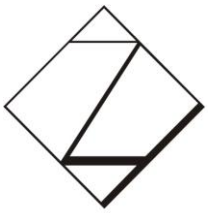

Westsächsische Hochschule Zwickau
University of Applied Sciences

Bachelorarbeit

Entwicklung einer mobilen Anwendung zur Verbindung von Recon Jet und Salesforce

Reiher, Matthias

geboren am 10. August 1994 in Zwickau

Studiengang Informatik

Westsächsische Hochschule Zwickau
Fakultät Physikalische Technik / Informatik
Fachgruppe Informatik

Betreuer, Einrichtung:	Prof. Dr. Mario Neugebauer, WHZ
	Prof. Dr.-Ing. Rainer Wasinger, WHZ
Abgabetermin:	13.02.2018

Autorenreferat

Thema dieser Bachelorarbeit ist die Entwicklung einer mobilen Anwendung mit einer Verbindung zu Salesforce. Die Anwendung bietet unterstützende Funktionen für Arbeiter, deren Aufgabe darin besteht, Artikel anhand von Arbeitsabläufen herzustellen. Es soll für jeden Arbeitsschritt eine Anleitung hinterlegt werden können und nach Abschluss des jeweiligen Schrittes soll ein Beleg im System gespeichert werden können. Diese beiden Funktionen beinhalten auch die Einbindung von Bildern und Videos. Daher muss eine weitere Einbindung eines Dienstes erfolgen, in dem die Medien synchronisiert gespeichert werden. In der Arbeit wird zunächst auf die bereits vorhandenen Teile des Systems eingegangen, dann eine Lösung für die Speicherung der Medien gefunden und die Entwicklung und der Aufbau der fertigen Anwendung beschrieben. Zum Schluss wird das Ergebnis ausgewertet und ein Ausblick auf zukünftig mögliche Erweiterungen gegeben.

Inhalt

1 Einleitung 1

1.1 Motivation 1

1.2 Die zugrundeliegende Anwendung in Android und Salesforce 2

1.3 Aufbau der Arbeit 2

2 Anforderungen an das zu entwickelnde System 3

3 Theoretische und praktische Grundlagen 5

3.1 Kommissionierung als zugrundeliegendes Modell für die bisherige Arbeit 5

3.2 Salesforce als Speicherort der Daten 5

3.3 Recon Jet als vorgesehene Plattform 6

3.4 Problem bei der Einbindung der Recon-Jet-Brille und Entscheidung für die Implementierung für Android-Smartphones 7

4 Diskussion der verschiedenen Lösungsansätze zum Speichern von Bildern und Videos 8

4.1 Lösungsansätze, die vor der Analyse ausgeschlossen wurden 8

4.1.1 Unternehmensinterner Webserver 8

4.1.2 YouTube 9

4.1.3 Vid.me 9

4.2 Lösungsansätze, die für den Vergleich in Betracht gezogen wurden 9

4.3 Anforderungen an die Plattform zum Speichern von Bildern und Videos 10

4.3.1 Vergleich hinsichtlich der kostenlosen Datenmenge 10

4.3.2 Vergleich hinsichtlich der unbegrenzten Dateigröße 10

4.3.3 Vergleich hinsichtlich der Kosten für die Nutzung mit 1TB Datenmenge 11

4.3.4 Vergleich hinsichtlich der Unterstützung einer Android-Einbindung 11

4.3.5 Entscheidung für Google Drive 12

5 Methodik 13

5.1 Bisheriges Datenmodell 13

5.2 Erweitertes Datenmodell 14
5.3 Komponenten der Anwendung 15
5.4 Menüführung und Oberfläche 16
6 Implementierung 17
6.1 Modellierung des Prozesses von der Anmeldung bis zum Hochladen eines Schrittbelegs mit Video 17
6.2 Modifikation des Layouts für die Verwendung per Smartphone 18
6.3 Aufnehmen von Bildern und Videos 21
6.4 Darstellung von Bildern und Videos innerhalb der App 24
6.5 Nutzung von Google Drive 25
6.5.1 Registrierung der App für die Verwendung von Google-Diensten 25
6.5.2 Hochladen von Bildern und Videos in Google Drive 28
6.5.3 Herunterladen von Bildern und Videos aus Google Drive 30
6.6 Nutzung von Salesforce 31
6.6.1 Erweiterung des Datenmodells in Salesforce 31
6.6.2 Autorisierung bei Salesforce 32
6.6.3 Aktualisieren von Werten in Salesforce 34
6.6.4 Anlegen neuer Datensätze in Salesforce 35
6.6.5 Aktualisierung der Teilmenge anhand eines Triggers 36
7 Auswertung und Ausblick 37
8 Literaturverzeichnis 38

Abkürzungsverzeichnis

API Application Programming Interface

CRM Customer-Relationship-Management

GPS Global Positioning System

SDK Software Development Kit

USB Universal Serial Bus

Abbildungsverzeichnis

Abbildung 1: Seitenansicht der Datenbrille Recon Jet [4] 6
Abbildung 2: Bisheriges Datenmodell aus der Arbeit von A. Kazakbaeva [1] 13
Abbildung 3: UML-Klassendiagramm für die Erweiterung des Modells.....14
Abbildung 4: Komponentendiagramm für das System.....15
Abbildung 5: Menüführung in der Android-App 16
Abbildung 6: Sequenzdiagramm für den Prozess von der Anmeldung bis zum Anlegen eines Belegs.....17
Abbildung 7: Klasse "StableArrayAdapter" in "StoreActivity" 19
Abbildung 8: Methode "initListView" der Klasse "StoreActivity" 20
Abbildung 9: Methode "onButtonPick" der Klasse "StoreActivity" 20
Abbildung 10: Registrierung der Kamera im AndroidManifest 21
Abbildung 11: Erlaubnis zum Schreiben in den externen Speicher im AndroidManifest 21
Abbildung 12: Methoden takePicture und dispatchTakePictureIntent in DriveService 22
Abbildung 13: Funktion createImageFile in DriveService.....23
Abbildung 14: Arbeitsschritt-Detail-Ansicht, MediaController am unteren Bildschirmrand ...24
Abbildung 15: Einstellungen zur Signatur im Debug-Modus.....25
Abbildung 16: Befehl zur Ausgabe aller Daten über den Debug-Schlüssel 25
Abbildung 17: Methoden "signIn" und "buildGoogleSignInClient" der Klasse "DriveService" 27
Abbildung 18: Methode "handleSignInResult" der Klasse "DriveService" 27
Abbildung 19: Generieren des Dateinamens in onActivityResult 28
Abbildung 20: Serialisierung eines Bildes in uploadImage 29

Abbildung 21: Serialisierung eines Videos in uploadVideo 29
Abbildung 22: Query für die Suche nach allen Dateien mit dem entsprechenden Namen im AppFolder.....30
Abbildung 23: Erzeugung einer lokalen Datei zum Herunterladen des Videos in openVideo .30
Abbildung 24: Schreiben der heruntergeladenen Videodaten in die lokale Datei 31
Abbildung 25: Erweitertes Datenmodell in Salesforce im SchemaBuilder.....32
Abbildung 26: Funktion zur Autorisierung bei Salesforce in SFService 33
Abbildung 27: Funktion zur Aktualisierung der Bild-ID eines Arbeitsschrittes in SFService 34
Abbildung 28: Funktion zum Anlegen eines neuen Ablaufsauftrags in SFService 35
Abbildung 29: BelegItemTrigger für das Objekt Schrittbeleg.....36

1 Einleitung

1.1 Motivation

Bei der Herstellung von Waren kommt es aufgrund von manuellem und bürokratischem Aufwand zu Verzögerungen. Wenn diese vermieden werden könnten, könnte die Effizienz gesteigert werden und in einem festgelegten Zeitraum mehr Arbeit verrichtet werden. Dabei sind in erster Linie zwei Dinge von Bedeutung: Bei der Einarbeitung eines Mitarbeiters in neue Arbeitsabläufe soll die Zeit verringert werden, welcher dieser für das Erlernen der einzelnen Arbeitsschritte benötigt. Das Öffnen einer physisch vorliegenden Anleitung, die der Mitarbeiter mit sich tragen muss, wird als nicht zeitgerecht erachtet. Außerdem soll bei der Fertigstellung der Aufwand für die Dokumentation verringert werden. Das manuelle Schreiben eines Belegs und die Übergabe an eine extra dafür angelegte Stelle, welche die Belege einordnen muss, ist ineffizient und bietet wenige Möglichkeiten zur tatsächlichen Beweisführung.

Die neue Technologie der Datenbrillen wie Recon Jet bietet Möglichkeiten, solche wirtschaftlichen Abläufe effizienter und für die Arbeiter komfortabler zu gestalten. Abgesehen von der Steuerung der Anwendung haben die Arbeiter stets beide Hände für die Arbeit zur Verfügung. Anleitungen können multimedial mit Bildern und Videos hinterlegt werden. Die Belege können ebenfalls mit Bildern und Videos aussagekräftig unterstützt werden und durch ein zentrales Speichern innerhalb einer CRM-Plattform wie Salesforce entfällt zusätzlicher Aufwand für das Organisieren der einzelnen Belege.

1.2 Die zugrundeliegende Anwendung in Android und Salesforce

Die neuen Funktionalitäten wurden als Erweiterung eines Systems geschrieben, das im Rahmen der Bachelorarbeit von A. Kazakbaeva [1] entstanden ist. Dieses System bestand aus zwei Komponenten: Einem in Salesforce hinterlegten Datenmodell und einer Android-App, mit der auf dieses zugegriffen werden konnte. Der behandelte Anwendungsfall war die Kommissionierung. Dementsprechend stehen Klassen und Methoden für die Verwaltung von zu Aufträgen zugeordneten Artikeln an verschiedenen Lagerorten zur Verfügung. In dieses Schema wurde die neue Funktionalität so eingepasst, dass bei der Herstellung von Waren die Menge von Artikeln erhöht wird, welche dann für Aufträge zur Verfügung stehen.

1.3 Aufbau der Arbeit

Zu Beginn dieser Arbeit werden die bereits bestehenden Teile des Systems beschrieben. Im nächsten Kapitel werden die Anforderungen an das zu entwickelnde System dargelegt. Im Folgenden werden die theoretischen und praktischen Grundlagen erörtert. Danach wird beschrieben, welche Software-Lösungen für die Speicherung von Bildern und insbesondere Videos in der Cloud in Betracht gezogen wurden und warum sich für die verwendete Lösung entschieden wurde. Danach wird die Implementierung detailliert beschrieben. Zum Schluss wird ein Fazit über das Projekt gezogen und ein Ausblick für mögliche Erweiterungen gegeben.

2 Anforderungen an das zu entwickelnde System

Es soll nun ein Überblick darüber gegeben werden, welche Anforderungen an das zu entwickelnde System bestehen.

1. **Die bisherigen Teile des Systems müssen erweitert werden.**

   Die neuen Funktionalitäten können nicht als eigenständige Anwendung entwickelt werden, sondern müssen auf dem bisherigen System aufbauen. Dabei handelt es sich um die vorliegende Android-App und das in Salesforce implementierte Datenmodell.

2. **Die abzubildenden Vorgänge von Produktionsabläufen mit einzelnen Schritten und entsprechenden Aufträgen mit Belegen müssen vollständig implementiert sein.**

   Die Erweiterung des Modells sieht alle diese Objekte vor. Sie müssen im Datenmodell in Salesforce implementiert werden und die Android-App muss darauf zugreifen.

3. **Die Produktionsabläufe müssen sich auf Artikel aus dem bisherigen Datenmodell beziehen und bei Abschluss eines Auftrages muss das hergestellte Teil im System hinterlegt werden.**

   Dafür muss das System automatisch erkennen, wenn ein Auftrag abgeschlossen wurde, und die Teilmenge in Salesforce aktualisieren.

4. **Für jeden Arbeitsschritt muss eine Anleitung mit optionaler Einbindung eines Bildes und eines Videos hinterlegt werden können.**

   Die Aufnahme, das Hochladen und die Darstellung der Medien muss in der Android-App möglich sein.

5. **Für jeden Beleg zum Abschluss eines Arbeitsschrittes muss eine Beschreibung mit optionaler Einbindung eines Bildes und eines Videos hinterlegt werden können.**
   Eine Einbindung in der Android-App analog zu Anforderung 4 ist erforderlich.

6. **Die in den Anforderungen 4 und 5 definierten Bilder und Videos müssen zentral online gespeichert werden.**

   Bei Aufruf der Medien ausgehend von verschiedenen Android-Geräten müssen bei gleichen Anmeldedaten die gespeicherten Bilder und Videos immer konsistent gefunden werden.

7. **Das System muss für die private Entwicklung kostenlos sein und die Kosten in der praktischen Anwendung müssen in einem angemessenen Rahmen bleiben.**

   Da kein Budget für die Entwicklung zur Verfügung steht, müssen ausschließlich für kleinere Datenmengen kostenlose Dienste genutzt werden. Die Kosten für die eingebundenen Software-Lösungen sollten bei einer praktischen Anwendung dennoch für ein Unternehmen mit unter 100 Mitarbeitern realistisch bleiben.

3 Theoretische und praktische Grundlagen

3.1 Kommissionierung als zugrundeliegendes Modell für die bisherige Arbeit

Die bisher abgebildeten wirtschaftlichen Abläufe entstanden im Rahmen des Bachelorprojekts von A. Kazakbaeva [1]. Im Mittelpunkt stand die Entwicklung einer App zum Zweck der Kommissionierung.

In seiner Dissertation „Methodik zur Planung und Steuerung der Kommissionierung in der logistischen Produktion des Versandhandels“ [2] nimmt der Autor U. Logemann Bezug auf eine Definition bezüglich der Kommissionierung: „Kommissionieren hat das Ziel, aus einer Gesamtmenge von Gütern (Sortiment) Teilmengen auf Grund von Anforderungen (Aufträge) zusammenzustellen“. Der abgebildete Ablauf entspricht einer statischen Bereitstellung der Ware. Logemann beschreibt diese wie folgt: „Bei der statischen Bereitstellung liegen die Artikelpositionen auf Lagerplätzen, zu denen sich der Kommissionierer zum Zwecke der Entnahme bewegt. Diese Variante wird umgangssprachlich "Mann-zur-Ware“ genannt“.

Dementsprechend war die bisherige App dafür vorgesehen, einen Kommissionierer dabei zu unterstützen, Waren an verschiedenen Lagerorten zu finden. Diese können anhand von Barcodes eingescannt werden. Dadurch wird dem System bestätigt, dass Bestandteile eines Kommissionierungsauftrages abgeschlossen wurden.

3.2 Salesforce als Speicherort der Daten

Wesentlicher Bestandteil der Aufgabenstellung war die Einbindung von Salesforce. Salesforce ist eine CRM-Plattform, die bereits von über 150.000 Unternehmen genutzt wird [3]. Es stehen in erster Linie Werkzeuge für die Verwaltung von Personen zur Verfügung, aber es können auch individuell eigene Objekte erstellt und die im Hintergrund eingebundene Datenbank in selbst erstellten Anwendungen verwendet werden. Diese Vielfältigkeit ist der Grund dafür, dass Salesforce bereits in der zugrundeliegenden Anwendung genutzt wurde und diese Schnittstelle im Laufe dieser Arbeit erweitert wurde.

3.3 Recon Jet als vorgesehene Plattform

Die Datenbrille Recon Jet ist die Plattform, für die die bisherige Android-App entwickelt wurde.

Abbildung 1: Seitenansicht der Datenbrille Recon Jet [4]

Der „Glas“-Teil der Brille ist ein durchsichtiges, in verschiedenen Farben erhältliches Stück Plastik. In der unteren rechten Ecke des Blickfeldes befindet sich ein 16:9 Display, dessen Auflösung vom Hersteller wie folgt beschrieben wird: „Virtual image appears as 30” HD display at 7’“ [5]. Genauere Angaben werden selbst in der Kategorie „Tech Specs“ nicht gegeben. Es sind ein 3D-Beschleunigungsmesser, ein 3D-Gyroskop, ein 3D-Magnetometer, ein Drucksensor und ein Infrarotsensor verbaut. Außerdem verfügt das Gerät unter anderem über GPS, Bluetooth, Wi-Fi und einen Micro-USB-2.0-Anschluss. Für die Steuerung steht ein kleines optisches Touchpad an der rechten Seite zur Verfügung, mit dem Gesten nach oben, unten, links und rechts gemacht werden können. Außerdem gibt es eine Bestätigen- und eine Zurück-Taste. Es sind eine Kamera sowie zwei Mikrofone zur Aufnahme von Bildern und Videos verbaut sowie ein Lautsprecher, mit dem die Audiospur eines Videos auch wieder ausgegeben werden kann. Die Batterie hält durchschnittlich 4 Stunden. Das Betriebssystem heißt ReconOS 4.4 und ermöglicht die Verbindung zu Smartphones mit Android oder iOS und kann frei genutzt und weiterentwickelt werden, weil das zugehörige SDK frei verfügbar ist [6].

3.4 Problem bei der Einbindung der Recon-Jet-Brille und Entscheidung für die Implementierung für Android-Smartphones

Bei der Erweiterung der Android-App ergab sich das Problem, dass die Herstellung einer Verbindung zwischen der Recon-Jet-Brille und Salesforce unmöglich war. Recherchen und praktische Versuche am Code haben ergeben, dass der identische Code auf einem Smartphone funktioniert, auf der Datenbrille aber zu einer Ablehnung der Anmeldedaten führt. Da das verwendete Modell offiziell nicht mehr unterstützt wird, muss von einer Inkompatibilität zwischen der Software der Brille und den neuen Versionen der Authentifizierungs-API ausgegangen werden. Es existiert kein Support mehr und das Problem konnte nicht behoben werden. Da nicht davon ausgegangen werden konnte, dass die Verbindung zu anderen Plattformen über die Brille dennoch funktioniert, wurde sich dafür entschieden, die App für den Betrieb auf gewöhnlichen Android-Smartphones umzustellen und nicht weiter für die Datenbrille zu entwickeln. So erklärt sich die Diskrepanz zwischen dem Titel dieser Arbeit und den Ausführungen und der Implementierung der folgenden Kapitel.

4 Diskussion der verschiedenen Lösungsansätze zum Speichern von Bildern und Videos

Für das Speichern und Synchronisieren von Bildern und Videos zwischen verschiedenen Geräten musste eine Lösung gefunden werden, die sich in das bestehende System einbetten lies. Hierbei wurden einige Möglichkeiten in Betracht gezogen. Es werden im Folgenden die Optionen aufgeführt und eine Vorauswahl anhand von Ausschlusskriterien getroffen. Anschließend werden Kriterien zum Vergleich aufgeführt und zum Schluss die Entscheidung für eine davon begründet.

4.1 Lösungsansätze, die vor der Analyse ausgeschlossen wurden

Die folgenden Optionen wurden bereits vor der genaueren Analyse ausgeschlossen, sollen an dieser Stelle aber dennoch erwähnt werden.

4.1.1 Unternehmensinterner Webserver

Die für die Anwendung in der Praxis naheliegendste Möglichkeit wäre die Einrichtung eines unternehmensinternen Webservers als Umsetzung des Prinzips der „Private Cloud“. Es müsste Hardware gekauft und auf dieser ein Webserver wie z. B. Apache installiert werden. Die Medien würden dann auf diesem Server gespeichert werden. Für die Anwendung im größeren Rahmen würde sich zusätzlich die Umstellung auf ein stärker ausgeprägtes verteiltes System anbieten.

Aufgrund der Komplexität des Themas musste die Einrichtung eines eigenen Servers mit zugehörigen Diensten jedoch ausgeschlossen werden, da dies nicht Teil dieses Projekts sein konnte.

4.1.2 YouTube

Bei den Überlegungen für die Online-Speicherung von Videos war YouTube aufgrund der weiten Verbreitung direkt eine der Möglichkeiten, die in Betracht gezogen wurden. Es existiert wie bei allen Google-Services auch eine API, die man in eine Android-App einbinden könnte und mit der im Hintergrund Videos hochgeladen werden könnten [7]. Dadurch würde die Plattform definitiv für die benötigte Funktionalität genügen. Es müsste nur darauf geachtet werden, dass die Videos auf privat gestellt sind, da YouTube in erster Linie zur öffentlichen Verbreitung von Videos genutzt wird und nicht dazu gedacht ist, Videos für den eigenen Gebrauch zu speichern. Außerdem sollte beachtet werden, dass Werbung auf allen Videos deaktiviert ist, da diese im Betrieb störend wäre. Ein Argument für die Nutzung von YouTube ist, dass die Plattform gänzlich kostenlos ist. Für die Speicherung von Bildern wäre jedoch eine zusätzliche Lösung nötig, da YouTube nur Videos unterstützt. Da mehrere Plattformen gefunden wurden, mit denen sowohl Bilder als auch Videos gespeichert werden können, wurde YouTube letztendlich zugunsten einer Gesamtlösung ausgeschlossen.

4.1.3 Vid.me

Ähnlich wie auch YouTube wurde Vid.me ursprünglich für die Speicherung von Videos in Betracht gezogen. Allerdings wurde die Plattform aufgrund der zu hohen Nutzungsrate und der nicht ausreichenden Serverstrukturen und der begrenzten Einnahmen am 15. Dezember 2017 eingestellt [8]. Dementsprechend hätte die Plattform ohnehin nicht mehr genutzt werden können.

4.2 Lösungsansätze, die für den Vergleich in Betracht gezogen wurden

Um nicht auf verschiedene Systeme zum Speichern von Bildern bzw. Videos zurückgreifen zu müssen, wurde sich für die Nutzung eines allgemeinen Filehosting-Dienstes entschieden. Es wurden dabei vier verschiedene Anbieter analysiert: Box [9], Dropbox [10], Google Drive [11] und OneDrive [12].

4.3 Anforderungen an die Plattform zum Speichern von Bildern und Videos

Nachdem sich dafür entschieden wurde, einen Filehosting-Dienst zu nutzen, wurde eine Reihe von Kriterien zur Bewertung und zum Vergleich der Optionen festgelegt. Die Bewertung reicht dabei von 0 für „Erfüllt die Anforderung nicht“ bis 3 für „Erfüllt die Anforderung am besten“. Die Anforderungen sind:

1. **Die Datenmenge bei kostenloser Nutzung sollte möglichst hoch sein, mindestens aber 5GB betragen.**
2. **Die Größe der Dateien sollte nicht beschränkt sein.**
3. **Der Preis bei einer professionellen Nutzung mit einer Datenmenge von einem Terabyte sollte möglichst gering sein.**
4. **Es muss eine Softwarelösung zur Einbindung in einer Android-App geben. Bei einer Spezialisierung der Plattform auf Android ist diese vorzuziehen.**

4.3.1 Vergleich hinsichtlich der kostenlosen Datenmenge

Die kostenlose Datenmenge reichte je nach Anbieter von 2GB bis zu 15GB. Die niedrigste Datenmenge mit nur 2GB stellt Dropbox zur Verfügung und fällt damit unter den Mindestwert von 5GB, weshalb für dieses Kriterium 0 Punkte vergeben werden müssen. Die weiteren Werte sind aufsteigend bei OneDrive 5GB, bei Box 10GB und bei Google Drive 15GB, weshalb in dieser Reihenfolge 1, 2 bzw. 3 Punkte vergeben werden.

4.3.2 Vergleich hinsichtlich der unbegrenzten Dateigröße

Bei der Recherche hat sich ergeben, dass Box in der kostenlosen Anwendung nur Dateigrößen bis maximal 250MB unterstützt. Da für die Größe von gewöhnlichen Videodateien 250MB nicht immer ausreichend sind, muss dieses Kriterium mit 0 Punkten bewertet werden. Die anderen drei Anbieter unterstützen beliebig große Dateien und werden daher mit 3 Punkten bewertet.

4.3.3 Vergleich hinsichtlich der Kosten für die Nutzung mit 1TB Datenmenge

Beim Vergleich hinsichtlich der Kosten für die professionelle Nutzung wurde ein Terabyte als Vergleichsgröße ausgewählt. Bei manchen Anbietern ist es üblich, generell per Nutzer abzurechnen. Da die Android-App so gestaltet ist, dass sich an jedem Gerät der gleiche Nutzer anmelden kann, soll an dieser Stelle mit nur einem Nutzer gerechnet werden.

Den niedrigsten Preis für die Nutzung mit einem Terabyte bietet OneDrive mit nur 4,20€ pro Monat pro Nutzer. Es folgen Dropbox mit 8,25€ pro Monat, Google Drive mit 9,99€ pro Monat und Box mit 12€ pro Monat pro Nutzer. Im Fall von Box ist die Speichermenge für diesen Preis bereits unbegrenzt, allerdings ist die maximale Dateigröße auf 5GB begrenzt. Die Punkte sind entsprechend verteilt.

4.3.4 Vergleich hinsichtlich der Unterstützung einer Android-Einbindung

Es stellte sich heraus, dass für jede Plattform eine Lösung zur Einbindung in Android-Apps existiert. Im Fall von Google Drive ist die Nutzung gezielt auf Android ausgelegt, wobei auch andere Schnittstellen existieren [13]. Da der Fokus auf Android liegt, bekommt Google Drive 3 Punkte. Die anderen Plattformen bekommen jeweils 2 Punkte, da es jeweils SDKs gibt, die auf der inneren API aufbauen: Die „Core API“ im Fall von Dropbox [14], die „Content API“ im Fall von Box [15] und die „OneDrive REST API“ im Fall von OneDrive [16].

4.3.5 Entscheidung für Google Drive

Die Punkte wurden addiert und miteinander verglichen. In der folgenden Tabelle wird das Ergebnis verdeutlicht:

	Box	Dropbox	Google Drive	OneDrive
Datenmenge	2	0	3	1
Dateigröße	0	3	3	3
Kosten	0	2	1	3
Schnittstelle	2	2	3	2
Gesamt	4	7	10	9

Die Plattform Box fiel aufgrund der zu kleinen Datengröße und den hohen Kosten hinter den anderen Anbietern zurück. Das größte Problem bei Dropbox stellt die zu geringe kostenlose Datenmenge dar. Die beiden besten Optionen waren Google Drive und OneDrive, wobei bei Google Drive die kostenlose Datenmenge höher war, aber bei OneDrive die Kosten für die professionelle Nutzung niedriger. Letztendlich war der entscheidende Unterschied die Spezialisierung der API von Google Drive auf Android.

5 Methodik

5.1 Bisheriges Datenmodell

Das grundlegende Datenmodell entstand im Rahmen der Bachelorarbeit von A. Kazakbaeva [1]. Um die Abläufe der Kommissionierung abzubilden, wurden 6 Klassen angelegt:

Ein *Auftrag* beinhaltet mehrere *AuftragItems*, welche als Bestandteile des Auftrags verstanden werden können. Diese Bestandteile nehmen immer Bezug auf ein *LagerTeil*, welches sich auf ein bestimmtes *Teil* bezieht, für das ein Name existiert und welches an einem bestimmten Ort gelagert wird. Da bei der Implementierung in Salesforce die beiden Klassen als eine Entität angelegt wurden, sollen sie im Folgenden nur noch als *Teil* bezeichnet werden. Die Teile werden an einem *LagerOrt* gelagert, welcher wiederum Teil eines *Lagers* ist.

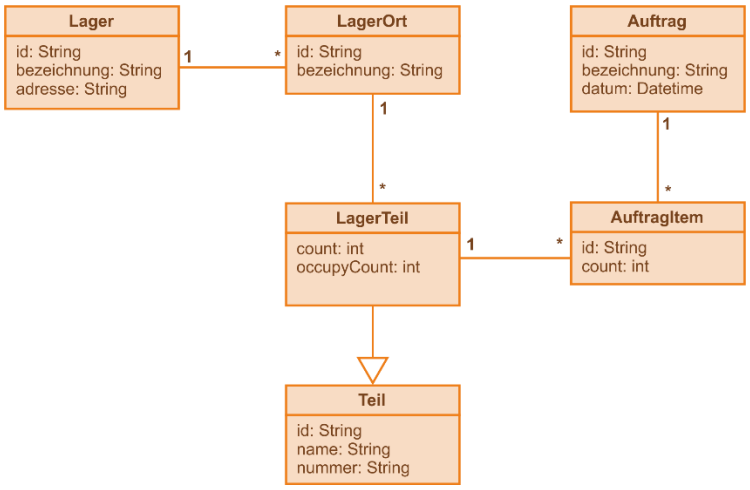

Abbildung 2: Bisheriges Datenmodell aus der Arbeit von A. Kazakbaeva [1]

5.2 Erweitertes Datenmodell

Das in Kapitel 3.1 beschriebene Modell sollte zur Implementierung der neuen Funktionen erweitert werden. Das oberste neue Datenelement ist hierbei die Klasse *Arbeitsablauf*, welche eine Abfolge von *Arbeitsschritten* abbildet. Es soll damit ein Produktionsvorgang simuliert werden, bei dem Angestellte Arbeit verrichten und dabei einen Artikel herstellen, der im richtigen Lager hinterlegt wird. Die ausführende Person bildet damit eine neue Rolle, zusätzlich zum Kommissionierer. Die so erstellen Artikel können dann vom Kommissionierer zusammengestellt und zum Erfüllen von Aufträgen genutzt werden. Zum besseren Verständnis soll für jeden Arbeitsschritt eine Anleitung in Form von Text, Bild oder Video hinterlegt werden können. Von Arbeitsabläufen sollen *Ablaufsaufträge* angelegt werden können, die auf den jeweiligen Ablauf verweisen. Bei der Erledigung eines solchen Auftrages muss die Erfüllung jedes einzelnen Schrittes in einem *Schrittbeleg* dokumentiert werden. Dieser enthält mindestens anhand seiner Existenz eine Aussage darüber, ob der Schritt schon bearbeitet wurde, kann aber auch eine Beschreibung als Text oder Beweise als Bild oder Video enthalten.

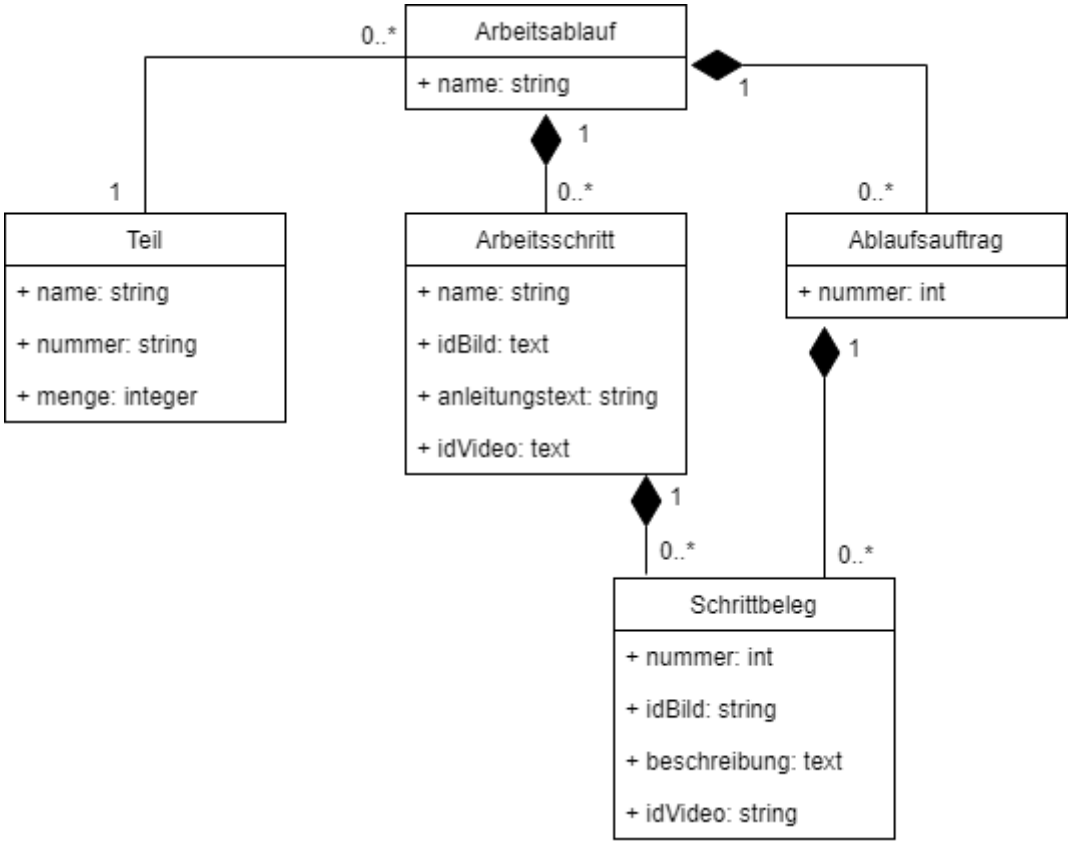

Abbildung 3: UML-Klassendiagramm für die Erweiterung des Modells

5.3 Komponenten der Anwendung

Die Android-App wurde intern in 4 Komponenten aufgeteilt, die voneinander abgegrenzte Funktionen und Inhalte darstellen und Schnittstellen anbieten. Dabei greifen die Activities bei Bedarf auf die Pakete *drive*, *sf* (kurz für Salesforce) und *data* (Klassen des Datenmodells) zu. Von außerhalb werden die Schnittstellen von Google Drive und Salesforce genutzt.

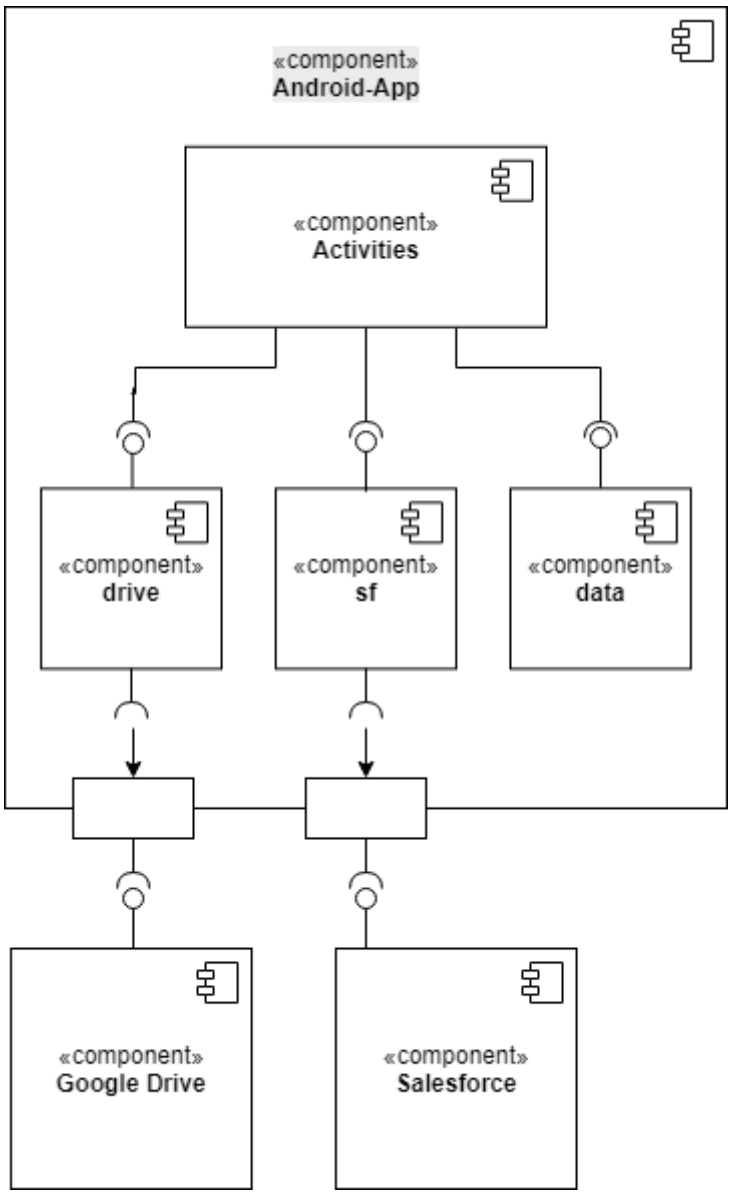

Abbildung 4: Komponentendiagramm für das System

5.4 Menüführung und Oberfläche

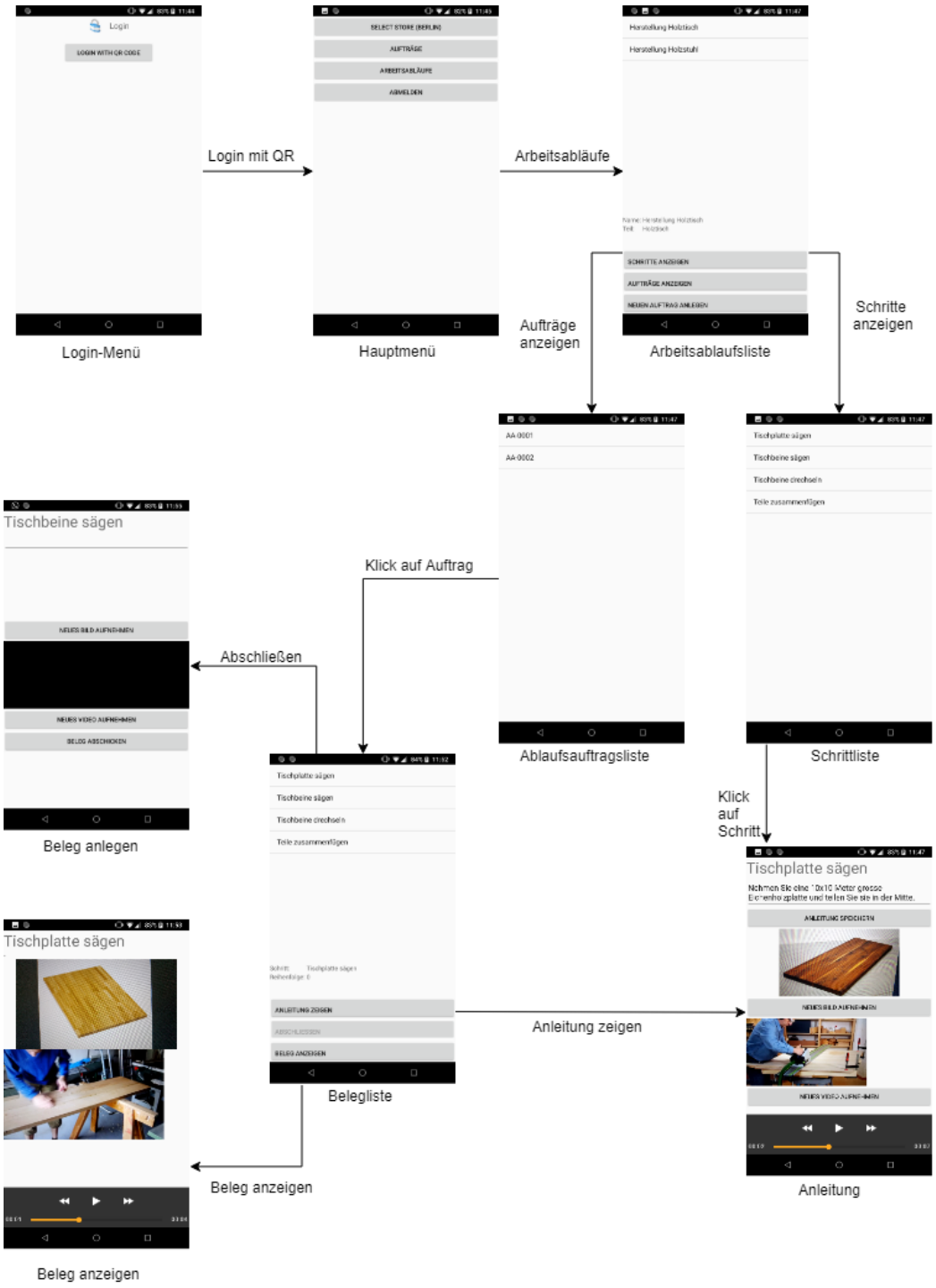

Abbildung 5: Menüführung in der Android-App

In Abbildung 5 ist die für den neuen Anwendungsfall relevante Menüführung abgebildet. Es kann, wie in Android-Apps üblich, immer über den Zurück-Button das vorherige Menü erreicht werden. Zur besseren Übersichtlichkeit wurden diese Wege im Diagramm ausgelassen. Außerdem wurden Screenshots aus der späteren Implementierung genutzt.

6 Implementierung

6.1 Modellierung des Prozesses von der Anmeldung bis zum Hochladen eines Schrittbelegs mit Video

Anhand des Beispiels „Prozess von der Anmeldung bis zum Hochladen eines Schrittbelegs mit Video“ soll an dieser Stelle verdeutlicht werden, wie die verschiedenen Teilsysteme zusammenarbeiten. Der Prozess ist in folgendem Sequenzdiagramm dargestellt:

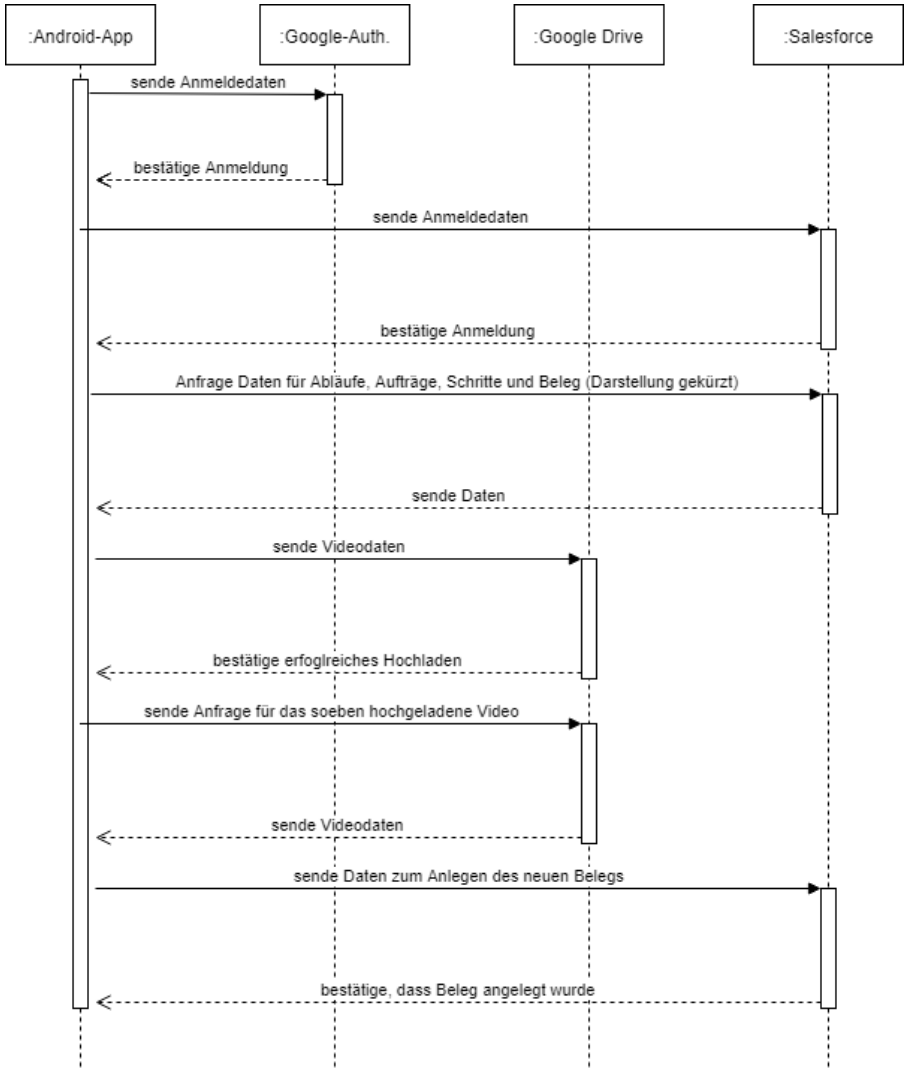

Abbildung 6: Sequenzdiagramm für den Prozess von der Anmeldung bis zum Anlegen eines Belegs

Wie aus der Abbildung hervorgeht, findet zuerst eine Autorisierung bei Google statt. Anschließend wird anhand des eingescannten QR-Codes auch die Verbindung zu Salesforce hergestellt. Beim Durchlaufen der Menüführung müssen mehrmals Datensätze aus

Salesforce abgerufen werden. Beim Speichern des Videos wird dieses zuerst hochgeladen und anschließend wieder abgerufen, um es in der App darzustellen. Zum Schluss werden alle Daten des Belegs inklusive eines Verweises auf die Video-ID in Google Drive zu Salesforce geschickt und der Beleg angelegt.

6.2 Modifikation des Layouts für die Verwendung per Smartphone

Um die App auf einem gewöhnlichen Android-Smartphone lauffähig zu machen, mussten alle Verwendungen der Recon-Jet-Bibliotheken aus dem Code entfernt werden. Da sämtliche Menüs für die Brille optimiert waren, wurde bei dieser Gelegenheit auch eine bessere Bedienbarkeit für die Verwendung mit einem berührungsempfindlichen Bildschirm beachtet.

Die meisten Activities führen mehrere Objekte der gleichen Klasse auf. Dafür genügt es, diese in einer Liste darzustellen. Als zusätzliche Funktionalität wurde in manchen Menüs eine Detailansicht am unteren Bildschirmrand hinzugefügt, die jeweils relevante Fakten über die Objekte abbildet. Da die Recon-Jet-Brille nur einen vergleichsweise kleinen Bildschirm hat und die Menü-Bibliothek keine weiteren Elemente außer den Menüeinträgen selbst vorsieht, handelt es sich hierbei um eine Erweiterung, die nur durch die Nichtverwendung der Brille ermöglicht wurde, aber bessere Bedienbarkeit gewährleistet. Für die Anzeige von Details im Fall der Brille wäre ein zusätzliches Menü denkbar, das mit einer alternativen Taste aufgerufen werden würde. Allerdings existiert an diesem Modell nur eine Taste zum Bestätigen und eine Taste für eine „Zurück/Abbrechen“-Funktion. Deshalb wäre dafür eine Verbindung nur durch ein zusätzliches Untermenü möglich, das aber die gewöhnliche Bedienung durch einen weiteren Tastendruck pro Objekt erschweren würde. Daher sei an dieser Stelle vermerkt, dass in der praktischen Anwendung die Oberfläche wieder an die Hardware angepasst werden müsste und Funktionalitäten jeweils entweder anders untergebracht oder entfernt werden müssten.

Die Implementierung des neuen Layouts erfolgte durch Nutzung der Android-Klasse „ListView“, für dessen grundlegende Anwendung ein Tutorial der vogella GmbH frei zur Verfügung steht [17].

Um auf die im ListView aufgeführten Objekte über die Oberfläche zugreifen zu können, musste eine Erweiterung der Klasse ArrayAdapter<String> erstellt werden.

```
private class StableArrayAdapter extends ArrayAdapter<String> {

    HashMap<String, Store> mIdMap = new HashMap<String, Store>();

    public StableArrayAdapter(Context context, int textViewResourceId,
                              List<String> objects) {
        super(context, textViewResourceId, objects);
        for (int i = 0; i < objects.size(); ++i) {
            mIdMap.put(objects.get(i), stores.get(i));
        }
    }

    public Store getStore(int position) {
        String item = getItem(position);
        return mIdMap.get(item);
    }

    @Override
    public boolean hasStableIds() {
        return true;
    }

}
```

Abbildung 7: Klasse "StableArrayAdapter" in "StoreActivity"

Der Adapter stellt eine Verbindung zwischen der Position innerhalb der Liste und dem jeweiligen Lager-Objekt her, über die anschließend die ID und die Adresse des jeweiligen Lagers bestimmt werden kann. Die Einbindung des Adapters erfolgt bei der Erstellung des ListViews.

```
private void initListView(){
    final ListView listview = findViewById(R.id.listview);

    final ArrayList<String> list = new ArrayList<>();
    for (int i = 0; i < stores.size(); ++i) {
        list.add(stores.get(i).getName());
    }
    final StableArrayAdapter adapter = new StableArrayAdapter(this,
            android.R.layout.simple_list_item_1, list);
    listview.setAdapter(adapter);

    listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> parent, final View view,
                                int position, long id) {
            selectedStore=adapter.getStore(position);
            TextView storeName = findViewById(R.id.storename);
            TextView storeAddress = findViewById(R.id.storeaddress);
            storeName.setText(selectedStore.getName());
            storeAddress.setText(selectedStore.getAdresse());
        }

    });
}
```

Abbildung 8: Methode "initListView" der Klasse "StoreActivity"

Um die Namen der jeweiligen Lager als Beschriftung der Einträge zu nutzen, muss aus ihnen zuerst eine neue Liste erstellt werden, die an den Adapter übergeben wird. Innerhalb der „onItemClick“-Methode wird das Lager über den Adapter ermittelt und Daten wie der Name und die Adresse innerhalb der Detailansicht am unteren Bildschirmrand ausgegeben. Die Auswahl des Lagers zur weiteren Verwendung funktioniert über den „Auswählen“-Button.

```
public void onButtonPick(View v) {
    Context context = getBaseContext();
    if(selectedStore!=null) {
        SFSession.setStoreId(context, selectedStore.getId());
        SFSession.setStoreName(context, selectedStore.getName());
        context.startActivity(new Intent(context, MainActivity.class));
    }else{
        Toast.makeText(context,"Bitte wählen Sie ein Lager aus!",
Toast.LENGTH_SHORT).show();
    }
}
```

Abbildung 9: Methode "onButtonPick" der Klasse "StoreActivity"

Bei der Bestätigung für die Auswahl des aktuellen Lagers werden die ID und der Name innerhalb der Salesforce-Session-Klasse gespeichert und das Hauptmenü geöffnet. Falls noch kein Lager ausgewählt wurde, wird der Nutzer gebeten, dies zu tun.

6.3 Aufnehmen von Bildern und Videos

Zentraler Bestandteil der Aufgabenstellung war das Einbinden von Bildern und Videos. Die grundlegende Funktionalität konnte dabei von einem offiziellen Tutorial für Fotos [18] und Videos [19] übernommen werden, aber um die Methoden richtig nutzen zu können, waren zahlreiche Anpassungen nötig.

Um die Kamera in einer Android-App nutzen zu können, muss dafür erst das nötige Feature zum Manifest hinzugefügt werden:

```
<uses-feature android:name="android.hardware.camera"
    android:required="true" />
```

Abbildung 10: Registrierung der Kamera im AndroidManifest

Dadurch ist die Verwendung für das Aufnehmen von sowohl Bildern als auch Videos möglich. Für Android-Versionen bis einschließlich 4.3 muss zum Speichern der Bilder zusätzlich folgender Zusatz eingefügt werden:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
    android:maxSdkVersion="18" />
```

Abbildung 11: Erlaubnis zum Schreiben in den externen Speicher im AndroidManifest

Für den Aufruf von allen Activities, in denen Bilder und Videos aufgenommen werden können, stellt die Klasse DriveService die öffentliche Methode „takePicture“ zur Verfügung, die durch Aufruf von privaten Methoden eine neue Datei anlegt und das jeweilige Medium darin speichert. Auch wenn die Funktionalität zum bloßen Aufnehmen noch nicht auf die Google-Drive-API zurückgreift, wurde sich trotzdem für eine Integrierung in diese Klasse entschieden, da der weitere Datenfluss sich immer auf ein anschließendes Hochladen in Google Drive und generell auch ein direktes Herunterladen von Google Drive bezieht. Eine Auslagerung in eine zusätzliche Klasse wurde daher als nicht zielführend erachtet.

Die Funktion zum Aufnehmen eines Bildes sieht folgendermaßen aus:

```
public static void takePicture(Activity activity){
    dispatchTakePictureIntent(activity);
}

private static void dispatchTakePictureIntent(Activity activity) {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    // Ensure that there's a camera activity to handle the intent
    if (takePictureIntent.resolveActivity(activity.getPackageManager()) !=
null) {
        // Create the File where the photo should go
        File photoFile = null;
        try {
            photoFile = createImageFile(activity);
        } catch (IOException ex) {
            // Error occurred while creating the File
            Log.e(FILETAG, "Error creating the file!");
        }
        // Continue only if the File was successfully created
        if (photoFile != null) {
            Log.v(FILETAG, "File created!");
            Uri photoURI = FileProvider.getUriForFile(activity,
                    "de.whz.tasksforce.fileprovider",
                    photoFile);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            activity.startActivityForResult(takePictureIntent,
REQUEST_TAKE_PHOTO);
        }
    }
}
```

Abbildung 12: Methoden takePicture und dispatchTakePictureIntent in DriveService

Es fällt auf, dass die erste Funktion nur die zweite aufruft. Dafür wurde sich entschieden, weil die private innere Funktion den Ablauf genauer beschreibt, aber dies für die Sicht von außen nicht notwendig war. Für die Activities ist es jeweils nur wichtig, dass ein Bild aufgenommen wird, nicht wie das geschieht.

Der Aufruf zum Aufnehmen des Bildes besteht in erster Linie aus dem Anlegen eines neuen Intents mit der Aktion „MediaStore.ACTION_IMAGE_CAPTURE“. Für diesen wird eine temporäre Datei angelegt und dann über die aufrufende Activity eine neue Activity gestartet, auf dessen Ergebnis wiederum in der aufrufenden Activity gewartet wird. Das Anlegen der Datei erfolgt in der Funktion createImageFile (siehe Abbildung 13).

```
private static File createImageFile(Activity activity) throws IOException {
    // Create an image file name
    String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
    currentImageFileName = "JPEG_" + timeStamp + "_";
    File storageDir = activity.getExternalFilesDir(Environment.DIREC-
TORY_PICTURES);
    File image = File.createTempFile(
            currentImageFileName,  /* prefix */
            ".jpg",         /* suffix */
            storageDir      /* directory */
    );

    // Save a file: path for use with ACTION_VIEW intents
    currentPhotoPath = image.getAbsolutePath();
    return image;
}
```

Abbildung 13: Funktion createImageFile in DriveService

Um die Einmaligkeit des Namens zu gewährleisten, wird der Dateiname mit einem Zeitstempel versehen, der bis auf die Sekunde genau ist. Über die Methode File.createTempFile wird die Datei angelegt und zu einem späteren Zeitpunkt wieder gelöscht, damit der Speicher nicht mit Bildern gefüllt wird, auf die später nie wieder zugegriffen wird. Für ein anschließendes Hochladen des Bildes wird der Dateipfad in einer privaten Variable gespeichert.

An dieser Stelle ist der Prozess des Aufnehmens abgeschlossen. Die gezeigten Funktionen existieren analog auch für Videos. Durch die Ähnlichkeit soll in diesem Fall auf weitere Details verzichtet werden.

6.4 Darstellung von Bildern und Videos innerhalb der App

Für die Darstellung der Bilder und Videos innerhalb der App wurden die Standardobjekte ImageView und VideoView verwendet. Zur Steuerung des Videos wird zusätzlich noch ein MediaController zur Laufzeit eingefügt, über den das Video gestartet und pausiert sowie der Zeitpunkt ausgewählt werden kann.

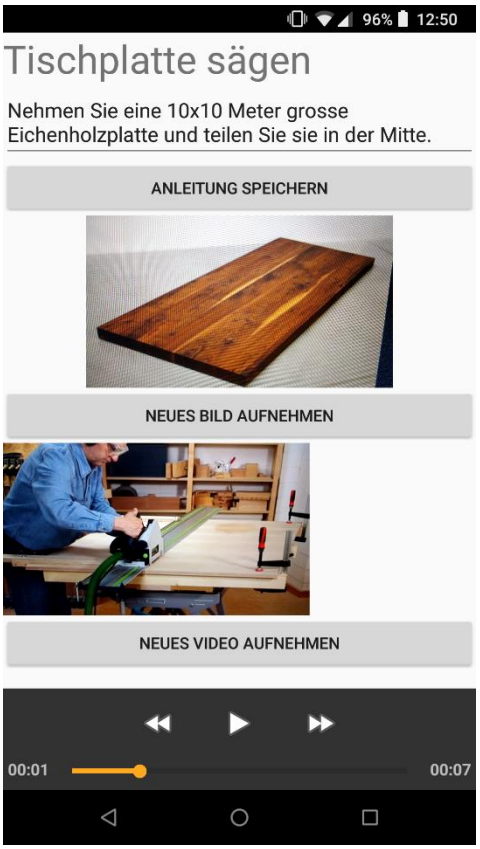

Abbildung 14: Arbeitsschritt-Detail-Ansicht, MediaController am unteren Bildschirmrand

In Bezug auf alle Oberflächen zur Darstellung von Bildern und Videos ist noch anzumerken, dass die Ladezeiten teilweise länger als erwünscht sind. So beträgt die Ladezeit für ein Bild manchmal über 5 Sekunden, was mit der BitmapFactory-Klasse zusammenzuhängen scheint, da die Videos durchschnittlich schneller geladen werden. Für den Nutzer ist dabei manchmal nicht ersichtlich, dass die App noch lädt. Bei Aufnahme von Bildern und Videos im Panaroma-Modus kommt erschwerend dazu, dass das Layout nicht rechtzeitig geladen wird, weil die Anwendung mit dem Laden beschäftigt ist. Es wurde sich jedoch dafür entschieden, die nicht optimale Oberfläche vorerst zu belassen, da sie erstens die Funktionalität nicht einschränkt und zweitens eine umfassendere Nutzung von Ladebildschirmen die verschiedenen Pakete sehr miteinander vernetzen würde, was dem Entwurf widerspricht.

6.5 Nutzung von Google Drive

6.5.1 Registrierung der App für die Verwendung von Google-Diensten

Um die Verwendung von Google Drive zum Speichern und Laden von Bildern und Videos zu ermöglichen, mussten zunächst einige Vorbereitungen getroffen werden. Google bietet hierfür ein speziell auf Google Drive zugeschnittenes Tutorial an [20].

Zuerst wurde in der „Google API Console“ ein neues Projekt für die Android-App angelegt. Für dieses Projekt wurde dann eine OAuth-Client-ID erstellt. Diese enthält den Package-Namen „de.whz.tasksforce“ der App sowie den SHA1-Fingerprint des Zertifikats, mit dem die App bei der Erstellung versehen wird. Um dieses Zertifikat einzubinden, waren wiederum zwei Schritte notwendig.

In der „build.gradle“-Datei der App musste eingestellt werden, dass die Debug-Version automatisch mit dem üblichen Android-Debug-Key zertifiziert wird. Dafür mussten folgende Zeilen eingefügt werden:

```
signingConfigs {
    debug {
        keyAlias 'androiddebugkey'
        keyPassword 'android'
        storeFile file('C:/Users/[Nutzername]/.android/debug.keystore')
        storePassword 'android'
    }
}
```

Abbildung 15: Einstellungen zur Signatur im Debug-Modus

Alle Daten des Schlüssels sind bei Nutzung des „debug.keystore“ vorgegeben und müssen nicht selbst festgelegt werden.

Anschließend musste mithilfe des Android-Keytools der SHA1-Fingerprint des verwendeten Schlüssels ermittelt werden. Das funktioniert mithilfe des folgenden Befehls:

```
keytool -exportcert -alias androiddebugkey -keystore path -list -v
```

Abbildung 16: Befehl zur Ausgabe aller Daten über den Debug-Schlüssel

Aus den angezeigten Informationen ließ sich der Wert herauslesen und konnte zur Erstellung der OAuth-Client-ID verwendet werden.

Es sei an dieser Stelle noch angemerkt, dass für eine veröffentlichte Release-Version der App auch ein entsprechender Schlüssel selbst generiert werden müsste, was aber für die Entwicklung noch nicht relevant war.

Bei der Verwendung des Codes, der im Google-Drive-Tutorial vorgeschlagen wurde, ergaben sich einige Probleme, weshalb noch ein zusätzlicher Google-Dienst in Anspruch genommen werden musste. Dabei handelt es sich um „Google Sign-In“, welches das Verbinden einer selbst geschriebenen App mit Google ermöglicht. Sign-In stellt eine Bibliothek zur Verfügung, die unter anderem Befehle zur Anmeldung enthält, wobei die bereits im jeweiligen Smartphone gespeicherten Google-Konten genutzt werden. Beim ersten Mal muss noch der Account gewählt und die Verwendung autorisiert werden, danach findet der Prozess im Hintergrund statt. Auch dafür stellt Google wieder ein Tutorial zur Verfügung [21].

Als erstes musste die Anwendung auch für Sign-In registriert werden. Genau wie bei der Registrierung für Drive mussten dafür der Package-Name und der SHA1-Fingerprint des Zertifikats angegeben werden. Daraus wurde eine Konfigurations-Datei erstellt, die heruntergeladen und im Dateisystem der App integriert wurde. Um die Bibliothek nutzen zu können, mussten in Gradle die Bibliothek und das Plugin des Pakets „com.google.gms:google-services“ hinzugefügt werden.

Für die Zentralisierung aller in der App vorkommenden Zugriffe auf die Google-Drive-API sowie des Anmeldeprozesses zu Beginn innerhalb einer Klasse wurde das Paket „drive“ mit der enthaltenen Klasse „DriveService“ erstellt. Ähnlich wie in „SFService“ werden darin ausschließlich statische Methoden zur Verfügung gestellt, die von anderen Klassen genutzt werden. Beim Starten der App wird automatisch die Funktion „signIn“ aufgerufen (siehe Abbildung 17).

```
public static void signIn(Activity activity){
    GoogleSignInClient mGoogleSignInClient = buildGoogleSignInClient(activ-
ity);

    Intent signInIntent = mGoogleSignInClient.getSignInIntent();
    GoogleSignInAccount account = GoogleSignIn.getLastSignedInAccount(ac-
tivity);
    if(account!=null)
        Log.w(DriveService.TAG, "Zuletzt angemeldeter Nutzer: " + ac-
count.getDisplayName());
    activity.startActivityForResult(signInIntent, RC_SIGN_IN);
}

private static GoogleSignInClient buildGoogleSignInClient(Activity activ-
ity) {
    GoogleSignInOptions signInOptions =
            new GoogleSignInOptions.Builder(GoogleSignInOptions.DE-
FAULT_SIGN_IN)
                    .requestScopes(Drive.SCOPE_APPFOLDER)
                    .build();
    return GoogleSignIn.getClient(activity, signInOptions);
}
```

Abbildung 17: Methoden "signIn" und "buildGoogleSignInClient" der Klasse "DriveService"

Innerhalb dieser beiden Methoden werden alle Parameter für das Starten des Anmeldeprozesses festgelegt und bei der aufrufenden Activity anschließend der Prozess gestartet. Dabei wird auch geprüft, ob bereits ein Nutzer angemeldet ist und gegebenenfalls dessen Name über die Konsole ausgegeben. Besondere Bedeutung hat die Festlegung auf den „SCOPE_APPFOLDER“. Dadurch kann die App nur auf alle durch sie selbst angelegten Daten zugreifen und alle anderen Daten im angemeldeten Google-Account sind sowohl vor lesendem als auch schreibendem Zugriff geschützt. Sobald das Ergebnis des Anmeldeversuches eintrifft, wird folgende Funktion aufgerufen:

```
public static void handleSignInResult(Task<GoogleSignInAccount> complet-
edTask, Activity activity) {
    try {
        googleSignInAccount = completedTask.getResult(ApiException.class);
        Log.w(TAG, "signInResult:success");
    } catch (ApiException e) {
        // The ApiException status code indicates the detailed failure rea-
son.
        // Please refer to the GoogleSignInStatusCodes class reference for
more information.
        Log.w(TAG, "signInResult:failed code=" + e.getStatusCode());
        signIn(activity);
    }
}
```

Abbildung 18: Methode "handleSignInResult" der Klasse "DriveService"

Dadurch werden bei erfolgreicher Anmeldung die benötigten Daten über das angemeldete Google-Konto für die Laufzeit des Programms gespeichert. Bei einem Fehlschlag wird eine Fehlermeldung mit einem zuordenbaren Fehlercode ausgegeben und der Anmeldeprozess neu gestartet.

6.5.2 Hochladen von Bildern und Videos in Google Drive

Das Hochladen von Bildern und Videos wird immer direkt nach dem Aufnehmen initiiert. In der Methode „onActivityResult“ der aktuellen Activity wird geprüft, welcher Request-Code vorliegt und ob das Ergebnis in Ordnung ist. Anschließend wird die Methode „uploadImage“ bzw. „uploadVideo“ in DriveService aufgerufen. Als Parameter wird dabei der Name der anzulegenden Datei übergeben, der einen Zeitstempel beinhaltet:

```
String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
String ImageId = "AS-" + timeStamp;
```

Abbildung 19: Generieren des Dateinamens in onActivityResult

Das ist nötig, weil die Dateinamen in Google Drive nicht eindeutig sind, d.h. mehrere Dateien im gleichen Ordner können den gleichen Namen haben. In der Dokumentation schreibt Google in Bezug auf den Dateinamen: „The name of the file. This is not necessarily unique within a folder. Note that for immutable items such as the top level folders of Team Drives, My Drive root folder, and Application Data folder the name is constant.” [22]. Um das Herunterladen der richtigen Datei zu garantieren, muss also jede Datei einen eigenen Namen besitzen. Der Zeitstempel, welcher auf Sekunden genau aufgenommen wird, wurde hier als ausreichend erachtet. In der Praxis könnte allerdings bei der Verteilung auf viele Geräte eine noch genauere Bezeichnung nötig werden.

Die Methode zum Hochladen von Dateien in den Appfolder bezieht sich auf das offizielle Google-Tutorial zu diesem Thema [23]. Der Code musste jedoch bezüglich des Dateityps angepasst werden. Die Serialisierung findet innerhalb dieser Zeilen statt:

```
ByteArrayOutputStream bitmapStream = new ByteArrayOutputStream();
Bitmap bitmap = BitmapFactory.decodeFile(currentPhotoPath);
bitmap.compress(Bitmap.CompressFormat.PNG, 100, bitmapStream);
try {
    outputStream.write(bitmapStream.toByteArray());
} catch (IOException e1) {
    Log.i("ERROR", "Unable to write file contents.");
}

MetadataChangeSet changeSet = new MetadataChangeSet.Builder()
        .setTitle(name)
        .setMimeType("image/jpg")
        .setStarred(true)
        .build();
```

Abbildung 20: Serialisierung eines Bildes in uploadImage

Mithilfe der Klasse BitmapFactory wird das zuvor temporär lokal gespeicherte Bild dekodiert und komprimiert und in den OutputStream geschrieben. Dadurch gelangen die Daten des Bildes auf Google Drive. Nach Abschluss der Methode ist das Bild dauerhaft abrufbar in Google Drive gespeichert. Die aufrufende Activity speichert anschließend noch die ID in Salesforce, worauf in Kapitel 6.6.3 genauer eingegangen wird.

Beim Hochladen von Videos läuft der Code etwas anders ab. In diesem Fall wird ein FileInputStream für die lokal gespeicherte Datei geöffnet und der gesamte Inhalt in einen Buffer geschrieben. Anhand dieses Buffers werden dann die Daten hochgeladen.

```
File video = new File(currentVideoPath);
FileInputStream videoStream = new FileInputStream(video);
byte[] buf = new byte[(int)video.length()];
videoStream.read(buf);
Log.e(TAG, String.valueOf(buf.length));
try {
    outputStream.write(buf, 0, buf.length);
} catch (IOException e1) {
    Log.i("ERROR", "Unable to write file contents.");
}
```

Abbildung 21: Serialisierung eines Videos in uploadVideo

6.5.3 Herunterladen von Bildern und Videos aus Google Drive

Für das Herunterladen von Bildern und Videos werden die Methoden downloadImage bzw. downloadVideo in DriveService aufgerufen. Um die DriveId der richtigen Datei zu erhalten, muss erst eine Abfrage gestartet werden, die nach dem entsprechenden Titel im AppFolder sucht:

```
final Query query = new Query.Builder()
        .addFilter(Filters.eq(SearchableField.TITLE, name))
        .build();
final Task<DriveFolder> appFolderTask = resourceClient.getAppFolder();
[…]
return resourceClient.queryChildren(appFolderTask.getResult(), query);
```

Abbildung 22: Query für die Suche nach allen Dateien mit dem entsprechenden Namen im AppFolder

Für gewöhnlich sollte diese Abfrage eine Menge von Metadaten mit genau einem Eintrag zurückgeben. Von diesem Eintrag wird dann die DriveId ermittelt und die Methode zum Öffnen dieser Datei gestartet.

Der Code zum Öffnen von Dateien orientiert sich am generellen Google-Tutorial für den Umgang mit Dateien in Drive [24]. Im Fall von Bildern ist die Methode trivial und besteht nur aus der erneuten Dekodierung der Daten mittels BitmapFactory. Bei Videos muss allerdings für das Anzeigen von Videos eine temporäre lokale Datei erzeugt werden, in der die Daten für die Laufzeit des Programmes gespeichert sind. Der Titel enthält zur eindeutigen Identifizierung wieder einen Zeitstempel:

```
File storageDir = activity.getExternalFilesDir(Environment.DIRECTORY_MOV-
IES);
String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
String videoName = "VD-" + timeStamp;
File video = File.createTempFile(
        videoName,  /* prefix */
        ".mp4",         /* suffix */
        storageDir      /* directory */
);
absVideoPath = video.getAbsolutePath();
```

Abbildung 23: Erzeugung einer lokalen Datei zum Herunterladen des Videos in openVideo

Da die Funktion createTempFile auch zusätzlich den Dateinamen beeinflusst, wird für das anschließende Anzeigen des Videos der absolute Pfad gespeichert. Die Daten werden dann aus dem InputStream in einen Puffer ausgelesen und direkt wieder in einen FileOutputStream für die neue Datei geschrieben:

```
try (OutputStream output = new FileOutputStream(video)) {
    byte[] buffer = new byte[4 * 1024];
    int read;

    InputStream contentsInput = contents.getInputStream();

    while ((read = contentsInput.read(buffer)) != -1) {
        output.write(buffer, 0, read);
    }
    output.flush();
}catch(Exception e){
    Log.e(TAG, e.toString());
}
```

Abbildung 24: Schreiben der heruntergeladenen Videodaten in die lokale Datei

6.6 Nutzung von Salesforce

6.6.1 Erweiterung des Datenmodells in Salesforce

Da die gesamten Daten mit Ausnahme der Bilder und Videos in Salesforce gespeichert werden, musste das vorhandene Datenmodell erweitert werden. Dazu wurden die Objekte Arbeitsablauf, Arbeitsschritt, Ablaufsauftrag und Schrittbeleg angelegt.

Im Arbeitsablauf ist hinterlegt, welches Teil im Laufe der Produktion hergestellt wird. Diese Information wird später beim Anlegen des letzten Schrittbelegs benötigt, um die Menge zu erhöhen. Die Beziehungen der neuen Objekte untereinander wurden als Master-Detail-Relationships angelegt, weil bei Entfernen des jeweiligen Arbeitsablaufs die entsprechenden Aufträge und Arbeitsschritte und dadurch auch die Belege an Bedeutung verlieren. Alle Daten beziehen sich immer auf einen bestimmten Arbeitsablauf. Die Verweise auf Bilder und Videos sind als Text-Variablen der Länge 30 hinterlegt und bezeichnen den jeweiligen Titel in Google Drive. Beim Einfügen von Anleitungen und Beschreibungen über die App fiel auf, dass die übermittelten Strings keine deutschen Sonderzeichen enthalten dürfen, d. h. im Fall von

Zeichen wie „ß“ wurde eine Fehlermeldung für die Umwandlung zurückgegeben. Dies wurde mit einer Information an den Nutzer behandelt, in der darüber informiert wird, dass nur ASCII-Zeichen zulässig sind.

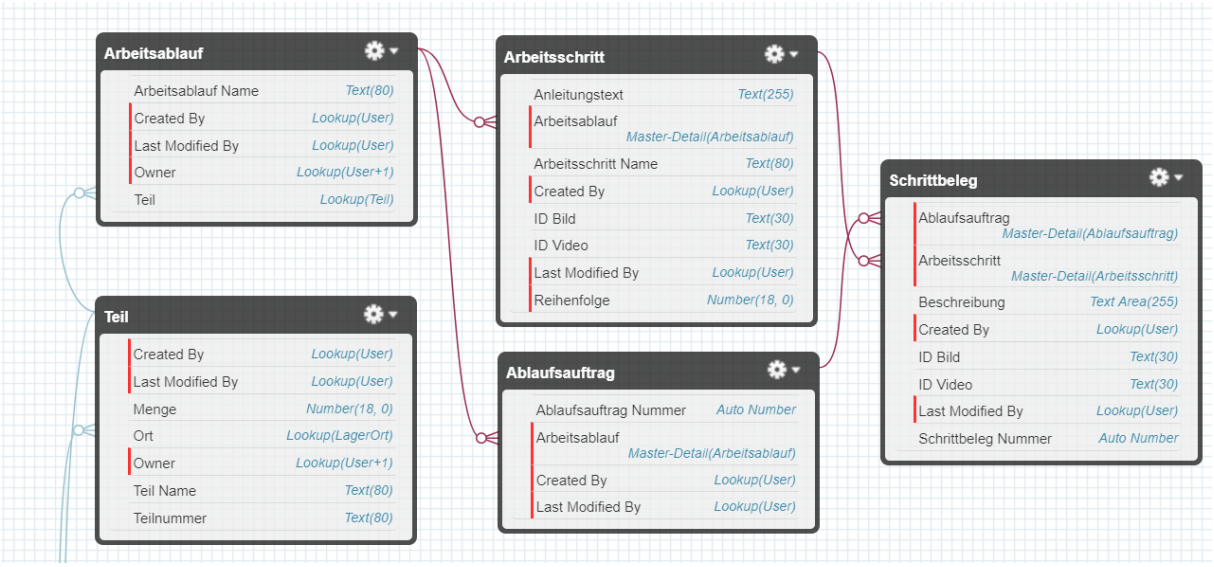

Abbildung 25: Erweitertes Datenmodell in Salesforce im SchemaBuilder

6.6.2 Autorisierung bei Salesforce

Um innerhalb der App auf die Salesforce-Daten zugreifen zu können, musste sich erst über OAuth autorisiert werden. Der folgende Code-Ausschnitt (Abbildung 26) zeigt diesen Vorgang. Es handelt sich um die gleiche Funktion, die auch Kazakbaeva in ihrer Arbeit genutzt hat [1]. Es sei an dieser Stelle nochmals angemerkt, dass es sich bei dieser Funktion um den Grund dafür handelt, dass das Projekt nicht mit Recon Jet fortgesetzt werden konnte: Der identische Code führte auf der Brille zu einer Abweisung, während er auf dem Smartphone durchlief. Da die Ursache dafür nicht gefunden werden konnte und auch keine Alternative zur Verfügung stand, musste die Verwendung der Datenbrille aufgegeben werden.

```
public static synchronized void init(final Context context, String login,
String password, final ActionCallback callback) {
    RequestParams params = new RequestParams();
    params.put("grant_type", "password");
    params.put("client_id", API_KEY);
    params.put("client_secret", SECRET_KEY);
    params.put("username", login);
    params.put("password", password);

    AsyncHttpClient client = new AsyncHttpClient();
    client.post("https://login.salesforce.com/services/oauth2/token",
params, new JsonHttpResponseHandler() {
        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject
response) {
            […]
            try {
                accessToken = response.getString("access_token");
                instanceUrl = response.getString("instance_url");
                tokenType = response.getString("token_type");
                signature = response.getString("signature");
            } catch (JSONException e) {
                callback.onFailure("Error by parsing...");
                return;
            }
         […]
        }

        @Override
        public void onFailure(int statusCode, Header[] headers, Throwable
throwable, JSONObject response) {
            […]
        }
    });
}
```

Abbildung 26: Funktion zur Autorisierung bei Salesforce in SFService

Die Nutzerdaten (E-Mail-Adresse und Passwort) werden nach wie vor als QR-Code eingescannt und an die Funktion übergeben.

6.6.3 Aktualisieren von Werten in Salesforce

Um die Änderung von Werten für die Anleitung als Text und die Titel der Bilder und Videos bei Arbeitsschritten zu ermöglichen, mussten Update-Funktionen in der App geschrieben werden. Diese funktionieren über die PATCH-Funktion von AsyncHttpClient, in der Werte für die zu ändernden Spalten angegeben sind. Der Name der Tabelle und die ID des entsprechenden Datensatzes sind in der aufgerufenen URL enthalten. Der Aufruf soll anhand der Funktion für die Bild-ID gezeigt werden:

```
public static synchronized void sendImageIdWorkStep(Context context, String
workStepId, String ImageId, final ActionCallback callback) {
    JSONObject object = new JSONObject();
    StringEntity entity;

    try {
        object.put("ID_Bild__c", ImageId);
        entity = new StringEntity(object.toString());
    } catch (JSONException | UnsupportedEncodingException e) {
        return;
    }

    String authorization = String.format("%s %s", SFSession.getToken-
Type(context), SFSession.getAccessToken(context));
    String url = String.format("%s/services/data/v26.0/sobjects/Ar-
beitsschritt__c/%s", SFSession.getInstanceUrl(context), workStepId);

    AsyncHttpClient client = new AsyncHttpClient();
    client.addHeader("Authorization", authorization);
    client.patch(context, url, entity, "application/json", new JsonHttpRe-
sponseHandler() { […] });
}
```

Abbildung 27: Funktion zur Aktualisierung der Bild-ID eines Arbeitsschrittes in SFService

6.6.4 Anlegen neuer Datensätze in Salesforce

Da über die App auch neue Aufträge für die verschiedenen Abläufe angelegt werden sollen und dafür zugehörige Belege hinterlegt werden, mussten auch Funktionen zum Anlegen neuer Datensätze in Salesforce geschrieben werden. Dies funktioniert analog zum Ändern von Daten mit dem Unterschied, dass die POST-Funktion genutzt wird und auf das Verzeichnis, statt auf den Datensatz, in der URL verwiesen wird. Es folgt ein Beispiel für das Anlegen eines neuen Ablaufsauftrags.

```
public static synchronized void createWorkOrder(Context context, String
workflowId, final ActionCallback callback) {
    JSONObject object = new JSONObject();
    StringEntity entity;

    try {
        object.put("Arbeitsablauf__c", workflowId);
        entity = new StringEntity(object.toString());
    } catch (JSONException | UnsupportedEncodingException e) {
        return;
    }

    String authorization = String.format("%s %s", SFSession.getToken-
Type(context), SFSession.getAccessToken(context));
    String url = String.format("%s/services/data/v26.0/sobjects/Ab-
laufsauftrag__c/", SFSession.getInstanceUrl(context));

    AsyncHttpClient client = new AsyncHttpClient();
    client.addHeader("Authorization", authorization);
    client.post(context, url, entity, "application/json", new JsonHttpRe-
sponseHandler() { […] });
}
```

Abbildung 28: Funktion zum Anlegen eines neuen Ablaufsauftrags in SFService

6.6.5 Aktualisierung der Teilmenge anhand eines Triggers

Bei Abschluss eines Ablaufsauftrags soll die Menge des hergestellten Teils um 1 erhöht werden. Der Abschluss findet in dem Moment statt, in dem der Schrittbeleg für den letzten Arbeitsschritt angelegt wird. Damit die Funktionalität sowohl bei einem Anlegen in Salesforce als auch über die App gewährleistet ist, musste die Logik als Trigger in Salesforce angelegt werden.

```
trigger BelegItemTrigger on Schrittbeleg__c (after insert) {
    if (Trigger.isInsert) {
       for (Schrittbeleg__c beleg: Trigger.new) {

          Arbeitsschritt__c schritt;
          decimal reihenfolge;
          Arbeitsablauf__c ablauf;

          schritt = [SELECT Reihenfolge__c, Arbeitsablauf__c FROM Arbeitsschritt__c WHERE Id = :beleg.Arbeitss
chritt__c];
          reihenfolge = schritt.Reihenfolge__c;

          List<Arbeitsschritt__c> steps = [SELECT Name FROM Arbeitsschritt__c WHERE (Arbeitsablauf__c = :sch
ritt.Arbeitsablauf__c)
                                    AND (Reihenfolge__c = :reihenfolge + 1)];

          if(steps.isEmpty()){
             ablauf = [SELECT Teil__c FROM Arbeitsablauf__c WHERE Id = :schritt.Arbeitsablauf__c];
             Teil__c teil = [SELECT Menge__c FROM Teil__c WHERE Id = :ablauf.Teil__c];
             decimal mengeAlt = teil.Menge__c;
             teil.Menge__c = mengeAlt + 1;
             update teil;
          }
       }
    }
}
```

Abbildung 29: BelegItemTrigger für das Objekt Schrittbeleg

Nach dem Einfügen eines Datensatzes des Typs Schrittbeleg wird geprüft, ob noch ein weiterer Arbeitsschritt für den Ablauf existiert, der zeitlich nach dem aktuellen Schritt stattfindet. Falls dies nicht der Fall ist, ist der Ablauf beendet und die Menge des hergestellten Teils kann als um 1 erhöht hinterlegt werden.

7 Auswertung und Ausblick

Das System konnte allen Anforderungen entsprechend weiterentwickelt werden. Auch wenn die Anwendung nicht mehr mit Recon Jet kompatibel ist, stellt sie dennoch ein hilfreiches Werkzeug für eine Anwendung in der Wirtschaft dar.

Es gibt weitere Anwendungsfälle, die man in Zukunft im System integrieren könnte: Zum Beispiel könnte die Spezialisierung von Salesforce auf CRM dazu genutzt werden, auch eine Kundenverwaltung in der App zu enthalten und so über Kontakte Aufträge zu generieren. Außerdem könnte nach dem Abschließen von Aufträgen eine Abrechnung der Kosten erfolgen.

8 Literaturverzeichnis

[1] A. Kazakbaeva, Entwicklung einer Datenbrillenanwendung für verschiedene Einsatzgebiete mit Anbindung an ein betriebliches Informationssystem, Zwickau: Westsächsische Hochschule Zwickau, 2016.

[2] U. Logemann, Methodik zur Planung und Steuerung der Kommissionierung in der logistischen Produktion des Versandhandels, 2007.

[3] Salesforce, "salesforce.com," Salesforce, [Online]. Available: https://www.salesforce.com/products/what-is-salesforce/. [Accessed 30 Januar 2018].

[4] Recon Instruments, "reconinstruments.com," Recon Instruments, [Online]. Available: https://www.reconinstruments.com/products/jet/. [Accessed 30 Januar 2018].

[5] Recon Instruments, "www.reconinstruments.com," Recon Instruments, [Online]. Available: https://www.reconinstruments.com/products/jet/tech-specs/. [Accessed 30 Januar 2018].

[6] Recon Instruments, "reconinstruments.com," Recon Instruments, [Online]. Available: https://www.reconinstruments.com/products/jet/jet-os4/. [Accessed 1 Januar 2018].

[7] Google, "developers.google.com," Google, 17 November 2017. [Online]. Available: https://developers.google.com/youtube/v3/docs/videos/insert. [Accessed 29 Januar 2018].

[8] W. Shaeffer, "medium.com," A Medium Corporation, 1 Dezember 2017. [Online]. Available: https://medium.com/vidme/goodbye-for-now-120b40becafa. [Accessed 29 Januar 2018].

[9] Box, Inc., "box.net," Box, Inc., [Online]. Available: https://www.box.com/en-gb/pricing. [Accessed 31 Januar 2018].

[10] Dropbox, „dropbox.com,“ Dropbox, [Online]. Available: https://www.dropbox.com/plans/individual. [Zugriff am 31 Januar 2018].

[11] Google, „google.com,“ Google, [Online]. Available: https://www.google.com/intl/de_ALL/drive/pricing/. [Zugriff am 31 Januar 2018].

[12] Microsoft, „onedrive.live.com,“ Microsoft, [Online]. Available: https://onedrive.live.com/about/de-DE/plans/. [Zugriff am 31 Januar 2018].

[13] Google, „developers.google.com,“ Google, [Online]. Available: https://developers.google.com/drive/android/. [Zugriff am 1 Februar 2018].

[14] Dropbox, „dropbox.com,“ Dropbox, [Online]. Available: https://www.dropbox.com/developers-v1/core/start/android. [Zugriff am 1 Februar 2018].

[15] Box, „github.com,“ Box, [Online]. Available: https://github.com/box/box-android-sdk. [Zugriff am 1 Februar 2018].

[16] Microsoft, „github.com,“ Microsoft, [Online]. Available: https://github.com/OneDrive/onedrive-sdk-android. [Zugriff am 1 Februar 2018].

[17] L. Vogel, "vogella.com," 9 November 2016. [Online]. Available: http://www.vogella.com/tutorials/AndroidListView/article.html#androidlists_overview.

[18] Google, "Developer Android," [Online]. Available: https://developer.android.com/training/camera/photobasics.html.

[19] Google, "Developer Android," [Online]. Available: https://developer.android.com/training/camera/videobasics.html.

[20] Google, "developers.google.com," Google, 6 November 2017. [Online]. Available: https://developers.google.com/drive/android/auth. [Accessed 10 Januar 2018].

[21] Google, "developers.google.com," Google, 6 November 2017. [Online]. Available: https://developers.google.com/identity/sign-in/android/sign-in?configured=true. [Accessed 10 Januar 2018].

[22] Google, „developers.google.com,“ 19 März 2017. [Online]. Available: https://developers.google.com/drive/v3/reference/files/create.

[23] Google, "developers.google.com," developers.google.com, 6 November 2017. [Online]. Available: https://developers.google.com/drive/android/appfolder. [Accessed 26 Januar 2018].

[24] Google, "developers.google.com," Google, 6 November 2017. [Online]. Available: https://developers.google.com/drive/android/files. [Accessed 26 Januar 2018].

[25] Google, „drive.google.com,“ Google, [Online]. Available: https://drive.google.com/settings/storage. [Zugriff am 30 Januar 2018].

Selbständigkeitserklärung gem. § 22 Absatz 5 BPO

Hiermit versichere ich, Matthias Reiher, dass ich die vorliegende Bachelorarbeit mit dem Titel

Entwicklung einer mobilen Anwendung zur Verbindung von Recon Jet und Salesforce

selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

.....	.....
Ort, Datum	Unterschrift