Heinrich Heine
Universität
Düsseldorf

INSTITUT FÜR INFORMATIK
Softwaretechnik und Programmiersprachen
Universitätsstr. 1 D–40225 Düsseldorf

App zur Digitalisierung und Optimierung von Lagerarbeiten auf Basis des Outbound-Prozesses

Bachelorarbeit

im Studiengang Informatik
zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt von

Jeewon Lee

Beginn der Arbeit: 02. November 2022
Abgabe der Arbeit: 02. Februar 2023

Erstgutachter: Prof. Dr. Michael Leuschel
Zweitgutachter: Dr. Chien-Cheng Huang

Selbstständigkeitserklärung

Hiermit versichere ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 02. Februar 2023

Jeewon Lee

Zusammenfassung

Nachdem Verkauf eines Artikels, muss dieser durch die Anwendung mehrere Schritte zum Käufer gelangen. Diese Schritte bilden dabei den Outbound-Prozess. Die Perixx Computer GmbH ist eine Firma die sich auf den Vertrieb von Computerperipheriegeräte wie Tastaturen und Mäuse spezialisiert hat. Dabei findet der Outbound-Prozess größtenteils manuell, ohne technologischer Unterstützung, statt. Insbesondere der Schritt der Zusammenstellung einer Bestellung zu einem versandfertigen Paket kann optimiert werden. Für diese Bachelorarbeit wurde in Kooperation mit der Firma Perixx die mobile Applikation Perixx Outbound App (POA) entwickelt. Gemeinsam wurden Anforderungen an die Applikation erarbeitet und in Features unterteilt. Die Implementierung der POA wurde mit Hilfe des Flutter Frameworks durchgeführt. Diese eignet sich besonders gut für das plattformübergreifende Entwickeln von mobilen Applikation. Dadurch war es möglich mit einer Code-Basis eine Applikation für die Plattformen Android und IOS zu entwickeln. Um zu überprüfen ob die POA den Outbound-Prozess optimieren kann, wurde eine Evaluation mit mehreren Mitarbeitern der Firma Perixx durchgeführt. Das Ergebnis zeigt, dass nach mehrtägiger Gewöhnungsphase die Dauer für die Zusammenstellung einer Bestellung von durchschnittlich einer Minute pro Bestellung (ohne POA) auf 36 Sekunden pro Bestellung (mit POA) gesunken ist. Auch ist die Anzahl der Fehler gesunken, da die Mitarbeiter nicht mehr die Artikelnummern manuell abgleichen müssen, sondern einfach mit einer Scanpistole einscannen und von der POA vergleichen lassen können.

Danksagung

Hiermit möchte ich mich bei allen Personen bedanken, die mich bei meiner Bachelorarbeit unterstützt haben. Besonders möchte ich mich bei Joshua Schmidt für die Betreuung meiner Arbeit, informativen Meetings und viel Geduld bei zahlreichen Fragen bedanken.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Ziele

Die Perixx Computer GmbH [1] ist ein Unternehmen, das Computerperipheriegeräte verkauft und sich besonders auf Eingabe- und Ausgabegeräte, Tastaturen und Mäuse spezialisiert hat. Die jährlichen Aufträge, die Perixx über verschiedene Einkaufskanäle erhält, nehmen stetig zu. Dabei treten immer wieder Probleme auf, die im gesamten Prozess von der internen Verarbeitung bis zur Auslieferung an die Kunden stattfinden. So kann es beispielsweise vor kommen, dass ein Mitarbeiter ein falsches Produkt für einen Auftrag gebucht oder verpackt hat, so dass ein Kunde das falsche Produkt erhält. Das führt dazu, dass Arbeitskräfte extra Zeit aufbringen müssen um die Fehler zu beheben. Um solche Fehlern zu vermeiden, müssen die Aufträge in jedem Schritt mehrfach überprüft werden, was ineffizient und umständlich ist. Dennoch kann auch durch eine solche Vorgehensweise nicht ausgeschlossen werden, dass Fehler geschehen. Für ein Unternehmen wie Perixx, das jedes Jahr einen steigenden Umsatz verzeichnet, ist es notwendig, ein System zu haben, das den gesamten Outbound-Prozess unterstützt. Das Ziel ist den Outbound-Prozess genauer und schneller zu gestalten und die Digitalisierung sowie Optimierung des gesamten Prozesses, wobei der Schwerpunkt der Bachelorarbeit auf der Entwicklung einer App liegt, die insbesondere beim Abholen eines passenden Produkts und das Verpacken der Bestellungen, unterstützen soll.

1.2 Aufbau der Arbeit

Die Bearbeitung der Bachelorarbeit ist in mehrere Kapitel aufgeteilt. Im Kapitel 2 werden Ablauf des Outbound-Prozesses, Grundlagen zu Flutter und GetX und die verwendeten Datenbank erläutert, welche zum Verstehen der Bachelorarbeit nötig sind. Im Kapitel 3 werden die Anforderungen an die POA vorgestellt. Das 4. Kapitel gibt einen kompletten Einblick in die POA. Dabei werden die Struktur der POA, sowie die Features gemäß den zuvor vereinbarten Anforderungen vorgestellt. Im Kapitel 5 wird auf die Ausführung von Tests eingegangen. Das 6. Kapitel beschäftigt sich mit der empirischen Evaluation, welche für die Applikation durchgeführt wurde. Unter anderem wird die Auswertung der Testnutzer vorgestellt, sowie ein statistischer Vergleich der Effektivität der App zum Outbound-Prozess dargestellt. In Kapitel 7 wird ein Ausblick auf mögliche Verbesserungen und Erweiterungen gegeben. Das 8. Kapitel schließt die Bachelorarbeit mit einem Fazit ab.

2 Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen, die für das Verständnis der Bachelorarbeit benötigt werden. Dabei wird auf den Ablauf des Perixx Outbound-Prozesses und

[1] Die Perixx Website : https://perixx.com

einige Merkmale von Flutter und GetX eingegangen. Schließlich werden die verwendete Datenbanken vorgestellt.

2.1 Perixx Outbound-Prozess

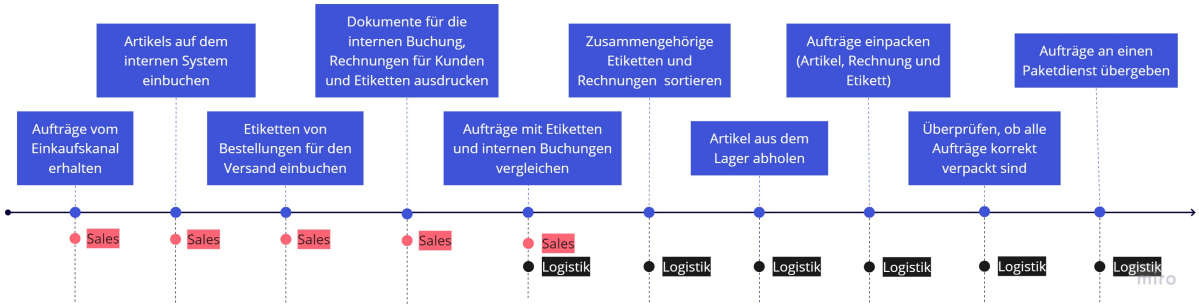


Abbildung 1: Ablauf des Perixx Outbound-Prozesses

Dieser Abschnitt beschreibt den allgemeinen Outbound-Prozess von Perixx, auf dem dieses Projekt basiert. Wenn die Salesabteilung eine Auftragsliste von einem Einkaufskanal erhält, müssen die Artikel, die an Kunden gesendet werden müssen im internen System gebucht werden. Danach werden die Rechnungen der Aufträge und die Etiketten für den Versand bereitgestellt. Die vorbereiteten Dokumente werden gedruckt und geprüft, ob Rechnungen und Etiketten für alle Aufträge vorhanden und in Ordnung sind. Ist dies der Fall, werden alle Dokumente an die Logistikabteilung weitergeleitet. In der Logistikabteilung werden die Dokumente nach Auftragsnummer sortiert, damit die Aufträge schneller verpackt werden können. Zwischendurch werden die Artikel aus dem Lager abgeholt und nach Auftragsnummern sortiert. Schließlich werden die Artikel und die zugehörigen Dokumente verpackt und ein letztes Mal auf Korrektheit geprüft, um sicherzustellen, dass kein Fehler aufgetreten ist. Zum Schluss werden die verpackten Artikel an den Paketdienst übergeben und dieser kümmert sich um die Lieferung der Artikel zum Kunden. Siehe Abbildung 1, um einen anschaulichen Blick über den Outbound-Prozess zu erhalten.

2.2 Flutter

Flutter [5] ist ein von Google entwickeltes Framework zur Entwicklung von Apps. Flutter verwendet eine objektorientierte Programmiersprache namens Dart, die auch von Google entwickelt wird. Es ermöglicht das Entwickeln von Web-, Desktop- und plattformübergreifenden Applikationen, die dann auf Android- und IOS-Geräten laufen können. Damit das funktioniert, wird eine Codebasis benötigt. Für ein Unternehmen wie Perixx, das dringend ein System benötigt, das auf mehreren Plattformen läuft, ist dies ein attraktiver Aspekt von Flutter. Die wichtigsten Elemente von Flutter werden im Folgenden beschrieben.

2.2.1 Widget

Im Flutter Framework bestehen alle Elemente aus Widgets und ein Widget besteht selbst auch aus zahlreichen Widgets [2]. Zum Beispiel benötigt ein Textbutton-Widget ein vordefiniertes Verhalten, wenn es angeklickt wird, und dieses benötigt wiederum ein Text-Widget, das sich im Button befindet. So entsteht dann aus vielen Widgets ein User Interface (UI). Es wird zwischen zwei Arten von Widgets unterschieden.

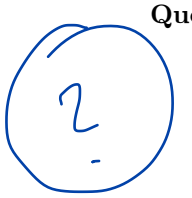
Stateless Widget : Diese Art von Widget hat keinen Zustand. Das bedeutet, dass ein solches Widget nach dem Erstellen keine Änderungen vornehmen kann. Der Zustand eines Widgets ist die Information, die das Widget gerade besitzt und somit wie es aussieht. Der Zustand kann sich ändern, wenn dieses Widget neue Informationen erhält. Mit den neuen Informationen ist dabei die neue Zuweisungen der Parameter gemeint.

Stateful Widget : Diese Art von Widget kann sich im Gegensatz zu Stateless Widgets nach dem Erstellen ändern. Das kann durch eine neue Zuweisung des Zustandes mit Hilfe einer `setState`-Methode erfolgen, welches in Quellcode 1 dargestellt wird.

```
1: onChanged: (value) {
2:   setState(() {
3:     _email = value;
4:   });
5: }
```

Quellcode 1: setState Methode

2.3 GetX



GetX ist ein schnelles, stabiles und leichtes Framework in Flutter, das State Management, das Route Management, das Dependency Management und weitere nützliche Funktionalitäten bietet [4]. Im Folgenden wird auf die Funktionalitäten eingegangen, die für die Entwicklung der Applikation verwendet werden.

2.3.1 Reactive State Management

State Management bedeutet in Flutter, den Zustand einer Variablen zu beobachten und jedes Mal, wenn sich der Zustand ändert, wird das zugehörige Widget neu gerendert. Laut der GetX-Dokumentation [7] ist die Implementierung des reaktiven State Managements kompliziert. Durch die Nutzung vom GetX-Framework wird dies stark vereinfacht. Damit eine Variable beobachtet werden kann, muss das Namensende der Variable mit `.obs` initialisiert werden. Indem ein Widget in `Obx(() => )` eingefügt wird, kann das UI jedes Mal aktualisiert werden, wenn sich die Variable ändert. Siehe Quellcode 2.

```
RxList<Order> orderList = <Order>[].obs;

Obx( SliverList(delegate: SliverChildBuilderDelegate(
          (context, index) {
         final order = _orderController.orderList[index];
       return OrderListView(index: index, order: order);
      },
      childCount: _orderController.orderList.length,
       ),
      ),
   ),
```

Quellcode 2: Reactive state management

2.3.2 Route Management

Es ist bequemer, einfacher und kürzer, das GetX Route Management zu verwenden als das Navigator-Widget, das in der Navigation Klasse von Flutter enthalten ist. Das GetX Route Management benötigt keinen Parameter `Buildcontext`, der normalerweise nötig wäre um zu wissen wo sich das Widget befindet. Um das Route Management zu nutzen, muss die App mit `GetMatrialApp` statt mit `MatrialApp` gestartet werden. Wenn die gewünschte Seite mit einem Pfad definiert ist, ist es auch möglich, sich zu der Seite mit diesem Pfad zu navigieren. Siehe Quellcode 3.

```
GetPage (
  name : '/ LOGIN ',
  page : () => const LoginView () ,
),

Get.to( LoginView ());
Get.toNamed ('/ LOGIN ');
```

Quellcode 3: Route Management

2.3.3 GetxController

Um guten Software Code zu garantieren, sollte auf bestimmte Grundsätze geachtet werden. Einer davon ist die Einteilung des Codes in mehrere disjunkte Bereiche, wobei jeder Bereich seine eigenen Aufgaben erledigt. Dieses Prinzip heißt Separation of Concerns [9]. Ein Beispiel ist die Trennung von Geschäftslogik und UI. Durch das GetX State Management können Geschäftslogik und UI getrennt werden, indem Geschäftlogik in einer Klasse definiert wird, die vom `GetxController` erbt. Wie die Nutzung eines Controllers funktioniert, wird im folgenden Unterkapitel beschrieben.

2.3.4 Dependency Management

Durch die Nutzung des GetX Frameworks können Klassen von einer engen Kopplung befreit werden. Mit der folgenden Codezeile wird der OrderController deklariert und global zur Verfügung gestellt. `Controller controller = Get.put<OrderController> (OrderController());` Wenn die Klasse in einer anderen Klasse aufgerufen werden soll, wird der OrderController mit dem Code `final orderController = Get.find <OrderController>();` aus dem Speicher aufgerufen. Durch die Verwendung einer Binding-Application Programming Interface (API) wird es ermöglicht, eine View mit einem Controller zu verbinden, ohne voneinander abhängig zu sein, Siehe Quellcode 4.

```
class MainBinding implements Bindings {
  @override
  void dependencies() {
    Get.put<OrderController>(
      OrderController(),
      permanent: true,
    );
    Get.put<PrintController>(
      PrintController(),
      permanent: true,
    );
  }
}

GetPage(
  name: '/ORDERLIST',
  page: () => const OrderView(),
  binding: MainBinding(),
),
```

Quellcode 4: GetX Dependency Management Binding Klasse

2.3.5 Internationalisierung

Wenn eine Software von Menschen aus aller Welt genutzt werden soll, ist das GetX Framework eine gute Wahl, um verschiedene Sprachen entsprechend derer Bedürfnisse anzubieten. Im GetX Framework werden die Übersetzungen in Form einer einfachen Key-Value-Map gespeichert. Als Schlüssel werden allgemeine Bezeichnungen von Aktivitäten und als Wert anzuzeigende Sätze mit passender Sprache gespeichert. Die Klasse, in der sich die Übersetzungs-Map befindet, sollte von der GetX Klasse Translations erben. Um die Übersetzung zu verwenden, sollte **.tr** an das Ende des Schlüssels angehängt werden. Siehe Quellcode 5.

```
class Languages extends Translations {
  Map<String, Map<String, String>> get keys => {
    'en_US': {
      'greeting': 'Hello',
    },
    'ge_GE': {
      'greeting': 'Hallo',
    }
  };

  Text('no_result'.tr);
}
```

Quellcode 5: GetX Übersetzung-Map

2.3.6 GetxService

GetxService funktioniert ähnlich wie GetxController. Der einzige Unterschied besteht darin, dass GetxService keine eigene Logik besitzt. Sie teilt dem GetX Dependency Management mit, dass eine Klasse nicht aus dem Speicher entfernt werden darf [4]. Aus diesem Grund ist es sinnvoll, Funktionen, die im Hintergrund laufen müssen in einer Klasse zu definieren die von `GetxService` erbt.

2.4 Datenbanken

Die POA nutzt als persistente Datenquelle zwei unterschiedliche Datenbanken, als NoSQL Datenbank wird Firebase genutzt und als relationale Datenbank wird MySQL genutzt. Wie diese funktionieren und welche Daten darin gespeichert werden, wird in den beiden folgenden Unterkapiteln erklärt.

2.4.1 Firebase

Firebase [6] ist eine von Google entwickelte Plattform, mit der mobile Applikationen aufgebaut, verbessert und erweitert werden können. Sie bietet zahlreiche Tools und Services an. Dazu gehören unter anderem Analytik, Authentifizierung, Datenbanken, Konfiguration und Dateispeicherung. Die Services werden in der Google Cloud gehostet. Das heißt, die Backend-Komponenten einer App, die Firebase nutzt, werden von Google ausgeführt. Die Frontend-Komponenten kommunizieren direkt über Firebase mit den Backend-Komponenten ohne zusätzliche Middleware [10]. Eine weitere Besonderheit von Firebase ist, dass Firebase als NoSQL-Datenbankprogramm kategorisiert wird. Dabei werden die Daten in Form von JSON-Objekte gespeichert. Die Datenbank selber hat eine JSON-Baum Struktur. In der POA wird das Firebase-Authentifizierungstool verwendet, um die Identität eines Benutzers zu überprüfen. In einer verbundenen Google-Cloud werden E-Mail und Passwort jedes

Benutzers gespeichert.

2.4.2 MySQL

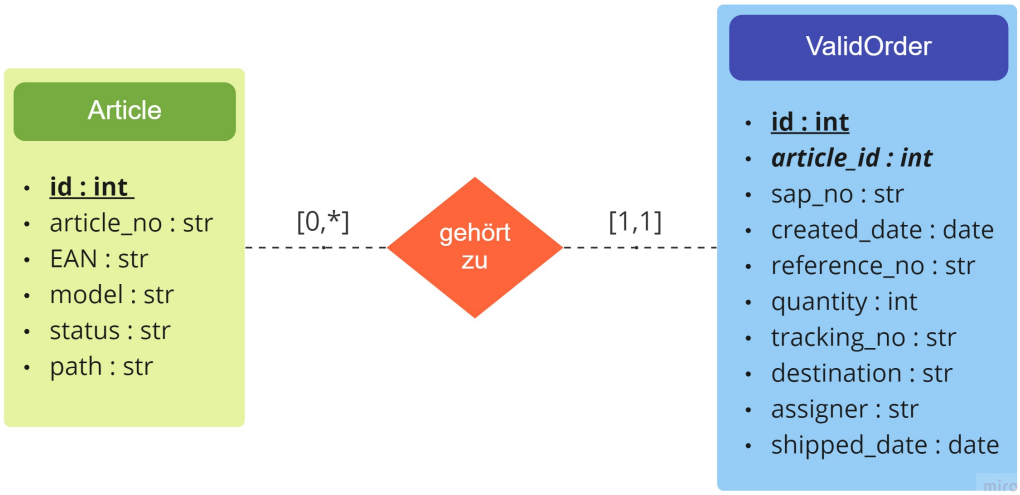


Abbildung 2: Datenstruktur

Zur Verwaltung der Aufträge- und Artikeldaten wird ein MySQL-Server 8.0.28 verwendet, den Perixx für sein internes System nutzt. Für die POA stehen eine Artikeltabelle und eine Auftragstabelle zur Verfügung. Ein Artikel hat eine ID als Primärschlüssel und andere benötigte Attribute, wie Artikelnummer und European Article Number (EAN)-Code[2]. Ein Auftrag hat ebenfalls eine ID als Primärschlüssel, die ID von einem Artikel als Fremdschlüssel und weitere Attribute. Außerdem sollte die Kombination von der Auftragsnummer und der ID von einem Artikel eindeutig sein. Somit ist die Beziehung zwischen der Artikeltabelle und der Auftragstabelle eine 1 zu N-Beziehung. Siehe Abbildung 2. Die eingelesenen Einträge aus der Auftragstabelle werden nach Auftragsnummer sortiert und so transformiert, dass alle zusammengehörigen Artikel in einem Auftragsobjekt gespeichert werden.

3 Anforderungen

Das Dokumentieren der Benutzeranforderungen ist in der Praxis für eine erfolgreiche Umset zung notwendig. Das liegt daran, dass die Nutzer oft nicht in der Lage sind, ihre Bedürfnisse und Wünsche vollständig mitzuteilen, und dass die Informationen, die sie liefern, unvollstän dig, ungenau und widersprüchlich sein können. Darüber hinaus erleichtern die vereinbarten Anforderungen späteren Abgleich, ob alle gewünschten Features implementiert wurden.

[2]Der Begriff European Article Number bezieht sich auf den Strichcode aus der EAN/UPC-Symbologie.

Nachfolgend werden die Anforderungen an die zu entwickelnde Applikation beschrieben. Diese wurden gemeinsam mit der Firma Perixx erarbeitet. Die Anforderungen sind nach Features gegliedert (Einloggen, Auftragsliste, Scannen, Drucken und Sonstiges).

- **Einloggen**

  A1 Nur Mitarbeiter, die dazu berechtigt sind, können die POA nutzen.

  A2 Benutzerkonten werden von Administratoren eingerichtet und dann den Mitarbeitern zur Verfügung gestellt.

  A3 Benutzer können sich anmelden und abmelden.

  A4 Werden Nutzer-Anmeldedaten falsch eingegeben, schlägt das Anmelden fehl und der Nutzer erhält eine Information darüber.

- **Auftragsliste**

  A5 Aufträge können nach Datum und Status (nicht bearbeitet/gescannt/versandt) gefiltert werden.

  A6 Wenn ein Auftrag den Status "versandt" hat, sollte er direkt mit der Druckseite von der Auftragslistenseite verlinkt sein.

  A7 Aufträge mit unterschiedlichen Status sollten unterschiedlich grafisch gekennzeichnet werden.

- **Scannen**

  A8 Alle Produkte von Perixx können mit der Scanpistole gescannt werden.

  A9 Wenn ein Artikel gescannt wird, der zu mehreren Aufträge gehört, zeigt die POA an, welcher Auftrag zuerst bearbeitet werden soll.

  A10 Wenn ein falscher Artikel gescannt wird, der nicht zu den aktuellen Aufträgen gehört, sollte dies dem Nutzer mitgeteilt werden.

  A11 Wenn alle Artikel eines Auftrags gescannt wurden, ändert die POA den Status des Auftrags auf "scanned" und ermöglicht das Drucken der Dokumente.

- **Drucken**

  A12 Alle Dokumente eines Auftrags, die zugestellt werden müssen, sollten automatisch gedruckt werden, damit sie nicht vergessen werden können.

  A13 Dokumente, die bereits gedruckt wurden, können bei Bedarf jederzeit erneut gedruckt werden.

- **Sonstiges**

  A14 Die POA sollte auf Android und auch auf IOS laufen. (Fokus liegt auf IOS)

  A15 Die Sprache der POA sollte entsprechend der Geräteeinstellung als Standard eingestellt sein.

A16 Zur Auswahl stehen die Sprachen Englisch und Deutsch.

A17 Durch die POA kann der Outbound-Prozesses effektiver, präziser und schneller durchgeführt werden.

4 Implementierung

In diesem Kapitel wird die Implementierung der POA im Detail beschrieben. Zunächst wird die Struktur der POA sowie die Art und Weise der Kommunikation mit den Komponenten betrachtet. Im Anschluss daran wird auf die Realisierung der POA eingegangen. Dabei werden die vier Hauptfeatures und deren Aufgaben, sowie Anwendungsfälle vorgestellt. Außerdem werden die genutzten Bibliotheken vorgestellt. Der komplette Source-Code befindet sich auf dem bereitgestellten Gitlab-Repository[3].

4.1 Struktur der App

Eine Schichtenarchitektur ermöglicht es, Applikationen leicht zu verändern, zu erweitern und geteilte Komponenten leicht wieder zu verwenden [1]. Aus diesen Gründen wird die Schichtenarchitektur für die POA eingesetzt. Sie besteht aus vier verschiedenen Schichten (Präsentation-, Applikations-, Domänen- und Datenschicht). Jede Komponente der POA ist einer Schicht zugeordnet, abhängig von dessen Funktion. Abbildung 3 zeigt an aus welchen Elementen die Architektur der POA besteht. Die Pfeile stellen die Zugriffe von einer Komponente auf eine andere dar. Dabei kann man gut erkennen, dass Zugriffseinschränkungen, welche bei einer Schichtenarchitektur gelten, nicht überschritten werden. Es sind nur Zugriffe von Links nach Rechts möglich. Im Folgenden wird näher auf die einzelnen Schichten eingegangen.

Präsentationsschicht : In dieser Schicht befinden sich die Klassen, die Elemente im User Interface darstellen (alle Dinge, die Benutzer sehen und mit denen sie auf dem Bildschirm interagieren können, wie zum Beispiel Buttons oder Display-Boxen). Die Komponenten der Präsentationsschicht sind vor allem für die Darstellung der benötigten Daten aus der Applikationsschicht und der Entgegennahme von Benutzerinteraktionen zuständig. Die Präsentationsschicht kann nur auf die Applikationsschicht zugreifen.

Applikationsschicht : Diese Schicht enthält die Controller-Klassen, in denen die Geschäftslogik definiert ist. Darüber hinaus sind Variablen, die global verfügbar sein müssen, über die Controller-Klassen zugänglich. Die Applikationsschicht erhält die durch die Präsentationsschicht übergebenen Benutzerinteraktionen und gibt die gewünschten Daten zurück. Das Besorgen dieser Daten wird durch einen Zugriff auf die Datenschicht ermöglicht.

[3] https://gitlab.cs.uni-duesseldorf.de/stups/abschlussarbeiten/jeewon-lee

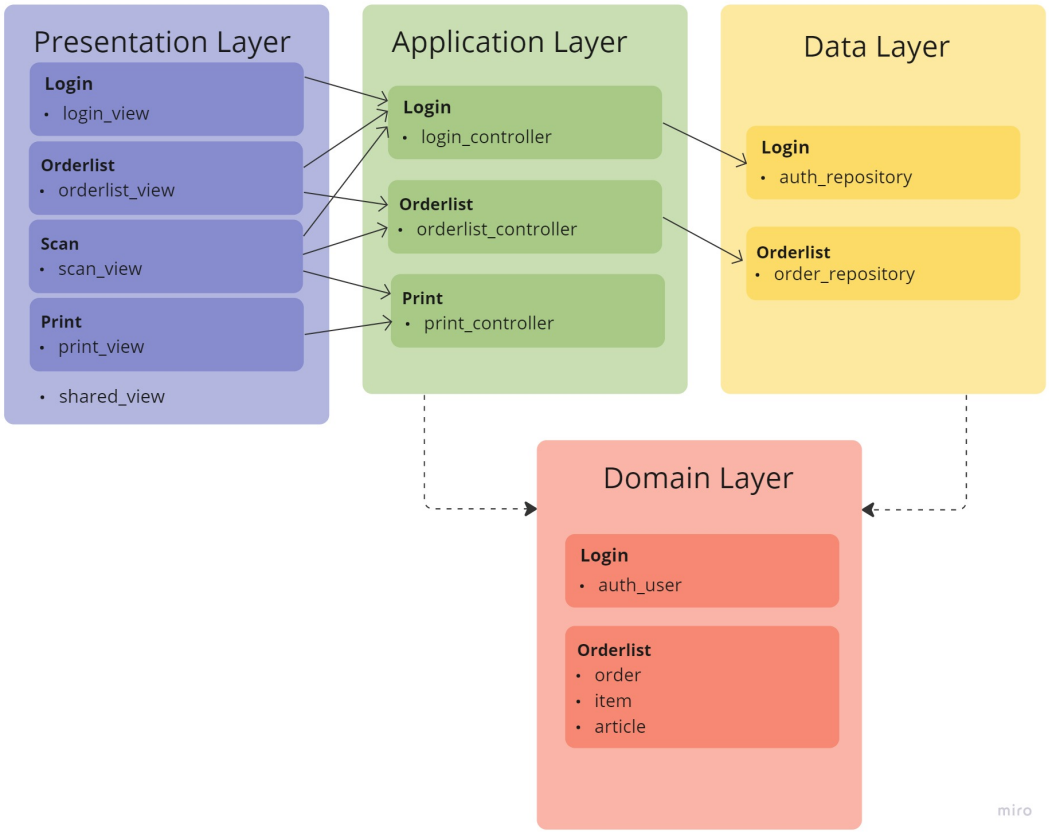


Abbildung 3: Schichtenarchitektur der POA

Datenschicht : In dieser Schicht werden die Daten vorbereitet, die aus der Datenbank abgerufen werden müssen. Die Daten werden in Form eines Modells aus der Domänenschicht zurückgegeben. Nur die Applikationsschicht kann auf die Datenschicht zugreifen. Einer der Existenzgründe der Datenschicht ist die Trennung von Geschäftslogik und direktem Zugriff auf die Datenbank. Draus erfolgt, dass es keine gegenseitige Abhängigkeit gibt.

Domänenschicht : Diese Schicht enthält die Entitätenklassen, in der Objekte von der POA definiert sind. Die Daten aus Datenbank werden in Form einer passenden Entität gespeichert.

Wie in Kapitel 3 bereits dargestellt, kann die POA auch nach Features zerlegt werden. Diese finden sich auch in der Architektur wieder, siehe Abbildung 3.

Einloggen (Login) : Benutzer können sich anmelden, um den Outbound-Prozess zu beginnen.

Auftragsliste (Orderlist) : Die Auftragsliste wird hier angezeigt. Die Auftragsliste

kann nach Datum und Status sortiert werden.

Scannen (Scan) : Aufträge werden in der Scan-Seite bearbeitet. Wenn ein Artikel gescannt wird, wird angezeigt, ob es zu versendende Aufträge gibt, die den Artikel enthalten.

Drucken (Print) : Die mit einem Auftrag verbundenen Dokumente werden auf der Druckseite zur Verfügung gestellt und sind bereits für den Druck vorbereitet.

Es gibt 2 Möglichkeiten, die oben beschriebenen Architekturansätze zu integrieren [1]. Falls alle Schichten in jeweils einem Feature eingelagert werden, handelt es sich um einen `Feature-First-Ansatz` (Schichten innerhalb von Features) und im umgekehrten Fall, handelt es sich um einen `Layer-First-Ansatz` (Features innerhalb von Schichten). Für die POA wird der `Layer-First-Ansatz` umgesetzt. Der Grund dafür ist, dass der `Layer-First-Ansatz` einfacher zu implementieren ist als der `Feature-First-Ansatz`, und dass einige Feature Komponenten (Controllers und Widgets) von anderen Features genutzt werden, so dass sie keine eigene Komponente bzw. Sicht benötigen.

4.2 Realisierung

In diesem Unterkapitel wird auf die vorab definierten Features eingegangen. Dabei wird erklärt wie das jeweilige Feature realisiert wurde und wie das Ergebnis aussieht.

4.2.1 Initialisierung

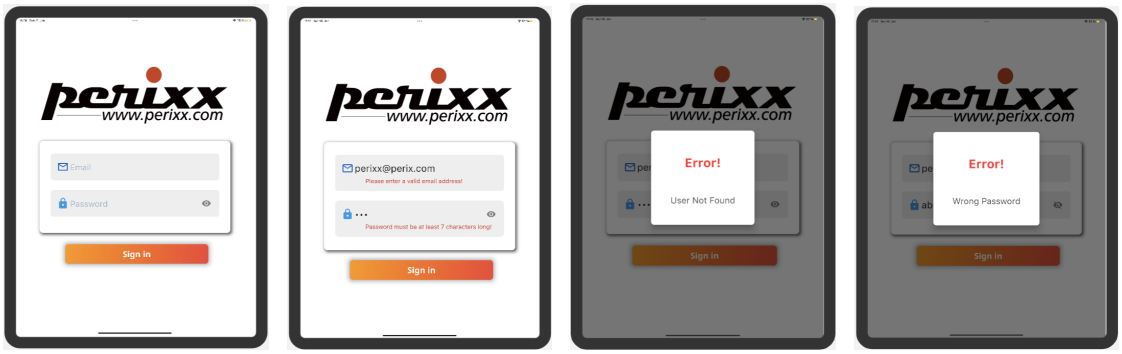
Wenn die POA gestartet wird, wird die `InitialSettingService.init` Methode aufgerufen. Diese stellt Verbindungen mit Firebase und dem MySqlServer bereit, wobei letzterer durch das Paket `Mysql1` [4] ermöglicht wird. Dadurch wird ein AuthController im Speicher abgelegt. Solange die POA ausgeführt wird, stehen Firebase und MySqlServer zur Verfügung und der AuthController wird nicht aus dem Speicher entfernt. So können alle Widgets ohne Abhängigkeit auf den AuthController und den angemeldeten Benutzer zugreifen.

4.2.2 Einloggen

Das UI der Einloggen-Seite besteht aus zwei Eingabefeldern und einem Button zum Ein loggen. Die Konten für die Anmeldung werden von Perixx bereitgestellt und es gibt keine Möglichkeit, sich selber zu registrieren. Wenn eine gültige E-Mail-Adresse und ein gültiges Passwort eingegeben wird, wird geprüft, ob ein Konto mit den übermittelten Informatio nen in Firebase gespeichert ist. Falls ein ungültiges E-Mail-Format oder Passwort-Format eingegeben wird, wird sofort eine entsprechende Meldung angezeigt, ohne dass in Firebase

[4] `https://pub.dev/packages/mysql1`

nachgeschaut wird. Siehe Abbildung Abbildung 4b. Falls in Firebase kein Konto mit den angegebe nen Daten gefunden wird, wird ebenfalls eine entsprechende Meldung angezeigt. Siehe Abbildungen 4c und 4d. Bei einem Erfolg des Einloggens wird die Auftragsliste angezeigt.



(a) Einloggen **(b)** Eingabe fehlgeschlagen **(c)** Nutzer nicht gefunden **(d)** Falsches Passwort

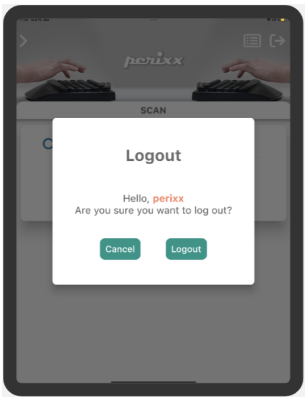
Abbildung 4: Einloggen

4.2.3 Auftragsliste

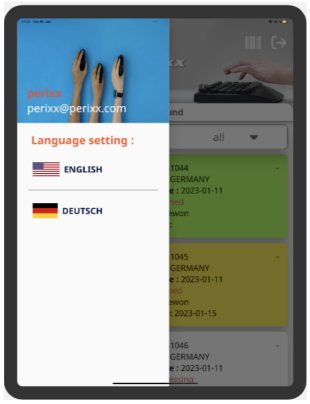
Wenn die Auftragsliste-Seite aufgerufen wird, wird der OrderController im Speicher abgelegt, so dass der Controller auf dieser Seite verfügbar ist. Er wird in einer Variable gespeichert und Variablen in diesem Controller, die mit `.obs` deklariert sind, werden über den Variablenname ihres Controllers aufgerufen und aktualisiert. Alle benötigten Widgets befinden sich wiederum im `CustomScrollView`-Widget, so dass der gesamte Bildschirm scrollbar ist. Das UI dieser Seite besteht aus Appbar-, Filter- und Auftragsliste- Widgets.

Appbar-Widget befindet sich ganz oben auf dem Bildschirm. Auf der Android Plattform wird ein Drawer/Sidebar-Widget für die Spracheinstellung durch Drücken des ganz linken Buttons angezeigt. Falls erforderlich, kann die Sprache, die in der POA angezeigt wird auf Englisch oder Deutsch eingestellt werden, welches in Abbildung 5b dargestellt ist. Da Apple anscheinend das Drawer-Widget nicht empfehlt und seinen eigenen Stil hat, ist es auf der IOS Plattform nicht möglich, über einen Button das Drawer-Widget auf- und zuzumachen [8]. Das zweite Button von Rechts navigiert zur Scan-Seite. Das wird durch die `Get.offNamed`-Methode [4] ermöglicht. Sie entfernt die aktuelle Seite und fügt eine neue Seite mit dem angegebenen Pfad hinzu. Durch Drücken des ganz rechten Buttons wird ein Dialog-Widget angezeigt, in dem der Benutzer gefragt wird, ob er sich wirklich abmelden möchte; wenn er dies bestätigt, wird er abgemeldet. Siehe Abbildung 5a. Dieser Widget wird auch in der Scan-Seite verwendet.

Filter-Widget Dieses Widget besteht wiederum aus 2 Widgets, einem Text-Button-Widget und einem Dropdown-Widget. Das Text-Button-Widget auf der linken Seite



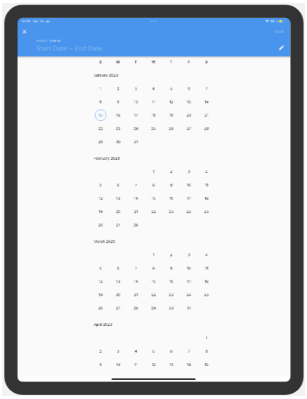
(a) Ausloggen



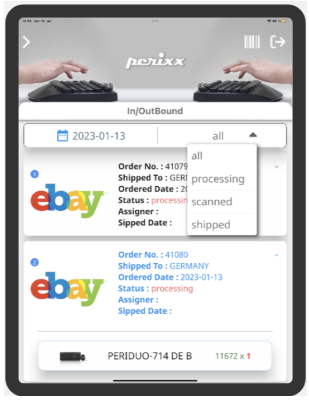
(b) Sprach-Einstellung

Abbildung 5: Appbar-Widget

ist mit einem bereits von Flutter bereitgestellten `showDateRangePicker` [5]-Widget verknüpft, mit dem der Benutzer den gewünschten Zeitraum einstellen kann. Siehe Abbildung 6a. Das rechte Widget ist mit einem `DropdownButtonHideUnderline`-Widget aus dem `dropdown_button2` [6] Paket verknüpft, mit dem das Dropdown-Widget nach Bedarf angepasst werden kann. Es gibt 4 mögliche Status der Aufträge im Dropdown -Widget (all, processing, scanned und shipped). Siehe Abbildung 6b. Jedes Mal wenn sich der Zeitraum oder/und der Status ändert, wird eine passende Methode vom OrderController aufgerufen, so dass die Variable `OrderController.orderList` mit einen neuen Wert zugewiesen wird und das Auftragsliste-Widget neu gerendert wird.



(a) Nach Datum

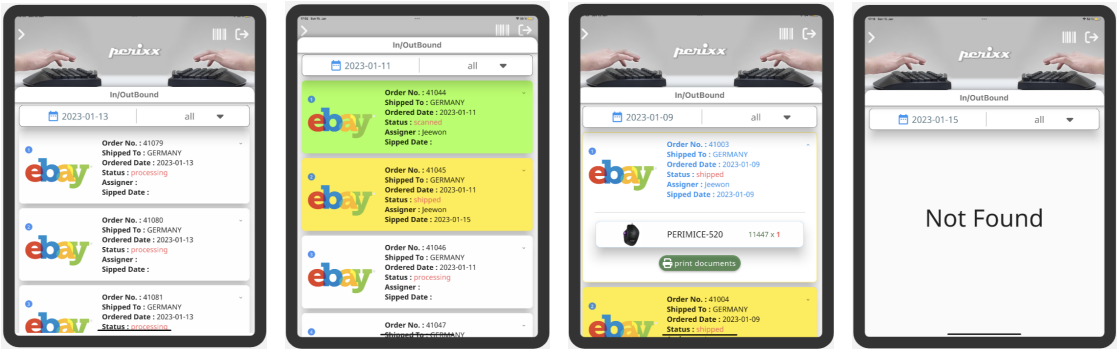


(b) Nach Status

Abbildung 6: Filterung für Auftragsliste

[5] https://api.flutter.dev/flutter/material/showDateRangePicker.html
[6] https://pub.dev/packages/dropdown_button2

Auftragsliste-Widget Aufträge, die in der Variable `OrderController.orderList` gespeichert sind, werden angezeigt. Abbildung 7a zeigt eine Auftragsliste mit mehreren Aufträgen, wobei Abbildung 7b eine leere Auftragsliste darstellt. Da die Variable `OrderController.orderList` beobachtet wird, wird das UI automatisch aktualisiert, falls sie geändert wird. Aufträge, die unterschiedliche Status haben, werden mit anderen Farben markiert. Aufträge mit dem Status "Scanned" haben einen grünen Hintergrund, "Shipped" einen gelben und "Processing" ohne Farbe. Siehe Abbildung 7b. Beim Drücken eines Auftrags werden Details angezeigt. Diese zeigen an welcher Artikel in welcher Menge gekauft wurde. Aufträge mit dem Status "Shipped" und "Scanned" haben einen Button mit einem direkten Link zur Druck-Seite. Siehe Abbildung 7c. Diese Druck-Seite wird durch den Aufruf der `Get.toNamed` Methode aufgebaut. Durch diese Methode können benötige Variablen zu einer anderen Seite übergeben werden. Für die Druck-Seite wird ein Order-Objekt weiter gegeben, damit die zugehörigen Dokumente eines Auftrags auf der Druck-Seite gedruckt werden können. Das Auftragsliste-Widget wird auch in der Scan-Seite weiter verwendet.



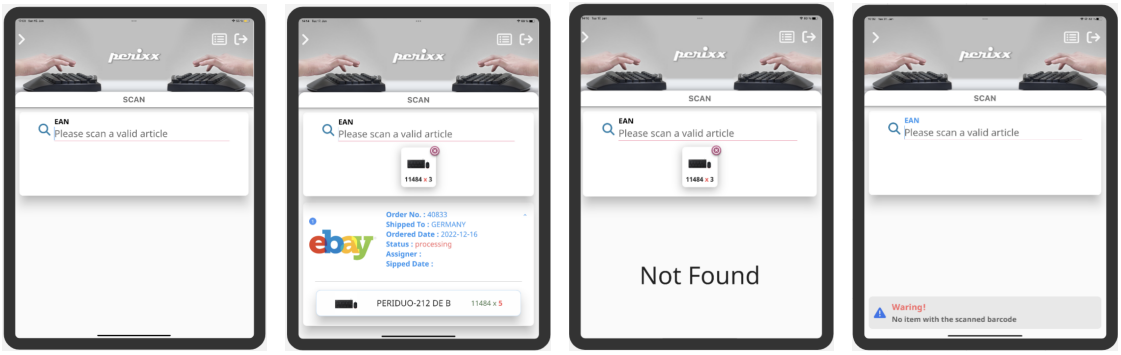
(a) Auftragsliste **(b)** mit Status **(c)** mit Auftragsdetails **(d)** keinen Auftrag

Abbildung 7: Auftragsliste

4.2.4 Scannen

Die Scan-Seite ist von der Struktur ähnlich wie die Auftragsliste-Seite. Wenn die Scan-Seite aufgebaut wird, wird der OrderController und der PrintController im Speicher abgelegt, somit kann auf die beiden Controller auf dieser Seite zugegriffen werden. Alle Widgets befinden sich wiederum im `CustomScrollView`-Widget. Die Appbar, die in der Auftragsliste verwendet wird, wird auch auf der Scan-Seite wieder verwendet. Der einzige Unterschied zur Appbar von der Auftragsliste ist, dass das zweite Button von rechts ein anderes Symbol hat und zurück zur Auftragsliste führt. In der Mitte befinden sich ein Eingabe-Widget, in dem eine EAN eingegeben werden kann. Unter dem Eingabe-Widget wird ein Artikelliste-Widget angezeigt, falls sich der gescannte Artikel im internen System befindet. Siehe Abbildung 8a.

Um eine EAN einzugeben, kann eine eine Scanpistole [7] verwendet werden, die den gelesenen Strichcode in eine EAN umwandelt und es in dem Eingabe-Widget einfügt. Da die EAN durch die Scanpistole übergeben wird, wird die virtuelle Tastatur von der Plattform mit `TextInputType.none` deaktiviert. Nachfolgend werden die möglichen Situationen und wie diese gehandhabt werden vorgestellt, die nach dem Scannen eines Artikels auftreten können. Es wird davon ausgegangen, dass eine korrekte EAN-Form (nur mit Ziffern) eingegeben wird. Wenn dies nicht der Fall ist, wird eine Fehlermeldung angezeigt.



(a) Scannen **(b)** Mit Aufträgen **(c)** Ohne Aufträge **(d)** Das Snackbar-Widget

Abbildung 8: Scannen

1. **Wenn es keinen Artikel mit der eingegebenen EAN gibt :**
   Es wird ein Snackbar-Widget mit einer Nachricht angezeigt, dass es keinen Artikel mit der eingegebenen EAN gibt. Siehe Abbildung 8d.

2. **Wenn es keinen zu versendenden Auftrag gibt, der den Artikel mit der eingegebenen EAN enthält :**
   Wie in Abbildung 8c zu sehen ist, wird der eingescannte Artikel mit seinem Bild, seiner Nummer und seiner Menge in einem Kästchen angezeigt. Auf dem Kästchen befindet sich ein Badge-Widget. Durch das Drücken des Badge-Widgets wird das Kästchen aus der Artikelliste entfernt. Da die Artikelliste eine Variable von einem StatefulWidget ist, wird das UI vom Artikelliste-Widget erneut gerendert.

3. **Wenn es zu versendende Aufträge gibt, die einen Artikel mit der eingegebenen EAN und andere Artikel enthalten :**
   Wie im zweiten Fall wird der eingescannte Artikel mit seinem Bild, seiner Nummer und seiner Menge in einem Kästchen angezeigt. Darunter werden alle zu versendende Aufträge aufgelistet, die den eingescannten Artikel enthalten. Das Widget für die Auftragsliste, das auf der Auftragsliste-Seite verwendet wurde, wird hier wieder eingesetzt. Siehe Abbildung 8b.

4. **Wenn es zu versendende Aufträge gibt, die nur einen Artikel mit der eingegebenen EAN enthalten :**

[7] https://eshop.standox.com/refinish/en_de/bcs-w1-wireless-barcode-scanner-32626.html

Die zu versendende Aufträge, die nur einen Artikel mit der eingegebenen EAN enthalten werden erneut in der Variable `orderController.orderlist` gespeichert und der erste Auftrag dieser Liste wird an die Druck-Seite mit der `Get.toNamed`-Methode übergeben. Gleichzeitig wird die `printDocuments`-Methode aus dem PrintController aufgerufen, damit die benötige Dokumente automatisch ausgedruckt werden. Über den PrintController wird im nächsten Unterkapitel berichtet.

4.2.5 Drucken

Auf der Druck-Seite ist nur der PrintController verfügbar. Da der zu verarbeitende Auftrag von der Scan-Seite übernommen wurde, ist es nicht notwendig, den OrderController im Speicher zu haben. Ganz links in der Appbar befindet sich ein Pfeil-Button, das zur vorherigen Seite führt. Auf diese Weise kann die Tätigkeit, an der der Benutzer gerade arbeitet, schnell fortgesetzt werden. Es gibt zwei mögliche Wege, um auf die Druck-Seite zu gelangen, entweder über die Auftragsliste-Seite oder die Scan-Seite.

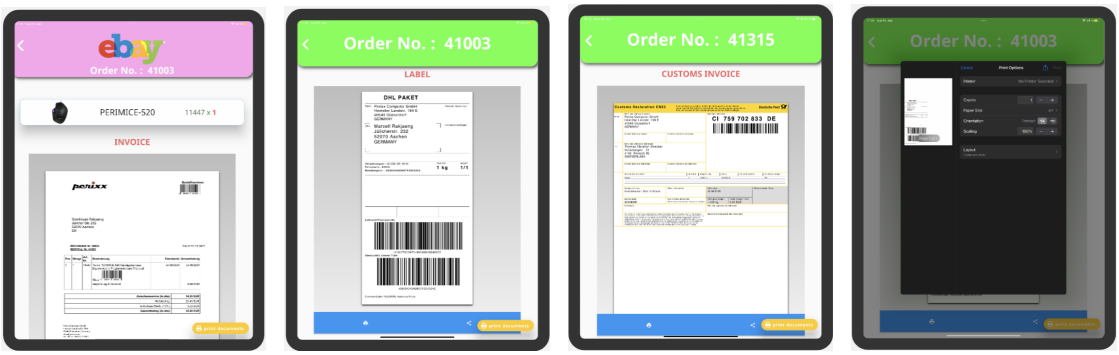
Über die Scan-Seite : Wie im vorherigen Unterkapitel erwähnt, wird die `directPrintPDF`-Methode aufgerufen und die Dokumente werden automatisch gedruckt, ohne dass ein Drucker ausgewählt und konfiguriert werden muss. Dadurch kann die Zeit für das Aussuchen eines Druckers und die Konfiguration gespart werden. Um diese Funktion auszuführen, muss die URL-Adresse eines Druckers übergeben werden. Für die POA wird ein `Brother-HL-L5100DN-Drucker` [8] für das Drucken der Rechnungen sowie Zollerklärungen und ein `Zebra-ZD420-Drucker` [9] für das Drucken des Versandetiketts verwendet. Die URL-Adressen werden in jeweils einer Konstante gespeichert und die Variablen werden übergeben. Diese Methode funktioniert nicht auf der Android-Plattform, da das Android SDK die Direktdruckfunktion nicht unterstützt [3]. Auf der Android-Plattform wird stattdessen die Methode `layoutPdf` aufgerufen, so dass automatisch ein Popup-Widget für die Druckerkonfiguration dargestellt wird.

Über die Auftragsliste-Seite : Es wird keine zusätzliche Funktion automatisch ausgeführt.

Unter der Appbar werden alle nötige Informationen im Bezug auf den Versand wie Artikelbeschreibung und Artikelmenge, sowie die Rechnung und das Versandetikett dargestellt. Sollte der Auftrag in ein Land das außerhalb der EU liegt versandt werden, wird eine Zollerklärung bereitgestellt. Siehe Abbildungen 9a und 9b. Für die Darstellung und das Drucken wird das `PRINTING` [10] Paket genutzt. Dokumente, die auf einem internen Server als PDF-Dateien gespeichert sind, werden vom Server abgerufen und in American Standard Code for Information Interchange (ASCII)-Werte umgewandelt. Eine ASCII-Liste stellt wiederum eine PDF-Datei dar. Die konvertierte ASCII-Liste wird dann an die Methode

[8] https://www.brother.de/drucker/laserdrucker/hl-l5100dn
[9] https://www.zebra.com/de/de/products/spec-sheets/printers/desktop/zd420.html
[10] https://pub.dev/packages/printing

`PDFPreview` übergeben, wodurch die PDF-Datei auf das UI angezeigt wird. Durch Drücken des Drucken-Buttons (zu erkennen an dem Button mit dem Drucker Icon) wird ein Pop-Up mit einer Druckerkonfiguration für das zugehörige Dokument angezeigt. Dadurch wird jedes Dokument mehrfach ausgedruckt, falls es nötig ist. Unten Rechts befindet sich ein Floating-Button-Widget, das ermöglicht, alle Dokumente durch Drücken des Buttons auszudrucken. Wie oben erläutert, wird auf der IOS-Plattform ein direkter Druckvorgang durchgeführt. Im Gegensatz dazu wird auf der Android-Plattform für jedes Dokument ein Popup-Widget angezeigt.



(a) Rechnung **(b)** Label **(c)** Zollerklärung **(d)** Mit Einstellung

Abbildung 9: Drucken

4.3 User Szenario

Für ein besseres Verständnis wie die POA von einem Benutzer verwendet werden kann wird im folgenden ein User Szenario beschrieben. Bildlich wird dieses in Abbildung 10 dargestellt.

1. Der Benutzer loggt sich ein. (Er bleibt beim nächsten Aufrufen der App eingeloggt, falls er sich nicht explizit ausgeloggt hat)

2. Dem Benutzer wird eine Auftragsliste angezeigt. Nach Bedarf können Aufträge nach dem Datum oder dem Status gefiltert werden.

3. Der Benutzer scannt zu versendende Artikel. Falls es einen Auftrag gibt, der nur den Artikel enthält, wird er zur Drucken Seite geleitet. Falls nicht, wird der Benutzer aufgefordert die restlichen Artikel zu scannen.

4. Beim Übergang vom Scannen zum Drucken werden alle benötigten Dokumente gedruckt. Bei Bedarf kann der Benutzer die Dokumente mehrmals drucken und einen geeigneten Drucker auswählen.

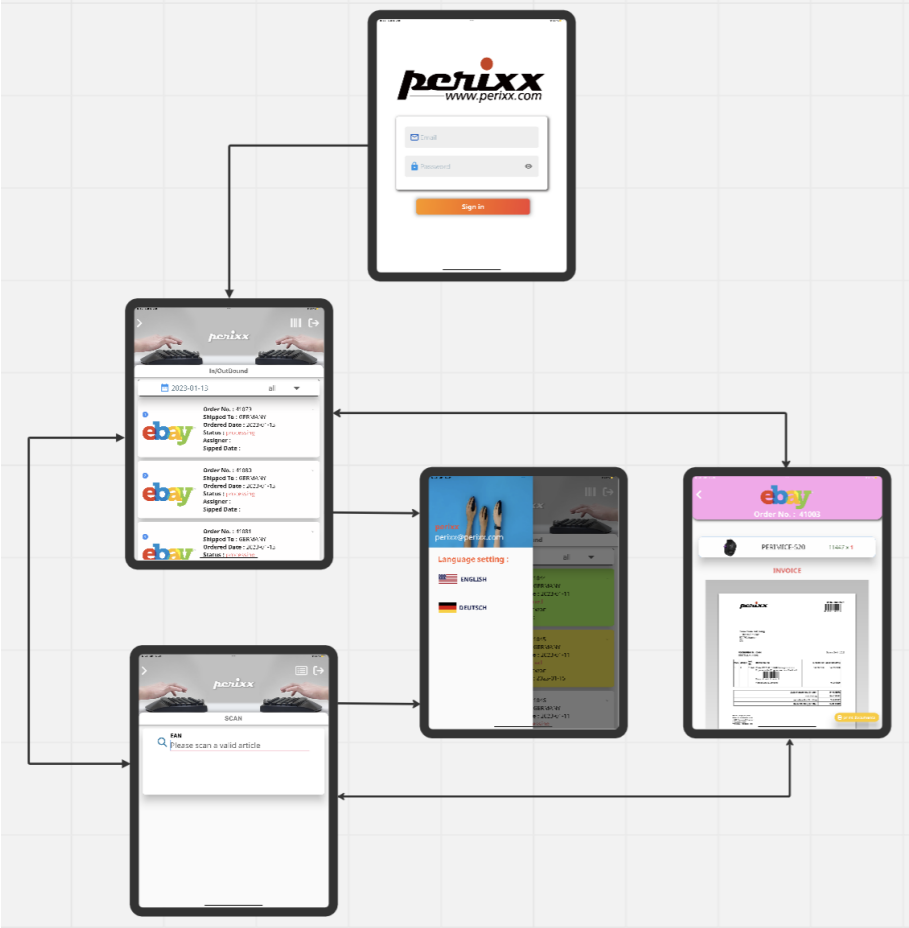


Abbildung 10: Das POA Verfahren

5. Der 2. und 3. Vorgang wird so lange wiederholt, bis kein zu versendender Auftrag mehr existiert.

6. Nachdem alle Aufträge gescannt und an den Paketdienst übergeben wurden, ändert der Benutzer den Status der Aufträge von "scanned" zu "shipped", durch Nutzung einer Swipe-Geste nach Rechts auf das Bar-Widget.

Nach dem 6. Vorgang ist der Perixx-Outbound-Prozess abgeschlossen.

5 Softwaretest

Tests werden benötigt um möglichst viele Fehler aufzudecken. Daher wurden in der POA nach jedem neu entwickelten Baustein Tests ausgeführt. Dadurch konnten Fehler sofort

aufgedeckt und behoben werden. Es wurde darauf geachtet, dass die Bausteine möglichst klein sind. Sind die Programmierbausteine zu groß, ist der Zeitaufwand für die Fehlersuche höher als bei kleineren Bausteinen. Flutter hat drei verschiedene Arten von Tests. Im Folgende werden die drei Test Arten erläutert.

5.1 Unit Test

Unit Tests werden für das Testen einzelner Komponenten und dessen Funktionalitäten benötigt. Dabei sollten die Elemente der einzelnen Komponenten unabhängig von den Elementen anderer Komponenten getestet werden. Damit das gelingt, sollte bereits während der Implementierung darauf geachtet werden, dass Features unabhängig zueinander sind. Das ist der Grund, warum für die POA das Dependency Management vom GetX verwendet wurde. Auch ist es wichtig, Methoden so klein wie möglich zu halten. Das erleichtert das Testen sehr. Falls eine Komponente getestet werden muss, die externen Services wie Datenbanken oder APIs benötigen, fällt es besonders schwer, da der Test durch die externen Services verlangsamt werden kann. Auch können dadurch die externen Service beeinflusst werden. Für die POA wurde daher das `mocktail` [11]Paket eingesetzt. Durch dieses Paket können künstliche Services erstellt und unter Kontrolle gebracht werden. Nachdem ein Service erzeugt wurde, kann der Rückgabewert manipuliert werden. Zum Schluss wird das reale Ergebnis und das erwartete Ergebnis verglichen. Der gesamte Ablauf wird in Quellcode 6 dargestellt.

```
1:  class MockOrderRepo extends Mock implements OrderRepository {}
2:
3:  final MockOrderRepo mockOrderRepo = MockOrderRepo ();
4:  final OrderController orderController =
5:  Get.put<OrderController>(OrderController(mockOrderRepo ));
6:
7:  test('Get an article by EAN', () async {
8:    // Arrange
9:    when (() => mockOrderRepo . getAllArticles ())
10:   .thenAnswer((_) => Future . value ( _getAllArticle()));
11:   // Act
12:   await orderController.init();
13:   // Assert
14:   expect(orderController.getOrderQuantity(), 2);
15:  });
```

Quellcode 6: Test mit `mocktail`

[11] https://pub.dev/packages/mocktail

5.2 Widget Test

Widget-Tests testen ein einzelnes Widget. Ein Widget Test prüft, ob das UI des Widgets wie erwartet aussieht und auf Nutzeraktionen richtig reagiert. Das Testen eines Widgets umfasst mehrere Klassen und erfordert eine Testumgebung, die den entsprechenden Lebenszyklus für das Widget bietet.

5.3 Integration Test

Integration Tests umfassen sowohl Unit Tests als auch Widget Tests zusammen mit externen Komponenten der Anwendung wie zum Beispiel Datenbanken. Durch diese Art von Tests wird überprüft, ob alle Widgets und Services wie erwartet zusammenarbeiten. Die Tests werden im Allgemeinen auf einem realen Gerät oder auf einem Betriebssystem-Emulator, wie dem IOS Simulator oder Android Emulator, durchgeführt. Für die Implementierung der POA wurde der Fokus stark auf die IOS-Plattform gesetzt. Daher wurde für die Integration Tests ein iPad Pro 12,9"5. Generation verwendet.

6 Empirische Evaluation

Um festzustellen, wie effizient und effektiv die POA beim Perixx-Outbound-Prozess unterstützt, wurden zwei Experimente durchgeführt. Im folgenden werden die Experimente näher beschrieben. Dabei wird auf das Setup eingegangen. Darüber hinaus werden Informationen über die Teilnehmer, sowie das Ergebnis des Experiments beschrieben.

6.1 Experiment 1 : Mit mehreren Testnutzern

6.1.1 Setup

Bei diesem Test wurden mehrere Testnutzer ausgewählt um die POA auszuprobieren. Anschließend wurden die Testnutzer gebeten ihre Meinung zur POA anhand eines Fragebogens mitzuteilen. Vor der Durchführung des Tests wurde den Testnutzern eine kurze Aufklärung über die POA gegeben. Das Experiment wurde so lange durchgeführt, bis alle Aufträge verpackt wurden. Im Durchschnitt gab es 25 Aufträge pro Tag. Die Zeit wurde nicht in Betracht gezogen. Der Grund dafür ist, dass die Testnutzer unterschiedliche Anpassungsfähigkeiten und Erfahrungen mit dem Gerät haben, auf dem der Test durchgeführt wurde, so dass dies auf das Ergebnis Einfluss haben kann. Nach dem Test wurden die Testnutzer gebeten, einen Fragebogen auszufüllen. Der Fragebogen wurde mit `Google Forms` [12] erstellt und ist noch über den Link verfügbar. Diesen konnten die Testnutzer anonym beantworten.

[12] `https://forms.gle/UG8pgWT3VbKJooKBA`

Die Fragen des Tests sind in 3 verschiedene Themenbereiche eingeteilt. Beim ersten Thema geht es um die Einfachheit der Nutzung der POA und die Zufriedenheit dabei. Das zweite Thema handelt von der Anordnung der Systeminformationen. Das letzte Thema ging um Effizient und Effektivität der POA im Vergleich zum aktuell noch von Perixx genutzten Outbound-Prozess (ohne POA).

6.1.2 Teilnehmer

Am Experiment nahmen insgesamt 8 Testnutzer teil. 75% der Teilnehmer hatten bereits zuvor mit dem Outbound-Prozess gearbeitet und 25% hatten keine Erfahrung mit dem Prozess. Alle Teilnehmer gaben an, dass sie mit der Elektronische Datenverarbeitung (EDV) gut umgehen können.

6.1.3 Ergebnis

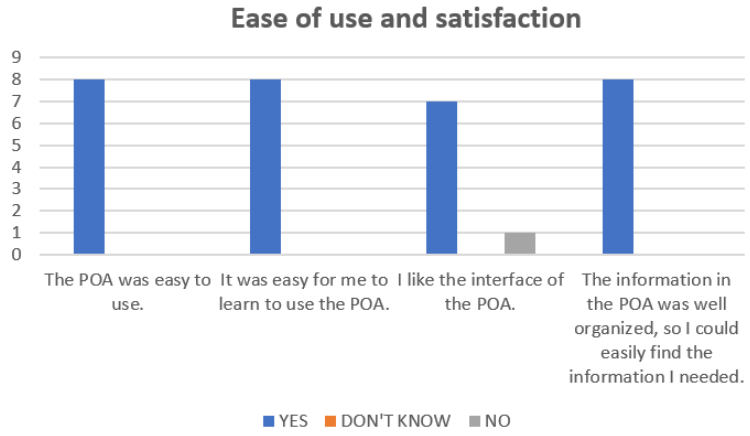


Abbildung 11: Einfachheit und Zufriedenheit

Abbildung 11 zeigt das Ergebnis von Themenbereich 1. Dieses ist größtenteils perfekt. Alle Testnutzer fanden, dass die POA leicht zu Nutzen ist. Auch fanden alle Testnutzer, dass es leicht war zu lernen wie die POA genutzt wird. Das Interface mochten bis auf einen Testnutzer alle. Die Informationen, die in der POA angezeigt werden, hielten alle Testnutzer für gut organisiert.

Abbildung 12 zeigt das Ergebnis von Themenbereich 2. Bis auf einen Testnutzer hatten alle kein Problem sich wieder schnell in der POA einzufinden nachdem sie einen Fehler gemacht hatten. Beim navigieren innerhalb der Applikation zwischen den unterschiedlichen Seiten hatte kein Testnutzer Probleme. Auch hatte kein Testnutzer Schwierigkeiten die Features der POA zu nutzen.

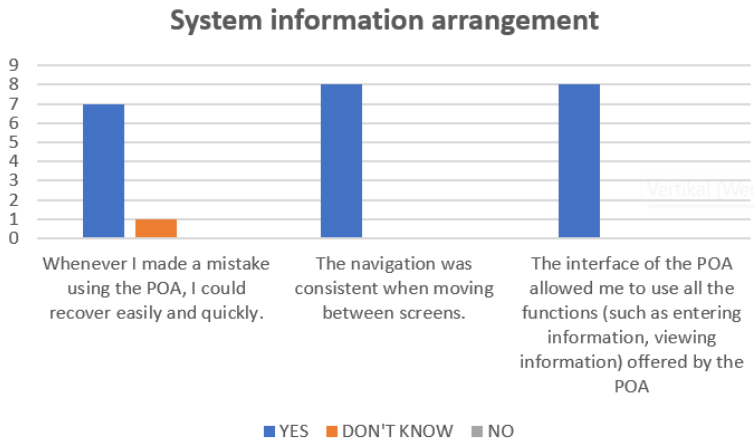


Abbildung 12: Anordnung der Systeminformationen

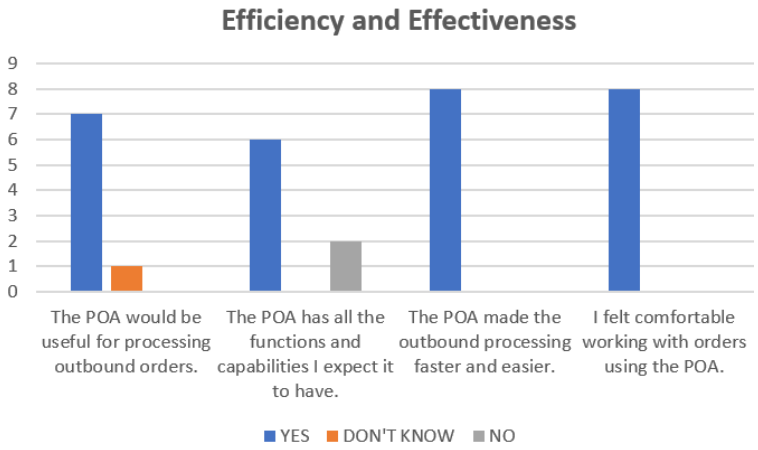


Abbildung 13: Effizienz und Effektivität

Abbildung 13 zeigt das Ergebnis von Themenbereich 3. Bis auf einen Testnutzer wurde die POA als hilfreiche Ergänzung zum Outbound-Process gesehen. Wobei diese eine Person angegeben hat, dass sie auf die Frage keine Antwort hat. Weiter wurden die Erwartungen an die angebotenen Features bei 25% der Testnutzer nicht erfüllt. Dafür gaben alle Testnutzer an, dass sie denken, dass der Outbound-Prozess durch die POA beschleunigt und vereinfacht wird. Ebenso fühlten sich alle Testnutzer wohl bei der Abarbeitung von Aufträgen mit Hilfe der POA. Die vorgestellten Ergebnisse zeigen, dass die POA auf Zustimmung unter den Testnutzern stoßt.

6.2 Experiment 2 : Mit einem Testnutzer

6.2.1 Setup

Bei Experiment 1 gab es die oben genannten Faktoren, die sich auf den Outbound-Prozess auswirken könnten. Daher wurde ein weiterer Test durchgeführt, um die Effizienz und Effektivität der POA besser bewerten zu können. Dieses Experiment dauerte 7 Tagen lang. Dabei hat immer derselbe Testnutzer die POA ausprobiert. Wie in Experiment 1 wurde ihm eine Einweisung zur POA gegeben. Im Durchschnitt gab es 25 Aufträge pro Tag.

6.2.2 Teilnehmer

Der Testnutzer hat lange Erfahrung mit dem aktuellen Outbound-Prozess und er kennt sich gut mit der EDV aus.

6.2.3 Ergebnis

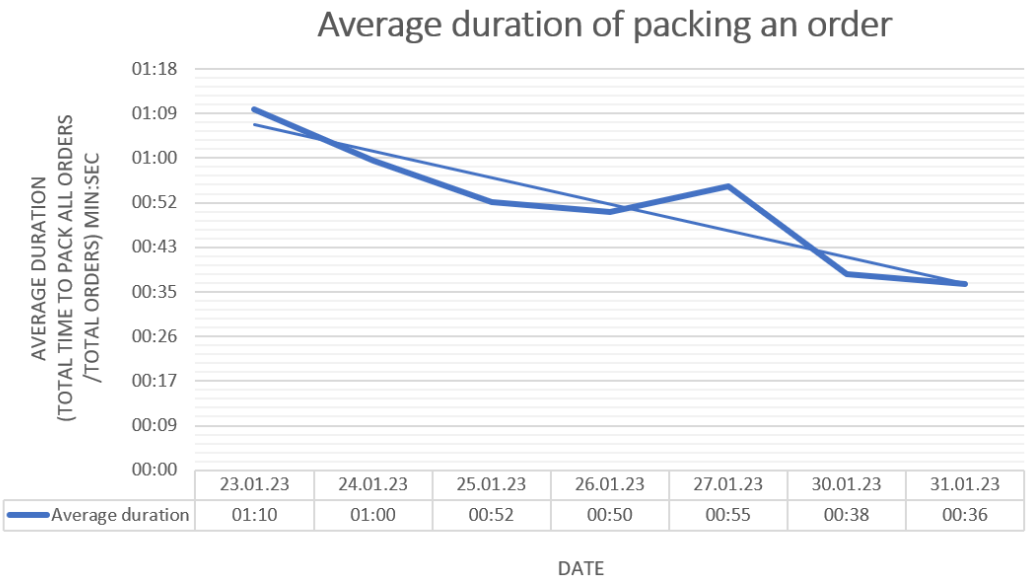


Abbildung 14: Zeit, um einen Auftrag abzuarbeiten

An jedem Tag, an dem das Experiment durchgeführt wurde, wurde notiert, wie viele Aufträge es gab und wie lange die Durchschnittswerte von mehreren Tagen sind. Die Werte können in einem Diagramm betrachtet werden, welches in Abbildung 14 zu sehen ist. Die x-Achse der Tabelle bezieht sich auf das Datum und die y-Achse auf die Dauer in der Einheit Minute. Mit dem aktuellen genutzten Perixx-Outbound-Prozess dauert es durchschnittlich

1 Minute einen Auftrag komplett für den Versand abzuarbeiten. Dabei wurden aber die benötigten Dokumente schon vorab gedruckt und das Abschlussprüfen ob alles richtig eingepackt wurde ausgelassen. Dieser Wert wurde ähnlich wie bei den Messungen mit der POA ausgerechnet. Der Unterschied ist, dass dafür keine POA genutzt wurde und, dass der Durchschnittswert von 20 Messungen genommen wurde. Wie in Abbildung 14 zu sehen ist, dauerte es mit der POA am ersten Tag durchschnittlich 1 Minute und 10 Sekunden für einen Auftrag. Betrachtet man die anderen Tage, sieht man wie die durchschnittliche Verarbeitungsdauer fast täglich sinkt und am siebten Tag bei nur noch 36 Sekunden liegt. Auch wurde in der Zeit kein einziger Fehler gemacht. Der Grund für den großen Unterschied zwischen Tag 1 und Tag 7 könnte daran liegen, dass sich der Testnutzer an die Arbeitsweise mit der POA vertraut machen musst. Da das Experiment nur 7 Tage lang gedauert hat, kann er nicht das volle Potential der POA widerspiegeln. Dieser Test zeigt jedoch eine Steigerung der Effizienz und Effektivität durch Nutzung der POA.

7 Ausblick

In diesem Kapitel wird auf Funktionalitäten eingegangen, die in Zukunft verbessert werden können. Dabei wird unterschieden, ob es sich um eine bereits implementierte und noch zu verbessernde Funktion oder um eine neue Funktion handelt.

7.1 Verbesserung der bestehenden Funktionalitäten

Instabile Verbindung mit der Datenbank : Jedes Mal, wenn die Internetverbindung kurz unterbrochen und wiederhergestellt wird, wird die Verbindung mit dem MySQL-Server nicht automatisch hergestellt, und es ist nicht möglich, die Verbindung manuell herzustellen, es sei denn, der POA wird vollständig neu gestartet. Bezüglich auf das Problem wurden ähnliche Fragen im MYSQL1-Github-Forum[13]. gestellt aber noch nicht behoben.

Nicht unterstütztes Sidebar-Widget auf der IOS Plattform : Laut [8], empfiehlt Apple das Sidebar-Widget nicht. Ein anderes Widget für die Spracheinstellung muss gesucht und eingesetzt werden.

Manchmal klein gedruckte Versandetikette : Bei der Verwendung von POA wurde festgestellt, dass die Versandetiketten nicht immer in der gleichen Größe gedruckt werden, obwohl sie alle die gleiche Druckeinstellung haben. Bisher wurde keine geeignete Ursache und Lösung gefunden.

[13] https://github.com/adamlofts/mysql1_dart/issues/99

7.2 Implementierung neuer Funktionalitäten

Erweiterung auf andere Einkaufskanäle : Momentan ist die POA einsetzbar nur für einen Einkaufkanal. Die POA könnte daher weiter entwickelt werden, damit Aufträge von anderen Einkaufkanälen durch die POA verschickt werden können.

Suche nach einem Auftrag über Artikelnummer statt EAN : Derzeit gibt es nur eine Möglichkeit, nach einem Artikel zu suchen, nämlich über die EAN des Artikels. Für den Fall, dass der Barcode defekt ist oder die Scanpistole nicht funktioniert, könnte eine neue Funktion hinzugefügt werden, um einen Artikel anhand seiner Artikelnummer zu finden. Diese Funktion erfordert eine Tastatur, um die Nummer einzugeben.

8 Fazit

Für diese Bachelorarbeit wurde eine mobile Anwendung konzipiert und realisiert, die es ermöglicht, Lagerarbeiten auf Basis des Perixx-Outbound-Prozesses zu digitalisieren und zu optimieren. Bei der Planung wurden Anforderungen von zukünftigen Nutzern gesammelt und daraus eine vereinbarte Anforderungsliste erstellt, wodurch eine schnelle und präzise Implementierung erzielt werden konnte. Mit der Hilfe von Flutter konnte die Anwendung so entwickelt werden, dass sie mit einer einzigen Codebasis sowohl auf der Android- als auch auf der IOS-Plattform läuft. Bei der Implementierung wurden jedoch einige Un terschiede zwischen den beiden Plattformen festgestellt. Zum Beispiel, dass Apple das Sidebar-Widget nicht unterstützt und Android die Methode `directPrintingPDF` nicht unterstützt. Als Architektur wurde die `Layer-First` Architektur gewählt, welche den ge samten Perixx-Outbound-Prozess abdeckt. Dieser wurde in die vier Features Einloggen, Auftragsliste, Scannen und Drucken unterteilt.

Weiter wurde auf die Tests eingegangen, die durchgeführt wurden um eine höchstmögliche Qualität der implementierten Komponenten zu gewährleisten. Zum Schluss wurde eine Evaluation der entwickelten POA durchgeführt. Dabei sollte dessen Effizienz im Vergleich zu dem von Perixx aktuell genutzten Outbound-Prozess überprüft werden. Dafür haben Test-nutzer, die im aktuellen Outbound-Prozess involviert sind, die POA ausprobiert und deren Eindrücke in Form eines Fragebogens wiedergegeben. Dieses Ergebnis zeigt die Effizienz der POA. Die meistens Teilnehmer waren zufrieden mit der POA und wünschten sich, dass die POA auch auf andere Einkaufskanäle erweitert wird. Darüber hinaus zeigte die Evaluation auch, dass die POA den Outbound-Prozess beschleunigt und die Wahrscheinlichkeit von auftretender menschlicher Fehler reduziert.

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungen

API Application Programming Interface

ASCII American Standard Code for Information Interchange

EAN European Article Number

EDV Elektronische Datenverarbeitung

POA Perixx Outbound App

UI User Interface

Literatur

[1] Code With Andrea. Flutter project structure. Zuletzt aufgerufen am 30.01.2023. URL: https://codewithandrea.com/articles/flutter-project-structure/#:~:text=Flutter%20App%20Architecture%20using%20data,application%3A%20services.

[2] AWS. What is flutter. Zuletzt aufgerufen am 30.01.2023. URL: https://aws.amazon.com/de/what-is/flutter/.

[3] DavBfr. How to send pdf to a print specific directly? Zuletzt aufgerufen am 30.01.2023. URL: https://github.com/DavBfr/dart_pdf/issues/567.

[4] GetX. Getx package 4.6.5. Zuletzt aufgerufen am 30.01.2023. URL: https://pub.dev/packages/get/.

[5] google. Flutter docs. Zuletzt aufgerufen am 30.01.2023. URL: https://docs.flutter.dev/.

[6] google. Fundamentals | firebase documentation. Zuletzt aufgerufen am 30.01.2023. URL: https://firebase.google.com/docs/guides.

[7] Jonataslaw. Getx state management. Zuletzt aufgerufen am 30.01.2023. URL: https://github.com/jonataslaw/getx/blob/master/documentation/en_US/state_management.md.

[8] Kaspi. Flutter multi-platform android / ios drawer menu. Zuletzt aufgerufen am 20.01.2023. URL: https://stackoverflow.com/questions/58117191/flutter-multi-platform-android-ios-drawer-menu.

[9] Philip A Laplante. *What every engineer should know about software engineering*. CRC Press, 2007.

[10] Doug Stevenson. What is firebase? the complete story, abridged., Oct 2018. URL: https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0.