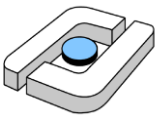


Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Schriftliche Projektarbeit zum Thema:

Entwicklung einer Java Android Applikation zur Verwaltung von Gruppenausgaben.

im Rahmen des Moduls
Programmierung 3 (MI),
des Studiengangs Informatik-Medieninformatik

Autor:	Arpad Horvath Dennis Brockmeyer
Matr.-Nr.:	1056196 1063293
E-Mail:	arpad.horvath@hs-osnab-rueck.de dennis.brockmeyer@hs-osnab-rueck.de
Dozent:	H. Plitzner

Abgabedatum: 25.07.2024 (Arpad Horvath)
Abgabedatum: 17.09.2024 (Dennis Brockmeyer)

Inhaltsverzeichnis

Abbildungsverzeichnis

Source-Code Verzeichnis

Abkürzungsverzeichnis

IDE	Integrierte Entwicklungsumgebung
JVM	Java Virtual Machine
ML	Machine Learning
NLP	Natural Language Processing
SDK	Software Development Kit
UI	User Interface / Benutzeroberfläche
UX	User Experience
ERM	Entity-Relationship Modell

1 Einleitung (Arpad)

1.1 Vorstellung des Themas

In der heutigen Zeit werden gemeinsame Ausgaben, sei es auf Reisen, bei Events oder innerhalb von Freundeskreisen, immer häufiger. Die Verwaltung dieser Ausgaben kann jedoch schnell zu einem mühsamen und chaotischen Unterfangen werden. Ausgaben werden auf Notizzetteln oder im WhatsApp-Chat gespeichert. Insbesondere wenn mehrere Personen beteiligt sind, ist es schwierig, den Überblick zu behalten, wer was ausgegeben hat, und wer wem Geld schuldet.

Um dieses Problem zu lösen, wurde im Rahmen des Moduls Programmieren-3 eine Android-App entwickelt, die das Management von Gruppenausgaben vereinfacht. Die App ermöglicht es den Nutzern, ihre Ausgaben gemeinsam zu erfassen, zuzuordnen und zu verrechnen. Die Idee und Funktionalität basieren auf der im Google PlayStore verfügbaren App „SplitWise“.

1.2 Ziel der Ausarbeitung

Ziel dieses Projektes war es Einblicke in die App-Entwicklung für Android zu bekommen, anhand eines realistischen Projektes, und dabei ein tieferes Verständnis für Softwareentwicklungskonzepte zu erarbeiten. In dieser Projektarbeit wird die entwickelte Android-App für das Management von Gruppenausgaben im Detail vorgestellt. Dabei werden sowohl die Funktionen und die Funktionsweise der App als auch die technische Implementierung erläutert. Im Einzelnen wird dargelegt, welche Funktionen die App zur Verwaltung von Gruppenausgaben bietet und wie sie funktioniert, einschließlich ihrer Bedienung. Darüber hinaus wird die technische Implementierung der App beschrieben, wobei auf die verwendeten technischen Komponenten eingegangen wird. Ein weiterer Aspekt dieser Arbeit ist die Erörterung der Herausforderungen, die während der Entwicklung der App auftraten, und die Darstellung der Lösungen, die zur Bewältigung dieser Probleme gefunden wurden. Schließlich wird untersucht, inwieweit die App das Problem der Verwaltung von Gruppenausgaben effektiv lösen kann.

1.3 Aufbau der Projektarbeit

Die Projektarbeit gliedert sich wie folgt: Im Grundlagenteil werden die theoretischen Grundlagen für die Entwicklung von Android-Apps erläutert, einschließlich der Konzepte von Java, XML und Android Studio. Der Hauptteil der Arbeit widmet sich der Implementierung der Android-App für das Management von Gruppenausgaben, wobei die einzelnen Funktionen der App, die verwendeten Technologien und die Implementierungsentscheidungen im Detail beschrieben werden. In der anschließenden Evaluation wird die

entwickelte App anhand der Aufgabenstellung aus der Einleitung bewertet. Dabei werden die Funktionen der App, die Benutzerfreundlichkeit und die technische Umsetzung der App näher betrachtet. Abschließend fasst das Fazit die wichtigsten Ergebnisse der Projektarbeit zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

1.4 Abgrenzung

In dieser Projektarbeit liegt der Fokus auf die Entwicklung der Android-App für das Management von Gruppenausgaben. Die Anwendung wurde als Single-User-Anwendung entwickelt, mit dem Ziel, sie in Zukunft zu einer Multi-User-Anwendung weiterzuentwickeln. Daher wurde auf die Implementierung eines Backends verzichtet, welches in dieser Arbeit ebenfalls nicht behandelt wird. Desweiteren laufen die verwendeten Dienste in einer Sandbox-Umgebung und benötigen einer Anpassung, um die Applikation echten Nutzern zur Verfügung stellen.

2 Darstellung der Grundlagen

2.1 Einleitung zu den Grundlagen (Arpad)

In diesem Kapitel werden die wesentlichen fachlichen Grundlagen erläutert, die der Entwicklung der Android-App zur Verwaltung von Gruppenausgaben zugrunde liegen. Ziel ist es, Wissenslücken der Lesenden zu schließen und darzulegen, dass diese Arbeit auf dem aktuellen technischen und wissenschaftlichen Stand basiert.

2.2 Grundlagen der Android-Entwicklung (Arpad)

2.2.1 Überblick über Android

Android ist ein mobiles Betriebssystem, das auf einem modifizierten Linux-Kernel basiert. Es wurde 2008 von Google herausgebracht und ist für Geräte mit Touchscreen wie Smartphones und Tablets optimiert. Seit seiner Einführung hat sich Android schnell zu einem der am weitesten verbreiteten Betriebssysteme für mobile Endgeräte entwickelt. Heute hält Android einen Marktanteil von über 70 % [@Sta].

Das rasante Wachstum in der Beliebtheit von Android ist auf mehrere Faktoren zurückzuführen. Zum einen ist Android Open Source. Das bedeutet, dass der Quellcode frei zugänglich ist. Dies ermöglicht Geräteherstellern und Entwicklern, das Betriebssystem nach ihren eigenen Bedürfnissen anzupassen. Zum anderen ist Android für Geräteherstellern und Entwickler kostenlos, was zu einer hohen Beliebtheit führt und die Produktionskosten für Android-Geräte senkt, wodurch diese zu einem niedrigeren Preis angeboten werden können.

2.2.2 Entwicklungsumgebung Android Studio

Android Studio ist die offizielle integrierte Entwicklungsumgebung (IDE) für die Entwicklung von Android Applikationen. Sie wurde von Google auf Basis von JetBrains IntelliJ IDEA entwickelt. Android Studio bietet eine Vielzahl von Werkzeugen und Funktionen, um die Entwicklung von Android Applikationen zu unterstützen [@Ast]:

- Der integrierte Code-Editor ermöglicht das Schreiben von Java- und Kotlin-Code, sowie Debugging und Refactoring.
- Mit dem Layout-Editor können schnell und einfach Benutzeroberflächen erstellt werden. Hierzu bietet Android Studio einen visuellen Editor sowie einen zugehörigen XML-Editor.

- Android Studio bietet die Möglichkeit, die App auf vielen verschiedenen virtuellen Geräten mit einem Emulator laufen zu lassen und zu testen. Es können auch physische Geräte verbunden werden, um die Apps auf diesen zu testen.
- Die Unterstützung des Build-Automatisierungstools Gradle gestaltet den Build-Prozess einfacher und effizienter.
- Versionskontrolle: Damit Teams gemeinsam an ihren Projekten arbeiten können, bietet Android Studio eine Integration für gängige Versionskontrollsysteme wie Git.

Neben den genannten Funktionen bietet Android Studio weitere Android-spezifische Werkzeuge. Unter anderem gibt es den Layout Inspector, um das Layout der Benutzeroberfläche detailliert zu untersuchen. Weitere Werkzeuge wie die Profiling-Tools zur Untersuchung der Performance der App und der Test Recorder zum Erstellen von Benutzertests machen Android Studio zu einer leistungsstarken und umfassenden Entwicklungsumgebung für die Entwicklung von Android-Applikationen.

2.3 Programmiersprachen und Technologien

2.3.1 Java (Arpad)

Java ist eine objektorientierte Programmiersprache, die 1995 von Sun Microsystems (jetzt Oracle) entwickelt wurde. Sie ist eine der beliebtesten Programmiersprachen und wird in einer Vielzahl von Bereichen eingesetzt, darunter gehören die Entwicklung von Web-Anwendungen, Web-Services und Embedded Systems wie Smart-TVs, Routern oder Fahrzeug-Infotainmentsystemen.

Java spielt insbesondere in der Entwicklung von Mobilen Anwendungen für das Android-Betriebssystem eine wichtige Rolle. Die Plattformunabhängigkeit dank der Java Virtual Machine (JVM) ermöglicht es Java Applikationen auf allen Geräten unter Android zu laufen. Die große Community und zahlreiche Bibliotheken bieten viele Ressourcen und Hilfe, um die Entwicklung von mobilen Anwendungen zu erleichtern.

Als Alternative zur Entwicklung von Android Applikationen wird die Programmiersprache Kotlin angeboten, welche eine neuere, von Java inspirierte Sprache ist und derzeit stark an Popularität gewinnt. Zudem gibt es die Möglichkeit C++ zu verwenden für die höchste Performance.

Im Folgenden dieser Ausarbeitung werden diese Alternativen nicht berücksichtig. Es wird lediglich auf die Entwicklung mit Java unter der Android API 33 und der Java Development kit Version 11 (JDK 11) eingegangen

2.3.2 XML (Arpad)

Extensible Markup Language (XML) ist eine textbasierte Datensprache und Datenformat zur Darstellung hierarchisch strukturierter Daten. Sie ist einfach zu lesen und zu schreiben und bietet eine Möglichkeit, Daten strukturiert zu speichern.

In der Android-Entwicklung wird das XML-Format verwendet, um die Benutzeroberfläche der Anwendung zu gestalten. Die einzelnen Elemente (z.B. Buttons, Textfelder, Bilder) werden in der Layoutstruktur der Anwendung definiert und deren Anordnung in XML-Dateien beschrieben.

Eine XML-Datei für die Benutzeroberfläche einer Android-Anwendung enthält eine Reihe von Elementen, die jeweils einen Teil der Benutzeroberfläche definieren. Jedes Element hat Attribute, die seine Eigenschaften festlegen.

```
0 | <Button
1 |     android:id="@+id/btn_change_split_ratio"
2 |     android:layout_width="wrap_content"
3 |     android:layout_height="wrap_content"
4 |     android:layout_weight="1"
5 |     android:text="@string/equal" />
```

Snippet 1: Darstellung eines Buttons im XML-Format

In Snippet 1 ist beispielhaft ein Button in XML-Format definiert dieses Element hat folgende Attribute:

- Android:id – legt die ID des Buttons fest
- Android:layout_width – legt die Breite des Buttons fest. In dem Fall soll er breit genug sein, um den Inhalt einzuschließen
- Android:layout_height – analog zur Breite
- Android:text – legt den Text auf dem Button fest wobei @string auf eine andere XML-Ressource verweist

```
0 | <LinearLayout
1 |     ...
2 |     <TextView
3 |         ... />
4 |
5 |     <Spinner
6 |         .../>
7 |
8 |     <Button
9 |         ... />
10| </LinearLayout>
```

Snippet 2: Verschachtelung von XML-Elementen

Snippet 2 zeigt, wie XML-Strukturen auch verschachtelt werden können, um komplexere Layouts darzustellen. Häufig werden hierzu Layout-Container wie z.B. das „Linear Layout“ verwendet, in dem mehrere Elemente angeordnet werden können.

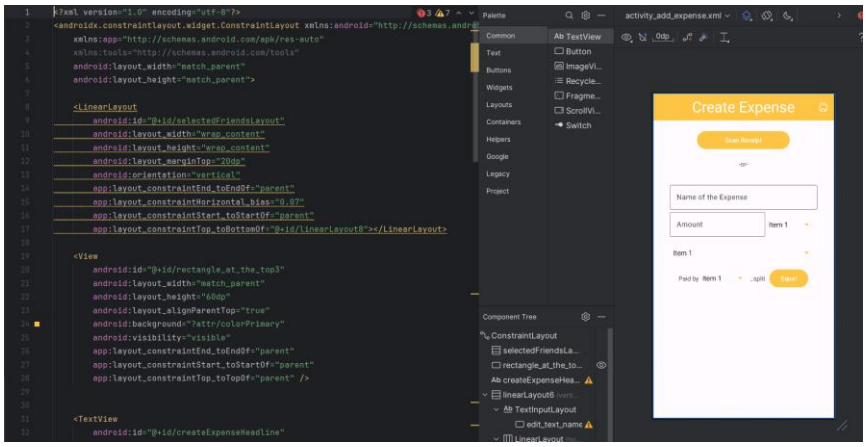


Abbildung 1: Android Studio bietet die Möglichkeit gleichzeitig im visuellen, als auch im XML-Editor zu arbeiten

Werden Layouts im visuellen Editor von Android Studio erstellt, so werden diese automatisch in Echtzeit in der zugehörigen XML-Datei sichtbar. So können beide Editor-Optionen, wie in Abbildung 1 dargestellt, mit ihren Stärken und Schwächen gleichzeitig genutzt werden

2.3.3 SQLite und Room (Dennis)

SQLite ist eine einfache, serverlose SQL-Datenbank, die sich besonders gut für mobile Geräte eignet. In der Android-Entwicklung wird sie häufig verwendet, um Daten lokal zu speichern.

Darauf aufbauend ist Room eine von Google entwickelte Bibliothek, die als Abstraktionsschicht über SQLite liegt und die Arbeit mit Datenbanken auf Android erleichtert und

sicherer gestaltet. Die Bibliothek bietet eine objektorientierte Schnittstelle zu SQLite, wodurch die Komplexität der SQL-Abfragen verringert wird. Es besteht im Wesentlichen aus drei Hauptkomponenten:

1. **Entity**: Diese Klassen repräsentieren die Tabellen in der Datenbank. Jede Entity ist eine einfache Java-Klasse, die mit Annotationen wie *@Entity* versehen wird.

2. **DAO** (Data Access Object): Dies sind Interfaces oder abstrakte Klassen, die Methoden enthalten, um auf die Datenbank zuzugreifen. Diese Methoden sind ebenfalls mit Annotationen wie *@Insert*, *@Update*, *@Delete* und *@Query* versehen.

3. **Database**: Eine abstrakte Klasse, die von `RoomDatabase` abgeleitet ist und als zentraler Zugangspunkt zur SQLite-Datenbank dient. Sie enthält den Code für die Instanziierung und Verwaltung der Datenbank und verbindet die DAOs mit den Entities.

Mit der Verwendung von Room wird Boilerplate-Code reduziert und werden Datenbankoperationen typensicher gemacht, was zu weniger Fehlern zur Laufzeit führt. Zudem bietet Room eine nahtlose Integration mit LiveData und dem Repository-Pattern, was die Arbeit mit reaktiven Datenströmen erleichtert.

Zusammengefasst bietet Room eine moderne, effiziente und sichere Methode, um SQLite-Datenbanken in Android Anwendungen zu verwalten und auf diese zuzugreifen.

2.3.4 Verwendung von APIs zur Integration externer Dienste (Arpad)

.a Paypal SDK

Um dem Benutzer die Möglichkeit zu bieten schnell und einfach seine Bilanz auszugleichen, wurde PayPal mithilfe des PayPal Android SDK in die App integriert. Der Benutzer überweist mithilfe dieser Integration den ausstehenden Betrag, welcher in der Live-Version der App automatisch an die anderen Teilnehmer verteilt wird.

Wegen der Abgrenzung in der Sandbox-Umgebung wurde nur die Integration eingearbeitet, ohne das echte Accounts verwendet werden und ohne dass tatsächlich Geld fließt. In dieser Ausarbeitung wird nur auf die Verwendung der PayPal SDK in der Android Anwendung eingegangen. Die Backend-Architektur zur Verteilung des Geldes zwischen den beteiligten Mitgliedern einer Gruppe wird im Rahmen dieses Projektes nicht berücksichtigt.

.b CameraX

Eine besondere Funktion der Applikation Bananasplit ist schnelle und bequeme Einscannen von Belegen und Rechnungen direkt mit der Handykamera. Damit wird vermieden diese manuell in das vorhandene Formular einzutippen. Dies beschleunigt den Prozess und motiviert dazu, seine Ausgaben mithilfe von Bananasplit transparent mit anderen zu teilen und gemeinsam im Blick zu behalten.

Für die Umsetzung dieser Funktionalität wird zunächst der Zugriff auf die Kamera benötigt. Android bietet hierfür die Jetpack-Bibliothek CameraX. Sie abstrahiert die Komplexität der kamerabasierten APIs für eine konsistente Verwendung auf verschiedenen Geräten und Android Versionen. CameraX bietet auch die Möglichkeit, eine Live-Vorschau des Kamerabildes in der Anwendung anzuzeigen. Im Funktionsumfang ist auch eine Liveanalyse des Bilds vorhanden, z.B. zum Scannen von QR-Codes, welche jedoch hier nicht verwendet wurden.

.c Google ML Kit

Für die tatsächliche Texterkennung bietet Google das Google Ml Kit, ein SDK, das Googles Machine Learning (ML)-Kapazitäten für Android und iOS-Applikationen zur Verfügung stellt. Es bietet leistungsstarke, jedoch einfach zu verwendende Pakete, die auch Entwicklern ohne tiefgehende ML-Kenntnissen, die Möglichkeit bieten, fortgeschrittene Funktionalitäten in ihre Apps zu integrieren. Es bietet bereits trainierte Modelle, welche auch auf dem Gerät ausgeführt werden können für schnellere Ergebnisse auch ohne Verbindung zum Internet. Zu den Features gehören Objekterkennung, Bildbeschreibung, Barcode-Scannung und für dieses Projekt relevante Funktionalitäten zur Texterkennung in Bildern.

.d Google Cloud API

Die Google Cloud API Suite bietet Entwicklern ein breites Spektrum an Tools und Diensten, um cloudbasierte Anwendungen zu erstellen und zu erweitern. Sie ermöglicht den Zugriff auf verschiedene Cloud-Ressourcen wie Datenspeicherung, Cloud-Computing und maschinelles Lernen.

Die Cloud Natural Language API ist ein Bestandteil der Google Cloud API Suite und bietet leistungsstarke Funktionen zur Verarbeitung natürlicher Sprache (NLP) in Texten. Sie umfasst unter anderem die Funktionen der Entitätserkennung, Tokenisierung und Coreferenz. Diese Funktionen unterstützen die Erkennung von Produkten und Dienstleistungen auf Belegen und ermöglichen die Zuordnung zwischen Produkt und Preis.

.e Weitere Erwägungen

Während der Entwicklungszeit wurde in Erwägung gezogen, anstelle der Google Cloud API das Stanford NLP Toolkit zu nutzen. Stanford NLP ist ein Open-Source-Toolkit zur Verarbeitung natürlicher Sprache in Java, dessen Funktionsumfang vergleichbar mit dem der Google Cloud API ist. Es stellte sich jedoch heraus, dass das zugrunde liegende Modell aufgrund seiner Komplexität und Größe nicht für den Einsatz auf mobilen Endgeräten geeignet ist. Beim Testen kam es häufig zu Abstürzen, die durch unzureichenden Arbeitsspeicher verursacht wurden.

Eine alternative Möglichkeit wäre gewesen, ein eigenes TensorFlow Lite-Modell für den lokalen Einsatz auf dem Gerät zu trainieren. Dies hätte es ermöglicht, die Funktionalität beizubehalten. Für das Training des Modells wären zufällige Kassenbelege, beispielsweise mit Python erstellt, sowie deren Tokenisierung erforderlich gewesen. Aufgrund der Komplexität und des begrenzten Zeitrahmens haben wurde entschieden, diese Lösung nicht weiter zu verfolgen.

2.4 Konzepte und Designprinzipien

2.4.1 Android Lifecycle (Dennis)

Der Lifecycle einer Android-App ist ein zentrales Konzept in der Android-Entwicklung und beschreibt die verschiedenen Zustände, die eine App vom Starten bis zum Beenden durchläuft. Die Kernkomponenten dieses Lifecycles sind die Aktivitätsmethoden, die von der Basisklasse *Activity* abgeleitet sind. Die wichtigsten Methoden und ihre Bedeutung sind die Folgenden:

1. **onCreate()**: Diese Methode wird aufgerufen, wenn die Aktivität zum ersten Mal erstellt wird. Hier werden grundlegende Initialisierungen wie das Laden von Layouts, das Einrichten von View-Komponenten und das Initialisieren von Datenquellen vorgenommen.

2. **onStart()**: Diese Methode wird aufgerufen, wenn die Aktivität sichtbar wird, aber noch nicht interaktiv ist. Hier können Ressourcen, die vom Benutzer sichtbar sein müssen, z.B. eine UI-Aktualisierung, vorbereitet werden.

3. **onResume()**: Diese Methode wird aufgerufen, wenn die Aktivität in den Vordergrund tritt und der Benutzer mit ihr interagieren kann. Dies ist der richtige Ort, um

Animations- oder Medienwiedergaben zu starten oder andere Aktionen auszuführen, die vom Benutzer benötigt werden.

4. **onPause()**: Diese Methode wird aufgerufen, wenn eine andere Aktivität in den Vordergrund tritt und die aktuelle Aktivität pausiert wird. Diese Methode sollte genutzt werden, um Operationen zu stoppen, die nicht weiterhin laufen sollen (wie Animationen) oder um weniger wichtige UI-Updates zu unterbrechen. Hier sollten auch Sensordaten und andere Ressourcen freigegeben werden.

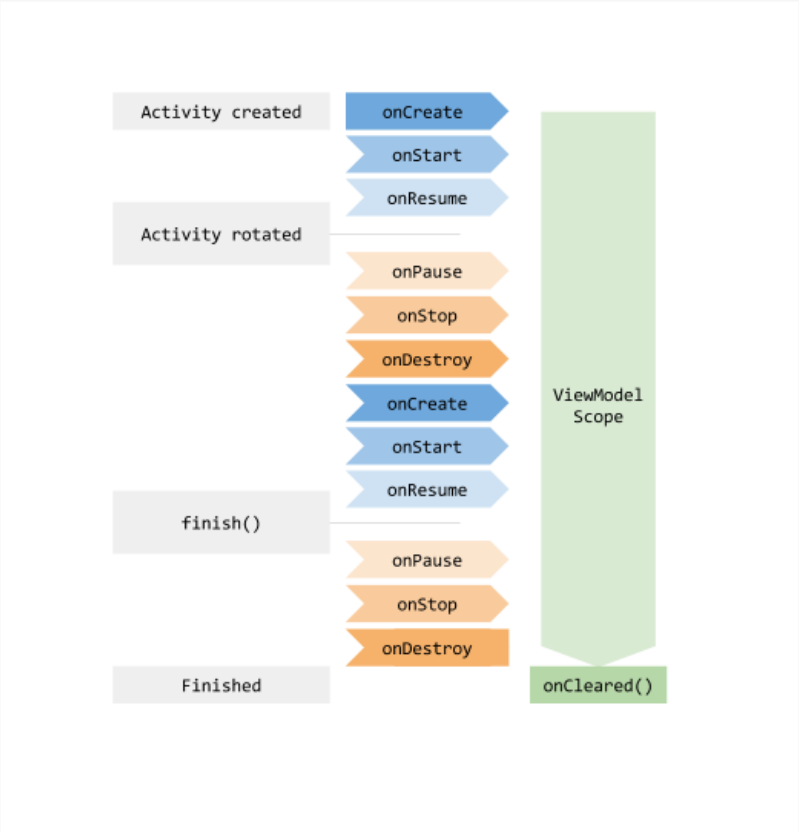
5. **onStop()**: Diese Methode wird aufgerufen, wenn die Aktivität für den Benutzer nicht mehr sichtbar ist. Hier können alle weiteren Ressourcen freigegeben oder Änderungen gespeichert werden, die an den Benutzerzustand oder das UI gebunden sind.

6. **onDestroy()**: Diese Methode wird aufgerufen, bevor die Aktivität endgültig zerstört wird. Hier sollten alle verbleibenden Ressourcen freigegeben und endgültige Bereinigungen durchgeführt werden.

7. **onRestart()**: Diese Methode wird aufgerufen, wenn die Aktivität gestoppt wurde und dann wieder gestartet wird. Hier kann der Zustand der Aktivität wiederhergestellt und die Interaktion mit dem Benutzer fortgesetzt werden.

Zusätzlich zu diesen Kernmethoden gibt es weitere Mechanismen, um Zustandsänderungen zu verwalten, insbesondere das Speichern und Wiederherstellen von Zuständen durch **onSaveInstanceState()** und **onRestoreInstanceState()**. Diese Methoden werden verwendet, um sicherzustellen, dass Benutzerdaten oder UI-Zustände bei Konfigurationsänderungen (wie Bildschirmdrehung) oder nach unerwarteten Beendigungen (wie Ressourcenmangel) erhalten bleiben.

Die korrekte Implementierung dieser Methoden ist entscheidend für das reibungslose und effiziente Verhalten einer Android-App. Dadurch wird sichergestellt, dass die Ressourcen sinnvoll genutzt werden und die Benutzererfahrung nahtlos bleibt, selbst wenn sich die App in verschiedenen Zuständen befindet.



Diese Abbildung zeigt beispielhaft den Lifecycle eines ViewModels.

2.4.2 ViewBinding (Arpad)

ViewBinding ist ein modernes Tool in der Android-Entwicklung. Wenn ViewBinding für ein Projekt aktiviert ist, werden automatisch Binding-Klassen für alle XML-Layout-Ressourcen erzeugt. Eine Instanz dieser Klassen enthält Referenzen zu allen Elementen, die eine ID besitzen.

```
1 | View contentView = inflater.inflate(R.layout.activity_groups, getContentContainer(),
false);
2 | binding = ActivityGroupsBinding.bind(contentView);
3 | binding.btnAddGroup.setOnClickListener(...)
```

Snippet 3: Erstellung eines Bindings und direkter Zugriff auf Views

Wie in Snippet 3 gezeigt, kann man nach der Erstellung eines Bindings, wie in Zeile 3 dargestellt, direkt auf die Elemente zugreifen. ViewBinding ersetzt dabei den Befehl

„findViewById“ und führt zu saubererem und übersichtlicherem Code. Neben der verbesserten Lesbarkeit bietet es auch Typensicherheit und Schutz vor NullPointerExceptions, die durch nicht gefundene IDs entstehen könnten. Es garantiert, dass Views nur dann verfügbar sind, wenn sie tatsächlich im Layout definiert wurden.

2.4.3 MVVM-Pattern (Dennis)

Das MVVM-Pattern (Model-View-ViewModel) ist ein Architekturmuster, das häufig in der Softwareentwicklung verwendet wird, insbesondere bei der Entwicklung von Benutzeroberflächen. Es trennt die logische Schicht einer Anwendung von ihrer Darstellungsschicht und erleichtert so Wartung, Testbarkeit und Wiederverwendbarkeit des Codes. Das MVVM-Pattern besteht aus drei Hauptkomponenten:

1. **Model**: In Android repräsentiert das Model die Daten und die Geschäftslogik. Es umfasst Datenquellen wie Datenbanken (z.B. Room), Netzwerkschnittstellen (z.B. Retrofit) und Repositories, die als Vermittler zwischen Datenquellen und dem Rest der Anwendung fungieren. Das Model ist unabhängig von der Benutzeroberfläche.

2. **View**: Die View in Android ist die Activity oder das Fragment, welche die Benutzeroberfläche darstellt. UI-Komponenten (z.B. TextViews, Buttons, RecyclerViews) sind in XML-Layouts definiert. Die View ist zuständig für das Rendern der UI, Datenbindung und das Empfangen von Benutzerinteraktionen.

3. **ViewModel**: Das ViewModel dient als Vermittler zwischen der View und dem Model. Es enthält LiveData oder StateFlow, um Änderungen in den Daten zu beobachten und der View mitzuteilen. Android bietet die `ViewModel`-Klasse als Teil der Architekturkomponenten, die helfen, ViewModels zu erstellen, die die UI-Logik kapseln und längere Lebenszyklen haben, unabhängig von Configuration Changes (wie z.B. Bildschirmdrehungen). [@VM]

Diese drei Komponenten wirken zusammen, indem die View die ViewModel-Instanz bindet und mittels Observer-Methoden dortige LiveData oder StateFlow-Objekte beobachtet. Veränderungen in diesen Objekten lösen automatische Updates in der UI aus. Das ViewModel holt Daten aus dem Model und stellt diese der View zur Verfügung. Es nimmt Benutzerinteraktionen entgegen, meist durch LiveData-Mechanismen oder Funktionen, und delegiert Aufgaben an das Model. Somit sorgt es dafür, dass die View immer auf dem aktuellen Stand der Daten ist. Das Model führt die eigentliche Datenverwaltung und Geschäftslogik aus. Es könnte z.B. Daten aus einer Room-Datenbank oder einer REST-API über Retrofit laden.

2.4.4 User Interface (Dennis)

Bei der Erstellung der Activities haben wir uns an den "Eight Golden Rules of Interface Design" von Ben Shneiderman orientiert [@Shn]. In unserem Design streben wir nach Konsistenz. Überall in der App finden sich einheitliche Farben, Schriftarten und Symbole. Funktionen wie das Hinzufügen von Ausgaben oder das Verwalten von Gruppen arbeiten immer nach den gleichen Mustern, was den Lernaufwand für die Nutzer minimiert und die Bedienung intuitiv macht. Bananasplit ermöglicht es Nutzern, ihre Aktionen leicht rückgängig zu machen. Wird eine falsche Ausgabe hinzugefügt oder eine falsche Eingabe gemacht, können diese Fehler mit nur wenigen Klicks korrigiert werden. Diese Funktion fördert das Experimentieren und die Entdeckungsfreude, da die Nutzer keine Angst haben müssen, irreversible Fehler zu begehen. Die App unterstützt ein Gefühl der internen Kontrolle beim Nutzer. Nutzer haben jederzeit die Kontrolle über ihre Aktionen und werden nicht von unerwarteten Systemereignissen überrascht. Die Navigation ist intuitiv und logisch aufgebaut, was den Nutzer im Zentrum der Interaktion belässt und das Gefühl der Autonomie verstärkt. Schließlich reduziert Bananasplit die Belastung des Kurzzeitgedächtnisses des Nutzers indem wichtige Informationen stets sichtbar und leicht zugänglich sind. Listen und Menüs sind klar strukturiert, und die Nutzung visueller Hinweise erleichtert das Zurechtfinden innerhalb der App. Diese Designstrategie minimiert die kognitive Belastung und fördert eine reibungslose Nutzererfahrung.

2.4.5 User Experience (Dennis)

Um die User Experience zu evaluieren, haben wir diese drei Personae als Referenz herangezogen. Diese Auswahl ermöglicht es uns, die Bedürfnisse und Erwartungen einer breiten Nutzerbasis abzudecken. Durch die Berücksichtigung vielfältiger Nutzerprofile wie einer Studentin, eines Familienvaters und einer freiberuflichen Fotografin können wir sicherstellen, dass Bananasplit nicht nur benutzerfreundlich ist, sondern auch flexibel auf unterschiedliche Nutzungsszenarien reagiert. Diese umfassende Betrachtung hilft uns dabei, spezifische Usability-Probleme zu identifizieren und gezielte Verbesserungen vorzunehmen, um eine optimale Erlebnissqualität für jede Gruppe zu gewährleisten.

Lisa Müller (Studentin)

Lisa ist eine 22-jährige Psychologiestudentin, die eine Wohnung mit drei Kommilitonen teilt. Sie nutzt Bananasplit zur Verwaltung gemeinsamer Ausgaben wie Miete, Lebensmittel und Haushaltsbedarf. Lisa findet die Benutzeroberfläche übersichtlich und kann ihre wöchentlichen Einkäufe problemlos eintragen. Die App bietet ihr klare Eingabefelder und zahlreiche Erklärungen. Die Echtzeit-Updates der geteilten Ausgaben geben ihr sofortiges Feedback über ihre und die Beiträge ihrer Mitbewohner. Sie hat jederzeit einen

klaren Überblick. Wenn Lisa versehentlich eine falsche Ausgabe einträgt, kann sie den Fehler leicht korrigieren. Die App zeigt ihr verständliche Fehlermeldungen und bietet unkomplizierte Lösungsvorschläge. Lisa schätzt die Fairness und die einfache Handhabung, die Streitigkeiten und Missverständnisse mit ihren Mitbewohnern minimieren. Dadurch hat sie mehr Zeit und weniger Stress, sich auf ihr Studium zu konzentrieren.

Andreas Schmidt (Familienvater)

Andreas ist ein 45-jähriger Lehrer, der regelmäßig Familienausflüge und Urlaube plant. Er nutzt Bananasplit, um die Ausgaben transparent zu dokumentieren und mit seiner Frau zu teilen. Für Familienausflüge kann Andreas unkompliziert gemeinsame Ausgaben erfassen - von Eintrittskarten bis hin zu Restaurantrechnungen. Die strukturierte Oberfläche führt ihn intuitiv durch den Prozess. Andreas und seine Frau können beide die App nutzen, um Ausgaben in Echtzeit zu sehen. Dies ermöglicht eine klare Budgetübersicht und verhindert Konflikte um finanzielle Angelegenheiten. Andreas muss sich keine Sorgen machen, wenn er eine Ausgabe falsch einträgt. Die App ermöglicht es ihm, Fehler einfach zu korrigieren, und gibt hilfreiches Feedback, das er leicht verstehen kann. Dank der transparenten Dokumentation der Ausgaben kann Andreas die finanziellen Aspekte von Familienaktivitäten stressfrei und effizient managen. Der gemeinsame Überblick stärkt zudem das Vertrauen in die finanzielle Handhabung.

Maria Gomez (Freiberufliche Fotografin)

Maria ist eine 30-jährige freiberufliche Fotografin, die häufig an Projekten mit anderen Kreativen zusammenarbeitet. Sie nutzt Bananasplit zur fairen Verrechnung gemeinsamer Kosten. Die App ermöglicht es ihr, detaillierte Projektausgaben für Shootings und andere Kreativprojekte schnell und genau einzugeben. Die professionell anmutende Benutzeroberfläche passt gut zu ihren Arbeitsstandards. Die Übersichtsfunktion zeigt Maria und ihren Projektpartnern jederzeit, wie die Ausgaben verteilt sind. Diese Transparenz fördert das Vertrauen und die Zusammenarbeit im Team. Maria kann Änderungen an Projektausgaben flexibel und unkompliziert durchführen. Die App bietet klare Anleitungen und schnelle Korrekturmöglichkeiten. Für Maria ist Bananasplit ein wertvolles Werkzeug, um die finanziellen Aspekte ihrer kreativen Projekte zu verwalten. Sie kann sich vollkommenauf ihren kreativen Prozess konzentrieren, ohne sich um die faire Kostenverteilung Sorgen machen zu müssen.

3 Anwendung

3.1 Einleitung zu der Anwendung (Arpad)

In diesem Kapitel wird die praktische Anwendung der entwickelten Android-App zur Verwaltung von Gruppenausgaben detailliert erläutert. Die wichtigsten Funktionen der App werden beschrieben und es werden praktische Anwendungsbeispiele gegeben, um die Nutzung der App zu veranschaulichen.

3.2 Benutzeroberfläche und Navigation (Dennis)

Die Benutzeroberfläche und Navigation der App Bananasplit sind darauf ausgelegt, den Nutzern ein intuitives, reibungsloses und angenehmes Erlebnis zu bieten. Klare visuelle Hierarchien und gut platzierte Designelemente sorgen dafür, dass wichtige Informationen auf einen Blick erfasst werden können.

Die Hauptnavigationsleiste, die sich am unteren Bildschirmrand befindet, bietet leicht zugängliche Symbole für die wichtigsten Funktionen der App: Gruppenverwaltung, Freundesliste, Aktivitäten und Benutzereinstellungen. Diese klare Struktur ermöglicht es den Nutzern, sich schnell und sicher durch die App zu bewegen.

Im Bereich der Gruppenverwaltung können Nutzer neue Gruppen erstellen oder bestehenden Gruppen beitreten. Jedes Gruppenprofil zeigt die Mitglieder, die aktuellen Ausgaben und den Gruppensaldo an.

Die Benutzereinstellungen sind ebenfalls klar und logisch aufgebaut. Hier können Nutzer ihre persönlichen Informationen verwalten und die bevorzugte Währung oder Sprache festlegen. Die Oberfläche ist darauf ausgelegt, dem Nutzer die volle Kontrolle über seine Daten und Präferenzen zu geben.

Ein wichtiger Aspekt der Navigation ist das Aktivitätstracking. Eine spezielle Seite innerhalb der App zeigt übersichtlich die letzten Aktivitäten und Interaktionen an, sodass Nutzer jederzeit nachvollziehen können, wer welche Aktionen durchgeführt hat. Diese Transparenz schafft Vertrauen und ermöglicht eine einfache Nachverfolgung von Änderungen und Updates.

3.3 Funktionen der App (Dennis)

Die App Bananasplit ist speziell darauf ausgelegt, das Teilen und Verwalten von Ausgaben innerhalb von Gruppen einfach und effizient zu gestalten. Eine zentrale Funktion der App ist die Möglichkeit, Gruppen zu erstellen und zu verwalten. Nutzer können neue

Gruppen anlegen, beispielsweise für WG-Mitglieder, Freundeskreise oder Reiseteams, und Mitglieder zu diesen Gruppen hinzufügen oder entfernen.

Ein weiterer Kernbereich der App ist das Hinzufügen und Verwalten von Ausgaben. Nutzer können neue Ausgaben für eine Gruppe erfassen und dabei Details wie Betrag, Datum, Beschreibung und Kategorie eingeben. Außerdem können sie angeben, wer die Ausgabe getätigt hat und welche Gruppenmitglieder daran beteiligt sind. Die App zentralisiert alle Ausgaben und bietet eine übersichtliche Darstellung der Gesamtausgaben sowie der individuellen Beiträge der Mitglieder.

Die App kümmert sich auch um die automatische Aufteilung und den Ausgleich von Ausgaben. Bananasplit berechnet automatisch, wie die Ausgaben unter den Mitgliedern aufgeteilt werden sollen, je nach deren angegebenen Beteiligungen. Nutzer können ihren aktuellen Schuldsaldo einsehen und erkennen somit auf einen Blick, wer wem wie viel Geld schuldet. Darüber hinaus können Nutzer Schulden ausgleichen und diese Zahlungen dokumentieren, um ein stets aktuelles Bild der finanziellen Verhältnisse zu erhalten. Nutzer können die App-Einstellungen an ihre Bedürfnisse anpassen, insbesondere Währungen, Sprache und Hellen oder Dunklen Modus.

Das Aktivitätstracking stellt eine weitere essentielle Funktion dar. Die App speichert alle Nutzeraktivitäten, um die durchgeführten Aktionen nachverfolgen zu können. Diese Funktion ermöglicht eine transparente Nachverfolgung, was zur Verbesserung der Nutzererfahrung beiträgt.

Mit diesen umfangreichen Funktionen fördert Bananasplit Fairness, Transparenz und eine effiziente Zusammenarbeit, indem die finanziellen Interaktionen innerhalb von Gruppen nahtlos und benutzerfreundlich organisiert werden. Die App erleichtert den Alltag für ihre Nutzer, indem sie eine klare und strukturierte Verwaltung gemeinsamer Ausgaben ermöglicht.

3.4 Technische Details der Implementierung

3.4.1 Paypal Integration (Arpad)

Um die Verbindung zu den PayPal-Servern herzustellen, muss der Client zunächst konfiguriert werden. PayPal bietet hierfür drei Möglichkeiten: „Card Payments“, „Native Payments“ und „Web Payments“ [@PPD].

- Card Payments: Wenn vollständige Kontrolle über den Zahlungsablauf benötigt wird. Der Zahlungsprozess kann genau an das Design der App anpasst werden

- Native Payments: Ermöglichte einen nahtlosen Bezahlvorgang innerhalb der App.
- Web Payments bieten eine schnelle Integration. Der Zahlungsvorgang wird vollständig im Browser abgewickelt.

Für eine ungestörte und schnelle Abwicklung der Transaktionen wurde die Entscheidung getroffen, die Integration von Native Payments umzusetzen.

```
CheckoutConfig config = new CheckoutConfig(
        this.getApplication(),
        ClientID,
        Environment.SANDBOX,
        CurrencyCode.USD,
        UserAction.PAY_NOW,
        "nativexo://paypalpay"
);
PayPalCheckout.setConfig(config);
```

Snippet 4: Konfiguration des PayPal-Clients

Snippet 4 zeigt die Konfiguration von PayPal innerhalb der Anwendung. Diese Konfiguration ist erforderlich, um eine Verbindung zu den PayPal-Services herzustellen. Die ClientID muss festgelegt werden, um eine Authentifizierung zu ermöglichen. Der Android App-Link „nativexo://paypalpay“ (Deep Link) verweist auf einen Punkt in der App, der nach erfolgreicher Zahlung aufgerufen werden soll. In diesem Beispiel wird auf die Sandbox-Umgebung von PayPal zugegriffen, die das Testen der Integration ermöglicht. Es werden nur Mock-Transaktionen durchgeführt, ohne echtes Geld zu verwenden.

```
binding. PaymentButtonContainer.setup(
        createOrderActions -> {
            ArrayList<PurchaseUnit> purchaseUnits = new ArrayList<>();
            purchaseUnits.add(
                    new PurchaseUnit.Builder()
                            .amount(
                                    new Amount.Builder()
                                            .currencyCode(CurrencyCode.USD)
                                            .value("10.00")
                                            .build()
                            )
                            .build()
            );
            OrderRequest order = new OrderRequest(
                    OrderIntent.CAPTURE,
                    new AppContext.Builder()
                            .userAction(UserAction.PAY_NOW)
                            .build(),
                            purchaseUnits
            );
            createOrderActions.create(order, (CreateOrderActions.OnOrderCreated) null);
        },
        approval -> approval.getOrderActions().capture(result -> {...
        })
);
```

Snippet 5 : Einstellung einer PayPal-Transaktion

In Snippet 4 zeigt, wie die Funktionalität zum Öffnen des nativen PayPal-Dialogs an die Klasse „PaypalPaymentContainer“ gebunden wird. Diese Klasse definiert nicht nur das

Aussehen des Buttons, sondern beinhaltet auch die Funktionalität. Zunächst muss eine Transaktion (Purchase Unit) definiert werden, die den zu überweisenden Betrag festlegt, und schließlich die Anfrage (Order Request), die herausgesendet wird. Diese Anfrage signalisiert, dass der Betrag sofort nach Transaktionsabschluss überwiesen werden soll. Die Lambda-Ausdruck unter „approval“ definiert das Verhalten nach Erhalt einer Antwort von den PayPal-Services. Hier kann der Nutzer beispielsweise über den Status seiner Transaktion informiert werden.

3.4.2 CameraX (Arpad)

Commented [ÁH1]: Todo anfang vervolsständigen

CameraX ist die moderne und benutzerfreundlichere Schnittstelle zur Nutzung der Kamera auf Android-Geräten. Im Gegensatz dazu bietet Camera2 eine tiefere Kontrolle über die Kamera Hardware. Dies erfordert jedoch ein detailliertes Verständnis der Kamera-Funktionalität und benötigt einer manuellen Konfiguration. Für dieses Projekt ist ein derart hohes Maß an Steuerung nicht notwendig, weshalb die Entscheidung getroffen, CameraX zu benutzen.

```
1 |     ProcessCameraProvider.getInstance(previewView.getContext())
2 |             .addListener(() -> {
3 |                 try {
4 |                     ProcessCameraProvider cameraProvider = ProcessCameraProvider.get-
Instance(previewView.getContext()).get();
5 |                     Preview preview = new Preview.Builder().build();
6 |                     preview.setSurfaceProvider(previewView.getSurfaceProvider());
7 |                     imageCapture = new ImageCapture.Builder().build();
8 |                     CameraSelector cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA;
9 | ...
10|                 } catch (ExecutionException | InterruptedException e) {...}
11|             }, ContextCompat.getMainExecutor(previewView.getContext()));
```

Snippet 6: Erstellung eines Kamera-Prozesses mit CameraX

In Snippet 6 wird beispielhaft ein Kamera Prozess gestartet. Hierbei wird spezifiziert, dass die Hauptkamera verwendet werden soll und es wird der Context mitgegeben in welchem das Preview erscheinen soll.

```
1 | imageCapture.takePicture(ContextCompat.getMainExecutor(previewView.getContext()),
captureCallback);
```

Snippet 7: Aufnahme eines bildes durch die Kamera

In Snippet 7 wird das tatsächliche Bild von der Kamera aufgenommen. “imageCapture.takePicture()” startet den AufnahmeProzess für das Bild. Als Argument wird der Executor festgelegt, der den Callback im Hauptthread ausführt.

```
@Override
public void onCaptureSuccess(@NonNull ImageProxy image) {
    processImage();
}
```

Snippet 8: Implementation zur Behandlung einer Erfolgreichen Bildaufnahme

Die Kamera verwendende Klasse kann wie in Snippet 8 die anonyme Funktion im Callback implementieren und das entstandene Bild weiterverarbeiten.

```
  private void processImage(ImageProxy image) {
        Bitmap bitmap = ImageUtils.imageProxyToBitmap(image);
        textRecognizer.processImage(bitmap, image.getImageInfo().getRotationDegrees());
```

Snippet 9 : Weiterverabeitung des Bildes.

Das entstandene ImageProxy Objekt wird nicht auf dem Gerät gespeichert, um den Speicherplatz und Fotogalerie des Nutzers nicht unnötig aufzufüllen. Stattdessen wird in Snippet 9 das bild als Bitmap an den Texterkennungsprozess weitergegeben

3.4.3 Google MIkit für Texterkennung (Arpad)

In diesem Abschnitt wird die Implementierung der Texterkennung mit Google ML Kit erläutert. Google ML Kit bietet leistungsstarke und einfach zu verwendende Funktionen für maschinelles Lernen, die direkt in mobile Apps integriert werden können. Hier wird die Texterkennung genutzt, um Text aus Bildern zu extrahieren als Vorbereitung für den NLP Prozess.

```
TextRecognition.getClient(TextRecognizerOptions.DEFAULT_OPTIONS)
        .process(inputImage)
        .addOnSuccessListener(callback::onTextRecognized)
        .addOnFailureListener(callback::onTextRecognitionFailed);
```

Snippet 10 : Konfiguration des TextRecognitionClients

In Snippet 10 wird der TextRecognitionClient konfiguriert. Dieser wird mit den Standardoptionen “TextRecognizerOptions.DEFAULT_OPTIONS“ initialisiert. Diese Optionen umfassen vordefinierte Einstellungen für die Texterkennung, welche für die meisten Anwendungsfälle geeignet sind. Dem TextCognitionClient wird das Bild übergeben und der Erkennungsprozess wird asynchron gestartet.

```
1 | public interface TextRecognizerCallback {
2 |     void onTextRecognized(Text result);
3 |     void onTextRecognitionFailed(Exception e);
4 | }
5 | ...
6 | private void setupTextExtraction() {
7 |         textRecognizer = new TextRecognizer(new TextRecognizer.TextRecognizerCallback() {
8 |             @Override
9 |             public void onTextRecognized(Text result) {
10|                 processTextBlock(result);
11|             }
12|
13|             @Override
14|             public void onTextRecognitionFailed(Exception e) {
15|                Toast.makeText(ScannerActivity.this, "Failed to recognize text",
                  Toast.LENGTH_SHORT).show();
16|             }
17|         });
18|     }
```

Snippet 11: Definition und Verwendung des Callback-Interfaces

Sobald der Prozess abgeschlossen ist, werden abhängig vom Status, die Methoden des in Snippet 11 definierten Callback-Interfaces aufgerufen. Diese Callback-Methoden können anonym in der aufrufenden Klasse definiert werden. Die Verwendung des Callback-Interfaces ermöglicht die einfache Wiederverwendung dieser Funktionalität im Projekt und steigert die Modularität. Zeilen 6-18 zeigen die Verwendung innerhalb des Scanners. Bei einem erfolgreich abgeschlossenen Text-Extrahierungsprozess, wird das Ergebnis an die nächste Station in der Pipeline übergeben. Sollte ein Fehler auftreten, wird der Nutzer darüber informiert.

3.4.4 Tokenisierung des Textes (Arpad)

Der durch Googles ML Kit extrahierte Text liegt nun als Zeichenkette vor und muss analysiert werden, um unerwünschte Informationen herauszufiltern. Dazu gehören beispielsweise Adressen, steuerrechtliche Informationen und Werbung, die häufig auf Kassenbelegen zu finden sind.

Bevor Anfragen an die Cloud Natural Language API gesendet werden können, ist eine Authentifizierung erforderlich.

```
1 | executorService.execute(() -> {
2 | try (InputStream credentialsStream = context.getAssets().open("credentials.json")) {
3 |     GoogleCredentials credentials = GoogleCredentials.fromStream(credentialsStream);
4 |     LanguageServiceSettings settings = LanguageServiceSettings.newBuilder()
5 |             .setCredentialsProvider(FixedCredentialsProvider.create(credentials))
6 |             .build()
7 | } catch (IOException e) {...}
```

Snippet 12: Authentifizierung gegenüber der Google Cloud API

Snippet 12 zeigt, wie aus einem JSON-Dokument die zu dem Google-Entwicklerkonto gehörenden Credentials gelesen und in einem “GoogleCredentials” Objekt hinterlegt werden. Ohne die korrekten Credentials werden sämtliche Anfragen an die Cloud API abgelehnt. Die Einstellungen werden mithilfe des Builder-Patterns zusammengestellt.

```
try (LanguageServiceClient languageServiceClient = LanguageServiceClient.create(settings))
{
    Document doc = Document.newBuilder()
            .setContent(zu_analysierender_text)
            .setType(Document.Type.PLAIN_TEXT)
            .setLanguage("de")
            .build();

    AnalyzeEntitiesResponse response = languageServiceClient.analyzeEntities(doc);
    List<Entity> entities = response.getEntitiesList();
    callback.onAnalysisSuccess(entities);
}
```

Snippet 13: Erstellen und Abschicken einer Anfrage an die Cloud Natural Language API

Die Einstellungen werden in Snippet 13 bei der Erstellung des Clients an diesen übergeben. Damit der Client die Anfrage absenden kann muss ein für den Client verständliches Dokument erstellt werden. Dieses legt den Inhalt, den Typ und die Sprache fest. Aufgrund der verschiedenen Kassenbelegformate wurde die Entscheidung getroffen, sich ausschließlich auf die Analyse der deutschen Sprache zu konzentrieren.

Diese Anfrage, die das Dokument enthält, wird durch den Client an den Server geschickt. Da diese Operation eine Netzwerkverbindung erfordert, muss sie in einem separaten Thread durchgeführt werden, weil Android keine Netzwerkverbindungen im Main-Thread zulässt [@Cnct].

Abbildung 2: Ergebnis des NLP Prozesses

Das resultierende Ergebnis ist die in Abbildung 2 dargestellte Liste an Entitäten, denen Eigenschaften und Kategorien zugewiesen wurden. Dazu gehören unter anderem: „NAME“, „NUMBER“ und „CONSUMER_GOOD“. Anhand dieser Einteilung können die einzelnen Einträge gefiltert werden. Übrig bleiben die Produkte sowie die dazugehörigen Zahlen wie Quantität und Preise.

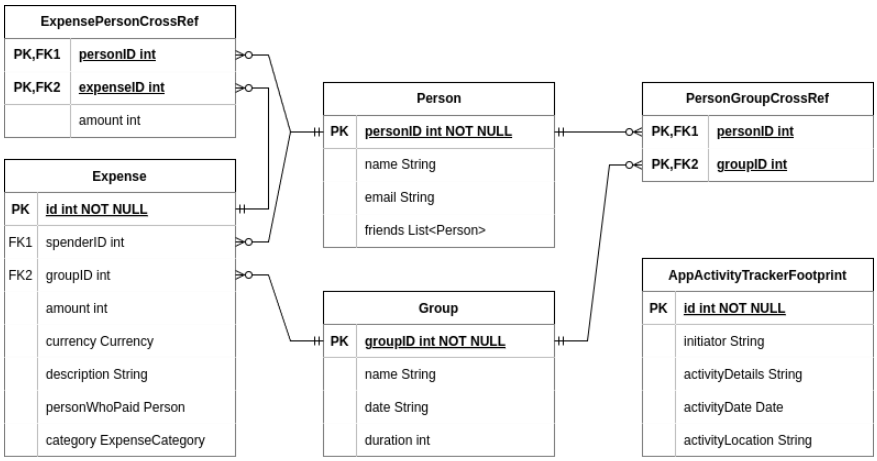
```
1 | public ScanEntry(String name, int quantity, double  unitPrice, double totalPrice) {}
```

Snippet 14: ScanEntry Aufbau

Im letzten Schritt werden diese Einträge miteinander gepaart und bilden die in Snippet 14 definierten „ScanEntry“ Objekte. Diese beinhalten im weiteren Prozess alle relevanten Daten, welche dem Benutzer der Anwendung präsentiert werden müssen, sowie die für die Berechnung der Bilanz erforderlichen Daten

3.5 Modellierung (Dennis)

Das Entity-Relationship Modell (ERM) der App Bananasplit umfasst sechs zentrale Tabellen, die miteinander in Beziehung stehen und verschiedene Aspekte der Anwendung abdecken.



Die Tabelle „Expense“ enthält Informationen zu den finanziellen Ausgaben innerhalb der App. Sie hat den Primärschlüssel *id*, der jede Ausgabe eindeutig identifiziert. Zusätzlich gibt es zwei Fremdschlüssel: *spenderID*, der auf die *personID* in der Tabelle *Person* verweist, und *groupID*, der auf die *groupID* in der Tabelle *Group* verweist. Dadurch wird jede Ausgabe einer bestimmten Person und einer bestimmten Gruppe zugeordnet.

Die Tabelle „Person“ speichert Informationen über die Nutzer der App. Der Primärschlüssel *personID* identifiziert jede Person eindeutig. Diese Tabelle bildet die Grundlage, auf die sich die Fremdschlüssel in anderen Tabellen beziehen.

Die Tabelle „ExpensePersonCrossRef“ dient als Verknüpfungstabelle, um die Beziehung zwischen Personen und Ausgaben zu verwalten. Sie enthält zwei Fremdschlüssel: *personID*, der auf die *personID* in der Tabelle *Person* verweist, und *expenseID*, der auf die

id in der Tabelle *Expense* verweist. Dies ermöglicht es, beliebig viele Personen mit einer Ausgabe zu verknüpfen, was nützlich ist, um zu erfassen, wer an welcher Ausgabe beteiligt ist.

Die Tabelle „Group“ enthält Informationen über die verschiedenen Gruppen in der App, wie z.B. Freundesgruppen oder WGs. Der Primärschlüssel *groupID* identifiziert jede Gruppe eindeutig. Die Gruppenstruktur hilft dabei, Ausgaben und Aktivitäten zu organisieren und den Überblick zu behalten, wer Teil welcher Gruppe ist.

Die Tabelle „PersonGroupCrossRef“ verwaltet die Mitgliedschaften von Personen in Gruppen. Sie enthält zwei Fremdschlüssel: *personID*, der auf die *personID* in der Tabelle *Person* verweist, und *groupID*, der auf die *groupID* in der Tabelle *Group* verweist. Dies ermöglicht eine flexible Zuordnung von Personen zu unterschiedlichen Gruppen.

Die Tabelle „AppActivityTrackerFootprint“ speichert alle Aktivitäten der Nutzer in der App. Sie enthält den Primärschlüssel *id*, der jede Aktivität eindeutig identifiziert. Diese Tabelle steht jedoch in keiner direkten Beziehung zu den anderen Tabellen, sondern dient zur separaten Aufzeichnung der Nutzeraktivitäten.

3.6 Herausforderungen und Lösungsansätze

3.6.1 Entitätserkennung und Zuweisung (Arpad)

Kassenbelege weisen häufig ein unterschiedliches Format auf. Selbst bei Geschäften oder Dienstleistern derselben Firma können diese Belege unterschiedlich formatiert sein und verschiedene, für die Scan-Funktionalität irrelevante Informationen enthalten. Der aus dem Bild extrahierte Text war sehr genau und wies nur wenige Fehler auf. Allerdings gab es aufgrund des variierenden Aufbaus keine Konsistenz in der Strukturierung des erkannten Textes.

Obwohl Positionen auf dem Kassenbeleg in der Regel mit dem Preis in einer Linie stehen, konnte die Texterkennung zwar die richtigen Zahlen und Produkte extrahieren, jedoch fehlte eine konsistente Struktur. So konnte es vorkommen, dass in der Zeichenkette drei Produkte aufeinander folgten, gefolgt von neun Zahlen. Dieses Ergebnis variierte selbst bei identischen Kassenbelegen.

Die Kategorisierung der Entitäten durch die Cloud Natural Language API konnte nicht alle Produkte korrekt erkennen und lieferte bei den Tests unterschiedliche Ergebnisse. Ein Produkt wie Frischkäse wurde beispielsweise oft nicht als „CONSUMER_GOOD“ eingestuft, sondern einer anderen Kategorie zugeordnet.

Die Variabilität sowohl bei der Texterkennung als auch bei der Einstufung führte zu scheinbar zufälligen Ergebnissen, wodurch die Zuordnung von Produkten oder Dienstleistungen mit dem jeweiligen Preis eine große Herausforderung darstellte.

Es wurde daher entschieden, den Stückpreis sowie die Quantität in der ersten Iteration nicht zu berücksichtigen. Durch die scheinbar zufällige Anordnung im erkannten Text war auch eine konsistente Zuweisung von Preis und Produkt nicht möglich.

3.6.2 PayPal SDK (Arpad)

Die Schnittstellen rund um Android entwickeln sich rasant und werden schnell von neueren Versionen überholt. An der PayPal SDK für Android wurden in den letzten Jahren viele Änderungen vorgenommen. Es existieren zahlreiche veraltete Versionen, von denen einige nicht mehr unterstützt werden und andere als deprecated markiert sind. Die in Snippet 5 vorgestellte Funktionalität beruht auf einer älteren Version der PayPal SDK. Die in Zeil 23 stehende ".capture()" Funktion ist als deprecated markiert. Jedoch bietet die offizielle PayPal SDK zum Zeitpunkt dieser Ausarbeitung keine geeignete Alternative für Java-basierte Applikationen innerhalb der PayPal SDK [@PP].

Derzeit ist nur die Dokumentation für Kotlin verfügbar, sowohl in der offiziellen Dokumentation als auch auf der GitHub-Seite. Java und Kotlin sind miteinander kompatibel, sodass es möglich gewesen wäre, die Integration in Kotlin durchzuführen. Aufgrund der Aufgabenstellung, ein Android-Projekt in Java zu erstellen, wurde dieser Ansatz jedoch nicht weiterverfolgt.

Eine weitere Möglichkeit bietet PayPal durch die Verwendung von Braintree. Dies wurde aufgrund der benötigten Server-Backend-Architektur und Nutzungsgebühren nicht weiter berücksichtigt.

3.6.3 Android Studio (Arpad)

Obwohl die Android Studio IDE ein mächtiges Werkzeug ist, das Entwicklern zahlreiche Funktionen für die Entwicklung von Android-Applikationen bietet [Kapitel 2.2.2], gibt es auch Herausforderungen, die die Effizienz beeinträchtigen können. In einigen Fällen treten Fehlermeldungen auf, die irreführend sind und auf andere Fehlerquellen verweisen. Dies führte oft zu langen, erfolglosen Fehlersuchen, da der Code in diesen Fällen keine tatsächlichen Fehler enthielt.

Neustarts und das Löschen des Caches der IDE wurden zur Routine, um diese Probleme zu beheben. Obwohl diese Maßnahmen die Fehler oft behoben, verlangsamten sie den Entwicklungsprozess erheblich. Die Fehler verschwanden nach diesen oder anderen

scheinbar unzusammenhängenden Lösungen, was darauf hinweist, dass die Ursachen der Probleme nicht immer offensichtlich oder logisch nachvollziehbar waren.

3.7 Anwendungsszenarien (Dennis)

Sobald die Bananasplit-App sich erfolgreich etabliert hat, sehen wir die folgenden Anwendungsszenarien für unsere Nutzer.

Wenn eine Gruppe von Freunden gemeinsam verreist, können verschiedene Ausgaben anfallen, wie Unterkunft, Verpflegung, Transport und Aktivitäten. Anstatt jedes Mal Geld hin- und herzuschieben, kann jede getätigte Ausgabe in Bananasplit eingetragen werden. Die App berechnet dann automatisch, wer wie viel zu bezahlen hat, was zu einer fairen Aufteilung der Reisekosten führt.

In einer Wohngemeinschaft (WG) können verschiedene gemeinsame Ausgaben entstehen, wie Miete, Nebenkosten, Internet oder Putzmittel. Mit Bananasplit kann jedes WG-Mitglied seine jeweiligen Kosten eintragen. Am Monats- oder Quartalsende zeigt die App, wer wem wie viel schuldet, wodurch Streitigkeiten und Verwirrungen vermieden werden. Auch Paare, die ihre Finanzen teilweise gemeinsam verwalten, finden Bananasplit nützlich. Sei es für Haushaltskosten, gemeinsames Essen oder Urlaubsbuchungen – die App ermöglicht eine klare Aufteilung und Nachverfolgbarkeit der Ausgaben.

Bei gelegentlichen Gruppenaktivitäten wie Abendessen, Freizeitparks oder Konzertbesuchen kann Bananasplit die Abrechnung erleichtern. Eine Person übernimmt die Zahlung und trägt die Kosten in die App ein, die dann den Anteil jeder Person berechnet. Auch bei geschäftlichen Reisen oder gemeinsamen Mittagessen unter Kollegen kann Bananasplit hilfreich sein.

Innerhalb von Familien können Ausgaben wie Geburtstagsfeiern, Familientreffen oder gemeinsamer Urlaub mit Bananasplit organisiert werden.

In allen diesen Szenarien bietet Bananasplit klare Vorteile in der Kostenverteilung, Transparenz und Stressreduktion, indem es eine einfache und faire Methode zur Verwaltung gemeinschaftlicher Finanzen bereitstellt.

4 Zusammenfassung und Fazit

4.1 Zusammenfassung

4.2 Ausblick (Dennis)

Der Ausblick für die Weiterentwicklung der aktuell existierenden App ist vielversprechend. Derzeit kann die Anwendung nur von einem einzigen Nutzer verwendet werden, was ihre Einsatzmöglichkeiten einschränkt. In der finalen Version der App ist ein Server-Backend geplant, welches die gleichzeitige Nutzung durch mehrere Anwender ermöglicht. Dies wird nicht nur die Skalierbarkeit der App erheblich verbessern, sondern auch neue Funktionen und erweiterte Interaktionen, wie beispielsweise Freundesanfragen oder Gruppeneinladungen, zwischen den Nutzern erlauben. Die Implementierung des Server-Backends ist ein entscheidender Schritt, um die App zukunftssicher zu machen und sie für ein breiteres Publikum attraktiv zu gestalten.

4.3 Fazit

4.4 Verwendung von Künstlicher Intelligenz (Arpad)

Der Einsatz von Künstlicher Intelligenz ist in den letzten Jahren in vielen Bereichen stark gestiegen. Unter anderem in der Softwareentwicklung. Werkzeuge wie ChatGPT oder Gemini sind heutzutage nicht mehr aus dem Arbeitsumfeld wegzudenken. Unter korrektem Einsatz können sie Entwickler bei ihrer Arbeit unterstützen und den Entwicklungsprozess beschleunigen und unterstützen. Es ist zu bedenken, dass diese mächtigen Werkzeuge viele Fehler machen und die Erfahrung von Entwicklern benötigen, um die Antworten korrekt auszuwerten und potenziell umzusetzen.

Während der Ausarbeitung dieses Projekts kamen Tools wie ChatGPT, Google Gemini und GitHub Copilot zum Einsatz.

GitHub Copilot bietet in der IDE live Code Vorschläge. Es wurde als ein autocomplete Feature verwendet, um Methoden oder Variablennamen nicht jedes Mal austippen zu müssen. Es ergänzt die Parameter und einige Methoden Felder automatisch, um die Tipparbeit zu minimieren. Es wurde auch verwendet bei der Generierung der Javadoc Kommentare, um Methoden und Parameter automatisch zu beschreiben.

ChatGPT und Gemini wurden genutzt bei der Erarbeitung der Komplexen verwendeten APIs (Siehe Kapitel 2.4) um in der kurzen Zeitspanne des Projekts einen Einblick über

die grundlegenden Funktionen und Verwendung zu bekommen und diente als eine ergänzende Antwortquelle zu den jeweiligen Dokumentationen, die oft Fragen aufwarfen.

Zur Analyse der berüchtigt langen Java Fehlermeldungen kamen diese Tools auch zum Einsatz, um Schneller potenzielle Fehlerquellen in der komplexen Android Architektur zu finden.

Code, welcher von den KI-Quellen übernommen wurde, wurde an den entsprechenden Stellen gekennzeichnet

Bei der Erstellung dieses Berichtes wurde KI verwendet, um Sprachliche und grammatikalische Fehler zu korrigieren. Alle Gedanken und Aussagen in den Textabschnitten wurden von uns formuliert, basierend auf dem Wissen aus den Vorlesungen und Quellen. Sie wurden lediglich auf sprachliche und grammatikalische Korrektheit von ChatGPT und Gemini überprüft.

5 Referenzen

Web-Seiten zuletzt am 25.07.2024 abgerufen.

[@Sta] Market share of mobile operating systems worldwide from 2009 to 2024, by quarter, https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

[@DevA] Android for Developers, https://developer.android.com/

[@AS] Android Studio, https://developer.android.com/studio

[@JDK] Java versions in Android builds, https://developer.android.com/build/jdks

[@DrR] Drawable resources, https://developer.android.com/guide/topics/resources/drawable-resource

[@AL] Handling Android App Links, https://developer.android.com/training/app-links

[@PP] PayPal Android SDK, https://github.com/paypal/paypal-android/

[@CXS] Camera Samples using CameraX, https://github.com/android/camera-samples/tree/main

[@CX] CameraX, https://developer.android.com/jetpack/androidx/releases/camera

[@MLK] Recognize text in images with ML Kit on Android, https://developers.google.com/ml-kit/vision/text-recognition/v2/android

[@GCD] Google Cloud Documentation, https://cloud.google.com/docs

[@VB] ViewBinding, https://developer.android.com/topic/libraries/view-binding

[@PPD] PayPal Android SDK Documentation, https://developer.paypal.com/docs/checkout/advanced/android/

[@BT] Integration von PayPal mittels Brainstree, https://developer.paypal.com/braintree/articles/guides/payment-methods/paypal/overview

[@Cnct] Connect to a Network, https://developer.android.com/develop/connectivity/network-ops/connecting

[@VM] ViewModel overview | Android Developers, https://developer.android.com/topic/libraries/architecture/viewmodel#java

[@Shn] Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., & Diakopoulos, N. (2016). Designing the user interface: strategies for effective human-computer interaction. Pearson

Eidesstattliche Erklärung

Hiermit erkläre ich/ Hiermit erklären wir an Eides statt, dass ich/ wir die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt habe/ haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Osnabrück, den 25.7.24 ... .
Ort, Datum Arpad Horvath

Osnabrück, den 17.09.24 .....
Ort, Datum Dennis Brockmeyer

Urheberrechtliche Einwilligungserklärung

Hiermit erkläre ich/ Hiermit erklären wir, dass ich/wir damit einverstanden bin/sind, dass meine/ unsere Arbeit zum Zwecke des Plagiatsschutzes bei der Fa. Ephorus BV bis zu 5 Jahren in einer Datenbank für die Hochschule Osnabrück archiviert werden kann. Diese Einwilligung kann jederzeit widerrufen werden.

....
Ort, Datum Unterschrift

Hinweis: Die urheberrechtliche Einwilligungserklärung ist freiwillig.