Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Anas Garwal
[redacted]

Konzeption und Realisierung einer OMCR Android App

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Media Systems
am Department Medientechnik
der Fakultät Design, Medien und Information
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Andreas Plaß
Zweitgutachter: Dipl.-Inf. Christian Dreyer

Autor

Anas Garwal

Thema der Bachelorarbeit

Konzeption und Realisierung einer OMCR Android App

Stichworte

Android, Java, XML, OMCR

Kurzzusammenfassung

Die Zunahme an mobilen Endgeräten ist kaum zu übersehen. Immer mehr Menschen und Unternehmen wollen die Vorteile der mobilen Endgeräte für sich nutzen. Die mobilen Endgeräte können verschiedene Betriebssysteme beinhalten. Dazu gehören Android, iOS und Windows Phone, um nur einige zu nennen. Diese Bachelorarbeit widmet sich der Gestaltung und Programmierung einer OMCR Android App, insbesondere vor dem Hintergrund der Konzeptionierung und Realisierung. Vorlage dieser OMCR Android App ist eine schon im Unternehmen genutzte Software.

I. Inhaltsverzeichnis

1 Einleitung

Die Zunahme an mobilen Endgeräten ist kaum zu übersehen. Immer mehr Menschen und Unternehmen wollen die Vorteile der mobilen Endgeräte für sich nutzen. Alle Parteien hegen persönliche Interessen an den mobilen Endgeräten. Viele Unternehmen wollen beispielsweise ihre Prozesse vereinfachen und einen schnellen Ablauf gewähren, sowohl für ihre Kunden als auch für ihre Mitarbeiter. Das führt dazu, dass die Kunden zum einen gerne wiederkommen und zum anderen, dass den Mitarbeitern das Arbeiten erleichtert wird. Ein weiteres Motiv ist, dass die Unternehmen so mehr Umsätze und folglich mehr Gewinne erzielen können.

Im Rahmen dieser Arbeit schreibe ich für die Gesellschaft Peek & Cloppenburg eine dementsprechende Software für eine Android App. Die Android App soll es den Mitarbeitern ermöglichen, die Handelsware in einem einfachen Prozess und dadurch zeitlich optimiert zu finden, um diese dann anschließend in die Versandbearbeitung weiterzugeben. Bisher nutzt die Gesellschaft ein stationäres System, aus dem sich der jeweilige Mitarbeiter eine Liste der gewünschten Artikel ausdrucken kann. Mit dieser Liste begibt sich der Mitarbeiter auf die Suche nach den Artikeln, welche sowohl im Lager als auch in der Filiale zu finden sind.

In dieser Arbeit wird der Leser chronologisch durch die Prozesse der Konzeptionierung und der Realisierung dieser von mir entwickelten Software geführt. Im ersten Schritt, in der Anforderungsanalyse, wird herausgearbeitet, welche Funktionen die App beinhalten muss. Danach wird sich die Arbeit der Konzeption der App widmen. Anschließend wird die grafische Bedienoberfläche als Idee vorgestellt. Dies wird anhand von Skizzen veranschaulicht. Darauffolgend wird die Softwarearchitektur behandelt. Am Ende der Arbeit wird die Realisierung behandelt, wobei darauf eingegangen wird, welche Punkte dafür maßgebend sind.

2 OMCR

Omnichannel Retailing (OMCR), was nicht mit Multi-Channel Retailing verwechselt werden darf, ist das Verfahren, des Produktverkaufes durch mehrere Plattformen. Hierbei gilt, dass Kunden Waren sowohl im Store als auch Online kaufen und reklamieren können. Dabei spielt es keine Rolle, ob die Ware auf einer anderen Plattform als beim Erwerb reklamiert wird. Des Weiteren wird es ermöglicht, den Bestand eines Artikels aus anderen Filialen anzufragen und diesen bei Bedarf in eine beliebige Filiale zu liefern. Das Multi-Channel Retailing macht hingegen eine Trennung zwischen Store- und Onlinebestand. Im Grunde geht es darum, dass alle zur Verfügung stehenden Plattformen für den Erwerb von Waren kombiniert genutzt werden, sei es ein lokales Geschäft, eine Webseite, eine App oder ein Katalog. Alle Plattformen nutzen den gleichen Bestand und führen zum gleichen Produkt. Dies ermöglicht es dem Kunden, den Kauf auf der von ihm bevorzugten Plattform abzuschließen. Zu beachten ist, dass das Netzwerk gut strukturiert und immer aktuell sein muss, sodass bei Änderungen alle involvierten Plattformen auf demselben Stand sind. Dies gilt auch für Rabattaktionen oder ähnliches. Sobald der Bestand eines Artikels verbraucht ist, wird der Artikel von allen Plattformen entfernt, bis dieser erneut lieferbar ist. Dabei ist es, wie schon erwähnt, irrelevant, ob der Einkauf über die Webseite stattgefunden hat oder über die App. Der Nutzer hat jedoch die Möglichkeit sich auf der Webseite hinsichtlich neuer oder bestehender Artikel zu orientieren, sie jedoch letzten Endes doch im lokalen Geschäft zu kaufen. Sollte der Kauf eines Artikel über eine der online zur Verfügung gestellten Plattformen erfolgen, wird der Artikel im Lager oder in den Filialen gesucht und an den Kunden geliefert. Dabei ist auch der Ort des Wareneinkaufs irrelevant. So kann der Kauf in jedem beliebigen Ort stattfinden. Darüber hinaus kann der Kunde den Artikel in einer Stadt kaufen und ihn bei Bedarf in einer anderen Stadt reklamieren. Sollte der Kunde einen Artikel in einer anderen Größe benötigen, kann dieser in anderen Filialen angefragt werden. Ein Weiterer großer Nutzen für das Omnichannel Retailing (OMCR) ist das Online Marketing. Je mehr Portale zu Verfügung stehen, desto mehr Möglichkeiten bestehen zur Kontaktaufnehme zu den Kunden.

Abbildung 1: OMCR Schema [24]

Anhand der oben dargestellten Abbildung ist zu erkennen, dass die Plattform des Wareneerwerbs für den Kunden irrelevant ist. Der Kunde kann die gewünschte Ware von jeder dieser Plattformen erhalten. Daher sollten Unternehmen den Kunden so viele Plattformen wie möglich zur Verfügung stellen. Wichtig dabei ist, dass der Bestand auf allen Ebenen einheitlich dargestellt wird. So sollten auch Coupons und Gutscheine auf allen Plattformen kaufbar und einlösbar sein. Unvorteilhaft wäre es, wenn der Kunde Gutscheine oder Coupons nur auf einer Plattform verwenden kann. Dies wäre eine Einschränkung, die den Kunden dazu zwingen könnte, nur eine bestimmte Plattform zu nutzen. Eine mögliche Folge dessen wäre ein Rückgang der Kundenzufriedenheit, was zu einem Verlust dieser Kunden führen könnte.

3 Grundlagen

Im Folgenden wird ein kurzer Einblick in die Geschichte gegeben. Dies hat den Vorteil schon bekanntes Wissen anwenden oder eventuell Neues lernen zu können.

Android wurde am 23.09.2008 erstmals veröffentlicht. Aktuell befindet sich Android auf der Hauptversion 10. Android ist nicht nur ein Betriebssystem, sondern auch eine Software-Plattform für mobile Endgeräte. Es handelt sich hierbei um eine freie Software; eine freie Software deshalb, weil zusammen mit der Software die dazugehörigen Nutzungsrechte geliefert werden. Außerdem hat der Nutzer das Recht den originalen Quellcode einzusehen und ihn nach Belieben zu verändern. Sowohl das Original als auch die veränderte Version dürfen veröffentlicht und publiziert werden. Die Basis des Betriebssystems beruht auf Linux-Kernal. Im Jahr 2017 befanden sich weltweit bereits circa 2,7 Milliarden Android- Smartphones im Umlauf.

Android gibt es nicht nur für das Smartphone, sondern auch für Smartwatches, Tablets, Fernsehgeräte und sogar für das Auto. Einer der großen Vorzüge ist, dass man alle Geräte ohne viel Aufwand miteinander verbinden kann.

Peek & Cloppenburg ist ein Modeunternehmen mit Sitz in Hamburg. Im Jahr 1912 wurde das erste Geschäft von Anton Cloppenburg und Paul Schröder eröffnet und besteht nunmehr seit über 100 Jahren. Im Laufe der Zeit gab es viele Veränderungen und Konkurrenten stiegen in den Mark ein.
Das Statistische Bundesamt (Destatis) berichtet, dass 2019 84 Prozent der Internetnutzer online einkauften. Grund dafür ist unter anderem, dass die Preise online direkt verglichen werden. Außerdem werden die Einkäufe nach Hause geliefert, was sehr bequem ist. Am beliebtesten bei den Online-Einkäufen waren in Deutschland im Jahr 2019 die Warengruppen Bekleidung und Sportartikel mit 69 Prozentpunkten. Der Vorteil von OMCR zeigt sich auch hier. Dem Kunden die freie Wahl zu geben, wie dieser seine Ware einkaufen möchte, erweitert das Spektrum an Kunden immens. Ein weiterer Grund für das Erweitern des lokalen Netzwerkes in das digitale Netzwerk sind Online-Shop-Giganten wie Amazon und Otto oder Zalando, um nur einige zu nennen. Auf der vom EHI Retail Institute erstellten Liste der 100 umsatzstärksten Online-Shops Deutschlands für das Jahr 2018 stehen Amazon, Otto und Zalando auf den ersten drei Positionen. Um konkurrenzfähig zu bleiben, müssen Unternehmen sich auf alle Bereiche ausdehnen. Peek & Cloppenburg präsentiert sich auf vielen Ebenen. Kunden können ihre Ware nicht nur in den 22 Filialen in Deutschland, sondern auch im Online-Shop oder über die App bestellen. Bei Peek & Cloppenburg befinden sich Zebra Mobilegeräte im Einsatz. Dies sind sehr robuste mobile Geräte, welche einen integrierten Scanner besitzen und mit dem Betriebssystem Android kompatibel sind. Als Vorgabe von Peek & Cloppenburg soll die OMCR App als Native App entwickelt werden.

3.1 Native vs. Web Apps

Native Apps sind betriebssystemspezifisch. Somit kann man beispielsweise mit den Sprachen Android (Java, XML) und Kotlin Apps für Android oder mit der Sprache Swift Apps für das IOS-Betriebssystem entwickeln. Die jeweiligen Apps funktionieren demnach nur auf Geräten, welche das jeweilige Betriebssystem unterstützen. Damit wird gewährleistet, dass die App sicher und problemlos mit der Hardware des Gerätes zusammenarbeitet. Web Apps hingegen werden mit HTML, CSS und JavaScript geschrieben. Hierbei wird eine HTML5-Webseite programmiert, die das mobile Endgerät erkennt und wobei sich die Ausgabe des Inhaltes diesem anpasst. Der Vorteil ist, dass jedes Endgerät mit einem Browser eine solche App nutzen kann. Weitere Vorteile von Native Apps sind, dass sie umfangreich benutzt werden und rechenintensiv sein können. Von besonderer Wichtigkeit ist hier zudem, dass die Apps offline nutzbar sein können. Außerdem können Hardware-Komponenten einfacher und besser verwendet werden. Bei Web Apps hingegen handelt es sich nicht um echte Apps, sondern vielmehr um Internetseiten. Sie werden als Lesezeichen auf dem mobilen Gerät gespeichert und müssen nicht wie Native Apps installiert werden. Des Weiteren ist bei Web Apps die Geschwindigkeit des Internets des Nutzers wichtig, um eine gute Performance gewährleisten zu können. Ein Nachteil ist, dass Web Apps nicht offline nutzbar sind und die Anbindung von Hardware-Komponenten nicht möglich ist.

Aufgrund der genannten Vor- und Nachteile fällt die Entscheidung eine Native App zu entwickeln leicht. Die App wird für ein mobiles Zebra-Gerät entwickelt. Die schon erwähnten Vorteile eine Native App zu entwickeln überwiegen denjenigen einer Web App. Am wichtigsten ist der in diesem Gerät installierte Scanner, sodass die Nutzer keine externen Scanner mit sich tragen müssen. Außerdem kann einfacher auf den Speicher des mobilen Gerätes Zugriff genommen werden. Dies hat den Vorteil Daten hinterlegen zu können. Im Unternehmen sind nur Android-Geräte im Einsatz, daher wäre es für Peek und Cloppenburg irrelevant ein anderes Betriebssystem zu entwickeln.

4 Anforderungsanalyse

Um zu verstehen, wie die Software auszusehen hat, sollte man wissen, wie der Prozess innerhalb der Organisation abläuft. Aufgrund dessen war ich an einem zuvor ausgewählten Tag als Kommissionierer in den Filialen und Lagerräumen tätig. Dadurch konnte ich bestehende Fehler bzw. Probleme erkennen, um Verbesserungen und eventuelle Vereinfachungen von Kommissionierungsabläufen in der neuen Android App Software zu berücksichtigen.

4.1 Bestehender Prozess

Der bestehende Prozess läuft wie folgt ab: Der Kunde kann im Online-Shop einen Artikel bestellen. Die Anfrage geht zunächst an das Distributionszentrum. Dort wird geprüft, ob der gewünschte Artikel vorhanden ist. Sollte der Artikel nicht im Distributionszentrum vorhanden sein, wird eine Anfrage an eine Filiale gesendet. Sowohl die Anwendung mit der Artikelanfrage als auch die Anwendung mit dem angeforderten Artikel wurden firmenintern entwickelt. Beide Anwendungen basieren auf Java. Auf der Anwendung in der Filiale ist dann zu sehen, welcher Artikel gefordert wird. Außerdem wird noch angezeigt, wann er angefordert wurde und wie dringend die Anfrage ist. Um eindeutig erkennen zu können, um welchen Artikel es sich handelt, hat jeder Artikel eine eindeutige und individuelle Nummer. Aus dieser Nummer ist herauszulesen, um welchen Artikel es sich handelt. Ebenfalls lassen sich die Größe, die Abteilung, in welcher der Artikel zu finden ist und die Verkaufssaison ermitteln.

Grundsätzlich verfügt jede Station über einen Windows-Rechner, einen Standard-Drucker, einen Scanner und gegebenenfalls über einen Etikettendrucker. Jedem Rechner ist die jeweilige Filiale zugewiesen. Dies ist für den späteren Verlauf der Kommissionierung wichtig. Abgesehen davon sind selbstverständlich weitere Einstellungen vorgenommen worden, die für den weiteren Verlauf der Arbeit jedoch nicht bedeutend sind.

Bevor die Anzeige des angeforderten Artikels erscheint, müssen einige Filterkriterien ausgewählt und gegebenenfalls erfüllt werden. Ein Pflichtfeld ist unter anderem das Erstellungsdatum der Anfrage, sprich wann der Artikel angefragt wurde. Daraus lässt sich ermitteln, welche Artikel eventuell Priorität haben. Zusätzlich dazu gibt es das Feld „Status“, worin die Statusmeldungen „dringend“, „offen“, „versendet“ oder „abgesagt“ erscheinen. Eines dieser Felder sollte gewählt werden. Als Standard wird der Status „offen“ angezeigt, da alle Artikel, die offen und dringend sind, gesucht werden. Des Weiteren wird automatisch die Filiale mit angegeben. Optional können der Bereich und das Ressort eingegeben werden. Die Wörter „Bereich“ und „Ressort“ sind im Unternehmen vorhanden, um unter verschiedenen Artikeln entscheiden zu können, sofern nur bestimmte Artikel angezeigt werden sollten. Ein Anwendungsfall wäre, wenn mehrere Mitarbeiter gleichzeitig arbeiten. Um Überschneidungen zu vermeiden, könnten die Bereiche unterteilt werden.

Nachdem die Filterkriterien ausgefüllt worden sind, wird ein Service-Aufruf gemacht, der dann alle angefragten Artikel für eine bestimmte Filiale mit den ausgewählten Filterkriterien liefert. Anschließend lässt sich die Liste noch sortieren. Die Liste lässt sich nach den Angezeigten Attributen sortieren, sowohl nach Bereich, Ressort, Saison oder Status, als auch nach Erstellungsdatum und Einkaufsnummer. Es können mehrere Artikel ausgewählt und gedruckt werden. Mit der ausgedruckten Liste wird dann nach dem Artikel gesucht, entweder im Lager der Filiale oder auch auf der Verkaufsfläche der Filiale, sollte der Artikel nicht im Lager vorhanden sein.

Zusätzlich lässt sich ein Foto von einem Artikel anzeigen, und zwar nachdem man einen Artikel aus der Liste ausgewählt hat und dann auf den Button „Fotos“ klickt. Es erscheint ein Fenster mit einem Foto des gewünschten Artikels.

Anschließend werden die gefundenen Artikel zurück zur Station gebracht und dort weiterverarbeitet. Sollte ein Artikel nicht gefunden werden, kann dieser abgesagt werden. In dieser Oberfläche muss dann ein Grund für die Absage ausgewählt werden. Anhand des Grundes lässt sich nachvollziehen, wieso der Artikel nicht geliefert werden kann. Außerdem gewährleistet dies die sorgfältige Arbeit des Mitarbeiters. Nachdem der Artikel abgesagt worden ist, wird dieser in einer anderen Filiale angefragt. Für die bereits gefundenen Artikel wird ein Lieferschein generiert, der von der Art des Artikels abhängig ist. Es bestehen zwei verschiedene Arten von Ware: Zum einen gibt es die Hängeware. Als Hängeware werden meist Artikel bezeichnet, die groß sind und an einem Bügel hängen. Zum anderen gibt es die Liegeware, wobei es sich meist um kleine Artikel handelt. Mehrere dieser Artikeln werden gemeinsam in einer Box verpackt.

Abschließend, nachdem alle angeforderten Artikel der Aufträge bearbeitet worden sind, wird ein Warenausgangsbeleg für alle Lieferscheine generiert. Somit sind alle Artikel zur Rücksendung ins Distributionszentrum erfasst. Von dort aus geht die Ware zum Kunden.

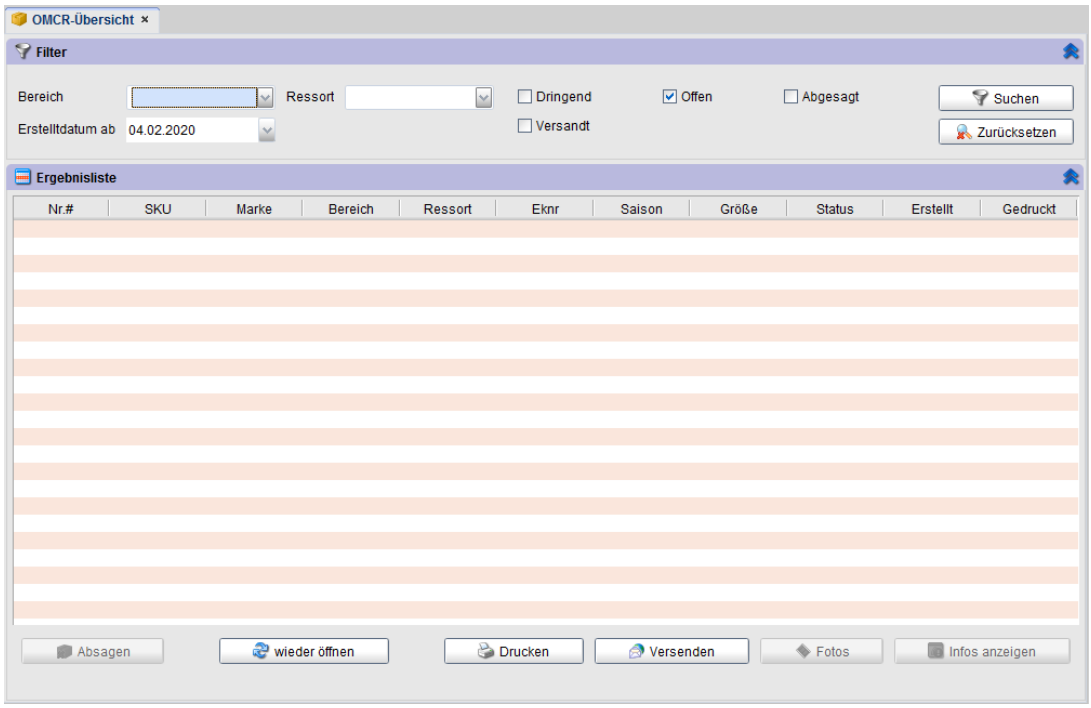

Abbildung 2: Bestehende Anwendung im Unternehmen

Abb. 2 zeigt die zuvor beschriebene Anwendung für den Prozess der Artikelanforderungen. Die Voreinstellungen betreffen das Erstellungsdatum und den Status der Anforderung. Das Erstellungsdatum ist so gewählt, dass es zwei Wochen zurückgeht. Dies ist dafür gedacht, dass der Nutzer schneller bzw. mit einem Klick alle Anforderungen einsehen kann. Die Bearbeitung einer Anforderung sollte nicht länger als zwei Wochen dauern, andernfalls kann das Datum auch verändert werden, sodass ältere Anforderungen angezeigt werden. Bei der Suche nach den Anforderungen werden in der Ergebnisliste die Zeilen mit dem jeweiligen Artikel und in den Spalten die einzelnen Werte zu dem Artikel angezeigt. Ein Zurücksetzen würde alle Artikel aus der Ergebnisliste löschen und veränderte Filterkriterien würden auf die Voreinstellungen zurückgesetzt. Mehrere Artikel können ausgewählt werden und mit dem Button „Drucken“ wird eine Liste von den ausgewählten Artikeln gedruckt. Zusätzlich dazu lassen sich zu jeder Anforderung bzw. Zeile eine genaue Informationen über die Zeit der Erstellung und die zuletzt bearbeitete Filiale anzeigen. Außerdem kann von jedem Artikel ein Foto angezeigt werden Die Anforderung des Artikels lässt sich auch absagen. Dafür erscheint ein Fenster, indem der Grund der Absage ausgewählt werden muss. Beim Wieder öffnen können Abgesagte Anforderungen erneut abrufen und bearbeitet werden. Abschließend können alle gefundenen Artikel versandt werden. Auch hier wird ein Fenster geöffnet, das prüft, ob der Artikel zu einer Anforderung gehört oder nicht. Dazu wird ein Lieferschein mit den Daten der Artikel gedruckt.

4.1.1 UML UseCase Diagramm

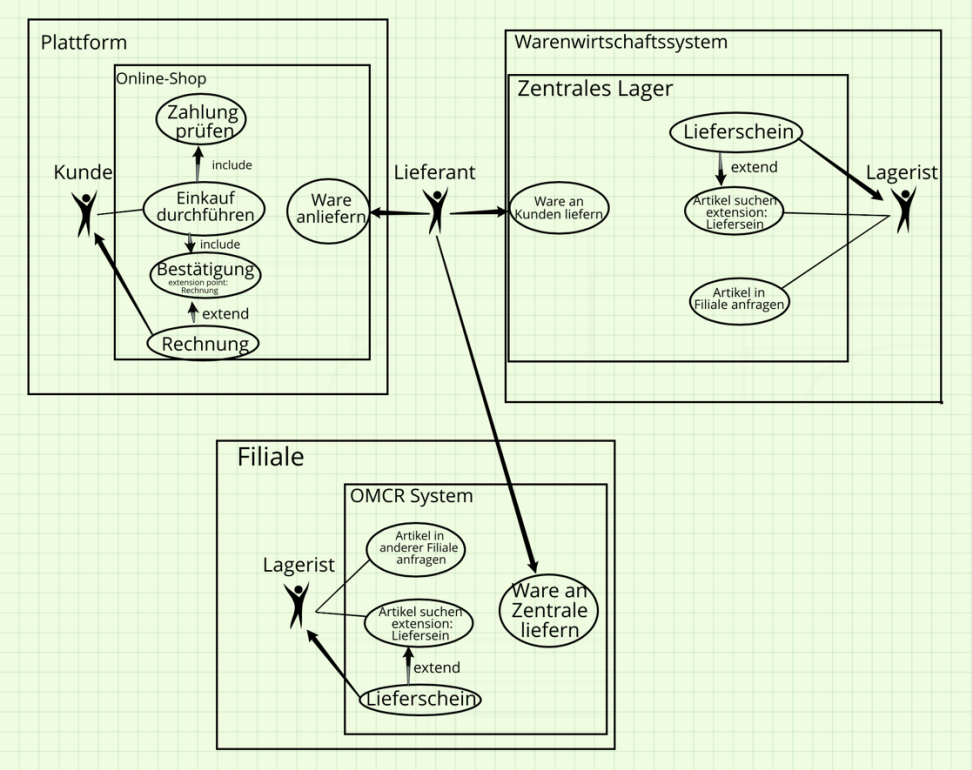

Abbildung 3: UML UseCase Diagramm [eigene Darstellung]

In Abb. 3 ist das UseCase Diagramm für den bestehenden Online-Prozess abgebildet. UML steht für Unified Modelling Language (UML) und ist eine Modellierungssprache. Hierbei wurden die Ansätze verschiedener Autoren zusammengesetzt, um eine Einheit zu erhalten. Die Modellierungssprache dient der Spezifikation, Visualisierung und Dokumentation von Systemen in allen Phasen ihrer Entwicklung.

Links oben ist das System der Plattform abgebildet. Der Kunde hat die Wahl seine Ware über die App oder den Online-Shop zu bestellen. Bei dem Prozess der Bestellung führt der Kunde seinen Einkauf selbst durch, wobei zeitgleich die Zahlung geprüft wird. Sowohl direkt auf der Plattform als auch durch eine Bestätigungs-E-Mail wird dem Kunden der Einkauf bestätigt. Mit der E-Mail folgt auch die Rechnung. In der Abbildung ist der Prozess des Online-Shops abgebildet, welcher sich jedoch nicht von dem der App unterscheidet. Bei einem Einkauf in der Filiale hingegen wird dem Kunden die Rechnung in Papierform übergeben. Nachdem der Artikel erfolgreich bestellt worden ist, bekommt die Zentrale den Auftrag. Die Zentrale prüft, ob sich der Artikel im Lager befindet. Sollte dies der Fall sein, wird der Artikel eingepackt und es wird ein Lieferschein dazu ausgedruckt. Abschließend wird der Artikel vom Lieferanten zum Kunden geliefert. Sollte jedoch der Artikel nicht im zentralen Lager aufgefunden werden, wird der Artikel in einer der 22 Filialen angefragt.

Nachdem der Auftrag in der Filiale angekommen ist, wird er im Lager der Filiale gesucht. Sollte der Artikel im Lager nicht zu finden sein, wird auf der Verkaufsfläche nach dem Artikel gesucht. Abschließend wird der Artikel zusammen mit dem Lieferschein an die Filiale geliefert. Sollte der Artikel jedoch nicht in der Filiale zu finden sein, da er zwischenzeitlich verkauft wurde, wird eine Anfrage an eine andere Filiale geschickt. Sammelpunkt aller Artikel ist das zentrale Lager. Vorteilhaft ist dabei, dass, sofern der Kunde mehrere Artikel bestellt, diese aus verschiedenen Filialen in die Zentrale geliefert und von da aus gesammelt an den Kunden weitergeleitet werden. Dies spart Verpackung und Transportkosten. Abgesehen davon erhält der Kunde lediglich ein Paket anstatt mehrere.

4.2 Anforderung

Anhand dieses Prozesses lassen sich spezielle Anforderungen an die neu zu entwickelnde App und die erforderlichen Features ableiten. Außerdem können Verbesserungen vorgenommen werden, die die Effizienz der Mitarbeiter fördern.

Einstellungen:

Die App soll so konfiguriert werden, dass der App zu jeder Zeit bekannt ist in welcher Filiale sie sich befindet, da es sich hier um ein großes Unternehmen mit 22 Filialen handelt. Hierbei sollte es sich um einen Wert handeln, der vorkonfiguriert wird. Dieser sollte nicht vom Benutzer veränderbar sein. Die stationären Systeme sind eindeutig einer Filiale zugewiesen, somit kann immer nachvollzogen werden, wo sich die Station befindet. Nur Administratoren können diese Einstellung ändern.

Scanner:

Wie im realen Prozess sollen auch in der App Etiketten gescannt werden. Ein Vorteil wäre es, direkt vom mobilen Gerät scannen zu können, ohne es mit einem externen Scanner verbinden zu müssen. Dies ist mit dem Zebra-Gerät möglich. Zebra liefert für das Entwickeln von Android Apps ein Enterprise Mobile Development Kit (EMDK), welches das Verwenden des Scanners erleichtert.

Filter:

Bei jedem Aufruf der bestehenden Anwendung soll das Filterkriterium ausgefüllt werden. Diese Funktion muss ebenfalls in der App vorhanden sein. Am besten sollte alles genauso übernommen werden. Ein Vorteil davon ist, dass keine aufwändige Einführung oder Schulung für die App notwendig ist. Dies spart viel Erklärungsbedarf. Somit lassen sich neue Eigenschaften erklären, wie zum Beispiel, dass das Scannen möglich gemacht werden kann.

Pflichtfelder wie das Erstellungsdatum müssen immer ausgefüllt werden. Des Weiteren sollte der Status der Anfrage auswählbar sein. Hierbei kann auch standardmäßig der Status „offen“ ausgewählt sein, was den Vorteil hat, dass mit einem Knopfdruck eine Übersicht von allen angefragten Artikeln angezeigt werden kann. Außerdem sollte wie im bestehenden Prozess „Bereich“ und „Ressort“ verwendbar sein.

Liste:

Eine Ansicht der angeforderten Artikel muss dem Nutzer angezeigt werden. Diese Liste sollte etwas komprimiert sein, sodass nur essentiale Daten angezeigt werden. Die App wird auf einem wesentlich kleineren Gerät installiert und da die Benutzeroberfläche kleiner ist, kann auch nur weniger angezeigt werden. Hierbei können vom bestehenden Prozess die folgenden wichtigsten Attribute übernommen werden: Erstellungsdatum, Status, Bereich und Ressort.

Foto:

Ebenfalls soll die App ein Foto des gewünschten Artikels liefern, da es sich als sehr praktisch erweist, mit einem mobilen Gerät durch das Lager oder die Verkaufsfläche zu gehen und dabei den Artikel vor Augen zu haben. So muss der Benutzer sich nicht jedes einzelne Etikett anschauen. Außerdem kann der erste Blick entscheiden

Offline Nutzbarkeit:

Da der Nutzer eventuell nicht überall Empfang hat, sollte die App einige offline Features haben. Eine solche Feature wäre die Vermeidung des Verlustes von geladenen Daten, je nachdem wie viele Datensätze geladen werden. Wichtig hierfür ist die Performance des Systems. Die App sollte nicht die Effizienz der Nutzer behindern, sondern vielmehr fördern. Die andere Feature wäre, alle Vorgänge zwischenzuspeichern. Somit wird nur der bisher bearbeitete Artikel zwischengespeichert. Im ersten Fall kann auch ohne Internet weitergearbeitet werden. Im zweiten Fall können nur die schon bearbeiteten Artikel weiterverarbeitet werden. Sobald Empfang besteht, können die Vorgänge verschickt werden.

Zielgruppe:

Während im Hintergrund Logistiker die Aufträge aufgeben, werden Kommissionierer im Vordergrund die App nutzen. Abgesehen von diesen Nutzern ist die App an niemand sonst gerichtet. Die App wird ausschließlich für den internen Gebrauch genutzt. Die Verteilung der App erfolgt auch intern und wird nicht im Google Store vorhanden sein. Es befindet sich in allen Filialen mindestens eine OMCR-Station, die sich um die angefragten Artikel kümmert. Somit sollte in jeder Filiale die App die vorhandene Station ersetzen.

4.3 Datenmodell

Zusätzlich zu der in der Anforderung beschriebenen offline Nutzbarkeit muss eine lokale Datenbank auf dem Gerät installiert sein. Hierfür gibt es verschiedene Datenmodelle, die jeweils Vor- und Nachteile haben. Das bekannteste und weitverbreitetste Modell ist die relationale Datenbank. Hierbei handelt es sich um ein Datenmodell, das Daten in verschiedenen Tabellen speichert. Zusätzlich dazu werden zwischen den Tabellen Beziehungen aufgebaut. Um Daten aus der Datenbank zu bearbeiten oder abzufragen, wird die Structured Query Language (SQL) verwendet. Grundsätzlich werden in Tabellen die Zeilen als Tupel und die Spalten als Attribute bezeichnet. Große Vorteile von relationalen Datenbanken sind zum einen die einfache Erstellung und zum anderen die vielfältige Einsetzbarkeit. Der Vorgang Normalisierung ist ein wesentlicher Aspekt in der relationalen Datenbank. Hierbei werden Datensätze in verschiedenen Tabellen gespeichert, die dann miteinander verknüpft werden. Das soll dazu führen, dass Daten eindeutig bestimmt werden können, sodass die Daten nicht mehrfach in der Datenbank aufzufinden sind. Um diesen Vorgang durchzuführen, werden Primär- und Fremdschlüssel verwendet. In einer Tabelle, die Länder widerspiegeln, können den Ländern Primärschlüssel zugewiesen werden. Dieser Primärschlüssel ist zur eindeutigen Identifikation des Tupels bestimmt. In der Tabelle, kann ein Attribut die Stadt sein. Für die Tabelle „stadt“ kann die Vorwahl als Primärschlüssel dienen. Dieser Primärschlüssel kann als Fremdschlüssel in der Tabelle „land“ hinterlegt werden, um so eine Beziehung zwischen den beiden Tabellen aufzubauen. So würden viele redundante Daten verfallen.

Config:

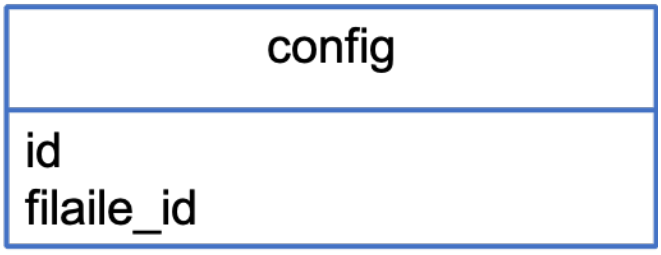

Abbildung 4: Lokale Datenbanktabelle "config" [eigene Darstellung]

Bei der Tabelle „config“, soll die Filiale hinterlegt werden. Das hat den Vorteil, dass bei der Konfigurierung und Installation des Gerätes die Filiale direkt mit angegeben werden kann. Dadurch muss die Filiale nicht immer mit der Datenbank abgeglichen oder abgefragt werden. Abgesehen von dem Attribut „id“ kann diese Tabelle einen Fremdschlüssel zur Tabelle „filiale“ haben.

Filiale:

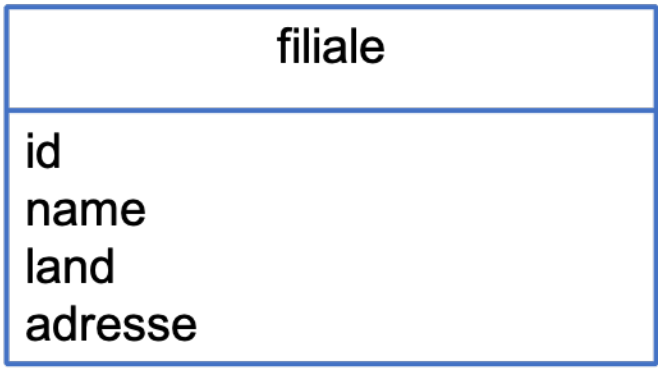

Abbildung 5: Datenbanktabelle "filiale" [eigene Darstellung]

Ausgenommen von den Attributen „name“, „land“ und „adresse“ ist der Primärschlüssel dieser Tabelle die „id“, welche der Fremdschlüssel von „config“ ist. Dadurch lässt sich eine Beziehung zu den beiden Tabellen bilden. Als Primärschlüssel könnte hier vielerlei Verwendung finden. Jedoch ist zu berücksichtigen, dass es sich dabei um einen eindeutigen Schlüssel handelt. Dieser kann ein bestimmter Wert oder eine Kombination aus „name“ und „adresse“ sein. Wie im oben genannten Beispiel können den Filialen Nummern zugewiesen werden, was auch von Peek & Cloppenburg so umgesetzt wird. Zusätzlich dazu könnten Tabellen für das Land und die Adresse geschaffen werden. Hierbei ist ebenfalls zu berücksichtigen, welche Daten notwendig sind und welche nicht. Unnötiges Zwischenspeichern von Daten kann das Datenmodell unnötig verkomplizieren. Abgesehen davon wird die Datenbank lokal auf einem mobilen Gerät gespeichert, welches nicht dieselbe Rechenleistung hat wie ein Computer. Zu viele Datensätze können zu Performance-Verlust beim mobilen Gerät führen.

Artikel:

Abbildung 6: Datenbanktabelle "artikel" [eigene Darstellung]

Bei der Tabelle „artikel“ wird ebenfalls ein Primärschlüssel verwendet. Dieser kann automatisch erhöht werden oder eine eindeutige Kombination von verschiedenen Werten bilden. Auch hier wird die Filiale als Fremdschlüssel gespeichert und kann so eine Beziehung zu der Tabelle der Filiale herstellen. Außerdem können hier alle Werte der Artikel gespeichert werden. Da die Daten der Artikel nach einem Aufruf des Backend realisiert werden soll, können die Objekte sich dem des Backend anpassen. Somit würden alle Daten, die hereinkommen, auch gespeichert. Bei speziellen Fällen können die Daten auch unterschiedlich groß und relevant sein, daher muss abgeschätzt werden, welche tatsächlich benötigt werden. Es könnten auch zu wenig Daten geladen werden, wodurch eine Anpassung im Backend erforderlich ist. In diesem Fall werden die wichtigen Daten zur Erkennung der Artikel geladen und zwischengespeichert. Beim Versenden der Artikel wird der Status nicht benötigt. Jedoch soll dieser Status den internen Status der Artikel beschreiben. Der Status könnte enthalten, ob ein Artikel gefunden wurde oder nicht. Das hätte den Vorteil, nicht viele Datenbankaufrufe tätigen zu müssen. Es könnten mehrere Artikel gebündelt versendet werden.

Vorgang:

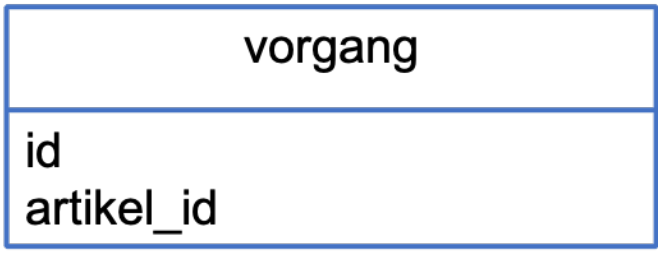

Abbildung 7: Datenbanktabelle "vorgang" [eigene Darstellung]

Wie zu der Tabelle der Artikel erwähnt, kann der Status der Artikel zwischengespeichert werden. Beim Versenden der Artikel bzw. beim Abfertigen der Artikel muss dann nur noch von einer Tabelle eingelesen werden. In der Tabelle „vorgang“ kann ein Attribut die „id“ sein, das sich automatisch erhöht, nachdem ein Artikel gefunden wurde. Hierbei würde es sich um einen Primärschlüssel handeln, da dieser eindeutig auf einen Vorgang verweist. Des Weiteren sollte ein Attribut „artikel_id“ vorhanden sein. Dadurch lässt sich eine Beziehung zur Tabelle „artikel“ bilden. Hierbei müssen die Artikeldaten nicht wieder zwischengespeichert werden, sondern können Anhand der „artikel_id“ auf den bestimmten Artikel verweisen. Ein weiterer Vorteil ist, dass bei der Prüfung der Artikel wieder auf die Tabelle „vorgang“ zugegriffen werden kann. Dadurch wird es dem mobilen Gerät erspart, alle Artikel zu durchforsten. Sollte die Absage gebündelt werden, könnte eine Tabelle mit genau denselben Werten erstellt werden. Hierbei muss selbstverständlich die Namensgebung der Tabellen verändert werden.

4.4 Stakeholder

Bei Stakeholdern handelt es sich um Personen oder Personengruppen, die Interesse an der Umsetzung des Projektes haben oder daran beteiligt sind. Je nach Struktur des Projektes und des Unternehmens oder der Beteiligten sind die Stakeholder immer andere. Stakeholder haben meist unterschiedliche Interessen. Die Stakeholder sollten vor der Anforderung bekannt sein. Außerdem sollten die Stakeholder gemeinsam Anforderungen stellen. Durch Gespräche mit ihnen kann eine Anforderungsanalyse erstellt werden. So sind die Stakeholder dieser App zum einen der Vorstand des Unternehmens und zum anderen die Logistiker und die Entwickler. Der Vorstand des Unternehmens hat ein berechtigtes Interesse an der Umsetzung der App, da diese dem Unternehmen nutzen und den Vorteil haben soll, dass effizienter und schneller gearbeitet wird. Dies führt dazu, dass die Prozesse schneller bearbeitet werden können. Schnellere Bearbeitung eines Prozesses führt dazu, dass mehr Arbeit erledigt werden kann. Auf der anderen Seite werden dadurch die Kosten reduziert, da der Nutzer der Software mehr Arbeitsleistung als zuvor erbringen kann. Das Interesse der Logistiker ist hingegen, dass die neue Software ihnen das Arbeiten erleichtert, d. h. dass sie einfachere und bessere Arbeit leistet als die bestehende Software. Eine Studie der Universität of Warwick zeigte, dass zufriedene Mitarbeiter bis zu 12 Prozent produktiver sind [32]. Somit würde dies wieder zu Vorteilen für das Unternehmen und die Mitarbeiter führen. Mehr oder bessere Arbeit zu leisten kann vom Arbeitgeber anerkannt werden.
Abschließend wären da der oder die Entwickler. Hierbei handelt es sich meist um eine größere Gruppe, bestehend aus IT-Leiter, Programmierer und Software Tester, die verschiedene Interessen haben. Für diese Stakeholder besteht das Interesse in der Umsetzung des Projektes und in der eventuellen Wartung. Außerdem sollte der Entwickler beide Stakeholder gleichermaßen zufrieden stellen. Die Absprache mit Stakeholdern ist sehr wichtig, eventuell kann ohne Absprache und Abklärungen am Ziel vorbeigearbeitet werden. Außerdem kann es vorkommen, dass Probleme oder Fehler nicht erkannt werden. Dabei könnten die Nutzer bzw. Mitarbeiter des bestehenden Prozesses helfen auf Probleme und Fehler aufmerksam zu machen, die nicht offensichtlich sind.

5 Konzeption

Nachdem in der Anforderungsanalyse der bestehende Prozess und die Anforderungen behandelt wurden, widmet sich dieses Kapitel der Konzeption.

5.1 Ziel

Das Ziel ist es, eine stationäre Anwendung als App zu realisieren. Diese App soll eine effizientere Lösung des bisherigen Systems sein. Die App ermöglicht dem Nutzer die angeforderten Artikel einzusehen, sie zu scannen und versandfertig zu machen. Um den Arbeitsprozess zu beschleunigen, ist es von Vorteil, an jedem Ort in der Filiale Zugriff auf die Daten zu haben. Dafür ist ein mobiles Endgerät die perfekte Lösung. Als mobiles Endgerät wird ein Zebra-Android-Gerät verwendet. Das mobile Endgerät hat den Vorteil eines eingebauten Scanners. Zusätzlich dazu bietet Zebra ein Software Development Kit, das die Entwicklung von Android Apps erleichtert. Die App richtet sich ausschließlich an Logistikmitarbeiter des Unternehmens. Um diesen nicht ein komplett neues Produkt zu liefern, ist es die Anforderung des Unternehmens gewesen, die Anwendung so nah wie möglich an der stationären Anwendung zu entwickeln. Aus diesem Grund sollen die Filterkriterien gleichbleiben, sowohl die Pflichtfelder als auch die optimalen Felder. Ein weiterer Vorteil ist es, bestehende Service Aufrufe nicht neu aufsetzen zu müssen. Diese können einfacher angepasst werden und benötigen nur kleine Veränderungen. Die App ist nicht nur darauf ausgelegt, dem Unternehmen zu nutzen, vielmehr soll auch der Nutzer effizienter und einfacher arbeiten können. Ein weiterer Vorteil alle Daten direkt abrufen zu können ist, dass der Nutzer so Wege abkürzen kann. Der Nutzer muss somit nicht ständig zur Station gehen, um Daten zu prüfen. Vor Ort kann geprüft werden, ob es sich um den angeforderten Artikel handelt. Die App muss es dem Nutzer ermöglichen, Etiketten zu scannen. Durch das Scannen von Etiketten lässt sich eindeutig bestimmen, ob es sich um den richtigen Artikel handelt oder nicht. Das führt dazu, dass kein falscher Artikel ausgewählt werden kann.

5.2 Zeit

Für die Arbeit der Bachelorthesis wurde ein Zeitraum von zehn Wochen eingerechnet. Realistische Projektplanung ist für die Realisierung sehr wichtig. Auf der einen Seite können Fehleinschätzungen Kosten verursachen, die zum Scheitern des Projektes führen. Auf der anderen Seite können Kunden vom Projekt abspringen, was eventuell auch mit Kosten verbunden ist. Außerdem wird nicht jede Stunde der Arbeitszeit genutzt, um die App zu entwickeln. Zehn Wochen und zwei Arbeitstagen die Woche mit acht Stunden pro Tag ergeben 20 Arbeitstage mit 160 Arbeitsstunden. Die Velocity lässt sich allerdings erst ermitteln, nachdem das Projekt abgeschlossen ist. Bei der Velocity handelt es sich um die Zeit, die tatsächlich dafür genutzt wird, an dem Projekt zu arbeiten. Dieser Wert wird in Prozent angegeben. Wie schon erwähnt, ist es bei der Anforderungsschätzung wichtig, realistisch zu bleiben. Die Zeiten für die Recherche und Einarbeitung in das Thema sollten ebenfalls eingeplant werden.

5.3 Vorgehensmodell

Vorgehensmodelle sollen das Erstellen von Software erleichtern. Die Vorgehensmodelle unterscheiden sich lediglich nach der Herangehensweise an die Umsetzung. Bei komplexer Software kann es ganz schnell zu Problemen und Fehlern kommen, wenn versucht wird alles auf einmal zu implementieren. Somit wird schrittweise implementiert, integriert und getestet. Bei größeren Teams ist es sinnvoll, sich an einem bestimmten Modell zu orientieren. Um nur einige zu nennen, kämen z.B. das Wasserfallmodell und die Agile Methode in Frage. Beim Wasserfallmodell handelt es sich um eines der ältesten Modelle. Hierbei gibt es aufeinanderfolgende Projektphasen. Somit baut jede Phase auf die vorherige auf und die Ergebnisse der vorherigen Phase werden für die nachfolgende benötigt. Jede Phase benötigt einen Anfangs- und einen Endpunkt. Es muss daher definiert werden, wo angefangen und wo beendet werden soll. Ein Vorteil dieses Modells ist, dass durch dessen Struktur Fehler vermieden werden können. Außerdem kann anhand der Projektphasen der Fortschritt nachvollzogen werden. Somit bestehen ein klares Verständnis und ein Überblick über abgeschlossene und noch zu erfüllende Schritte. Des Weiteren führt das Modell zu umfangreichen technischen Dokumentationen. Diese können dabei helfen, das Projekt nochmals zu testen oder neuen Entwicklern das Einarbeiten in das Projekt zu erleichtern.
Bei der Agilen Methode handelt es sich um ein Modell, bei dem die Absprache mit dem Kunden im Vordergrund steht, daher ist es wichtig, offen für Veränderungen zu sein. Auch wird der Fokus bei diesem Modell darauf gelegt, dass der Code funktioniert und immer lauffähig ist. Somit sind bei dieser Methode funktionierende Programme wichtiger als eine ausführliche Dokumentation.

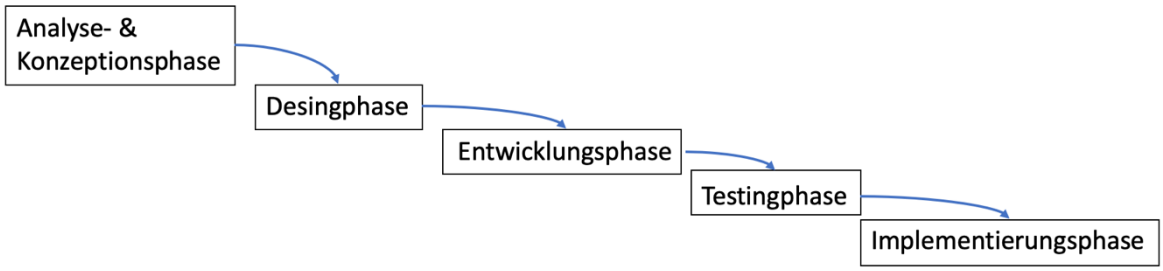

Abbildung 8: Wasserfallmodell [30]

Der erste Schritt im Wasserfallmodell ist die Analyse- und Konzeptionsphase. Hierbei wird die Anforderung an die Software gestellt. Sollten Anforderungen vom Kunden oder Auftraggeber gestellt werden, sollten diese zusammengefasst werden, sodass ein Konsens über die Punkte geschaffen wird. Es ist wichtig, die Analyse der Anforderungen genau zu erarbeiten, um im späteren Verlauf nicht auf Probleme zu stoßen, die den Projektverlauf behindern und eventuell fatale Folgen haben können. Außerdem werden hier grobe Bedingungen für die technische Realisierung aufgeführt.
Als nächstes folgt die Designphase, in der beschrieben wird, welche technischen Attribute in der Entwicklung benötigt werden. Hierfür werden die Spezifikationen der Datenbank, Server, Schnittstellen und Plattformen entschieden. Bei der Designphase werden spezielle Fachkenntnisse gefordert, daher ist es wichtig, dass die Spezifikationen von jemandem vom Fach zusammengefasst werden. Die darauffolgende Entwicklungsphase ist der Schritt der Realisierung. Jetzt wird mit der Entwicklung der Software begonnen. Schrittweise werden die Anforderungen in Codes umgesetzt. Um strukturell an die Arbeit gehen zu können, können Ticketsysteme sehr hilfreich sein. Jira ist eine webbasierte Projektmanagementsoftware, die das Verwalten von Tickets erleichtert. Bei Tickets handelt es sich um Aufgaben. Zum Beispiel kann eine Anforderung ein Ticket widerspiegeln. Jedoch sollten Tickets viel weiter unterteilt werden, sodass das schnelle und effiziente Abarbeiten der Tickets gewährleistet wird. In Jira gibt es Spalten wie TODO, Waiting, In Progress, Done und Closed. Sobald ein Ticket erstellt wird, gelangt dieses in die TODO-Spalte. Wenn der Entwickler ein Ticket bearbeitet, zieht er das Ticket in die Spalte „In Progress“. Sollten Probleme auftreten oder der Vorgang nicht weiterbearbeitet werden können, dann wir das Ticket in „Waiting“ gezogen. Nachdem der Entwickler die Aufgabe gelöst hat, sollte das Ticket dann in die Spalte „Done“ verschoben werden. Hier kann dann von einem anderen Entwickler, Tester oder Projektleiter die Lösung begutachtet und getestet werden. Nachdem ein Ticket erfolgreich getested worden ist, wird dieses dann in die Spalte „Closed“ geschoben. Sollte die Lösung falsch sein oder Fehler aufweisen, sollte das Ticket dem Entwickler zurückgegeben werden. Eine weitere Variante des Projektmanagements ist Trello, die ebenfalls von Atlassian stammt, jedoch nur eine Lite-Version von Jira ist. Der nächste Schritt in dem Wasserfallmodell ist die Testingphase. Hierbei wird die erste Version der Software getestet. Sie wird anhand von echten Testfällen begutachtet, um mögliche Fehler oder Bugs herauszufiltern.

Die Tests sollten am besten von mehreren durchgeführt werden. Dabei sollte möglichst eine Testumgebung geschaffen werden, die der produktiven Umgebung gleichkommt.
Abschließend kommt die Implementierungsphase. Nachdem alle Aufgaben erledigt worden sind und die Software getestet und abgenommen worden ist, kann mit der Erstellung der Software Live begonnen werden. Hierbei kann es zu verschiedenen Umsetzungen kommen. Bei der einen Software müssten eventuell nur Datenbanken und Schnittstellen angebunden, bei einer anderen müsste die Software in das gesamte, bereits existierende System implementiert werden. Diese Punkte sind zur Einhaltung des Wasserfallmodells zu beachten.

6 Beschreibung der App

Nachdem die Anforderungen an die App gestellt worden sind, wird hier die App beschrieben. Es werden jeder einzelne Bereich und selbstverständlich ebenfalls auch die Zusammenhänge erklärt. Die App wird in drei große Bereiche unterteilt. Der Filterbereich wäre der erste. Anschließend folgt die Liste der angeforderten Artikel und abschließend eine Liste der abzuarbeitenden Artikel.

Für kleine Skizzen nutze ich die Webanwendung „Sketch.io“, welche sich als sehr nützlich und effektiv erwiesen hat. Sie ist sehr effizient und schnell zu erlernen.

Bei den Skizzen handelt es sich lediglich um Ideen, die eventuell bei der Realisierung etwas anders sein können.

6.1 Filter

Abbildung 9: Skizzierung des Filters [eigene Darstellung]

Eine Skizze des Filters ist in Abb. 9 zu sehen. Wie schon bei der Anforderungsanalyse erwähnt, wurden hier die wichtigen Filterkriterien hinzugefügt.

Da es mehrere Benutzeroberflächen geben wird, ist es von Vorteil, dass der Benutzer weiß, auf welcher er sich zurzeit befindet. Darum soll ganz oben der Name der Oberfläche zu sehen sein. In der darauffolgenden Zeile sollen zwei EditText-Felder vorhanden sein, in die der Benutzer die erforderlichen Werte für den Bereich und das Ressort eingeben kann. Ebenfalls könnte hier ein Spinner verwendet werden. Dies ist aber nur zu empfehlen, wenn es sich um geringe Datenmengen handelt. Bei großen Datenmengen wäre die Suche nach dem richtigen Objekt eine Qual. Die Daten dafür statisch zu generieren oder einen Aufruf einer Datenbank würde die Sache nur verkomplizieren. Des Weiteren soll ein Spinner für das Erstellungsdatum vorhanden sein. Ein Spinner ist in diesem Fall von Vorteil, da nur ein bestimmtes Datum gefordert wird. Eine Liste dieser Daten lässt sich einfach ermitteln und erfordert auch keinen Datenbank-Aufruf.
In der nächsten Zeile sollen RadioButtons den Status der Aufforderungen übernehmen. Es soll nur ein Status zur selben Zeit auswählbar sein. Abschließend gibt es zwei Buttons. Einer der Buttons soll alle Werte zurücksetzen, in unserem Fall der linke Button. Alle Werte sollen auf die Standardwerte zurückgesetzt werden. Der rechte Button soll anhand der angegebenen Daten eine Suche nach den speziellen Artikeln auslösen.

6.2 Liste der angeforderten Artikel

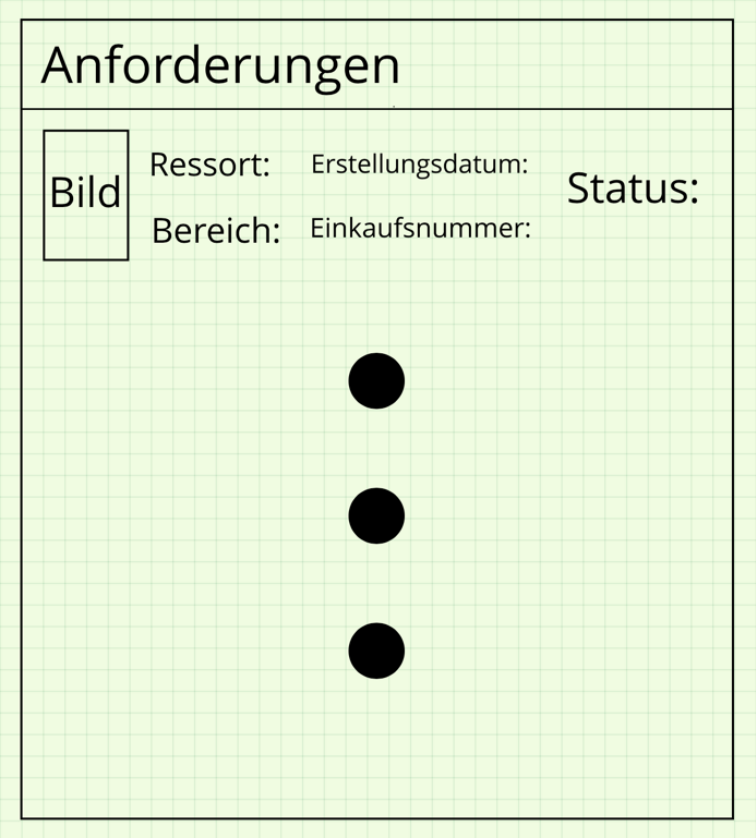

Abbildung 10: Skizzierung der Liste der angeforderten Artikel [eigene Darstellung]

In der oben abgebildeten Skizze sieht man, wie die Liste der angeforderten Artikel aussehen soll. Wie hinsichtlich der ersten Oberfläche schon erwähnt, soll auch hier der Benutzer wissen wo er sich befindet.

Ein Artikel soll ein Objekt in der Liste darstellen. Zu jedem Artikel kommt ein kleines Foto. Dazu werden noch Bereich, Ressort, die Einkaufsnummer, das Erstellungsdatum und der Status der Anforderung angezeigt. In dieser Ansicht sollen nur die wichtigsten Daten zu sehen sein. Hier wird die Liste noch nicht abgearbeitet, somit ist eine detaillierte Ansicht an dieser Stelle überflüssig. Außerdem können so schnell mehrere Artikel angesehen werden. Abschließend soll noch der Status angezeigt werde. Dieser ist wichtig, weil es Artikel mit dem Dringlichkeitsstatus gibt. Diese sollen zuerst bearbeitet werden.

6.3 Liste der zu bearbeitenden Artikel

Abbildung 11: Skizzierung der persönlichen Liste [eigene Darstellung]

Wie auf der oben abgebildeten Skizze zu sehen ist, habe ich nur einige wenige Punkte hinzugefügt. Zuerst kommt wie immer der Name des Prozesses. Danach folgt dieselbe Liste, in der ich lediglich zwei weitere Details hinzugefügt habe, und zwar die Größe und die Saison. Zur eindeutigen Identifizierung des Artikels werden diese Punkte benötigt. Außerdem soll in dieser Oberfläche der Scanner für die Artikel aktiviert werden. Wie in der Anforderungsanalyse schon besprochen, ist das eine der wichtigsten Eigenschaften der App. Der Vorteil Gebrauch vom eingebauten Scanner zu machen verhindert die Auswahl falscher Artikel. Durch das Scannen eines Artikels können in der App die Daten verglichen und dem Nutzer kann eine Mitteilung gegeben werden.

6.4 Ansicht der Artikeldetails

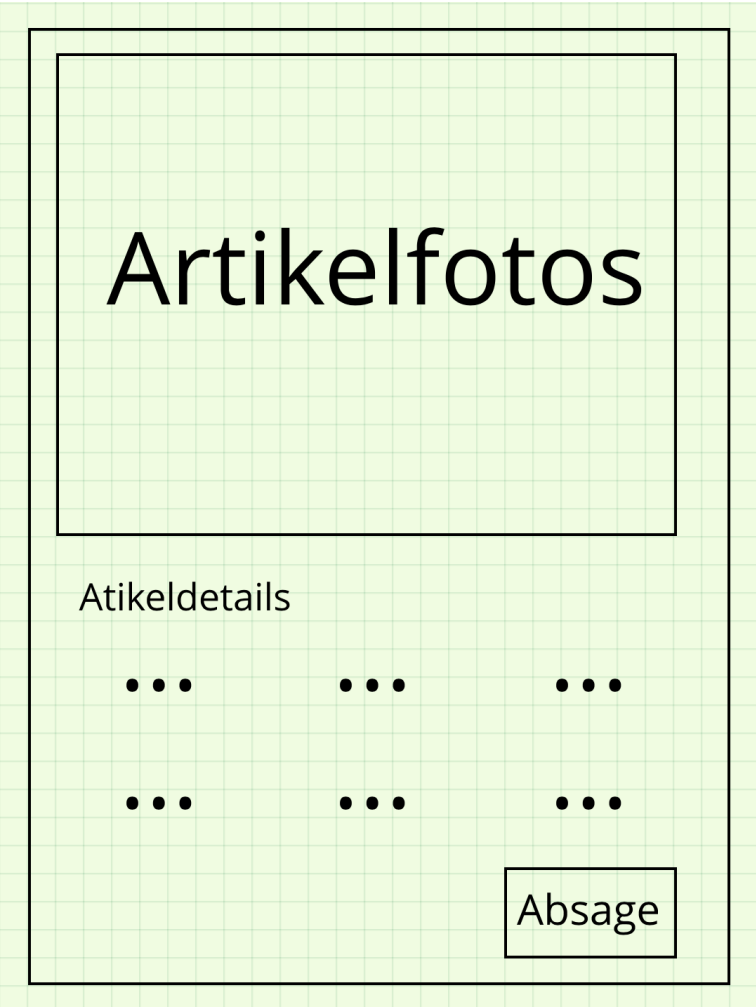

Abbildung 12: Skizzierung der Artikeldetails-Ansicht [eigene Darstellung]

Nachdem ein Artikel aus der Liste ausgewählt oder angetippt worden ist, sollte eine größere, detaillierte Ansicht der Artikeldaten und des Artikelfotos angezeigt werden. So kann noch genauer eingesehen werden, um welchen Artikel es sich handelt. Des Weiteren soll die Ansicht die Absage dieses Artikels ermöglichen. Dies würde dem Nutzer viel Zeit ersparen. Der wesentliche Vorteil davon wäre, dass der Nutzer die Daten nicht wieder per Hand eintragen muss. Da die Daten schon vorhanden sind, können diese auch sofort verwendet werden. Nachdem auf den Button „Absage“ gedrückt worden ist, sollte eine weitere Ansicht erscheinen, ein sogenannter Dialog.

6.5 Ansicht der Absage

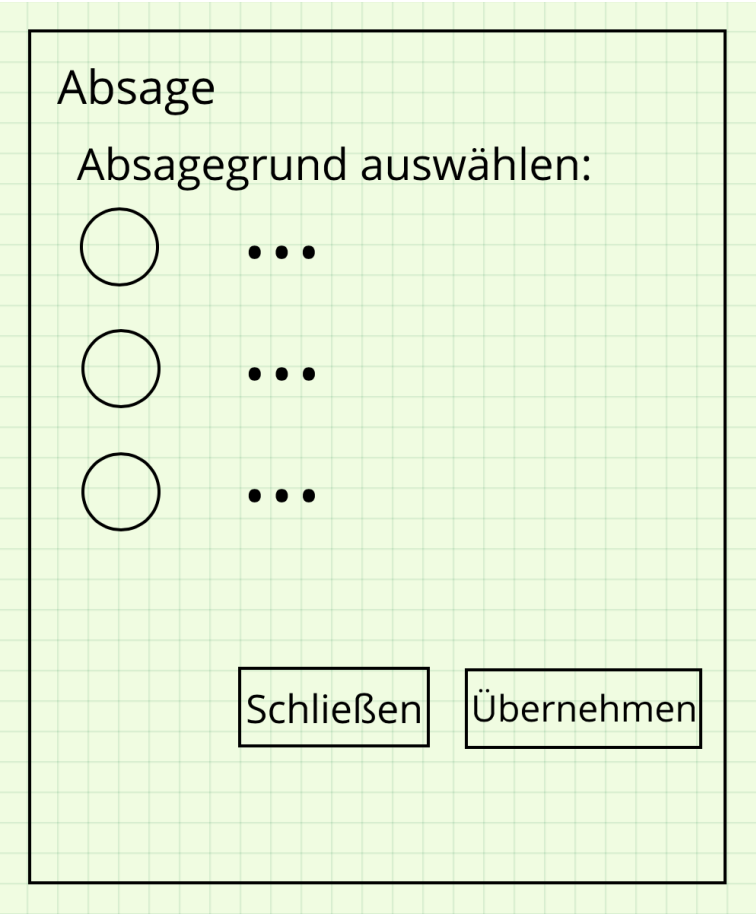

Abbildung 13: Skizzierung der Absage-Ansicht [eigene Darstellung]

Bei einem Dialog handelt es sich um eine kleine Ansicht, die den Nutzer dazu auffordert, eine Entscheidung zu treffen. In Dialogen werden keine komplexen Aufgaben getätigt, vielmehr nur einfache Abfragen. Ein weiterer Zweck wäre es, Informationen zu erfragen. Als Beispiel könnte ein Dialog vor einem Wechsel der Ansicht angezeigt werden, um fehlerhafte Daten zu korrigieren. In der Ansicht sollen Radio Buttons angezeigt werden, von denen der Nutzer einen auswählen kann. Die Radio Buttons sollen den Grund der Absage widerspiegeln. Mit dem Button „schließen“ kann der Prozess geschlossen bzw. abgebrochen werden. Der Button „übernehmen“ soll die Eingabe bzw. die Auswahl übernehmen. Diese kann dann weiterverarbeitet werden.

6.6 Ansicht zur Versendung der Artikel

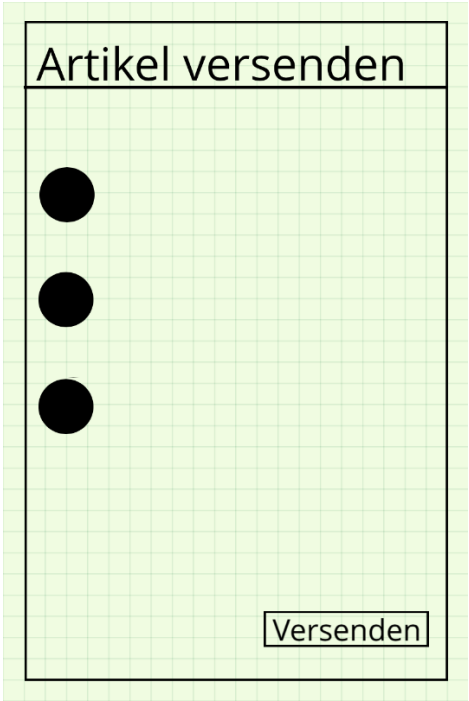

Abbildung 14: Skizzierung der Versendung-Ansicht [eigene Darstellung]

Bei der Versendung der Artikel wird erneut geprüft, ob es sich um den Artikel handelt, der angefordert wurde. Hierbei kann verhindert werden, dass versehentlich Artikel mitgenommen werden. Ebenfalls könnte hier eine maximale Obergrenze an zu scannenden Artikeln gesetzt werden. Derzeit läuft es beim bestehenden Prozess oder bei der Erstellung des Lieferscheins derzeit so ab, dass nach Gefühl entschieden wird, wie viel Artikel lieferfertig gemacht werden. So könnte eine Obergrenze für Hänge- und Liegeware gesetzt werden, um eine Einheit zu bilden. Außerdem könnte dadurch verhindert werden, dass neue Nutzer bzw. Mitarbeiter versehentlich mehr Artikel scannen und versenden als möglich, was zu Verlusten von Artikel führen könnte. Bei Beschädigung der Pakete oder Folien (für die Hängeware) könnten eventuell Artikel vertauscht oder ebenfalls beschädigt werden.

6.7 Zusatz

Wie vorher schon kurz erwähnt, habe ich zwei Listen vorgesehen, und zwar aus den folgenden Gründen. Die erste Liste soll einen groben Überblick über die Anzahl der angeforderten Artikel geben. Außerdem sollen nicht zu viele Daten gespeichert werden müssen, denn je weniger Daten im Hintergrund gespeichert werden, desto höher ist die Performance der App. Aus diesem Grund soll eine weitere Liste generiert werden, die in der Anzahl nicht alle angeforderten Artikel enthalten soll, sondern nur einen Bruchteil davon. In der Liste sollen dann alle wichtigen Punkte zur Identifizierung des Artikels angezeigt werden. Des Weiteren sollen in der personifizierten Liste die Artikel gescannt werden können. Außerdem sollen Artikel, die nicht vorhanden sind, abgesagt werden können. Abschließend sollen die Artikel versendet werden. Wie man sehen kann, steckt sehr viel in der letzten Oberfläche, was ein weiterer Grund ist die Listen zu trennen. Dazu kommt noch, dass so Artikel, die nicht gefordert oder überhaupt nicht vorhanden sind, versehentlich abgeschickt werden.

6.8 Design

Zum Aufbau der App kommt zusätzlich das Design. Die Farben werden auf Basis der Grundfarben des Unternehmens gewählt, nämlich die Farben marineblau, marineblau dunkel und rot. In der App werden die Farben in die XML-File gesetzt, und zwar ColorPrimary, PrimaryDark und Accent. Für ColorPrimary wird marineblau eingesetzt, welches den Wert „#002045“ hat, für ColorPrimaryDark setzen wir marineblau dunkel ein, der Wert dafür ist „#00001f“ und zu guter Letzt wird ColorAccent mit dem Wert „#61527“ gewählt, was die Farbe Rot ergibt.

7 Softwarearchitektur

Da wir bislang die Anforderungsanalyse und die Beschreibung der App behandelt haben, beschäftigen wir und in diesem Kapitel mit der Softwarearchitektur. Bei der Softwarearchitektur handelt es sich um das Zusammenspiel von Komponenten im System. Dabei geht es um die strukturelle Ordnung und die Hierarchie der Komponenten. Es wird der generelle Aufbau erklärt.

7.1 System

Die Idee ist es, das System so aufzubauen, dass alle Komponenten reibungslos miteinander arbeiten können. Da die Komponenten ein Bestandteil des Systems sind, sollten diese gut durchdacht werden. Sie sollen ihrer Umgebung viele Dienste bieten und über eine gut definierte Schnittstelle genutzt werden. Gute Komponenten definieren sich dadurch, dass sie einfach auszutauschen sind. Dennoch sollten sie sehr effektiv sein und reibungslos mit den Schnittstellen zusammenarbeiten. Außerdem sollten sie wieder verwendbar sein. Eine Komponente hat eine oder mehrere Schnittstellen. Der Begriff Schnittstelle bedeutet jedoch vielmehr eine Verbindung als eine Trennung.

Um ausgezeichnete und so gut wie fehlerfreie Komponenten und Schnittstellen zu entwickeln, wird viel Erfahrung benötigt. Hier kommt das allbekannte Sprichwort ins Spiel: „Aus Fehlern wird man klug“. In diesem Sinne sollte man einfach so viele Szenarien wie möglich ausprobieren und sich intensiv Gedanken darüber machen.

7.2 Architekturmodell

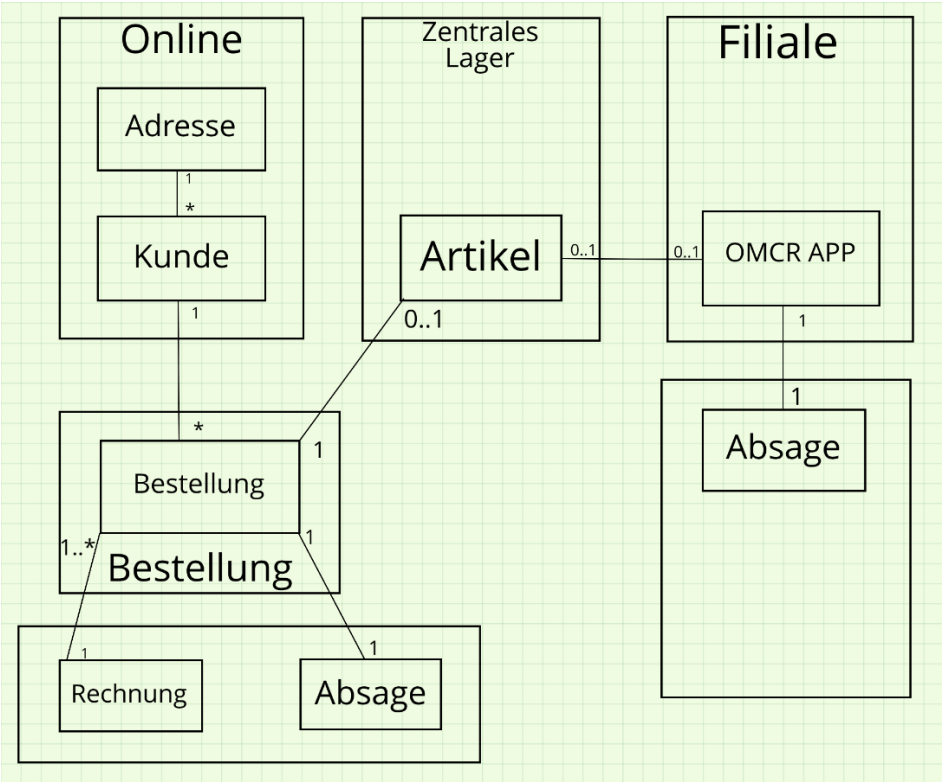

Abbildung 15: Architekturmodell [eigene Darstellung]

In der Abbildung 15 ist das Architekturmodell des Prozesses zu sehen. Der Ablauf ist relativ einfach. Ein Kunde gibt eine Bestellung ab. Die Bestellung wird vom Zentralen Lager aufgenommen und weiterverarbeitet. Sollte der Artikel nicht im Lager zu finden sein, wird er zur App weitergeleitet. Der Nutzer der App sucht den Artikel zunächst im Filiallager, sollte er auch dort nicht vorrätig sein, wird überprüft, ob er auf der Verkaufsfläche vorhanden ist. Wenn der Artikel zwischenzeitlich verkauft wurde, kann der Nutzer den angeforderten Artikel absagen. Daraufhin schickt das Zentrale Lager wiederum die Anfrage an eine andere Filiale.

7.3 Aufbau der App

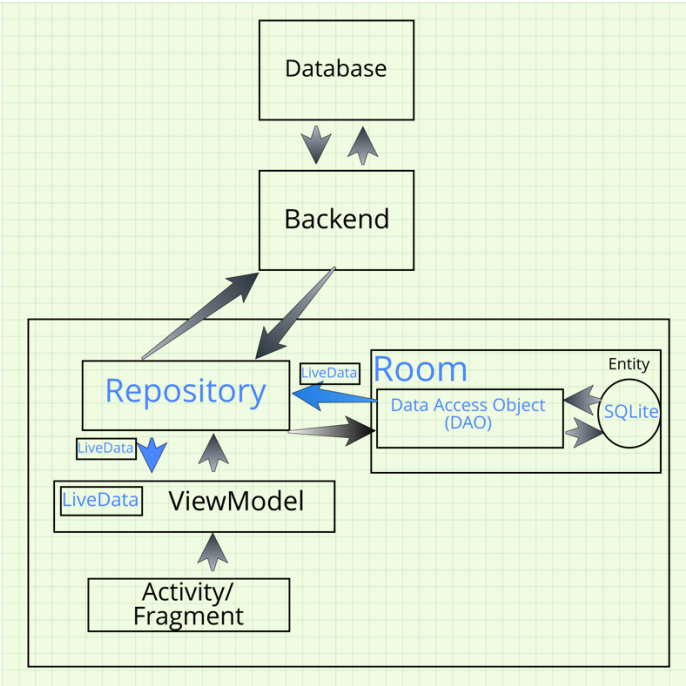

Abbildung 16: Skizze des Aufbaus der App [eigene Darstellung]

Aus der Abb. 16 ist der generelle Aufbau der App zu entnehmen und auch wie diese mit dem Backend zusammenarbeitet. Begriffe wie ViewModel, Room, Repository und LiveData werden in den folgenden Unterkapiteln genauer erklärt. Im Grunde wird die App so aufgebaut, dass sie Daten, die entweder zur Anzeige oder zur Bearbeitung gedacht sind, vom Backend lädt. Diese Daten werden dann lokal im Handy gespeichert, um bei Netzverlust oder anderen Störungen nicht auf das Backend angewiesen zu sein. Damit wird gewährleistet, dass sowohl die Daten als auch der Fortschritt nicht verloren gehen.

7.4 Architecture Components

Nachdem erklärt wurde, was Softwarearchitektur und Komponenten sind und wie der Aufbau der App sein wird, wird im Folgenden darauf eingegangen, welche Android Architecture Components in der App benutzt werden. Android Architecture Components sind, wie der Name schon hergibt, eine Ansammlung von Bibliotheken und Komponenten, die einem Entwickler das Schreiben von Apps erleichtern sollen. Dabei soll die App wartbarer gemacht werden und somit sollen Änderungen einfacher eingepflegt werden können.

7.4.1 ViewModel

Bei der ViewModel Komponente handelt es sich um eine Klasse, die so konzipiert ist, dass sie die Daten der Benutzeroberfläche abspeichert und verwaltet. Außerdem ist die Klasse bewusst über den Lebenszyklus. Der Lebenszyklus einer Klasse ist der Prozess von der Erstellung bis hin zur Zerstörung der Klasse. Vielen Android-Entwicklern ist bekannt, dass nach Änderung der Einstellungen wie das Rotieren des Displays die ganze Benutzeroberfläche zerstört wird und somit Daten verloren gehen. Die ViewModel-Klasse erlaubt es, dieses zu überstehen.

7.4.2 Data Binding

Bei der Data Binding Library handelt es sich um eine Ansammlung von Benutzeroberflächen-Komponenten. Die Komponenten erleichtern die Implementierung von Objekten aus der Benutzeroberfläche. Vielmehr wird nur das ViewModel im Layout deklariert, somit wird im ViewModel nur noch auf das dafür erstellte Objekt zugegriffen. Außerdem lassen sich aus dem ViewModel auch Werte in der Benutzeroberfläche über einen simplen Aufruf ändern. Es handelt sich hierbei um das „Two-way data binding“. Hierbei wird im Layout lediglich das Gleichheitszeichen „=“ hinzugefügt.

```
<layout ...>

  <data>
    <variable
      name:"viewmodel"
      type:"com.example.MyViewModel" />
  </data>

  <ConstraintLayout
    ...>
    <EditText
      android:text"@={viewmodel.text}"
      .../>
```

Ein einfaches Data Binding benötigt lediglich das @{}, wobei in den Klammern die Referenz der Variablen und das Objekt stehen. Dadurch wird dem ViewModel der Wert übergeben. Wie im Quellcode zu sehen ist, wird das ViewModel mit dem Variablennamen „viewmodel“ deklariert. Anschließend kann jedes Android Widget mit den Variablen arbeiten. In diesem Fall wird im EditText Widget das Attribut „text“ verwendet. Durch das Gleichheitszeichen im Widget „text“ kann das Viewmodel nun auch Werte zurück zum Layout schicken.

7.4.3 LiveData

Bei der LiveData Komponente handelt es sich um eine Observable Klasse, die das Halten von Werten ermöglicht. Das Besondere an dieser Klasse ist, dass sie keine Standard Observable Klasse ist. Viel mehr arbeitet sie bewusst mit dem Lebenszyklus anderer Komponenten. Somit werden Daten nur aktualisiert, wenn die Komponente auch wirklich aktiv ist.

```
public class MyViewModel extends ViewModel {

    private MutableLiveData<String> text = new MutableLiveData<>();
```

Mit einer simplen Implementation eines LiveData Objekts kann nun der Text aus der View eingelesen und verändert werden. Sollte sich der Wert „text“ verändern, wird die Änderung sofort auf der Benutzeroberfläche angezeigt.

7.4.4 Room

Die Room Persistence Library vereinfacht die Zusammenarbeit mit SQLiteDatase. Hierbei können nicht nur gesammelte Daten zwischengespeichert werden, sondern auch Daten, die von außerhalb geladen wurden. Ein wichtiger Punkt ist, dass der Nutzer auf gespeicherte Daten zugreifen kann, ohne eine Internetverbindung zu haben, da die Datenbank lokal auf dem mobilen Gerät installiert wird.

7.4.5 Repository

Repositories werden als Controller-Klasse für das Zusammenarbeiten mit Datenbanken geschrieben. In einer Repository ist meistens ein Webservice und das dazu gehörende DAO (Data Access Object, zu Deutsch Datenzugriffsobjekt) zu finden. Somit soll das Repository für das ViewModel mit der Datenbank arbeiten.

7.5 Entwurfsmuster

Bei den Android Architecture Components hält sich Android eine endgültige Entscheidung offen, welches Entwurfsmuster das Beste ist. Somit ist es jedem Entwickler selbst überlassen, das für ihn oder seine App passende Entwurfsmuster zu finden. Ob nun MVP (Model View Presenter), MVC (Model View Controller) oder MVVM (Model View ViewModel), für eines dieser Modelle muss man sich letzten Endes selbst entscheiden. Für diese App wurde das MVVM gewählt. Das Entwurfsmuster dient dazu, die Darstellung und Logik der Benutzeroberfläche zu trennen.

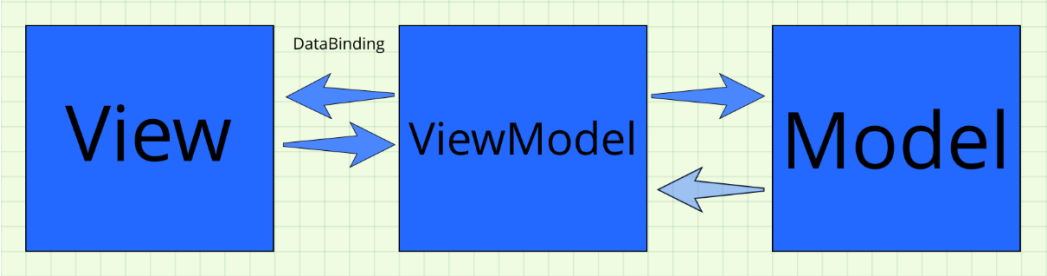

Abbildung 17: Skizze des Entwurfsmusters MVVM [eigene Darstellung]

In der Abbildung 17 ist die Verbindung dargestellt. Bei der View wird alles was dem Benutzer angezeigt wird verwaltet. Ebenfalls dient sie dazu, die Benutzereingaben weiterzuleiten. Außerdem ist die View direkt verbunden mit dem ViewModel. Beim ViewModel wird die Logik der Präsentation verwaltet. Hierbei werden der View öffentliche Methoden/Funktionen und Dienste wie Service-Aufrufe zur Verfügung gestellt. Außer der Verbindung zur View dient sie als Schnittstelle zwischen der View und Model. Das ViewModel darf keine Attribute der View haben. Des Weiteren ist das ViewModel dazu da, um Ereignisse der View zu verwalten, wie zum Beispiel Botten Klicks. Zu guter Letzt ist es das Model, welches als Datenzugriffsschicht dient. Es enthält Daten, die dem Benutzer angezeigt werden. Außerdem kann das Model auch Daten abspeichern, die von einem Nutzer in die View eingegeben wurden.

7.6 Dagger 2

Bei Dagger handelt es sich um eine Automatisierung von dependency injection. Der Vorteil davon ist, dass einmalig Module deklariert und anschließend ganz einfach anhand der jeweiligen Annotationen verwendet werden können. Somit lässt sich eine Klasse für alle Webservices schreiben, die wiederum nach dem Build ganz einfach verwendet werden, können. Zum einen gibt es die fundamentalen Dagger-Klassen. Diese Klassen vereinfachen den Arbeitsprozess. Da wären die Dagger Application, die Dagger Appcompat Activity und die Dagger Fragment Classes. Diese sind neu in Dagger 2 und bieten viele Möglichkeiten. Sie wurden speziell für Dagger entwickelt, um diese zusammen in Android zu nutzen. Als nächstes wären da die Costum Scopes und Annotationen. Bei Scopes geht es darum, Singleton- Objekte, die im Speicher hinterlegt sind und nicht zwangsläufig immer bereitstehen müssen, zu verwalten. Bei einer großen App sollte dies gut verwaltet werden, da ansonsten die Performance darunter leiden kann.

Außerdem gibt es drei Arten von Injections, die Constructor Injection, die Field Injection und die Method Injection. Die ViewModel Injection kommt sehr gelegen. Da die App mit dem Entwurfsmuster MVVM geschrieben wird, kann dadurch vieles vereinfacht werden. Weiterhin gibt es noch das Repository Pattern, welches zuvor schon erklärt wurde, und abschließend Retrofit 2 mit Dagger.

7.7 Retrofit 2

Retrofit ist ein typsicherer http-Client für Android und Java. Dieser wurde von Square entwickelt. Retrofit erleichtert die Verwendung von JSON- und XML-Daten, die in POJO's (Plain old Java Objekts) analysiert werden. Auf die Implementierung wird im folgenden Kapitel eingegangen. Der Entwickler muss dadurch weniger fehleranfälligen Code schreiben, um einen Aufruf zu starten.

7.8 Scanner

Die App wird nativ für ein Zebra-Gerät entwickelt. Es gibt zwei Möglichkeiten den Scanner im mobilen Gerät anzusteuern, zum einen das EMDK (Enterprise Mobile Development Kit) von Zebra und zum anderen die DataWedge App, die sich ebenfalls auf dem Gerät befindet, zu nutzen. Beide Varianten geben die Möglichkeit zu entscheiden welche Arten von Barcodes zu scannen sind, wie lang diese sein sollen und wie diese Daten zur App gelangen. Das EMDK wird in die App integriert und muss dementsprechend selbst konfiguriert werden. Jede Einstellung muss in den Code implementiert werden. Außerdem ist das EMDK sehr rechenlastig und kann bei falscher Implementierung zu Performance-Verlusten führen. Hierbei muss jeder Schritt implementiert werden. Dabei muss berücksichtigt werden in welchem Zustand sich der Scanner beim Drücken und loslassen des Hardware Buttons befindet. Außerdem muss auch berücksichtigt werden, in welchem Zustand sich der Scanner befindet und wann der Scanner seinen Zustand ändert. Die DataWedge API hingegen bietet die Eigenschaften ohne eine Zeile „Code" zu schreiben. Hierbei kann für die App ein Profil erstellt werden, über das gesteuert wird, welche Barcodes gescannt werden oder wie lang diese sein sollen. Außerdem bietet es die Möglichkeit in der DataWedge App jedes Profil einzusehen und gegebenenfalls zu ändern. Es besteht die Möglichkeit, im Code die Profile für die App zu erstellen, dadurch kann nach Installation der App jedes Profil direkt erstellt werden. Des Weiteren ist es einfacher, mit der DataWedge den Scanner zu wechseln oder ihn nur für eine bestimmte App zu wechseln. Ein Nachteil der DataWedge API wäre, dass der Input der gescannten Daten nur durch ein Broadcast Receiver einzuholen ist, was bedeutet, dass die App beobachtet was ihm zurückgegeben wird. Alles in allem bietet es sich an, die DataWedge API zu nutzen, da die Vorteile überwiegen und weniger Fehler bei der Implementation gemacht werden können.

8 Realisierung

Sowohl die Anforderungsanalyse als auch die Beschreibung der App und die Softwarearchitektur wurde behandelt. Das bringt uns zu dem Punkt der Realisierung. Im folgenden Kapitel wird näher auf die Realisierung der App eingegangen.

8.1 Entwicklungsumgebung

Android Studio ist die offizielle Entwicklungsumgebung von Google. Sie wurde in Zusammenarbeit mit JetBrains entwickelt. Es besteht die Möglichkeit, andere Entwicklungsumgebungen zu nutzen, wie zum Beispiel Eclipse oder IntelliJ. Android Studio hat den Vorteil, dass es ausschließlich für das Entwickeln von Android Apps genutzt wird. Außerdem bietet es viele Möglichkeiten zusätzliche Plug-Ins zu installieren. Des Weiteren steht Android Studio für MacOS, Windows, Linux und Chrome OS zur Verfügung.

8.2 SDK

SDK steht für Software Development Kit, wobei es sich um eine Ansammlung von Standardbibliotheken und Entwicklerwerkzeugen handelt, die zur Entwicklung benötigt werden. Bei der Android SDK gibt es eine Unterteilung in zwei Stufen. Zunächst wäre da die SDK- Plattform. Dazu gehören standardmäßig die Android-Plattform und das dazugehörige API Level. Die aktuelle Android-Version ist Android 10, auch als Android Q bekannt, und die aktuelle API Level Version ist 29. Des Weiteren gibt es das SDK-Tool, welches die Entwicklerwerkzeuge liefert, dazu gehört zum Beispiel der Intelligent Code Editor, welcher nach Eingabe von wenigen Buchstaben schon Wortvorschläge macht.

Es stellt sich an dieser Stelle die Frage, welche Version und wieso diese Version für die App genutzt werden sollte. Es gibt drei SDK-Versionen in der App. Die erste wäre die „compileSDK-Version“. Die compile SDK-Version wird von der Entwicklungsumgebung genutzt, um den Code als „.apk Datei“ zu schreiben. Diese kann dann genutzt werden, um die App in Android Store zu veröffentlichen. Als nächstes gibt es die „targetSDKVersion“. Diese Version legt fest auf welcher Version die App laufen soll. Außerdem sollten hier die V compileSDKVersion und die „targetSDKVersion“ übereinstimmen. Zu guter Letzt möchte ich die „minSDKVersion“ erwähnen, wobei es sich um die Version handelt, die mindestens benötigt wird, um die App zum Laufen zu bringen. Wichtig ist auch, dass eventuell einige Funktionen nicht unterstützt werden, sollte eine niedrig angesetzte Version ausgewählt werden. Wenn man nun eine App entwickeln möchte, die von so vielen Personen wie möglich benutzt werden soll, kann die min SDK-Version weit heruntergesetzt werden.

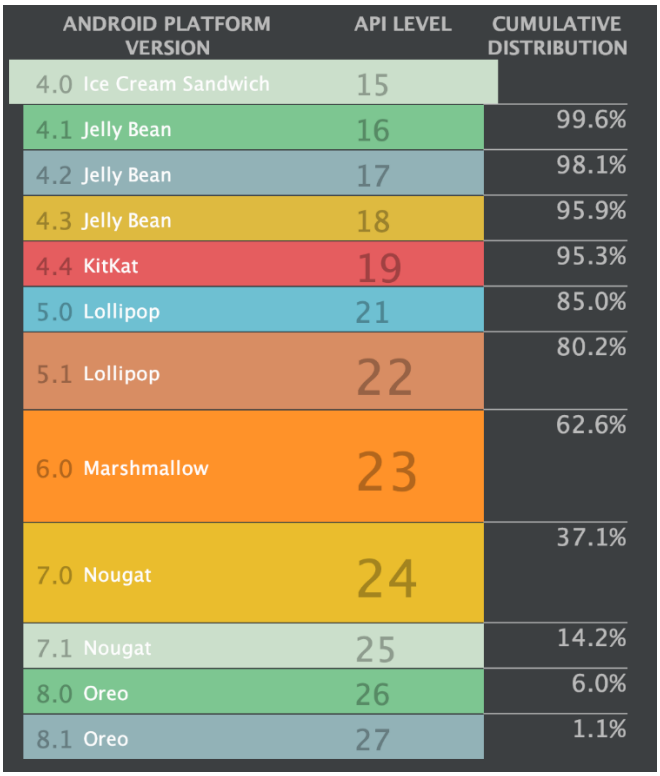

Abbildung 18: Android API Statistik [1]

Um alle Android-Nutzer anzusprechen, kann die Version 4.0 API Level 15 ausgewählt werden, welche den Vorteil hat, dass mehr Menschen die App sehen, benutzen und bewerten können. Folglich führt dies zu mehr Nutzern. In unserem Fall wird die App nicht veröffentlicht und soll auch nicht alle möglichen mobilen Geräte ansprechen da es eine App wird, die intern im Unternehmen benutzt werden soll. Somit wird mehr Wert auf die Funktionen der App als auf die Erreichbarkeit von Nutzern gelegt. Als target SDK-Version kann somit immer die neuste Version genommen werden. Dieselbe Version wird auch für die compile SDK-Version genommen, um nicht nur Fehlern vorzubeugen, sondern auch alle Funktionen und Eigenschaften beim Entwickeln zu nutzen. Bei der min SDK-Version sollte beachtet werden, welche Eigenschaften und Funktionen die App mitbringt und welche man benötigt, um diese zu entwickeln. Da es sich hierbei um eine lokale App handelt und der Fokus auf Funktionen gelegt wird, nehmen wir als min SDK-Version die 27.

[1] *Bei der Abbildung 18 handelt es sich um Statistiken, die Android Studio bereitstellt. Die Statistik wurde in der Zeit, in der die Bachelorarbeit geschrieben wurde, aufgerufen.*

8.3 Version Control

Android Studio bietet eine Schnittstelle zu anderen Versionsverwaltungssoftwares. Hierbei kann zwischen Git, Mercurial oder Subversion gewählt werden. Nachdem eine Software ausgewählt wurde, kann über die Benutzeroberfläche der Lokale Code hoch- oder heruntergeladen werden. So gut wie alle Eigenschaften der regulären Versionsverwaltungen werden angeboten. Versionsverwaltung ist ein wichtiger Punkt in der Entwicklung der App. Sie bietet den Vorteil, dass sie zu einem bestimmten Zeitpunkt zurückzuspringt, dadurch können ein fehlerhafter Code und vor allem größere Probleme vermieden werden. Außerdem ist der Code immer Online, bei Verlust oder Beschädigung des Gerätes kann er somit wiederhergestellt werden. Des Weiteren können mehrere Entwickler gleichzeitig an einem Projekt arbeiten, was dazu führt, dass das Projekt schneller fertiggestellt werden kann. Zusätzlich können Branches erstellt werden. Zum Beispiel kann je ein Branch für die Entwicklung und für den Produktiven Code erstellt werden. Bei Erreichen eines bestimmten Standes kann der Branch der Entwicklung dann in den Branch der Produktion eingespielt werden.

8.4 Plugins

Wie schon erwähnt ist es möglich, zusätzliche Plugins zu installieren. Plugins lassen sich leicht über Android Studio installieren. Dafür muss man lediglich unter Einstellungen in das Untermenü Plugins gehen und das gewünschte Plugin auswählen. Zwei wichtige Plugins werden nachfolgend kurz vorgestellt.

8.4.1 Save Actions

Bei dem Plugin Save Action handelt es sich um ein von JetBrain entwickelter Code,-Verwaltungssystem. Hierbei wird der Code, je nach Einstellungen, zurechtgesetzt. Dabei können Imports optimiert werden. Sollte ein Import nicht genutzt werden, wird dieser einfach entfernt. Außerdem kann der Code neu formatiert werden. Zeilen werden eingerückt und eventuelle leere Zeilen werden gelöscht. Dies führt zu einem sauberen, ästhetisch schöneren Code, welcher sich einfacher lesen lässt. Des Weiteren hat dies den Vorteil, dass in einer Klasse kein unterschiedlich formatierter Code existiert, wenn mehrere Entwickler an einem Projekt arbeiten. Das würde nämlich beim Aktualisieren des Codes über die Versionsverwaltung zu Konflikten führen.

8.4.2 XML Sorter

Bei dem XML Sorter handelt es sich ebenfalls wie bei dem Plugin Save Actions um ein Code-Verwaltungssystem. Hier werden XML-Dateien sortiert. Bei dem stings.xml handelt es sich um eine Datei, die eine Referenz zu einem Text anbietet. Somit sollten alle Texte dort hinterlegt werden. Dadurch wird es den Entwicklern einfacher gemacht Änderungen von Texten vorzunehmen, denn die Texte befinden sich nur in einer Datei ist anstatt in endlos vielen Dateien. Außerdem wird damit vereinfacht, die App in mehreren Sprachen anzubieten. Eine gute Namenskonvention kann auch das Sortieren von Inhalten vereinfachen. Ebenso wie bei der Save Action hat der XML Sorter den Vorteil, dass bei der Versionsverwaltung keine unterschiedlich sortierten Dateien hochgeladen werden.

8.5 Gradle

Gradle basiert auf Java und ist ein Build-Management-Automatisierungs-Tool. Gradle ermöglicht es, den Code schnell und effizient anzubieten, sodass das Endprodukt getestet oder weiterverarbeitet werden kann. Hierbei wird sowohl auf inkrementelle als auch auf parallel ablaufende Build-Prozesse gesetzt, was zum einen bedeutet, dass nur ein Code, der verändert wurde oder damit zusammenhängt, wieder gebaut wird und zum anderen, dass der Task beim Build-Prozess parallel auf verschiedenen CPU’s oder Rechnern läuft. Vereinfacht ausgedrückt: Der Java oder Kotlin Code und der XML Code wird genommen, um eine apk-Datei zu erstellen. Am Ende ist es die apk-Datei, die zur Installation auf dem Handy genutzt wird. Von Gradle werden drei benutzerdefinierte Dateien benutzt, um einen einfachen Build Prozess zu ermöglichen. Die bereits erwähnte build.gradle Datei dient der Definition des Builds der Tasks und der Dependencies (Abhängigkeiten) eines Projekts. Gradle bietet zwei dieser Dateien an, zum einen auf dem Top-Level und zum anderen auf dem Module-Level. Das Top-Level build.gradle befindet sich im Stammverzeichnis des Projekts. Es hat die Hauptfunktion, die Build-Konfigurationen zu definieren, die auf allen Modulen im Projekt angewendet werden. Für die meisten Projekte werden hier keine Änderungen vorgenommen. Außerdem wird die Datei beim Einbinden von wichtigen Dependencies selbst überschieben. Viel wichtiger ist die build.gradle Datei im Module-Level. Hier werden alle App Components hinzugefügt, die die App benötigt. Die Module-Level build.gradle Datei erlaubt verschiedene Einstellungsmöglichkeiten für die App. Der Zweig Android bietet die Versionsverwaltung der Apk an. Hier kann angegeben werden, welche Version für die compileSdk verwendet werden soll. Außerdem besteht noch ein Zweig der defaultConfig, welcher die Verwaltung der minSdk und targetSdk-Version übernimmt. Weiterhin kann hier die Version der entwickelten App definiert werden.

Sowohl Top-Level als auch Module-Level build.gradle werden von Gradle benutzt, um die apk-Datei zu liefern.

```
// Room DB
def room_version = "1.1.1"
implementation "android.arch.persistence.room:runtime:$room_version"
annotationProcessor "android.arch.persistence.room:compiler:$room_version"
```

Im Quellcode wird gezeigt, wie die Dependencies hinzugefügt werden. Es wird das Android Architecture Component Room hinzugefügt, damit dieses in der App verwendet werden kann.

8.6 Logging

Logging ist einer der wichtigsten Bestandteile der App. Hierdurch kann der Entwickler die einzelnen Schritte zurückverfolgen, die zu Fehlern und anderen Problemen führen. Sowohl bei der Entwicklung als auch nach der Fertigstellung der App ist Logging sehr hilfreich. So kann auch noch im Nachhinein nachvollzogen werden, welche Schritte der Nutzer durchlaufen hat, bevor es zu Problemen gekommen ist. Jede gute App benötigt ein gutes Logging- System. Android liefert eine leichte Version, da die Logbefehle lediglich Daten in der Console ausgeben. Im Grunde wird von der Klasse der Befehl „println“ aufgerufen. Der große Nachteil dieser Standard Api ist, dass nichts hinterlegt wird. Somit wird nur das ausgegeben, was gerade auf der App passiert. SLF4J bietet eine effiziente Lösung, um den Code zu loggen und im Handy abzuspeichern. Ein großer Vorteil ist, dass auch nach einigen Tagen nachvollzogen werden kann, was zum Fehler geführt hat.

```
// Logging
implementation 'org.slf4j:slf4j-api:1.7.25'
implementation 'com.github.tony19:logback-android:2.0.0'
```

Der Quellcode zeigt die Anbindung der Api. Nachdem diese in die build.gradle-Datei hinzugefügt wurde und das Projekt synchronisiert worden ist, kann das Framework benutzt werden. Hierbei sollte ein assets ordern im Verzeichnis der main app hinzugefügt werden. Eine Xml-Datei mit dem Namen “logback.xml“ sollte erstellt werden. Diese enthält die Grunddaten der zu erstellenden Logdatei im Gerät.

```
<configuration>
  <appender name="logcat" class="ch.qos.logback.classic.android.LogcatAppender">
    <tagEncoder>
      <pattern>%logger{12}</pattern>
    </tagEncoder>
    <encoder>
      <pattern>[%thread] %msg</pattern>
    </encoder>
  </appender>
  <!-- Erstellt eine Datei für die Logs in dem Datenverzeichnis der App -->
  <appender name="file"a class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/data/data/com.example.MyApp/files/log/myExampleApp.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <!-- tägliche wiederholung -->
      <fileNamePattern>
        /data/data/com.example.MyApp/files/log/myExampleApp.%d{yyyy-MM-dd}.log
      </fileNamePattern>
      <!-- Behält Daten für sieben Tage mit einer Kapazität von 100MB -->
      <maxHistory>7</maxHistory>
      <totalSizeCap>100MB</totalSizeCap>
    </rollingPolicy>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
    </encoder>
  </appender>
  <root level="DEBUG">
    <appender-ref ref="logcat" />
    <appender-ref ref="file" />
  </root>
</configuration>
```

Alle wichtigen zu definierenden Felder sind vorkommentiert. Hierbei sollte lediglich der Name so angepasst werden, dass die Datei im Verzeichnis der App hinterlegt wird. Der Name der Datei kann den Namen der App widerspiegeln. Dies hat den Vorteil, dass bei mehreren Log-Dateien die eindeutige Identifizierung der eigenen Log-Datei vereinfacht wird. Zusätzlich erhält die Datei, die am selben Tag erstellt wird, den Standard-Namen. Sollte am nächsten Tag eine neue Datei geschrieben werden, so werden der Name und das Datum der alten Version durch den neuen Namen und das neue Datum ersetzt. Somit kann nachvollzogen werden, von welchem Tag die Daten sind. Diese werden auf sieben Dateien, mit jeweils höchstens 100 MB Speicherkapazität, begrenzt. Der Encoder dient zur Struktur der Log-Einträge. Hierbei werden das Datum und die Uhrzeit vorangestellt. Als nächstes erscheint der Threadname, der den Eintrag generiert. Darauf folgt die Stufe der Logereignisse auf fünf Zeichen. Anschließend kommt Name des Loggers, der auf 35 Zeichen verkürzt wurde und abschließend erscheint die Lognachricht.

```
private static final Logger log = LoggerFactory.getLogger(MainActivity.class);
```

Bei der Initialisierung muss die Klasse, in der der Logger sich gerade befindet, mit übergeben werden. Dadurch kann nachvollzogen werden, wo was passiert. Nach der Initialisierung kann der Logger großzügig benutzt werden.

8.7 Kommentar

Kommentare werden sehr oft vernachlässigt. Jedoch leisten sie einen großen Beitrag zur Entwicklung der App. Kommentare sind eine klare Erklärung des Codes. Hierbei kann sehr schnell auch von Laien oder Anfängern verstanden werden, was passiert oder passieren soll. Bei einem größeren Team können diese sehr viel Zeit sparen und den Entwicklungsprozess vereinfachen. Bei der Entwicklung durch einen einzigen Programmierer können sie den Vorteil haben, dass nicht jeder einzelne Code-Schnipsel auswendig gelernt werden muss, sondern dass ihn eine kurze Beschreibung schnell wieder auf die Sprünge helfen kann. Außerdem bietet Android die Möglichkeit innerhalb der Kommentare Annotationen wie „@param“ zu verwenden. Die Parameter der Methode oder Klasse können dann bei einem Aufruf oder bei der Initialisierung angezeigt werden. Dadurch kann im Vorfeld schon entschieden werden, ob die richtige Methode oder Klasse verwendet wird oder nicht. Um diese Funktion zu nutzen, muss man die Annotationen innerhalb der /**/ Kommentare hinterlegen.

8.8 Android Manifest

Jedes App-Project muss eine Android Manifest-Datei haben. Hierin werden die wichtigen Eigenschaften der App beschrieben und an die Build Tools, das Betriebssystem und Google Play vermittelt. Neben vielen anderen Informationen müssen in der Manifest-Datei der Packagename der App und außerdem die Komponenten, wie zum Beispiel die Activities, Services, Broadcast Receiver und Content Providers, hinterlegt werden. Hinzu kommt noch, dass hier die Erlaubnisse der App vergeben werden, wie z.B. ob die App auf das Internet oder auf den internen Speicher zugreifen darf. All diese Dinge werden hier auch eingetragen. Weiterhin muss noch angegeben werden, welche Hardware- und Software-Eigenschaften die App benötigt.

```
<!-- required permissions -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.VIBRATE" />
<!-- permissions which need approval -->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Wie schon vorab geklärt wurde, soll die App Webservice-Aufrufe durchführen können. Aus diesem Grund müssen wir in der Manifest-Datei angeben, dass die App auf das Internet zugreifen möchte und zusätzlich dazu den Status des Netzwerkes und die Vibration Funktion. Durch das Einsehen in den Status des Netzwerkes kann der Nutzer die Mitteilung erhalten, ob der Verbindungsversuch überhaupt erfolgreich war oder nicht. Um Daten anzufragen und abzuschicken, muss eine Verbindung zum Internet bestehen. Die Vibrationsfunktion kann nützlich sein, um dem Nutzer etwas mitzuteilen. So kann zum Beispiel im Falle von Fehlern die Vibration angestoßen werden, um den Nutzer physisch auf sie aufmerksam zu machen. Abschließend muss für das Lesen und Schreiben auf einem internen oder externen Speicher eine Bestätigung des Nutzers erfragt werden.

8.9 Virtual Device

Um die App zu testen, wird kein physisches Gerät benötigt. Hierzu kann ein virtuelles Gerät erstellt werden. In der oberen Leiste kann der AVD-Manager gestartet werden. Es besteht die Möglichkeit, sowohl vorgefertigte Profile (virtuelle Geräte) zu nutzen als auch eigene Profile zu erstellen. Dies kann zum Beispiel erforderlich sein, wenn die gewünschten Geräte nicht angeboten werden. Nachdem ein Profil erstellt worden ist, wird das Betriebssystem (API) ausgewählt und die virtuelle Maschine kann installiert werden. Es können auch mehrere gleichzeitig installiert werden. Von nun an kann die App auf das virtuelle Gerät installiert und getestet werden.

8.10 Aufbau

Nachdem alle Grundfunktionen besprochen und ausgeführt wurden, kann mit der Entwicklung der notwendigen Funktionen begonnen werden.

8.10.1 Dagger 2 Implementierung

Wie schon bei der Softwarearchitektur erwähnt, dient Dagger 2 der Vereinfachung von Anhängigkeiten. Dagger 2 im Zusammenspiel mit Retrofit 2 und Room bietet der App viel einfachere Zugriffe. So wird nur einmal in einem Modul angegeben, welche Services es gibt. Diese Services können von überall aus verwendet werden. Ein Modul ist eine Oberklasse aller dazugehörigen Komponenten.

Zuallererst wird die Basisklasse für die App erstellt, die von der DaggerApplication erbt. Die Klasse stellt den AndroidInjector der Basis-Klasse bereit.

```
public class OmcrApp extends DaggerApplication {

   @Override
   protected AndroidInjector<? extends DaggerApplication > applicationInjector() {
      return DaggerOmcrAppComponent.builder().create(this);
   }
```

Die Klasse ist ziemlich klein und übersichtlich. Hierbei wird eine Klasse erstellt, die nach dem ersten Build der App zur Verfügung steht. Anschließend können die Module geschrieben werden. Die Basis-Klasse kann nun für die gesamte App zur Verfügung gestellt werden.

```
@Module
abstract class OmcrAppModule {

   @Binds
   @Singleton
   abstract Application application(OmcrApp app);
```

Das OmcrAppModule, das Basis-Modul der App, stellt die Application, Activities und Fragments zur Verfügung. Alle zu erstellenden Benutzeroberflächen, seien es Activities, Fragments oder Dialoge, müssen in das Modul geschrieben werden. Mit der Annotation „Module“ teilt Dagger mit, dass es sich um ein Dagger-Modul handelt. Der Vorteil von Annotationen ist, dass der zu schreibende Code reduziert wird. Außerdem sorgt Dagger dafür, dass die generierten Codes optimiert werden. Um die Parameter zurückzuliefern, kann die Annotation „Binds“ verwendet werden. Im Quellcode wird die Application mit dieser Annotation versehen. Da die Methode abstrakt ist, wird Dagger somit das Objekt nicht instanziieren. Stattdessen nutzt Dagger direkt den angebotenen Parameter OmcrApp. Singleton sorgt dafür, dass es von dem Objekt nur eine einzige Instanz geben kann. Jedes Mal, wenn die Klasse mit der Annotation „Inject“ injiziert wird, handelt es sich um dieselbe Instanz. Dies ist nur solange möglich, solange die App sich in derselben App Component befindet. Nachdem das App-Modul erstellt wurde, sollte eine Modul-Klasse für das ViewModel, die Webservices und die Datenbanken erstellt werden. Hierbei ist der Ablauf sehr ähnlich.

Nachdem alle Module erstellt worden sind, muss die AppComponent-Klasse erstellt werden, die bereits kurz erwähnt wurde. Hierbei handelt es sich um ein Interface, dass die Basis- Komponente der App und des Abhängigkeitsgraphen ist. Der Abhängigkeitsgraph erstellt die Abhängigkeiten von mehreren Objekten voneinander.

```
@Singleton
@Component(modules = {AndroidSupportInjectionModule.class, OmcrAppModule.class,
ViewModelModule.class, WebserviceModule.class, DatabaseModule.class})
public interface OmcrAppComponent extends AndroidInjector<OmcrApp> {

    @Component.Builder
    abstract class Builder extends AndroidInjector.Builder<OmcrApp> {
    }
}
```

Auch hier wird die Annotation „Singleton“ verwendet, da es wichtig ist, genau diese eine AppComponent-Klasse zu haben. Mit der „Component“-Annotation wird festgelegt, welche Module zu dieser AppComponent gehören. Dagger generiert den Code für die Module, sodass sie über das AppComponent zur Verfügung stehen. Das Modul AndroidSupportInjectionModule stellt die Verwendung von dagger.android und dagger.support Framework-Klassen da.

8.10.2 Room

Wie schon in der Anforderungsanalyse besprochen, ist es wichtig, die App offline nutzen zu können. Hierfür wird Room benutzt. Dabei werden die Daten, die wir aus der Datenbank einlesen, lokal gespeichert. Dies hat den Vorteil, dass bei Verbindungsabbrüchen oder bei keinem Empfang trotzdem mit den Daten weitergearbeitet werden kann. Außerdem können dabei die Daten nach Absturz oder Schließung der App aus der lokalen Datenbank ausgelesen werden, anstatt einen Serviceaufruf starten zu müssen. Hierfür wird eine lokale Datenbank erstellt. Die automatische Generierung der Datenbank lässt sich auch anhand von Annotationen steuern. Dafür muss für ein Objekt, das für eine Tabelle in der Datenbank steht, nur die Annotation „Entity“ benutzt werden.

```
@Entity
public class OmcrAppConfig {

    @PrimaryKey
    private Integer id;
    private Integer filiale;
```

Die Annotation „Entity“ erstellt eine Tabelle eines Artikels mit den entsprechenden Spalten für die Werte. PrimaryKey steht für die eindeutige Id jedes einzelnen Eintrages in die Tabelle und wird selbstständig erstellt und erhöht. Um die Tabelle einen anderen Namen als den Namen der Klasse zuzuweisen, kann mit der „Entity“-Annotation der Name definiert werden. Außerdem müssen Getter- und Setter-Methoden der Attribute erstellt werden, um Zugriff auf die Werte zu erhalten.

```
@Entity(tableName = "einstellungen")
```

Dadurch lässt sich die Namenskonvention der Java-Klassen und der SQL-Datenbanken beibehalten. Nachdem definiert worden ist, welche Tabellen in die Datenbank gehören, müssen für alle Tabellen sogenannte DAO‘s (Data Access Object) erstellt werden. In den DAO‘s werden vorgefertigte Query-Befehle geschrieben.

```
@Dao
public interface OmcrAppConfigDao {

    @Query("SELECT * FROM omcrappconfig")
    OmcrAppConfig findOmcrAppConfig();
```

Die Befehle unterscheiden sich nicht sonderlich viel von regulären SQL-Befehlen. Der oben zu sehende Befehl gibt die Werte, die in der Datenbank liegen, als Objekt wieder. Diese Tabelle der Einstellungen ist wichtig für die App, da hier alle Grunddaten gespeichert werden können. Wie in der Anforderung besprochen, soll die App bzw. das Gerät wissen, in welcher Filiale es sich befindet, um nur Aufträge oder Anforderungen der jeweiligen Filiale zu erhalten. Des Weiteren muss noch eine Tabelle für die Artikel erstellt werden, um diese nach dem Aufruf aus dem Backend zwischenzuspeichern. Der Ablauf ist derselbe. Hierfür werden selbstverständlich andere Attribute bzw. Spalten verwendet. Dies hat, wie schon mehrfach erwähnt, den Vorteil, dass Daten lokal zu jeder Zeit abgerufen werden können, ohne eine Verbindung zum Internet haben zu müssen. Nachdem alle Tabellen und die dazugehörigen DAO’s erstellt worden sind, müssen diese in der Datenbank definiert werden.

```
@Database (entities = {OmcrAppConfig.class, Artikel.class}, version = 1)
public abstract class OmcrDatabase extends RoomDatabase {

   public abstract OmcrAppConfigDao omcrAppConfigDao();

   public abstract ArtikelDao artikelDao();
```

Hier wird die Datenbank definiert. Alle Tabellen der Datenbank und die dazugehörigen DAO’s müssen hier eingefügt werden. Es besteht die Möglichkeit, mehrere lokale Datenbanken zu erstellen. Abschließend kann die Datenbank erstellt werden. Dies geschieht im Datenbank-Modul, das an die AppComponent-Klasse übergeben wird.

```
@Module
public class DatabaseModule {

   @Singleton
   @Provides
   public static OmcrDatabase provideOmcrDatabase(Application app) {
      return Room.databaseBuilder(app, OmcrDatabase.class, "omcrpapp.db").build();
   }
```

Hierbei wird die Datenbank ebenfalls mit der „Singleton“-Annotation ausgezeichnet. Die Datenbank soll stets über die gesamte Zeit aufrufbar sein. Außerdem wird diese mit der Annotation „Provides“ versehen, die dafür sorgt, dass innerhalb der App eine Instanz der Datenbank zur Verfügung steht. Beim Erstellen der Daten wird der Kontext der App gefordert. Des Weiteren wird die Klasse der Datenbank, die zuvor erstellt wurde, benötigt. Hierbei muss die Klasse mit der Annotation „Database“ versehen werden und von der RoomDatabase-Klasse erben. Abschließend muss der Name der Datenbank übergeben werden. Nachdem die Datenbank erstellt worden ist, können Daten gespeichert und abgerufen werden.

8.10.3 DataWedge

Die Implementierung der DataWedge API benötigt ein wenig Recherche und Vorarbeit. Hierbei muss berücksichtigt werden, welche Arten von Barcodes sowie die Mindest- und Maximallänge gescannt werden und außerdem welche Art von Scanner benötigt wird. Anschließend kann in der Dokumentation von Zebra jedes Detail gefunden werden. Um ein Profil erstellen zu können, muss der DataWedge App mittgeteilt werden, was zu tun ist. Dies geschieht über das Intent. Ein Intent ist eine durchzuführende Operation. In diesem Fall werden Daten von einer App zu einer anderen geschickt bzw. Oberflächen werden angesprochen. Über das Intent lassen sich die Daten anhand eines Schlüssels und eines Wertes übermitteln. Der Schlüssel ist wichtig für die Erkennung des Ziels. Über diesen Schlüssel kann die App, von der die Operation ausgeht, definieren, was in dem Wert steckt. Die App hingegen, die das Intent erwartet, kann anhand des Schlüssels diesen Wert abrufen. Die API gibt bestimmte Schlüssel vor. Bei einigen Werten kann beliebig frei entschieden werden. Bei anderen hingegen müssen die Werte aus der API verwendet werden.

```
Bundle profileConfig = new Bundle();
profileConfig.putString("PROFILE_NAME", "OMCR-App");
profileConfig.putString("PROFILE_ENABLED", "true");
profileConfig.putString("CONFIG_MODE", "UPDATE");
```

Zunächst wird das Hauptbündel der Einstellungen erstellt. Hier werden der Name der App ausgewählt und das Profil aktiviert. Auch wenn das Profil neu erstellt wird, wird der Status der Einstellung auf „Aktualisieren“ gesetzt. Dies verhindert, dass die Einstellungen bei bereits existierendem Profil nicht übernommen werden.

```
Bundle barcodeConfig = new Bundle();
barcodeConfig.putString("PLUGIN_NAME", "BARCODE");
barcodeConfig.putString("RESET_CONFIG", "true");
```

Anschließend werden die Einstellungen der Barcodes definiert. Dazu kommt noch ein Bündel, welches die Parameter der Barcodes mit sich bringt.

```
Bundle barcodeProps = new Bundle();
barcodeProps.putString("scanner_selection", "auto");
barcodeProps.putString("scanner input enabled", "true");
barcodeProps.putString("decoder_code128", Code128Value);
```

Im Quellcode ist zu sehen, dass der Barcode Code128 aktiviert wird. Dieser wird automatisch gewählt und dem Scanner wird erlaubt Barcodes zu erhalten. Bei den Parametern des Barcodes kann die Liste lang werden, deshalb sollte überlegt werden, welche Barcodes wichtig sind. In demselben Prozedere wird eine Liste mit allen Activities erstellt, um die gescannten Daten an die Ansicht zu liefern.

```
barcodeConfig.putBundle("PARAM_LIST", barcodeProps);
profileConfig.putBundle("PLUGIN_CONFIG", barcodeConfig);
sendDataWedgeIntentWithExtra("com.symbol.datawedge.api.ACTION",
      "com.symbol.datawedge.api.SET_CONFIG",
      profileConfig);
```

Abschließend werden die Bündel zusammengetragen und an die App weitergeleitet. Bei der Definierung der Activities in der Manifest-Datei können der Activity eine oder mehrere Actions zugewiesen werden. Sollte jedoch der Activity keine Action zugewiesen werden, bedeutet das, dass die Activity kein Intent akzeptiert. Nachdem die Einstellungen für den Scanner zur DataWedge App übermittelt wurden, kann der BroadcastReceiver implementiert werden.

```
   IntentFilter filter = new IntentFilter();
   filter.addCategory(Intent.CATEGORY_DEFAULT);
   filter.addAction("com.example.omcrapp.ACTION");
   registerReceiver(myBroadcastReceiver, filter);
}

private BroadcastReceiver myBroadcastReceiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      String action = intent.getAction();
      if (action.equals("com.example.omcrapp.ACTION")) {
         String barcode = intent.getStringExtra("com.symbol.datawedge.data_string");
      }
   }
};
```

Für den BroadcastReceiver wird das Intent verwendet. Wie schon erwähnt, dient Intent zur Kommunikation zwischen Activities. Dies ist sogar App übergreifend möglich. Bei der Registrierung des Receivers muss angegeben werden, worauf gewartet wird. Dafür wird der IntentFilter benötigt. Dem IntentFilter wird die Kategorie übergeben, welche für einen spezifischen Intent-Aufruf relevant ist, andernfalls wird immer die Standard-Kategorie ausgewählt. Außerdem wird die Action der Activity übergeben. Anschließend kann der BroadcastReceiver registriert werden. Bei der Initialisierung des BroadcastReceivers können im Anschluss die Daten verwendet und verarbeitet werden. Das Intent liefert mehr Daten zurück als nur den gescannten Barcode. Zum Beispiel kann noch eingesehen werden, um welchen Barcode es sich handelt.

8.10.4 UI

Die Realisierung der UI bedarf keiner ausführlichen Erklärung. Es handelt sich um Grundlagen der Android-Programmierung. Das Einsetzen von Attributen lässt sich sogar per drag and drop realisieren. In der Beschreibung der App wurde darauf hingewiesen, wie die App auszusehen hat. Für jede Ansicht der Listen kann eine Activity verwendet werden. Der Vorteil ist, dass viele Daten erwartet werden. Für die detaillierte Ansicht der Artikel kann ein Fragment genutzt werden. Über das Fragment lassen sich mehrere Benutzeroberflächen aufeinander stapeln. Außerdem lassen sich Fragments leichter wiederverwenden und ersetzen. Die Anzeige der Details benötigt nicht viele Funktionen, deshalb ist es von Vorteil ein Fragment zu verwenden. Für die Ansicht der Absage sollte ein Dialog verwendet werden. Ein Dialog lässt sich schnell implementieren. Die Aufgabe besteht lediglich aus einer Abfrage. Im Fragment kann dann die Eingabe weiterverarbeitet werden. Außer diesen Ansichten sind keine weiteren Benutzeroberflächen geplant.

9 Auswertung

Nachdem die Themen Anforderungsanalyse sowie Konzeption behandelt und anschließend die Beschreibung der App Realisierung vorgenommen worden sind, werde ich hier die von mir entwickelte App vorstellen.

9.1 Hauptmenü

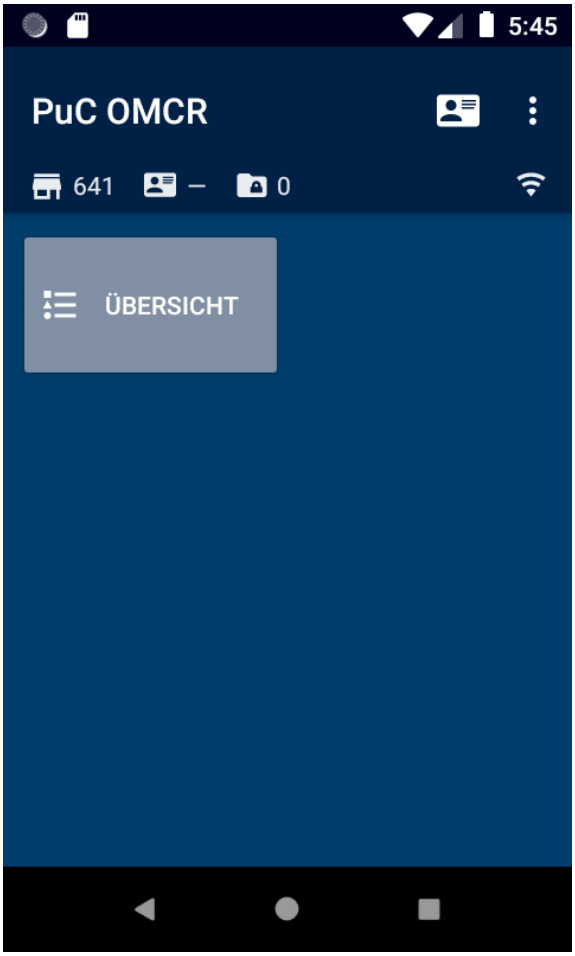

Abbildung 19: Hauptbildschirm ohne Personal [eigene App]

In der Abb. 19 ist die Startansicht der App zu sehen. Die Änderung des Startbildschirms vom Filter zu einer Oberfläche mit einem Button hat den Hintergrund, dass die Implementierung späterer Funktionen in die App vereinfacht wird. Außerdem können aus dieser Ansicht Grundeinstellungen vorgenommen werden. Zum Beispiel kann von hier aus die Personalnummer des Nutzers erfasst werden. Dies hat den Vorteil, dass nachvollzogen werden kann, welcher Mitarbeiter zu welcher Zeit welche Artikel bearbeitet hat. Bei der Eingabe der Personalnummer wird dem Benutzer ein Dialog angezeigt.

9.2 Personalnummer

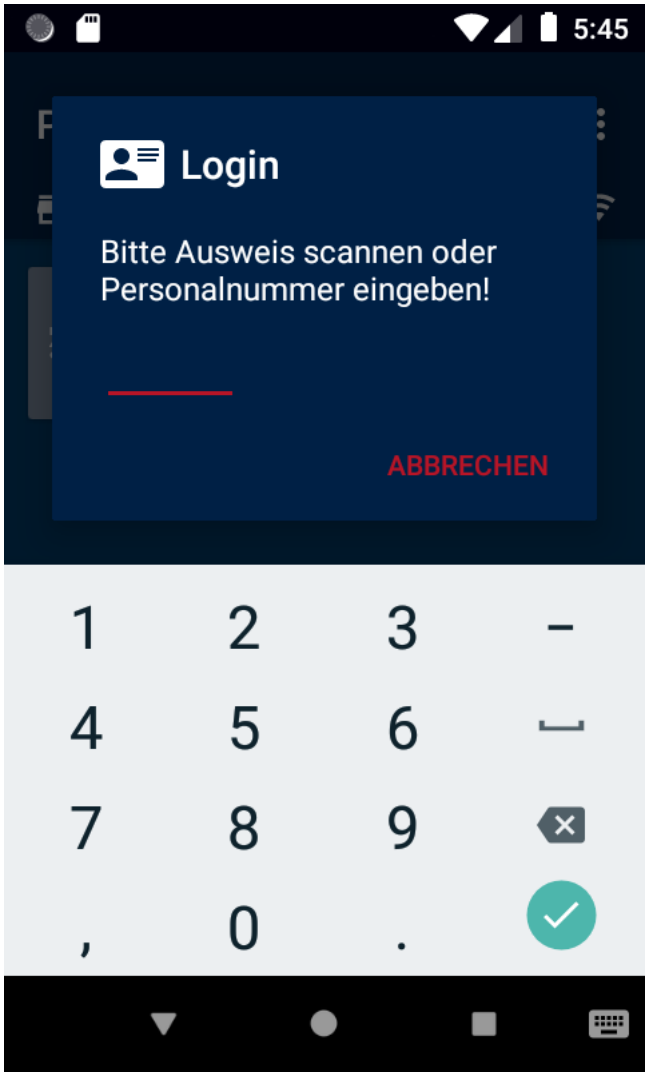

Abbildung 20: Personalnummer Dialog [eigene App]

In der Abbildung 20 ist der Dialog für die Personalnummer Erfassung zu sehen. Hier kann der Nutzer seine Personalnummer eingeben. Nachdem diese eingegeben wurde, wird die Personalnummer im Kopfbereich der gesamten App angezeigt. Weiterhin wurde der Kopfbereich so festgelegt, dass dieser über alle Ansichten einsehbar ist. Des Weiteren lassen sich die Filiale, in der das mobile Gerät sich befindet und die schon bearbeiteten Vorgänge anzeigen. Zusätzlich dazu wird in der App der Status der Internetverbindung angezeigt.

9.3 Filter

Abbildung 21: Ansicht des Filters [eigene App]

Nachdem der Button „Übersicht“ betätigt worden ist, wird die Ansicht mit den Filterkriterien angezeigt. Hierbei wurde, wie in der Anforderungsanalyse beschrieben, der Filter an den des bestehenden Systems angelehnt. Der Bereich und der Ressort können bei Bedarf ausgefüllt werden. Das Erstellungsdatum hingegen muss gewählt werden. Voreingestellt ist immer der Tag, an dem die App geöffnet wird. Eine weitere Voreinstellung ist der Status. Hierbei wird „offen“ als Standard gewählt. Dadurch lassen sich alle Anforderungen anzeigen, sowohl die offenen als auch die dringenden. Bei Zurücksetzen werden alle Eingaben und Auswahlen zu den Voreinstellungen zurückgesetzt. Dabei werden das Ressort und der Bereich gelöscht.

9.4 Liste der Anforderungen

Abbildung 22: Liste der Anforderungen [eigene App]

Nachdem die Filterkriterien ausgefüllt und ausgewählt worden sind, wird nach dem Buttonklick „suchen“ die Liste der angeforderten Artikel angezeigt. Hierbei wurde darauf verzichtet, den Status erneut anzeigen zu lassen. Stattdessen sollen die Farben der Einträge den Status widerspiegeln. Das Ziel der Liste ist, dass so viele Anforderungen wie möglich angezeigt werden. Außerdem werden dem Nutzer nur essentielle Daten angezeigt, die bei der Lokalisierung helfen. Dadurch kann der Nutzer einschätzen, wie viele Artikel in dem bestimmten Bereich vorhanden sind. Abgesehen davon lässt sich die Liste der Einträge sortieren. Die Liste soll sich nach dem Wert aus dem Dropdown-Menü sortieren. Zu einem gibt es die Sortierung nach Bereich, Ressort, Bereich und Ressort, Ressort und Bereich.

Eine zweite Liste wurde überdacht und nicht umgesetzt. Dies hat den Hintergrund, dass im Filterbereich angegeben werden kann, welchen Bereich und welches Ressort die Artikel haben sollen. Dadurch erübrigt es sich die Erstellung einer zweiten Liste, was auch dazu führt, dass unnötige Mehrarbeit vermieden wird und was die App verkomplizieren würde. Die Artikel lassen sich scannen und nach einem erfolgreichen Scan soll die Hintergrundfarbe des Eintrages geändert werden. Alle Anforderungen werden, wie in der Anforderungsanalyse beschrieben, lokal in der Datenbank gespeichert. Dadurch können die Daten nach einem Absturz abgerufen werden. Abgesehen davon wird anhand des Etikettenscans der Barcode validiert, der mit den Daten der Anforderungen abgeglichen wird. Dadurch kann verifiziert werden, ob es sich um den richtigen Artikel handelt. Zusätzlich dazu wird ein Eintrag in der Datenbank hinterlegt, dass ein Artikel erfolgreich gescannt wurde. Hierbei wird eine Referenz zum Artikel gesetzt, sodass bei der Versendung des Artikels entweder eine Liste angezeigt oder das erneute Scannen der gefundenen Artikel ermöglicht wird. Sollte der Nutzer sich ausloggen, werden alle bestehenden Vorgänge gelöscht. Außerdem werden alle Ansichten geschlossen und der Nutzer wird zum Hauptmenü gebracht. Dies hat den Hintergrund, dass der Nutzer zu jeder Zeit angemeldet sein muss. Dadurch lässt sich immer nachvollziehen, welcher Mitarbeiter welche Arbeit geleistet hat. Eine Überprüfung der Produktivität der Mitarbeiter bedeutet dies nicht. Vielmehr geht es darum, dass bei Fehlern nachvollzogen werden kann, wie dieser zustande gekommen ist. Gegebenenfalls kann der Mitarbeiter dazu befragt werden. Des Weiteren kann die Sicht des Nutzers eine große Rolle spielen, um die Fehlerquelle herauszufinden. Alle Vorgänge werden ebenfalls dem Nutzer zugewiesen. Zu den Vorgängen gehören nicht die erstellten Listen für die Versendung, sondern vielmehr auch die schon gescannten Artikel in der Ansicht der Liste von allen Anforderungen.

9.5 Artikeldetails

Abbildung 23: Artikeldetails [eigene App]

Nachdem ein Artikel in der Liste angeklickt worden ist, erscheint die Ansicht der Artikeldetails. Hier werden dem Nutzer die genauen Daten zum Artikel angezeigt. Des Weiteren soll ein Foto des Artikels angezeigt werden. Wie in der Anforderungsanalyse beschrieben, ist dies ein großer Vorteil der mobilen Geräte. Auch hier hat der Nutzer die Möglichkeit sich auszuloggen. Außerdem kann der Artikel, sollte er nicht ausfindig gemacht werden, wieder abgesagt werden. Dafür muss der Nutzer lediglich auf den Button „Absagen“ klicken.

9.6 Absage

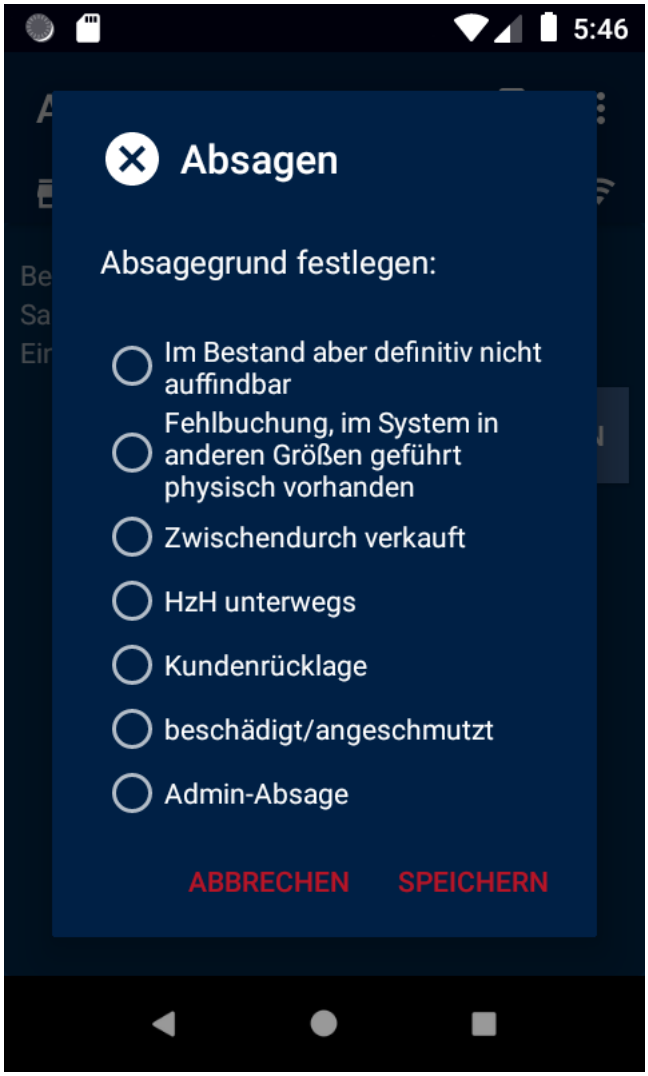

Abbildung 24: Dialog der Absage [eigene App]

Nachdem der Nutzer entschieden hat, hat den Artikel abzusagen, öffnet sich ein Dialogfeld. Hierbei werden alle Gründe aus der bestehenden Anwendung übernommen. Die Reihenfolge der Gründe ist ebenfalls gleichgeblieben. Das hat den Vorteil, dass der Nutzer sofort versteht, was in diesem Dialog gefordert ist. Den Dialog aus der Ansicht der Artikeldetails zu öffnen hat den Vorteil, dass alle Artikeldaten bereits vorhanden sind. Dadurch kann der Aufruf zum Backend sofort abgewickelt werden. Nachdem ein Grund ausgewählt worden ist, gelangt der Nutzer zurück zur Liste der Anforderungen.

9.7 Log

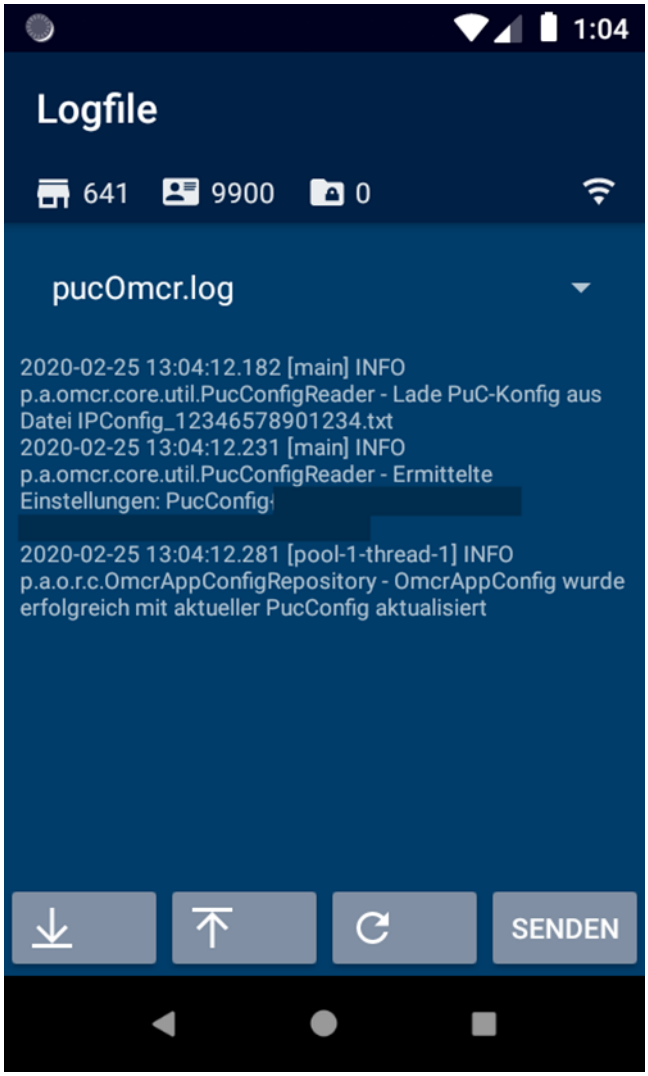

Abbildung 25: Logfile [eigene App]

Im Logfile werden alle wichtigen Schritte, die die App im Quellcode abläuft, gespeichert. Dadurch kann im Nachhinein nachgelesen werden, welche Schritte vom Nutzer durchgeführt wurden. Das Logfile kann der IT-Abteilung zugesandt werden. Diese kann anhand des Logfiles extern noch einmal alle Schritte durchgehen, um nachvollziehen zu können, was zu Fehlern geführt hat. Außerdem können Dateien von bis zu einer Woche angezeigt oder an die IT-Abteilung verschickt werden.

9.8 Versenden

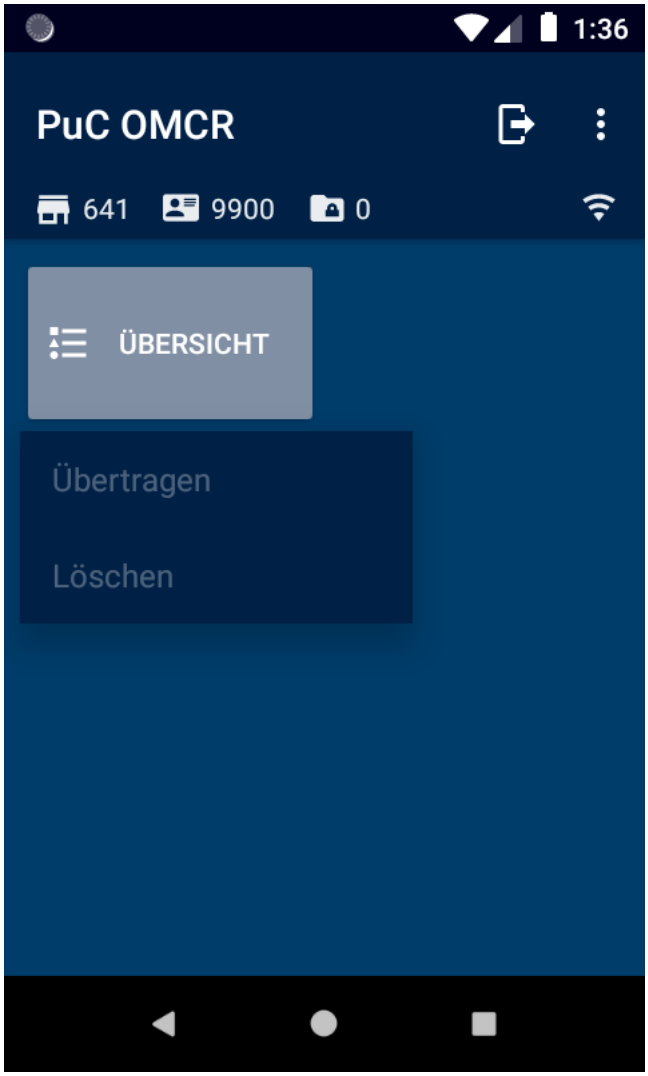

Abbildung 26: Versenden der Vorgänge [eigene App]

Aus dem Hauptmenü können alle bearbeiteten Vorgänge versendet werden. Bei längerem halten des Buttons erscheint das Menü. Über diesem Menü kann der Nutzer zu einer Ansicht weitergeleitet werden, die alle bearbeiteten Anforderungen bereithält. Dazu kann erneut der Artikel gescannt werden, um erneut zu verifizieren ob es sich um den richtigen Artikel handelt. Abschließend soll der dazugehörige Lieferschein gedruckt werden. Diese Ansicht wurde noch nicht implementiert. Außerdem sollen Vorgänge gelöscht werden können.

9.9 Weiteres

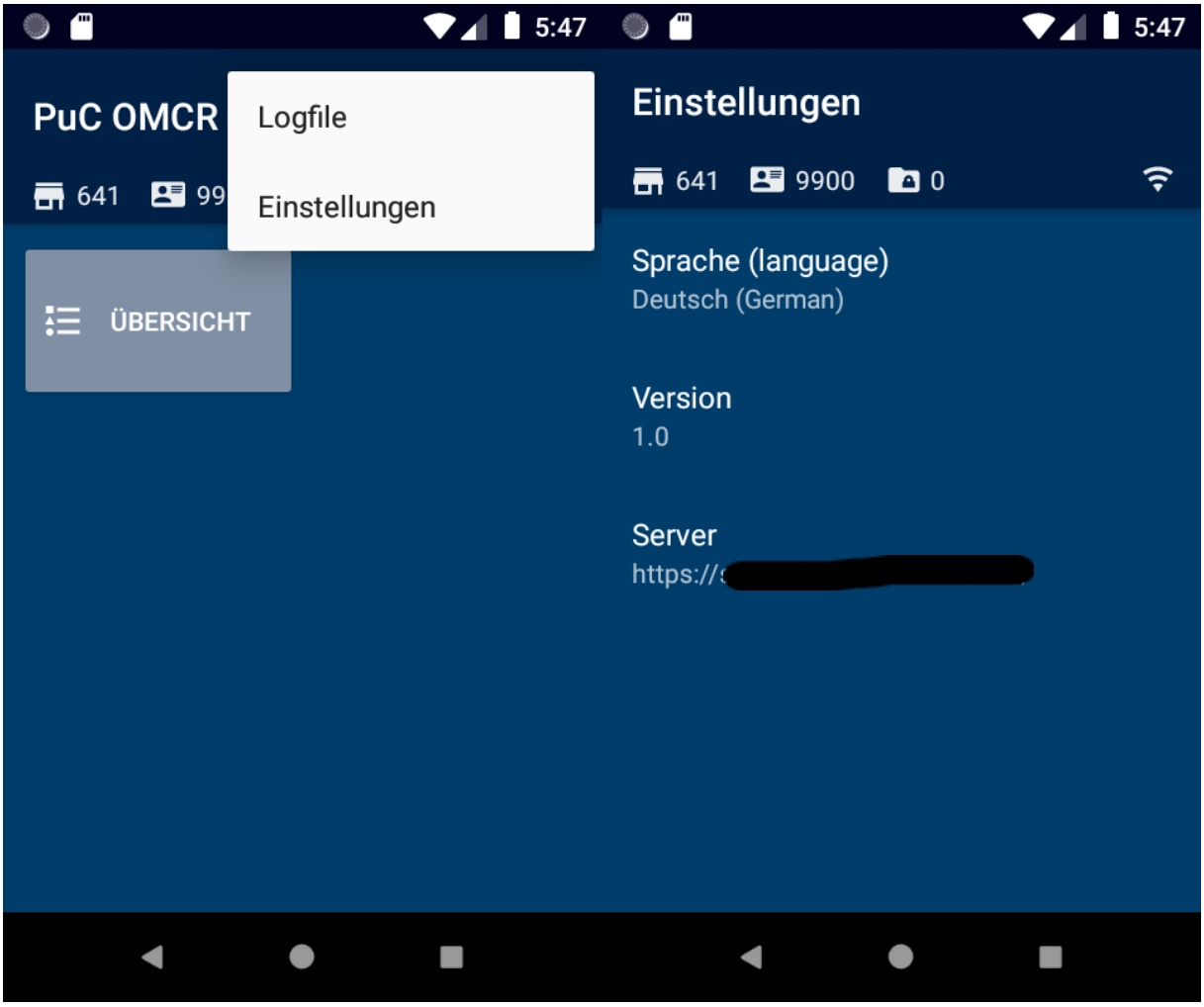

Abbildung 27: Menü und Einstellungen [eigene App]

Von jeder Ansicht aus lässt sich das Dropdown-Menü anzeigen. Der Nutzer hat hier die Wahl, sich entweder die Logfiles oder die Einstellungen anzeigen zu lassen. In den Einstellungen lässt sich die Sprache einsehen und verändern. Da Peek & Cloppenburg im Ausland als VanGraaf vertreten ist, kann dies in Zukunft das Einspielen mehrerer Sprachen erleichtern. Des Weiteren wird die aktuelle Version angezeigt. Diese ist ebenfalls ein Feature für die Zukunft. Bei mehreren Versionen kann dabei nachvollzogen werden, ob die App auf dem aktuellen Stand ist oder nicht. Außerdem wird der Server angezeigt. Auch hier kann schnell eingesehen werden, ob die App gegen den richtigen Server, von denen das Unternehmen sehr viele in den verschiedenen Bereichen besitzt, läuft oder nicht.

10 Fazit

Das Entwickeln von Android Apps entwickelt sich stets weiter. Mit der Zeit werden viele Erweiterungen für das Software Development Kit vorgenommen. Die Entwickler von Android versuchen es App-Entwicklern leichter zu machen, Apps zu entwickeln. Das Entwickeln der App ist lediglich ein Bestandteil des ganzen Prozesses. Die Projektausarbeitung hat eine wesentlich höhere Priorität als das reine Entwickeln der App. Bei der Anforderungsanalyse oder der Konzeption lassen sich durch präventive Arbeit im Voraus viele Fehler vermeiden. Hierfür ist es wichtig, mit dem Kunden oder dem Projektleiter das Gespräch zu suchen. Anschließend kann der App-Entwickler recherchieren und das erlangte Wissen der Theorie in die Praxis umsetzen. Dabei ist Präventionsarbeit von großem Vorteil, da die Ausarbeitung von relevanten und irrelevanten Dingen wegfällt. Der App-Entwickler kann sich auf die Umsetzung konzentrieren und somit bleibt ihm mehr Zeit zum Recherchieren übrig. Um ein guter Android App-Entwickler zu sein, sollte stets darauf geachtet werden, dass die zu entwickelnde App stets auf dem aktuellen Stand ist. Dadurch wird gewährleistet, dass die App reibungslos mit dem Betriebssystem des Mobilen Endgerätes zusammenarbeitet. Immer auf dem Aktuellen Stand zu bleiben erfordert viel Eigeninitiative. Somit müssen Entwickler viel Recherche auf die Entwicklung von Apps betreiben. Dadurch, dass das Software Development Kit dem Entwickler mit weniger Code schreiben zur App verhilft, muss verstanden werden, welche Auswirkungen die neuen Funktionen in der Entwicklung haben. Außerdem gibt es für simple Apps verschiedene Art und Weisen, eine Android App zu entwickeln. Von daher ist es von großer Bedeutung, eine gute Projektplanung durchzuführen. Hierbei muss von vorneherein erkennbar sein, was die App anzubieten hat (welche Funktionen die App anbietet). Dadurch lässt sich erkennen, wie die App in Zukunft entwickelt werden soll. In dieser Arbeit wurde stets darauf geachtet, dass die App die neusten Funktionen der Android App Entwicklung verwendet. Abschließend lässt sich sagen, dass die App auf dem Aktuellen Stand ist. Diese kann die Arbeit des Nutzers um ein Vielfaches erleichtern. Außerdem kann die App produktiv getestet werden, bevor diese an alle Filialen verteilt wird. Zukünftige Meilensteine sind die Pflege der Benutzeroberfläche und Erweiterung oder Vereinfachung der Lokalen Datenbank.

II. Abbildungsverzeichnis

III. Literatur- und Quellenverzeichnis

1 Bronwall Moffat: „Omnichannel vs. Multichannel“ (abgerufen am 24.12.2019) https://www.the-future-of-commerce.com/2017/09/13/omnichannel-vs-multichannel/

2 Statista: „Android Statistik“ (abgerufen am 24.12.2019) https://de.statista.com/themen/1355/android/

3 Wikipedia: „Android (Betriebssystem)“ (abgerufen am 24.12.2019) https://de.wikipedia.org/wiki/Android_(Betriebssystem)

4 Wikipedia: „Softwarearchitektur“ (abgerufen am 04.01.2020) https://de.wikipedia.org/wiki/Softwarearchitektur

5 Android: „Architecture Components“ (abgerufen am 10.01.2020) https://developer.android.com/topic/libraries/architecture

6 Android: „ViewModel“ (abgerufen am 10.01.2020) https://developer.android.com/topic/libraries/architecture/viewmodel

7 Android: „Data Binding Library“ (abgerufen am 10.01.2020) https://developer.android.com/topic/libraries/data-binding

8 Android: „Two-way data binding“ (abgerufen am 10.01.2020) https://developer.android.com/topic/libraries/data-binding/two-way

9 Android: „LiveData“ (abgerufen am 11.01.2020) https://developer.android.com/topic/libraries/architecture/livedata

10 Android: „Room Persistence Library“ (abgerufen am 11.01.2020) https://developer.android.com/topic/libraries/architecture/room

11 Wikipedia: „MVVM“ (abgerufen am 11.01.2020) https://de.wikipedia.org/wiki/Model_View_ViewModel

12 Android Studio: „Abbildung 9: Android API Statistik“ (abgerufen am 11.01.2020)

13 Jet Brains: „Save Actions“ (abgerufen am 19.01.2020) https://plugins.jetbrains.com/plugin/7642-save-actions/

14 GitHub: „Android-XML-Sorter“ (abgerufen am 23.01.2020) https://github.com/roana0229/android-xml-sorter

15 Gradle: „Gradle“ (abgerufen am 23.01.2020)
https://gradle.org

16 Wikipedia: „Gradle vs Maven“ (abgerufen am 23.01.2020)
https://gradle.org/maven-vs-gradle/

17 Wikipedia: „Gradle“ (abgerufen am 23.01.2020) https://de.wikipedia.org/wiki/Gradle

18 Android: „Comment“ (abgerufen am 24.01.2020) https://developer.android.com/reference/org/w3c/dom/Comment

19 Android: „Dagger“ (abgerufen am 24.01.2020) https://developer.android.com/training/dependency-injection/dagger-android

20 Square: „Retrofit“ (abgerufen am 24.01.2020) https://square.github.io/retrofit/

21 Android: „Android Manifest“ (abgerufen am 24.01.2020) https://developer.android.com/guide/topics/manifest/manifest-intro

22 Peek & Cloppenburg: „P&C Nord“ (abgerufen am 07.02.2020) https://de.wikipedia.org/wiki/Peek_%26_Cloppenburg_(Hamburg)

23 AEV: „Nativ VS Web App“ (abgerufen am 07.02.2020)
https://app-entwickler-verzeichnis.de/faq-app-entwicklung/11-definitionen/586-unterschiede-und-vergleich-native-apps-vs-web-apps-2

24 Ryte: „OMCR“ (abgerufen am 08.02.2020) https://de.ryte.com/wiki/Omnichannel_Retailing

25 Destatis: „Digitale Gesellschaft“ (abgerufen am 08.02.2020) https://www.destatis.de/Europa/DE/Thema/Wissenschaft-Technologie-digitaleGesellschaft/Online_Shopping.html

26 EHI: „Top 100 Online Shops in Deutschland“ (abgerufen am 08.02.2020)
https://www.ehi.org/de/top-100-umsatzstaerkste-onlineshops-in-deutschland/

27 Wikipedia: „Vorgehensmodell“ (abgerufen am 12.02.2020) https://de.wikipedia.org/wiki/Vorgehensmodell

28 Wikipedia: „Vorgehensmodell der Softwareentwicklung (abgerufen am 12.02.2020)
https://de.wikipedia.org/wiki/Vorgehensmodell_zur_Softwareentwicklung

29 Android: „Dialog“ (abgerufen am 12.02.2020) https://developer.android.com/guide/topics/ui/dialogs

30 Projektmanagement: „Wasserfallmodell“ (abgerufen am 13.02.20202) https://www.online-projektmanagement.info/agiles-projektmanagement-scrum-methode/scrum-versus-wasserfallmodell/das-wasserfallmodell/

31 Atlassian: „Jira“ (abgerufen am 13.02.2020)https://www.atlassian.com/de/software/jira

32 AP-Verlag: „Zufriedene Mitarbeiter“ (abgerufen am 14.02.2020) https://ap-verlag.de/studie-glueckliche-mitarbeiter-leisten-bessere-arbeit/32601/

33 Zebra: „EMDK“ abgerufen am (14.02.2020) https://www.zebra.com/de/de/products/software/mobile-computers/mobile-app-utilities/emdk-for-android.html

34 Zebra: „DataWedge“ (abgerufen am 14.02.2020) https://www.zebra.com/de/de/products/software/mobile-computers/datawedge.html

35 Android: „Room local database“ (abgerufen am 15.02.2020) https://developer.android.com/training/data-storage/room

36 Zebra: „DataWedge“ (abgerufen am 15.02.2020) https://techdocs.zebra.com/datawedge/7-6/guide/input/barcode/#code128

37 Android: „BroadcastReceiver“ (abgerufen am 15.02.2020) https://developer.android.com/reference/android/content/BroadcastReceiver

38 Zebra: „Single Decode Mode“ (abgerufen am 15.02.2020) https://techdocs.zebra.com/datawedge/6-6/guide/output/intent/#singledecodemode

Eidesstattliche Versicherung zur Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____