieren. Retrofit baut dabei standardmäßig auf der OkHttpClient Bibliothek auf, die in Unterabschnitt 3.8 erklärt wird und als Netzwerkschicht agiert. Ähnlich wie Moshi aus Unterabschnitt 3.6 nutzt auch Retrofit Annotationen, um Methoden zu indizieren, die bei den jeweiligen HTTP-Anfragen ausgeführt werden. Der nachfolgende Befehl veranschaulicht dies.

```
@GET("/api/users") suspend fun getUsers(): Response<List<UserResponse>>
```

---

<sup>13</sup><https://github.com/square/retrofit>

Hierbei wird die Funktion `getUsers()` deklariert und mit der Annotation `@GET("/api/users")` versehen. Somit wird während der Ausführung der Methode eine HTTP GET-Anfrage an den in den Klammern definierten Endpunkt versendet. In diesem Fall an `"/api/users"`. An obigem Befehl lässt sich gut die Verbindung der Bibliotheken Retrofit, OKHttp aus Unterabschnitt 3.8 und Moshi aus Unterabschnitt 3.6 veranschaulichen. Wie bereits erwähnt, nutzt Retrofit OKHttp als zugrundeliegende Netzwerkschicht, um die eigentliche HTTP-Anfrage durchzuführen. Die Antworten dieser Anfragen werden dann mittels eines vordefinierten Converters, in diesem Fall Moshi, serialisiert. Der Datentyp `UserResponse` sieht dabei wie folgt aus:

```
1 @JsonClass(generateAdapter = true)
2 data class UserResponse(
3     @Json(name = "id")
4     val id: String,
5     ...
6 )
```

*Listing 4: Auszug aus dem annotierten Datenmodell UserResponse*

In Zeile 1 von Listing 4 wird mit der `JsonClass(generateAdapter=true)` Annotation definiert, dass diese Klasse aus einer JSON Antwort serialisiert werden kann. Moshi bietet hierbei die Möglichkeit Adapter automatisch zu generieren. Zeile 3 und 4 definieren die jeweiligen Felder/Variablen die zwischen der JSON Datei und der Klasse verlinkt und in Unterabschnitt 3.6 im Detail erklärt werden. Die Definition des zugrunde liegenden HTTP Clients für Retrofit geschieht durch folgende Funktion

```
Retrofit.Builder().client(okHttpClient).build()
```

die eine Retrofit Instanz zurück gibt und dabei den in den Klammern von `client()` definierten HTTP-Client nutzt, der in Unterabschnitt 3.8 im Detail erklärt wird [12].

## 3.10 Room

Room <sup>14</sup> ist eine Bibliothek zur Bereitstellung bzw. Abstrahierung von SQLite Datenbanken und Teil von Jetpack. Die Bibliothek besteht dabei aus drei zentralen Komponenten:

1. Die **Datenbankklasse**, in der die zu verwendende Datenbank definiert und deklariert wird. Diese Klasse dient weiterhin als Zugriffspunkt auf den persistenten Datenspeicher der Applikation.

---

<sup>14</sup><https://developer.android.com/jetpack/androidx/releases/room>

2. Das **Datenobjekt** dient als Repräsentation der einzelnen Tabellen innerhalb der Datenbank.
3. Den **Datenzugriffsobjekte** Data Access Object (DAO), die als Schnittstelle zwischen der Datenbank und dem Rest der Applikation dienen und Methoden für Lese- und Schreibzugriffe bereitstellen.

Damit die Applikation auf Daten innerhalb der Datenbank zugreifen kann, wird zunächst eine DAO-Instanz von der Datenbankklasse angefordert. Innerhalb der DAO sind in SQL Syntax Methoden definiert, die den Lese- und Schreibzugriff regeln und definieren. Mit Hilfe dieser Instanz kann die Applikation damit dann über die vordefinierten Methoden auf die Datenbank zugreifen. Diese grundlegende Beziehung zwischen den einzelnen Komponenten wird in Abbildung 2 illustriert [13].

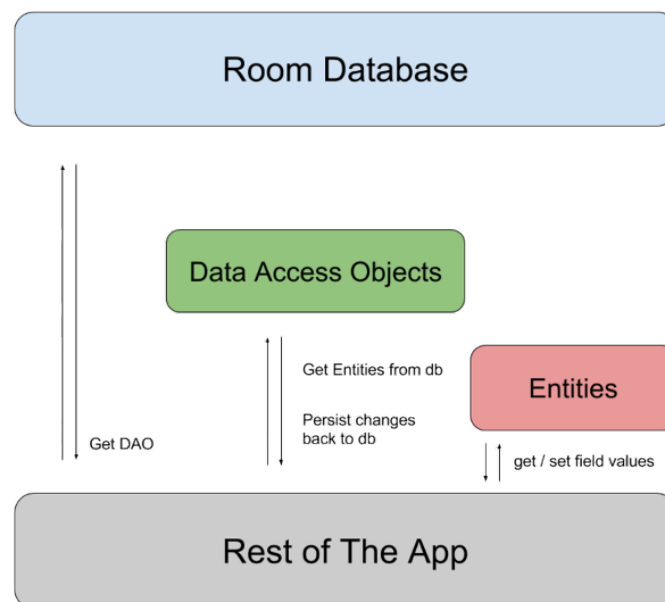


Abbildung 2: Architektur der Room Bibliothek [13, Figure 1]

Das Projekt nutzt die Bibliothek um Daten, die von API erhalten wurden, zu persistieren und somit eine grundlegende Funktionalität auch bei Verlust der Internetverbindung zu ermöglichen.

### 3.11 ZXing

ZXing<sup>15</sup> ist eine Bibliothek, um Barcodes zu verarbeiten. Zusätzlich zu ZXing wird im Projekt eine Bibliothek verwendet, welche nicht von den offiziellen Projektbeteiligten entwickelt wurde. Diese nennt sich ZXing Android Embedded<sup>16</sup> und ist eine

<sup>15</sup><https://github.com/zxing/zxing>

<sup>16</sup><https://github.com/journeyapps/zxing-android-embedded>

Barcode-Scanning Bibliothek für Android, welche ZXing zum decoden verwendet. Im Projekt werden diese Bibliothek verwendet, um QR-Codes einzuscannen oder zu generieren.

## 4 Features

In den folgenden Unterkapiteln werden die implementierten Funktionalitäten der Applikation beschrieben. Dabei werden schrittweise alle implementierten Funktionalitäten und Sichten der nutzenden Person beschrieben. Screenshots aus der Anwendung sind absichtlich zugeschnitten, um in der Dokumentation die Features besser beschreiben zu können. Für eine komplette Übersicht über alle Inhalte sollte die Applikation, mithilfe der Anleitung aus Abschnitt 5, ausgeführt werden.

### 4.1 Authentifizierung

Sobald die Applikation gestartet wird erscheint ein SplashScreen mit dem Apptitel, einem Logo und der Möglichkeit auf Login oder Sign Up zu klicken. Nachdem auf einer der beiden Knöpfe gedrückt wird erscheint die jeweilige Ansicht die Daten anzugeben.

(a) Login

(b) Registration

Abbildung 3: Authentifizierungsmöglichkeiten

Nutzende haben die Möglichkeit sich mit Hilfe der grafischen Oberfläche aus Abbildung 3a anzumelden. Dabei fallen bereits mehrere Details, die typischerweise auch in anderen Anwendungen aufzufinden sind, auf. Eine dieser Funktionalitäten ist das Textfeld, das neben dem eingegebenen Text Informationen über den einzugebenden Inhalt enthält. Somit kann es nicht passieren, dass Nutzende anfangen Inhalte einzutippen, vergessen was genau benötigt wurde und dies nicht mehr auffinden können.

Eine weitere Funktionalität ist das Umschalten der Passwort Anzeige zwischen Klartext und unkenntlichem Text, durch Klicken auf das Augapfelsymbol. Ein weiteres Feature ist die Remember me Funktion, durch die beim Start der Applikation Nutzernamen und Passwort automatisch geladen werden.

Durch Klicken auf Sign Up können Nutzende dann in die Übersicht aus Abbildung 3b gelangen. Dort werden zunächst alle bereits eingegebenen Daten übernommen. Das passiert ebenfalls, wenn man von der Anmeldung zum Login springt. Die Übersicht bietet ähnliche Funktionalitäten wie die Übersicht aus Abbildung 3a. Eine weitere Funktionalität ist, dass Nutzende zur Sicherheit ihr Passwort doppelt eingeben müssen. Nachdem Nutzende sich durch bestätigen auf den Knopf Sign Up registrieren gelangen sie zurück in die Oberfläche aus Abbildung 3a. Allerdings muss das Passwort dort zur Sicherheit erneut eingegeben werden. Nach Eingabe und bestätigen durch den Loginknopf werden Nutzende zur Hauptanwendung geleitet.

Dabei wird nach erstem erfolgreichem Login der JWT gespeichert und verwendet. Da dieser Token nach einiger Zeit ablaufen kann muss sichergestellt werden, dass Nutzende authentifiziert bleiben. Wie bereits in Unterabschnitt 3.8 kurz angeschnitten wird ein BearerInterceptor verwendet, um diese Authentifizierung mittels einem Interceptor sicherzustellen. Sobald der Server auf eine Anfrage mit dem Code 401 Unauthorized antwortet, wird ein erneuter Login durchgeführt. Anschließend wird die vorher nicht autorisierte Anfrage erneut geschickt.

## 4.2 Applikations- und Navigationsleiste

Innerhalb jeder der nachfolgenden Übersichten aus den nachfolgenden Unterkapiteln existieren eine Applikations- und eine Navigationsleiste.



Abbildung 4: Applikationsleiste

Die Applikationsleiste ist in Abbildung 4 zu sehen und gibt Aufschluss über den aktuellen Screen. Ebenfalls ist der aktuelle Kontostand der nutzenden Person angezeigt. Die Leiste selbst befindet sich am oberen Ende einer Übersicht.



Abbildung 5: Navigationsleiste



Im Vergleich dazu befindet sich die Navigationsleiste in Abbildung 5 immer am unteren Rand einer Übersicht. Dabei gibt es vier Navigationselemente, die jeweils einen Namen und ein Icon besitzen. Aktive Elemente werden, wie in der Abbildung 5, weiß hervorgehoben. Durch klicken auf das weiße Element kann die aktuelle Übersicht neu geladen werden.

### 4.3 Getränkeübersicht

In dem Getränkeübersicht Tab befinden sich die Übersicht der verfügbaren Getränke. Diese lassen sich durch die Suchleiste (Abbildung 6), die sich unter der Applikationsleiste befindet, mithilfe von FuzzySearch filtern.

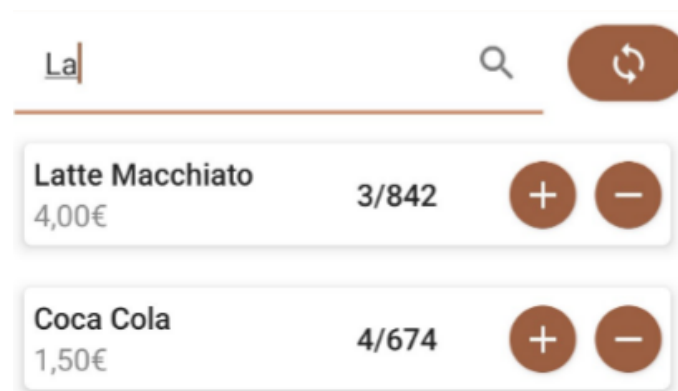


Abbildung 6: Suchleiste Getränkeübersicht

Zu jedem verfügbaren Getränk wird in der Liste, wie ebenso in Abbildung 6 zu sehen, der Name, der Preis, die Anzahl der Getränke im Warenkorb und wie viele auf Lager sind angezeigt. Außerdem lässt sich durch den Plusknopf das Getränk zum Warenkorb hinzufügen und durch den Minusknopf wieder entfernen. Dabei wird aber bei jedem drücken berücksichtigt, dass nicht weniger als 0 und nicht mehr, als die Anzahl des Getränks im Lager, im Einkaufswagen sein kann. Ebenso erscheint ein Toast, wenn nicht genügend Geld auf dem Konto ist um das aktuelle Getränk hinzuzufügen.

Recommended:



Abbildung 7: Empfohlenes Getränk

Abbildung 7 zeigt das Getränk, welches nach einem bestimmten Algorithmus vorgeschlagen wird. Basierend auf den vorherigen Einkäufen werden von der eingeloggten

Person alle Getränke gefiltert. Diese Getränke sind ebenfalls nach der aktuellen Uhrzeit  $\pm$  drei Stunden gefiltert. Ebenfalls muss ein Artikel mindestens drei Mal gekauft werden bevor er als relevant genug eingestuft wird.

Oberhalb der Navigationsleiste wird ein `ExtendedFloatingActionButton` angezeigt Abbildung 8a. Dieser zeigt immer die aktuelle Summe der Kosten der Getränke im Einkaufswagen an. Wenn man ihn drückt, öffnet sich eine Übersicht der Produkte im Einkaufswagen, deren Anzahl und der Gesamtpreis Abbildung 8b. Am unterem Ende der Übersicht besteht die Möglichkeit diese durch das Drücken von "Cancel" zu schließen, oder den Kauf durch das drücken von "Buy" abzuschließen.

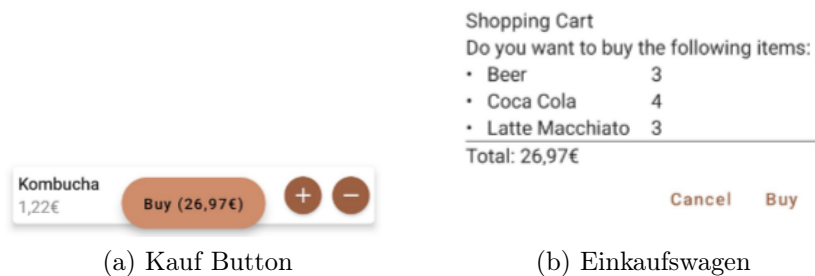


Abbildung 8: Getränke kaufen

Wenn der Benutzer Administratorrechte besitzt, wird neben der Suchleiste ein weiteres Symbol angezeigt, mit dem es möglich ist, zu der Adminansicht der Getränkeübersicht zu gelangen (Abbildung 9a).

Um ein bereits existierendes Getränk zu bearbeiten, muss auf dessen Eintrag in der Liste gedrückt werden. So öffnet sich ein neues Fenster, in dem sich, wie in Abbildung 9b zu sehen, die aktuellen Werte (ID, Name, Preis, Menge) des Getränks bearbeiten lassen. Zum Abschluss der Bearbeitung wird am unteren Rand des Fensters der "Ok" Button gedrückt. Alternativ befindet sich hier auch der "Delete" Button zum Löschen des Produkts, oder der "Cancel" Button um das Bearbeiten abubrechen und das Fenster zu schließen.

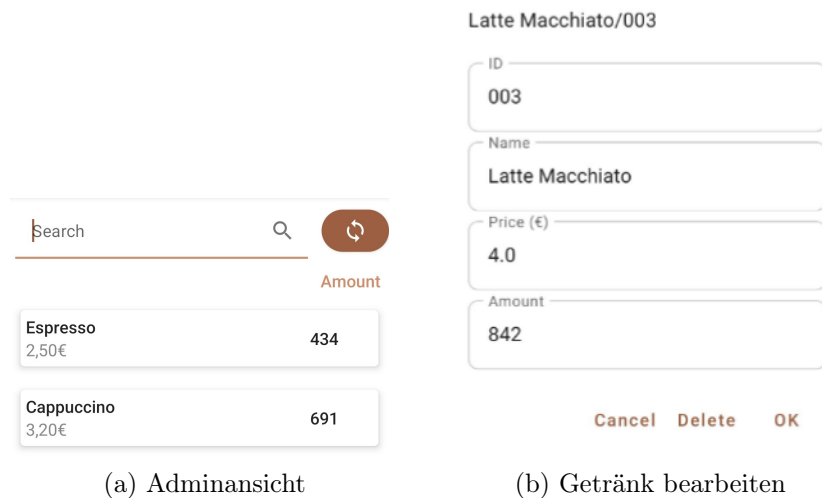


Abbildung 9: Getränke in Adminansicht bearbeiten

Der Kauf Button wird in der Adminansicht durch einen Button zum Hinzufügen neuer Getränke ersetzt (Abbildung 10a). Wenn dieser gedrückt wird, erscheint ein Fenster (Abbildung 10b), in der Name, Preis und die Anzahl des neuen Getränks angegeben wird. Am unteren Rand kann mit "Cancel" das Fenster wieder geschlossen, und mit "Ok" die Erstellung abgeschlossen werden. Um anschließend wieder zur Kaufansicht zu gelangen, kann erneut der Button neben der Suchleiste gedrückt werden.

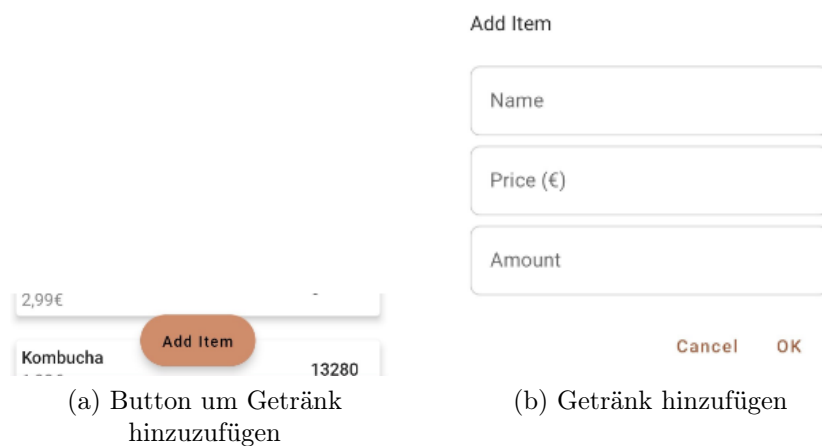
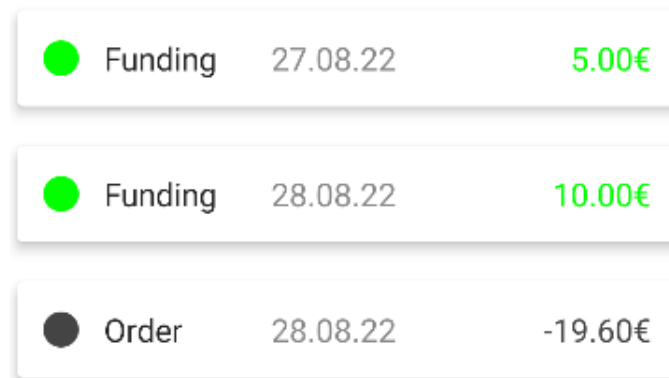


Abbildung 10: Getränke in Adminansicht bearbeiten

## 4.4 Transaktionsverlauf und Statistiken

Der Transaktionsverlauf, die Nutzerstatistiken ebenso wie ein Button zum Erzeugen eines QR-Codes um Guthaben zu verschicken, befinden sich im **History** Tab der Navigationsleiste, siehe Abbildung 5. Der Transaktionsverlauf ist dabei zum Einen in Listenform chronologisch sortiert wie in Abbildung 11 dargestellt, wobei zur Übersichtlichkeit als Zeitstempel lediglich das Datum und nicht die genaue Uhrzeit angezeigt wird. Zum Anderen auch in einer grafischen Variante in Abbildung 12. Jede Transaktion besitzt ein eigenes Kartenelement in dem Typ und Name der Transaktion angezeigt werden, im konkreten Fall also grün und **Funding** für das Aufladen von Guthaben und schwarz und **Order** für jegliche Abbuchung, ob Bestellung oder das Versenden von Guthaben. Neben dem Zeitstempel ist auch der konkrete Wert der Transaktion in den jeweiligen Farben dargestellt um eine schnelle Übersicht zu erhalten.



● Funding	27.08.22	5.00€
● Funding	28.08.22	10.00€
● Order	28.08.22	-19.60€

Abbildung 11: Transaktionsverlauf

Die Nutzerstatistiken werden ebenfalls im **History** Tab angezeigt. Im oberen Teil von Abbildung 12, werden die Top 5 der am häufigsten bestellten Getränke in Form eines Kuchendiagrammes dargestellt. Die Gesamtzahl aller Bestellungen eines jeweiligen Getränkes wird dabei innerhalb der dazugehörigen Teile des Kuchendiagramms angezeigt.

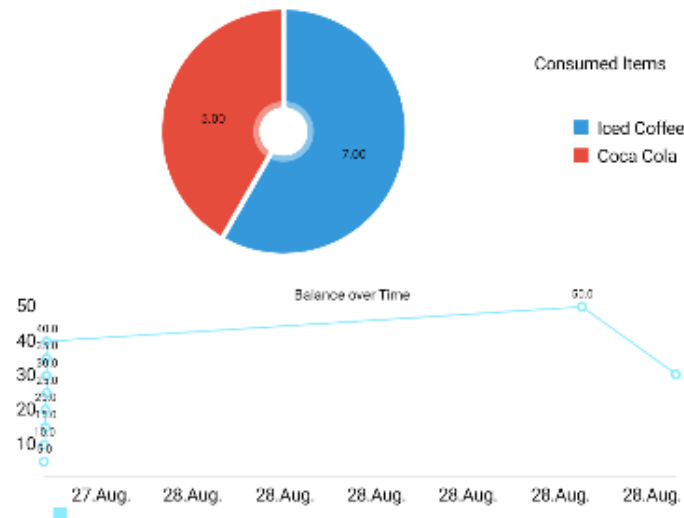


Abbildung 12: Nutzerstatistiken

Im unteren Teil von Abbildung 12 wird Transaktionsliste im zeitlichen Verlauf dargestellt, wobei sich beliebig weit in die Grafik hineinzoomen lässt. Weiterhin ist es möglich, einzelne Datenpunkte anzuklicken, um detaillierte Informationen wie den genauen Zeitstempel angezeigt zu bekommen. Die Statistiken befinden fest verankert sich im oberen Teil der Ansicht, wohingegen die Transaktionsliste aus Abbildung 11 im unteren Teil der Ansicht ist. Das bedeutet, dass beim Scrollen durch die Transaktionsliste die Statistiken immer am selben Ort bleiben.

Beim Klick auf die Möglichkeit Guthaben über einen QR-Code zu teilen, öffnet sich ein Pop-up, bei dem Guthaben versandt oder empfangen werden kann. Wenn die Möglichkeit zum Versenden ausgewählt wird, muss zunächst ein Betrag eingegeben werden und anschließend mit der Kamera ein QR-Code eingescannt werden. Der einzuscannende QR-Code wird dann bei einer anderen Person durch Klick auf die Option Geld zu empfangen generiert.

## 4.5 Nutzerübersicht

Die Nutzerübersicht stellt, wie der Name bereits vermuten lässt, alle nutzenden Personen übersichtlich in einer Liste dar. Dabei wird das Profilbild, der Name und die Möglichkeit Geld zu teilen je Person pro Zeile abgebildet. Dabei sei angemerkt, dass es aufgrund der Implementierung im Server auch möglich ist negative Beträge mit Personen zu teilen. Administratorfunktionen beinhalten, zusätzlich zu den bereits genannten Funktionalitäten, das Geld jeder einzelnen Person einzusehen und dieses manipulieren zu können. Um die Bilder darzustellen, werden vom Server pro Person der Zeitstempel für die letzte Aktualisierung des Bildes mit dem Zeitstempel in der Datenbank verglichen. Wenn der Zeitstempel vom Server aktueller ist, bedeutet dies, dass ein aktuelleres Bild vorliegt und dieses heruntergeladen werden kann.

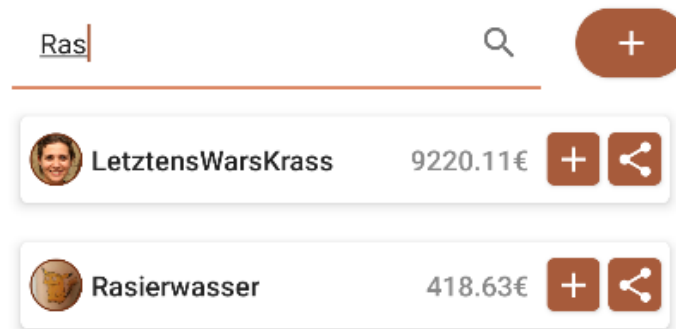



Abbildung 13: Nutzerübersicht

Die erwähnte Nutzerliste ist aus Administratorsicht in Abbildung 13 abgebildet. Zusätzlich zu der Nutzerliste gibt es, ähnlich wie in Unterabschnitt 4.3, die Möglichkeit die dargestellte Liste mithilfe von FuzzySearch zu filtern. Durch drücken auf den größeren Plusknopf neben der Suchleiste lassen sich über ein Pop-up als Administrator weitere Nutzer hinzufügen. Diese beinhalten, wie beim registrieren, einen Namen, eine ID und die zweifache Eingabe des Passworts. Allerdings ist es als Administrator möglich dem Nutzer Adminrechte zu geben. Der Dialog bzw. das Pop-up können ebenfalls abgebrochen werden, falls kein neuer Nutzeraccount erstellt werden soll.

## 4.6 Profilübersicht

Innerhalb der Profilansicht gibt es mehrere Möglichkeiten Informationen zur eingeloggten Person zu erfahren oder zu verändern. Dabei sieht das Aktualisieren der Daten ähnlich zur Registrationsansicht aus Unterabschnitt 4.1 aus.



ID  
Rasierwasser

Name  
Rasierwasser

Password 👁

Re-enter password 👁

Update profile

(a) Nutzerdaten

Logout

Achievement Overview

version 1.0

Delete account

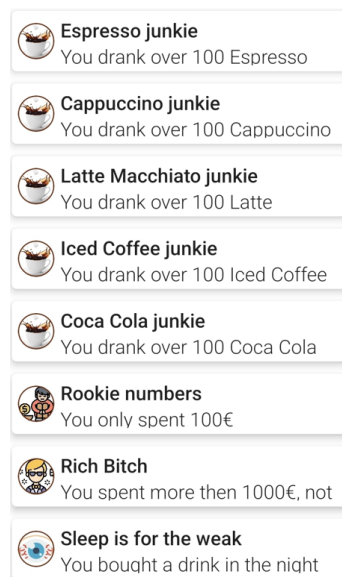
(b) Optionen in der  
Profilansicht

Abbildung 14: Profilübersicht

Abbildung 14 zeigt die komplette Profilübersicht. Wie bereits erwähnt zeigt ist die Aktualisierung der Daten in Abbildung 14a ähnlich zur Registrierung. Dabei ist der einzige Unterschied, dass das Profilbild angepasst werden kann. Dass dies möglich ist soll durch den Stift am Profilbild symbolisiert werden. Innerhalb dieser Profilübersicht gibt es ebenfalls mehrere Optionen, wie in Abbildung 14b dargestellt. Nutzende haben hier die Möglichkeit sich auszuloggen oder Achievements anzusehen.

Die erreichten Achievements werden in einem Fenster Abbildung 15 angezeigt. Diese Lassen sich durch das Kaufen von Getränken mit der App erreichen. Hierzu wird für jedes existierende oder neu hinzugefügte Getränk ein Achievement erzeugt, welches durch das Kaufen von je 100 Stück erreicht wird. Außerdem gibt es Achievements für das Ausgeben bestimmter Summen (100€, 1000€, 10000€) und wenn Nachts (0:00 - 4:00) ein Getränk gekauft wird. Ob ein neues Achievement erreicht wird, wird nach jedem Kauf geprüft. Wenn sich mit einem Account zum ersten mal eingeloggt wird, der eigentlich schon Achievements erreicht hat, werden diese beim ersten Kauf generiert.

Beim Erreichen eines Achievements wird ebenso eine Push Notification erzeugt, in der angezeigt wird, welches Achievement erreicht wurde.



*Abbildung 15: Achievement Übersicht*

Ebenfalls wird die aktuelle Version der Applikation in Abbildung 14b übersichtlich angezeigt. Für den Fall, dass Nutzer ihren Account nicht mehr verwenden wollen besteht die Möglichkeit diesen zu löschen. Die erste Überlegung ist es gewesen den Knopf dafür rot zu färben, um darauf hinzuweisen, dass dieser Knopf besonders ist und größere Konsequenzen mit sich bringen kann. Allerdings bringt der rote Knopf ungewollte Aufmerksamkeit mit sich, weswegen dieser Knopf eine weniger auffällige rote Textfarbe besitzt und extra separiert von der anderen Knopfgruppe ist[14].

## 4.7 NFC-Scan

Ein weiteres Feature der App ist das Lesen von NFC Tags. Wenn ein gelesenes NFC Tag die Id eines Getränks beinhaltet, wird das jeweilige Getränk zum Einkaufswagen hinzugefügt.

Zum Umsetzen der Funktion wurde "NfcAdapter.ReaderCallback" verwendet. Zum einen wurde sich dazu entschieden, da es mit Intents Probleme bei der Implementierung gab. Zum anderen hat es den Vorteil, dass es im Gegensatz zum Intent, nicht auf dem UI Thread, sondern auf einem separaten Thread läuft und somit die App beim Lesen des Tags nicht pausiert.

## 4.8 Bekannte Bugs

In der finalen Abgabe der Anwendung existiert ein Fehler. Beim Senden eines aktualisierten Accounts an den korrekten Endpunkt mit PUT wird der Fehlercode 400 Bad Request zurückgegeben. Dies passiert, wenn folgende JSON übermittelt wird.

```
{"id": "R", "name": "R", "password": "asdasdasd", "isAdmin": true}
```

Das interessante an diesem Fehler ist, dass das nicht optionale `isAdmin` weggelassen werden kann und derselbe Endpunkt mit folgender JSON funktioniert.

```
{"id": "R", "name": "R", "password": "asdasdasd"}
```

Der Grund dafür konnte bis zur Abgabe nicht ermittelt werden. Aus diesem Grund sorgt die Aktualisierung von Accounts aus Unterabschnitt 4.6 dafür, dass diese ebenfalls die Adminrechte verlieren.

# 5 Anleitung

Diese Anleitung setzt voraus, dass Sie bereits Android Studio installiert und eingerichtet haben. Ebenso sollten alle Updates installiert sein. Zum Ausführen der App wird Software Development Kit (SDK) 26 oder höher vorausgesetzt

1. Projekt herunterladen<sup>17</sup> oder beigefügte zip Datei verwenden.
2. Projekt in Android Studio importieren.
3. Auf Fertigstellung des Gradle Builds warten.
4. In Android Studio entweder emuliertes Gerät oder eigenes Smartphone als "running device" auswählen.
5. "Run App" klicken.
6. Profit.

---

<sup>17</sup><https://github.com/Morphclue/code-camp-2>



## Literatur

- [1] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
- [2] Josh Smith. Patterns - WPF Apps With The Model-View-ViewModel Design Pattern. *MSDN Magazine*, 24(2), 2009.
- [3] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley-Longman, 2002.
- [4] Medium - Return Early Pattern. <https://medium.com/swlh/return-early-pattern-3d18a41bba8>. Online abgerufen am 27.08.2022.
- [5] Martin Fowler - Inversion of Control Containers and the Dependency Injection pattern. <https://martinfowler.com/articles/injection.html>. Online abgerufen am 27.08.2022.
- [6] System UI Controller for Jetpack Compose. <https://google.github.io/accompanist/systemuicontroller/>. Online abgerufen am 27.08.2022.
- [7] Coil. [https://coil-kt.github.io/coil/getting\\_started/](https://coil-kt.github.io/coil/getting_started/). Online abgerufen am 27.08.2022.
- [8] Linus Muema. Declarative vs Imperative UI in Android. <https://www.section.io/engineering-education/declarative-vs-imperative-ui-android/>. Online abgerufen am 27.08.2022.
- [9] Jetpack compose — before and after. <https://medium.com/androiddevelopers/jetpack-compose-before-and-after-8b43ba0b7d4f>. Online abgerufen am 27.08.2022.
- [10] Square - Moshi. <https://github.com/square/moshi>. Online abgerufen am 27.08.2022.
- [11] Square - OkHttp. <https://square.github.io/okhttp/>. Online abgerufen am 27.08.2022.
- [12] Anupam Chugh. Retrofit Android Example Tutorial. <https://www.digitalocean.com/community/tutorials/retrofit-android-example-tutorial>. Online abgerufen am 28.08.2022.
- [13] Android Developers - Room. <https://developer.android.com/training/data-storage/room>. Online abgerufen am 27.08.2022.
- [14] UXMovement - How to Design Destructive Actions That Prevent Data Loss. <https://uxmovement.com/buttons/how-to-design-destructive-actions-that-prevent-data-loss/>. Online abgerufen am 29.08.2022.