Dokumentation ProductAR

Maximilian Rehberger

July 29, 2019

1 Inhaltsverzeichnis

2 Einleitung

2.1 Zweck

Produkte können zum Beispiel beim Einkaufen mit dem Smartphone gescannt werden und erkannt werden. Informationen werden angezeigt wie zum Beispiel Bilder oder ein Preisvergleich. Mithilfe der App soll man einen Barcode einscannen können und Informationen zu den Produkten erhalten. Weiterhin kann der Nutzer ein Produkt in Augmented Reality (AR) testen und sieht somit wie es in Wirklichkeit aussehen wird, wenn er es kaufen würden.

3 Allgemeine Übersicht

3.1 Beschreibung Ausgangssituation

Es gibt bereits viele Shopping-Apps wie zum Beispiel Ikea, H&M oder S'Oliver. Das Problem ist, dass jeder am Ende für jedes Geschäft eine eigene App auf dem Smartphone hat. Diese App soll die Möglichkeiten geben mehrere unterschiedliche Produkte in einer App zu speichern und zu verwalten. Also eine App für alle Produkte.

3.2 Produkteinsatz

Die App kann zum Beispiel als Einkaufsliste oder Wunschliste für Produkte eingesetzt werden. Darüber hinaus bieten sich noch viele weitere Möglichkeiten.

3.3 Produktumfeld

Die App wird hauptsächlich im privaten Umfeld umgesetzt, beim Einkaufen in Geschäften oder Online-Einkauf.

3.4 Produktfunktionalität

Scannen von Produkten, Informationen zu Produkten, Preisvergleich, Bilder hochladen für Produkte, Produkte in AR testen.

3.5 Personas

3.5.1 Nutzer

Die Hauptzielgruppe, für welche die App interessant ist, sind "Käufer" und "Verkäufer". Deshalb sind diese für Nutzer noch einmal in einer Unterkategorie zusammengefasst.

3.5.1.1 Käufer

Nutzer 1:
Name: Jonas
Alter: 21 Jahre
Status: ledig
Eigenschaften: zurückhaltender & stiller Typ
Beschäftigung: E-Commerce Student
Hobbys: Zocken, Serien schauen

Jonas beginnt gerade sein E-Commerce Studium in Würzburg. Mit viel Glück hat er eine Wohnung in der Stadt gefunden. Er befindet sich gerade im Umzug. Das Geld, welches er sich in den Ferien erarbeitet hat, nimmt er um die Möbel für seine neue Wohnung einzukaufen. Um nicht ständig alle Produkte aufschreiben zu müssen, möchte er eine App benutzen, mit der sich Produkte einfach abspeichern lassen. Außerdem möchte er

Fotos von den Möbeln machen um sie seinen Eltern zeigen zu können. Er fände es cool, wenn er die Möbel schon gleich bei sich in der Wohnung mit Augmented Reality (AR) ausprobieren kann.

Nutzer 2:
Name: Larissa
Alter: 24 Jahre
Status: In einer Beziehung
Eigenschaften: extrovertiert & unternehmenslustig
Beschäftigung: BWL Studentin
Hobbys: Shopping, Fotografie, Mit Freunden treffen

Larissa muss immer das neuste haben. Egal ob Kleidung, Schmuck oder Elektronik. Sie möchte ungern das Produkt von gestern besitzen, sondern immer das neueste, damit Sie es gleich ihren Freunden zeigen kann. Sie findet eine App sehr praktisch mit der sie Produkte abspeichern kann um sie hinterher wieder zu finden. Außerdem schaut sie sich gerne Fotos von anderen Nutzern zu bestimmten Produkten an, sowie deren Bewertungen um hinterher zu wissen, wie diese die Produkte finden.

Nutzer 3:
Name: Thorsten
Alter: 35 Jahre
Status: ledig
Eigenschaften: vielbeschäftigt, hat kaum Zeit
Beschäftigung: Geschäftsmann
Hobbys: Keine

Thorsten hat gar keine Zeit um sich stundenlang im Internet das beste Angebot für zum Beispiel ein neues Handy herauszusuchen. Er ist vielbeschäftigt und möchte daher eine App benutzen, welche ihm anzeigt, bei welchem Geschäft das Produkt gerade am günstigsten ist. Wenn ein Produkt stark reduziert ist beziehungsweise im Angebot ist möchte er automatisch eine Benachrichtung erhalten.

3.5.1.2 Verkäufer

Nutzer 4:
Name: Lukas
Alter: 32 Jahre
Status: ledig
Eigenschaften: handwerklich begabt
Beschäftigung: Besitzer eines Möbelgeschäftes in Würzburg
Hobbys: Möbel entwerfen und zusammen bauen, Grillen mit Freunden

In den letzten Jahren läuft das Geschäft, welches Lukas betreibt immer schlechter. Die Leute bestellen meistens im Internet, oder kaufen beim Konkurrenten IKEA ein. Dabei sind seine Möbel selbst nach bester Handwerkskunst gefertigt und kosten auch nicht viel mehr im Vergleich zu Markenmöbeln. Lukas muss sich etwas inovatives einfallen lassen. Er würde gerne seine Möbel online anbieten, damit mehr Leute bei Ihm einkaufen. Mithilfe der App kann er seine Möbel mit Bildern integrieren, damit die Leute sie in der App finden können. Er hat die Möglichkeit ein individuelles 3D Model für seine Möbel zu erstellen, damit man diese in der App in Augemented Reality (AR) testen kann.

3.5.2 Content-Manager

Als Content-Manager möchte ich Updates (und Change-logs) erstellen können um die App immer auf den aktuellsten Stand zu halten, um zum Beispiel neue Produkte oder neue 3D Modelle hinzufügen zu können. Auch soll man in der Lage sein, die Sperrwörter liste aktualisieren zu können, um zu verhindern das illegale Produkte erstellt werden können.

3.5.3 Admin

Als Admin möchte ich Analysen erstellen können, damit ich zum Beispiel ermitteln kann, wie viele Nutzer die App benutzt haben um ein Produkt für IKEA hinzuzufügen. Außerdem möchte ich auf die Datenbank zugreifen können, damit ich neue Tabellen Einträge erstellen kann oder um Fehler beheben zu können.

3.6 Anforderungsanalyse

Die App muss die Produkte anhand

- Eines Barcodes (Kamera) erkennen und mithilfe einer Tabelle zuordnen können
- Ihrer Form entsprechend erkennen und zuordnen können (konzeptionell). Diese Funktion wurde nicht in der fertigen App umgesetzt.

Unterscheiden können zwischen den Produkten
- zum Beispiel zwischen einer Banane und einem Apfel
- Einfache Klassifizierer oder KI

Möglichkeit zum Testen von Produkten innerhalb der App
- Hierbei wird ein AR-Objekt erzeugt

3.6.1 Anforderungen der Personas

Die Anforderungsanalyse wird nun ergänzt mit den Anforderungen, welche die Personas sich von der App vorgestellt haben.

- Möglichkeit zum Speichern von Produkten in der App
- Fotos von Produkten erstellen
- Fotos von anderen zu einem Produkt sehen können
- Produkt bewerten und Bewertungen einsehen können
- Produkte in Augmented Reality (AR) testen
- Preisvergleich (bei welchem Geschäft zurzeit am günstigsten)
- Benachrichtigungen, wenn Produkte im Angebot sind
- Die Möglichkeit für Verkäufer Produkte einzustellen
- Die Möglichkeit für Verkäufer eigene 3D Modelle hochzuladen

4 Architekturkonzept und Entwurf

4.1 Ursprüngliches Architekturkonzept

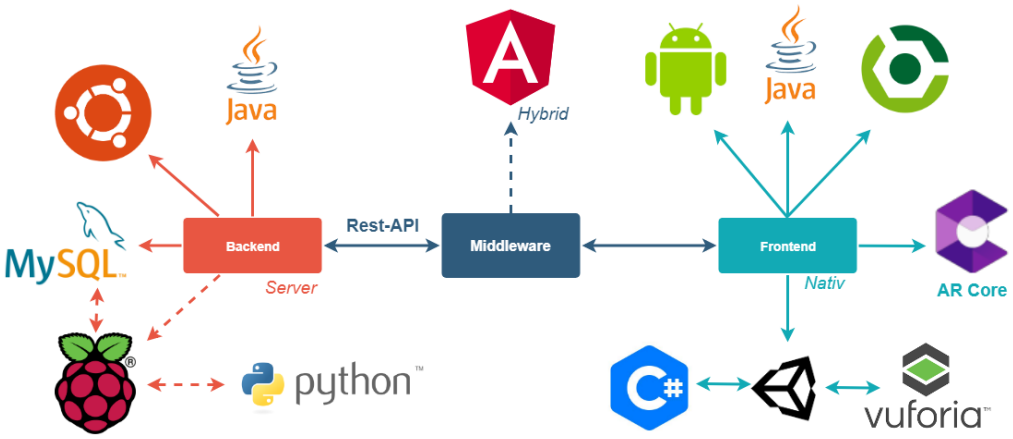

Figure 1: Ursprüngliches Architekturkonzept

4.2 Aktualisiertes Architekturkonzept

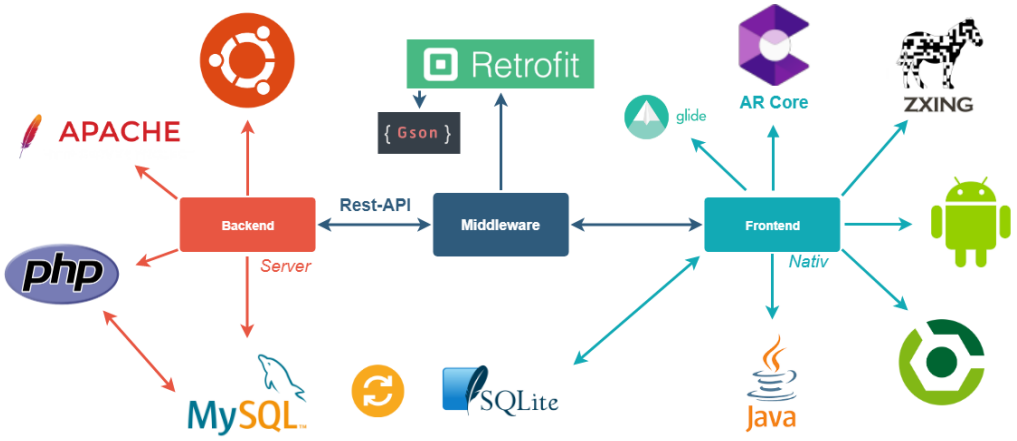

Figure 2: Aktualisiertes Architekturkonzept

4.3 Anfängliche Skizze Datenbankentwurf

4.3.1 MySQL Datenkbank (Remote)

Ursprünglich war geplant, dass die Daten ausschließlich auf dem Server in einer MySQL Datenbank gespeichert werden.

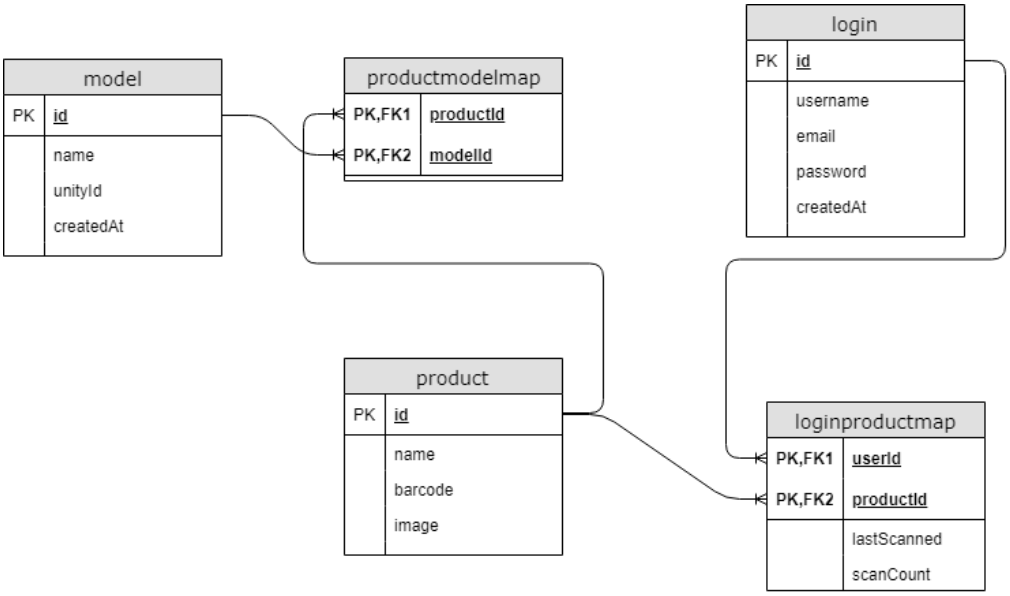

Figure 3: Anfängliche Skizze Datenbankentwurf

4.4 Anfängliche Skizze Java Klassen

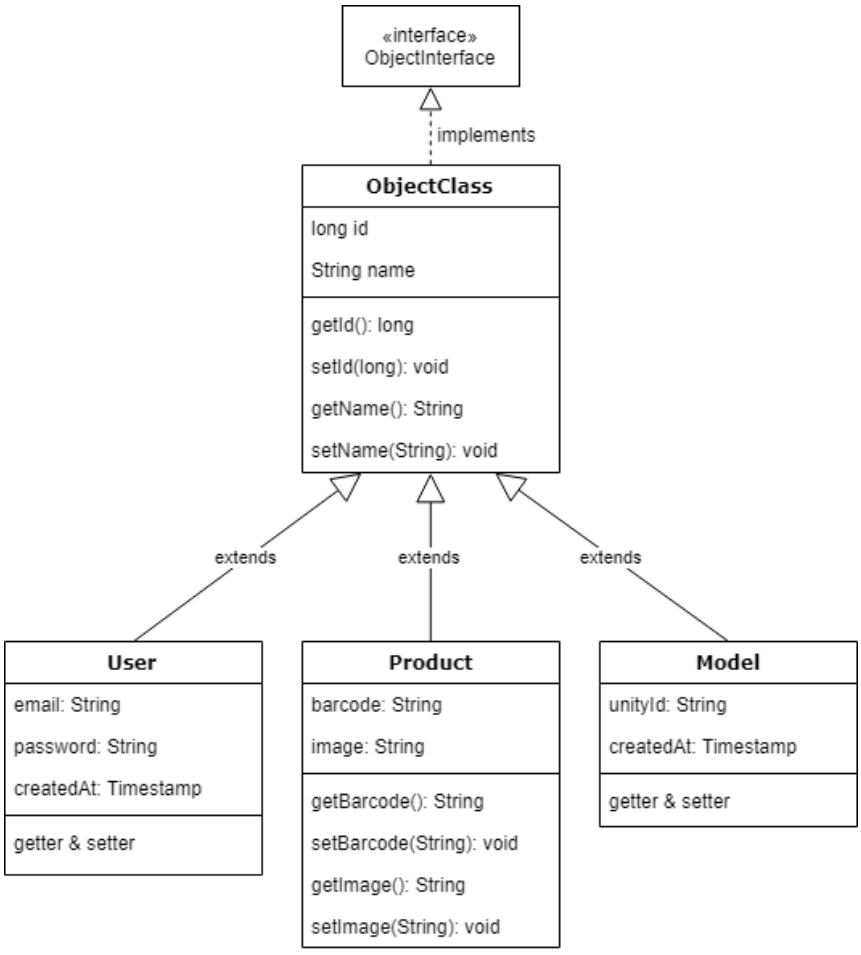

Figure 4: Anfängliche Skizze Datenbankentwurf

4.5 Endgültige Skizze Datenbankentwurf

4.5.1 SQLite Datenbank (Lokal)

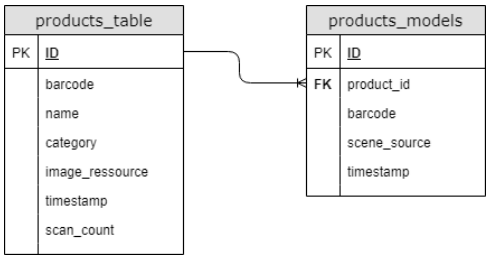

Figure 5: Skizze Datenbankentwurf: SQLite

4.5.2 MySQL Datenbank (Remote)

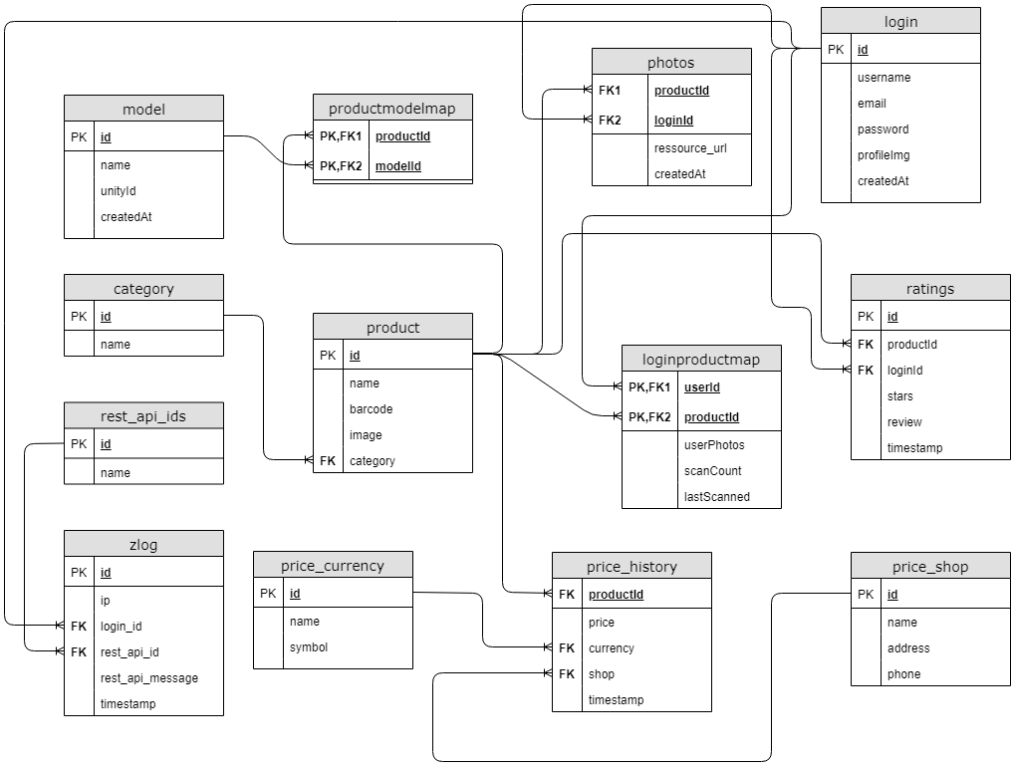

Figure 6: Aktualisierte Skizze Datenbankentwurf: MySQL

4.6 Endgültige Skizze Java Klassen

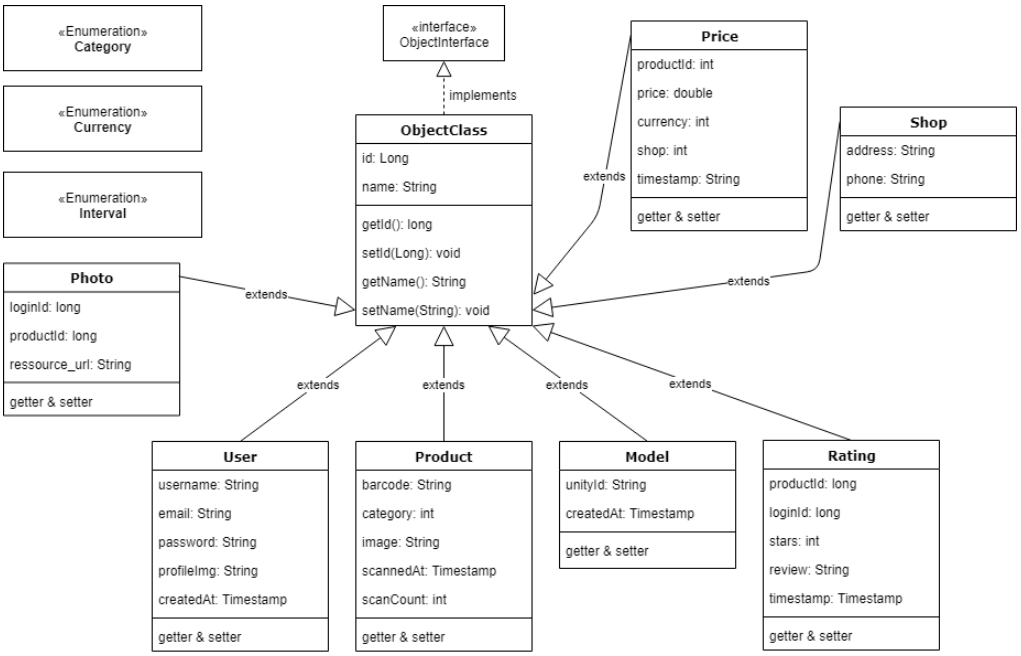

Figure 7: Aktualisierte Skizze: Java Klassen

4.7 Übersicht Backend Server

Der Backend Server ist ein gemieteter Server von Hosteurope.
Produktbezeichnung: "Virtual Server Linux Advanced 8.2".
Dieser hat folgende Linux Version installiert: Ubuntu 16.04.6 LTS.

Die Technischen Spezifikationen lauten wie folgt:

4 virtuelle Kerne
6 GB RAM
200GB SSD

Es handelt sich hierbei um einen virtuellen Server, das bedeutet, dass sich der Server mit anderen "Containern" die Hardware eines realen Servers teilen.

Der Server hat eine eigene Domain: www.nimoo.de.

4.8 Übersicht REST API

Die Rest Schnittstelle wurde mit PHP auf dem Webserver umgesetzt welcher vom Backend Server bereits zur Verfügung gestellt wurde. Für jede Ressource existiert ein Pfad, mit entsprechender PHP Datei.

Der Hauptpfad für die App auf dem Webserver: "https://www.nimoo.de/apps/productar"

Folgende Pfade existieren auf dem Webserver:

../products/
../products/images/
../products/photos/
../products/prices/
../products/ratings/
../users/
../users/images
../models/

4.9 Technische Entscheidungen

4.9.1 Warum Android?

Die Entscheidung, die App für Android zu entwickeln wurde getroffen, da Android zumindest in Deutschland einen höheren Marktanteil besitzt als iOS. Vor allem die Studenten der Fakultät Informatik und Wirtschaftsinformatik (FIW) und in der Vertiefung Mobile Solutions benutzen mehrheitlich Android Smartphones. Ein weiterer Grund ist, dass Android Java basiert ist und dafür sehr gut geeignet ist, wenn bereits fortgeschrittene Erfahrungen mit der Programmiersprache Java gegeben sind. Weiterhin gibt es beim Entwickeln keine Mehrkosten, da es bereits viele Open-Source Erweiterungen (Bibliotheken) gibt und Anleitungen, die das Entwickeln weiter vereinfachen.

4.9.2 Welche Androidversion?

Als minimal unterstützte Android Version (minSdkVersion) für die App musste die Api 24 (Android 7) verwendet werden. Dies liegt daran, dass die AR Funktionialität mit der Google AR Core Erweiterung erst ab Android Version 7 (Api 24) unterstützt wurde und alle vorherigen Versionen keine Unterstützung haben. Dies hat den Nachteil, dass nur ca. 37,1 % aller Android Geräte unterstützt werden im Vergleich zu den 95,3 % die mit Android 4.4 (Api 19) unterstützt würden.

4.9.3 Welche Entwicklungsumgebung?

Zum Entwickeln der App wurde hauptsächlich die Entwicklungsumgebung von Android Studio und IntelliJ genutzt.

4.9.4 Wieso Google AR Core?

Googles neuestes Framework für Augmented Reality Anwendungen heist "AR Core". Im Vergleich zu einer AR Anwendung mit Unity lässt es es sich sehr einfach in die App integrieren (Als Fragment oder View in der Aktivity). Weiterhin lassen sich Modelle (.OBJ) sehr einfach mit dem Sceneform Plugin einbinden und bearbeiten.

4.9.5 Wieso einen Barcode-Scanner?

Für Produkte, welche in einem lokalen Geschäft angeboten werden, wird meistens ein numerischer Barcode verwendet, da dieser leicht an der Kasse über das Band gezogen und gescannt werden kann um so erfasst zu werden.

4.9.6 Wieso eine MySQL Datenbank?

Zum einen war die MySQL Datenbank ebenfalls schon auf dem Backend Server aufgesetzt, somit war keine weitere Konfiguration notwendig. Weiterhin ist es sehr einfach eine Datenbank mit SQL zu erstellen und Abfragen durchzuführen.

4.9.7 Wieso eine REST API?

Die Rest API ist die Schnittstelle zwischen der App und der Datenbank auf dem Server. Diese wird benötigt, da man aus Sicherheitsgründen keine direkte Verbindung zwischen App und Datenbank zulassen darf.

4.9.8 Vergleich mit Alternativlösungen

4.9.8.1 Firebase von Google .

Die Backendlösung von Google ist "FireBase" und wäre erheblich einfacher umzusetzten und hätte ebenfalls den Vorteil, dass kein externer Server benötigt wird. Warum wurde diese Lösung in diesem Projekt jedoch nicht verwendet? Die Datenbank enthält sensible Daten, wie zum Beispiel Nutzerdaten. Diese sollen nicht an Google gesendet werden.

4.9.8.2 Alternative Datenbankmodelle .

PostgreSQL und MongoDB.

5 Technische Dokumentation

Die Dokumentation der einzelnen Java Klassen befindet sich im generierten JavaDoc Verzeichnis. Die nachfolgende Dokumentation wurde aus den JavaDoc Kommentaren übernommen. Bestimmte Klassen können doppelt vorkommen.

5.1 Android Manifest

5.2 Java Interfaces

5.2.1 ObjectInterface

Das Interface "ObjectInterface" definiert die Vorgaben, welches ein Objekt erfüllen muss. Ein Objekt benötigt eine id als eindeutigen Identifizierer und einen Namen. Entsprechende Getter und Setter sind hier definiert.

5.2.2 ScanResultReceiver

Das Interface "ScanResultReceiver" definiert die Methoden, welche nach dem Scannen eines Barcodes ausgeführt werden.

Methode scanResultData(NoScanResultException noScanData)
Die Methode "scanResultData" wird aufgerufen, wenn das Scannen des Barcodes fehlgeschlagen ist.

Methode scanResultData(java.lang.String codeFormat, java.lang.String codeContent)
Die Methode "scanResultData" wird nach dem erfolgreichen Scannen des Barcodes aufgerufen.

5.2.3 IRetrofitCRUD

Das Interface "IRetrofitCRUD" definiert die Methoden, welche aufgerufen werden um über Retrofit Anfragen an den Server zu stellen.

5.2.4 JsonPlaceHolderApi

Das Interface "JsonPlaceHolderApi" ist die direkte Schnittstelle zwischen Retrofit und dem Zielserver. Verwendete HTTP Verbs: GET und POST.

5.3 Java Klassen

5.3.1 Objekt Klassen

5.3.1.1 Object Class (Abstract) Die Klasse "ObjectClass" ist eine abstrakte Klasse, welche die benötigten Methoden für ein Objekt implementiert.

5.3.1.2 Product Die Klasse "Product" stellt die Objektklasse für ein Produkt dar. Ein Produkt ist gleichzeitig ein Objekt. Hier werden wichtige Methoden und Konstruktoren implementiert. Die Werte können über Getter und Setter Methoden abgefragt werden.

5.3.1.3 User Die Klasse "User" stellt die Objektklasse für einen Benutzer dar. Ein Benutzer ist gleichzeitig ein Objekt. Hier werden wichtige Methoden und Konstruktoren implementiert. Die Werte können über Getter und Setter Methoden abgefragt werden.

5.3.1.4 Model Die Klasse "Model" stellt die Objektklasse für ein (AR) Model dar. Ein Model ist gleichzeitig ein Objekt. Hier werden wichtige Methoden und Konstruktoren implementiert. Die Werte können über Getter und Setter Methoden abgefragt werden.

5.3.1.5 Photo Die Klasse "Photo" stellt die Objektklasse für ein Foto dar. Ein Foto ist gleichzeitig ein Objekt. Hier werden wichtige Methoden und Konstruktoren implementiert. Die Werte können über Getter und Setter Methoden abgefragt werden.

5.3.1.6 Price Die Klasse "Price" stellt die Objektklasse für einen Preis dar. Ein Preis ist gleichzeitig ein Objekt. Hier werden wichtige Methoden und Konstruktoren implementiert. Die Werte können über Getter und Setter Methoden abgefragt werden.

5.3.1.7 Shop Die Klasse "Shop" stellt die Objektklasse für einen Shop dar. Ein Shop ist gleichzeitig ein Objekt. Hier werden wichtige Methoden und Konstruktoren implementiert. Die Werte können über Getter und Setter Methoden abgefragt werden.

5.3.1.8 Category (Enum) Die Enumklasse "Category" beinhaltet die Produktkategorien. Es gibt folgende Kategorien: Accessoires, Auto, Baumarkt, Beauty, Bücher, Computer, Drogerie, Elektronik, Filme, Garten, Haushalt, Kleidung, Lebensmittel, Möbel, Musik, Schuhe, Serien, Spiele, Spielzeug, Sport.

5.3.1.9 Currency (Enum) Die Enumklasse "Currency" beinhaltet die aktuell unterstützten Währungen. In diesem Fall: Dollar und Euro.

5.3.1.10 Interval (Enum) Die Enumklasse "Interval" beinhaltet die Möglichkeiten für ein Updateinterval der Benachrichtigungen. Folgende Intervalle sind für Benachrichtigungen möglich: Täglich, Wöchentlich, Monatlich.

5.3.2 Aktivity Klassen

5.3.2.1 MainActivity Die Klasse "MainActivity" wird beim Starten der App ausgeführt, direkt nach dem "SplashScreen". Es können Barcodes gescannt und danach das Ergebnis angezeigt werden.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Hier werden Werte initialisiert, zum Beispiel TextViews oder Buttons. Es wird eine Datenbankverbindung zur lokalen SQLite Datenbank initialisiert. Ein OnClickListener wird für den Button "btn_scan_now" initialisiert. Für Android Versionen größer 23 (ab 24) wird ein NetworkMonitor Receiver erzeugt.

Methode onCreateOptionsMenu(android.view.Menu menu)
Die Methode "onCreateOptionsMenu" erzeugt das Menü für die AktionsLeiste Es wird zuerst überprüft ob der Nutzer eingeloggt ist. Nutzername und Password werden in einem User Objekt gespeichert. Das Menü "menu_loggedin" wird hier verwendet. Der Nutzername wird in das Feld "action_username" eingetragen und ein OnClickListener erstellt mit der Methode "goToProfile". Für den Logout Button wird ebenfalls ein OnClickListener erstellt, welcher den Nutzer ausloggt. Ist der Nutzer allgemein nicht eingloggt, so wird stattdessen das Standardmenü geladen.

Methode onDestroy()
Die Methode "onDestroy" wird beim verlassen der Activity ausgeführt.

Methode onOptionsItemSelected(android.view.MenuItem item)
Die Methode "onOptionsItemsSelected" wird ausgeführt, wenn ein Menüelement ausgewählt wurde. Je nachdem um welches Element es sich handelt werden unterschiedliche Aktionen ausgeführt. Bei "action_search": Man wird zur Suche weitergeleitet. Bei "action_login": Wenn eingeloggt, dann wird man zum Profil weitergeleitet. Wenn nicht eingeloggt, dann wird man zum Login weitergeleitet. Bei "action_settings": Man wird zu den Einstellungen weitergeleitet. Bei "action_info": Man wird zu den Informationen über die App weitergeleitet. Bei "action_close": Die App wird beendet.

Methode scanNow(android.view.View view)
Die Methode "scanNow" wird ausgeführt, wenn der Button "btn_scan_now" geklickt wurde. Es wird ein ScanFragment erzeugt, welches als nächstes geöffnet wird. Die Kamera wird aktiviert und der Barcode Scanner wird initialisiert.

Methode scanResultData(java.lang.String codeFormat, java.lang.String codeResult)
Die Methode "scanResultData" wird ausgeführt, wenn der Barcode Scanner einen Code erfolgreich gescannt hat. Zuerst wird geprüft ob der barcode existiert (!nullCheck) Als nächstes beginnt die Ladeanimation (loadingStart()) Es wird versucht den Barcode in

eine Long Variable umzuwandeln um zu prüfen ob der Barcode numerisch ist. Wenn keine NumberFormatException abgefangen worden ist wird in der lokalen SQLite Datenbank nach einem Barcode gesucht, welcher schon existiert. Von diesem wird der Name und das Bild benötigt. Wenn kein Barcode lokal existiert, dann wird eine Abfrage mit Retrofit ausgeführt, welche prüft ob ein Produkt mit dem Barcode in der MySQL Datenbank auf dem Server vorhanden ist. Sollte ein Produkt auf dem Server existieren, dann wird es in die Lokale Datenbank übertragen Es wird zusätzlich überprüft ob ein Model zu dem Produkt in der lokalen Datenbank existiert, wenn nicht, dann wird eins vom Server angefragt und in die Datenbank übertragen. Wenn kein Produkt auf dem Server existiert, dann wird die Methode "createNewBarcode" aufgerufen um einen neues Produkt auf dem lokalen Gerät zu erstellen. Wenn der Barcode bereits lokal existiert, wird der Zeitstempel für das Produkt aktualisiert und die Anzahl der Scans um 1 inkrementiert. Außerdem wird das Produkt als "bereit zum Synchronisieren" gekennzeichnet Das Ergebnis für das Bild und den Namen aus der lokalen Datenbank wird nun angezeigt und in die davor vorgesehenen Views geladen. Abschließend wird die Ladeanimation wieder beendet (loadingEnd())

Methode scanResultData(NoScanResultException noScanData)
Für den Fall das der Scan fehlgeschlagen ist.

Methode createNewBarcode(java.lang.String newBarcode)
Die Methode "createNewBarcode" leitet auf die "CreateProductActivity" weiter und übergibt dieser den gescannten Barcode.

Methode goToProfile()
Die Methode "goToProfile" leitet einen zum Nutzerprofil weiter.

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.2 SplashScreen Die "SplashScreen" Activity wird ganz am Anfang gestartet. Es wird ein Drawable angezeigt. Anschließend wird auf die "MainActivity" weitergeleitet.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" initialisiert Variablen und ruft die Methode "scheduleJob" auf. Weiterhin wird ein CountdownTimer eingestellt, welcher beim Ablauf auf die "MainActivity" weiterleitet.

Methode scheduleJob()
Die Methode "scheduleJob" plant einen Job, welcher im Hintergrund ausgeführt werden soll. Dieser ist notwendig um den Nutzer in einem bestimmten Zeitinterval über Neuigkeiten oder Aktualisierungen informieren zu können. In diesem Fall wird der Nutzer über neue Angebote zu seinen Produkten informiert.

Methode cancelJob()
Die Methode "cancelJob" entfernt den geplanten Job wieder.

5.3.2.3 ProductArActivity Die Klasse "ProductArActivity" wird ausgeführt, wenn der Nutzer ein Produkt in AR testen möchte. Das Produkt kann auf eine beliebige gefundene Fläche in AR platziert werden. Wenn kein Produkt zum Testen ausgewählt wurde, dann erscheint ein leerer Einkaufswagen als Model zum Testen.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Wenn ein Barcode von einem Produkt an die Activity übergeben wurde, dann wird das Model aus der lokalen SQLite Datenbank abgefragt, ansonsten wird ein Standardmodel verwendet. Wenn die Einstellung "AR Marker" nicht aktiv ist, dann geht es weiter. Das ArFragment wird initialisiert und es wird ein OnTapArPlaneListener erstellt, welcher ausgeführt wird, wenn man auf eine gefundene AR Fläche tippt. An dieser Stelle wird dann ein Ankerpunkt erzeugt, auf welchen das Model platziert wird.

Methode addModelToScene(com.google.ar.core.Anchor anchor, com.google.ar.sceneform.rendering.ModelRenderable modelRenderable)
Die Methode "addModelToScene" fügt der Scene das Model hinzu.

5.3.2.4 ProductScanActivity Die Klasse "ProductScanActivity" wird ausgeführt, wenn der Nutzer ein Produkt in AR testen möchte. Anders als bei der "ProductArActivity" wird das Produkt nur auf einen vorher generierten QR Code platziert, welcher dem Namen oder den Barcode des Produkts entspricht. Dies geschieht automatisch. Wenn kein Produkt zum Testen ausgewählt wurde, dann erscheint ein leerer Einkaufswagen als Model zum Testen.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Wenn ein Barcode von einem Produkt an die Activity übergeben wurde, dann wird das Model aus der lokalen SQLite Datenbank abgefragt, ansonsten wird ein Standardmodel verwendet. Wenn die Einstellung "AR Marker" aktiv ist, dann geht es weiter. In diesem Fall wird ein CustomArFragment initialisiert und ein OnUpdateListener hinzugefügt.

Methode onUpdate(com.google.ar.sceneform.FrameTime frameTime)
Die Methode "onUpdate" wird aufgerufen, wenn eine Aktualisierung in der AR Scene stattgefunden hat. Für jedes Image Target wird überprüft, ob es in der Scene getrackt wird. Wird ein Image Target getrackt, dann wird überprüft ob der key name des getrackten Images mit denen der festgelegten Image Targets übereinstimmt. Wenn es übereinstimmt, dann wird eine Toast Nachricht angezeigt. Anschließend wird ein Ankerpunkt in der Mitte des Image Targets platziert und die Methode "createModel" aufgerufen und dieser den Ankerpunkt übergeben.

Methode setupDatabase(com.google.ar.core.Config config,

com.google.ar.core.Session session)
Die Methode "setupDatabase" erzeugt die AugmentedImageDatabase, in welcher die Bilder sind, welche in der AR Scene getrackt werden müssen. Es werden 3 Image Targets hinzugefügt. 1. Image Target: QR Code: "fox" 2. Image Target: QR Code: Name vom Produkt 3. Image Target: QR Code: Barcode vom Produkt.

Methode createModel(com.google.ar.core.Anchor anchor)
Die Methode "createModel" erzeugt das Model auf den Ankerpunkt.

Methode placeModel(com.google.ar.sceneform.rendering.ModelRenderable modelRenderable, com.google.ar.core.Anchor anchor)
Die Methode "placeModel" platziert das Model.

5.3.2.5 CaptureActivityPortrait Die Klasse "CaptureActivityPortrait" ist dafür da, dass der Barcode Scanner im Hochkant Format ausgeführt wird und nicht beim Drehen des Devices mitrotiert.

5.3.2.6 LastScannedProductsActivity Die Klasse "LastScannedProductsActivity" zeigt die zuletzt gescannten Produkte der Reihenfolge absteigend an. Es existiert eine "ListView" in der die Objekte geladen werden.

Methode onCreate(@Nullable android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Hier werden wichtige Werte initialisiert zum Beispiel eine ListView. Es wird eine Datenbankabfrage auf die Lokale SQLite Datenbank erzeugt, welche alle lokal gespeicherten Produkte nach Zeitstempel sortiert (neuesten zuerst) wieder zurück gibt. Wenn es keine Produkte gibt, so wird eine TextView "noContentText" sichtbar gemacht. Ansonsten werden die gefundenen Produkte nach und nach erzeugt und einer Liste hinzugefügt. Es wird ein ProductListAdapter mit dieser Liste erzeugt, welcher für die ListView gesetzt wird. Außerdem wird ein OnItemClickListener für jedes Item der ListView erzeugt, welcher auf die "ProductDetailActivity" für das Produkt weiterleitet und dieser den Barcode des Produkts übergibt.

Methode onCreateOptionsMenu(android.view.Menu menu)
Die Methode onCreateOptionsMenu erzeugt das Menü für die AktionsLeiste Es wird zuerst überprüft ob der Nutzer eingeloggt ist. Nutzername und Password werden in einem User Objekt gespeichert. Das Menü "menu_loggedin" wird hier verwendet. Der Nutzername wird in das Feld "action_username" eingetragen und ein OnClickListener erstellt mit der Methode "goToProfile". Für den Logout Button wird ebenfalls ein OnClickListener erstellt, welcher den Nutzer ausloggt. Ist der Nutzer allgemein nicht

eingloggt, so wird stattdessen das Standardmenü geladen.

Methode onOptionsItemSelected(android.view.MenuItem item)
Die Methode "onOptionsItemsSelected" wird ausgeführt, wenn ein Menüelement ausgewählt wurde. Je nachdem um welches Element es sich handelt werden unterschiedliche Aktionen ausgeführt. Bei "action_search": Man wird zur Suche weitergeleitet. Bei "action_login": Wenn eingeloggt, dann wird man zum Profil weitergeleitet. Wenn nicht eingeloggt, dann wird man zum Login weitergeleitet. Bei "action_settings": Man wird zu den Einstellungen weitergeleitet. Bei "action_info": Man wird zu den Informationen über die App weitergeleitet. Bei "action_close": Die App wird beendet.

Methode addNewProduct()
Die Methode "addNewProduct" setzt einen OnClickListener auf den ActionButton "addNewProductActionButton" welcher auf die "CreateProductActivity" weiterleitet.

Methode goToProfile()
Die Methode "goToProfile" leitet einen zum Nutzerprofil weiter.

5.3.2.7 CreateProductActivity Die Klasse "CreateProductActivity" ist dazu da um ein neues Produkt zu erstellen, welches in der lokalen SQLite Datenbank abgespeichert wird.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Zunächst wird geprüft ob ein Barcode mit übergeben wurde. Wenn ein Barcode existiert, dann wird dem EditText "editBarcode" dieser als Text gesetzt. Es werden Werte initialisiert, zum Beispiel Buttons, TextViews oder EditText Felder.

Methode onCreateOptionsMenu(android.view.Menu menu)
Die Methode onCreateOptionsMenu erzeugt das Menü für die AktionsLeiste Es wird zuerst überprüft ob der Nutzer eingeloggt ist. Nutzername und Password werden in einem User Objekt gespeichert. Das Menü "menu_loggedin" wird hier verwendet. Der Nutzername wird in das Feld "action_username" eingetragen und ein OnClickListener erstellt mit der Methode "goToProfile". Für den Logout Button wird ebenfalls ein OnClickListener erstellt, welcher den Nutzer ausloggt. Ist der Nutzer allgemein nicht eingloggt, so wird stattdessen das Standardmenü geladen.

Methode onOptionsItemSelected(android.view.MenuItem item)
Die Methode "onOptionsItemsSelected" wird ausgeführt, wenn ein Menüelement ausgewählt wurde. Je nachdem um welches Element es sich handelt werden unterschiedliche Aktionen ausgeführt. Bei "action_search": Man wird zur Suche weitergeleitet. Bei "action_login": Wenn eingeloggt, dann wird man zum Profil weitergeleitet. Wenn nicht

eingeloggt, dann wird man zum Login weitergeleitet. Bei "action_settings": Man wird zu den Einstellungen weitergeleitet. Bei "action_info": Man wird zu den Informationen über die App weitergeleitet. Bei "action_close": Die App wird beendet.

Methode AddDataListener()
Die Methode "AddDataListener" erstellt einen OnClickListener für den Button "btnAdd". Es werden die Produktdaten in die lokale SQLite Datenbank übertragen. Zuerst wird überprüft ob das Feld für den Barcode leer ist. Ist dies der Fall wird eine aussagekräftige Fehlermeldung in ein TextView geladen. Ansonsten wird als nächstes versucht den Barcode in eine Long Variable umzuwandeln. Dies dient dazu, herauszufinden ob der Barcode numerisch ist. Ist dies nicht der Fall, wird eine aussagekräftige Fehlermeldung in ein TextView geladen. Als nächstes wird überprüft ob der Barcode bereits in der lokalen SQLite Datenbank oder in der Datenbank auf dem Server schon existiert. Ist dies der Fall, wird eine aussagekräftige Fehlermeldung in ein TextView geladen. Ist der Barcode noch nicht vorhanden so geht es weiter. Als nächstes wird überprüft ob der Name des Produkts leer ist. Ist dies der Fall, wird eine aussagekräftige Fehlermeldung in ein TextView geladen. Weiterhin wird überprüft ob der Bild URL leer ist. Ist dies der Fall, wird eine aussagekräftige Fehlermeldung in ein TextView geladen. Sind alle Produktdaten korrekt, dann werden diese in die lokale SQLite Datenbank übertragen. Wenn diese erfolgreich übertragen wurden, werden die Daten mit dem Server synchronisiert. Außerdem wird eine aussagekräftige Toast Nachricht erzeugt. Zum Schluss wird der Nutzer zurück zur "MainActivity" geleitet, falls er von da gekommen ist.

Methode imageUploadListener()
Die Methode "imageUploadListener" setzt einen OnClickListener für den Button "btnImgUpload" und fragt die Erlaubnis für den Dateizugriff auf die Fotogalerie an.

Methode takePhotoListener()
Die Methode "takePhotoListener" setzt einen OnClickListener für den Button "btnTakePhoto" und fragt die Erlaubnis für die Benutzung der Kamera an.

Methode imageUploadListener()
Die Methode "imageUploadListener" setzt einen OnClickListener für den Button "btnImgUpload" und fragt die Erlaubnis für den Dateizugriff auf die Fotogalerie an.

Methode requestCameraPermissions()
Die Methode "requestCameraPermissions" fragt die Erlaubnis für den Zugriff auf die Kamera an. Diese wird benötigt um Fotos vom Produkt zu machen und diese hochzuladen.

Methode deleteBarcodesListener()
Die Methode "deleteBarcodesListener" setzt einen OnClickListener auf den Button "btnDelete". Dieser ist standardgemäß ausgeblendet. Es werden alle Produkte aus der

Datenbank gelöscht.

Methode fillSpinnerWithCategoryData()
Die Methode "fillSpinnerWithCategoryData" füllt das DropDown Menü mit den Produktkategorien. Zuerst werden die Kategorien abgefragt und in ein String-Array gespeichert. Der aktuelle "categoryString" entspricht dem ersten Element des Arrays. Als nächstes wird ein ArrayAdapter erzeugt mit diesem String Array. Der ArrayAdapter wird anschließend für den "categorySpinner" gesetzt. Zum Schluss wird noch ein OnItemSelectedListener definiert, welcher den "categoryString" für jedes ausgewählte Element neu setzt.

Methode requestFilePermission()
Die Methode "requestFilePermissions" fragt die Erlaubnis für den Zugriff auf das Dateisystem an. Diese wird benötigt um die lokale Fotogalerie zu öffnen.

Methode onRequestPermissionsResult(int requestCode, java.lang.String[] permissions, int[] grantResults)
Die Methode "onRequestPermissionsResult" wird ausgeführt, wenn die Erlaubnis erteilt oder verweigert wurde. Wenn die Erlaubnis für das Dateisystem erteilt wurde, wird die Fotogalerie geöffnet. Wenn die Erlaubnis für die Kamera erteilt wurde, wird die Kamera geöffnet.

Methode openCamera()
Die Methode "openCamera" erzeugt einen neuen Intent (ACTION_IMAGE_CAPTURE). Bevor die Kamera geöffnet wird, wird mithilfe der Methode "createPhotoFile" ein neuer Dateipfad für das Foto ezeugt, welches die Kamera aufnehmen wird, damit es lokal gespeichert werden kann. Anschließend wird der Pfad als URI an den Intent mit übergeben, welcher anschließend gestartet wird. Der Nutzer wird zur Kamera weitergeleitet.

Methode openFilePicker()
Die Methode "openFilePicker" öffnet die lokale Bildergalerie, also die Fotos welche auf dem Gerät gespeichert sind. Dazu wird ein neuer Intent erstellt (ACTION_PICK) mit dem Type "image/*". Dieser wird anschließend gestartet.

Methode createPhotoFile()
Die Methode "createPhotoFile" erzeugt einen neuen Dateipfad für das Bild, welches von der Kamera aufgenommen wird. Dieser setzt sich aus dem Standardpfad für Bilder und dem Dateinamen zusammen. Der Dateiname wird mit "IMG_" + "yyyMMdd_HHmmss" + ".jpg" erzeugt.

Methode onActivityResult(int requestCode, int resultCode, android.content.Intent data)
Die Methode "onActivityResult" wird ausgeführt, wenn der Nutzer wieder von der Kamera oder der Galerie zurück geleitet wurde. Es wird zunächst überprüft ob der Nutzer

von der Kamera oder von der Galerie zurück geleitet wurde. Wenn der Nutzer von der Galerie zurück geleitet wurde, dann wird überprüft ob die übermittelten Daten nicht null sind (nullCheck()) und die URI erstellt, welche dem ausgewählten Bild entspricht. Wenn der Nutzer von der Kamera zurück geleitet wurde, dann ist der entsprechende Bildpfad, derjenige, welcher vor dem Aufruf der Kamera mit der Methode "createPhotoFile" erzeugt wurde. In beiden Fällen wird der Bildpfad in der Variable "imgUplPath" gespeichert.

Methode closeKeyboard()
Die Methode "closeKeyboard" schließt die Onscreen Tastatur.

Methode goToProfile()
Die Methode "goToProfile" leitet einen zum Nutzerprofil weiter.

Methode goToMainActivity()
Die Methode "goToMainActivity" leitet einen zur "MainActivity" weiter.

5.3.2.8 ProductDetailActivity Die Klasse "ProductDetailActivity" zeigt alle Einzelheiten zu einem Produkt an.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Hier werden Werte initialisiert, zum Beispiel TextViews und Buttons. Der Barcode, welcher von der vorherigen Activity übergeben wurde, wird hier wieder von den Intent Extras übergeben. Mithilfe des Barcodes werden aus der lokalen SQLite Datenbank alle wichtigen Informationen zum Produkt abgefragt. Wenn keine Informationen gefunden werden, wird eine aussagekräftige Fehlermeldung angezeigt. Ansonsten werden die zum Produkt gefundenen Informationen in die TextViews geladen. Das Bild wird in die ImageView geladen. Es wird zusätzlich ein zweites Bild erzeugt, welches dem Barcode entspricht. Für die Buttons werden Methoden aufgerufen, welche OnClickListener festlegen. Wenn der Nutzer eingeloggt ist, dann werden die Buttons für das Hochladen von Fotos mithilfe der Kamera oder der lokalen Fotogalerie initialisiert. Am Ende wird der Preis mit der Methode "fetchCurrentPrice" abgefragt.

Methode onCreateOptionsMenu(android.view.Menu menu)
Die Methode onCreateOptionsMenu erzeugt das Menü für die AktionsLeiste Es wird zuerst überprüft ob der Nutzer eingeloggt ist. Nutzername und Password werden in einem User Objekt gespeichert. Das Menü "menu_loggedin" wird hier verwendet. Der Nutzername wird in das Feld "action_username" eingetragen und ein OnClickListener erstellt mit der Methode "goToProfile". Für den Logout Button wird ebenfalls ein OnClickListener erstellt, welcher den Nutzer ausloggt. Ist der Nutzer allgemein nicht eingloggt, so wird stattdessen das Standardmenü geladen.

Methode onOptionsItemSelected(android.view.MenuItem item)
Die Methode "onOptionsItemsSelected" wird ausgeführt, wenn ein Menüelement ausgewählt wurde. Je nachdem um welches Element es sich handelt werden unterschiedliche Aktionen ausgeführt. Bei "action_search": Man wird zur Suche weitergeleitet. Bei "action_login": Wenn eingeloggt, dann wird man zum Profil weitergeleitet. Wenn nicht eingeloggt, dann wird man zum Login weitergeleitet. Bei "action_settings": Man wird zu den Einstellungen weitergeleitet. Bei "action_info": Man wird zu den Informationen über die App weitergeleitet. Bei "action_close": Die App wird beendet.

Methode productPhotosActionListener()
Die Methode "productPhotosActionListener" setzt einen OnClickListener für den Button "buttonProductPhotos", welcher den Nutzer zu den Produktfotos weiterleitet und den Barcode des Produkts mit übergibt.

Methode btnTakePhotoActionListener()
Die Methode "btnTakePhotoActionListener" setzt einen OnClickListener für den Button "btnTakePhoto" und fragt die Erlaubnis für die Benutzung der Kamera an.

Methode btnUploadImageActionListener()
Die Methode "btnUploadImageActionListener" setzt einen OnClickListener für den Button "btnImageUpload" und fragt die Erlaubnis für den Dateizugriff auf die Fotogalerie an.

Methode requestFilePermission()
Die Methode "requestFilePermissions" fragt die Erlaubnis für den Zugriff auf das Dateisystem an. Diese wird benötigt um die lokale Fotogalerie zu öffnen.

Methode requestCameraPermissions()
Die Methode "requestCameraPermissions" fragt die Erlaubnis für den Zugriff auf die Kamera an. Diese wird benötigt um Fotos vom Produkt zu machen und diese hochzuladen.

Methode onRequestPermissionsResult(int requestCode, java.lang.String[] permissions, int[] grantResults)
Die Methode "onRequestPermissionsResult" wird ausgeführt, wenn die Erlaubnis erteilt oder verweigert wurde. Wenn die Erlaubnis für das Dateisystem erteilt wurde, wird die Fotogalerie geöffnet. Wenn die Erlaubnis für die Kamera erteilt wurde, wird die Kamera geöffnet.

Methode openCamera()
Die Methode "openCamera" erzeugt einen neuen Intent (ACTION_IMAGE_CAPTURE). Bevor die Kamera geöffnet wird, wird mithilfe der Methode "createPhotoFile" ein neuer Dateipfad für das Foto ezeugt, welches die Kamera aufnehmen wird, damit es lokal gespeichert werden kann. Anschließend wird der Pfad als URI an den Intent mit übergeben, welcher anschließend gestartet wird. Der Nutzer wird zur Kamera weitergeleitet.

Methode openFilePicker()
Die Methode "openFilePicker" öffnet die lokale Bildergalerie, also die Fotos welche auf dem Gerät gespeichert sind. Dazu wird ein neuer Intent erstellt (ACTION_PICK) mit dem Type "image/*". Dieser wird anschließend gestartet.

Methode createPhotoFile()
Die Methode "createPhotoFile" erzeugt einen neuen Dateipfad für das Bild, welches von der Kamera aufgenommen wird. Dieser setzt sich aus dem Standardpfad für Bilder und dem Dateinamen zusammen. Der Dateiname wird mit "IMG_" + "yyyMMdd_HHmmss" + ".jpg" erzeugt.

Methode onActivityResult(int requestCode, int resultCode, android.content.Intent data)
Die Methode "onActivityResult" wird ausgeführt, wenn der Nutzer wieder von der Kamera oder der Galerie zurück geleitet wurde. Wenn der resultCode OK ist, dann wird die Ladeanimation gestartet (loadingStart()) Es wird zunächst überprüft ob der Nutzer von

der Kamera oder von der Galerie zurück geleitet wurde. Wenn der Nutzer von der Galerie zurück geleitet wurde, dann wird überprüft ob die übermittelten Daten nicht null sind (nullCheck()) und die URI erstellt, welche dem ausgewählten Bild entspricht. Wenn der Nutzer von der Kamera zurück geleitet wurde, dann ist der entsprechende Bildpfad, derjenige, welcher vor dem Aufruf der Kamera mit der Methode "createPhotoFile" erzeugt wurde. In beiden Fällen wird anschließend die Methode "imageUploadAction" aufgerufen.

Methode btnAddPriceAction(java.lang.String barcode)
Die Methode "btnAddPriceAction" setzt einen OnClickListener für den Button "btnAddPrice". Der Nutzer soll die Möglichkeit haben einen Preis für das Produkt hinzuzufügen. Es wird ein neuer Intent erstellt, welcher auf die "CreatePriceActivity" weiterleitet.

Methode btnPriceHistoryAction(java.lang.String barcode)
Die Methode "btnPriceHistoryAction" setzt einen OnClickListener für den Button "btnPriceHistory". Der Nutzer wird auf den Preisverlauf des Produkts weitergeleitet. Es wird ein Intent erstellt, welcher auf die "PriceHistoryActivity" weiterleitet.

Methode btnTestAction(java.lang.String barcodeTest)
Die Methode "btnTestAction" setzt einen OnClickListener für den Button "btnTest". Zuerst wird überprüft ob das AR Model in der lokalen Datenbank vorhanden ist. Sollte es nicht vorhanden sein, so wird der Button "btnTest" deaktiviert, die Hintergrundfarbe auf Grau gesetzt und der Text des Buttons auf "No Model". Wenn ein Model existiert, dann wird ein OnClickListener für den Button "btnTest" erzeugt. Weiterhin wird überprüft ob die Einstellung "Ar Marker" aktiviert ist. Wenn Ja, dann wird der Nutzer auf die "ProductScanActivity" weitergeleitet Ansonsten wird der Nutzer auf die "ProductArActivity" weitergeleitet.

Methode btnDeleteAction(java.lang.String barcodeDelete)
Die Methode "btnDeleteAction" setzt einen OnClickListener für den Button "btnDelete". Das Produkt wird aus der lokalen Datenbank gelöscht und der Nutzer wird wieder zurück zur Produktübersicht geleitet.

Methode btnShareAction(java.lang.String name, java.lang.String barcode)
Die Methode "btnShareAction" setzt einen OnClickListener für den Button "btnShare". Der Nutzer hat die Möglichkeit die Produktinformationen zu teilen. Es wird ein Intent erzeugt (ACTION_SEND) mit dem Type ("text/plain") Diesem wird die Nachricht zum Teilen übergeben.

Methode btnRateAction(java.lang.String productName, java.lang.String barcode)
Die Methode "btnRateAction" setzt einen OnClickListener für den Button "btnRate". Der Nutzer hat die Möglichkeit das Produkt zu bewerten.

Methode fetchCurrentRating(java.lang.String barcode)
Die Methode "fetchCurrentRating" fragt die Bewertungen für das Produkt ab. Der Barcode wird dann an die Methode "getAllRatingsForProduct" weitergegeben. Wenn die Ratings erfolgreich abgefragt wurden, wird der arithmetische Mittelwert ermittelt und dieser Wert an die RatingBar weitergeben. Die Ratingsbar zeigt nun die durchschnittliche Bewertung für ein Produkt an.

Methode fetchCurrentPrice(java.lang.String barcode)
Die Methode "fetchCurrentPrice" fragt den aktuellen Preis des Produktes ab. Der Barcode wird dann an die Methode "getProductLatestPrice" weitergegeben. Wenn der Preis erfolgreich abgefragt wurde, wird die Währung ermittelt und das Währungssymbol abgefragt. Der Preis wird zusammen mit dem Währungssymbol in die TextView "detailPrice" eingefügt. Weiterhin wird der zugehörige Shop mithilfe der Methode "getShopFromPrice" abgefragt. Ist die Abfrage erfolgreich, dann wird der Name des Shops dem Preis angefügt.

Methode imageUploadAction()
Die Methode "imageUploadAction" lädt das Bild auf den Server hoch. Zuerst wird ein Multipart RequestBody mit der Datei (vom Bildpfad) erstellt. Dieser wird zusammen mit dem Barcode und den Login Daten an die Methode "uploadProductPhoto" von der Klasse "RetrofitCRUD" übergeben, welche das Bild an den Server überträgt. Wenn das Foto erfolgreich hochgeladen wurde, dann wird eine Toast Nachricht angezeigt. Wenn das Foto zu groß ist, oder keine Internetverbindung besteht wird ebenfalls eine aussagekräftige Fehlermeldung über eine Toast Nachricht angezeigt. In jedem Fall wird die Ladeanimation wieder beendet (loadingEnd())

Methode goToProfile()
Die Methode "goToProfile" leitet einen zum Nutzerprofil weiter.

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.9 ProductPhotoGalleryActivity

Die Klasse "ProductPhotoGalleryActivity" ist die Fotogallerie, welche die Fotos für ein Produkt in einer GridView anzeigt.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Es werden Werte initialisiert, wie zum Beispiel eine GridView. Als erstes erhält man den Barcode des Produkts, für welches man die Bilder abfragen möchte. Die Ladeanimation wird gestartet

(loadingStart()). Als nächstes werden die Produktfotos vom Server abgefragt. Dazu wird der Barcode an die Methode "getProductPhotosByBarcode" übergeben Wenn die Anfrage erfolgreich war, dann werden die Fotos in einer Liste gespeichert. Es wird ein PhotoAdapter initialisiert und die Liste wird diesem übergeben. Anschließend wird für die GridView der Adapter gesetzt. Es wird noch ein OnItemClickListener für jedes Element in der GridView gesetzt, mit dem man dann auf die ProductPhotoDetailActivity weitergeleitet wird. Hierfür wird der Barcode und die Ressource URL des Fotos mit übergeben. In jedem Fall wird die Ladeanimation anschließend wieder beendet.

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.10 ProductPhotoDetailActivity Die Klasse "ProductPhotoDetailActivity" zeigt ein Produktfoto in voller Größe an.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Der Barcode und die Ressource URL wird aus den Intent Extras abgefragt. Das Foto wird in die ImageView geladen. Es werden noch Buttons zum Teilen und Herunterladen des Bildes hinzugefügt. Wurde das Bild vom Nutzer selbst erstellt, dann wird ein zusätzlicher Button zum löschen des Bildes hinzugefügt.

Methode shareBtnAddListener()
Die Methode "shareBtnAddListener" fügt einen OnClickListener für den Button "sharePhotoBtn" hinzu. Es wird ein Dateiname für das zu teilende Bild erzeugt. Als nächstes wird ein Intent (ACTION_SEND) erstellt, welcher die URI des Bildes erhält. Zuvor wird das Bild jedoch lokal gespeichert, damit es geteilt werden kann. Als Type wird ("image/*") gesetzt. Zum Schluss wird der Intent gestartet.

Methode downloadBtnAddListener()
Die Methode "downloadBtnAddListener" fügt einen OnClickListener für den Button "downloadPhotoButton" hinzu. Mithilfe der Methode "saveImageBitmapUsingPicasso" wird das Bild lokal auf dem Gerät gesichert.

Methode deleteBtnAddListener()
Die Methode "deleteBtnAddListener" fügt einen OnClickListener für den Button "deletePhotoButton" hinzu. Über die Methode "deletePhoto" wird eine Anfrage an den Server geschickt mit der URL des Fotos und den Login Informationen. Ist die Anfrage erfolgreich, so wurde das Foto gelöscht und der Nutzer wird zurück zur Foto Galerie geleitet.

Methode galleryAddPic(java.lang.String saveFileName)
Die Methode "galleryAddPic" fügt das Bild der Fotogalerie auf dem Gerät hinzu.

Methode saveImage(android.graphics.Bitmap image, java.lang.String fileName)
Die Methode "saveImage" speicher das Bild lokal auf dem Gerät.

5.3.2.11 CreatePriceActivity Die Klasse "CreatePriceActivity" ist dazu da, dass der Nutzer einen Preis für das Produkt erstellen kann. Der Nutzer kann angeben, wie viel ein Produkt kostet, um welche Währung es sich handelt und bei welchen Geschäft er es entdeckt hat.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Es werden Werte initialisiert, zum Beispiel TextViews. Der Barcode wird aus den Intent Extras abgefragt.

Methode fillSpinnerWithCurrencyData()
Die Methode "fillSpinnerWithCurrencyData" fügt die Währungen in ein Dropdown Menü ein. Die Namen der Währungen werden zuerst in einem String Array gespeichert. Der "currencyString" erhält zunächst den Wert des ersten Elements des Arrays. Es wird ein neuer ArrayAdapter initialisiert, welchem das String Array übergeben wird. Der Array Adapter wird anschließend für den "currencySpinner" gesetzt. Zuletzt wird ein onItemSelectedListener gesetzt, welcher den Wert für den "currencyString" dem gerade ausgewählten Wert setzt.

Methode fillSpinnerWithStoreData()
Die Methode "fillSpinnerWithStoreData" fügt die Geschäfte in ein Dropdown Menü ein. Es wird zunächst eine Anfrage an den Server gestellt, welche alle Möglichen Geschäfte zu einem Produkt abfrägt. Wenn die Anfrage erfolgreich war, dann bekommt man als Antwort eine Liste an Shops. Als nächstes wird das Dropdown Menü für die Shops erstmal sichtbar gemacht. Alle Shopnamen werden dann in ein String Array gespeichert. Der "storeString" erhält zunächst den Wert des ersten Elements des String Arrays. Als nächstes wird ein ArrayAdapter initialisiert, welcher die Shopnamen erhält. Der Array Adapter wird anschließend für den "storeSpinner" gesetzt. Zuletzt wird ein OnItemSelectedListener gesetzt, welcher den Wert für den "storeString" dem gerade ausgewählten Wert setzt.

Methode validatePrice()
Die Methode "validatePrice" validiert den Preis Als Kriterium muss dieser erfüllen: - Der Preis darf nicht leer sein - Der Preis darf eine Länge von 8 Zeichen nicht überschreiten

Methode validateStore()
Die Methode "validateStore" validiert das eingegebene Geschäft. Als Kriterium muss dieses erfüllen: - Geschäft darf nicht leer sein. - Name des Geschäfts darf nicht länger als 50 Zeichen sein.

Methode validateStoreString()
Die Methode "validateStoreString" validiert den String, welcher durch das Dropdown Menü ausgewählt wurde. Als Kriterium muss dieser erfüllen: - Der "storeString" darf nicht leer sein.

Methode confirmPrice(android.view.View v)
Die Methode "confirmPrice" überprüft ob alle Nutzereingaben zum Preis auch valide sind. Wenn der "storeString" nicht valide ist und entweder der Preis oder das Geschäft invalide sind, dann wird die Methode zurückgegeben. Wenn der Wert des EditText Feldes für den Store leer ist, dann wird zunächst überprüft ob der Preis valide ist. Ist der Preis invalide wird die Methode zurückgegeben. Weiterhin wird überprüft, ob der "storeString" valide ist. Ist dieser ebenfalls invalide, so wird die Methode zurückgegeben, ansonsten wird dieser weiterverwendet. Nun wird die Ladeanimation gestartet (loadingStart()) Zum Schluss wird die Methode "createPriceAndReturnToDetailActivity" ausgeführt.

Methode createPriceAndReturnToDetailActivity(java.lang.String price, java.lang.String currency, java.lang.String store)
Die Methode "createPriceAndReturnToDetailActivity" sendet die Preisinformationen an den Server und leitet den Nutzer zurück zur "ProductDetailActivity" Zunächst werden alle wichtigen Informationen in eine Map gespeichert: - Barcode - Preis - Währung - Geschäft Als nächstes wird die Map an die Methode "createPriceForProduct" weitergegeben, welche die Anfrage an den Server schickt. Ist die Anfrage fehlgeschlagen, dann wird eine aussagekräftige Toast Nachricht erzeugt. Wenn die Anfrage erfolgreich gewesen ist, dann wird der Nutzer zurück zur "ProductDetailActivity" geleitet, welcher der Barcode übergeben wird. In jedem Fall wird die Ladeanimation wieder beendet (loadingEnd())

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.12 PriceHistoryActivity Die Klasse "PriceHistoryActivity" zeigt dem Nutzer den Preisverlauf des Produkts an.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Es werden Werte initialisiert, wie zum Beispiel eine GraphView. Zuerst wird die Ladeanimation gestartet. Als nächstes wird der Barcode aus den Intent Extras abgefragt. Dieser wird der Methode "getProductPrices" übergeben, welche eine Anfrage an den Server schickt. Wenn die Anfrage erfolgreich war, dann wird eine Liste von Preisen als Antwort vom Server zurückgegeben. Wenn diese nicht "null" ist (nullCheck()), nicht leer ist und mindestens 2 Elemente beinhaltet, dann wir eine LineGraphSeries initialisiert, welcher Werte (Datenpunkte (x,y)) hinzugefügt werden können. Die Preise werden mit aufsteigenden x-Werten je einem y-Wert zugeordnet. Nun muss die maximale Anzahl an Datenpunkten

für die x-Achse noch einmal manuel gesetzt werden. Anschließend wird die LineGraph-Series dem "priceHistoryGraph" hinzugefügt. Somit werden alle Werte in Form eines einfachen Graphes visualisiert. In jedem Fall wird die Ladeanimation wieder beendet (loadingEnd())

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.13 CreateRatingActivity Die Klasse "CreateRatingActivity" ist dafür da, dass Nutzer eine Bewertung für ein Produkt erstellen können.

Methode onCreate(@Nullable android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Es werden wichtige Variablen initialisiert wie zum Beispiel eine RatingBar oder ein Textfeld. Außerdem werden die Methoden "ratingStarsListener()" und "rateButtonListener" aufgerufen.

Methode fillWithUserRating()
Die Methode "fillWithUserRating" füllt die RatingBar und das Textfeld mit der zuvor abgegebenen Bewertung des Nutzers aus, falls diese existiert, damit der Nutzer die Möglichkeit hat die Bewertung zu bearbeiten.

Methode ratingStarsListener()
Die Methode "ratingStarsListener" setzt einen OnRatingBarChangeListener für die RatingBar. In dem Fall wird die Anzahl der Sterne bei einer Bewertung in die int Variable stars gespeichert.

Methode rateButtonListener()
Die Methode "rateButtonListener" setzt einen OnClickListener für den "rateButton". Zuerst wird die Methode "validateRatingInput" aufgerufen, wenn diese true zurückliefert, wird die Eingabe an den Server geschickt, die Bewertung wird dann auf den Server gespeichert.

Methode validateRatingInput()
Die Methode "validateRatingInput" validiert die Eingabe der RatingBar und des Textfeldes. Diese besitzt folgende Vorgaben: - RatingBar muss mindestens einmal bewertet worden sein und - Anzahl der Sterne dürfen nicht 0 sein. - Text des Rating Reviews darf nicht leer sein.

Methode loadingStart()

Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.14 ProductSearchActivity Die Klasse "ProductSearchActivity" stellt die Suchfunktion zur Verfügung. Die Datenbank auf dem Server kann komplett nach Produkten durchsucht werden.

Methode onCreate(@Nullable android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Hier werden wichtige Werte initialisiert, wie zum Beispiel eine ListView oder ein Textfeld für die Suche. Mithilfe eines "addTextChangedListeners" und eines "TextWatchers" lässt sich eine Methode aufrufen sobald der Nutzer eine Änderung am Textfeld vorgenomen hat. Sind Suchergebnisse vorhanden, wird die ListView nun mit den Produkten, welche auf dem Server gefunden wurden, gefüllt. Ansonsten wird eine TextView angezeigt, dass keine Suchergebnisse gefunden wurden. Sobald der Nutzer auf ein Element der ListView tippt, wird er, sollte das Produkt lokal schon existieren direkt zur "ProductDetailActivity" weitergeleitet, ansonsten wird das Produkt erst heruntergeladen und der Nutzer wird nach einer kurzen Zeit auf die "ProductDetailActivity weitergeleitet.

Methode goToProductDetailActivity(java.lang.String itemBarcode)
Die Methode "goToProductDetailActivity" leitet einen auf die Informationsseite des Produktes weiter.

Methode showNoSearchResult()
Die Methode "showNoSearchResult" blendet die ListView aus und zeigt stattdessen eine TextView an mit der Nachricht, dass keine Suchergebnisse gefunden wurden.

Methode hideNoSearchResult()
Die Methode "hideNoSearchResult" blendet die TextView (keine Suchergebnisse) wieder aus.

5.3.2.15 RegisterActivity Die Klasse "RegisterActivity" ist dafür da, dass sich Nutzer ein Konto anlegen können um sich dann später in der App einloggen zu können.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Es werden Werte initialisiert: TextInputLayout: - Email - Nutzername - Passwort - Passwort wiederholen

Methode validateEmail()
Die Methode "validateEmail" validiert die E-Mail Adresse. Als Kriterium muss diese erfüllen: - E-Mail Adresse darf nicht leer sein - E-Mail Adresse muss @-Zeichen enthalten

Methode validateUsername()
Die Methode "validateUsername" validiert den Benutzernamen. Als Kriterium muss dieser erfüllen: - Benutzername darf nicht leer sein - Benutzername darf nicht länger als 25 Zeichen sein

Methode validatePassword()
Die Methode "validatePassword" validiert das Passwort. Als Kriterium muss dieses erfüllen: - Passwort darf nicht leer sein.

Methode validateRepeatPassword()
Die Methode "validateRepeatPassword" validiert das vom Nutzer doppelt eingegebene Passwort. Als Kriterium muss dieses erfüllen: - Doppeltes Passwort darf nicht leer sein - Doppeltes Passwort muss dem Passwort entsprechen

Methode confirmInput(android.view.View v)
Diese Methode überprüft alle Eingaben des Nutzers und wird zurückgegeben, wenn eine Eingabe invalide sein sollte. Sind alle Eingaben valide, wird die Methode "createUser" ausgeführt.

Methode createUser()
Die Methode "createUser" legt ein Nutzerkonto auf dem Server an. Dass Passwort wird vorher mit MD5 verschlüsselt. Die Werte E-Mail, Nutzername und Passwort werden in eine Map übertragen, welche an die Methode "createUser" übergeben wird. Diese stellt nun die Anfrage an den Server. Ist die Anfrage erfolgreich, so wird eine aussagekräftige Toast Nachricht erzeugt. In jedem Fall wird die Methode "clearTextFields" aufgerufen.

Methode clearTextFields()
Die Methode "clearTextFields" löscht alle Nutzereingaben aus den Textfeldern.

5.3.2.16 LoginActivity Die Klasse "LoginActivity" stellt einen "Anmeldebildschirm" zur verfügung. Sie ist dazu da, dass sich der Nutzer mit seinem Konto in der App an-

melden kann.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Es werden Werte initialisiert für die Felder Nutzername, Password sowie eine CheckBox. Weiterhin wird ein Button für die Registrierung eines Kontos hinzugefügt.

Methode addRegisterButton()
Die Methode "addRegisterButton" fügt einen OnClickListener für den Button "register_button" hinzu. Es wird ein Intent erstellt, welcher den Nutzer auf die RegisterActivity weiterleitet.

Methode validateUsername()
Die Methode "validateUsername" validiert den Benutzernamen. Als Kriterium muss dieser erfüllen: - Benutzername darf nicht leer sein - Benutzername darf nicht länger als 25 Zeichen sein

Methode validatePassword()
Die Methode "validatePassword" validiert das Passwort Als Kriterium muss dieses erfüllen: - Das Passwort darf nicht leer sein.

Methode confirmLogin(android.view.View v)
Wenn der Nutzername oder das Password invalide ist, dann wird die Methode zurückgegeben. Die Ladeanimation startet. Der Nutzername und das Passwort wird aus den EditText Feldern der TextInputLayouts zwischengespeichert. Das Passwort wird mit MD5 verschlüsselt. Anschließend wird die Methode "loginUser" aufgerufen.

Methode loginUser(java.lang.String username,
java.lang.String password, boolean stayLoggedIn)
Die Methode "loginUser" vergleicht die Login Informationen, welche vom Nutzer eingegeben wurden mit denen, welche sich auf dem Server befinden. Mithilfe der Methode "loginUser" wird eine Anfrage über Retrofit an den Server gestellt. Ist die Anfrage erfolgreich, so wird der Nutzer auf die "ProfileActivity" weitergeleitet. Ist stayLoggedIn true, dann werden die Anmeldedaten mithilfe des LoginHelpers in den SharedPreferences gespeichert. In jedem Fall wird die Ladeanimation wieder beeendet (loadingEnd())

Methode clearTextFields()
Die Methode "clearTextFields" löscht alle Nutzereingaben aus den Textfeldern. Sie wird hier nicht verwendet, da der Nutzer die Möglichkeit haben sollte, bei einem gescheiterten Anmeldeversuch den Nutzername oder Passwort ausbessern zu können.

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.17 ProfileActivity Die Klasse "ProfileActivity" zeigt das Nutzerprofil an, sowie alle wichtigen Informationen. Darunter zählen: - E-Mail Adresse - Nutzername - Bild Außerdem gibt es die Möglichkeit ein neues Bild hochzuladen. Weiterhin hat der Nutzer die Möglichkeit sein Konto wieder zu löschen.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der Activity ausgeführt. Hier werden Werte gesetzt, wie zum Beispiel TextViews. Über die Intent Extras wird der Nutzername und das Passwort übergeben. Die Ladeanimation startet (loadingStart()) Mithilfe der Methode "getUserByUsernameAndPassword" wird eine Anfrage an den Server erstellt, welcher alle wichtigen Nutzerinformationen abfrägt. Für den Fall dass diese nicht erfolgreich ist, wird eine aussagekräftige Toast Nachricht erzeugt und die Activity beendet. Wenn die Anfrage erfolgreich war, dann wird der Nutzername, die E-Mail Adresse sowie das Bild des Nutzers geladen. Zum Schluss werden die Methoden "btnChooseFileAction" und "btnDeleteAction" ausgeführt.

Methode btnChooseFileAction()
Die Methode "btnChooseFileAction" setzt einen OnClickListener für den Button "btnImageUpload". Die Methode "requestFilePermission" wird augeführt.

Methode btnDeleteAction()
Die Methode "btnDeleteAction" setzt einen OnClickListener für den Button "btnDelete". Es wird ein AlertDialog erstellt, in dem der Nutzer gefragt wird, ob er sein Konto wirklich löschen möchte.

Methode requestFilePermission()
Die Methode "requestFilePermissions" fragt die Erlaubnis für den Zugriff auf das Dateisystem an. Diese wird benötigt um die lokale Fotogalerie zu öffnen. Die Methode "openFilePicker" wird ausgeführt, wenn die Erlaubnis vorliegt.

Methode onRequestPermissionsResult(int requestCode,
java.lang.String[] permissions, int[] grantResults)
Die Methode "onRequestPermissionsResult" wird ausgeführt, wenn die Erlaubnis erteilt oder verweigert wurde. Wenn die Erlaubnis für das Dateisystem erteilt wurde wird die Fotogalerie geöffnet.

Methode openFilePicker()
Die Methode "openFilePicker" öffnet die lokale Bildergalerie, also die Fotos welche auf dem Gerät gespeichert sind. Dazu wird ein neuer Intent erstellt (ACTION_PICK) mit dem Type "image/*". Dieser wird anschließend gestartet.

Methode onActivityResult(int requestCode,
int resultCode, android.content.Intent data)

Die Methode "onActivityResult" wird ausgeführt, wenn der Nutzer von der Galerie wieder zurück geleitet wurde. Wenn der result code OK ist und die übermittelten Daten nicht null sind (nullCheck()) wird eine URI erstellt, welche dem ausgewählten Bild entspricht. Anschließend wird die Methode "imageUploadAction" aufgerufen.

Methode imageUploadAction()
Die Methode "imageUploadAction" lädt das Bild auf den Server hoch. Zuerst wird ein Multipart RequestBody mit der Datei (vom Bildpfad) erstellt. Dieser wird zusammen mit dem Barcode und den Login Daten an die Methode "uploadProductPhoto" von der Klasse "RetrofitCRUD" übergeben, welche das Bild an den Server überträgt. Wenn das Foto erfolgreich hochgeladen wurde, dann wird eine Toast Nachricht angezeigt. Wenn das Foto zu groß ist, oder keine Internetverbindung besteht wird ebenfalls eine aussagekräftige Fehlermeldung über eine Toast Nachricht angezeigt. In jedem Fall wird die Ladeanimation wieder beendet (loadingEnd())

Methode loadingStart()
Die Methode "loadingStart" startet die Ladeanimation.

Methode loadingEnd()
Die Methode "loadingEnd" beendet die Ladeanimation.

5.3.2.18 SettingsActivity

Die Klasse "SettingsActivity" sind die Einstellungen.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Start der Activity ausgeführt. Es wird ein "SettingsHelper" und eine "DatabaseHelper" initialisiert. Als nächstes wird die Methode "configureSettingsElements" aufgerufen.

Methode configureSettingsElements()
Die Methode "configureSettingsElements" konfiguriert die Einstellungen. Als erstes wird die Einstellung für den Switch "ArMarker" konfiguriert. Dazu wird der aktuelle Wert der SharedPreferenes über den "SettingsHelper" abgefragt. Als letztes wird ein onCheckedChangeListener erstellt, welcher bei einer Änderung diese in den "SettingsHelper" wieder unter den SharedPreferences abspeichert. Analog passiert dies auch für den Switch für die "Special Deals". Für die "dealPercentage" wird die Methode "fillDealPercentageSpinner" aufgerufen. Für das "dealInterval" wird die Methode "fillDealIntervalSpinner" aufgerufen. Zuletzt wird die gesamte Anzahl aller Scans aus der lokalen Datenbank abgefragt und angezeigt.

Methode fillDealPercentageSpinner()
Die Methode "fillDealPercentageSpinner" fügt die Prozentzahlen für die Deal Notifications in ein Dropdown Menü ein. Das Array mit den Prozentzahlen wird zuerst vom

"PriceHelper" abgefragt. Es wird ein neuer ArrayAdapter initialisiert, welchem das String Array übergeben wird. Der Array Adapter wird anschließend für "dealPercentage" gesetzt. Als aktuelle Auswahl wird der in den SharedPreferences gespeicherte Wert aus dem "SettingsHelper" ausgelesen. Zuletzt wird ein OnItemSelectedListener gesetzt, welcher den gerade ausgewählten Wert für "dealPercentage" wieder mithilfe des SettingsHelpers abspeichert.

Methode fillDealIntervalSpinner()
Die Methode "fillDealIntervalSpinner" fügt das Interval für die Deal Notifications in ein Dropdown Menü ein. Zuerst wird ein String Array mit den möglichen Intervallen erstellt. Es wird ein neuer ArrayAdapter initialisiert, welchem das String Array übergeben wird. Der Array Adapter wird anschließend für "dealInterval" gesetzt. Als aktuelle Auswahl wird der in den SharedPreferences gespeicherte Wert aus dem "SettingsHelper" ausgelesen. Zuletzt wird ein OnItemSelectedListener gesetzt, welcher den gerade ausgewählten Wert für "dealInterval" wieder mithilfe des SettingsHelpers abspeichert.

Methode btnShowModelsAction()
Die Methode "btnShowModelsAction" setzt einen OnClickListener für den Button "showModelsBtn" Es werden alle Modelle von der lokalen SQLite Datenbank abgefragt und angezeigt.

5.3.2.19 InfoActivity

Die Klasse "InfoActivity" zeigt die App Infos an (z.B. Icon, Version, Author)

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten der App ausgeführt. Es wird die Versionsnummer bestimmt und für die TextView "app_version" gesetzt.

5.3.3 Fragment Klassen

5.3.3.1 ScanFragment

Die Klasse "ScanFragment" ist ein Fragment für das Scannen von Barcodes.

Methode onCreate(android.os.Bundle savedInstanceState)
Die Methode "onCreate" wird beim Starten des Fragments ausgeführt.

Methode checkPermission()
Die Methode "checkPermission" überprüft ob die Erlaubnis für die Benutzung der Kamera erteilt wurde.

Methode requestPermission()

Die Methode "requestPermission" fragt die Erlaubnis für die Benutzung der Kamera an.

Methode onRequestPermissionsResult(int requestCode,
java.lang.String[] permission, int[] grantResults)
Die Methode "onRequestPermissionsResult" wird aufgerufen nachdem die Abfrage für die Erlaubnis der Kamerabenutzung stattgefunden hatte.

Methode onActivityResult(int requestCode,
int resultCode, android.content.Intent intent)
Die Methode "onActivityResult" wird ausgeführt wenn ein Barcode gescannt wurde.

Methode displayAlertMessage(java.lang.String message,
android.content.DialogInterface.OnClickListener listener)
Die Methode "displayAlertMessage" generiert einen Fehlerdialog.

5.3.3.2 CustomArFragment

Die Klasse "CustomArFragment" ist ein ArFragment, welches in der "ProductScanActivity" verwendet wird.

Methode getSessionConfiguration(com.google.ar.core.Session session)
Die Methode getSessionConfiguration wurde überschrieben.

5.3.4 Adapter Klassen

5.3.4.1 ProductListAdapter

Die Klasse "ProductListAdapter" stellt eine Adapterklasse dar für die Produktelemente, welche in der "LastScannedProductsActivity" in einer ListView angezeigt werden.

Innere Klasse ViewHolder
ViewHolder Klasse, welche folgendes enthält: TextView name, TextView barcode, TextView scannedAt, ImageView image.

Methode getView(int position,
android.view.View convertView, android.view.ViewGroup parent)
Die Methode "getView" wird von der Klasse "ArrayAdapter" überschrieben. Zuerst werden die Produktinformationen in lokale Variablen gespeichert. Als nächstes wird ein Produkt erzeugt mit den entsprechenden Variablen. Weiterhin wird ein ViewHolder initialisiert. Dieser wird in den nächsten Schritten mit den Informationen des Produktes gefüllt. Es wird eine Animation für das Laden von weiteren Elementen festgelegt. Das Bild des Produktes wird in die dafür vorgesehene ImageView geladen.

5.3.4.2 SearchListAdapter Die Klasse "SearchListAdapter" stellt eine Adapterklasse dar für die Produktelemente, welche in der "SearchProductActivity" in einer ListView angezeigt werden.

Innere Klasse ViewHolder
ViewHolder Klasse, welche folgendes enthält: TextView name, TextView category, RatingBar rating, ImageView image.

Methode getView(int position,
android.view.View convertView, android.view.ViewGroup parent)
Die Methode "getView" wird von der Klasse "ArrayAdapter" überschrieben. Zuerst werden die Produktinformationen in lokale Variablen gespeichert. Als nächstes wird ein Produkt erzeugt mit den entsprechenden Variablen. Weiterhin wird ein ViewHolder initialisiert. Dieser wird in den nächsten Schritten mit den Informationen des Produktes gefüllt. Es wird eine Animation für das Laden von weiteren Elementen festgelegt. Das Bild des Produktes wird in die dafür vorgesehene ImageView geladen. Die durchschnittliche Bewertung des Produkts wird berechnet und der Wert wird in der RatingBar des ViewHolders gesetzt.

5.3.4.3 PhotoAdapter Die Klasse "ProductListAdapter" stellt eine Adapterklasse dar für die Fotos, welche in der "ProductPhotoGalleryActivity" in einer GridView angezeigt werden.

Methode getCount()
Gibt die Anzahl der Fotos zurück.

Methode getItem(int position)
Gibt das Foto zurück, welches sich an einer bestimmten Position befindet.

Methode getItemId(int position)
Methode "getItemId" gibt die Id eines Items zurück.

Methode getPhotos()
Alle Fotos bekommen.

Methode setPhotos(java.util.List¡Photo¿ photos)
Liste Fotos setzen.

Methode addPhoto(Photo photo)
Einzelnes Foto hinzufügen.

Methode removePhoto(Photo photo)

Einzelnes Foto entfernen.

Methode getView(int position,
android.view.View convertView, android.view.ViewGroup parent)
Methode "getView" Das Foto wird in die dafür vorgesehene ImageView in der GridView geladen.

5.3.5 Hilfs Klassen

5.3.5.1 GeneralHelper Die Klasse "GeneralHelper" enthält allgemeine Variablen und Hilfsmethoden.

Methode toastMessage(java.lang.String message, android.content.Context context)
Die Methode "toastMessage" sendet eine Toast Message

Methode showMessage(java.lang.String title, java.lang.String message, android.content.Context context)
Die Methode "showMessage" zeigt einen Fehlerdialog an.

Methode alertDialog(java.lang.String message, android.content.Context context)
Die Methode "alertDialog" zeigt einen Fehlerdialog an.

Methode getTimestampStringNow()
Die Methode "getTimestampStringNow" gibt den aktuellen Timestamp in Form eines Strings aus.

Methode convertFromTimestamp(java.lang.String timestamp)
Die Methode "convertFromTimestamp" gibt den Timestamp als String an im folgenden Format: DD.MM.YYYY HH:II:SS

Methode convertFromTimestampWithoutSec(java.lang.String timestamp)
Die Methode "convertFromTimestampWithoutSec" gibt den Timestamp als String an im folgenden Format: DD.MM.YYYY HH:II

Methode TimestampIsBefore(java.lang.String timestamp_a, java.lang.String timestamp_b)
Die Methode "TimestampIsBefore" überprüft ob ein Timestamp zeitlich vor einem anderen Timestamp liegt.

Methode MD5(java.lang.String md5)
Die Methode "MD5" generiert einen MD5 Hashwert für einen bestimmten Eingabestring.

Methode getRealPathFromUri(android.net.Uri uri, android.content.Context context)
Die Methode "getRealPathFromUri" gibt den Pfad einer Uri zurück.

Methode getNames(java.lang.Class $\langle ?extendsjava.lang.Enum\langle ?\rangle\rangle$ **e)**
Die Methode "getNames" gibt die Namen eines Enums in Form eines String Arrays zurück.

Methode getPositionFromStringArray(java.lang.String[] array, java.lang.String elem)
Die Methode "getPositionFromStringArray" gibt die Position eines Elements in einem Stringarray zurück.

Methode nullCheck(java.lang.Object obj)
Die Methode "nullCheck" überprüft ob ein Objekt null ist / ob ein Objekt existiert / initialisiert wurde.

Methode nullToString(java.lang.Object input)
Methode nullToString(java.lang.Integer input)
Methode nullToString(java.lang.Long input)
Methode nullToString(java.lang.Double input)
Methode nullToString(java.lang.String input)

Die Methode "nullToString" fügt den Platzhalter "null" ein wenn ein Objekt in einer Ausgabe null sein sollte.

Methode getVersionNumber(android.content.Context context)
Die Methode "getVersionNumber" fragt die Version der App ab.

5.3.5.2 BarcodeHelper

Die Klasse "BarcodeHelper" bietet Hilfsmethoden für Barcodes an.

Methode encodeAsBitmap(java.lang.String contents, com.google.zxing.BarcodeFormat format, int img_width, int img_height) throws com.google.zxing.WriterException
Die Methode "encodeAsBitmap" wandelt den Barcode in ein Bild um.

Methode guessAppropriateEncoding(java.lang.CharSequence contents)
Die Methode "guessAppropriateEncoding" ist eine Hilfsmethode für die Methode "encodeAsBitmap".

Methode generateBarCodeCode128(java.lang.String data)
Die Methode "generateBarCodeCode128" generiert ein Bild von einem Barcode im Format CODE128.

Methode generateBarCodeEAN(java.lang.String data)
Die Methode "generateBarCodeEAN" generiert ein Bild von einem Barcode im Format EAN_13.

5.3.5.3 QRCodeHelper Die Klasse QRCodeHelper bietet Hilfsmethoden für QR Codes an.

Methode qrCreateBitmap(java.lang.String str)
Die Methode "qrCreateBitmap" erstellt ein Bild von einem QR Code.

5.3.5.4 LoginHelper Die Klasse LoginHelper verwaltet wichtige Login Variablen sowie den gespeicherten Login.

**Methode saveLogin(java.lang.String username,
java.lang.String password, android.content.Context __context)**
Die Methode "saveLogin" speichert die Login Daten in den sharedPreferences.

**Methode checkIfSharedPrefsExistsAndNotEmpty(
android.content.Context __context)**
Die Methode "checkIfSharedPrefsExistsAndNotEmpty" überprüft ob die gespeicherten Login Daten existieren und nicht leer sind.

Methode getSharedUsername(android.content.Context __context)
Die Methode "getSharedUsername" gibt den gespeicherten Benutzernamen zurück.

Methode getSharedPassword(android.content.Context __context)
Die Methode "getSharedPassword" gibt das gespeicherte Passwort zurück.

Methode doLogout(android.content.Context __context)
Die Methode "doLogout" meldet den Benutzer ab.

5.3.5.5 SettingsHelper Die Klasse SettingsHelper verwaltet die gespeicherten Einstellungen.

Methode saveBoolean(java.lang.String key, java.lang.Boolean value)
Die Methode "saveBoolean" speichert eine Boolean in den sharedPreferences ab.

Methode saveString(java.lang.String key, java.lang.String value)
Die Methode "saveString" speichert einen String in den sharedPreferences ab.

Methode saveArSwitch(java.lang.Boolean switchValue)
Die Methode "saveArSwitch" speichert den Wert des Switches "ArMarker" ab.

Methode saveSpecialDeal(java.lang.Boolean specialDeal)

Die Methode "saveSpecialDeal" speichert den Wert des Switches "SpecialDealNotifications" ab.

Methode saveDealPercentage(java.lang.String dealPercentage)
Die Methode "saveDealPercentage" speichert den Wert (in Prozent) für Deals ab.

Methode saveInterval(java.lang.String interval)
Die Methode "saveInterval" speichert den Wert des Intervals für die Notifications ab.

Methode getArSwitch()
Die Methode "getArSwitch" gibt den aktuellen Zustand für den Switch "ArMarker" zurück.

Methode getSpecialDeal()
Die Methode "getSpecialDeal" gibt den aktuellen Zustand für den Switch "SpecialDealNotifications" zurück.

Methode getSpecialDealPercentage()
Die Methode "getSpecialDealPercentage" gibt den aktuellen Wert (in Prozent) für Deals zurück.

Methode getSpecialDealInterval()
Die Methode "getSpecialDealInterval" gibt das Interval für die Notifications zurück.

5.3.5.6 ImageHelper Die Klasse "ImageHelper" stellt eine generelle Hilfsklasse für Bilder dar.

Methode getBitmapFromURL(java.lang.String src)
Die Methode "getBitmapFromURL" gibt ein Bild von einer Ressource url zurück.

Methode saveImageBitmapUsingPicasso(java.lang.String image_ressource, java.lang.String title, java.lang.String description, android.content.ContentResolver contentResolver, android.content.Context context)
Die Methode "saveImageBitmapUsingPicasso" speichert ein Bild ab.

Methode setImageViewLocalImage(java.lang.String imagePath, android.widget.ImageView imageView)
Die Methode "setImageViewLocalImage" lädt ein lokales Bild in eine ImageView.

Methode isRemoteImage(java.lang.String imagePath)

Die Methode isRemoteImage überprüft ob es sich um ein Bild aus dem Internet handelt.

Methode getBitmapFromView(android.widget.ImageView imageView)
Die Methode getBitmapFromView gibt das Bild von einer ImageView zurück.

Methode getRightAngleImage(java.lang.String photoPath)
Die Methode "getRightAngleImage" ermittelt, wie ein Bild gedreht ist (0 Grad, 90 Grad, 180 Grad, 270 Grad).

Methode rotateImage(int degree, java.lang.String imagePath)
Die Methode "rotateImage" dreht ein Bild.

5.3.5.7 PhotoHelper

Die Hilfsklasse "PhotoHelper" enthält wichtige Hilfsmethoden und Variablen für die Fotogallerie.

Methode userCheck(java.lang.String photo_ressource_string, java.lang.String barcode)
Die Methode "userCheck" überprüft ob das Bild des Produkts vom Nutzer selbst hinzugefügt wurde. Nur dieser kann es löschen.

5.3.5.8 UploadHelper

Die Hilfsklasse "UploadHelper" enthält alle wichtigen Variablen für den Upload einer Datei.

5.3.5.9 PriceHelper

Die Hilfsklasse "PriceHelper" enthält alle wichtigen Variablen und Methoden für den Preis eines Produktes.

5.3.6 Retrofit Schnittstelle

Die Klasse "RetrofitCRUD", welche die Methoden des Interfaces "IRetrofitCRUD" implementiert, stellt die Methoden für Retrofit zur Verfügung und ist die Schnittstelle zum Backend Server.

Im Folgenden eine Übersicht der Methoden:

Modifier and Type	Method and Description
retrofit2.Call<okhttp3.ResponseBody>	**createNewRatingForProduct**(java.lang.String barcode, int stars, java.lang.String review, java.lang.String username, java.lang.String password)
retrofit2.Call<okhttp3.ResponseBody>	**createPriceForProduct**(java.util.Map<java.lang.String,java.lang.String> fields)
retrofit2.Call<okhttp3.ResponseBody>	**createProduct**(java.util.Map<java.lang.String,java.lang.String> fields)
retrofit2.Call<okhttp3.ResponseBody>	**createUser**(java.util.Map<java.lang.String,java.lang.String> fields)
retrofit2.Call<java.util.List<User>>	**createUser**(java.lang.String username, java.lang.String email, java.lang.String password)
retrofit2.Call<java.util.List<User>>	**createUser**(User user)
retrofit2.Call<okhttp3.ResponseBody>	**deletePhoto**(java.lang.String delete_img_url, java.lang.String delete_img_username, java.lang.String delete_img_password)
retrofit2.Call<java.util.List<Product>>	**GetAllProducts**()
retrofit2.Call<java.util.List<Rating>>	**getAllRatings**()
retrofit2.Call<java.util.List<Rating>>	**getAllRatingsForProduct**(java.lang.String barcode)
retrofit2.Call<java.util.List<Model>>	**getModelByProductId**(java.lang.String productid)
retrofit2.Call<Price>	**getProductCheapestPrice**(java.lang.String barcode_cheapest_price)
retrofit2.Call<Price>	**getProductLatestPrice**(java.lang.String barcode_latest_price)
retrofit2.Call<java.util.List<Photo>>	**getProductPhotosByBarcode**(java.lang.String barcode)
retrofit2.Call<java.util.List<Price>>	**getProductPrices**(java.lang.String barcode_prices)
retrofit2.Call<java.util.List<Product>>	**GetProductsBySearch**(java.lang.String searchString)
retrofit2.Call<okhttp3.ResponseBody>	**getProductSpecialDeal**(java.lang.String price_deal_barcode, java.lang.String price_deal_percentage)
retrofit2.Call<Shop>	**getShopFromPrice**(java.lang.String price_shop)
retrofit2.Call<java.util.List<Shop>>	**getShopsFromBarcode**(java.lang.String price_shop_barcode)
retrofit2.Call<java.util.List<Product>>	**GetSingleProductByBarcode**(java.lang.String barcode)
retrofit2.Call<java.util.List<Product>>	**GetSingleProductById**(int productId)
retrofit2.Call<java.util.List<Rating>>	**getSingleRatingForUser**(java.lang.String barcode, java.lang.String username, java.lang.String password)
retrofit2.Call<java.util.List<User>>	**getUserByUsernameAndPassword**(java.lang.String username, java.lang.String password)
retrofit2.Call<okhttp3.ResponseBody>	**loginUser**(java.lang.String username, java.lang.String password)
retrofit2.Call<okhttp3.ResponseBody>	**updateScanCount**(java.util.Map<java.lang.String,java.lang.String> fields)
retrofit2.Call<okhttp3.ResponseBody>	**uploadProductImageSync**(okhttp3.MultipartBody.Part image, java.lang.String barcode, java.lang.String username, java.lang.String password)
retrofit2.Call<okhttp3.ResponseBody>	**uploadProductPhoto**(okhttp3.MultipartBody.Part image, java.lang.String barcode, java.lang.String username, java.lang.String password)
retrofit2.Call<okhttp3.ResponseBody>	**uploadProfileImage**(okhttp3.MultipartBody.Part image, java.lang.String username, java.lang.String password)

Figure 8: Methoden der Klasse **RetrofitCRUD**

5.3.7 Network Monitor

Die Klasse "NetworkMonitor" überwacht die Netzwerkaktivität. Bei entsprechenden Änderungen werden Aktionen ausgeführt.

Methode onReceive(android.content.Context context, android.content.Intent intent)
Die Methode "onReceive" wird aufgerufen wenn sich der Netzwerkstatus geändert hat.

Methode checkNetworkConnection(android.content.Context context)

Die Methode "checkNetworkConnection" überprüft ob eine Internetverbindung besteht.

Methode syncData(android.content.Context context)
Die Methode "syncData" synchronisiert die erstellten Produkte mit der Datenbank auf dem Server.

Methode syncScanCount(android.content.Context context, java.lang.String syncedBarcode, int scanCount)
Die Methode "syncScanCount" synchronisiert die Anzahl der Scans.

Methode syncNewProductImage(android.content.Context __context, java.lang.String uploadBarcode, java.lang.String imageUrlPath)
Die Methode "syncNewProductImage" synchronisiert das Bild des Produkts.

5.3.8 Background Service

Die Klasse "ProductDealJobService" ist ein JobService für Benachrichtungen zu aktuellen Deals für Produkte.

Methode onStartJob(android.app.job.JobParameters params)
Die Methode "onStartJob" wird beim ausführen des Jobs ausgeführt.

Methode onStopJob(android.app.job.JobParameters params)
Die Methode "onStopJob" wird ausgeführt wenn der Job abgebrochen wurde.

Methode doBackgroundWork(android.app.job.JobParameters params))
Die Methode "doBackgroundWork" fragt beim Server an ob es für die auf dem Smartphone gespeicherten Produkte Sonderangebote gibt. Bei einem Sonderangebot werden entsprechende Push Benachrichtungen angezeigt.

5.3.9 Notifications

Die Klasse "SendNotification" ist eine Hilfsklasse zum versenden von Benachrichtigungen.

Methode sendDealPushNotification(java.lang.String title, java.lang.String notificationMessage, android.content.Intent startIntent)
Die Methode "sendDealNotification" sendet eine Benachrichtigung für ein Produkt mit einem Sonderangebot.

5.4 Ressourcen

5.4.1 Layout

5.4.1.1 activity_main
Das Layout für die Activity "MainActivity".

5.4.1.2 activity_create_price
Das Layout für die Activity "CreatePriceActivity".

5.4.1.3 activity_create_product
Das Layout für die Activity "CreateProductActivity".

5.4.1.4 activity_create_rating
Das Layout für die Activity "CreateRatingActivity".

5.4.1.5 activity_info
Das Layout für die Activity "InfoActivity".

5.4.1.6 activity_last_scanned
Das Layout für die Activity "LastScannedProductsActivity".

5.4.1.7 activity_login
Das Layout für die Activity "LoginActivity".

5.4.1.8 activity_price_history
Das Layout für die Activity "PriceHistoryActivity".

5.4.1.9 activity_product_ar
Das Layout für die Activity "ProductArActivity".

5.4.1.10 activity_product_detail
Das Layout für die Activity "ProductDetailActivity".

5.4.1.11 activity_product_photo_detail
Das Layout für die Activity "ProductPhotoDetailActivity".

5.4.1.12 activity_product_photo_gallery
Das Layout für die Activity "ProductPhotoGalleryActivity".

5.4.1.13 activity_product_scan
Das Layout für die Activity "ProductScanActivity".

5.4.1.14 activity_product_search
Das Layout für die Activity "ProductSearchActivity".

5.4.1.15 activity_profile
Das Layout für die Activity "ProfileActivity".

5.4.1.16 activity_register
Das Layout für die Activity "RegisterActivity".

5.4.1.17 activity_settings
Das Layout für die Activity "SettingsActivity".

5.4.1.18 adapter_view_layout
Das Layout für den Adapter "ProductListAdapter".

5.4.1.19 search_adapter_view_layout
Das Layout für den Adapter "SearchListAdapter".

5.4.1.20 spinner_spimple_item
Das Layout für ein Dropdown Menü.

5.4.1.21 list_layout
Das Layout für die Activity "ListDataActivity". Kann ignoriert werden.

5.4.2 Drawable Icons

Im Ordner "drawable" befinden sich Icons, welche innerhalb der App, beispielsweise in einem Layout, verwendet werden.

5.4.3 App Icon

Das Logo der App befindet sich auf dem Pfad "/res/mipmap". Mithilfe des Asset Studios in Android Studio lassen sich eigene Logos und andere Icons erstellen, ebenfalls auch Ladeanimationen. Das Logo der App wurde mit dem Asset Studio generiert und hat einen blauen Hintergrund sowie ein Drawable Icon in der Mitte (Vordergrund). Ein weiterer Vorteil ist, dass das Logo für alle verschiedenen Größen und Formate generiert wird. (z.B. auch runde Logos).

Figure 9: Logo der App "ProductAR"

5.4.4 Animation

5.4.4.1 SplashScreen

Hierfür werden die beiden XML Dateien "fade_in" und "fade_out" verwendet. Mit "fade_out" wird der SplashScreen ausgeblendet und mit "fade_in" der Übergang zur MainActivity eingeblendet.

5.4.4.2 Übergang zu einer neuen Activity

Es gibt 2 mögliche Animationen zum Übergang zu einer neuen Activity.

Mit "slide_in_left" und "slide_out_right" wird die neue Activity von links in das Bild geschoben und die vorherige Activity verschwindet nach rechts.

Mit "slide_in_right" und "slide_out_left" wird die neue Activity von rechts in das Bild geschoben und die vorherige Activity verschwindet nach links.

5.4.4.3 Laden neuer Elemente einer ListView

Es gibt hierbei auch 2 mögliche Animationen.

Mit "load_down_anim" wird ein neues Element von oben nach unten eingeschoben.

Mit "load_up_anim" wird ein neues Element von unten nach oben eingeschoben.

5.4.5 Menu

bottom_nav_menu
Es handelt sich hierbei um das Layout für Navigationsmenü am unteren Rand des Bildschirms, welches die 3 Schaltflächen "Test Product" (links), "Scan Barcode" (Mitte), "Products" (rechts) beinhaltet.

menu_home
Das Layout "menu_home" ist für das Menü zuständig, welches sich ganz oben in der App, jedoch noch unterhalb der Android Statusleiste befindet. Es wird für Nutzer, welch sich nicht in der App angemeldet haben angezeigt. Menüelemente, welche direkt angezeigt werden sind: Suchsymbol, Nutzersymbol. Weiterhin gibt es ein Dropdown Menü, welches folgende Elemente beeinhaltet: "Settings", "Info" und "Exit".

menu_logged_in
Das Layout "menu_logged_in" ist für das Menü zuständig, welches sich ganz oben in der App, jedoch noch unterhalb der Android Statusleiste befindet. Es wird für Nutzer, welch sich eingeloggt haben angezeigt. Menüelemente, welche direkt angezeigt werden sind: Suchsymbol, Nutzersymbol und Nutzername. Weiterhin gibt es ein Dropdown Menü, welches folgende Elemente beeinhaltet: "Settings", "Info", "Logout" und "Exit".

5.4.6 Assets

Der Ordner "Assets" befindet sich außerhalb des Ressource Ordners. Dennoch habe ich ihn mit in diese Kategorie genommen, da hier wichtige Ressourcen liegen. Hier werden die 3D Modelle für die AR Szene ("ProductArActivity" und "ProductScanActivity") gespeichert.

Als Platzhalter falls kein Produkt zum Testen ausgewählt wurde, wird in der "ProductArActivity" oder "ProductScanActivity" ein leerer Einkaufswagen in AR angezeigt.

Figure 10: Einkaufswagen in AR

5.4.7 Values

In diesem Ordner sind Dateien, welche String Variablen oder Konstanten enthalten, Dimensions Vorgaben, Farb und Style Vorgaben. Hier kann beispielsweise ein API Key gespeichert werden, falls in Zukunft die Google Maps API in der App verwendet wird.

5.5 Rest Api

Im Folgenden werden alle Möglichkeiten der Rest Schnittstelle noch einmal dokumentiert. Es werden keine Klassen oder Methoden beschrieben sondern der Pfad welcher auf dem Webserver existiert.

5.5.1 Pfad ../

Auf diesem Pfad befindet sich die Datenschutzerklärung der App (index.php), welche für einen Eintrag im Google Play Store erforderlich ist.

5.5.2 Pfad ../products/

5.5.2.1 Hauptpfad

Auf dem Hauptpfad befindet sich eine index.php

GET: Keine Parameter
Alle Produkte werden im JSON Format ausgegeben.

GET: ?productid=1
Das Produkt mit der Id "1" wird im JSON Format ausgegeben.

GET: ?barcode=1234567890128
Das Produkt mit dem Barcode "1234567890128" wird im JSON Format ausgegeben.

GET: ?search=banan
Produkte welche die Zeichenkette "banan" im Namen oder im Barcode enthalten werden im JSON Format ausgegben. Dabei spielt die Groß- und Kleinschreibung keine Rolle.

POST: Parameter: name, barcode, image, barcode, loginusername, loginpassword
Ein Produkt wird auf dem Server gespeichert, wenn es noch nicht vorhanden ist. Ist es bereits vorhanden, so wird nur eine Produkt-Nutzer Referenz hinzugefügt.

POST: scancount, scanbarcode, scanusername, scanpasswd
Die Anzahl an Scans werden für einen Nutzer aktualisiert.

5.5.2.2 Unterpfad ../products/images/

Auf diesem Pfad befindet sich eine index.php.

POST: product_title_img, product_title_img_barcode, product_title_img_username, product_title_img_password
Ein neues Produktbild wird auf den Server hochgeladen.

5.5.2.3 Unterpfad ../products/photos/

Auf diesem Pfad befindet sich eine index.php.

GET: Keine Parameter
Alle Fotos zu den Produkten werden im JSON Format ausgegeben, beginnend mit dem neuesten.

GET: ?barcode=1234567890128
Alle Fotos zu dem Produkt mit dem Barcode "1234567890128" werden im JSON Format ausgegeben, beginnend mit dem neuesten.

POST: product_img, product_img_barcode,
product_img_username, product_img_password
Nutzer welche eingeloggt sind können neue Fotos zu einem Produkt hochladen.

POST: delete_img_url, delete_img_username, delete_img_password
Nutzer welche eingeloggt sind können Ihre hochgeladenen Fotos zu einem Produkt wieder löschen.

5.5.2.4 Unterpfad ../products/prices/

Auf diesem Pfad befindet sich eine index.php.

GET: ?barcode_latest_price=1234567890128
Der letzte Preis zu dem Produkt mit dem Barcode "1234567890128" wird abgefragt und im JSON Format zurückgegeben.

GET: ?barcode_cheapest_price=1234567890128
Der günstigste Preis zu dem Produkt mit dem Barcode "1234567890128" wird abgefragt und im JSON Format zurückgeben.

GET: ?barcode_prices=1234567890128
Alle Preise zu dem Produkt mit dem Barcode "1234567890128" werden abgefragt und im JSON Format zurückgegeben.

GET: ?price_shop_id=5
Gibt den Shop mit der Id "5" im JSON Format aus.

GET: ?price_shop_barcode=1234567890128
Gibt alle verfügbaren Shops für den Barcode "1234567890128" im JSON Format zurück.

GET: ?price_deal_barcode=1234567890128&price_deal_percentage=0.5
Gibt an ob das Produkt (mit dem Barcode "1234567890128") gerade im Sonderangebot ist (0.5 bedeutet 50% Rabatt). Hierbei wird der aktuelle Preis mit dem ursprünglichen UVP verglichen. Wenn das Produkt im Angebot ist wird "Special Deal!" ausgegeben,

ansonsten "No Special Deal!".

POST: Parameter: price_barcode, price_value, price_currency, price_shop
Ein neuer Preis wird für das Produkt mit dem Barcode "1234567890128" in die Datenbank geschrieben.

5.5.2.5 Unterpfad ../products/ratings/

Auf diesem Pfad befindet sich eine index.php.

GET: Keine Parameter
Gibt alle Bewertungen im JSON Format zurück.

GET: ?barcode=1234567890128
Gibt alle Bewertungen für das Produkt mit dem Barcode "1234567890128" im JSON Format zurück.

GET: ?barcode=1234567890128
&username=test&password=e10adc3949ba59abbe56e057f20f883e
Gibt die Bewertung des Nutzers "test" für das Produkt mit dem Barcode "1234567890128" im JSON Format zurück.

POST: new_rating_barcode, new_rating_stars,
new_rating_review, new_rating_username, new_rating_password
Erstellt eine neue Bewertung von einem Nutzer für ein Produkt. Wenn schon bereits eine Bewertung für den Nutzer existiert, dann wird diese aktualisiert.

5.5.3 Pfad ../users/

5.5.3.1 Hauptpfad

Auf dem Hauptpfad befindet sich eine index.php

GET: ?username=test&password=e10adc3949ba59abbe56e057f20f883e
Es wird geprüft ob die Anmeldedaten (Benutzer: "test", Passwort: MD5("123456")) mit den Informationen in der Datenbank übereinstimmen. Wenn Ja, dann wird die Nachricht "Successfully logged in!" ausgegeben, ansonsten die Nachricht "Login failed! Username or Password incorrect!".

GET: ?username=test&password=e10adc3949ba59abbe56e057f20f883e&profile=profile
Wenn die Informationen für den Login stimmen, dann wird das Nutzerprofil im JSON Format zurückgegeben.

POST: username, email, password
Ein neuer Benutzer wird registriert.

5.5.3.2 Unterpfad ../users/images/

Auf diesem Pfad befindet sich eine index.php.

POST: profile_img, profile_img_username, profile_img_password
Ein neues Profilbild für einen Nutzer wird hochgeladen.

5.5.4 Pfad ../models/

5.5.4.1 Hautpfad

Auf dem Hauptpfad befindet sich eine index.php

GET: ?productid=1
Das Model für das Produkt mit der Id "1" wird im JSON Format zurückgegben.

6 Veröffentlichung im Google Play Store

6.1 Store Eintrag

Die App ist unter folgendem Link im Google Play Store erreichbar:
`https://play.google.com/apps/testing/de.nimoo.productar`

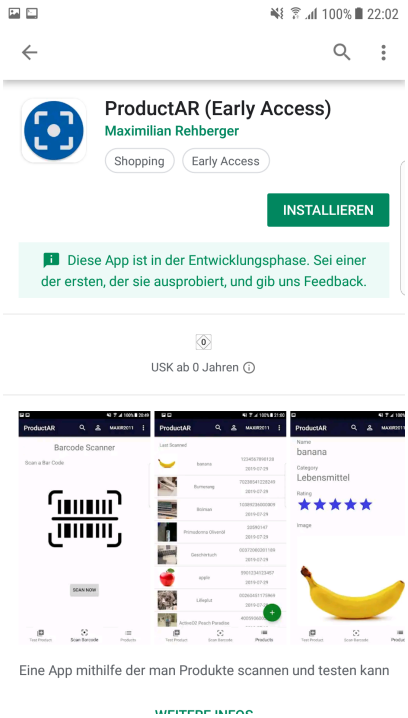

Figure 11: Screenshot des Eintrags im Google Play Store

6.1.0.1 Kurzbeschreibung

Eine App mithilfe der man Produkte scannen und testen kann

6.1.0.2 Beschreibung

Die App hat folgende Features:

- Barcodes von Produkten scannen
- Produkte lokal hinzufügen
- Informationen über Produkte erhalten

- Produkte Testen [BETA]

Bald im Play Store verfügbar!

6.2 Screenshots

Im folgenden die Screenshots, welche dem Eintrag im Google Play Store angefügt wurden.

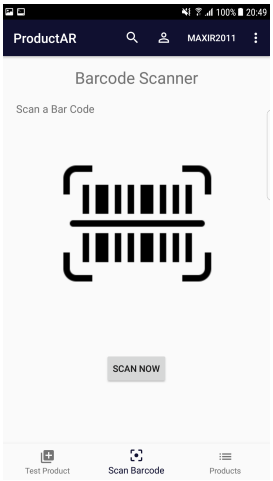

Figure 12: MainActivity

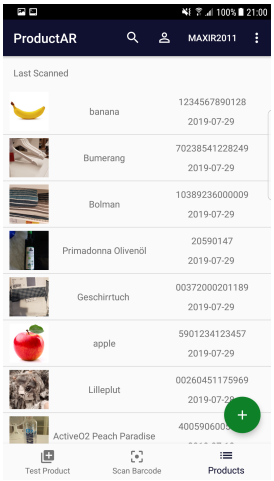

Figure 13: Products

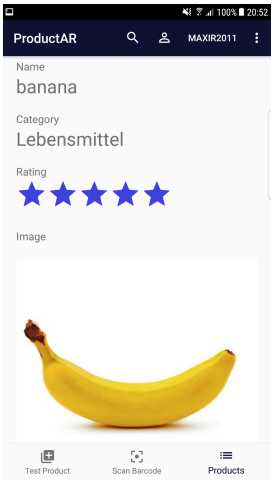

Figure 14: Details

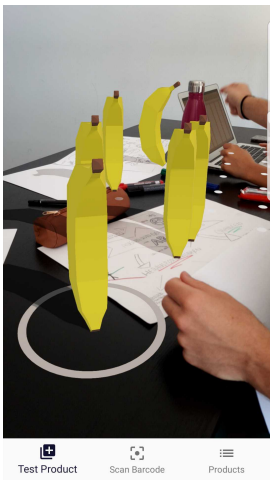

Figure 15: AR

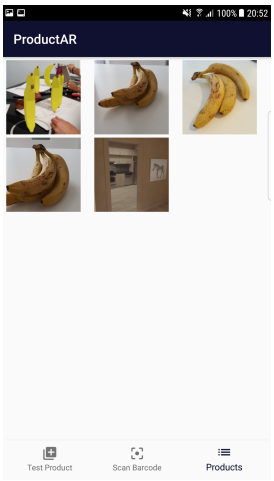

Figure 16: Photos

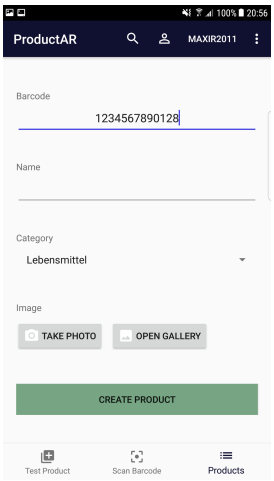

Figure 17: Create

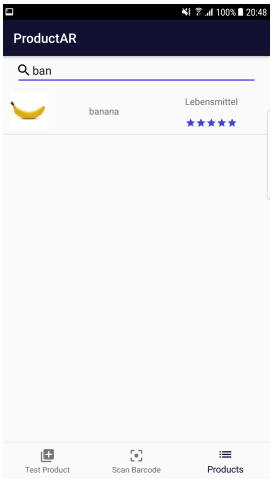

Figure 18: Search

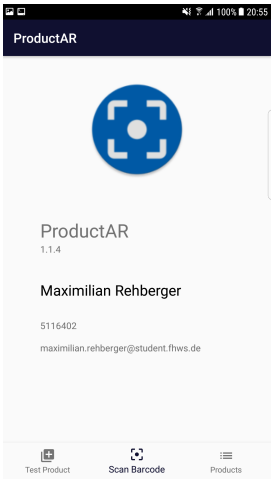

Figure 19: Info

6.3 Alpha Test

6.4 Beta Test

6.5 Testen im Ikea in Würzburg

7 Zukünftige Entwicklungen

7.1 Erweiterungen

Die App kann dahingehend erweitert werden, dass AR Modelle während der Laufzeit aus dem Internet geladen und direkt in der Welt platziert werden können. Aktuell ist hierbei die Ladezeit / Latenz ein großes Problem, die das Modell zum herunterladen benötigt, weshalb das Feature nicht mehr rechtzeitig in die fertige App übernommen werden konnte.

Für Verkäufer könnte ein Webinterface implementiert werden, auf welchen eine Liste an Produkten im csv Format hochgeladen werden kann. Eine weitere Fuktion des Webinterfaces wäre das Hochladen von 3D Modelle für die einzelnen Produkte.

Eine wichtige Erweiterung für die App könnte das Erstellen von einzelnen Einkaufslisten sein, in die mehrere Produkte gespeichert werden können.

Eine weitere Funktion wäre, dass der Nutzer sein erstelltes Konto wieder löschen kann. Hierbei wird der Nutzer zu erst gefragt ob er denn wirklich seinen Account auch wirklich löschen möchte. Dies ist soweit in der App implementiert, jedoch gibt es aktuell keine Backend Funktion, welche dies auch tatsächlich umsetzt.

Ab einer gewissen Nutzerzahl macht es Sinn, dass ein Supportsystem eingeführt wird.

7.2 Optimierung & Verbesserung

Die ProductDetailActivity ist sehr überladen mit Informationen und Buttons. Ein klares Design und eine Simplifizierung der Funktionen könnten die User Experience verbessern.

Auf Seite des PHP Backends könnte der Bilderupload optimiert werden. Man könnte für ein Produktbild sowohl ein Bild in Originalgröße als auch ein Thumbnail generieren, welches dann für die ListView genutzt werden kann, damit Elemente schneller geladen werden können.

Beim Abfragen von allen Produkten (GET ../products/) könnte irgendwann der Fall eintreten, dass zu viele Produkte in der Datenbank vorhanden sind. In diesem Fall könnte Pagination Abhilfe schaffen, da nur eine gewisse Anzahl an Produkten angefragt werden.

Bei der Abfrage des Preises in der ProductDetailActivity, wird kurz danach ein 2ter Request gestellt, welcher den Shop abfrägt. Man könnte dies vermeiden, in dem man den Shop als Subressource in den Preis einbettet.

Bei einer noch größeren Menge an Daten, welche erzeugt werden, wenn die App rasch an Nutzer gewinnt, könnte man überlegen, ob die Datenbank auf mehrere Server aufgeteilt

werden sollte (horizontale Skalierung). Dies kann entweder mit Partitioning oder mit Sharding umgesetzt werden. Mithilfe eines Load Balancers lässt sich die Last auf einzelne Server aufteilen.

Softwaretechnisch könnte man mit Einsatz von Map/Reduce ebenfalls eine weitere Optimierung erzielen.

8 Fazit

8.0.1 Was könnte in Zukunft besser gemacht werden?

9 Quellenangabe

9.1 Verwendete Technologie, Frameworks und Software

Android Studio (Primäre Entwicklungsumgebung)
Link: *https://developer.android.com/studio*

IntelliJ IDE (Entwicklungsumgebung)
Link: *https://www.jetbrains.com/idea/*

Android SDK
Android Emulator
Android 9.0 (Testen)

Java 8 (Entwicklung und Veröffentlichung)
Link: *https://www.java.com/de/download/faq/java8.xml*

Java JDK 8
Link: *https://www.oracle.com/technetwork/java/javase/documentation/jdk8-doc-downloads-2133158.html*

Gradle 3.4.2
Link: *https://gradle.org/*
Link: *https://docs.gradle.org/current/userguide/userguide.html*

Google Play Services 16
Link: *https://developers.google.com/android/guides/overview*

Google AR Core
Link: *https://developers.google.com/ar/*
Link: *https://developers.google.com/ar/discover/*

Google AR Sceneform
Link: *https://developers.google.com/ar/develop/java/sceneform/*
Link: *https://developers.google.com/ar/develop/java/sceneform/samples*

Google AR Sceneform Tools Plugin für Android Studio / IntelliJ
Link: *https://plugins.jetbrains.com/plugin/10698-google-sceneform-tools-beta-*

Google Developer Console
Link: *https://console.developers.google.com/*

Google Play Console (Veröffentlichung von Beta Versionen)
Link: *https://play.google.com/apps/publish/*

ZXing Barcode Scanner
Link: *https://github.com/zxing/zxing*

Retrofit2 (Anbindung zur Rest Api)
Link: *https://square.github.io/retrofit/*
Link: *https://github.com/square/retrofit*

Gson ((De-)Serialisierung von Java Objekten in JSON)
Link: *https://github.com/google/gson*

Picasso (Bibliothek zum Herunterladen von Bilder)
Link: *https://square.github.io/picasso/*
Link: *https://github.com/square/picasso*

Glide (Bibliothek zum Herunterladen von Bilder)
Link: *https://bumptech.github.io/glide/*
Link: *https://github.com/bumptech/glide*

SQLite (Datenbankverwaltungssystem)
Link: *https://www.sqlite.org/index.html*
Link: *https://www.sqlite.org/docs.html*

Linux Ubuntu (Remote Server)
Link: *https://ubuntu.com/*

Plesk (Oberfläche Server)
Link: *https://www.plesk.com/*

Apache (Webserver)
Link: *http://www.apache.org/*
Link: *https://httpd.apache.org/*
Link: *https://httpd.apache.org/docs/*

PHP (Rest Api und Backend)
Link: *https://www.php.net/manual/de/intro-whatis.php*

phpMyAdmin (Webinterface Datenbank)
Link: *https://www.phpmyadmin.net/*

MySQL (Datenbankverwaltungssystem)
Link: *https://www.mysql.com/de/*

Git (Versionsverwaltung)
Link: *https://git-scm.com/doc*

Bitbucket (Repository)
Link: *https://www.atlassian.com/de/software/bitbucket*
Link: *https://bitbucket.student.fiw.fhws.de:8443/dashboard*

Github (Repository)
Link: *https://github.com/*
Link: *https://github.com/maxr2011/*

Trello (Aufgabenverwaltung)
Link: *https://trello.com/de*

OneDrive (Cloud-Speicherung)
Link: *https://onedrive.live.com/about/de-de/*

Draw.io (Skizzenanfertigung)
Link: *https://www.draw.io/*

Putty (SSH Verbindung zum Remote Server)
Link: *https://www.putty.org/*

WinSCP (DateiBrowser um auf Server Dateien hochzuladen / zu verändern)
Link: *https://winscp.net/eng/index.php*

HeidiSQL (SQL Client um Abfragen auf dem Server durchzuführen)
Link: *https://www.heidisql.com/*

Postman (Testen der Rest Schnittstelle)
Link: *https://www.getpostman.com/*

Google Poly (Beziehen der 3D Modelle)
Link: *https://poly.google.com/*

9.2 Verlinkung Repositories

Die folgenden Repositories sind private Repositories auf Bitbucket oder GitHub.

App: ProductAR
Bitbucket: *https://bitbucket.student.fiw.fhws.de:8443/users/k40216/repos/productar/*
GitHub: *https://github.com/maxr2011/productar*

PHP Backend / Rest Api
Bitbucket: *https://bitbucket.student.fiw.fhws.de:8443/users/k40216/repos/productar_php_backend/*
GitHub: *https://github.com/maxr2011/product_ar_backend*

Dokumentation in LaTeX Quellcode und JavaDoc
GitHub: *https://github.com/maxr2011/productar_ducumentation*

SQL Backup
Bitbucket: *https://bitbucket.student.fiw.fhws.de:8443/users/k40216/repos/productar_sql_backup/*
GitHub: *https://github.com/maxr2011/productar_sql_backup*

9.3 Verlinkung Tutorials

Google AR Core Tutorial (inklusive Playlist)
https://www.youtube.com/watch?v=EWXGaypl2ms&list=PLsOU6EOcj51cEDYpCLK_bzo4qtjOwDWfW&index=2

QR und Barcode Scanner Tutorial
https://www.youtube.com/watch?v=otkz5Cwdw38

SQLite Database Tutorial
https://www.youtube.com/watch?v=KUq5wf3MhOc

Coding in Flow Tutorials
https://www.youtube.com/channel/UC_Fh8kvtkVPkeihBs42jGcA/playlists

TextInputLayout Tutorial
https://www.youtube.com/watch?v=veOZTvAdzJ8

Retrofit Tutorial (inklusive Playlist)
https://www.youtube.com/watch?v=4JGvDUlfk7Y&list=PLrnPJCHvNZuCbuD3xpfKzQWOj3AXybSaM

CodingWithMitch Tutorials
https://www.youtube.com/channel/UCoNZZLhPuuRteu02rh7bzsw/playlists

SQLite Database Tutorial
https://www.youtube.com/watch?v=aQAIMY-HzL8

ListView Tutorial
https://www.youtube.com/watch?v=cKUxiqNB5y0