Dokumentation ProductAR

Maximilian Rehberger

July 27, 2019

1 Inhaltsverzeichnis

1 Inhaltsverzeichnis **2**

2 Einleitung **5**
2.1 Zweck . . . 5

3 Allgemeine Übersicht **6**
3.1 Beschreibung Ausgangssituation . . . 6
3.2 Produkteinsatz . . . 6
3.3 Produktumfeld . . . 6
3.4 Produktfunktionalität . . . 6
3.5 Personas . . . 6
3.5.1 Nutzer . . . 6
3.5.2 Verkäufer . . . 6
3.5.3 Admin . . . 6

4 Architekturkonzept und Entwurf **7**
4.1 Ursprüngliches Architekturkonzept . . . 7
4.2 Aktualisiertes Architekturkonzept . . . 7
4.3 Anfängliche Skizze Datenbankentwurf . . . 8
4.3.1 MySQL Datenkbank (Remote) . . . 8
4.4 Anfängliche Skizze Java Klassen . . . 9
4.5 Endgültige Skizze Datenbankentwurf . . . 10
4.5.1 SQLite Datenbank (Lokal) . . . 10
4.5.2 MySQL Datenbank (Remote) . . . 10
4.6 Endgültige Skizze Java Klassen . . . 11
4.7 Übersicht Backend Server . . . 11
4.8 Übersicht REST API . . . 12
4.9 Technische Entscheidungen . . . 12
4.9.1 Warum Android? . . . 12
4.9.2 Welche Androidversion? . . . 12
4.9.3 Welche Entwicklungsumgebung? . . . 12
4.9.4 Wieso Google AR Core? . . . 13
4.9.5 Wieso eine MySQL Datenbank? . . . 13
4.9.6 Wieso eine REST API? . . . 13
4.9.7 Vergleich mit Alternativlösungen . . . 13
4.9.7.1 Firebase von Google . . . 13
4.9.7.2 Alternative Datenbankmodelle . . . 13

5 Technische Dokumentation **14**
5.1 Android Manifest . . . 14

5.2 Java Interfaces . . . 14
5.2.1 ObjectInterface . . . 14
5.2.2 ScanResultReceiver . . . 14
5.2.3 IRetrofitCRUD . . . 14
5.2.4 JsonPlaceHolderApi . . . 14
5.3 Java Klassen . . . 14
5.3.1 Objekt Klassen . . . 14
5.3.1.1 Object Class (Abstract) . . . 14
5.3.1.2 Product . . . 15
5.3.1.3 User . . . 15
5.3.1.4 Model . . . 15
5.3.1.5 Photo . . . 15
5.3.1.6 Price . . . 15
5.3.1.7 Shop . . . 15
5.3.1.8 Category (Enum) . . . 15
5.3.1.9 Currency (Enum) . . . 15
5.3.1.10 Interval (Enum) . . . 15
5.3.2 Aktivity Klassen . . . 16
5.3.2.1 MainActivity . . . 16
5.3.2.2 SplashScreen . . . 18
5.3.2.3 ProductArActivity . . . 19
5.3.2.4 ProductScanActivity . . . 19
5.3.2.5 CaptureActivityPortrait . . . 20
5.3.2.6 LastScannedProductsActivity . . . 20
5.3.2.7 CreateProductActivity . . . 21
5.3.2.8 ProductDetailActivity . . . 25
5.3.2.9 ProductPhotoGalleryActivity . . . 28
5.3.2.10 ProductPhotoDetailActivity . . . 29
5.3.2.11 CreatePriceActivity . . . 30
5.3.2.12 PriceHistoryActivity . . . 31
5.3.2.13 RegisterActivity . . . 32
5.3.2.14 LoginActivity . . . 33
5.3.2.15 ProfileActivity . . . 35
5.3.2.16 SettingsActivity . . . 36
5.3.2.17 InfoActivity . . . 37
5.3.3 Fragment Klassen . . . 37
5.3.3.1 ScanFragment . . . 37
5.3.3.2 CustomArFragment . . . 38
5.3.4 Adapter Klassen . . . 38
5.3.4.1 ProductListAdapter . . . 38
5.3.4.2 PhotoAdapter . . . 39
5.3.5 Hilfs Klassen . . . 40
5.3.5.1 GeneralHelper . . . 40
5.3.5.2 BarcodeHelper . . . 41

5.3.5.3 QRCodeHelper . . . 42
5.3.5.4 LoginHelper . . . 42
5.3.5.5 SettingsHelper . . . 42
5.3.5.6 ImageHelper . . . 43
5.3.5.7 PhotoHelper . . . 44
5.3.5.8 UploadHelper . . . 44
5.3.5.9 PriceHelper . . . 44
5.3.6 Retrofit Schnittstelle . . . 45
5.3.7 Network Monitor . . . 45
5.3.8 Background Service . . . 45
5.3.9 Notifications . . . 45
5.4 Ressourcen . . . 45
5.4.1 Layout . . . 45
5.4.2 Drawable Icons . . . 45
5.4.3 App Icon . . . 45
5.4.4 Animation . . . 45
5.4.5 Menu . . . 45
5.4.6 Assets . . . 45
5.4.7 Values . . . 45
5.5 Rest Api . . . 45

6 Veröffentlichung im Google Play Store 46
6.1 Store Eintrag . . . 46
6.2 Screenshots . . . 46
6.3 Alpha Test . . . 46
6.4 Beta Test . . . 46

7 Zukünftige Entwicklungen 47

8 Fazit 48

9 Verwendete Technologie, Frameworks und Software 49

10 Verlinkung Repositories 50

11 Verlinkung Tutorials 51

12 Quellenangabe 52

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

3.5.2 Verkäufer

3.5.3 Admin

4 Architekturkonzept und Entwurf

4.1 Ursprüngliches Architekturkonzept

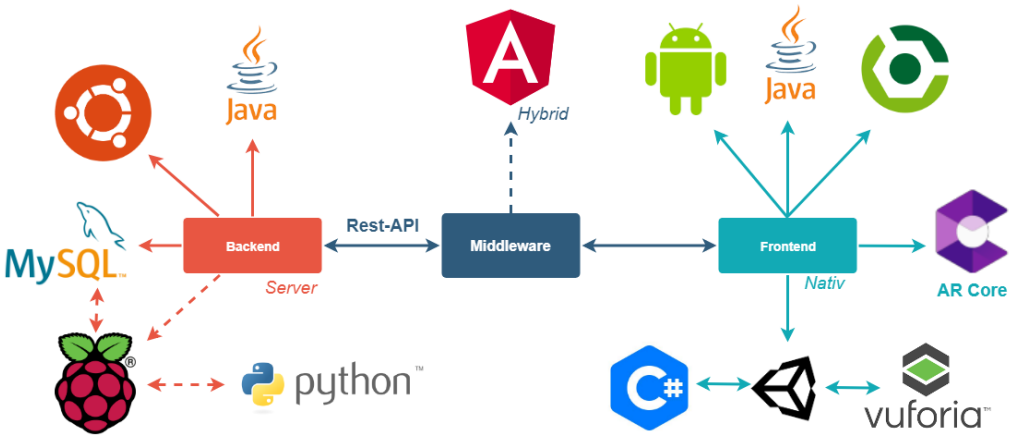

Figure 1: Ursprüngliches Architekturkonzept

4.2 Aktualisiertes Architekturkonzept

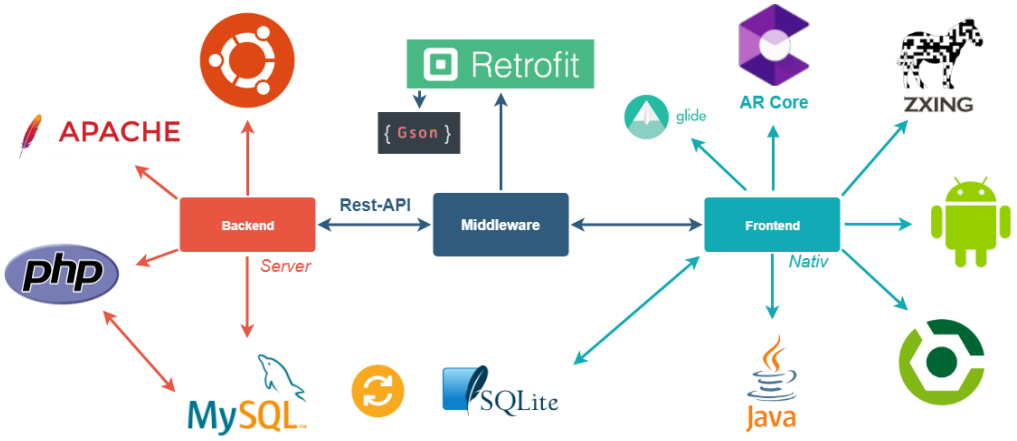

Figure 2: Aktualisiertes Architekturkonzept

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.

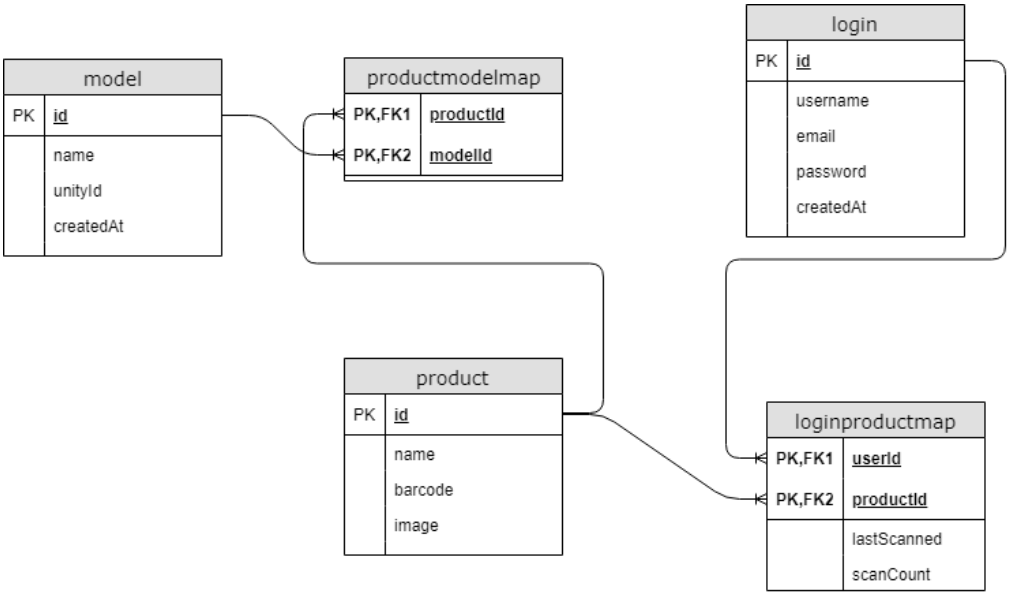

Figure 3: Anfängliche Skizze Datenbankentwurf

4.4 Anfängliche Skizze Java Klassen

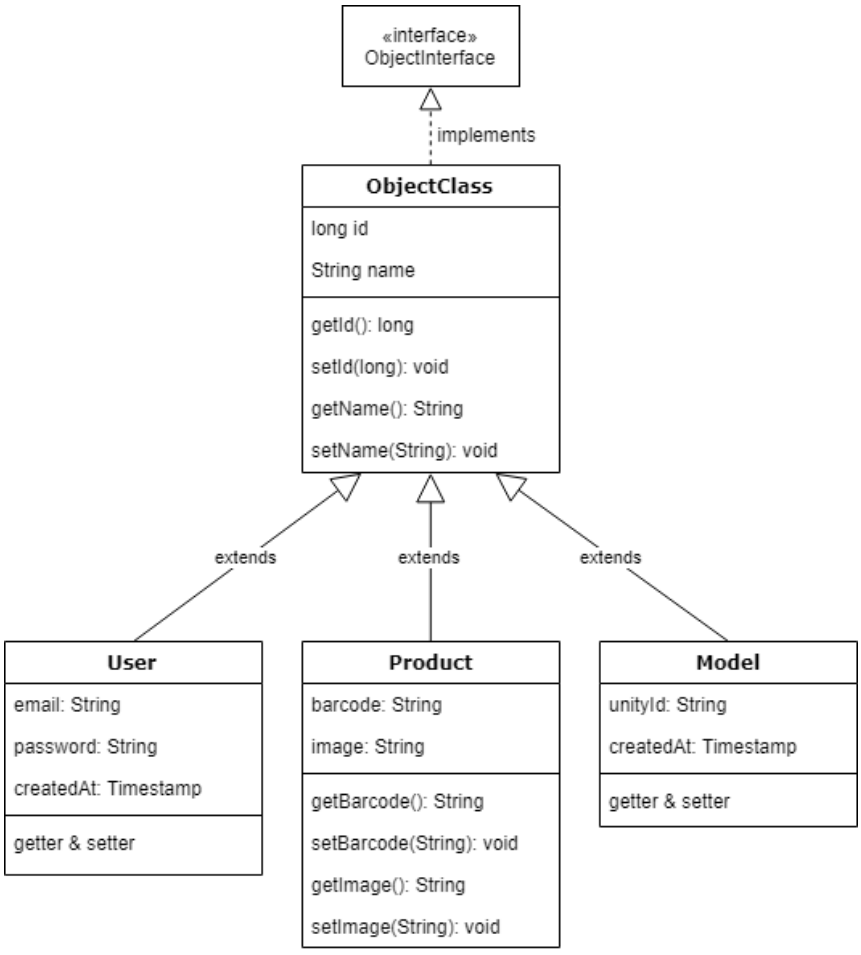

Figure 4: Anfängliche Skizze Datenbankentwurf

4.5 Endgültige Skizze Datenbankentwurf

4.5.1 SQLite Datenbank (Lokal)

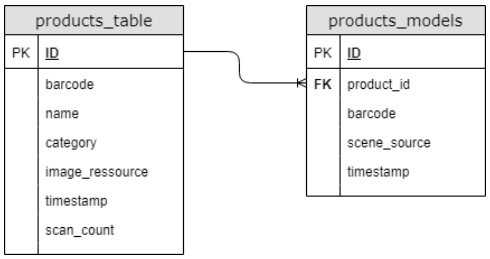

Figure 5: Skizze Datenbankentwurf: SQLite

4.5.2 MySQL Datenbank (Remote)

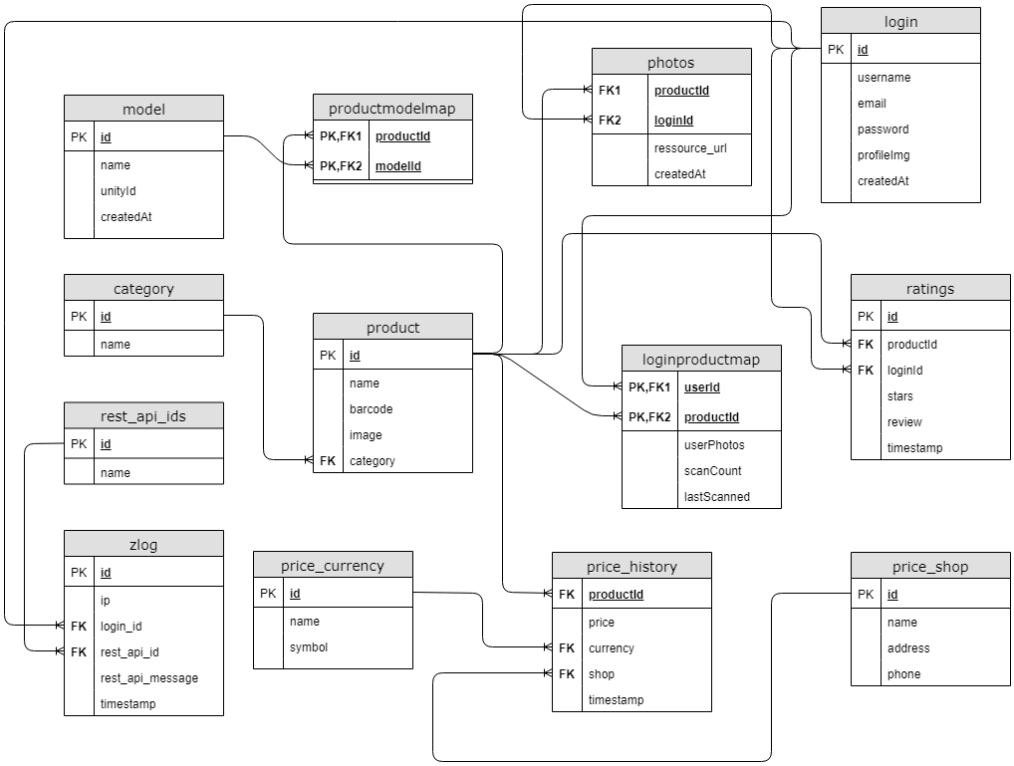

Figure 6: Aktualisierte Skizze Datenbankentwurf: MySQL

4.6 Endgültige Skizze Java Klassen

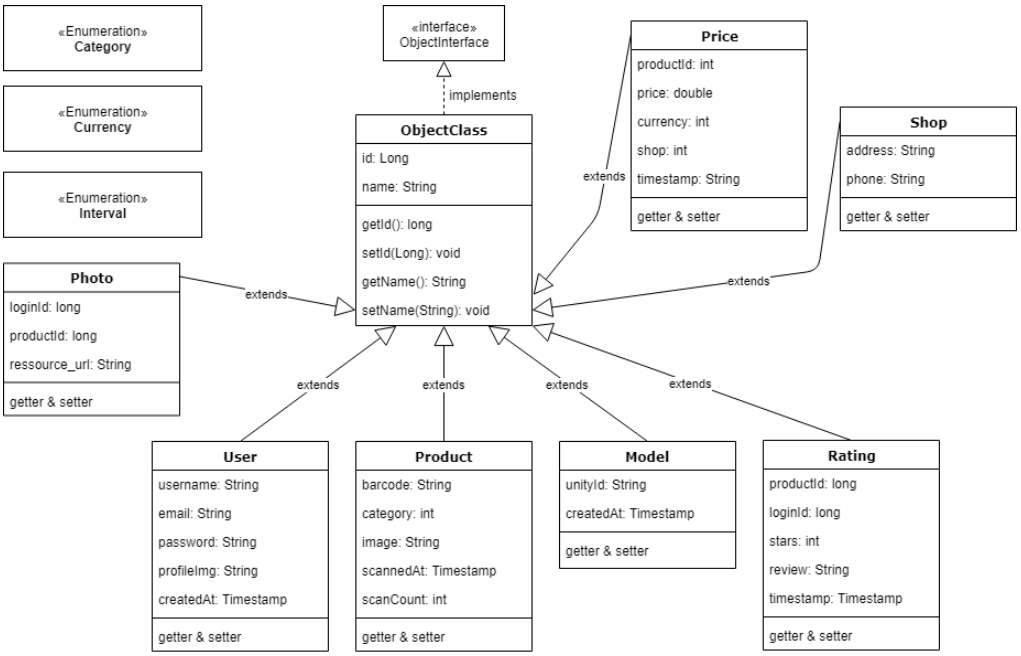

Figure 7: Aktualisierte Skizze: Java Klassen

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 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 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.

4.9.6 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.

4.9.7 Vergleich mit Alternativlösungen

4.9.7.1 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.

4.9.7.2 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_login": Wenn eingeloggt, dann wird mann 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_login": Wenn eingeloggt, dann wird mann 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_login": Wenn eingeloggt, dann wird mann zum Profil weitergeleitet. Wenn nicht eingeloggt, dann wird man zum Login weitergeleitet. Bei "ac-

tion_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_login": Wenn eingeloggt, dann wird mann 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 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 "de-

tailPrice" 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 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.

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

nUser" 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.

5.3.2.15 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.

5.3.2.16 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.

5.3.2.17 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 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
5.3.7 Network Monitor
5.3.8 Background Service
5.3.9 Notifications
5.4 Ressourcen
5.4.1 Layout
5.4.2 Drawable Icons
5.4.3 App Icon
5.4.4 Animation
5.4.5 Menu
5.4.6 Assets
5.4.7 Values
5.5 Rest Api

6 Veröffentlichung im Google Play Store

6.1 Store Eintrag

6.2 Screenshots

6.3 Alpha Test

6.4 Beta Test

7 Zukünftige Entwicklungen

8 Fazit

9 Verwendete Technologie, Frameworks und Software

10 Verlinkung Repositories

11 Verlinkung Tutorials

12 Quellenangabe