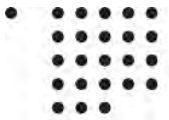

Realisierung einer Bilderkennung zur Filterung von Produktinformationen auf mobilen Android-Geräten

Bachelorthesis

ausgearbeitet von

Danny Pape
Matrikelnummer - 11060162

vorgelegt an der

Fachhochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Medieninformatik

Erster Prüfer:	Prof. Dr. Wolfgang Konen Fachhochschule Köln
Zweiter Prüfer:	Dr. Sven Abels Ascora GmbH

Gummersbach, der 29. September 2013

Adressen: Danny Pape
Helmut-Denker-Weg 36
27777 Ganderkesee
dannypape@smail.fh-koeln.de

Prof. Dr. Wolfgang Konen
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
wolfgang.konen@fh-koeln.de

Dr. Sven Abels
Ascora GmbH
Birkenallee 43
27777 Ganderkesee
abels@ascora.de

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsgbehörde vorgelegen.

Ganderkesee, 21. Juni 2013

Danny Pape

Kurzfassung

Die Bilderkennung ist ein anerkanntes Forschungsgebiet und hat in den letzten 10 Jahren einen großen Fortschritt gemacht. Basierend auf der vorangegangenen Arbeit „Konzeption und Vergleich von Bilderkennungsverfahren zur Filterung von Produkteigenschaften“ wird in einem praktischen Beispiel die Gebrauchstauglichkeit in einem verteilten System getestet. Dabei wird die Implementation auf mobilen Androidsystemen und einem Java basierten Dienstanbieter vorgenommen. Es wird geprüft, ob das ORB Verfahren für den praktischen Gebrauch in einem großen System nutzbar ist und ein Bildvergleich effektive Ergebnisse liefert.

Abstract

Image recognition is a recognized research area and has made over the past 10 years a great progress. Based on the previous work "Design and comparison of Image Recognition method for filtering of product characteristics“ usability is tested in a distributed system in a practical example. The implementation is done on mobile Androidsystems and a Java-based service provider. It is checked whether the ORB method is useable in a large system and provides effective image matching results.

Inhaltsverzeichnis

1. Einleitung **10**
1.1. Problemstellung . . . 10
1.2. Open Product Data Information System . . . 10
1.3. Nutzungskontext . . . 11
1.4. Bilderkennungskomponente . . . 12
1.4.1. Vorteile der Bilderkennung . . . 13
1.4.2. Aufgaben . . . 13
1.5. Gebrauchstauglichkeit . . . 15
1.5.1. Fehlervermeidung . . . 15
1.5.2. Lösungsansätze . . . 15
1.6. Projektziel . . . 16

2. ORB-Bilderkennungsverfahren **17**
2.1. oriented FAST and rotated BRIEF . . . 17
2.1.1. Detektor . . . 17
2.1.2. Deskriptor . . . 18
2.1.3. Bildvergleich . . . 19
2.2. Bildoptimierung . . . 19
2.2.1. Referenzbild . . . 19
2.2.2. Zielbild . . . 20
2.3. Marktanalyse . . . 22

3. Systemumgebung und Architektur **24**
3.1. Systemarchitektur . . . 24
3.1.1. Systemkomponenten . . . 25
3.1.2. Dienstanbieter . . . 25
3.1.3. Dienstnutzer . . . 27
3.1.4. Kommunikationsmodell . . . 27
3.1.5. Datenmodell . . . 29
3.2. Softwarekomponenten . . . 30
3.2.1. JDK und Android SDK . . . 30
3.2.2. Spring Framework . . . 30
3.2.3. OpenCV . . . 31
3.2.4. ElasticSearch . . . 32

4. Entwurfs- und Implementierungsaspekte **33**
4.1. Dienstnutzer . . . 33
4.1.1. Anwendungslogik . . . 33
4.1.2. Präsentationslogik . . . 38
4.2. Dienstanbieter . . . 41
4.2.1. Anwendungslogik . . . 41
4.2.2. Ressourcennmanagement . . . 42

4.3. Error-Handling . . . 43
4.3.1. Dienstanbieter . . . 43
4.3.2. Dienstnutzer . . . 44

5. Ergebnisse und Diskussion **46**
5.1. Testumgebung . . . 46
5.2. Testdurchlauf und Erkenntnisse . . . 46
5.2.1. Systemtests . . . 46
5.2.2. Einzeltests . . . 48
5.2.3. Praxistest . . . 52
5.3. Evaluation . . . 52
5.4. Zukunftsausblick . . . 54

6. Fazit **56**

Anhang **59**

A. User Stories **61**
A.1. User Story A: keine oder eingeschränkte Konnektivität . . . 61
A.2. User Story B: erfolgreicher Produktscan . . . 61
A.3. User Story C: erfolgloser Produktscan . . . 61
A.4. User Story D: bedingt erfolgreicher Produktscan . . . 61
A.5. User Story E: Distanz, Hintergrund und Perspektive . . . 61
A.6. User Story F: Bildqualität . . . 61

B. ElasticSearch **62**
B.1. Produktdaten . . . 62

Abbildungsverzeichnis

1.1. Hierarchical Task Analysis: Bilderkennung durchführen. 14

2.1. Beispiel eines Referenzbildes 20
2.2. Bewertung der Referenzbilder 21

3.1. Systemarchitektur: Modellierung der Systemkomponenten in eine Systemarchitektur 24
3.2. Kommunikationsmodell: Modellierung einer Kommunikation zwischen dem Dienstnutzer und Dienstanbieter bei einem Bildvergleich 28
3.3. Datenmodell: Modellierung der internen Kommunikationstypen von OPDIS-KNOW 29

4.1. ImageMatrix.class 34
4.2. FileManager.class 35
4.3. ImageManager.class 36
4.4. KeyPointManager.class 37
4.5. NetworkManager.class 38

5.1. Testreihe zu Tabelle 5.1 49
5.2. verschiedene Rotationen und Perspektiven 50
5.3. Ansicht der Merkmalsextraktionen aus verschiedenen Distanzen 51
5.4. Anwendungsszenario im Einzelhandel 51
5.5. Vergleich eines Produktes im Einzelhandel 53
5.6. Vergleich eines Produktes direkt aus dem Regal 53
5.7. Vergleich eines Produktes direkt aus dem Regal 54
5.8. Merkmalsextraktion im Schriftzug 54

Tabellenverzeichnis

1.1. Plansequenz für das HTA in Abbildung 1.1 14

2.1. ORB-Ergebnisswerte des Skalierungsinvarianz-Tests 20
2.2. Ergebnissewerte des Perspektivtests 22
2.3. Ergebnissewerte der Schnelligkeit 22

5.1. Distanztests 48
5.2. Ergebnissewerte des Perspektivtests 50
5.3. ORB Ergebniswerte des Luminanzinvarianztests 52

1. Einleitung

1.1. Problemstellung

Der Aufschwung der mobilen Endgeräte in den letzten Jahren ist nicht zu bestreiten. Im Januar 2013 befanden sich ca. 800.000 Apps im Google Playstore für Android Geräte und ca. 775.000 Apps im Apple Appstore[1]. Die Tendenz der Statistik ist steigend, da auch Tablets eine immer breitere Akzeptanz finden. Durch die Vielzahl der aktuellen Apps gewinnt der Benutzer eine große Auswahl an Möglichkeiten, seine individuellen Bedürfnisse auszuschöpfen.

Nutzerführungen und Interaktionsmöglichkeiten bestimmen oftmals den Erfolg einer App. Im Bereich der Wissenschaft wird versucht neue Möglichkeiten zu finden, um die User Experience zu steigern und somit Kunden an ihr Produkt zu binden. Seit einiger Zeit wird eine Vielzahl von Apps angeboten, Produkte mit dem Smartphone zu scannen. Allerdings gibt es seit der Einführung eines Barcodescanners für Strichcodes und QR-Codes keinen weiteren Fortschritt in diese Richtung. Aus diesem Grund soll eine neue Interaktionsmöglichkeit geschaffen und in ein bestehendes System integriert werden.

1.2. Open Product Data Information System

Das in der Entwicklung stehende Open Product Data Information System (Abk.: OPDIS) ist ein offenes Informationssystem zur Ermittlung von Produktdaten [2]. Es richtet sich hauptsächlich an Konsumenten und Anwendungsentwickler, die mobile Anwendungen im verbraucherspezifischen Nutzungskontext entwickeln.

Im späteren Einsatz wird eine Transparenz der Hersteller, Lieferanten, Produkte und Qualitätslabel für die Konsumenten geboten. Der Dienstanbieter stellt eine REST- Schnittstelle zur Verfügung, aus der detaillierte Produktdaten abgefragt werden können. Auf Dienstnutzerseite wird ein Framework für Anwendungsentwickler entwickelt, in dem sich die Anwendungslogik verbirgt. Dabei handelt es sich um das Scannen der Produkte, sowie das Verwenden von Suchfiltern. Das Framework wird für das Android Betriebssystem und für das iOS entwickelt und richtet sich speziell an die Anwendungsentwickler, damit diese OPDIS nutzen können.

[1] http://de.statista.com/themen/882/apps-app-stores/infografik/810/anzahl-der-verfuegbaren-apps-in-den-top-app-stores/

[2] http://opdis.de/ - zuletzt gesichtet am 28.09.2013

1.3. Nutzungskontext

Die Analyse des Nutzungskontextes ist im Usability Engineering eine etablierte Methode, um das richtige Ausmaß an Funktionalitäten zu finden. Die Ergebnisse aus dem Usability Engineering werden im Designprozess und auch bei der Planung der Systemarchitektur berücksichtigt. Während des Prozesses ist allerdings kein begleitendes Usability Engineering vorgesehen. Der in der ISO 9241-210 [fS10] beschriebene Nutzungskontext, bietet in dem projektspezifischen Fall eine Grundlage, um alle wichtigen Funktionalitäten auszuwählen und zu bewerten.

OPDIS ist auf den mobilen Nutzungskontext spezialisiert. Der Einsatzort ist im Allgemeinen nicht bestimmbar, da es jederzeit, an jedem Ort verwendet werden kann. Die einzige Bedingung zur erfolgreichen Nutzung, ist die Anbindung an das mobile World Wide Web. Die primäre Nutzung für den Prototypen wird an Orten stattfinden, wo Konsumgüter aus dem Lebensmittel- und Drogeriebereich gehandelt werden.

In Industrieländern werden Konsumgüter hauptsächlich in Supermärkten angeboten. Dort ist man sehr häufig künstlichem Licht ausgesetzt und durch die Konstruktion großer Märkte ist oftmals nur eine eingeschränkte Konnektivität für mobile Geräte vorhanden. Das System wird durch unterschiedliche Verhältnisse in Supermärkten oder Wochenmärkten extremen Bedingungen ausgesetzt.

Weiterhin sind verschiedene Einkaufsverhalten der Konsumenten, wie Hektik oder Gelassenheit, mitzuberücksichtigen. Bei der Nutzung eines üblichen Barcodescanners können verschiedene Probleme auftauchen. Ist der Aufdruck eines EAN Codes nicht lesbar oder existiert dieser nicht, so ist es unmöglich weitere Informationen über das System abzugreifen. Daher wird die Alternative einer manuellen Eingabe angeboten, in der auch weitere Einstellungen (z.B.: Filter) vorgenommen werden können. Auch der Einsatz im heimischen Bereich kann gefragt sein. Wenn beispielsweise ein Lebensmittelskandal veröffentlicht wird, können die Produkte im privaten Bereich abgescannt werden, um zu sehen ob diese davon betroffen sein könnten oder nicht.

Zu den potentiellen Nutzern der Funktionalitäten werden besorgte und bewusste Konsumenten gezählt. Zu diesem Personenkreis gehören Menschen, die bei Ihrem Einkauf nicht nur auf das Produkt achten, sondern auch auf die Bedingungen bezüglich der Herstellung und Lieferung. Diese Konsumenten legen neben der Liste von Zutaten auch besonderen Wert auf Inhaltsstoffe, Qualitätslabel, Allergika oder auch Arbeitsbedingungen der Mitarbeiter verschiedener Unternehmen. Die Anzeige von entsprechenden Siegeln wie „Fairtrade“[3] kann darüber Auskunft geben.

Weiterhin zählen dazu Menschen, die sich bewusst ernähren wollen und eine vegetarische oder vegane Nahrungsaufnahme bevorzugen. Trotz den vielen Möglichkeiten sich über eine alternative Ernährung zu informieren, ist die Transparenz dennoch für Konsumenten nicht ausreichend gegeben. Das Durchsuchen von Zutaten und Inhaltsstoffen dauert lange und setzt eine hohe Kenntnis voraus.

[3] http://www.fairtrade-deutschland.de/ueber-fairtrade/ - gesichtet am 15.07.2013

Zusätzlich können Allergiker auch schnell und unkompliziert Filter einsetzen, um Produkte auf bestimmte Inhaltsstoffe zu testen. Durch das OPDIS sollen solche Information strukturiert und bewertend zur Verfügung gestellt werden. Medienwirksame Skandale ziehen immer mehr Benutzer an, die mehr Transparenz von Produkten fordern.

Eine weitere Zielgruppe sind Softwareentwickler mobiler Applikationen. OPDIS- FRAME richtet sich an Softwareentwickler, die den Datenbestand und die Funktionalität von OPDIS-KNOW benutzen möchten. Daher ist es wichtig, dass die im Framework angebotenen Funktionalitäten einfach und verständlich dokumentiert sind. Die Namensgebung der bereitgestellten Klassen und ihrer Methoden sollen schon den Zweck der Klasse deuten lassen. Um die Übersicht zu wahren, ist es ebenso wichtig, die Sichtbarkeit richtig zu differenzieren. Nur die Methoden, die dem Entwickler von Interesse sein könnten, sollten die Sichtbarkeit „public“ besitzen.

Die soziale Umgebung kann ebenso viele verschiedene Formen annehmen. Aufgrund eines hohen Diskussionsbedarf, der im Kontext der Verbraucherpolitik aufkommen kann, eignet sich das System zur Aufklärung von Unklarheiten oder Missverständnissen. Durch die manuelle Eingabemöglichkeit sollten auch Produkte gefunden werden können, wenn sie einem nicht vorliegen.

Als Softwareentwickler trägt man den Benutzern gegenüber eine Verantwortung. Dies bedeutet, dass das Ziel lautet, Vorgänge einfach zu gestalten und dabei in jeglicher Form Schaden zu vermeiden. Da OPDIS für mobile Geräte konzipiert wurde, benötigt der Benutzer ein Smartphone oder Tablet. Dieses muss mit einer Kamera und mindestens mit den Betriebssystemen Android 4.0 oder iOS 6 ausgestattet sein. Ein weiterer wichtiger Punkt auf den geachtet werden muss, sind verbrauchbare Ressourcen, die die mobilen Geräte beziehen. Dies bedeutet, dass ressourcenintensive Vorgänge nicht direkt auf dem Dienstnutzer verarbeitet werden, sondern dass diese auf dem Dienstanbieter ausgelagert werden. Damit soll die Belastung des Akkumulators und der Netzverbindung reduziert werden.

1.4. Bilderkennungskomponente

Um sich von anderen Produkten abzuheben, wird eine weitere Interaktionsmöglichkeit für den Benutzer angeboten. Die Bilderkennung ist ein wissenschaftliches Forschungsgebiet und hat sich in den letzten Jahren stets weiterentwickelt. In der vorangegangenen Arbeit [Pap] wurde ein Bilderkennungsverfahren ausgewählt, das viele Kriterien erfüllt und eine gute Identifizierungsquote aufweist [ICC11]. Diese Methode ist das oriented FAST and rotated BRIEF Verfahren und soll dem OPDIS hinzugefügt werden. Durch das Scannen einer Produktrepräsentation in Form eines Bildes, soll das Produkt identifiziert werden und die zugehörigen Daten dem Benutzer liefern.

1.4.1. Vorteile der Bilderkennung

Die Vorteile dieser Interaktionsmöglichkeit liegen vorwiegend in der Gebrauchstauglichkeit und Steigerung der User Experience. Statt nach einem Barcode auf einem Produkt zu suchen, reicht es aus, ein Foto vom Produkt mitsamt des Labels zu machen. Dieses Bild wird mit einer Menge von Referenzbildern im Datenbestand des Dienstanbieters verglichen. Bei einer Übereinstimmung werden die referenzierten Produktdaten als Ergebniswerte ausgegeben. Der Unterschied zwischen den Methoden „Barcodescan“ und „Bilderkennung“ lauten wie folgt.

Beim Barcodescan wird durch das Scannen eines Strichcodes eine eindeutige Binärfolge von Zeichen bestimmt und an den Dienstanbieter gesendet. Anhand dieser binären Zeichenfolge wird eine EAN Nummer generiert und mit den Nummern im Datenbestand verglichen [Len00]. Bei der Bilderkennung hingegen gibt es keine eindeutigen Zuweisungen. Die Bilderkennung generiert aus dem Zielbild eine Matrix mit einer festgelegten Anzahl von Merkmalsextraktionen. Diese werden mit anderen Merkmalsextraktionen der Referenzbilder verglichen. Beim anschließenden Bildvergleich wird nach Übereinstimmungen in der Matrix gesucht. [ICC11]

Da die Bilderkennung noch nicht zu 100% ausgereift ist, kann keine vollständige Effektivität zugesichert werden. Allerdings bietet diese Interaktionsmöglichkeit für viele Konsumenten eine Qualitätssteigerung beim Produktscan. Wie bereits erwähnt, reicht ein Foto von einem Produkt aus, um erweiterte Informationen zu bekommen. Vor allem für Menschen mit eingeschränkten körperlichen Fähigkeiten ist diese Funktionalität von enormer Bedeutung, da die Produkte für einen Barcodescan nicht in die richtige Position gedreht werden müssen.

Auch Menschen mit Einschränkung ihres Sehvermögens wird die Benutzung erleichtert. So reicht es aus, das Produkt an sich zu fotografieren, statt ein Detail des Produkts zu suchen und zu scannen. Rollstuhlfahrer müssen beispielsweise nicht zwingend das Produkt aus dem Regal entnehmen, sondern können es direkt fotografieren. Diesbezüglich gewinnt man aus unternehmerischer Sicht weitere potentielle Benutzer für das System.

1.4.2. Aufgaben

Die Aufgabe einer Bilderkennung ist es, ein Bild nach Merkmalen zu durchsuchen. Die Merkmale werden anschließend nach Gewichtung sortiert und deren Eigenschaften protokolliert. Dahingegen müssen aktuelle Schwierigkeiten berücksichtigt werden, die im Nutzungskontext erwähnt wurden. Weiterführende Aufgaben für das System sind daher nicht zu vernachlässigen.

Durch Konnektivitätsprobleme sollten keine großen Datenpakete über das mobile Internet gesendet werden, da dieser Vorgang - je nach Verbindung - relativ schnell aber auch aber auch sehr langsam ausgeführt werden kann.

Zudem beeinträchtigt nicht nur die Zeit den Anwender, sondern auch die Auslastung des Volumenkontingents vom Provider des Benutzers. Daher sollen keine Bilder an den Dienstanbieter versendet werden, sondern direkt die Ergebnisse der Bilderkennung

vom Dienstnutzer. Somit kann das Datenvolumen eines Bildes von ca. 4 MByte auf ein 100Kbyte große Matrix reduziert werden.

Die Interaktionsschritte des Benutzers sollten auf das wesentliche Aufgabengebiet reduziert werden. Ein genereller Ablauf der Bilderkennung und des Bildvergleichs ist der Hierarchical Task Analysis 1.1 zu entnehmen. Dort werden die Tätigkeiten des Benutzers dargestellt und in der Tabelle 1.1 der Ablauf näher erläutert.

Zu den schwierigen Bedingungen im Supermarkt bzw. Wochenmarkt gehören verschiedene Luminanzen, Farbtemperaturwerte oder Fremdkörper[4]. Weiterhin können temporale Ereignisse, wie zum Beispiel hektische Vorgänge, die Qualität der Bilderkennung beeinflussen. Demnach wurde ein Bilderkennungsverfahren gewählt und im Vorfeld getestet, dass gegenüber diesen Bedingungen robust ist. Der Test und Vergleich mit dem etablierten Bilderkennungsverfahren SURF ist [Pap] oder [ICC11] zu entnehmen.

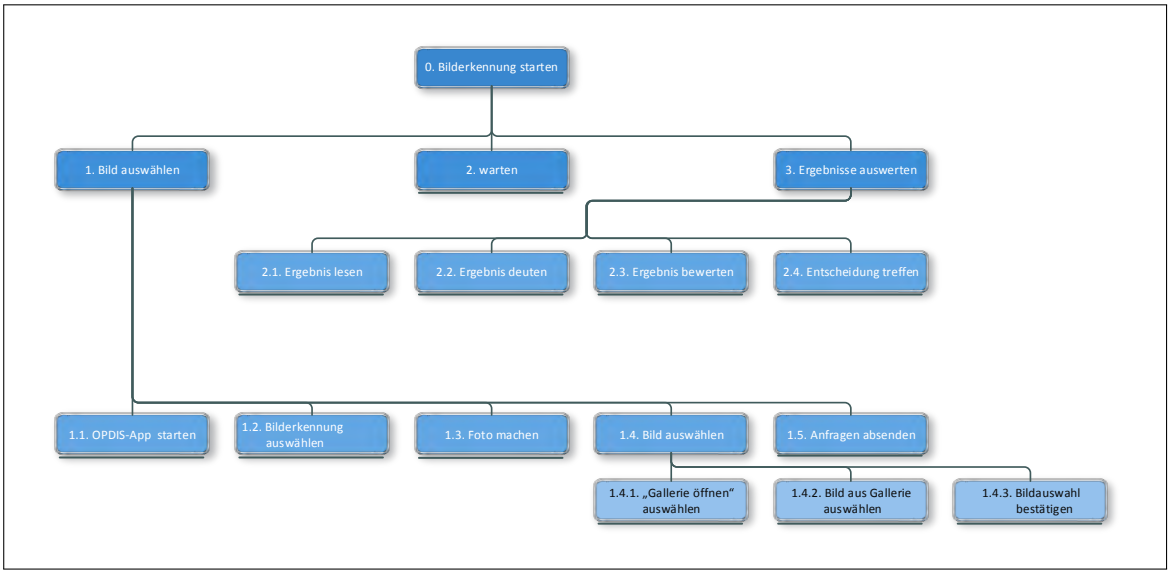

Abbildung 1.1.: Hierarchical Task Analysis: Bilderkennung durchführen.

Tabelle 1.1.: Plansequenz für das HTA in Abbildung 1.1

Vorgang	**Beschreibung**
Plan 0	DO 1 THEN 2 THEN 3
Plan 1	DO 1.1 and Wait until App is loaded completely THEN DO 1.2. THEN IF Image is NOT available DO 1.3 ELSE DO 1.4 THEN DO 1.5 THEN go to 2 OR EXIT
Plan 1.4	DO 1.4.1 THEN DO 1.4.2. THEN IF Image was selected successfully DO 1.4.3 THEN go to 1.5. OR EXIT
Plan 2	Wait until OPDIS-KNOW sent the result of the Image Recognition and Image Matching
Plan 3	Do 2.1 until User read it all THEN DO 2.2. until User understand the result THEN DO 2.3. THEN DO 2.4 THEN Repeat Procedure and go to 1 OR EXIT

;

[4]Produkte im Hintergrund die zu einem fehlerhaften Bildvergleich führen können

1.5. Gebrauchstauglichkeit

Die neue Interaktionsmöglichkeit der Bilderkennung sollte gebrauchstauglich in der alltäglichen Benutzung sein. Um in der Entwicklung stets darauf zu achten und Anhaltspunkte für Verbesserungen zu bekommen, werden im Folgenden wichtige Punkte gesammelt, um dies zu gewährleisten. Es werden kurze User Stories dargelegt, um gewisse Tätigkeiten der Benutzer zu verdeutlichen. Die Reaktion darauf sollte eine robuste Entwicklung der Bilderkennung sein. Zudem wird versucht zukünftigem Benutzerverhalten entgegenzuwirken und herkömmliche kognitive Muster der Benutzer beizubehalten. Dies hat zur Auswirkung, dass eine kognitive Entlastung des Benutzers erreicht wird. [Dix04]

1.5.1. Fehlervermeidung

Die Attraktivität einer Interaktionsmöglichkeit misst sich durch die Gebrauchstauglichkeit und Anfälligkeit von Fehlern. Demnach sollten alle bereits gesammelten Informationen stetig bei der Anwendungsentwicklung beachtet werden und eine vorausschauende Fehlerkorrektur gewährleistet werden. Der Ansatz des „Usability Engineerings“ stellt eine gute Grundlage dar. In Verbindung mit den „User Stories“ aus Anhang A, die aus den Aufgaben des Nutzungskontext definiert wurden, werden die potentiellen groben Fehler abgedeckt. Während der Anwendungsentwicklung wird bewusst auf Java Exceptions eingegangen, um einen fehlerfreien Ablauf im Bilderkennungsdialog zu gewährleisten.

1.5.2. Lösungsansätze

Bezüglich der User Stories A werden Lösungsansätze aufgezeigt, die für den Verlauf des Projekts verfolgt werden. in der User Story A.1 kommt es zu keinem Produktscan, weil keine Konnektivität zustande kommt. Um dennoch einen Produktscan gewährleisten zu können, wird durch eine Kamerafunktionalität das Bild in einem extra angelegten Ordner archiviert. Zu einem späteren Zeitpunkt kann der Benutzer dieses Bild aus der Android-Galerie auswählen und mit der Bildauswahl einen Produktscan vornehmen.

Der Vorteil gegenüber der manuellen Eingabe von Produktinformationen (z.B.: Produktname) ist die kognitive Entlastung des Benutzers. Dieser muss sich kein Produktnamen oder Ähnliches merken oder notieren, sondern kann direkt auf das Bild aus der Android Galerie zugreifen und anhand der Bilderkennung die erweiterten Produktinformationen abrufen.

In der User Story A.2 wird ein erfolgreicher Produktscan durchgeführt. Der Dialogablauf sollte zu jedem Zeitpunkt der Bilderkennung den aktuellen Status anzeigen. Der Dialogablauf der Bilderkennung wird automatisiert in einem asynchronen Prozess ablaufen, so dass die Benutzeroberfläche während des Prozesses aktualisiert werden kann. Sollte optimalerweise nur ein Produkt dem Bild zugeordnet werden können, ist es empfehlenswert direkt auf die Produktseite zu verweisen. Sollte aber nur ein bedingt erfolgreicher Produktscan wie in der User Story DA.4 durchlaufen und mehrere Produkte zum Ziel-

bild finden, dann sollte diese in einer Liste ausgegeben werden. Diese Liste wird sortiert, sodass das Element an oberster Stelle in der Liste die meisten Übereinstimmungen hat.

Sollte kein Produkt gefunden werden, muss dieses dem Benutzer ebenfalls mitgeteilt werden. Das kein Ergebnis erzielt werden kann, hängt von vielen Faktoren ab. Unter anderem ist es möglich, dass keine Datenbankeinträge zu Produkten vorhanden sind, Labelveränderungen an Produkten vorgenommen wurden, die Qualität des Zielbilder keine genaue Bilderkennung zulässt oder das Fehler im System auftreten.

1.6. Projektziel

In meiner vorangegangenen Praxisprojektarbeit „Konzeption und Vergleich von Bilderkennungsverfahren zur Filterung von Produkteigenschaften“ [Pap] wurden Bilderkennungsmethoden auf Brauchbarkeit und Robustheit getestet. Es wurde die Bilderkennungsmethode ORB (= oriented FAST and rotated BRIEF) ausgewählt, um dieses Verfahren für den Produktscan im Forschungsprojekt OPDIS (= Open Product Data Information System) einzusetzen.

In dieser Arbeit werden die Möglichkeiten und Schwierigkeiten bei der Implementierung aufgegriffen und die Brauchbarkeit des ORB Verfahrens in einem praktischen Umfeld getestet. Durch den Einsatz im praxisbezogenen Projekt soll die Einsatzfähigkeit in wechselnden Umgebungen getestet werden.

2. ORB-Bilderkennungsverfahren

Basierend auf den Ergebnissen der Praxisprojektarbeit „Konzeption und Vergleich von Bilderkennungsverfahren zur Filterung von Produkteigenschaften“ wurde die Methode oriented FAST and rotated BRIEF ausgewählt. Im Bezug auf Performanz und Effektivität eignet sich das Verfahren auch für mobile Geräte. Diesbezüglich kann die Bilderkennung und der Bildvergleich logisch voneinander getrennt werden, sodass eine verteilte Anwendung entsteht. ORB ist in der Bildbearbeitungsbibliothek OpenCV enthalten, sodass die OpenCV Umgebung in ein verteiltes Java-System implementiert wird. Für den Dienstanbieter wird Java 6 [1] verwendet und für den Dienstnutzer wird Java basierend auf dem Android SDK umgesetzt.

2.1. oriented FAST and rotated BRIEF

Das oriented FAST and rotated BRIEF Bilderkennungsverfahren wurde erstmalig auf der International Conference for Computer Vision 2011[2] vorgestellt. Das Verfahren besteht aus einem Detektor-Deskriptor-Schema. Der Name setzt sich aus dem Detektor „oriented FAST“ und aus dem Deskriptor „rotated BRIEF“ zusammen. Die Abhandlung, die aus der ICCV 2011 entstanden ist, beinhaltet zusätzlich zur Erklärung des Verfahrens auch Vergleiche mit anderen etablierten Bilderkennungsverfahren. [ICC11] Im Folgenden werden die einzelnen Komponenten des ORB-Verfahrens beschrieben.

2.1.1. Detektor

Ein Detektor ist für das Auffinden von Merkmalen in einem Bild zuständig. Je nach Detektortyp können unterschiedliche Faktoren eine Rolle spielen. Die Merkmalsextraktion beim „oriented FAST“ wird anhand einer Eckpunktdetektion des FAST-9 Detektors vorgenommen. Die 9 bestimmt den zirkulären Radius, um einen zentralen Pixel, der einen Eckpunktkandidaten darstellt. Die Pixel auf dem Umfang des Radius sind für die Ermittlung des Eckpunktes zuständig. [Ros99]

Während der Bilderkennung wird angestrebt mehr Merkmalsextraktionen zu finden als angegeben wurden. Die Merkmalsextraktionen werden anschließend nach [**?**] sortiert. So wird sichergestellt, dass die aussagekräftigsten Merkmale als Ergebnis geliefert werden.

Der FAST Detektor erkennt vor der Bearbeitung des Bildes keine direkten Eckpunkte. Daher würde FAST auch Kanten bearbeiten. Um Berechnungszeit einzusparen, werden

[1] Aus Kompatibilitätsgründen mit dem Automatisierungstool Gradle wird Java 7 noch nicht verwendet. Siehe auch 3.2.1

[2] http://www.iccv2011.org/ - gesichtet am 21.06.2013

Bildinformationen entzogen. Durch das Anwenden des Harris Corner Filters [HS88] werden Kanten verworfen und nur die Eckpunkte beachtet. Weiterhin ermöglicht der Einsatz des Pyramidenschemas eine Skalierungsinvarianz. Damit können Bilder mit unterschiedlichen Auflösungen verglichen werden. Die Funktionsweise einer Pyramidenskalierung wird unter anderem in [Jäh05, S.147] behandelt.

Anders als die meisten Detektoren besitzt FAST keine Ausrichtungskomponente zur Orientierung des Eckpunktes. Daher wird der Zusatz „oriented“ verwendet, um zu verdeutlichen, dass ein zusätzlicher Operator hinzugefügt wurde. Dieser Operator ist der “Intensity Centroid„ [Ros99]. Beim Intensity Centroid wird die Ausrichtung vom Eckpunkt (O) zum Intensitätsschwerpunkt(C) ermittelt. Dabei wird ausgegangen, dass der Intensitätsschwerpunkt abseits des Eckenmittelpunktes ist. Somit entsteht der Ortsvektor $\overrightarrow{OC}$ der die Orientierung des Keypoints bestimmt. Die folgenden Eigenschaften werden dem Deskriptor zur Weiterverarbeitung für jeden Keypoint übermittelt:

- X-Koordinate
- Y-Koordinate
- Ausrichtung
- Durchmesser
- Stärke des Keypoints
- Pyramidenebene

Der Vorteil des FAST Detektors ist der geringe Berechnungsaufwand der auf den Einsatzgeräten stattfindet. Daher eignet sich dieser Detektor besonders gut für mobile Geräte, wie zum Beispiel Tablets oder Smartphones.

2.1.2. Deskriptor

Der Deskriptor beschreibt die Merkmalsextraktionen des Detektors näher. Der in ORB angewendete Deskriptor heißt rotated Binary Robust Independet Elementary Features (Abk.: rBRIEF) und besteht ebenfalls aus zwei Komponenten. Es ist ein Deskriptor, der sensibel gegen Drehungen ist und somit nur wenige Grad an perspektivischen Unterschieden zulässt [CLSF10] . Um dem entgegenzuwirken wird eine Rotationskomponente hinzugefügt. Durch die Umwandlung in ein Integralbild wird es ermöglicht, das Bild zu glätten und auf einer Bildbearbeitungsebene zu arbeiten, die bekannt für schnelle Operationen ist.

Jeder Keypointkandidat wird in einem separaten Bildausschnitt bearbeitet. Dabei werden binäre Intensitätstests in geglätteten Bildausschnitten durchgeführt. Es wird ein 256 Bit Vektor erstellt, dessen Inhalt ein binärer String mit Intensitätswerten ist. Die binären Tests bewerten den Mittelwert einer Distanz von einem Ausgangspixel zu der Nachbarschaftsumgebung. Nur wenn die Tests unkorreliert sind, werden diese als Ergebniswert verwendet. [ICC11]

Nach Abschluss werden 500 Deskriptoren in eine Matrix eingetragen, die jeweils einen Vektor mit der Länge von 32 Werten bilden. Dieses Verfahren wird ausführlich in [RRKB11] beschrieben. Der Vorteil beim rBRIEF ist die Robustheit gegenüber Drehungen und verschiedenen perspektivischen Sichtwinkeln. Durch diese Robustheit erlangt das ORB Verfahren einige Invarianzen, die vom Projektkontext verlangt werden.

2.1.3. Bildvergleich

Der Bildvergleich beim ORB Verfahren verwendet eine „Nearest Neighbour Search“. Dabei wird das „Locality Sensitive Hashing“ (Abk.: LSH) [GIM99] eingesetzt, um Deskriptorübereinstimmungen zu finden. Der Vergleich findet mit einem Abfragedeskriptor in verschiedenen Hashtabellen statt. Potentielle Kandidaten für Übereinstimmungen werden über eine BruteForce-Methode ermittelt, deren Hamming Distanz ausschlaggebend für eine positive Übereinstimmung ist. Durch das Hashing wird ein Performanzgewinn im Gegensatz zu linearen Algorithmen erzielt. Einen genaueren Einblick in das Hashverfahren bietet das Buch [CSRL01]

2.2. Bildoptimierung

ORB bietet viele Invarianzen an, um bei verschiedenen Eigenschaften ein positives Matching zu gewährleisten. Allerdings gaben Tests darüber Aufschluss, dass die Anzahl der Übereinstimmungspunkte zurückgewichen ist , sobald zu große Skalierungsunterschiede vorhanden waren(siehe auch [Pap]). Daher sollen beim Dienstnutzer Vorbereitungen getroffen werden, die eine gemeinsame Basis der zu vergleichenden Bilder schaffen.

2.2.1. Referenzbild

Referenzbilder sind Abbildungen von Produkten, die sich im Datenbestand des Dienstanbieters befinden. Um keine irreführenden oder verfälschten Ergebnisse während des Bildvergleichs zu erhalten, sollten die Referenzbilder einige Bedingungen erfüllen.

Ein Referenzbild sollte einen neutralen Hintergrund besitzen, der sich vom Produktbild absetzt. Das Bild sollte optimalerweise zugeschnitten werden, damit nur wenig Hintergrund existiert. Durch diese Maßnahmen wird das Risiko der falschen Zufallstreffer während eines Bildvergleichs verringert, da nur das Objekt an sich erkannt wird. Ein Beispiel eines Referenzbildes ist in der Abbildung 2.1 zu sehen.

Die Bilderkennungsmethode ORB ist skalierungsinvariant, dennoch waren bei vorangegangenen Tests Unterschiede in der Merkmalsfindung zu sehen. Die Auflösung 500x666 Pixel erwies sich aus zwei Gründen als akzeptabel 2.1. Zum einen verläuft die Bilderkennung schneller als auf Bildern mit hoher Auflösung, da bei einer höheren Auflösung auch mehr Merkmalsextraktionen gefunden werden. Und zum anderen wird eine geringere Speicherkapazität für den Dienstanbieter benötigt. Um eine hohe Wiedererkennungsrate zu erzielen, ist es empfehlenswert, von einem Produkt mehrere Referenzbilder anzulegen.

Abbildung 2.1.: Beispiel eines Referenzbildes

Tabelle 2.1.: ORB-Ergebnisswerte des Skalierungsinvarianz-Tests

	1500x2000	1000x1333	750x1000	500x666	400x533	300x400	150x200
1500x2000	100 %	100 %	94 %	94 %	83 %	0 %	0 %
1000x1333	100 %	100 %	97 %	100 %	91 %	88 %	0 %
750x1000	94 %	97 %	100 %	100 %	94 %	91 %	100 %
500x666	94 %	100 %	100 %	100 %	98 %	100 %	86 %
400x533	83 %	91 %	94 %	98 %	100 %	100 %	95 %
300x400	0 %	88 %	91 %	100 %	100 %	100 %	97 %
150x200	0 %	0 %	100 %	86 %	95 %	97 %	100 %

Denn bei einer perspektivischen Drehung können Bilder zwar auch erkannt werden, Allerdings werden mehr Übereinstimmungen gefunden, wenn das Zielbild und das Referenzbild aus der selben Perspektive aufgenommen werden. Sollten es die Rahmenbedingungen also zulassen, sind mehrere Bilder aus verschiedenen Perspektiven empfehlenswert.

In Abbildung 2.2 ist die Verteilung der Merkmalsextraktion auf verschiedenen Referenzbildern zu sehen. Aus der Verteilung der Merkmalsextraktionen lässt sich schließen, dass bei einem guten Referenzbild die Konzentration der Punkte auf der Produktrepräsentation liegt. Merkmale außerhalb des Produktes werden wenig oder gar nicht extrahiert.

In dem Negativbeispiel sind mehrere Produkte auf einmal beworben. Da beim ORB Verfahren nur eine begrenzte Anzahl an Merkmalsextraktionen verwendet wird, verteilen diese sich über das komplette Bild. Die Konzentration der Keypoints befinden sich zwar überwiegend auf den Produktrepräsentationen, da allerdings mehrere Repräsentation verschiedener Produkte darauf enthalten sind, kann dies bei einem Bildvergleich zu Störungen führen. Zudem wird die geringere Anzahl der Merkmalsextraktionen auf einem Abbild die Chance eines Bildvergleichs verschlechtern.

2.2.2. Zielbild

Das Zielbild ist die Abbildung von dem Objekt, das mit der Smartphonekamera aufgenommen wurde. Wie bereits in 2.2.1 erwähnt, ist es empfehlenswert, Bilder zu vergleichen

Abbildung 2.2.: Bewertung der Referenzbilder

deren Auflösungen nur gering voneinander abweichen. Bei der Android-Entwicklung stehen zwei Methoden zur Auswahl.

Die erste Methode ist die Verwendung der Android-internen Kameraapplikation. Wird die interne Androidkamera während der Entwicklung benutzt, so sollte eine manuelle Manipulation des aufgenommenen Bildes gemacht werden. Durch den Skalierungskoeffizienten inSampleSize für Bitmaps lässt sich ein vorhandenes Bild in die gewünschte Auflösung skalieren. Der InSampleSize-Koeffizient muss für jede Auflösung neu berechnet werden. Als erstes müssen die Seitenverhältnisse von Breite und Höhe berechnet werden. Anschließend wird geprüft, welches Seitenverhältnis geringer ist. Das geringere Seitenverhältnis wird danach gerundet und als InSampleSize-Koeffizient genutzt.

$$Breitenverhältnis = aktuelleBreite/Zielbreite$$
$$Höhenverhältnis = aktuelleHöhe/Zielhöhe$$

Der Koeffizient InSampleSize berechnet sich aus

$$Höhenverhältnis < Breitenverhältnis?Höhenverhältnis : Breitenverhältnis$$

Weiterhin werden die Bilder bei einigen Androidgeräten bei der Generierung durch die Kamera um 90° gedreht. Um das Bild auf seinen Ursprung zurückzudrehen, müssen Metainformationen des Bildes ausgelesen und überprüft werden. Diese Metainformationen befinden sich in einem EXIF Eintrag des Bildes. Darin befindet sich unter anderem auch die Orientierung des Bildes. Diese Option sollte bei jedem Bild ausgelesen werden und wenn eine Rotation erforderlich ist, muss dieser Wert geändert werden.

Die zweite Möglichkeit besteht in der Entwicklung einer eigenen Kamera. Die Kamera API, die vom Android SDK bereitgestellt wird, enthält viele weitere Eigenschaften, die

manuell hinzugefügt werden können. Wählt man die Option der eigenen Kamera, so kann man die Methode setPictureSize(width, height) benutzen, um eine Zielauflösung zu definieren.

2.3. Marktanalyse

Im Jahr 2010 wurde bereits ein System entwickelt, das Produkte über eine Bilderkennung erkennen konnte. Das „mobile product recognition“ [TCC+10] wurde auf Basis des SURF Verfahrens [BTG06] in Verbindung mit der CHoG Kompression [CTC+09] entwickelt. In der Ressourcenverwaltung dieses Systems waren mehr als eine Millionen Einträge die laut Publikation [TCC+10] erfolgreiche Bildvergleiche generierten. Da die vorangegangenen Tests aus [RRKB11] allerdings effektiver und effizienter waren und die eigenen Tests dies bestätigten 2.2 2.3, besteht die Annahme eine besseres System zu schaffen.

Perspektivische Drehung	SURF Übereinstimmungen	ORB Übereinstimmungen
90°	15	17
100°	26	23
120°	4	5
140°	5	6
160°	1	8
180°	1	2

Tabelle 2.2.: Ergebnissewerte des Perspektivtests

Verfahren	Bildgröße	Bildauflösung	Merkmale	Detektorzeit	Deskriptorzeit
ORB	2,33 MB	3264x2448 Px	500	78 ms	595 ms
SURF	2,33 MB	3264x2448 Px	2155	3391 ms	595 ms

Tabelle 2.3.: Ergebnissewerte der Schnelligkeit

Die mächtigste mobile Bilderkennungsapplikation bietet zur Zeit Google Inc. mit Goggles [Goo]. Bei Goggles wird nicht nur eine Bilderkennung durchgeführt, sondern auch eine Texterkennung. Weiterhin ist in der Kameraapplikation auch ein Barcodescanner integriert. Für den produktspezifischen Kontext ist Goggles in der Lage Logos zu erkennen und zuzuordnen. Bei einer erfolgreichen Bilderkennung wird eine Suche in der Google Suchmaschine initialisiert und die Suchergebnisse dem Benutzer präsentiert. Durch die Nutzung der Google Suchmaschine ist die Applikation in der Lage, ein sehr breites Spektrum an Ressourcen abzudecken. Allerdings werden bei einer Produktsuche nur Preise und allgemeine Informationen bereitgestellt. Spezialisierte Informationen werden nicht explizit als Ergebniswerte übertragen. Zudem besteht der Nachteil, dass Goggles jederzeit im Hintergrund aktiv ist und dabei ebenfalls die Standard Androidkamera benutzt. So werden auch zu einem unerwarteten Zeitpunkten Ergebnisse geliefert.

Ebenfalls werden Frameworks angeboten, die das Backend für eine Bilderkennung bilden. Kooaba [Koo] ist einer der bekanntesten Anbieter und bildet die komplette Anwendungslogik für die Bilderkennung und auch das Ressourcenmanagement wird von

Kooaba übernommen. Mit Hilfe dieses Anbieters, ist es gegen Gebühren möglich, eine eigene mobile Applikation zu entwerfen. Der Vorteil bei diesem Vorgehen ist der minimale Entwicklungsaufwand. Es kann sich auf die Präsentationslogik und auf die Gebrauchstauglichkeit konzentriert werden und den Backend Service den Rest überlassen. Unsicher ist die Individualisierbarkeit der benötigten Funktionalitäten.

Weiterhin gibt es diverse andere Bilderkennungsapplikationen, um Plakate zu fotografieren und anschließend direkt auf Ticketbestellungsportale oder andere spezielle Websites weitergeleitet zu werden. Diese Form der Bilderkennung wird mit Wasserzeichen innerhalb der Bilder gewährleistet. Diese sind für das menschliche Auge nicht direkt sichtbar und werden als eine Art Barcode kodiert. Für diese Art der Bilderkennung ist es notwendig, auch die Zielressource zu manipulieren. Dieses Verfahren wurde aus diesem Grund von Beginn an nicht berücksichtigt und wird nur der Vollständigkeit halber erwähnt.

3. Systemumgebung und Architektur

In diesem Kapitel wird eine Übersicht geboten, die die konzeptionelle Darstellung des Systems genauer betrachtet. Dabei werden die genutzten Technologien schematisch dargestellt und die einzelnen Komponenten beschrieben. Auch der Datenverkehr und die genutzten Kommunikationsmittel werden näher erläutert.

3.1. Systemarchitektur

Das System besteht aus mobilen Dienstnutzern, die eine Kommunikation zu einem RESTful Webservice aufbauen. Der RESTful Werbservice enthält persistente Daten, die bei Abfragen aufbereitet und zur Verfügung gestellt werden. Für den Dienstnutzer wird ein eigenes Framework entwickelt, um Anwendungsentwicklern den Umgang mit dem RESTful Dienstanbieter zu vereinfachen. Dabei wird darauf geachtet, dass eine gute Performanz eingehalten wird und der Kommunikationsablauf zwischen Dienstnutzer und Dienstanbieter möglichst ohne Probleme stattfindet. Der Dienstnutzer kann individuell eingerichtet werden und mit Hilfe des Frameworks eine Verbindung zum RESTful Webservice aufbauen. In der Abbildung 3.1 ist zu erkennen, dass es ein verteiltes Sys-

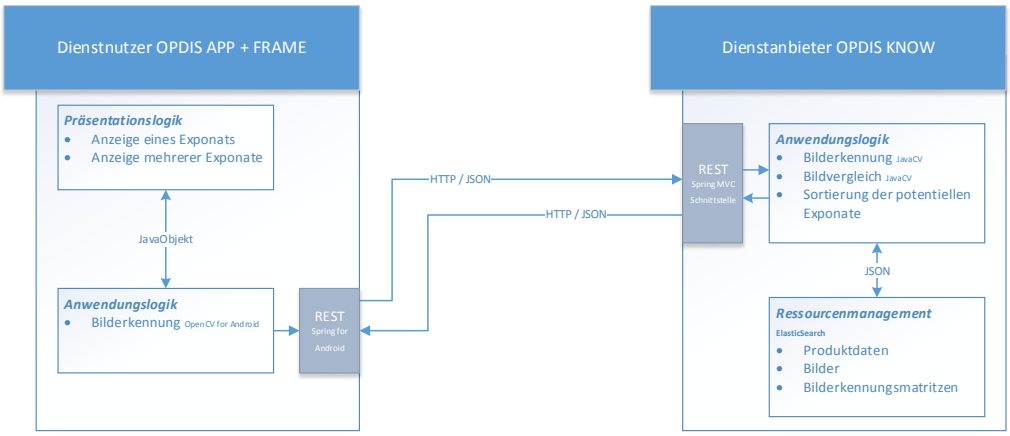

Abbildung 3.1.: Systemarchitektur: Modellierung der Systemkomponenten in eine Systemarchitektur

tem ist. Die Anwendungslogik wird auf zwei verschiedenen Komponenten verteilt. Auf dem Dienstnutzer wird die Bilderkennung ausgeführt und auf der Gegenseite der Bildvergleich mit der Datenbasis. Der Zugriff auf die jeweiligen Logiken erfolgt über eine REST-Schnittstelle.

Auf dem Dienstnutzer wird die Präsentationslogik durch die Anwendungslogik angesteuert. Dabei werden die benötigten Daten, die für das Aufbauen des User Interfaces zuständig sind, bereitgestellt. Der Austausch zwischen der Anwendungslogik und der Präsentationslogik erfolgt über eine asynchrone Kommunikation.

Die asynchrone Kommunikation ermöglicht einen Hintergrundprozess, der parallel zum User Interface-Prozess läuft. Die Ergebnisse aus dem Hintergrundprozess können über bestimmte Methoden, die Zugriff auf das User Interface haben, an den Vordergrundprozess gesendet werden.

Das Ressourcenmanagement wird von der Anwendungslogik des Dienstanbieters benutzt, um den Bildvergleich zu realisieren. Dabei werden vor dem Bildvergleich die Bildmatrizen aus der ElasticSearch Datenbank ausgelesen. Sollte der Bildvergleich erfolgreich sein, werden vom Dienstnutzer die erforderlichen Ressourcen aus der Datenbasis bereitgestellt.

3.1.1. Systemkomponenten

Das Open Product Data Information System (OPDIS) ist eine verteilte Anwendung, die den Benutzern mehr Transparenz bei ihren Kaufentscheidungen bereitstellt. Die Anwendung spezialisiert sich auf Produkte aus dem Drogerie- und Lebensmittelbereich. Dabei werden unterschiedliche Quellen verwendet, um an Informationen wie Produktbeschreibung, Produktwarnungen, Allergiehinweise, Qualitätslabel, Inhaltsstoffe, Hersteller und den Vertrieb zu gelangen. Um als Benutzer die Daten zu bekommen, werden verschiedene Wege angeboten. Zum einen kann der Benutzer die mobile Applikation erforschen, indem dieser mehrere Kategorien nach Produkten, die in einer Liste angeordnet sind, durchsucht. Zum anderen werden Funktionalitäten angeboten, die eine gezielte Suche über eine Eingabemaske ermöglichen und einen Scan des Barcodes am Produkt vornehmen können. Um sich von anderen mobilen Applikationen abzuheben, wird eine Bilderkennungskomponente eingebunden. Die Bilderkennungskomponente ermöglicht nach einer Abbildung des Produktes zu suchen. Aus dieser Abbildung werden Merkmale extrahiert, die mit anderen Merkmalen von unterschiedlichen Bilder verglichen werden. Diese Softwarekomponente wird auf mehrere Systemkomponenten verteilt. Die Trennung von Anwendungslogik, Präsentationslogik und Ressourcenmanagement wird insgesamt auf zwei Systemkomponenten verteilt.

3.1.2. Dienstanbieter

Der Dienstanbieter im System heißt OPDIS-KNOW und ist ein RESTful Webservice. OPDIS KNOW ist für die Bereitstellung strukturierter Daten zuständig und enthält die Anwendungslogik der Bilderkennung, des Bildvergleichs und das Ressourcenmanagement. Mittels der standardisierten HTTP-Requests GET, POST, PUT und DELETE wird auf Controller zugegriffen, die auf weitere verschiedene Services verweisen. Die REST Controller, die die REST-Schnittstelle bilden, werden mit dem Spring MVC Framework umgesetzt. Um Objekte über die REST-Schnittstelle zu senden, wurde das JSON

Format als Standard definiert. Die Nutzung des HTTP Headers und des HTTP Bodys ermöglicht es, Objekte genauer zu spezifizieren. Die Serialisierung und Deserialisierung findet mit dem Spring MVC kompatiblen Jackson FasterXML statt. Mithilfe des Spring MVC Frameworks und Jackson FasterXML werden Objekte beim Senden serialisiert und beim Empfangen deserialisiert.

Für die Umsetzung der Bilderkennung und des Bildvergleichs wird eine neue Komponente entwickelt. Diese beinhaltet einen Controller, der auf die Methoden eines zusätzlichen Bilderkennungsservices zugreift. Der Service stellt die Anwendungslogik für den projektspezifischen Kontext dar und beinhaltet den Bildvergleich des Zielbildes vom Dienstnutzer und die Referenzbilder der Datenbank.

Dazu wird das Java Native Interface (JNI) JavaCV verwendet, um auf die C-basierten Klassen von OpenCV zuzugreifen. Bedeutsam ist die Entscheidung, welche Daten über das mobile Netzwerk gesendet werden sollen und welche Daten der Dienstanbieter entgegennehmen kann. Die Verantwortlichkeit der Applikation gegenüber dem Benutzer, sollte ein schneller Ablauf und das Senden von so wenigen und kleinen Datenpaketen wie nötig sein . Daher wäre es unverantwortlich, ein komplettes Bild dem Dienstanbieter zu übertragen. Das Zersetzen und Zusammenfügen eines Bildes könnte im Base64 Format ablaufen, aber dennoch sind die Datenpakete zu groß und nehmen auf Dauer zu viel Übertragungsvolumen des Benutzers in Anspruch.

Die bessere Alternative ist, das Resultat einer Bilderkennung in Form einer Matrix zu übertragen. Somit ist es möglich, eine Matrix zu empfangen die ca. 70 - 100 KByte im JSON Format groß ist, anstatt ein Bild welches 2-4 MB groß sein kann. Somit wird auch ein Teil der Anwendungslogik auf den Dienstnutzer verlagert und Ressourcen werden für weitere Prozesse auf dem Dienstanbieter freigeräumt.

Die Auslastung des Dienstanbieters wird weiterhin gering gehalten, indem eine Initialbilderkennung der Referenzbilder stattfindet. Somit muss nicht für jeden Bildvergleich eine erneute Bilderkennung vorgenommen werden, sondern es kann auf fertige Matrizen im JSON Format zurückgegriffen werden. Diese Matrizen müssen zusätzlich mit einer Produkt ID und einer Bild ID versehen werden, um die Matrizen eindeutig zuordnen zu können. Die Referenzbilder selbst werden im Base64-Format in einer NoSQL Datenbank abgelegt. Genau wie die Matrizen der Bilderkennung werden auch die Bilder im JSON Format abgespeichert und einem Produkt mit einer Produkt ID zugeordnet. Weiterhin ist eine selbständige Bilderkennung geplant, die durchgeführt wird, sobald ein neues Bild der Datenbank hinzukommt.

Der Bildvergleich selbst findet auf Basis von Matrizen statt. Vom Dienstnutzer wird eine bereits fertige Bilderkennungsmatrix vom Controller in Empfang genommen und an den Bilderkennungsservice weitergeleitet. Diese Matrix wird anschließend mit allen in der Datenbank vorhandenen Matrizen abgeglichen. Während des Abgleichs wird eine Liste angelegt, die in sortierter Reihenfolge nach Anzahl der Übereinstimmungen die Ergebnisse repräsentiert.

Das Ressourcenmanagement befindet sich ebenfalls auf OPDIS KNOW in Form einer ElasticSearch Datenbank. Darin werden alle Produktdaten, Bilder und Matrizen im JSON Format archiviert. Die Bilder sind im Base 64 Format kodiert, sodass eine einfache Übertragung gewährleistest ist. Die Bilder und auch die Matrizen besitzen IDs, um zum richtigen Produkt referenzieren zu können. Die Bilder erhalten eine Produkt ID und die Matrizen werden mit einer Produkt ID und Bild ID erweitert. Beim Bildvergleich werden nacheinander die Matrizen aus der ElasticSearch Datenbank ausgelesen und bei der Bilderkennung werden die Matrizen darin gespeichert.

3.1.3. Dienstnutzer

Der Dienstnutzer besteht aus zwei Komponenten. Zum einen wird ein Framework entwickelt, in dem der Verbindungsaufbau, die Anwendungslogik und die Kommunikation mit dem Dienstanbieter geregelt wird. Und zum anderen eine individuelle mobile Applikation, die auf das Framework zugreift.

Die Anwendungslogik ist im Kontext des OPDIS-Prototypen gänzlich auf das Framework ausgelagert. Die Präsentationslogik wird vollständig von der mobilen Applikation übernommen. Der Verbindungsaufbau und die HTTP Requests zum Dienstanbieter werden durch einen Networkmanager bereitgestellt. Diese Klasse greift auf das externe Framework „Spring for Android“ zurück.

Das Framework bietet kompakte Methoden an, um sichere Verbindungen mit wenig Overhead aufzubauen. Nur leichte Modifikationen an den Einstellungen sind vorzunehmen, um zum Beispiel dem Server über einen HTTP Headereintrag mitzuteilen, dass das Datenpaket im JSON Format vorhanden ist. Die JSON-Objekte werden durch Jackson FasterXML verwaltet. Spring for Android bietet die Funktion an, JSON Objekte in einen HTTP Request umzuwandeln, sodass die Entwicklung einer manuellen Serialisierung beziehungsweise Deserialisierung nicht gebraucht wird. Die mobile Applikation wird im Rahmen eines Prototypen entwickelt, die die Funktionalitäten des Frameworks hervorheben soll. Die Bilderkennung des Zielbildes wird auf Dienstnutzerseite durchgeführt. Die Logik dazu wird im Framework mit Hilfe des „OpenCV for Android SDK“ umgesetzt. Dort wird das Zielbild, welches aus einer Kameraaufnahme oder aus der Galerie gewählt wird, analysiert. Die Ergebnisse des ORB Verfahrens werden in einer Matrix gespeichert, die an den Dienstanbieter gesendet wird.

Zusätzlich werden Hilfsklassen benötigt, die die Zielbilder für den Bildvergleich vorbereiten. Zum einen eine Ressourcenverwaltung, die eine schnelle und individuelle Archivierung gewährleistet. Zum anderen eine Optimierungsklasse, die das Bild vor der Bilderkennung skaliert und wenn nötig rotiert.

3.1.4. Kommunikationsmodell

Der Dienstanbieter im System heißt OPDIS-KNOW und verwaltet Produktdaten. OPDIS-KNOW ist ein RESTful Webservice. Das heißt, dass mittels normaler HTTP-Anfragen Ressourcen abgerufen werden können. Zu den HTTP Anfragen gehören die Prädikate

GET, POST, PUT und DELETE. Die REST-Schnittstelle bietet dem Dienstnutzer mittels dieser Prädikate an, Produktdaten mit verschiedenen Mitteln abzurufen. Es kann eine direkte Auswahl anhand von Produkt IDs oder Produktnamen stattfinden. Diese Möglichkeiten würden ein GET-Request über das HTTP veranlassen und somit die Ressource eines Produktes zur Verfügung stellen. Für detailliertere Informationen über eine RESTful Kommunikation verweise ich auf [Til09]. Im Kommunikationsmodell 3.2

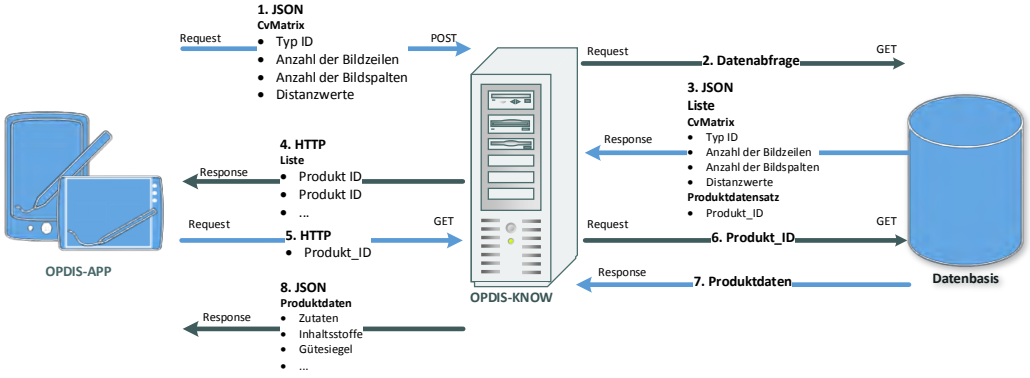

Abbildung 3.2.: Kommunikationsmodell: Modellierung einer Kommunikation zwischen dem Dienstnutzer und Dienstanbieter bei einem Bildvergleich

wird dargestellt, welche Prädikate genutzt werden und welche Daten übertragen werden. Durch die Bilderkennung auf dem Dienstnutzer wird ein JSON-Objekt an den Dienstanbieter gesendet. Dieses Objekt muss von der REST-Schnittstelle mithilfe eines @consumes verarbeitet werden. Das @consumes definiert die einkommenden Daten über die REST-Schnittstelle. Dabei wird festgelegt, dass nur JSON Dateien akzeptiert werden.

Durch das HTTP Prädikat @POST wird definiert, dass eine Ressource erstellt oder aktualisiert wird. Dabei wird das JSON Objekt aus der JSON Datei extrahiert und als Matrix Objekt auf dem Dienstanbieter verarbeitet. Mit diesem Matrix-Objekt wird ein anschließender Bildvergleich mit den Ressourcen aus der Datenbank vorgenommen. Die übereinstimmenden Ressourcen werden anschließend einer Liste hinzugefügt. Da allerdings ein GET-Request auf HTTP Basis die REST-Schnittstelle ansprechen muss, um detaillierte Produktdaten zu erhalten, wird die Liste der übereinstimmenden Ressourcen mit einer HTTP Response an den Dienstnutzer zurückgesendet. Dem Dienstnutzer wird anhand des HTTP Response eine Liste von 0 bis 3 Entitäten angeboten, aus dem der Benutzer das passende Produkt wählen kann. Ein optimales Ergebnis ist ein einziger Eintrag in der Liste, der genau das Produkt widerspiegelt, welches die Bilderkennung auf dem Dienstnutzer veranlasste. Erst nach der Wahl des Produktes aus der Liste, wird ein GET-Request an den Dienstanbieter gesendet.

Dabei wird die Produkt ID mitgeliefert und die Ressource kann eindeutig auf dem Dienstanbieter zugeordnet werden. Nach der Zuweisung der Ressource werden die Daten von OPDIS-KNOW aufbereitet. Durch den Zusatz @produces wird bei der @GET Methode definiert, dass ein bestimmtes Format gewählt wird, um eine Response zu ver-

senden. In diesem Fall wird JSON gewählt, um eine Abbildung der Ressource an den Dienstnutzer zu senden.

3.1.5. Datenmodell

Das Datenmodell 3.3 stellt eine Abbildung der internen Kommunikation der Softwarekomponenten dar. Dabei werden die Formate und auch die wichtigsten Objektdaten definiert. Beim Dienstanbieter OPDIS-KNOW wird die Anwendungslogik im Bezug auf den Ablauf eines Bildvergleichs dargestellt. Die REST-Schnittstelle empfängt eine JSON-

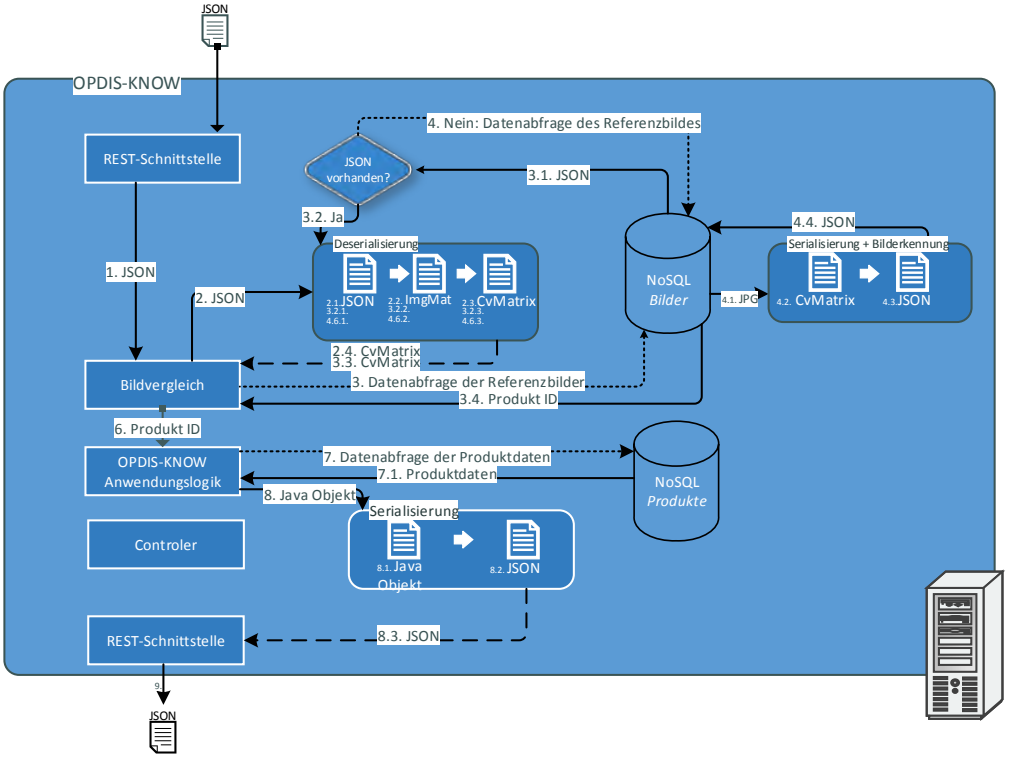

Abbildung 3.3.: Datenmodell: Modellierung der internen Kommunikationstypen von OPDIS-KNOW

Datei vom Dienstnutzer. Diese Datei wird direkt an die Bildvergleichs-Komponente weitergeleitet, da die Bilderkennung schon auf dem Dienstnutzer ausgeführt wurde. Dabei wird das JSON Objekt aus der Datei in ein ImageMatrix-Objekt deserialisiert. Die darin enthaltenen Daten sind die Anzahl der Zeilen, Spalten und Keypoints einer OpenCV internen Matrix, die im Folgenden CvMatrix genannt wird.

Die Daten des eigenen ImageMatrix-Objektes werden anschließend in eine CvMatrix übertragen. Danach wird eine Datenabfrage von allen bereitgestellten Keypoints, die aus den Referenzbildern ermittelt wurden, an die ElasticSearch Datenbank gesendet. Da die Keypoints der Referenzbilder ebenfalls als JSON-Objekt hinterlegt werden, müssen diese den Deserialisierungsschritt auch durchlaufen. Um eine Identifikation mit den Ressourcen herzustellen, wird weiterhin eine Produkt ID mit dem JSON-Objekt übergeben.

Beim Bildvergleich wird die CvMatrix mit allen CvMatrizen der Referenzbilder verglichen und die Ergebnisse gefiltert. Die Referenzbilder mit den meisten Übereinstimmungen werden einer Liste übergeben. Wie bereits im Kommunikationsmodell3.2 dargestellt, wird dann eine Response via http mit der Liste an potentiellen Ergebnissen an den Dienstnutzer gesendet. Anschließend wird eine GET-Methode vom Dienstanbieter aufgerufen, die eine Produkt ID enthält. Die Produkt ID dient als Identifikationsindikator und sorgt für eine korrekte Zuweisung einer REST Ressource.

3.2. Softwarekomponenten

Während der Entwicklung werden externe Softwarekomponenten verwendet, um bestimmte Aufgaben zu erfüllen. In den nächsten Abschnitten werden die wichtigsten Softwarekomponenten erläutert.

3.2.1. JDK und Android SDK

Um eine native mobile Applikation für das Android Betriebssystem zu entwickeln, benötigt man das Android Software Development Kit. Im Projektantrag des Forschungsprojektes zum Projekt OPDIS wurde beschlossen, eine mobile Applikation ab der Androidversion 4.0 - IceCreamSandwich zu entwickeln. Seit der Androidversion 4.0 basieren Android Smartphones und Android Tablets auf dem selben Betriebssystem. Davor wurden zwei unterschiedliche Androidversionen verwendet. Das Android SDK wird genutzt, um die Komponenten OPDIS APP und OPDIS FRAME zu entwickeln. Für die Dienstanbieterkomponente wird das Java Development Kit 1.6.4 genutzt und nicht die aktuelle Version 1.7, da Gradle bislang die aktuelle JDK Version nicht unterstützt.

3.2.2. Spring Framework

Das Spring Framework wird genutzt, um eine unkomplizierte Verbindung zwischen Dienstnutzer und Dienstanbieter herzustellen. Es bietet eine Menge von nützlichen Tools an und erleichtert dem Programmierer die Entwicklung, da er sich auf seine Kernkomponenten konzentrieren kann. Auf Dienstnutzerseite kommt das „Spring for Android“zum Einsatz. Das darin enthaltene Resttemplate ermöglicht eine schnelle und einfache Konfiguration der REST-Komponente. So sind die HTTP Requests in wenigen Zeilen realisierbar. Beim Aufruf des Konstruktors werden weitere unterstützende Objekte angelegt, die eine Vorkonfiguration für das Resttemplate aufbauen. Auch die Serialisierung der HTTP Requests und die Deserialisierung der HTTP Responses werden automatisiert vorgenommen. Um diese Vorgänge ohne großen Aufwand vorzunehmen, werden native Android HTTP client Bibliotheken vom Spring Framework verwendet.

Auf Dienstanbieterseite wird das Spring Security Framework benutzt. Die Hauptaufgabe liegt in der Authentifizierung der HTTP Anfragen und den Zugriff auf die Ressourcen. Ebenso wird dadurch die REST Schnittstelle durch weniger Konfigurationsaufwand

vereinfacht und ermöglicht die Konzentration während der Entwicklung auf die Kernkomponenten der Services.

3.2.3. OpenCV

OpenCV ist eine Bildbearbeitungsbibliothek, die auf vielen verschiedenen Plattformen angeboten wird. Mittlerweile gehört sie zu den bekanntesten und größten Bibliotheken, die bereits von vielen Programmen erfolgreich angewendet wird. Die Basisprogrammiersprache ist C. Da dieses Projekt allerdings mit der Programmiersprache Java geschrieben wird, kommen zwei unterschiedliche SDKs zum Einsatz. Zum einen JavaCV für den Dienstanbieter und zum anderen OpenCV for Android für den Dienstanbieter. Diese beiden unterschiedlichen SDKs werden im folgenden genauer erläutert.

JavaCV

Die Implementierung des ORB Proof-of-Concepts wird mit JavaCV umgesetzt. JavaCV ist ein Java Native Interface (JNI), dass auf C++ Bibliotheken von OpenCV zugreift. OpenCV ist wie ImageJ eine weit verbreitete Bildverarbeitungsbibliothek, die allerdings auf C++ basiert. Um OpenCV benutzen zu können, muss eine Installation des OpenCV Basispakets vorgenommen werden. Anschließend muss der Pfad der C++ Bibliotheken in die Systemumgebungsvariable PATH eingetragen werden. Es ist bislang die einzige Implementierungssmöglichkeit ORB über Java zu realisieren. JavaCV bzw. OpenCV ist vor allem im Bereich für die Softwareentwicklung zuständig. Es besitzt im Gegensatz zu ImageJ keine eigene Oberfläche. Funktionalitäten, die bei ImageJ über Plugins manuell hinzugefügt werden müssen, bietet JavaCV/OpenCV schon von Haus aus an. Dabei werden in der Bilderkennung viele verschiedene Detektoren und Deskriptoren angeboten. So wäre auch ohne großartigen Implementierungsaufwand eine Änderung des Bilderkennungsverfahren möglich.

OpenCV for Android

Um OpenCV auf mobilen Android Geräten nutzen zu können, benötigt man das OpenCV for Android Software Development Kit. Dieses lässt sich auf zwei verschiedene Arten einfügen. Die erste Methode ist die Installation eine externen Android Applikation. Der OpenCV Manager ist eine mobile Applikation, die das OpenCV for Android SDK beinhaltet. Die Entwickler empfehlen diese Methode, da diese Applikation immer auf dem aktuellsten Stand gehalten wird. Dahingegen ist es fraglich, ob diese Methode eine akzeptable Gebrauchstauglichkeit für Anwender aufweist. Die zweite Methode ist das manuelle Einfügen der SDK in die mobile Applikation. Der Vorteil dieser Methode ist, dass keine zusätzliche Installation einer externen Quelle erfolgen muss. Der Nachteil liegt in den unterschiedlichen Chipsätzen verschiedener Smartphones. Für jede Art erfolgt ein separates SDK. Um alle möglichen Modelle zu unterstützen, müsste man vier unterschiedliche Software Development Kits einbinden, die die Größe der mobilen Applikation wachsen

lassen. Zusätzlich zum OpenCV for Android SDK, muss ein OpenCV Library-Projekt referenziert werden, um die in Java geschriebenen angebotenen Objekte und Methoden nutzen zu können.

Im Gegensatz zu JavaCV basieren die Methoden auf andere Objekte, die zum einen eine reduzierte Funktionalität bieten und zum anderen anders benannt worden sind. Die reduzierte Funktionalität des OpenCV for Android SDK hatte zur Folge, dass eine automatische Generierung von XML Dokumenten entfällt. Zwecks dieser fehlenden Funktionalität musste die zuvor geplante Softwarearchitektur geändert werden.

Aus der geplanten Systemarchitektur war zu entnehmen, dass die internen OpenCV Keypoints direkt in das XML Format serialisiert werden. Fortführend sollte das XML Dokument in das JSON Format konvertiert werden. Der Dienstanbieter würde diesen Prozess rücklaufend durchführen, sodass aus dem JSON Format ein XML Dokument generiert wird.

Die OpenCV Komponente wäre dann in der Lage, das XML Dokument auszulesen und daraus eine CvMatrix abstrahieren. Diese fehlende Funktionalität wurde entgegengekommen, indem ein Hilfsobjekt generiert wurde. Dieses Hilfsobjekt enthält alle wichtigen Daten aus der OpenCV Matrix und kann direkt vom Java Objekt in das JSON Format serialisiert werden. Dieser Umweg stellte sich dann aber als bessere Lösung dar, da eine Serialierungs- bzw. Deserialisierungsstufe ausgelassen werden kann.

3.2.4. ElasticSearch

ElasticSearch ist eine Datenbank-Suchmaschine, die eine hohe Suchgeschwindigkeit aufweist, eine RESTful API beinhaltet und die Ressourcen auf mehreren Nodes verteilt. Bei der Such- und Analysefunktion wird intern Apache Lucene genutzt. Zudem ist ElasticSearch selbst organisierend und bietet mit dem Konzept der Nodes, Shards, Replicas und Indexes eine sehr hohe Skalierbarkeit und wächst mit dem Datenbestand mit. Dazu ist es lediglich nötig, weitere Nodes zu starten. Die Ressourcen werden eigenständig auf die Nodes und Shards verteilt. Zudem werden durch Replicas Redundanzen gebildet, um einen Datenverlust zu vermeiden. Die Suchmaschine basiert auf der Programmiersprache Java und verwaltet die Ressourcen im JSON Format. Durch die standardisierten REST Prädikate ist es möglich unkompliziert Daten anzulegen und abzufragen. [1]

[1] http://www.elasticsearch.com/

4. Entwurfs- und Implementierungsaspekte

In diesem Kapitel wird der konzeptuelle Teil verwendet, um die einzelnen Komponenten zu entwickeln. In diesem Kapitel werden die entwickelten Komponenten erläutert und ihre Zusammenhänge verdeutlicht.

4.1. Dienstnutzer

Der Dienstnutzer besteht aus einer Anwendungslogik und einer Präsentationslogik. Dabei wurde der Dienstnutzer in zwei Komponenten gegliedert und die Logiken von einander getrennt. OPDIS FRAME enthält die Anwendungslogik und wird für Anwendungsentwickler bereitgestellt. Diese sollen das Framework nutzen, um mit dem Dienstanbieter kommunizieren zu können. Die prototypische Komponente OPDIS APP beinhaltet die Präsentationslogik und demonstriert die Funktionalitäten des Frameworks. Dabei benutzt OPDIS APP das Framework, um die darin aufbereiteten Daten in der Präsentationslogik darzustellen.

4.1.1. Anwendungslogik

Durch die Nutzung vieler verschiedener Anwendungsentwickler sollte das Framework gut strukturiert, dokumentiert und verständlich sein. Ein entscheidender Punkt für die Gebrauchstauglichkeit ist auch die Namensgebung der Klassen und Methoden. Zum Verständnis der Entwickler werden sprechende Namen gewählt, die auch die Bedeutung ihrer Klasse oder Methode widerspiegelt. Aus diesem Grund wurde sich an die Java Code Konventionen gehalten. Im Kontext der Bilderkennung übernimmt das Framework vier Kernaufgaben. Es beinhaltet einen FileManager4.2, ImageManager4.3, KeyPointManager4.4 und einen NetworkManager4.5. Hinzu wird eine Objektklasse ImageMatrix4.1 gemeinsam vom Dienstnutzer und Dienstanbieter verwendet.

ImageMatrix

Die Klasse ImageMatrix 4.1 bildet ein Hilfsobjekt, um die resultierenden Daten der Bilderkennung an den Dienstanbieter zu senden. Diese Klasse wird um die abstrakte Klasse Entity erweitert und erhält damit noch zwei zusätzliche Variablen, die vorzugsweise von der Dienstanbieterseite genutzt werden.

Bei der Erstellung eines ImageMatrix-Objektes werden die Dimensionen der Reihen und Spalten in rows und cols eingetragen und die Keypointextraktionen werden als String in die Variable data gespeichert. Die einzelnen Reihen der Matrix symbolisieren

einen Vektor, so dass 500 Merkmalsextraktionen 500 Vektoren entsprechen. Jeder Vektor hat eine Länge von 32 Spalten und stellt einen Deskriptor dar. Die einzelnen Werte des Deskriptors beschreiben die Distanz zur umliegenden Nachbarschaft.

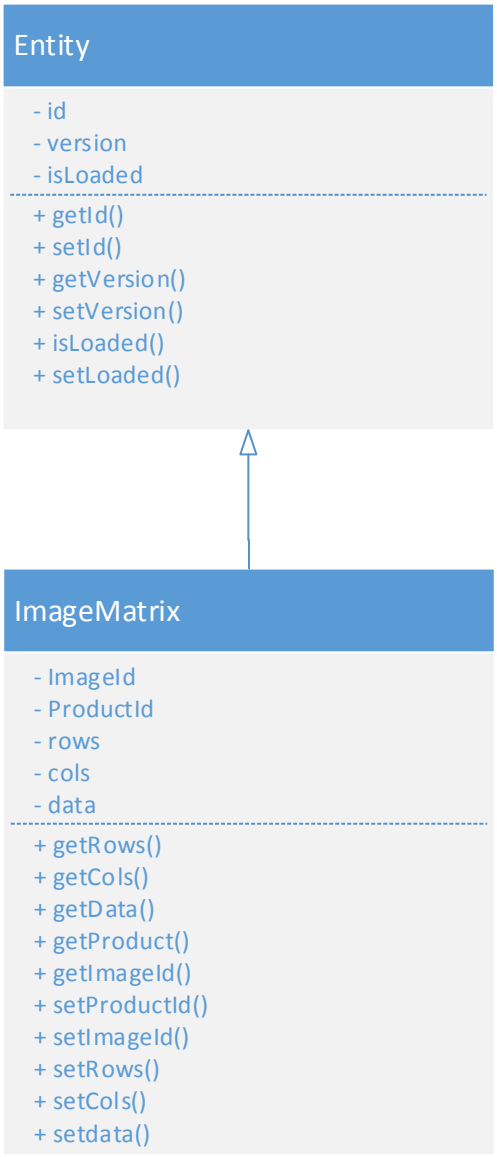

Abbildung 4.1.: ImageMatrix.class

FileManager

Der FileManager 4.2 stellt fest, welche Speicherkapazitäten auf dem Gerät genutzt werden können. Die Standardeinstellung besagt, dass zuerst geprüft wird, ob eine SD Karte zur Verfügung steht. Wenn eine SD Karte eingelegt ist, wird die Bilddatei darauf archiviert. Ist dies nicht der Fall, wird der interne Gerätespeicher verwendet um das Bild zu archivieren.

Bei der Nutzung einer androidinternen Kamera ist es notwendig eine leere Datei zu erstellen. Mit der setDataPath() Methode können drei Parameter übergeben werden. Als erstes der Pfad in dem die Bilder gespeichert werden sollen, als zweites der Dateiname und als letztes das Datenformat. Bisher werden die zwei Standardformate JPG und PNG unterstützt. Diese Variablen wurden als Konstanten in der Klasse fest definiert. Es muss sichergestellt werden, dass die Datenformate mit JavaCV kompatibel sind.

Weiterhin wurde eine Dateinamenserweiterung implementiert, dass eine fortlaufende Nummerierung angibt (zum Beispiel: $bild_01.jpg, bild_02.jpg, bild_03.jpg, etc.$). Die Nummerierung wurde gewählt, um dem Benutzer einen transparenten Überblick über die Aktivitäten zu gewährleisten. Für die weitere Entwicklung ist vorgesehen, dass eine Einstellungsoption zur Verfügung gestellt wird, um den Benutzer entscheiden zu lassen, ob die Bilddateien archiviert werden oder nach Abschluss aller Tätigkeiten wieder gelöscht wird.

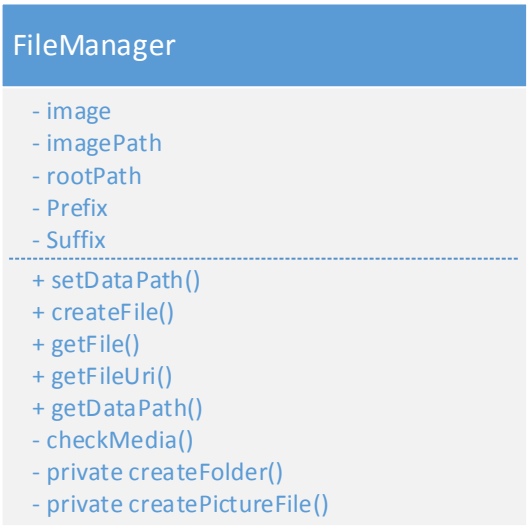

Abbildung 4.2.: FileManager.class

ImageManager

Das Bilderkennungsverfahren ORB ist zwar Skalierungsinvariant aber dennoch ließen sich qualitative Unterschiede in vorangegangenen Tests bei zu großen Auflösungsunterschieden erkennen. Daher besteht Optimierungsbedarf bei Bildern, die mit der Androidkamera aufgenommen wurden. Da die Referenzbilder eine Durchschnittsauflösung von circa 500x666 Pixel besitzen, sollten die Zielbilder angepasst werden. Die URI des Bildes muss als String an den Konstruktor übergeben werden, sodass der Konstruktor den Pfad zum Bild intern weiterverarbeiten kann. Das ist nötig, um ein Bitmap-Objekt zu erstellen, damit das Bild modifiziert werden kann. Mit Hilfe der InSampleSize-Variable2.2.2, die in der Zielbildspezifikation beschrieben ist, wird die Skalierung des Zielbildes an

die Skalierung der Referenzbilder angepasst, ohne Verzerrungen oder Ähnliches zu erzeugen. Als einzige öffentliche Methode der Klasse ImageManager 4.3, wird optimize() zur Verfügung gestellt. Bei der Entwicklung im Androidbereich sollte vermehrt auf die Speicherverwaltung Acht gegeben werden, da einer mobilen Applikation nur 15 MByte zur Verfügung stehen. Daher werden für die vorbereitenden Optimierungsmaßnahmen keine Bilddaten geladen, sondern nur Metadaten. Für die Optimierungszwecke werden dafür die Bildauflösung und der Orientierungseintrag in EXIF verwendet. Wurden diese Daten gesammelt, wird eine Rotationsmatrix erstellt. Durch den EXIF Orientierungseintrag, kann die Rotation ermittelt werden und durch die Methode exifToDegrees wird die Gradzahl der Rotation ermittelt. Anhand dieser Gradzahl kann die Rotationsmatrix erstellt werden. Sollte beim Erstellen der Rotationsmatrix ein Fehler auftreten, so muss die Rotationsinvarianz der Bilderkennungsmethode ORB ausreichen.

Anschließend wird die Skalierung des Zielbildes an die der Referenzbilder angepasst. Wie bereits in 2.2.2 erklärt wurde, wird mit der inSampleSize-Variable das Verhältnis zwischen Ist-Bildauflösung und Soll-Bildauflösung festgelegt. Nach Abschluss dieses Vorganges wird eine endgültige Bitmap erstellt, die mit der Rotationsmatrix gedreht und mit der InSampleSize Variable skaliert wurde.

Aus der optimierten Bitmap wird anschließend eine temporäre Datei erzeugt, in der das Bild geschrieben wird. Diese Datei wird nach Abschluss aller Bilderkennungshandlungen wieder gelöscht. Nach Speicherung der temporären Datei, werden alle Bitmaps die benötigt wurden recycled, um Speicherkapazitäten freizuräumen

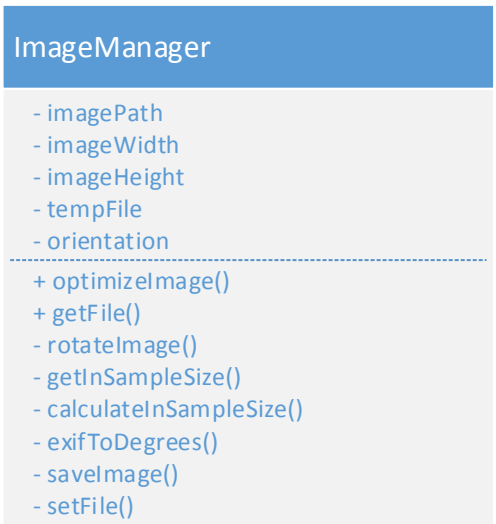

Abbildung 4.3.: ImageManager.class

KeyPointManager

Der KeyPointManager 4.4 ist für die komplette Bilderkennung zuständig. Der Benutzer kann ohne Hintergrundwissen eine Bilderkennung durchführen, indem er dem Konstruktor einen Bildpfad als String übergibt. Anschließend reicht es aus, mit dem Objekt zwei Methoden aufzurufen, um den Bilderkennungsprozess anzustoßen. Die einzigen beiden Methoden, die die Sichtbarkeit public erhalten haben, sind „generate()“ und „prepareMatching()“. Die Methode „generate()“ generiert eine Matrix mit KeyPoints, die durch den Detektor „oriented FAST“ gefunden und mit dem Deskriptor „rotated BRIEF“ näher beschrieben werden. Durch die Nutzung von privaten Methoden, kann nicht direkt auf die Methoden mit den Algorithmen zugegriffen werden. Der zweite Schritt für einen Bildvergleich benötigt eine Vorbereitung der erhaltenen Daten. Bevor die Methode „prepareMatching()“ aufgerufen wird, existiert nur ein OpenCV internes Objekt mit den benötigten Daten. Die Methode „prepareMatching()“ leitet die Daten vom internen OpenCV Objekt zum ImageMatrix-Objekt weiter. Dieser Schritt ist notwendig, um die Daten in ein JSON konformes Objekt zu konvertieren. Die einzig wichtigen Daten zur Weiterleitung, sind die die Anzahlen der Spalten und Reihen der Matrix und die einzelnen Werte der Matrixzellen. Die Besonderheit bei der Konvertierung in ein ImageMatrix-Java Objekt ist, dass die Keypointextraktionen von einem Double Array in einen String konvertiert werden müssen. Dies ist notwendig, um den Dienstanbieter die Verarbeitung, in der Zusammenarbeit mit ElasticSearch, zu ermöglichen.

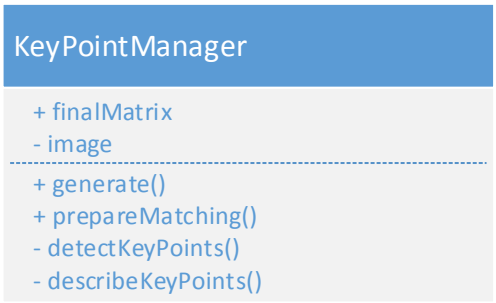

Abbildung 4.4.: KeyPointManager.class

NetworkManager

Der NetworkManager 4.5 ist für die Verbindung und die Kommunikation mit dem Dienstanbieter verantwortlich. Er optimiert den Zugriff auf bestehende Ressourcen, mit Hilfe des HttpResponseCaches. Dieser Vorgang spart Zeit und Bandbreite ein. Das Anlegen des Caches geschieht durch eine statische Initialisierung einer Instanz, mit der Methode getInstance(). Diese prüft ob bereits eine Instanz vorhanden ist. Wenn diese vorhanden ist, wird die aktuelle Instanz benutzt. Ansonsten wird eine neue Instanz erstellt und dabei der Cache angelegt. Bevor ein HTTP Request auf die REST Schnitt-

stelle gesendet werden kann, wird die Methode prepareRestTemplate() aufgerufen. Darin wird die Standardkonfiguration des RestTemplates vorgenommen. Zum einen wird die kritische Zeit des Verbindungsaufbaus auf 5 Sekunden gesetzt. Sollte eine Verbindung innerhalb von 5 Sekunden nicht aufgebaut werden können, so wird der Vorgang abgebrochen. Zum anderen wird dort die Vorbereitung für die Serialisierung und Deserialisierung mit dem MappingJackson2HttpMessageConverter vorgenommen.

Der MappingJackson2HttpMessageConverter bereitet den Aufbau des Datenpakets vor. So wird ein einfaches Java Klassen-Objekt in ein JSON serialisiert, um es in eine HTTP konforme Nachricht zu konvertieren. Für einen POST Request muss weiterhin der HTTP Header erweitert werden. Da auf dem Dienstanbieter *@consumesapplication/json* definiert wurde, muss im HTTP Header der Content Type angegeben werden.

Der Content Type gibt an, um welches Format der POST Request sich handelt. Der Content Type muss den Bestimmungen entsprechen die auf Dienstanbieterseite definiert wurden. Wurde nach dem HTTP Request eine Response entgegengenommen, kann die Instanz mit der Methode destroyInstance() zerstört werden.

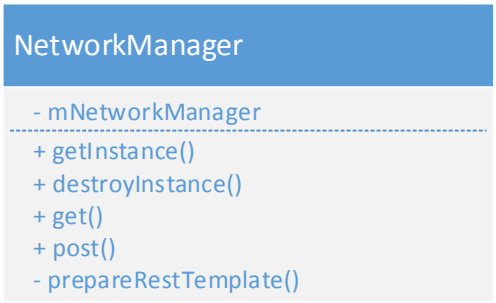

Abbildung 4.5.: NetworkManager.class

4.1.2. Präsentationslogik

Die Präsentationslogik wurde auf die OPDIS-APP Komponente verteilt. Direkt auf der Start-Activity befindet sich ein Button mit einer Kamera und der Unterschrift Bilderkennung. Dies ist der Einstiegspunkt in die Bilderkennungskomponente. Nachdem man diese Button aktiviert hat, wird man auf die nächste Aktivität geleitet, in der der Benutzer zwischen der Android-Galerie und der Android-Kamera wählen kann. Die Galerie wurde mit beachtet, damit Benutzer auf Grund von Konnektivitätsproblemen zu einem späteren Zeitpunkt eine Bilderkennung von Produkten durchführen können.

Nachdem ein Bild ausgewählt wurde oder auch durch die Kamera neu aufgenommen wurde, wird man auf die nächste Aktivität verwiesen, in der die Bilderkennung stattfindet. Während der Bilderkennung wird jederzeit Feedback gegeben, in welchem Status sich die Bilderkennung befindet. Die darauffolgende Aktivität ist vom Ergebnis des Bild-

vergleichs abhängig. Wurde genau ein passendes Produkt zugeordnet, so wird direkt auf die Produktaktivität geleitet, in der nähere Details ausgegeben werden.

Wurden mehrere Produkte dem Bild zugeordnet, so erhält der Benutzer eine Liste dieser Produkte, in der der Benutzer das passende Produkt auswählen muss. Sobald kein Produkt gefunden wurde oder ein Kommunikationsproblem mit dem Dienstanbieter aufgetreten ist, wird es dem Benutzer mitgeteilt.

ImageScanActivity

Die ImageScanActivity wird durch einen Intent aus der StartActivity aufgerufen. Durch diese Aktivität kann der Bilderkennungsprozess gestartet werden. Um die Bilderkennung zu starten, wird ein Bild benötigt, dass aus einer Galerie ausgewählt wird oder auch durch eine Kamera generiert wird. Sollte die Kamera ausgewählt werden, wird ein Prozess eingeleitet, in der eine leere Datei erstellt wird. Durch den Zugriff auf den File-Manager im Framework, wird eine leere Datei erstellt. Der Pfad der leeren Datei wird der Kamera übergeben, sodass das Ergebnis der Kamera in die Datei gespeichert wird. Beim Start der Kamera wird durch ein Intent auf die Kameraapplikation, die aktuelle mobile Applikation verlassen und kehrt zurück, sobald das Bild gespeichert wurde. Beim erneuten Aufruf wird die Methode onActivityResult aufgerufen, die dann zur nächsten OPDIS-Activity weiterleitet. Dabei wird der Pfad als String mit übergeben. Wird die Galerie ausgewählt, so wird keine leere Datei erzeugt, da die bereits bestehende genutzt werden kann. Kehrt man aus der Galerie zur OPDIS App zurück, so wird eine URI mitgeliefert. Diese URI muss zunächst in einen String konvertiert werden, um diese für die Bilderkennung nutzen zu können.

ImageRecognitionActivity

Wenn die ImageRecognitionActivty gestartet wird, werden verschiedene User Interface Elemente Initialisiert, um mehrere Szenarien abzudecken. Diese Elemente werden allerdings nur initilaisiert, wenn ein Intent an die Activity mit einem Pfad übergeben wurde. Als erstes wird eine TextView initialisiert, die noch nicht auf der Oberfläche abgebildet wird. Diese wird im späteren Verlauf für Benachrichtigungen an den Benutzer verwendet. Als weiteres User Interface Element wird eine ListView erstellt, die ebenfalls noch nicht auf der Oberfläche abgebildet wird, da diese noch leer ist. Falls mehr als ein Ergebnis bei der Bilderkennung gefunden wurde, werden mehrere Produkte der ListView hinzugefügt. Nach der Initialisierung der User Interface Komponenten, wird die Bilderkennung durch die Methode doImageRecognition() gestartet. In dieser Methode wird zuerst das Bild durch die Klasse ImageManager für die Bilderkennung optimiert und anschließend eine asynchrone Aufgabe eingeleitet, die parallel zum Aufbau des User Interfaces durchlaufen wird.

Der parallele Prozess entlastet den Hauptprozess, da dieser zu viele Prozesse abarbeiten muss. Der asynchrone Vorgang wird in einer inneren Klassen vorgenommen, damit globale Variablen aus der ImageRecogntionActivity mitgenutzt werden können. Beim

Aufruf des asynchronen Prozesses, wird der Dateipfad des temporären Bildes aus der ImageManager-klasse übergeben.

Die innere Klasse wird durch die abstrakte Klasse AsyncTask erweitert, so dass die drei Hauptmethoden benutzt werden können, um den asynchronen Prozess ordnungsgemäß ablaufen zu lassen. Die beiden Methoden onPreExecute() und onPostExecute() sind in der Lage, Manipulationen am User Interface durchzuführen. Die dritte Methode doInBackround() hat keinen Zugriff auf das User Interface und ist nur für die Verarbeitung für Daten gedacht.

- onPreExecute() Bevor die eigentliche Aufgabe gestartet wird, wird das User-Feedback für den Benutzer gestartet. Während der Aufgabenabwicklung soll der Benutzer genaustens über die Schritte informiert werden, die zur Zeit geschehen. Dazu wird eine ProgressBar verwendet, die den Fortschritt visualisiert. Dabei wird ein Ladekreis dargestellt und mit einer Statusnachricht erweitert. Zu beginn wird der Status auf „Bilderkennung wird gestartet...“gesetzt.

- doInBackground() Diese Methode verwendet den Dateipfad Parameter und startet damit den Bilderkennungsprozess. Dem KeyPointManager wird der Dateipfad übergeben, um im KeypointManager eine Matrix zu erstellen. Anschließend werden die öffentlichen Methoden des KeypointManagers ausgeführt, wobei die prepareMatching() Methode eine ImageMatrix erzeugt und diese im asynchronen Prozess erzeugt. Nachdem der Bilderkennungsprozess abgeschlossen ist, wird die ProgressBar mit Hilfe der onProgressUpdate()-Methode, die auch Zugriff auf das User Interface hat, aktualisiert. Demnach wird die aktuelle Benachrichtigung auf „Bildvergleich wird gestartet...“geändert. Anschließend wird eine Verbindung zum Dienstanbieter mit dem NetworkManager hergestellt, um einen POST-Request an die REST Schnittstelle zu senden. Der Aufbau des POST Requests lautet wie folgt:

```
1 List<LinkedHashMap<Integer, String>> response = nManager.
2                     post("/imagematrix/_search", im, ArrayList.class);
```

  Als Rückgabewert vom Dienstanbieter wird eine Liste erwartet. In dieser Liste wird für jeden Produktkandidaten eine LinkedHashMap mit den Produktdaten geliefert. Damit man ein Ergebnis erhält, sollte ein valider REST Pfad, das ImageMatrix Objekt und die erwartete Rückgabeklasse hinzugefügt werden. Die Response vom Dienstanbieter wird anschließend an die onPostExecute() Methode weitergeleitet

- onPostExecute()

  In der onPostExecute()-Methode wird die Response weiterverarbeitet, da noch keine Produktobjekte übermittelt wurden. Diese Methode ist für die Aktualisierung der User Interface Komponenten zuständig. Wenn Ergebnisse in der Response-Liste vorhanden sind, werden diese in einer for-Schleife durchlaufen und aus jedem Produkteintrag wird eine eindeutig identifizierbare Produkt ID gefiltert. Mit dieser Produkt ID wird später ein GET Request an den Dienstanbieter gesendet, um das

Produkt-Objekt zu erhalten. Sollte nur ein Produkt gefunden worden sein, wird direkt auf die Detailseite des Produkts verwiesen. Bei mehreren Ergebnissen wird die ListView aktualisiert und die Produkte eingetragen. Dabei wird ein Parcelable Objekt des Produktes erstellt und in die Liste eingesetzt. Die toString() Methode des Objekts wurde überschrieben, damit in der Liste der Produktname eingetragen wird. Bei keinem Ergebnis oder bei einem Fehler wird die bereits initialisierte TextView mit Fehlermeldungen aktualisiert.

Wurde die Liste gefüllt und visualisiert, wird bei einer Auswahl ein Adapter aktiviert, der den Vorgang der Touch-Geste verwaltet. So wird bei Auswahl eines Elementes, zur Laufzeit ein GET Request für dieses Produkt erstellt und als Response wird ein Produkt-Objekt empfangen.

4.2. Dienstanbieter

Der Dienstanbieter stellt eine REST Schnittstelle für die Dienstnutzer bereit, die genutzt werden kann, um Ressourcen aus dem Datenbestand zu empfangen und auch die Anwendungslogik für Prozesse verwenden können. Die Schnittstelle für die Bilderkennungskomponente ist der ImageMatrixController. Dieser stellt eine POST Methode zur Verfügung, die vom Dienstnutzer ein ImageMatrix-Objekt empfangen kann. Um nicht jede Anfrage bedingungslos zu bearbeiten, werden Bedingungen an die HTTP Requests gestellt. Es wird vom Dienstnutzer erwartet, dass im Header des HTTP Pakets ein Content Type existiert, der besagt, um was für ein Format es sich handelt. Diese POST Methode akzeptiert nur JSON Objekte und erwartet als Parameter ein ImageMatrix-Objekt. Weiterhin wird im Methodenkopf definiert, dass die Response vom Dienstanbieter in den HTTP Body-Bereich geschrieben wird und das empfangene ImageMatrix-Objekt sich im HTTP Body-Bereich befindet. Das ImageMatrix-Objekt wird anschließend weiter an den ImageMatrixService gesendet, in dem die Anwendungslogik stattfindet.

4.2.1. Anwendungslogik

Die Anwendungslogik befindet sich in der Klasse ImageMatrixService. Angestoßen wird sie vom Controller, der mit der Methode matchImageMatrix() nur eine ImageMatrix an den Service übergibt. Diese ImageMatrix wird im Service mit allen, in der Datenbasis befindlichen, Matrizen verglichen. Um dies zu ermöglichen wird aus der ElasticSearch Datenbank jeder vorhandene Datensatz, der eine ImageMatrix darstellt, in einer Liste als JSON Objekt gespeichert. Wurden die Referenzen alle organisiert, so beginnt die Verarbeitung der Ziel-ImageMatrix. Dabei wird mit JavaCV eine OpenCV interne Matrix erstellt. Dabei ist es notwendig, dass die Keypointextraktionen, die als String vorliegen, in ein Double Array konvertiert werden.

Nachdem das Zielbild für den Bildvergleich vorbereitet wurde, werden die Ergebnisse in einer TreeMap gespeichert. Die TreeMap hat im Gegensatz zu einer normalen Liste den Vorteil, dass die Ergebnisse mit „Collections.reverseOrder“ nach Häufigkeit der

Übereinstimmungen die Ergebnisse absteigend sortiert werden können. Nach der Initialisierung wird auf der TreeMap die Methode findMatches() durchgeführt.

Diese Methode erwartet als Parameter die Liste der Matrizen aus den Referenzbildern und einen Schwellwert zur Ermittlung qualitativer Ergebnisse. In dieser Methode wird der eigentliche Bildvergleich durchgeführt. Die Matrizen der Referenzbilder werden mit der Zielbildmatrix verglichen. Dabei wird bei jedem Vergleich die Anzahl der Übereinstimmungen protokolliert. Der Schwellwert ist eine Variable, die eine Mindestanzahl an Übereinstimmungen erwartet. Der Startwert des Schwellwertes wurde auf 18 festgelegt. Diese Anzahl liegt im oberen Durchschnitt eines erfolgreichen Bildvergleichs.

Wird die Anzahl der Übereinstimmungen jedoch nicht erreicht, wird dieses Produkt vorerst nicht weiter beachtet. Sollten allerdings keine Übereinstimmungen gefunden werden, wird der Schwellwert um zwei reduziert. Dieser Vorgang wiederholt sich solange, bis der Schwellwert auf sechs gesunken ist oder Produkte zugewiesen werden konnten. Da die Effektivität eine große Rolle spielt und dem Benutzer nicht nur willkürliche Ergebnisse angezeigt werden sollen, werden weniger als sechs Übereinstimmungen in einem Bildvergleich nicht beachtet.

Sollten mehr als 3 Ergebnisse gefunden werden, wird der Schwellwert um zwei erhöht. So wird sicher gegangen, dass eine qualifizierte Auswahl der Ergebnisse an den Benutzer gesendet wird.

Sollten die Produkte nun zugeordnet sein, so werden die Detailinformationen aus der ElasticSearch Datenbank ausgelesen und in einer LinkedHashMap gespeichert.

Der Bildvergleich benutzt die Merkmalsextraktionen, die eine Distanz zu der Nachbarschaftsumgebung bilden, um Matrizen mit einander zu vergleichen. Dabei wird eine Zieldistanz festgelegt, die nicht überschritten werden darf, um als Übereinstimmung gezählt zu werden. Je höher diese Zieldistanz angesetzt wird, desto mehr Übereinstimmungen werden gefunden aber desto ungenauer und Fehleranfälliger wird ein Bildvergleich. Durch Tests hat sich ergeben, dass eine Distanz von 55 sich bewährt hat. Dieser Distanzschwellwert wurde durch eine Annäherung ermittelt. Bei einer Distanz von weniger als 50 wurden zu wenig Übereinstimmungen gefunden, um ein aussagekräftiges Ergebnis zu erzielen. Bei einer Distanz von über 60 wurden zu viele Zufalls-Übereinstimmungen gefunden. Daher schien der Mittelwert von 50 und 60 als optimal und hat sich auch in den Tests bewährt.

4.2.2. Ressourcennmanagement

Das Ressourcennmanagement wird von der ElasticSearch Datenbank vorgenommen. Die wichtigsten Daten für den Bilderkennungsprozess sind die Produktdaten, Bilder und Bildmatrizen. Die ElasticSearch Datenbank ist RESTful aufgebaut und ermöglicht durch eine REST API den Zugriff auf die Daten mittels der HTTP Request-Methoden GET, PUT, POST und DELETE. Da die Datenbank dokumentenorientert arbeitet, werden alle Daten als JSON Datei gespeichert. Die ElasticSearch Datenbank indexiert die JSON Dokumente automatisch und stellt damit die Suchbarkeit her. Um ein konformes REST

zu gewährleisten, sollte ein Zugriff auf eine Ressource erfolgen und die Abbildung einer Ressource an den Dienstnutzer gesendet werden. Der Pfad zu einer Ressource wird über eine class, type und id Variable hergestellt. Um weitere Detailinformationen zu erhalten, werden weitere Referenz Ids verwendet. Zusätzlich zur Pfadangabe müssen die JavaKlassen definiert werden, um die Datenbankdaten in Objekte schreiben zu können. Dazu wird der Eintrag „Klasseneintrag“ verwendet und gibt dabei die Paketinformationen für den Standort der Klasse bereit. Ein ausführliches Beispielprodukt befindet sich im Anhang B.1.

Um an die Produktdaten zu gelangen muss der Bildvergleich durchgeführt werden. Dazu werden alle vorhandenen Matrizen ausgelesen und zur Laufzeit in Java Objekte umgewandelt. Wie im nächsten Codeausschnitt zu erkennen ist, wird das Bild nicht direkt übertragen sondern in der Base64 Kodierung. Durch diese Kodierung muss keine Referenz auf eine Bilddatei erstellt werden, sondern kann direkt im JSON Format übertragen werden.

```
1 {"index":{"_index":"opdis","_type":"Image",
2 "_id":"jdapyw"}}{"_class": "de.opdis.core.Image",
3 "name":"nivea_creme.JPG",
4 "position": 0, "base64": "/9j/4TX+RXhp...ZgAASUkqAA"}
```

Im weiteren Codeabschnitt wird dargestellt, wie eine ImageMatrix in der ElasticSearch Datenbank aufgebaut ist und auf die verwendeten Ressourcen verweist. Der ImageId ist zu entnehmen, zu welchem Bild die Merkmalsextraktion gemacht wurde. Die ProductId ist der Verweis zu dem zugehörigen Produkt.

```
1 {"index":{"_index":"opdis","_type":"ImageMatrix",
2 "_id":"5cf38fc6-b1ec-47ab-810f-4ef0c22c9f33"}
3 {"_class": "de.opdis.core.ImageMatrix",
4 "cols":32,"rows":500,"data":" [17.0,133.0,110.0,37.0,...] ",
5 "productId":"522268dc-b61c-4aec-b2d9-d7b9deb90340","imageId":"jdapyw"}
```

4.3. Error-Handling

Das Error-Handling ist in der Entwicklung ein enormer Faktor zur Produktstabilität. Mögliche Fehler sollten zu jederzeit abgefangen werden um einen Systemabbruch zu verhindern. Vor allem kritische Vorgänge sollten abgesichert werden. Im folgenden wurden die kritischen Fehler aufgelistet und dessen Lösungen und Workarounds dargestellt.

4.3.1. Dienstanbieter

Beim Testen der Bildvergleichskomponente auf dem Dienstanbieter war kein Debugging möglich. Daher mussten die HTTP Statuscodes, bei der Verbindung mit dem Dienstnutzer, mit dem Google Webtool Kit gefiltert werden und analytisch an die Fehlerbehebung ran gegangen werden. Der Umgang mit Statuscodes ist daher praktisch, da diese genau

angeben, auf welcher Komponente der Fehler auftritt und welche Stelle den Fehler erzeugt. [FGM+99]. Durch dieses Verfahren wurden offensichtliche Fehler gefunden und Lösungen ermittelt.

Auf dem Dienstanbieter wurde der HTTP Statuscode 500 erzeugt. Dies ist ein Sammel-Statuscode für unerwartete Serverfehler. Detailierte Informationen deckten auf, das die ImageMatrix, die vom Dienstnutzer an den Dienstanbieter gesendet wurde, nicht richtig verarbeitet wurde. Aus dem ImageMatrix Objekt konnte daher kein internes OpenCV Matrix Objekt erstellt werden. Dieser Fehler tritt nur auf, wenn ein Bild in Landscape-Sicht aufgenommen wurde. Dabei ist das Bild breiter als hoch. Die ImageMatrix die vom Dienstnutzer gesendet wurde, war mit validen Daten gefüllt und gaben keinen Anlass zur Fehlerursache. Um diesen Fehler zu umgehen wurde eine RuntimeException implementiert, die einen Rückgabewert von „null“liefert. Erkennt der Dienstnutzer den gesendeten Rückgabewert, erscheint eine Fehlermeldung auf dem Display des Nutzers.

4.3.2. Dienstnutzer

Der Dienstnutzer hatte einige mehr Fehler aufzuweisen als der Dienstanbieter. Der HTTP Statuscodebereich 400 deutet auf Fehler des Dienstnutzers hin und für jeden Statuscode werden Erklärungen im RFC 2616 [FGM+99] geliefert.

HTTP Statuscode 403

Die Beschreibung des HTTP Statuscodes besagt, dass die angeforderte Ressource nicht gefunden wurde. Da diese Fehlermeldung im 400er Bereich des Statuscodes liegt, liegt es nahe, dass der Fehler auf Dienstnutzerseite zu suchen ist. Dennoch ist es auch möglich, dass der Ressourcenzugriff auf Dienstanbieterseite keinen korrekten Ressourcenaufruf startet. Bei genauerer Betrachtung stellte sich heraus, dass im Bereich der prototypischen Realisierung von OPDIS APP, auf Androidseite nur Produkte eingepflegt sind, die aus dem Kosmetikbereich kommen. In der Datenbank bestehen allerdings auch Daten aus dem Lebensmittelbereich. Somit wurden beim POST-Request des Dienstnutzers, Textinformationen eines Lebensmittelprodukts geliefert und intern als Kosmetikprodukt behandelt. Beim nächsten GET-Request konnte anschließend kein passendes Produkt zugeordnet werden, da aus der Kosmetikklasse kein Produkt existiert, das eine Lebensmittelprodukt Id aufweist.

HTTP Statuscode 415

Als die Kommunikation zwischen Dienstnutzer und Dienstanbieter hergestellt wurde, wurden Testdaten an den Dienstanbieter gesendet. Daraufhin kam der HTTP Statuscode 415 als Response zurück. Dieser besagt, das ein anderer Medientyp erwartet wurde. Der Fehler lag im Umgang mit dem Spring for Android Framework. Der Grund für diese Fehlermeldung lag darin, das der Content Type „json/application“ im Header des HTTP Datenpakets nicht übermittelt wurde. Dieser wurde zuerst absichtlich nicht

übermittelt, da die Dokumentation des Spring for Android Frameworks besagt, dass der Content Type standardmäßig auf JSON eingestellt ist. Durch das manuelle Eintragen des Content-Types wurde dieser Fehler behoben.

```
1 httpHeaders.setContentType(MediaType.APPLICATION_JSON_VALUE);
```

OutOfMemoryException

Eine Schwachstelle der Smartphones ist der Arbeitsspeicher. Das Android Betriebssystem stellt pro laufender Anwendung einen begrenzenden Speicherplatz zur Verfügung. Berechnet wird dies durch den gesamten Arbeitsspeicher des Smartphones. Der Minimalwert liegt bei 16MB pro mobile Applikation. Da die Bildbearbeitung arbeitsspeicherintensiv ist, muss darauf geachtet werden, dass nicht benötigte Ressourcen freigegeben werden und den Speicher entlasten. Bei der Optimierung des Zielbildes trat dieser Fehler wiederholt auf. Beim rotieren des Bildes wurden zu Anfang die Bildinformationen mitgeladen. Dies verursachte ein zu großes Speichervolumen und das Betriebssystem beendete die mobile Applikation automatisch. Die Lösung für dieses Problem war die Flag-Setzung von „inJustDecodeBounds“. Dabei werden nur die Metainformationen eines Bildes geladen und die Bilddaten ignoriert. Nachdem diese Flag gesetzt wurde, war der Fehler behoben.

Activity Lifecycle

Der Activity Lifecycle gehört zu den Grundlagen der Androidprogrammierung. Dieser muss verstanden werden um vorausschauend zu programmieren. Bei vielen Handlungen die der Nutzer vornehmen kann, wird der Activity Lifecycle durchlaufen. Vor allem bei Rotationen des Smartphones wird zwischen zwei Sichten gewechselt. Die Landscape-Sicht (horizontal) und die Portait-Sicht (vertikal). Bei jeder Rotation durchläuft eine Aktivität den Activity Lifecycle und wird neu aufgebaut. Bei diesem Vorgang besteht die Möglichkeit das Daten verloren gehen und bei einer neuen Orientierung nicht mehr genutzt werden können.

5. Ergebnisse und Diskussion

Abschließend werden die erzielten Ergebnisse resümiert und kritisch diskutiert. Während und nach der Implementierung wurden einige Tests durchgeführt, die Aufschluss über Parameter und Schwellwerte gaben und es ermöglichten Schwellwerte auf ein optimales Maß zu bestimmen. Durch eine Menge von verschiedenen Tests und einer Dokumentation sind die Endergebnisse entstanden.

5.1. Testumgebung

Die Testumgebung bestand aus einem Dienstanbieter mit einem Intel i5 1,7GHz Prozessor und einem Windows 8 (64 Bit) Betriebssystem. Dazu war das Ultrabook mit einem Arbeitsspeicher von 4 GB ausgestattet. Die Dienstnutzerhardware bestand aus einem Samsung Galaxy S3 Smartphone mit einem 8 Mega Pixel Sensor in der Kamera. Die Standardauflösung der Kamera liegt bei 3264x2448 Pixeln und wurde durch die Optimierung für den Bildvergleich auf 500x666 reduziert 2.1.

In der Datenbank befanden sich zum Testzeitpunkt 28 Produkteinträge. Für die jeweiligen Testfälle wurde ein Produkt ausgewählt und mit den restlichen Referenzbildern verglichen. Das Verhalten bei mehreren Tausend Produkteinträgen kann variieren und demnach sollten weitere Anpassungen am ORB-Bilderkennungsprozess vorgenommen werden.

5.2. Testdurchlauf und Erkenntnisse

Um ein angemessenes Maß an relevanten Tests zu finden, wurden die User Stories A betrachtet. Dabei konnten die Anwendungsszenarien extrahiert und entsprechende Tests veranlasst werden. Zu Beginn wurden die Tests einzeln vollzogen, um das Verhalten des ORB-Bilderkennungsverfahrens explizit zu bestimmen. Anschließend wurden kombinierte Tests vorgenommen, da diese in der Realität am häufigsten in Erscheinung treten. In Betracht der kombinierten Tests wird die Robustheit im praktischen Anwendungsfeld bestimmt.

5.2.1. Systemtests

Um die Tests durchzuführen, die den User Stories A entsrechen, mussten Einstellungen in der Bildvergleichskomponente vorgenommen werden. Dabei werden Probleme erläutert, die den Bildvergleich erschwert haben und welche Maßnahmen getroffen werden müssen, um Übereinstimmungskandidaten besser differenzieren zu können.

Bildskalierung

Eine angemessene Bildskalierung ist ein wichtiger Bestandteil erfolgreicher Bildvergleiche. Je höher die Auflösung eines Bildes ist, desto mehr Merkmalsextraktionen können gefiltert werden. Tests haben ergeben, dass bei einer Auflösung von 3264x2448 Pixeln eine feste Anzahl von 500 Merkmalsextraktionen zu wenig sind. Gerade im Bezug auf den Hintergrund, der potentiell viele Merkmalskandidaten bereithält, sollte eine höhere Anzahl gewählt werden.

Da dieser Projektkontext auf mobile Geräte spezialisiert ist und eine höhere Anzahl an Merkmalsextraktionen auch mehr Leistung der Geräte verlangt, wurde entschieden eine geringere Bildauflösungsbasis zu finden. Vorangegangene Tests aus [Pap] haben ergeben, dass im Bereich des mobilen Bildvergleichs eine gute Performanz bei einer Auflösung von 500x666 Pixeln erreicht wird. Bei dieser Auflösung sind 500 Merkmalsextraktionen ausreichend, um Bildvergleiche erfolgreich durchführen zu können.

Bildvergleich

Der Bildvergleich ist von zwei Variablen abhängig. Zum einen die Distanzmessung der jeweiligen Deskriptorvergleiche und zum anderen einen Schwellwert zur Erkennung eines positiven Bildvergleichs. Die Bilder werden nach 2.1.3 verglichen. Dabei werden beim Bildvergleich die Distanzwerte der Deskriptoren zweier Bilder ausgewertet und deren Distanzdifferenz mit einem festgelegten Wert verglichen. Bei Betrachtung der unterschiedlichen Ergebnisse, stellte sich heraus, dass die Deskriptoren bei einer Durchschnittsdistanz zur Nachbarschaftsumgebung von 55 ein gutes Maß ergeben hat. Um so höher die Durschnitsdistanz gesetzt ist, desto mehr Übereinstimmungen werden durch den ORB Detektor gefunden.

Ein Bild zählt als Übereinstimmungskandidat, wenn mindestens eine Übereinstimmung gefunden wurde. Da der Aussagewert bei einer geringen Anzahl von Übereinstimmungen allerdings nicht viel aussagt, wird ein Schwellwert festgelegt. Dieser Schwellwert wird auf 18 Übereinstimmungen voreingestellt und verhält sich im Verlauf des Bildvergleichsprozesses variabel 4.2.1. Der Schwellwert 18 ist aus mehreren Tests hervorgegangen, die ein positiven Bildvergleich hervorgebracht haben. Dieser Durchschnittswert wurde genommen, um so wenig Iterationen wie möglich durchlaufen zu lassen. Der Schwellwert ist vom Distanzwert abhängig.

Um so höher die Toleranz des Distanzwertes steigt, desto mehr Ergebnisse werden beim gesetzten Schwellwert gefunden. Darunter können sich auch falsche Übereinstimmungen befinden, die durch einen höheren Übereinstimmungsschwellwert ausgeglichen werden müssen.

5.2.2. Einzeltests

Die Einzeltests wurden aus den User Stories A gefiltert und ergaben, dass die wichtigsten Kriterien in Tests der Distanzmessung, Bildqualität, perspektivische Unterschiede, Hintergrundrobustheit und den Lichtverhältnisse liegen.

Distanzmessung

Bei der Distanzmessung wurde vorerst versucht, das Zielbild so zu gestalten, dass die Produktrepräsentation den größten Teil des Bildes einnimmt. Dabei wurde die Hintergrundfläche gering gehalten, um den Anforderungen der Referenzbilder (Vgl.: 2.2) nahe zu kommen. Bei einem Abstand von 10-30cm wurden die Bilder nahezu immer erkannt. Bei 20 Testversuchen eines Bildvergleiches, bei dem der Abstand zwischen Kamera und Produkt bei ca. 30cm lag, wurden 2 Fehlversuche registriert. Zwei Fehlversuche entsprechen in dem Fall eine Erfolgsquote von 90%. Dabei wurde darauf Wert gelegt, dass bei jeder weiteren Entfernung (vgl. 5.1) auch der Hintergrund mehr Details beinhaltet, um die Merkmalsextraktionen zu streuen. Die Erfolgschance wird dadurch zwar verringert, dennoch reichen auch wenige Merkmalspunkte auf dem Produkt aus, um einen Bildvergleich erfolgreich abzuschließen.

Weiterhin wurde eine Stichprobentestreihe durchgeführt, bei der pro Entfernung drei Bildvergleiche initialisiert wurden. In der Tabelle 5.1 ist zu sehen, dass auch die Stichproben, die Erkenntnisse aus den Langzeittests widerspiegeln. Ab einer Entfernung von ca. 40cm kann kein zuverlässiger Bildvergleich durchgeführt werden. (siehe dazu auch Abbildung 5.1

Distanz	Versuch 1	Versuch 2	Versuch 3
10 cm	ok	ok	ok
15 cm	ok	ok	ok
20 cm	ok	ok	ok
25 cm	ok	ok	ok
30 cm	ok	ok	ok
35 cm	ok	ok	ok
40 cm	ok	-	-
45 cm	-	-	ok

Tabelle 5.1.: Distanztests

Bildqualität

Das ORB-Verfahren nutzt einen Detektor, der nach markanten Eckpunkten Ausschau hält. Dazu wird ein geglättetes Integralbild verwendet. Sollte die Bildqualität bei der Erzeugung schon mangelhaft sein, so wird die Eckpunktdetektion erschwert. Aus dem wissenschaftlichen Bericht [ICC11] ist zu entnehmen, dass das Verfahren robust gegenüber verwischten Bildern ist. Allerdings wurden keine Aussagen über den Grad der Verwischung gemacht. Nach eigenen Tests wurde herausgefunden, dass verwischte Bilder

Abbildung 5.1.: Testreihe zu Tabelle 5.1

nur in den seltensten Fällen positiv verglichen wurden. Daher sollte man bei der Generierung eines Bildes darauf achten , dass der Fokus auf das Produkt gelegt wird und dabei ein guter Schärfegrad des Produktes erzeugt wird.

Rotation und Perspektive

Es wurden sechs verschiedene Ansichtswinkel eines Bildes aufgenommen, um diese mit einem Referenzbild abzugleichen. Es wurden gleichmäßige Abstände der Ansichtswinkel vorgenommen, um auch eventuelle Tendenzen zu erkennen. Die Versuchsreihe befindet sich in Abbildung 5.2. Die ersten drei positiven Übereinstimmungen hatten allesamt verschiedene Rotationen und auch Perspektiven. Beim vierten Testbild wurde ein extremer Winkel gewählt, indem viele markante Stellen weggefallen sind. Dieser Test war negativ, da keine Übereinstimmungen gefunden wurden.

Abbildung 5.2.: verschiedene Rotationen und Perspektiven

Das Testergebnis aus Tabelle 5.2.2 spiegelt wieder, dass eine frontale Ansicht mit leichten Drehungen ein gutes Ergebnis liefern. Ab einer Gradwanderung auf 120° wird die Erkennungsrate rapide schlechter. Das ist wahrscheinlich darauf zurückzuführen, dass viele markante Stellen (z.B.: Logo oder Schriftzug) durch die perspektivische Verzerrung wegfallen.

Perspektivische Drehung	ORB Übereinstimmungen
90°	17
100°	23
120°	5
140°	6
160°	8
180°	2

Tabelle 5.2.: Ergebnissewerte des Perspektivtests

Hintergrundrobustheit

Wie schon in 2.2 erwähnt, wird die Qualität der Merkmalsextraktionen verbessert, in dem ein neutraler Hintergrund verwendet wird. In der Abbildung 5.3 ist zu erkennen, dass bei einer größeren Distanz zwischen Kamera und Produkt der Hintergrund wächst und somit auch weitere Merkmalskandidaten entstehen. Dieses Extrembeispiel wird im projektbezogenen Kontext häufiger auftauchen, da im Einzelhandel üblicherweise weitere Produkte im Hintergrund zu sehen sind (Vgl. dazu 5.4. Durch eine angemessene Distanz sollte dieses Problem allerdings behoben werden. Im Bezug auf Hintergrund und Distanz

wird im normalen Anwendungsfall eine Distanz von einer halben Armlänge erwartet. Im Normalfall sollte diese Distanz ausreichen, um einen positiven Bildvergleich zu erreichen.

Abbildung 5.3.: Ansicht der Merkmalsextraktionen aus verschiedenen Distanzen

Abbildung 5.4.: Anwendungsszenario im Einzelhandel

Luminanzinvarianz

Das Ergebnis aus Tabelle 5.3 des Luminanztests zeigt, dass sich das optimale Luminanzverhältnis aus natürlichem Sonnenlicht im Schatten zusammensetzt. Das Ergebnis dieses Tests deutet an, dass die beste Farbtemperatur bei ca. 10.000 Kelvin liegt. Außergewöhnlich ist auch die geringe Anzahl von Übereinstimmungspunkten bei direktem Sonnenlicht mit ca. 5600 Kelvin. Im Allgemeinen wird ein konstanter Wert erzielt, in dem das Zielbild frontal fotografiert wird. Die frontale Perspektive sollte die Kameraausrichtung einer Zielperson am nächsten kommen. Das primäre Zielumfeld wird allerdings eine

Farbtemperatur von ca. 3000-3200 Kelvin betragen. Doch auch in diesem Spektrum werden akzeptable Werte ermittelt und sollten kein großes Problem für einen Praxiseinsatz darstellen.

	Frontal	Links	Rechts	Oben	Unten
10000 Kelvin	14	13	15	9	20
5600 Kelvin	11	2	1	8	10
3200 Kelvin	11	11	9	11	14
3000 Kelvin	9	10	10	11	10

Tabelle 5.3.: ORB Ergebniswerte des Luminanzinvarianztests

5.2.3. Praxistest

Um praxistaugliche Bedingungen herzustellen, wurde ein Test in einem Einzelhandelsgeschäft vorgenommen. Dabei wurden die Erkenntnisse aus den Einzeltests berücksichtigt und gezielt eingesetzt. Es wurden Szenarien ausgewählt, in denen viele Merkmalskandidaten im Hintergrund gesetzt sind. Aus den Abbildungen 5.5, 5.6 und 5.7 lässt sich ersehen, dass trotz außerhalb liegende Merkmalsextraktionen ein positiver Bildvergleich durchgeführt wird. Weiterhin wurden erfolgreiche Bildvergleiche bei Produkten registriert, die aus dem Regal fotografiert wurden. Die Erkenntnisse aus den Anwendungsszenarien (siehe 5.6 und 5.7), die aus den User Stories A.5 und A.6 hergeleitet wurden, bekräftigte die Erweiterung der Stakeholder für Personen mit Beeinträchtigungen (z.B.: Rollstuhlfahrer). Bei den Tests wurden die künstlichen Lichtbedingungen und auch verschiedene Perspektiven gewählt, um die Aussagekraft dieses Tests zu überprüfen.

Ein Konnektivitätstest konnte nicht getestet werden, da zum Entwicklungszeitpunkt noch kein zentraler Dienstanbieter zur Verfügung stand. Der Dienstanbieter wurde bis dahin stets lokal ausgeführt. Daher wurden die Bilder aufgenommen und im internen Speicher des Smartphones archiviert. Als der Dienstanbieter lokal zur Verfügung stand, wurden die Bildvergleiche durchgeführt.

5.3. Evaluation

Die durchgeführten Tests haben sowohl die Stärken als auch die Schwächen der ORB Bilderkennungs- und Bildvergleichskomponente aufgezeigt. Bei einer Datenbankgröße von 28 Produkten, konnte die Bilderkennung in weiten Teilen erfolgreich durchlaufen.

Die Stärken lagen vor allem in der Performanz für mobile Geräte, perspektivische Veränderungen und in der Robustheit der Luminanz- und Rotationsinvarianz. Ein kombinierter Test aller Faktoren wurde in der Praxis umgesetzt und konnte als praxistauglich eingestuft werden. Vor allem die Toleranz zwischen den perspektivischen Unterschieden des Referenzbildes und des Zielbildes hat die Erwartungen übertroffen. Weiterhin ist auch die Robustheit gegenüber Merkmalen im Hintergrund zu erwähnen, da diese im

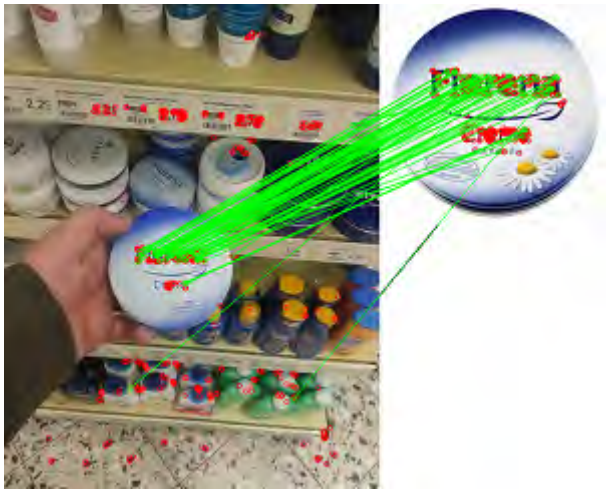

Abbildung 5.5.: Vergleich eines Produktes im Einzelhandel

Abbildung 5.6.: Vergleich eines Produktes direkt aus dem Regal

praktischen Gebrauch häufig auftreten und dennoch bei wenigen Merkmalsextraktionen ein qualitativer Bildvergleich stattfindet.

Des Weiteren ist aufgefallen, dass die Bilderkennung besonders gut auf Schriftzüge reagiert. Wie in Abbildung 5.8 zu sehen ist, werden sehr viele Merkmalsextraktionen aus den Buchstaben extrahiert. In erster Linie wird diese Eigenschaft als Schwäche angesehen. Bei schriftzuglastigen Repräsentationen besteht die Gefahr, dass ein positiver Bildvergleich stattfindet, obwohl sich nur die Schriftart ähnelt. Das es dennoch grundlegende verschiedene Produkte sind, registriert die Bilderkennung nicht. Weiterhin wird die bedingte Skalierungsinvarianz als Schwäche angesehen, da die Zielbilder an die Referenzbilder angepasst werden müssen. Um keine zu große Auslastung auf mobilen Geräten zu erzeugen, ist es empfehlenswert, geringere Auflösungen zu verwenden. Dadurch entsteht ebenso der positive Nebeneffekt, dass auf dem Dienstanbieter Datenvolumen eingespart und eine schnellere Übertragung vom Dienstnutzer zum Dienstanbieter erreicht wird.

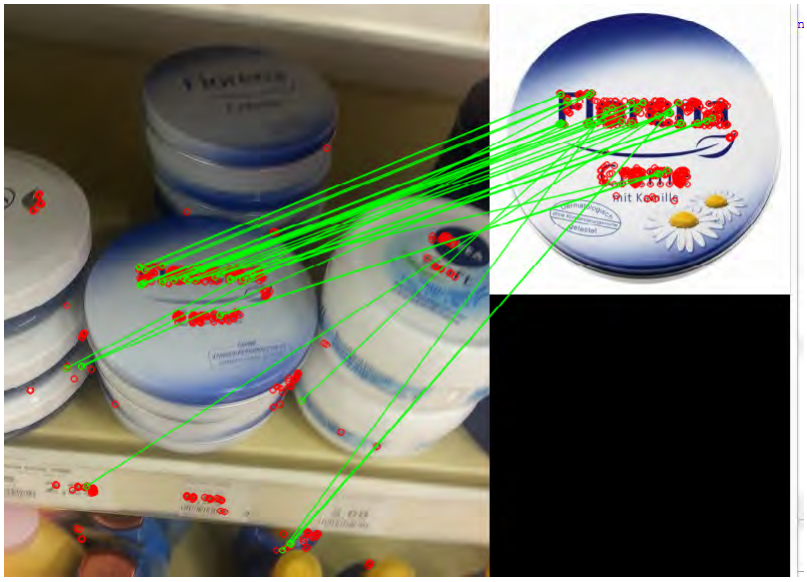

Abbildung 5.7.: Vergleich eines Produktes direkt aus dem Regal

Abbildung 5.8.: Merkmalsextraktion im Schriftzug

5.4. Zukunftsausblick

Zur Zeit wird die Standardkamera von Android genutzt, um eine Bildrepräsentation zu erstellen. Unterstützend wird eine Bildoptimierung bereitgestellt, die den Bilderkennungsprozess in Bezug auf den Bildvergleich verbessert. Für Entwickler ist es erstrebenswert die Kamera API von Android zu verwenden, um eine eigene Kamera zu entwickeln. Die Einstellungsmöglichkeiten können auf individuelle Bedürfnisse angepasst werden und das Design kann aus dem Applikationskontext übernommen werden. Durch die Vielfalt der Kamera API-Optionen, kann somit die Bildoptimierung entfallen. Die Voraussetzung für die Entwicklung einer maßgeschneiderten Kamera ist ein detailliertes Verständnis der Androidprogrammierung.

Weiterhin ist die Entwicklung der Bildvergleichskomponente auf dem Dienstanbieter nur für wenige Zugriffe gleichzeitig aufgebaut. Optimalerweise sollten dahingegen parallele und asynchrone Prozesse angewendet werden, um eine Vielzahl von Benutzern parallel Ergebnisse liefern zu können.

Für die Bildverarbeitung wird OpenCV verwendet. OpenCV liefert standardmäßig eine externe mobile Applikation, die als Bibliothek genutzt wird. Um die Bildverarbeitungsprozesse nutzen zu können, greift OpenCV auf diese App zurück. Fehlt diese App, so ist keine Bilderkennung auf dem Dienstnutzer möglich. Weiterhin ist der Aspekt einer zusätzlichen mobilen Applikation eine starke Einschränkung der Gebrauchstauglichkeit. Allerdings wird eine statische Initialisierung der Pakete ebenfalls angeboten. Diese ist

allerdings im Zusammenhang mit Gradle komplizierter einzupflegen. Damit würde keine zusätzliche mobile Applikation benötigt werden.

Weiterhin ist es im Android-bereich noch nicht möglich, Feineinstellungen der ORB-Komponente vorzunehmen. Sollten diese Funktionalitäten hinzugefügt werden, müssten hinsichtlich des Laufzeitverhaltens weitere Einstellungen getestet werden. Sollten in Zukunft bessere Bilderkennungsverfahren in der OpenCV Umgebung angeboten werden, ist es eine Leichtigkeit, diese auszutauschen. Die Komponenten wurde alle modular entwickelt und somit auch für externe Entwickler leicht zu deuten.

Im Bereich der Usability sollte ein weiterer Dialogprozess getestet werden, um die User Experience weiterhin zu steigern. Zur Zeit wird die Kamera zur Aufnahme von Bildern verwendet, die nach Aufnahme eines Bildes einen Bildvergleich durchführt. Sollte keine Übereinstimmung erfolgen, müsste der Benutzer ein weiteres mal ein Bild aufnehmen und den Dialog erneut anstoßen.

Um den Dialogverlauf für den Benutzer zu vereinfachen, besteht die Möglichkeit eine Videoaufnahme zu entwickeln, die alle 1-2 Sekunden Bilder aufnimmt und diese vergleicht, bis der Benutzer ein positiven Bildvergleich erhält. Dieser Dialogverlauf wird von QR-Code Scannern verwendet. Allerdings sollte darauf geachtet werden, dass nicht zu viele Daten über das mobile Netz gesendet werden.

Im fortlaufenden Prozess des OPDIS-Projektes soll es ermöglicht werden, eine Bilderkennung selbstständig auf dem Dienstanbieter durchführen zu können. Sobald ein neues Produkt eingepflegt wird und dieses Bilder enthält, kann die Bilderkennung einen JSON Eintrag in der Datenbank vornehmen und die Merkmalsextraktionen eintragen. Die Beziehungen zwischen Bilder und Matrizen sollen weiterhin so geregelt sein, dass bei Entfernung eines Bildes auch die Matrix gelöscht wird.

6. Fazit

Das Ergebnis der vorangegangenen Arbeit „Konzeption und Vergleich von Bilderkennungsverfahren zur Filterung von Produkteigenschaften“ [Pap] hat bereits die Vorzüge des ORB Verfahrens beschrieben. Die Entwicklung mit OpenCV und dem ORB Verfahren konnte diesen Eindruck bestätigen. Die Bilderkennungs- und Bildvergleichskomponente wurden erfolgreich in die bestehende Systemarchitektur eingebunden. Dabei mussten einige Systemvoraussetzungen von OPDIS eingehalten werden, um die interne Kommunikation zu ermöglichen.

Nach Abschluss aller Entwicklungstätigkeiten ergaben die Tests, dass bei einer Entfernung bis zu 35 cm die Bilderkennung sehr zuverlässig war. Es sind nur wenige fehlgeschlagene Bildvergleiche zu verzeichnen gewesen. Wie in 5.2.2 bereits erwähnt, wurde in einem Test von 20 Versuchen eine Übereinstimmungsquote von 90% erreicht. Die Qualität der einzelnen Übereinstimmungen in den jeweiligen Bildvergleichen wurden nicht gemessen. Es wurden lediglich die Ergebnisse der korrekten Zuordnungen ausgewertet. Zudem wurden die Stärken der Invarianzen genutzt und die Schwächen durch zusätzliche Komponenten ausgeglichen. Der Workaround, der Entwicklung eines Hilfsobjektes, hat sich als besser erwiesen als die OpenCV interne Lösung. Dadurch wird ein Serialisierungsschritt ausgelassen und somit weniger Overhead produziert.

Dennoch wurden auch einige Schwächen offenbart. Zum einen gibt es Probleme, wenn das Zielbild in der Landscapesicht aufgenommen wurden und zum anderen sind einige Inkonsistenzen der verschiedenen OpenCV SDKs aufgetreten. Die Panoramasicht hat die Eigenschaft, dass das Bild breiter ist als hoch. Bei der Übermittlung der Bilderkennungsmatrix an den Dienstanbieter, konnte aus dieser Matrix keine interne OpenCV Matrix generiert werden. Für diesen Fall wurde eine RuntimeException eingebunden, die den Prozess abbricht. Die Inkonsistenzen der OpenCV SDKs machten sich in den verschiedenen Funktionalitäten bemerkbar. Bei der Entwicklung im Android-Bereich fiel auf, dass zum einen Funktionalitäten fehlten und zum anderen verschiedene Datentypen benutzt wurden, um die Algorithmen durchzuführen. Mit Hilfe der offiziellen Dokumentationen, konnten diese Unterschiede schnell aufgeklärt werden.

Eine effektive Ergebnisdarstellung und Ergebnisauswertung war in diesem Projekt stets gegenläufig. Entweder wird der Fokus auf den Benutzer gelegt, um eindeutige Ergebnisse liefern zu können oder der Fokus wird auf die Auslastung des Dienstanbieters gelegt, um möglichst viele Prozesse parallel laufen zu lassen. Bei einer fortlaufenden steigenden Anzahl der Produkte in der Datenbasis, sollten die Schwell- und Grenzwerte nochmals überprüft werden, um die Anzahl der falschen Zufallstreffer zu reduzieren.

Abschließend komme ich zu dem Entschluss, dass das ORB Verfahren gut für mobile Bilderkennungsaufgaben geeignet ist. Die Ergebnisse sind zumindest bei wenigen Testprodukten zuverlässig. Eine hundertprozentige Effizienz ist beim jetzigen Stand der Technik allerdings nicht zu erwarten. Dennoch ist das ORB Verfahren für mobile Geräte zu empfehlen, da ressourcenschonende Algorithmen eingesetzt werden.

Eine Prognose, wie performant das System bei mehreren Tausend Datenbankeinträgen auf Dienstanbieterseite sein wird, kann nicht gegeben werden. Als Risikofaktor werden die Iterationsvorgänge angesehen. Diese müssen gegebenenfalls angepasst werden, so dass weniger Iterationen durchgeführt werden. Ansonsten besteht die weitere Möglichkeit ohne Iterationen auszukommen, indem ein Durchgang durchgeführt wird und die Ergebnisse sortiert in eine Liste übergeben werden.

Der abschließende Praxistest hat bewiesen, dass diese Komponente im alltäglichen Gebrauch einsatzfähig ist und den Status eines Prototypen erreicht hat. Die Stärken des Verfahrens wurden genutzt und die Schwächen durch zusätzliche Komponenten ausgeglichen.

Literaturverzeichnis

[BTG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[CTC+09] Vijay Chandrasekhar, Gabriel Takacs, David M. Chen, Sam S. Tsai, Radek Grzeszczuk, and Bernd Girod. Chog: Compressed histogram of gradients a low bit-rate feature descriptor. In *CVPR*, pages 2504–2511. IEEE, 2009.

[Dix04] A. Dix. *Human-computer Interaction*. Pearson/Prentice-Hall, 2004.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999.

[fS10] International Organization for Standardization. Iso 9241-210:2010 ergonomics of human-system interaction – part 210: Human-centred design for interactive systems. Directly by the International Organization for Standardization, 2010.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Goo] Google. Google goggles. [Online; Stand 25. September 2013].

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[ICC11] *ICCV '11: Proceedings of the 2011 International Conference on Computer Vision*, Washington, DC, USA, 2011. IEEE Computer Society.

[Jäh05] B. Jähne. *Digitale Bildverarbeitung*. Springer, 2005.

[Koo] Kooaba. Kooaba. [Online; Stand 25. September 2013].

[Len00] B. Lenk. *Handbuch der Automatischen Identifikation 1: ID-Techniken, 1D-Codes, 2D-Codes, 3D-Codes*. Handbuch der automatischen Identifikation. Lenk Monika Fachbuchverla, 2000.

[Pap] Danny Pape. Konzeption und vergleich von bilderkennungsverfahren zur filterung von produkteigenschaften.

[Ros99] Paul L. Rosin. Measuring corner properties. *Comput. Vis. Image Underst.*, 73(2):291–307, February 1999.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[TCC+10] Sam S. Tsai, David Chen, Vijay Chandrasekhar, Gabriel Takacs, Ngai-Man Cheung, Ramakrishna Vedantham, Radek Grzeszczuk, and Bernd Girod. Mobile product recognition. In *Proceedings of the international conference on Multimedia*, MM '10, pages 1587–1590, New York, NY, USA, 2010. ACM.

[Til09] Stefan Tilkov. *REST und HTTP: Einsatz der Architektur des Web für Integrationsszenarien.* dpunkt, Heidelberg, 2009.

Anhang

A. User Stories

A.1. User Story A: keine oder eingeschränkte Konnektivität

Der Benutzer möchte ein Produkt fotografieren und anschließend die Ergebnisse des Produktscans betrachten. Allerdings hat der Benutzer zur Zeit keine Internetverbindung und kann somit keine Verbindung zum Dienstanbieter herstellen.

A.2. User Story B: erfolgreicher Produktscan

Der Benutzer ist sich beim Kauf eines Produktes nicht sicher, ob es frei von Allergika ist. Durch die Nutzung von OPDIS scant er das Bild ein, erhält ein eindeutiges Ergebnis und möchte die Informationen deuten und erkennen ob das Produkt frei von Allergika ist.

A.3. User Story C: erfolgloser Produktscan

Der Benutzer scant ein Bild mit der OPDIS-APP ein und erkennt das die Bilderkennung zwar durchlaufen wird, aber kein Treffer angezeigt bekommt.

A.4. User Story D: bedingt erfolgreicher Produktscan

Der Benutzer scant ein Bild mit der OPDIS-APP ein und erhält, entgegen seiner Erwartungen mehrere Ergebnisse.

A.5. User Story E: Distanz, Hintergrund und Perspektive

Der Benutzer sitzt im Rollstuhl und möchte ein Produkt direkt aus dem Regal fotografieren, um detaillierte Informationen darüber zu bekommen.

A.6. User Story F: Bildqualität

Der Benutzer hat ein Kind im Arm und wenig Bewegungsspielraum. Weiterhin wird dieser ununterbrochen durch das Kind in Bewegung gehalten. Trotzdem möchte der Benutzer ein Bild fotografieren, um detaillierte Informationen zu bekommen

B. ElasticSearch

B.1. Produktdaten


```
{ "index":{"_index":"opdis","_type":"Cosmetics",
"_id":"522268dc-b61c-4aec-b2d9-d7b9deb90341"} }
{"_class": "de.opdis.core.Cosmetics","additionalDescription": "UTF-8: ???",
"additionalInfoList": [{"_ref_id": "c6f52459-74e1-4349-88ed-17630da3dc3a",
"_ref_class": "de.opdis.core.AdditionalInfo"},
{"_ref_id": "7bdb95ce-9eb9-424c-a031-ee488be5d7f5",
"_ref_class": "de.opdis.core.AdditionalInfo"}],
"allergyInfoList": [{"_ref_id": "2fdf1504-cb32-4a16-84eb-54836d80d150",
"_ref_class": "de.opdis.core.AllergyInfo"},
{"_ref_id": "24a319eb-e047-45e9-a003-d92cb6d684e2",
"_ref_class":"de.opdis.core.AllergyInfo" }],
"averageRating": 2.460447,
"brand": {"_ref_id": "02fdc1b2-a9c6-4920-ba10-8c7d8389ce40",
"_ref_class": "de.opdis.core.Brand"},
"company": {"_ref_id": "4f395481-f4d7-416b-8aa7-07d986d2264e",
"_ref_class": "de.opdis.core.Organization"},
"description": "Description Braucht Ihre Haut viel Feuchtigkeit? Dazu gibt es
eine ganz unkomplizierte Pflege: versorgt die Haut 24 Stunden mit Feuchtigkeit.
Vitamin E schützt die Haut,
die Creme ist leicht zu verteilen und zieht sofort ein.",
"eanList": ["4021457605033","3600520276489"],
"imageSetList": [
{"_class": "de.opdis.core.ImageSet",
"imageList": [ {"_ref_id": "aelx", "_ref_class": "de.opdis.core.Image"},
{"_ref_id": "jdapyw", "_ref_class": "de.opdis.core.Image"},
{"_ref_id": "td", "_ref_class": "de.opdis.core.Image"}], "name": "ImageSet 1"},
{"_class": "de.opdis.core.ImageSet",
"imageList": [{"_ref_id": "mxoy", "_ref_class": "de.opdis.core.Image"}],
"name": "ImageSet 2"}],
"ingredients": "Aqua, Paraffinum Liquidum, Cera Microcristallina, Polyglyceryl-4
Isostearate , Glycerin, Paraffin , Lanolin Alcohol, Magnesium Sulfate, Geraniol,
Citronellol , Linalool , Butylphenyl Methylpropional, Limonene, Parfum",
"labelList": [{"_ref_id": "030f5790-83e8-452e-a554-5290c762a932",
"_ref_class": "de.opdis.core.Label"}],
"name": "Florena Creme 1",
"nameLong": "Florena Creme 1 150 ml",
"numberOfRatings": 2053746445,
"packagingMaterial": "Kunststoffdose",
"packagingUnit": {"_class": "de.opdis.core.Quantity","quantity": "24.94",
"unit": "ml"},
"producer": {"_ref_id": "4f395481-f4d7-416b-8aa7-07d986d2264e",
```

```
42 "__ref__class": "de.opdis.core.Organization"},
43 "productCategory": {"__ref__id": "4530c099−3699−43bf−9df1−b29d64d43c09",
44 "__ref__class": "de.opdis.core.ProductCategory"}, "warning": "nicht␣bekannt"}
```