Digitaler Lieferschein

Studienarbeit

Frühjahrssemester 2022

Mathias Lenz
Khoa Tran

Betreuer: Prof. Frank Koch
Projektpartner: AdaptIT GmbH, St. Gallen

Computer Science
University of Applied Sciences of Eastern Switzerland
Rapperswil-Jona

Abstract

Für die Logistik zwischen Firmen werden in der Regel Lieferscheine in Papierform verwendet. In Rahmen dieser Arbeit soll ein Prototyp einer Applikation entwickelt werden, der den Lieferschein digitalisiert.

Die Applikation soll für Absender, Spediteure und Empfänger gleichermassen nutzbar sein und die jeweiligen Bedürfnisse abdecken. Die zentrale Aufgabe ist es, die Lieferung zu kommissionieren, unterwegs nachzuverfolgen und zum Schluss anhand des Lieferscheins den Wareneingang zu machen. Für die Identifikation der Lieferungen wird ein QR-Code verwendet.

Die Entwicklung wird mit agilem Projektplanung und -management durchgeführt. Nach der Analyse der Domäne wurde ein Konzept für die Realisation erstellt. Anschliessend wurde die Applikation entwickelt und in die Cloud deployt.

Der Use Case Katalog zeigt auf, welche Funktionen benötigt und von welchen Akteuren diese genutzt werden. Auf der Konzeptebene ist der zu digitalisierende Prozess stark statusabhängig. Jede Lieferung hat verschiedene Zustände, die ihrerseits unterschiedliche Aktionen zulassen. Das Resultat der Umsetzung ist eine lauffähige und deployte Applikation, die den Anforderungen gerecht wird. Durch Usability Tests wird die Benutzbarkeit der Applikation bestätigt.

Als nächster Schritt wird empfohlen, die Applikation in der Industrie vorzustellen und Partner zu gewinnen. Mit diesen zusammen kann dann evaluiert werden, was noch an Features benötigt wird und welche Teile der Applikation ausgebaut werden sollen.

Management Summary

Digitaler Lieferschein

Studenten	Khoa Tran, Mathias Lenz
Betreuer	Prof. Frank Koch
Themengebiet	Application Design
Industriepartner	AdaptIT GmbH, Michael Güntensperger, St. Gallen

Aufgabenstellung

Unser Auftrag war es ein Prototyp für eine Webapplikation zu entwickeln, um den Zugriff auf Informationen innerhalb der Lieferkette zu digitalisieren. Meist werden Lieferscheine in der Industrie manuell verarbeitet, wodurch fehleranfällige und aufwändige Mehrfacherfassungen entstehen können.

Der Fokus ist der Prozess ausgehend von der Bestellerfassung über das Tracking der Lieferung bis hin zum Wareneingang. Im Zentrum steht ein QR-Code, der eine Lieferung eindeutig identifiziert und der die verschiedenen Akteure direkt auf die Anwendung führt. Dort soll der Verlauf einer Lieferung nachverfolgt werden können.

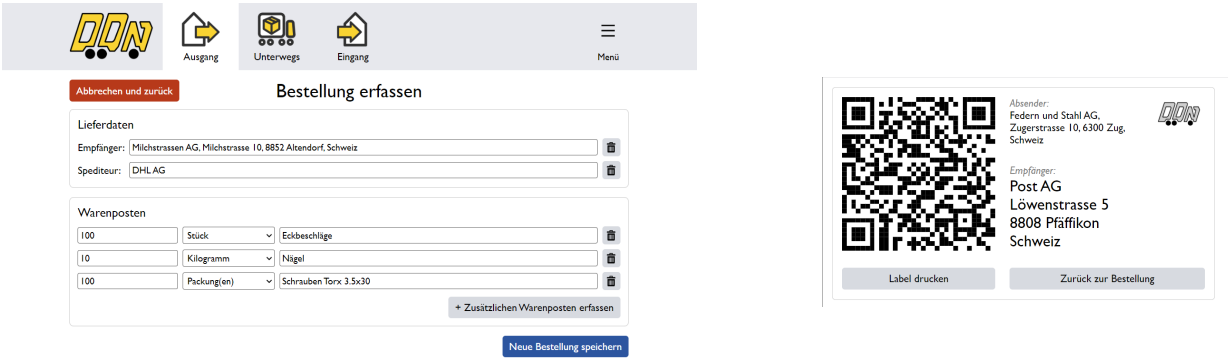


Abbildung 1: Formular zur Bestellerfassung und QR-Code auf Versandetikette

Vorgehen

In der Analyse stellten wir fest, dass es sich um einen Businessprozess handelt, der stark statusabhängig ist. Durch die drei Hauptnutzergruppen Spediteure, Absender und Empfänger ergeben sich verschiedene Perspektiven auf die jeweiligen Status einer Lieferung.

Damit der QR-Code für die verschiedenen Status und Parteien funktioniert, wurde ein Ablauf implementiert, der sicherstellt, dass die Benutzer nach einem Scan auf die korrekte Webseite weitergeleitet werden.

Neben dem QR-Code ist das Dashboard der zentrale Zugangspunkt zur Applikation. Mit dem Dashboard können alle Lieferungen, an denen ein Benutzer beteiligt ist, verarbeitet und nachverfolgt werden.

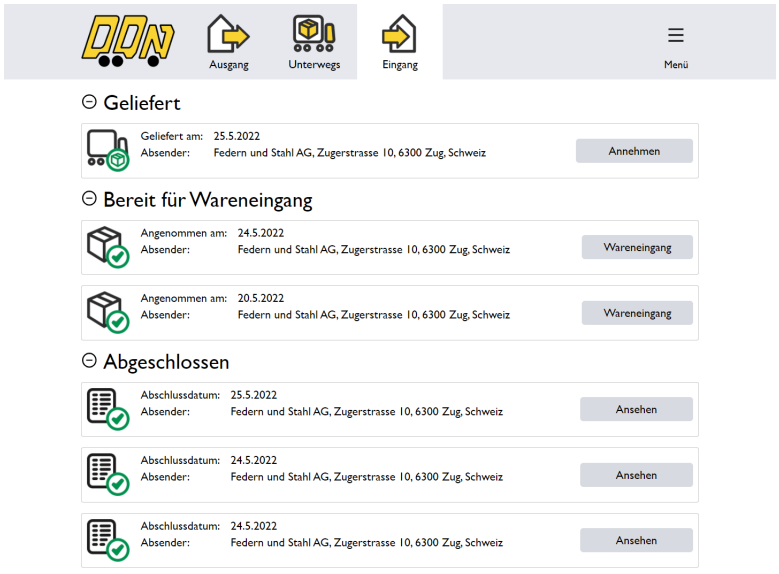


Abbildung 2: Das Dashboard mit einigen exemplarischen Lieferungen

Ergebnisse

Die resultierende Webapplikation bildet die gesamte Lieferkette ab. Durch die verschiedenen Nutzergruppen und die vielen Zustände, die eine Lieferung durchlaufen kann, ergibt sich eine Vielzahl an Möglichkeiten und Optionen. Durch eine übersichtliche Gestaltung der Nutzeroberfläche konnte sichergestellt werden, dass die Bedienung trotzdem einfach bleibt.

Danksagungen

Als erstes möchten wir uns herzlich bei Frank Koch für die Betreuung bedanken. Natürlich geht unser Dank auch an Michael Güntensperger für die Beantwortung unserer Fragen und die Inputs. Des weiteren möchten wir uns bei den Testpersonen für unsere Usability Tests als auch bei den Gegenlesern bedanken.

Inhaltsverzeichnis

Abstract **2**

Management Summary **3**

Danksagungen **5**

Aufgabenstellung **8**

1 Einleitung **12**

2 Analyse **13**
2.1 Begriffsdefinitionen . . . 13
2.2 Domain Model . . . 14
2.3 Vision . . . 15
2.4 Use Cases . . . 15

3 Design **19**
3.1 Businessprozess . . . 19
3.2 Berechtigungen . . . 21
3.3 User Interface . . . 23
3.4 Datenmodell . . . 29

4 Umsetzung **31**
4.1 Technologien . . . 31
4.2 Gesamtsystem Architektur . . . 32
4.3 Umsetzung Backend . . . 32
4.4 Umsetzung Frontend . . . 35
4.5 Logging . . . 37
4.6 Error-Handling Backend . . . 38
4.7 API . . . 39
4.8 CI / CD . . . 42
4.9 Deployment . . . 42

5 Qualitätssicherung **46**
5.1 Usability Testing . . . 46

6 Ergebnisse **48**
6.1 Evaluation der nichtfunktionalen Anforderungen (NFA) . . . 48
6.2 Bezug zur Aufgabenstellung . . . 48

6.3 Empfehlungen 50

7 Projektauswertung **53**

8 Zusammenfassung **54**
8.1 Schlusswort 54
8.2 Persönliche Reflexionen 54

9 Anhang **56**
9.1 Glossar 57

OST
Ostschweizer
Fachhochschule

Aufgabenstellung Semesterarbeit „Digitaler Lieferschein“

Autor: **Frank Koch**

Version: **1.0**

Anpasst am: **07. Oktober 2021**

1. Beteiligte Personen

- Studierende: Khoa Tran und Mathias Lenz
- Industriepartner: AdaptIT GmbH, Michael Güntensperger
- Betreuer: Frank Koch

2. Problembeschrieb

Lieferscheine werden vom Lieferanten ausgedruckt, mit der Ware versendet und vom Empfänger angenommen. Bei der Warenannahme in Industrieunternehmen müssen die Lieferscheine eingelesen und mit den zugehörigen Bestellungen abgeglichen werden. Zudem müssen die Waren auf Vollständigkeit geprüft und ins Lager eingebucht werden. Die resultierende Mehrfacherfassung und Medienbrüche machen diesen Prozess aufwändig und fehleranfällig.

Mit dieser Arbeit soll der Prozess digitalisiert werden. Die Lieferscheine sollen aus bestehenden Tools exportiert werden können (in einem ersten Schritt sollen sie manuell online ausgefüllt werden können). Zu jedem Lieferschein soll ein QR-Code (oder ähnliches) generiert werden, der vom Spediteur und Empfänger eingelesen werden kann. Die Daten werden in einer Web-Applikation verwaltet, so dass alle Beteiligten einsehen können, wer logistisch wann was gemacht hat. Optional könnte hierzu eine Blockchain eingesetzt werden.

3. Aufgabenstellung

Grundlage dieser Aufgabenstellung ist der für das FS22 gültige Leitfaden für Bachelor- und Studienarbeiten[1].

Diese Arbeit hat das Ziel ein erstes abgespecktes MVP ohne Integrationen in z.B. SAP zu erstellen.

Technische Umgebung

Für die Umsetzung wird mit Web-Technologien gearbeitet.

- JavaScript
- Node.js
- React / Angular / Vue

Funktionale Anforderungen

- Registration neuer Kunden
- Eingeschränkter Zugriff auf die Applikation / Daten (Versender / Spediteur / Zoll / Empfänger)
- Erfassen einer Bestellung / eines Lieferscheins mit allen Daten
- Generieren eines Codes, der auf ein Paket geklebt werden kann.
- Scannen des QR-Codes durch alle beteiligten
- Verfolgung des Paketes (wer hat es zuletzt gescannt)
- Deployment in die Cloud (DigitalOcean)

Optionale Anforderungen

1. Drucken des QR-Codes von einem Etikettendrucker

[1] https://teams.microsoft.com/l/file/A7522E3F-6931-46F7-A965-B2A9081E71EF?tenantId=a6e70fa3-1c7a-4aa2-a25e-836eea52ca22&fileType=pdf&objectUrl=https%3A%2F%2Fostch.sharepoint.com%2Fteams%2FTS-StudiengangInformatik%2FFreigegebene%20Dokumente%2FStudieninformationen%2FLeitfaden%20BA%20SA%20v1.0.pdf&baseUrl=https%3A%2F%2Fostch.sharepoint.com%2Fteams%2FTS-StudiengangInformatik&serviceName=teams&threadId=19:88ac24bbee8e4682a73e81839cd2103c@thread.tacv2&groupId=4ad37fbb-4c36-4982-8bb5-971b8a539d2d

2. Verteilung der Sendung auf mehrere Teile (Paket / Palett)
3. Integration in SAP
4. Entwicklung eines Mobile-Apps für das Scannen
5. Entwicklung auf einer Blockchain statt einer Datenbank
6. Verknüpfung mit der ID des Distributors
7. Existierende Tracker-Tools anbinden, so dass die Sendung per GPS getrackt werden kann.

Nicht-Funktionale Anforderungen

- Das Entwicklerteam implementiert die Features gemäss der abgesprochenen Priorität mit dem Kunden
- Wenn ein QR-Code gescannt wird, sollte es nicht länger als 1sek. dauern, bis ein Resultat angezeigt wird
- Das Backend sollte 1000 Requests pro Minute handeln können
- Jede Seite sollte nicht länger als 200ms für das Laden benötigen
- Die Web-Applikation sollte auf Firefox, Chrome und Safari auf Windows Surface Tablets, Android und Apple Handys laufen.
- Via Internet sollte auf eine vom Kunden zur Verfügung gestellte Domain zugegriffen werden können.
- Drei von vier Test-user sollten das UI (Kategorien: layout, responsiveness, colour, content) der Applikation mit einem Tablet mit einer Note von mindestens 8 von 10 bewerten, wobei 10 das beste ist.
- Die Datenbank soll bis zu 10'000 Lieferscheine managen können.
- Errors sollen keine Systemfehler erzeugen, aber eine Error Nachricht Zeigen und das System auf den vorherigen Zustand zurücksetzen.
- Jeder Error soll im System geloggt werden
- Jede Kommunikation zwischen Fron- und Backend soll mit einem SSL-Zertifikat verschlüsselt werden.
- Daten welche in Eingabefelder abgefüllt werden, sollen zuerst validiert werden, bevor diese durch das System verarbeitet werden. SQL Injection test der Eingabefelder sollte keine Verletzlichkeiten zeigen.
- User-Passwörter werden nicht in plain-text in der Datenbank gespeichert.
- Wenn sich ein User in die Web-Applikation einloggt, werden ihm auch nur seine Daten / auf Daten die er Zugriff haben soll, angezeigt.
- Businesslogik im Backend soll modular aufgebaut werden, so dass sie erweitert werden kann.
- Die Backend-API soll durch API-testing Tools getestet werden.
- Datenbank, Backend und Frontend sollen auf unterschiedlichen Instanzen deployed werden.

4. Zur Durchführung

Mit dem Betreuer finden Besprechungen gemäss Absprache statt. Die Besprechungen sind von den Studierenden mit einer Traktandenliste vorzubereiten und die Ergebnisse in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben.

5. Dokumentation und Abgabe

Siehe Leitfaden Abschnitt 5.5 "Umfang und Form der Abgabe".

6. Termine

Es gelten folgende Termine:

- 21.02.2022 Beginn der Studienarbeit, Ausgabe der Aufgabenstellung durch den Betreuer/die Betreuerin.

Vorlagen sowie eine ausführliche Anleitung betreffend Dokumentation stehen auf Teams zur Verfügung.

- 31.05.2022 Die Studierenden geben den Abstract zur Kontrolle an ihren Betreuer/Examinator frei. Die Studierenden erhalten vorgängig vom Studiengangsekretariat die Aufforderung mit den Zugangsdaten zur Online-Erfassung des Abstracts.
- 03.06.2022 Der Betreuer/die Betreuerin gibt das Dokument mit dem korrekten und vollständigen Abstract zur Weiterverarbeitung an das Studiengangsekretariat frei.
- 03.06.2022 Hochladen aller verlangten Dokumente auf https://avt.i.ost.ch/. Abgabe des Berichts an den Betreuer/die Betreuerin bis 17.00 Uhr

7. Bewertung

Siehe Leitfaden Abschnitt 6 "Bewertung", insbesondere 6.4.

Rapperswil, den 21.02.22

Frank Koch

1 Einleitung

Analoge Lieferscheine sind fehleranfällig und aufwändig. Im Rahmen der Studienarbeit soll ein MVP erstellt werden, der den analogen Lieferschein durch eine digitale Variante ablöst. Die entwickelte Applikation soll neben eines optimierten und weniger fehleranfälligen Prozesses auch zusätzliche Vorteile bringen. Dazu gehört eine Sendungsverfolgung und die Nachvollziehbarkeit von Aktionen auf einer Lieferung. Der Fokus für die Realisierung liegt neben der Funktionalität auf guter Bedienbarkeit und ansprechender Gestaltung.

Nach einer eingehenden Domainanalyse wird ein Konzept erarbeitet, wie der digitale Lieferschein optimal umgesetzt werden kann. Dieses Konzept wird dann mit Web-Technologien realisiert und in die Cloud deployt. Das MVP wird abschliessend mittels Usability Tests geprüft, angepasst und es wird evaluiert, wie das Projekt in Zukunft weitergeführt werden könnte.

2 Analyse

2.1 Begriffsdefinitionen

In der Analyse der Problemstellung sind verschiedene Begriffe aufgekommen. Diese sind im folgenden genauer definiert, damit eindeutig ist, was damit gemeint ist. Daneben gibt es weitere Begriffe, die in den folgenden Abschnitten wichtig werden. Da gerade im Code aber auch in anderen Bereichen Englisch verwendet wird, ist jeweils auch der englische Begriff notiert.

Organisation (engl. organization) Eine Organisation ist ein Betrieb, die sich an der Applikation beteiligt. Als Organisationen gelten Firmen, Spediteure und Zoll.

Firma (engl. company) Firmen beteiligen sich aktiv am Logistiknetzwerk der Applikation. Sie können als Absender und Empfänger fungieren. Kunde kann synonym mit Firma verwendet werden.

Absender (engl. sender) Absender erfassen Bestellungen, kommissionieren und versenden Lieferungen.

Empfänger (engl. receiver) Empfänger können Lieferungen annehmen und im Wareneingang auf vollständigkeit kontrollieren.

Benutzer (engl. user) Benutzer sind einer Organisation zugeordnet. Benutzer können sich anmelden und die Software nutzen. Eine Organisation kann mehrere Nutzer haben.

Zoll (enlg. customs) Die Zoll verarbeiten Lieferungen bei Grenzübergängen.

Spediteur (engl. shipper) Spediteure übernehmen den Transport von Lieferungen.

Bestellung (engl. order) Eine Bestellung beinhaltet eine Lieferung.

Lieferung (engl. shipment) Von einer Lieferung ist die Rede, sobald eine Bestellung kommissioniert wurde. Damit ist die physische Ware gemeint, die von einem Ort an einen anderen geliefert wird.

Warenposten (engl. itemlist) Warenposten beschreiben die Ware sowie deren Anzahl, die zu versenden sind.

Statuslog (engl. status history) Der Statuslog bildet den gesamten Verlauf der Lieferung durch eine Liste mit den wichtigen Ereignissen ab. Sendungsverfolgung kann synonym mit Statuslog verwendet werden.

2.2 Domain Model

Für die Analyse der Domain wurde ein Domain Model (Abbildung 2.1) erstellt. Im Zentrum steht hierbei die Lieferung, welche zu einer bestimmten Bestellung gehört und immer einen Absender, einen Spediteur sowie einen Empfänger hat. Die Lieferung beinhaltet den für den Versand essenziellen QR-Code und kann mehrere Warenposten haben.

Absender, Spediteur sowie Empfänger sind jeweils Benutzer mit einem Login und gehören zu einer bestimmten Organisation.

Damit der ganze Verlauf der Lieferung zeitlich nachverfolgt werden kann, gibt es zudem ein Statuslog, in welchem für jede Änderung des Lieferungsstatus ein Eintrag erstellt wird.

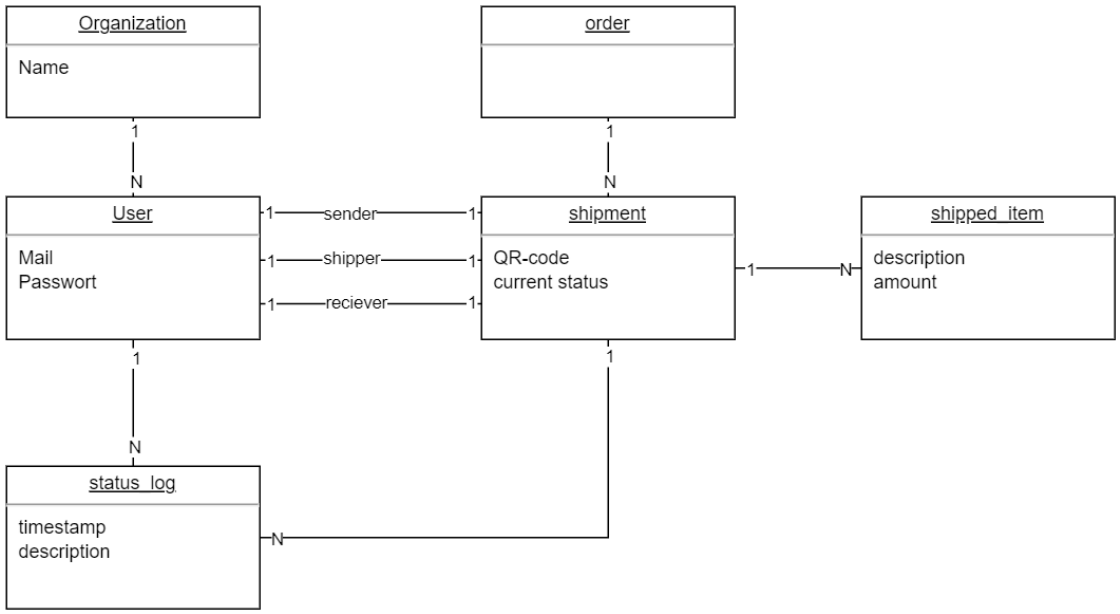


Abbildung 2.1: Das Domain Model zeigt die Zusammenhänge zwischen einzelnen Klassen aus der Domain

2.3 Vision

Ausgehend von der Domainanalyse, anhand der Definitionen und mit den Informationen zum Prozess ergibt sich ein ungefähres Bild, was die Applikation können muss. Die Applikation ist eine Möglichkeit, Bestellungen zu erfassen und diese zu verarbeiten. Es braucht die Möglichkeit, direkt auf eine Lieferung zuzugreifen und es muss die Bedürfnisse der verschiedenen Akteure abbilden.

Es gibt im wesentlichen zwei verschiede Hauptnutzergruppen. Dies sind einerseits Firmen als Kunden, Empfänger und Absender, andererseits Spediteure als auch Zoll als Beitragende zum Service. Firmen benötigen eine umfangreiche Funktionen zur Verarbeitung der Lieferung und eine gute Bedienbarkeit. Spediteure und Zoll benötigen dagegen eine hohe Effizienz in der Bedienung als auch eine schnelle und einfache Verarbeitung.

Daher soll es für Firmen ein Dashboard geben, dass alle Bestellungen anzeigt und die verarbeitung der Bestellungen ermöglicht. Für Spediteure und Zoll soll es die Möglichkeit geben, mittels QR-Code direkt auf eine Lieferung zuzugreifen.

2.4 Use Cases

Für die geforderte Funktionalität wurden insgesamt 16 Use Cases erarbeitet, die in vier Gruppen aufgeteilt wurden. Der erste Bereich ist die Authentifizierung mit Registration und Login. Dann gibt es Use Cases für das Dashboard, das in erster Linie der Übersicht über die Lieferungen dient. Die Usecases für die Bestellungen decken das Verarbeitungsspektrum über den Bestellungen und Lieferungen ab. Schlussendlich sind die QR-Code Use Cases dafür da, die Funktionen des QR-Codes zu erfüllen. In der Abbildung 2.2 sind die Use Cases im Verhältnis zu Ihren Akteuren dargestellt.

Use Cases sind darauf ausgelegt, eine hypotetische Lieferkette darzustellen und zu bedienen. Verschiedene Entscheidungen wurden daher so getroffen, dass sie praktikabel zu implementieren sind, um den Funktionsumfang abzudecken, sind aber für den praktischen Gebrauch nicht praktikabel.

Für diesen Abschnitt wesentliche Kenntnisse wurden im Modul SE2 [5] erarbeitet.

Authentifizierung

- (UC.1) Registration: Firmen können sich bei der Plattform registrieren. Sie werden so zu Kunden. Es können sich mehrere Nutzer pro Firma registrieren.
  - Neue Registrierende werden automatisch als Firma erfasst.
  - Für Zoll und Spediteure werden Logins vorab erstellt.

  - Für die Registration kann ein Registrationscode verwendet werden, um sich einer bestehenden Firma anzuschliessen, oder es wird eine neue Firma erstellt, wobei auch eine Adresse erfasst wird.
  - Der Registrationscode eines Nutzers ist im Dashboard ersichtlich.
- (UC.2) Login: Firmen wie auch Spediteure und Zoll können sich mit Email und Passwort einloggen.

Dashboard

- (UC.3) Dashboard Ausgang: Eine Firma kann die im Dashboard erfassten und abholbereiten Bestellungen sehen. Vom Dashboard aus können Bestellungen erfasst und kommissioniert werden.
- (UC.4) Dashboard Unterwegs: Im Dashboard kann die Firma die eigenen versendeten und eingehenden Bestellungen ansehen. Ebenfalls kann Status-History der Bestellungen eingesehen werden.
- (UC.5) Dashboard Eingang: Sobald der Spediteur die Lieferung als zugestellt markiert, wird sie im Wareneingang des Kunden angezeigt. Der Kunde kann vom Dashboard aus die Lieferung annehmen und die Wareneingangskontrolle starten.

Bestellungen verarbeiten

- (UC.6) Bestellung erfassen: Benutzer können eine neue Bestellung erfassen. Dazu wird Empfänger, Spediteur und die Warenposten erfasst.
  - Für die Erfassung des Empfängers und des Spediteurs gibt es ein Eingabefeld mit Autovervollständigung für alle bestehenden Empfänger beziehungsweise Spediteure.
  - Bei den Warenposten soll Anzahl, Beschreibung und eine Einheit erfasst werden. Die Einheiten sollen sich an den Standard von SAP halten.
  - Die Firma des erstellenden Benutzers wird automatisch als Absender eingerichtet. Falls eine Firma mehrere Adressen hat, kann eine davon ausgewählt werden.
- (UC.7) Bestellung bearbeiten: Absender können Änderungen an den Bestellposten vornehmen oder den Spediteur anpassen. Auch können Absender eine Bestellung stornieren. Änderungen oder Stornierungen sollen nur möglich sein, solange die Lieferung noch nicht kommissioniert ist.
- (UC.8) Kommissionierung: Für die Kommissionierung kann die Firma sich die Bestellung ansehen. Einzelne Warenposten können als verpackt und die Bestellung/Lieferung dann als versandbereit markiert werden.
  - Für jede Lieferung wird ein QR-Code mit der URL der Lieferung generiert. Dieser kann als Versandetikette, zusammen mit den Adressen von Absender und Empfänger gedruckt werden.
- (UC.9) Wareneingang: Eine Firma kann Lieferungen annehmen, sie auf vollständigkeit prüfen und Probleme erfassen.
- (UC.10) Statuserfassung: Der Spediteur sieht den Absender, Empfänger und die Statushistorie der aktuellen Lieferung. Bei der Lieferung kann er einen Eintrag im Statuslog hinterlegen.

  - Ein Statuslogeintrag umfasst immer den Benutzer, der die Änderung vorgenommen hat, den Zeitpunkt und den aktuellen Status. Darüber hinaus gibt es die Möglichkeit, einen Standort und einen Kommentar zu erfassen.
- (UC.11) Sendungsverfolgung: Absender, Empfänger und Spediteur können den Statuslog ansehen. Im Statuslog werden alle Statusveränderungen der Bestellung festgehalten.
- (UC.12) Verzollung: Der Zoll kann eine Lieferung als verzollt markieren.

QR-Code

- (UC.13) Scan Kunde: Beim Scannen des QR-Codes wird der Kunde automatisch auf die Lieferung weitergeleitet und kann diese gemäss dem aktuellen Status bearbeiten/ansehen.
- (UC.14) Scan Spediteur: Ein Spediteur wird durch den Scan des QR Codes auf die Lieferung weitergeleitet.
- (UC.15) Scan Zoll: Ein Zoll kann durch Scannen des QR Codes Warenliste der Lieferung einsehen.
- (UC.16) Scan ohne Login: Bei einem Scan eines QR-Codes ohne angemeldet zu sein wird ein Anmeldefenster angezeigt. Nach der Anmeldung wird der Benutzer, je nach Status und Berechtigung, auf die korrekte Ansicht weitergeleitet oder blockiert.

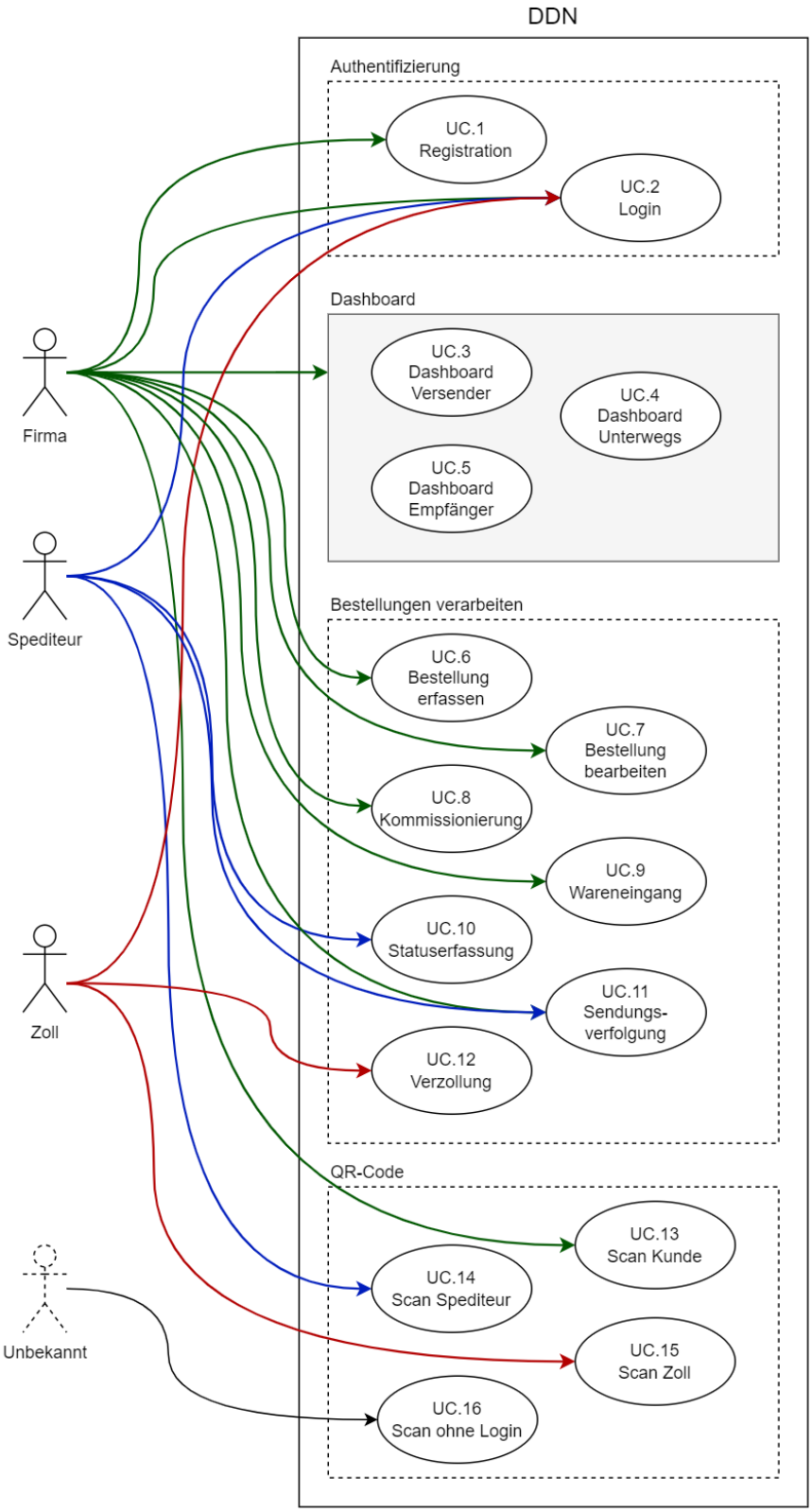


Abbildung 2.2: Use Case Diagramm

3 Design

3.1 Businessprozess

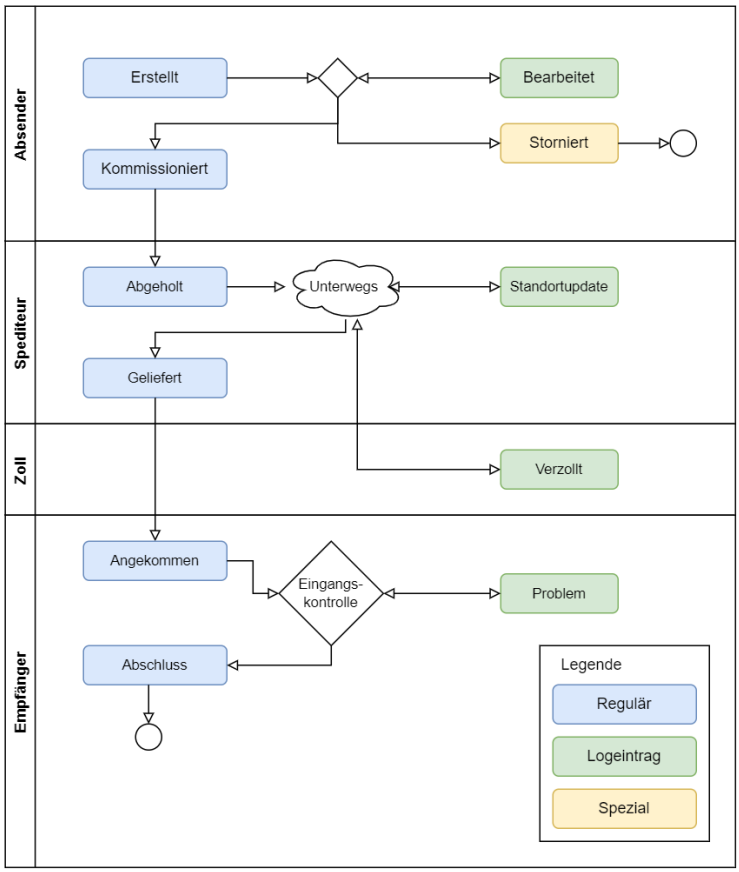


Abbildung 3.1: Statusdiagramm mit allen möglichen Status und Logeinträgen, die einer Lieferung zugewiesen können.

Anhand der vom Projektpartner zur Verfügung gestellten Unterlagen wurde ein Prozess modelliert, nach dem Bestellungen verarbeitet werden können. Eine Bestellung durchläuft verschiedene Status. Der Status einer Bestellung zeigt die aktuelle Situation und bestimmt die Sichtbarkeit und verfügbaren Aktionen. Der Ablauf ist dabei deterministisch und unidirektional.

Für die Sendungsverfolgung gibt es neben den Status weitere Einträge, die zusätzliche Informationen bereitstellen, aber die möglichen Aktionen nicht beeinflussen. In Abbildung 3.1 werden

die einzelnen eingefärbten Logeinträge im Kasten des jeweiligen Akteurs dargestellt. Es gibt drei Arten von Einträgen im Statuslog. Bei den regulären Statusveränderungen (blau) wird der Status der Bestellung geändert. Die reinen Logeinträge (grün) verändern den Status einer Bestellung nicht. Ein Spezialstatus (gelb) wird nur bei Spezialereignissen erfasst und ist abschliessend. Der Status wird dabei geändert. Aktuell gibt es nur *Storniert* als Spezialstatus. Der Vorteil der Unterscheidung zwischen Status und Logeintrag ist, dass die Verarbeitung weniger komplex wird, und die möglichen Aktionen verständlich und berechenbar sind. In der folgenden Auflistung sind die einzelnen Status und Einträge im Detail erklärt.

Erstellt Dieser Status ist der Initialstatus und wird einer neu erstellten Bestellung zugewiesen.

Bearbeitet Der Status *Bearbeitet* wird Bestellungen zugewiesen, die noch verändert werden mussten. Dies kann nötig sein, wenn beispielsweise der Besteller anruft und noch einen Änderungswunsch hat. Es ist vorgesehen, dass dies nur vor der Kommissionierung möglich ist.

Storniert Eine Bestellung kann durch den Absender storniert werden. Danach ist die Bestellung nicht weiter verarbeitbar.

Kommissioniert Sobald der Absender die Kommissionierung durchgeführt hat, ist die Lieferung verpackt und abholbereit. Bei der Kommissionierung muss auch die Versandetikette mit dem QR-Code gedruckt werden.

Abgeholt Wenn der Spediteur die Lieferung abholt, scannt er die Lieferung und markiert sie als abgeholt. Ab dann ist sie unterwegs.

Standortupdate Der Standortupdate dient dazu, den Warenfluss unterwegs nachzuvollziehen. Bei Verlust kann dies auch zeigen, wo die Lieferung zuletzt war.

Verzollt Der Zoll kann eine Lieferung als verzollt markieren.

Geliefert Sobald der Spediteur die Lieferung ablädt, markiert er diese als geliefert.

Angekommen Beim Empfänger wird die Ankunft der Lieferung bestätigt.

Problem Falls es beim Wareneingang ein Problem gibt, kann dies mit dem Status *Problem* festgehalten werden.

Abgeschlossen Wenn beim Wareneingang alles passt und allfällige Probleme addressiert wurden, wird durch den Abschluss des Wareneingangs die Lieferung auf den Status *Abgeschlossen* gesetzt.

3.2 Berechtigungen

Einführung

Die Berechtigungen zur Ansicht als auch zur Bearbeitung auf die verschiedenen Lieferungen stellt sich als komplexes Problem heraus. Es hängt nämlich nicht nur von den einer Lieferung zugewiesenen Nutzern ab, sondern auch vom Status einer Sendung. In diesem Dokument wird die Situation dargestellt als auch ein Lösungskonzept erarbeitet. Dabei geht es nicht nur um die Rechte, sondern auch um die Einordnung der Lieferungen in das UI, da dies Hand in Hand geht.

Allgemeines

Hauptgegenstand der Zugriffsberechtigung beziehungsweise -beschränkung ist die Lieferung. Hier befinden sich die Daten, die nicht öffentlich einsehbar sein sollen. Daneben gibt es einen Benutzer-/Einstellungsbereich, der nur für den jeweiligen User einsehbar und veränderbar ist.

- Kunden sehen nach der Anmeldung ein Dashboard, wo Sie Bestellungen erfassen und ein- und ausgehende Lieferungen verfolgen können. Daneben können Sie auch via QR-Code auf Lieferdetails gelangen.
- Zoll und Spediteure haben nur die Möglichkeit, mit einem QR-Code Scan eine Bestellung einzusehen.
- Die Einschränkungen basieren auf dem aktuellen Status der Bestellung und auf der Funktion innerhalb der Sendung (Absender/Spediteur/Empfänger)

Statusbezogene Berechtigung

In der Grafik (TODO Statusablauf) ist ablesbar, welcher Akteur welchen Status erfassen darf. Die Änderung des Status wird durch verschiedene Aktionen im UI verursacht und darf nur von bestimmten Akteuren verändert werden, wie oben beleuchtet. All dies ist in der Tabelle 3.1 festgelegt. Je nach Status ist ein Kommentar möglich beziehungsweise vorgesehen.

Durch das Dashboard kann die Lieferung auf Empfängerseite als auch auf Absenderseite nachverfolgt werden. Die Tabelle 3.2 zeigt auf, welche Rolle (in der Sendung) die Lieferung wo im UI sieht. Für Zoll und Spediteur ist es etwas einfacher. Diese dürfen Bestellungen mit Status Abgeholt scannen und verarbeiten. Zu beachten ist hier, dass der Spediteur nur ihm zugewiesene Bestellungen ansehen darf, der Zoll jedoch darf alle Bestellungen einsehen.

Status/Logeintrag	Akteur	Methode Erfassung	Kommentar
Erstellt	Absender	Speichern neuer Bestellung	
Bearbeitet	Absender	Bestellung bearbeiten	
Storniert	Absender	Bestellung bearbeiten -> Stornierung	
Kommissioniert	Absender	Kommissionierungsprozess	
Abgeholt	Spediteur	Scan -> Statuseintrag	Möglich
Standortupdate	Spediteur	Scan -> Statuseintrag	Möglich
Verzollt	Zoll	Scan -> Verzollung	
Geliefert	Spediteur	Scan -> Statuseintrag	Möglich
Angekommen	Empfänger	Scan/Dashboard -> Ankunft bestätigen	
Problem	Empfänger	Scan/Dashboard -> Wareneingang	Möglich
Abgeschlossen	Empfänger	Scan/Dashboard -> Wareneingang	

Tabelle 3.1: Änderbarkeit des Status einer Bestellung

Status/Logeintrag	Absender	Empfänger
Erstellt	Ausgang - Kommissionsbereit	Nicht sichtbar
Storniert	Ausgang - Storniert	Nicht sichtbar
Kommissioniert	Ausgang - Abholbereit	Nicht sichtbar
Abgeholt	Unterwegs - ausgehend	Unterwegs - eingehend
Geliefert	Unterwegs - ausgehend	Eingang - Geliefert
Angekommen	Unterwegs - ausgehend	Eingang - Angekommen
Abgeschlossen	Unterwegs - ausgehend	Eingang - Abgeschlossen

Tabelle 3.2: Sichtbarkeit einer Bestellung im Dashboard

3.3 User Interface

Jede Software kann immens von einer durchdachten Benutzeroberfläche profitieren. Deshalb ist hier im Detail vorgestellt, welche Überlegungen getroffen wurden und wie das User Interface aufgebaut ist. Das Ziel der Entwicklung war, dass es eine Plattform gibt, die isch für alle Benutzer eignet. Es soll übersichtlich und schnell navigierbar sein. Daneben soll es auch verständlich für uneingeführte Benutzer sein, dabei aber keine unnötigen Schritte einbauen.

Die Abbildung 3.2 zeigt, wie der Aufbau des User Interface ist. Es gibt einen Authentifizierungs-Bereich für die Registration und Anmeldung der Benutzer. Von diesem gelangt man auf die jeweiligen Dashboards. Firmen können auf Ihrem Dashboard direkt auf Bestellungen zugreifen. Dazu dient die Detailansicht, die es in diversen Ausführungen gibt. Alle Nutzer können den QR-Code einer Bestellung scannen und werden entsprechend weitergeleitet.

Für diesen Abschnitt wesentliche Kenntnisse wurden im Modul HCD [7] erarbeitet.

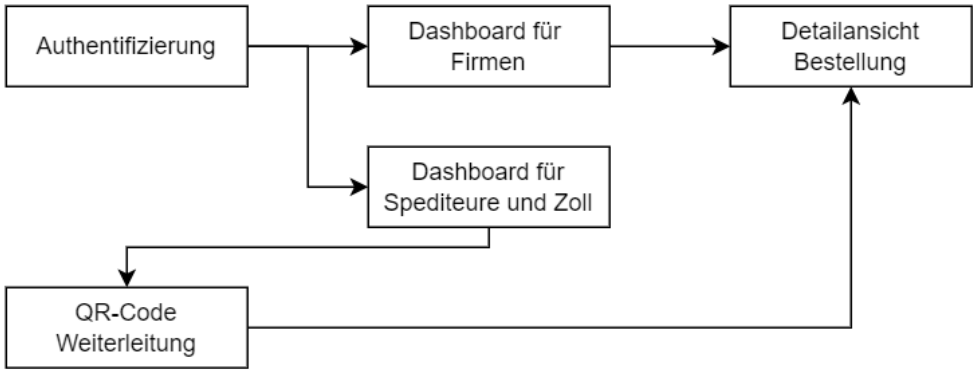


Abbildung 3.2: Übersicht über die wichtigsten Bestandteile des User Interface

Um das Zusammenspiel der einzelnen Views, die möglichen Aktionen und deren Auswirkung auf den Status zu verdeutlichen, werden stellenweise User Flows verwendet. Die Darstellung erfolgt ähnlich wie ein Aktivitätsdiagramm, die Views und Aktionen werden mit Pfeilen verbunden. Varianten derselben View werden durch einen dunkelblauen Rahmen verbunden, wie in der Abbildung 3.3 ersichtlich ist.

Da das UI auf Computer, Tablets und Mobilgeräten funktionieren soll, wurde die Oberfläche responsive gestaltet. Das heisst, dass sich die Elemente automatisch an die gegebene Grösse anpassen. Als Entwicklungsansatz wurde dabei Desktop First gewählt, dabei wird die Komplexität der Struktur für die Mobileansicht reduziert und alles tendentiell untereinander angezeigt.

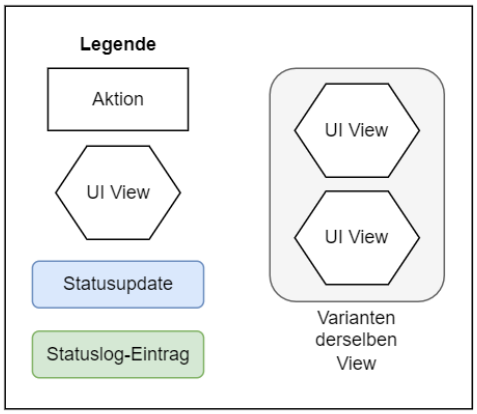


Abbildung 3.3: Legende zu User Flows

Authentifizierung

Da die Applikation in erster Linie nutzerbezogene Aktionen ausführt, braucht es auch eine Möglichkeit, sich anzumelden. Zudem sollen Registrationen möglich sein. TODO

Dashboard

Das Dashboard muss für als Plattform für den Empfang und den Versand von Lieferungen gleichermassen gut funktionieren. Zudem muss es übersichtlich und schnell zu navigieren sein. Es soll dazu mit nur minimaler Einführung verständlich sein.

Das Dashboard besteht im wesentlichen aus einem Navigationsbereich und aus einer Listenansicht. Im Navigationsbereich gibt es neben den drei Hauptbereichen die Option, Details zum Account anzusehen, als auch sich abzumelden. Die Listenansicht ist gruppiert nach Status und dementsprechend benannt. Die einzelnen Listenelemente enthalten die wichtigsten Informationen zu einer Lieferung und den verfügbaren Aktionen. Ein Klick auf eine Aktion bringt den Nutzer zu einer Detailansicht der Lieferung.

Die einzelnen Status sind gemäss Tabelle 3.3 auf die drei Bereiche und die jeweiligen Abschnitte des Dashboards aufgeteilt.

Das Dashboard für Zoll und Spediteur ist bewusst minimal gehalten. Dies aus dem Grund, dass diese Nutzergruppe keine Übersicht über Bestellungen braucht. Diese Nutzer melden sich im Zweifelsfall sowieso nur an, um eine gescannte Bestellung zu verarbeiten, und sehen dieses Dashbord gar nicht zwingend. Wenn sie sich aber regulär anmelden, werden sie auf dieses Dashboard weitergeleitet.

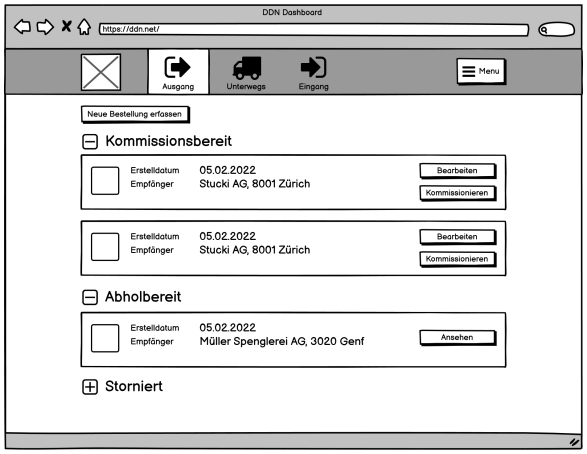


Abbildung 3.4: Das Dashboard

Abbildung 3.5: second figure

Ausgang	Unterwegs	Eingang
Kommissionsbereit: [Erstellt]	Eingehende Lieferungen: [Abgeholt]	Geliefert: [Geliefert]
Abholbereit: [Kommissioniert]	Ausgehende Lieferungen: [Abgeholt, Geliefert, Angekommen, Abgeschlossen]	Bereit für Wareneingang: [Angekommen]
		Abgeschlossen: [Abgeschlossen]

Tabelle 3.3: Angezeigte Status in den Bereichen des Dashboards (der Status ist in eckigen Klammern)

Detailansicht

Die Detailansicht dient dazu, eine Bestellung anzusehen oder weiterzuverarbeiten. Die wichtigsten drei Abschnitte sind die Lieferdaten, die Warenliste und der Tracking-Verlauf. Da die Detailansicht verschiedenen Zwecken dient, gibt es verschiedene Variationen davon, unter anderem mit weiteren Komponenten, die Aktionen auf der Lieferung zulassen. Die Abbildung 3.6 zeigt anhand Wireframes, wie die Darstellung angedacht ist. Die möglichen Komponenten einer Detailansicht sind im Abschnitt Varianten der Detailansicht erläutert.

Varianten der Detailansicht

Neue Bestellung Selbstsprechend kann hier die Bestellung erfasst werden.

Detailansicht Die Detailansicht zeigt die Lieferdaten, die Warenliste und den Statuslog.

Bestellung bearbeiten Diese Ansicht ist dazu da, die Warenliste als auch den Spediteur anzupassen, oder gegebenenfalls auch die Bestellung zu stornieren.

Kommissionierung Bei der Kommissionierung wird die Warenliste als Checkliste abgearbeitet, um sicherzustellen, dass die Lieferung komplett ist.

Ankunft Hier wird die Ankunft der Lieferung bestätigt.

Wareneingang Beim Wareneingang wird, ähnlich zur Kommissionierung, die Warenliste als Checkliste abgearbeitet, ermöglicht aber daneben auch die Erfassung von Problemen.

Update Diese Ansicht dient der Statuserfassung durch den Spediteur. Er kann hier die nötigen Daten für die Sendungsverfolgung der Lieferung erfassen.

Zoll Der Zoll sieht eine simple Warenliste und kann die Lieferung als verzollt markieren.

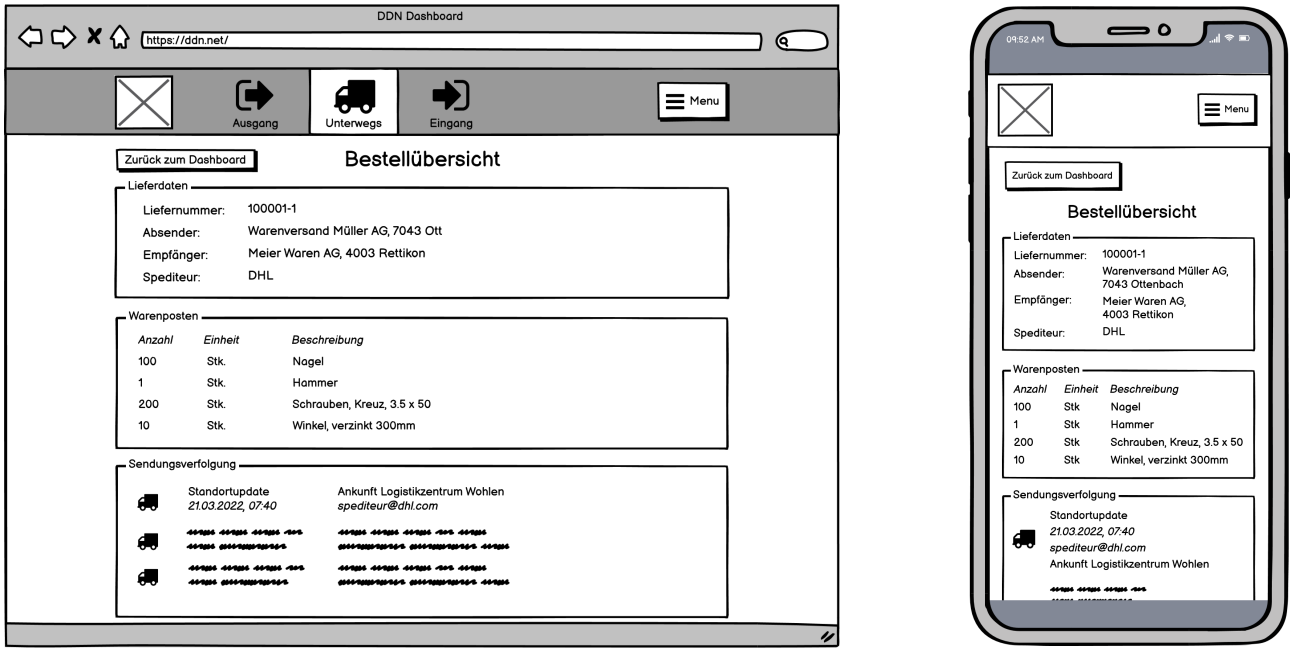


Abbildung 3.6: Das Wireframe der Bestellübersicht in der Desktop und mobilen Version ist der Ausgangspunkt für alle Varianten der Detailansicht.

QR-Code

Für den QR-Code wird eine eigene Identifikationsnummer verwendet, damit die Sendungsnummer nur berechtigten Personen bekannt ist. Anhand dieser Nummer wird eine URL und daraus ein QR-Code generiert, damit Nutzer beim Scan direkt zur richtigen Stelle weitergeleitet werden. Beim Einstieg zu einem QR-Code wird zuerst geprüft, ob der Nutzer angemeldet ist. Falls

nicht, erscheint ein Login-Formular. Der Nutzer wird dann entsprechend seiner Rolle in der Lieferung weitergeleitet oder blockiert, wie dies auf Abbildung 3.7 sichtbar ist. Für die Unterscheidung nach Status zur Weiterleitung wird die Tabelle 3.2 verwendet. Spediteur und Zoll können die entsprechenden Ansichten nur mit Status *Abgeholt* abrufen.

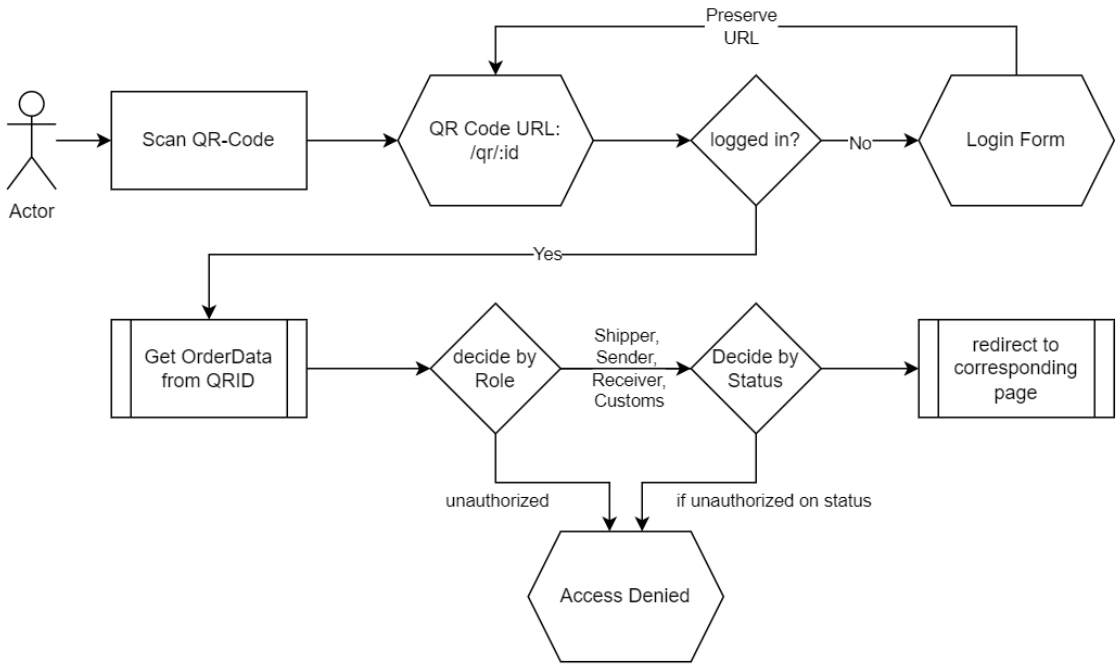


Abbildung 3.7: Dieses Diagramm zeigt, wie ein Aufruf der QR-Code URL bearbeitet wird.

Das Zusammenspiel

All die bisher vorgestellten Bereiche und Ansichten sind interaktiv miteinander verbunden. Die Abbildungen 3.8 und 3.9 zeigen anhand von User Flows die genauen Abläufe.

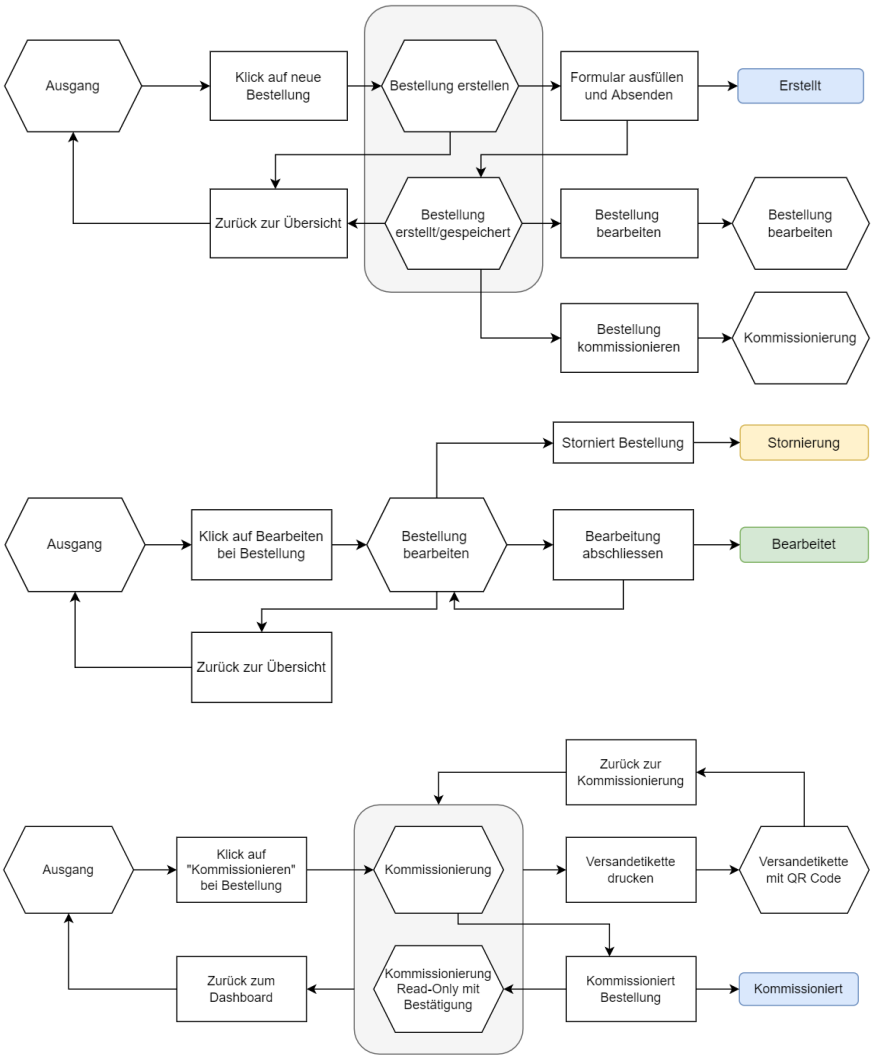


Abbildung 3.8: Die dargestellen User Flows zeigen die Benutzerführung beim Warenausgang.

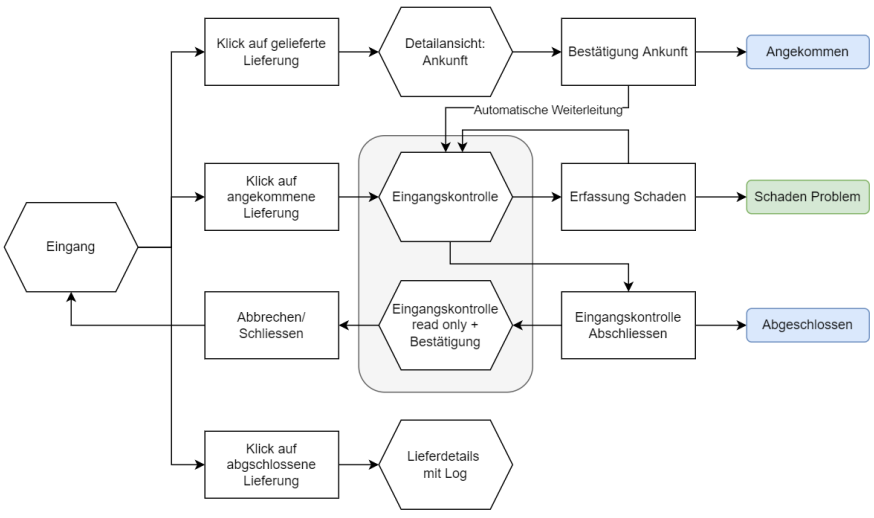


Abbildung 3.9: Dieser User Flow zeigt die Benutzerführung für den Wareneingang.

3.4 Datenmodell

In der Design-Phase stellte sich heraus, dass für das Data Model gewisse Beziehungen nicht funktionieren. Damit mehrere Benutzer für eine Organisation Bestellungen verwalten können, mussten diese direkt mit der Organisation verknüpft werden, statt mit einem Benutzer. Als Absender, Spediteur sowie Empfänger einer Bestellung ist somit neu die Organisation verknüpft, nicht mehr der Benutzer.
Weiterhin ist die Verknüpfung der Akteure nicht mehr mit der Lieferung, sondern mit der Bestellung. Dies entstand aus dem Entscheid, dass allfällige Teil-Lieferungen vorerst keinen unterschiedlichen Spediteur oder Empfänger haben können.

Nebst den für die Datenerfassung wichtigen Entitäten wie Land, Einheit und Status ist die Adresse hinzugekommen. Diese wurde als separate Entität identifiziert, da eine Organisation grundsätzlich mehrere Adressen für Versand wie auch Empfang haben kann.

Damit ein Auditing möglich ist, wurden bei allen Entitäten, welche vom Benutzer erfasst oder modifiziert werden können, zusätzliche Properties wie »createdByUser«, »createdOn«, »lastUpdatedByUser« sowie »lastUpdatedOn« hinzugefügt.

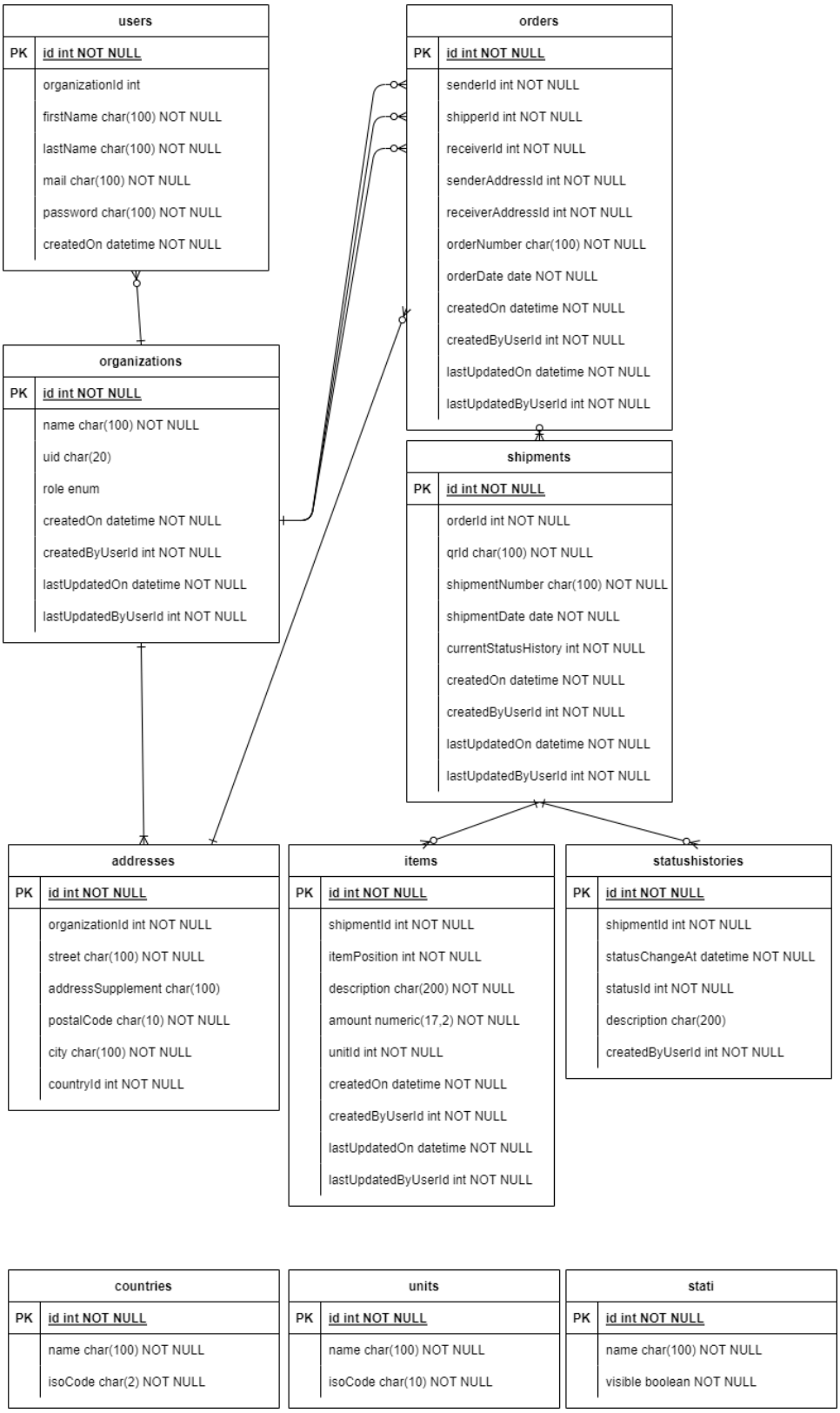


Abbildung 3.10: Das entworfene Data Model.

4 Umsetzung

4.1 Technologien

Hosting

Es war vorgegeben, dass die Applikation auf der Cloud Hosting Plattform »DigitalOcean« deployed wird. Dafür haben wir von unserem Partner AdaptIT einen DigitalOcean Account zur Verfügung gestellt bekommen.

Auf DigitalOcean kann unter anderem per Mausklick ein sogenanntes Droplet erstellt werden, auf dem dann beispielsweise die Applikation deployed werden kann. Droplets sind Linux-basierte Virtual Machines (VM).

Für dieses Projekt wurden jedoch keine Droplets verwendet, sondern die neuere App-Plattform. DigitalOcean bietet mit der App-Plattform vorgefertigte VMs, welche für die gängigen Apps schon das Nötigste installiert haben. So kann bei der Erstellung der App ein GitHub oder GitLab Repository angegeben werden, von welchem der Code automatisch gepullt und deployed wird.

Darauf wird im Abschnitt 4.9 näher eingegangen.

Front- und Backend

Die Aufgabenstellung gab vor, dass folgende Web-Technologien verwendet werden:

JavaScript

Node.js

Für die Umsetzung des Web-Frontends war eines der folgenden drei Technologien zu verwenden:

React

Angular

Vue

Entscheidung Frontend-Technologie

Es stellte sich schnell heraus, dass React für dieses Projekt die passendste Technologie war. Bisher hatte nur Mathias Erfahrung mit einem der Technologien, nämlich mit React.
React hat zudem die tiefste Lernkurve und aus diesem Grund wurde entschieden, dass das Frontend mit React umgesetzt werden würde.

Datenbank

DigitalOcean bietet die Möglichkeit, einen Datenbank-Cluster zu erstellen. Zu den möglichen Datenbank-Engines für einen Datenbank-Cluster gehören MongoDB, PostgreSQL, MySQL sowie Redis.

Da wir beide beruflich sowie privatlich schon viel mit PostgreSQL gearbeitet haben, ist die Entscheidung einfach gefallen. PostgreSQL ist zudem open-source und gehört zu den beliebtesten Datenbank-Engines.

Für die Datenbank wurde entsprechend ein Datenbank-Cluster erstellt.

4.2 Gesamtsystem Architektur

Die Applikation hat eine 3-Tier Architektur bestehend aus dem React Web-Frontend, dem Node.js Express Backend sowie der PostgreSQL Datenbank.

4.3 Umsetzung Backend

Das Backend besteht aus einem Node.js Express Server. Um das Datenmodell umzusetzen, wurde Prisma [2] als Object-relational Mapping (ORM) Tool verwendet. Mithilfe des Prisma Clients liessen sich so die CRUD-Operationen im Backend relativ einfach umsetzen.

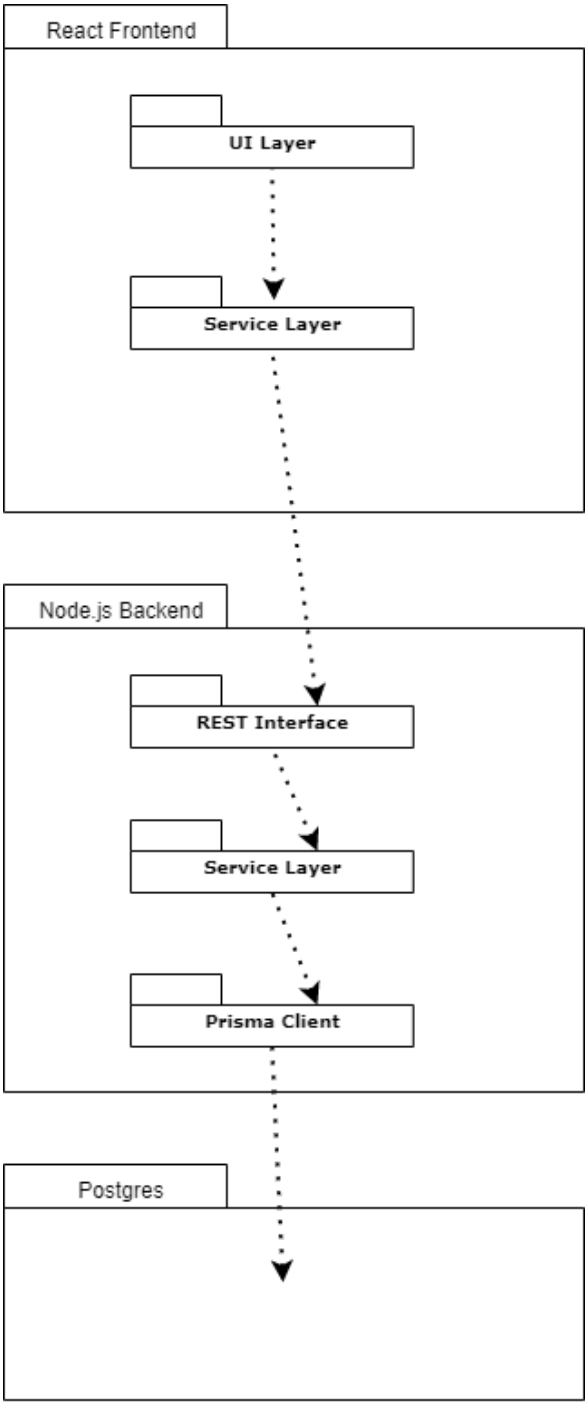


Abbildung 4.1: Das Architekturdiagramm mit 3 Tiers.

Setup Prisma

Um Prisma aufzusetzen, musste es zuerst als Development Dependency zum Server-Projekt hinzugefügt werden.

npm install prisma –save-dev

Prisma kann automatisch aus einem Schema-File die benötigten Datenbank-Skripte generieren und auch auf der Ziel-Datenbank ausführen.
Um dieses Schema-File zu initialisieren, musste folgender Befehl ausgeführt werden.

npx prisma init

Dieser Befehl erstellt einen neuen Ordner *prisma* mit dem Schema-File *schema.prisma*. Zusätzlich wird im Root-Ordner des Projekts ein .env-File erstellt, um die Umgebungsvariablen für die Datenbank-Connection zu definieren.

Sobald Prisma aufgesetzt war, konnte das Datenmodell im Schema-File definiert werden. Jedes definierte Model im Schema korreliert später mit einer Datenbanktabelle.

```
model Organization {
  id                  Int              @id @default(autoincrement())
  name                String           @db.VarChar(100)
  uid                 String?          @db.VarChar(20)
  registrationcode    String           @unique @db.VarChar(36)
  role                OrganizationRole @default(COMPANY)
  createdon           DateTime?
  createdbyuserid     Int?
  lastupdatedon       DateTime?
  lastupdatedbyuserid Int?

  users           User[]
  sendingorders   Order[]   @relation(name: "sender")
  shippingorders  Order[]   @relation(name: "shipper")
  receivingorders Order[]   @relation(name: "receiver")
  addresses       Address[]

  createdbyuser     User? @relation(name: "createdbyuserorganizations", fields: [createdbyuserid], references: [id])
  lastupdatedbyuser User? @relation(name: "lastupdatedbyuserorganizations", fields: [lastupdatedbyuserid], references: [id])
  @@map("organizations")
}
```

Abbildung 4.2: Beispiel des Organization Models im Prisma Schema.

Wurde dies getan und wurde die Datenbank-Connection entsprechend im .env-File konfiguriert, konnte mit folgendem Befehl die Datenbank automatisch von Prisma erstellt werden.

npx prisma migrate dev

SSL Kommunikation mit Backend

Mit Verwendung der neuen App-Plattform von DigitalOcean [4] ist es automatisch so aufgesetzt, so dass jede Kommunikation von und zum Backend mit SSL verschlüsselt wird. Hier musste also nichts mehr unternommen werden.

4.4 Umsetzung Frontend

Das Frontend wurde mit React in Typescript entwickelt. Die Umsetzung hält sich an die Resultate der Analyse und die Konzepte aus dem Kapitel 3. Für diesen Abschnitt wesentliche Kenntnisse wurden im Modul WE3 [6] erarbeitet.

Die Codebasis ist auf 6 Ordner aufgeteilt, damit es übersichtlich bleibt. In Abbildung 4.3 ist ersichtlich, dass die Datei App.tsx der Einstiegspunkt ist für die Applikation. Im Ordner *pages* sind alle Seiten abgelegt, darunter das Dashboard, die Anmeldung und weitere. Im Ordner *shipment* sind die Seiten für die Detailansicht abgelegt und im Ordner *shipmentComponents* die einzelnen Bestandteile für die Detailansicht. *components* sind React-Komponenten mit verschiedener Funktion, sei dies für Darstellung von Lieferungen im Dashboard oder spezielle Formularfelder. *utils* beherbergt einige Skripts für spezielle Anwendungen und unter *services* sind die Schnittstellen zum Server zu finden.

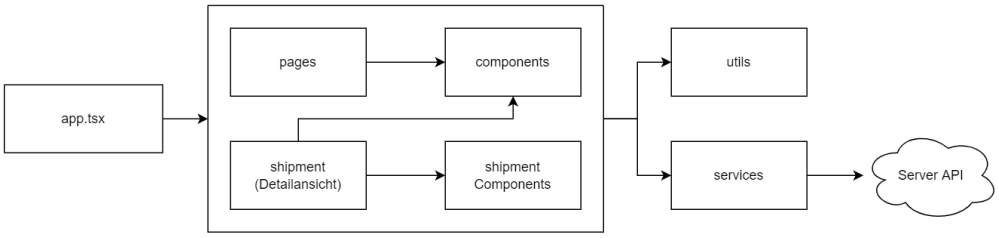


Abbildung 4.3: Programmstruktur im Frontend

Für die Gestaltung des Frontend wurde der gesamte HTML und CSS Code selber entwickelt. Daneben wurde auch eigens ein Logo und Icons für die Bereiche des Dashboards und die Sendungsverfolgung erstellt.

Für die Navigation auf dem Frontend wurde ein Routing eingerichtet, wie es in Abbildung 4.4 sichtbar ist. Dies unterstützt die Navigation des Nutzers und programmintern ermöglicht es, Weiterleitungen zu verwenden.

Bezüglich Libraries wurden mehrheitlich Standard-Libraries verwendet. Zu den nennenswerten gehören einerseits *react-icons*, das für kleine Icons verwendet wurde, wie beispielsweise die Mülltonne als Löschzeichen. Andererseits wurde *react-qr-code* verwendet, um den QR-Code zu generieren.

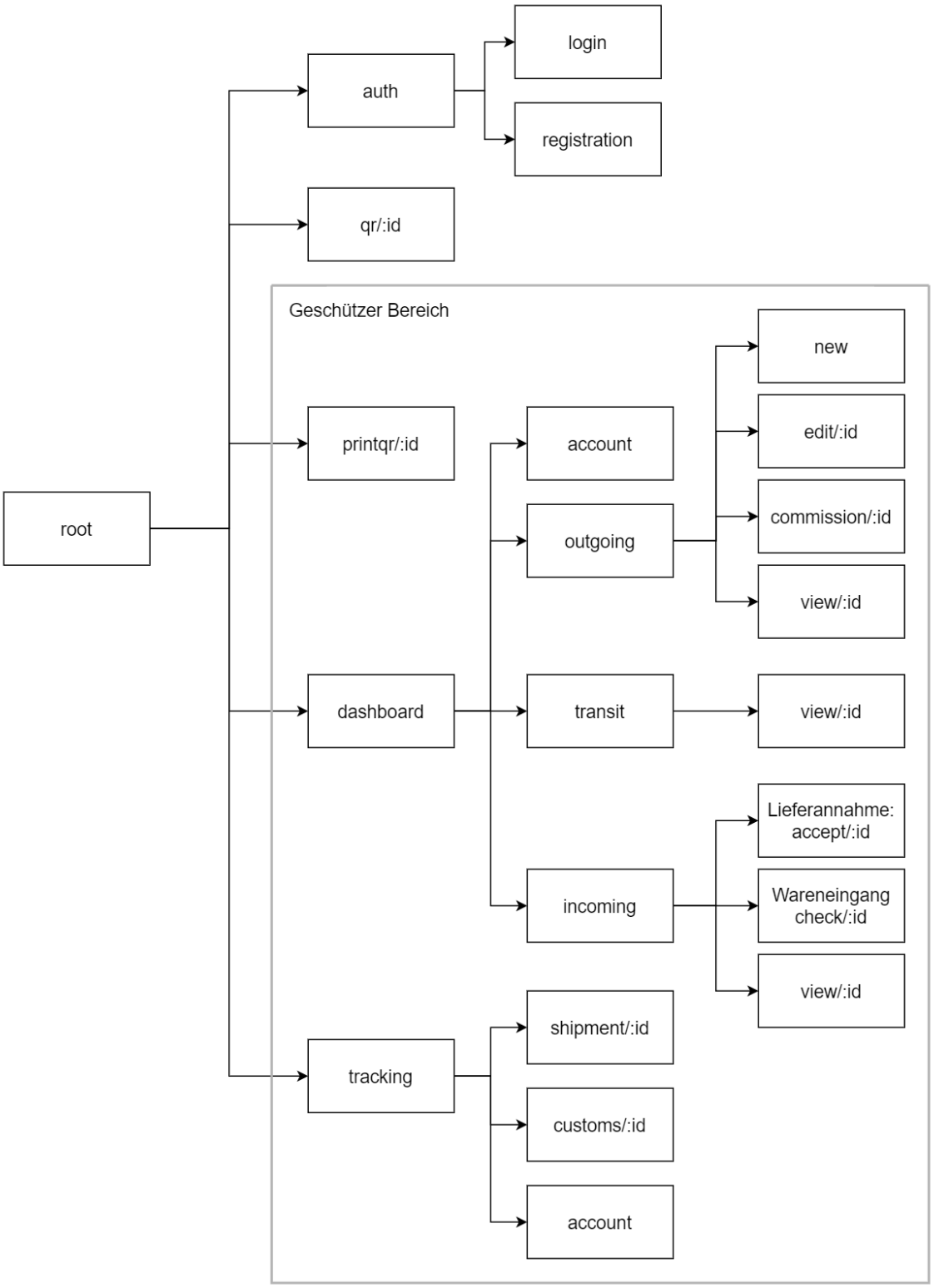


Abbildung 4.4: Routing im Frontend

4.5 Logging

Damit jegliche Zugriffe auf die spätere REST API des Backends und auch Errors im Backend nicht unbemerkt bleiben, wurde ein Logging mit Winston implementiert.
Im Backend konnte dazu ExpressWinston als Middleware verwendet werden.

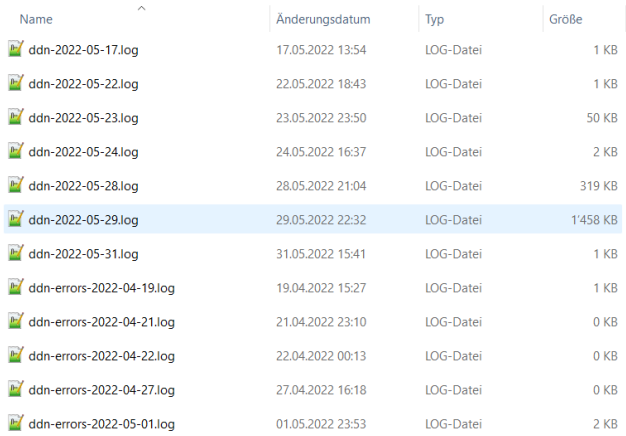
```
app.use(
  expressWinston.logger({
    transports: [
      new winston.transports.Console({
        format: winston.format.combine(      You, last month • added
          winston.format.colorize(),
          winston.format.json(),
          winston.format.timestamp(),
          winston.format.printf(({ timestamp, level, message }) => {
            return `[${timestamp}] ${level}: ${message}`;
          })
        ),
      }),
      new DailyRotateFile({
        level: "info",
        dirname: "logs",
        filename: "ddn-%DATE%.log",
        format: winston.format.combine(winston.format.json()),
      }),
      new DailyRotateFile({
        level: "error",
        dirname: "logs",
        filename: "ddn-errors-%DATE%.log",
        format: winston.format.combine(winston.format.json()),
      }),
    ],
    format: winston.format.combine(
      winston.format.timestamp(),
      winston.format.metadata()
    ),
    meta: false,
    msg: "HTTP {{req.method}} {{req.url}}",
    expressFormat: true,
    colorize: false,
    ignoreRoute: function (req, res) {
      return false;
    },
  })
);
```

Abbildung 4.5: Winston Logging Konfiguration

Um das Durchforsten der Log-Files zu vereinfachen, wurde es so konfiguriert, dass täglich ein neues Log-File mit dem entsprechenden Datum erstellt wird.

Zusätzlich werden die Log-Einträge im JSON-Format erstellt, wodurch es künftig in ein Log-Analyse Tool eingespiesen werden kann.

Weiterhin ist es möglich, mithilfe der im Backend- sowie im Frontend implementierten Util-Klasse »logger.ts« entsprechende Log-Einträge für relevante Events wie beispielsweise dem Login oder Errors zu generieren.



Name	Änderungsdatum	Typ	Größe
ddn-2022-05-17.log	17.05.2022 13:54	LOG-Datei	1 KB
ddn-2022-05-22.log	22.05.2022 18:43	LOG-Datei	1 KB
ddn-2022-05-23.log	23.05.2022 23:50	LOG-Datei	50 KB
ddn-2022-05-24.log	24.05.2022 16:37	LOG-Datei	2 KB
ddn-2022-05-28.log	28.05.2022 21:04	LOG-Datei	319 KB
ddn-2022-05-29.log	29.05.2022 22:32	LOG-Datei	1'458 KB
ddn-2022-05-31.log	31.05.2022 15:41	LOG-Datei	1 KB
ddn-errors-2022-04-19.log	19.04.2022 15:27	LOG-Datei	1 KB
ddn-errors-2022-04-21.log	21.04.2022 23:10	LOG-Datei	0 KB
ddn-errors-2022-04-22.log	22.04.2022 00:13	LOG-Datei	0 KB
ddn-errors-2022-04-27.log	27.04.2022 16:18	LOG-Datei	0 KB
ddn-errors-2022-05-01.log	01.05.2022 23:53	LOG-Datei	2 KB

Abbildung 4.6: Täglich wechselnde Log-Files

4.6 Error-Handling Backend

Für das Error-Handling im Backend wurden eigene Error-Klassen definiert. Diese erweitern den Standard-Error um den Parameter **status** und auch den originalen **cause** des Errors.

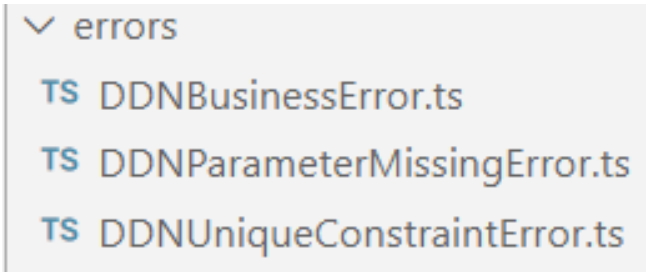


Abbildung 4.7: Custom Error-Klassen des Backends

```
export class DDNBusinessError extends Error {        You, 4 days ago • add
    status: number;

    constructor(message: string, status: number, cause: any) {
        super(cause != null ? message + "\n" + cause.message : message);
        this.name = this.constructor.name;
        this.status = status;
    }
}
```

Abbildung 4.8: Code einer Custom Error-Klasse

Dieser **status** dient dazu, der Error-Handling Middleware im Backend einen HTTP Status-Code zu übermitteln, der so als Response an das Frontend zurückgeschickt werden kann im Error-Fall. Wird kein **status** angegeben, schickt das Backend im Error-Fall den Standard HTTP Status-Code 500 zurück.

Dem optionalen **cause** wird die originale Error-Message entnommen und an unsere Custom Error-Message angehängt. Dies wird dann entsprechend als Error geloggt und auch als Message an das Frontend übermittelt.

```
// Handle errors.
app.use(function (err: any, req: any, res: any, next: Function) {
  logger.error(`${err.message}`);
  res.status(err.status || 500);
  res.json({ error: err.message });
});
```

Abbildung 4.9: Error Middleware im Backend

Jeder API Endpoint, der Parameter entgegen nimmt, überprüft die benötigten Parameter und wirft eine **ParameterMissingError**, wenn diese fehlen.

Weiterhin gibt es den **UniqueConstraintError**, der geworfen wird, wenn ein Unique Index verletzt wird, beispielsweise durch Registrieren eines Benutzers mit bereits existierender Mail-Adresse.

Alle Errors, die nicht weiter spezifiziert werden können, werden vom Backend als Business Errors behandelt.

4.7 API

Im Backend ist eine REST API implementiert worden, die es dem Frontend erlaubt, Create-, Read- und Updateoperationen auf den Daten durchzuführen. Delete-Operationen werden aktuell noch nicht unterstützt, da noch keine Funktionalität vorgesehen ist, die Daten löschen soll.

Alle vorhandenen API Endpoints sind im Anhang unter Abschnitt **??** dokumentiert und werden hier nur einzeln weiter behandelt.

Die API Endpoints sind unterteilt in **Secure** sowie **Non-secure** Routes.

Authentifizierung

Für die Secure Routes wurde ein Authentifizierungsverfahren mit dem Modul **Passport** umgesetzt. Hierbei authentifizieren sich die Benutzer zuerst über den non-secure **/login** Endpoint mit ihrem Login (Mailadresse) sowie ihr Passwort. Können die Login-Credentials erfolgreich validiert werden, erhält der Aufrufer ein sogenanntes JSON Web Token (JWT) zurück. Die Verwendung von JWT wurde in der Vorlesung WED2 [8] empfohlen.

Um die Secure Routes aufzurufen, muss jeweils dieses JWT wieder mitgeschickt werden, welches von Passport validiert wird. Ist die Validation erfolgreich, wird der Request entsprechend vom Backend verarbeitet.

Das JWT hat in einer ersten Version eine Ablaufzeit von einer Stunde. Somit bleibt ein Benutzer während einer Stunde eingeloggt und kann ohne nochmaligem Login weiterarbeiten.

Silent-Refresh Mechanismus

Es war angedacht, einen zusätzlichen Silent-Refresh Mechanismus einzubauen über den Endpoint **/refreshtoken**. Bei jedem Login wird neben dem normalen JSON Web Token auch ein Refresh-Token generiert. Dieses hat eine längere Ablaufzeit und wird entsprechend als signed Cookie beim Benutzer gespeichert und auch in der Datenbank bei jedem Login.

So würde in einem bestimmten Intervall, beispielsweise alle fünf Minuten, dieser **/refreshtoken** Endpoint aufgerufen werden. Diesem müsste der aktuelle Refresh-Token aus dem Cookie, sofern vorhanden, mitgeschickt werden. Dieses wird von Passport ähnlich wie dem JWT signiert und dann mit dem Refresh-Token aus der Datenbank verglichen. Der Vergleich mit dem Refresh-Token aus der Datenbank dient als zusätzliche Absicherung im Falle, dass jemand die Cookies des Benutzers auslesen kann. Stimmen diese überein, wird dem Benutzer ein neues JWT zugeschickt.

Auf diese Weise würde ein Benutzer bei aktiver Arbeit nicht ausgeloggt werden. Aus zeitlichen Gründen konnte bisher jedoch nur der Endpoint umgesetzt werden, jedoch noch nicht das ständige Aufrufen vom Frontend aus.

Non-secure Routes

Nicht alle Routes benötigen eine Authentifizierung. Damit sich neue Benutzer registrieren können, gibt es auch die non-secure Routes **/signup** sowie **/countries**.
Letzterer dient dem Frontend dazu, um eine Liste von Ländern für die Adresserfassung abzufragen.

Passwort-Handling

Bei der Registration wird das Passwort des Benutzers jeweils über zehn Runden mit einem generierten Salt gehashed und anschliessend in der Datenbank gespeichert.
Zusätzlich ist im Frontend sowie Backend ein Check implementiert, der die Länge des Passworts auf mindestens 12 Zeichen prüft.

Generierung Bestellungsnummer

Die Bestellungs- sowie Lieferungsnummer haben einen Unique Index gesetzt, da es im System keine Dupletten geben soll. Beim Hinzufügen einer Bestellung wird eine Bestellungsnummer, beginnend mit 100'000, generiert, worauf die bestehende Anzahl Bestellungen plus eins addiert wird.

Beispiel: 100001

Dies ist nicht die optimalste Lösung, da es zu einer Racing Condition kommen könnte, wenn mehrere Bestellungen gleichzeitig hinzugefügt werden. Problematisch ist dies insofern auch, sobald Bestellungen gelöscht werden können, denn dann könnte eine bereits bestehende Bestellungsnummer generiert werden, da durch das Löschen der Count der Bestellungen reduziert wird.

Die bessere Lösung wäre hier das Verwenden einer Datenbank-Sequenz gewesen. So würde die Datenbank beim Hinzufügen eine Zahl aus einem vorgenerierten Pool in der Sequenz zurückgeben, welche für die Bestellungsnummer verwendet werden kann. Leider unterstützt Prisma das Verwenden von Datenbank-Sequenzen nur für CockroachDB [1].

Da in dieser ersten Version noch keine Löschfunktionalität vorgesehen ist und auch noch wenige Benutzer gleichzeitig arbeiten werden, wird diese Lösung vorerst so belassen.

Generierung Lieferungsnummer

Die Lieferungsnummer basiert auf der Bestellungsnummer der dazugehörigen Lieferung. Darauf basierend wird das Trennzeichen »-« verkettet und anschliessend die Nummer der Lieferung innerhalb der Bestellung. Ist dies die erste Lieferung der Bestellung, ergibt sich beispielsweise folgende Nummer:

Beispiel: 100001-1

Ursprünglich wurde als Trennzeichen ein »#« verwendet.
Da beim Endpoint **/api/shipments/shipmentNumber** jedoch die Lieferungsnummer als Parameter übergeben wurde, kam es zu einem Parse-Problem im Backend, wobei die Nummer nur bis zum Trennzeichen gelesen wurde.
Aus diesem Grund wurde entschieden, das Trennzeichen auf das »-« anzupassen.

Generierung QR-ID

Die QR-ID, auf welchem der QR-Code basiert, wird beim Hinzufügen der Lieferung automatisch generiert. Diese wird über die Funktion **randomUUID()** des Moduls **crypto** generiert, damit

sie eindeutig ist. Es wurde davon abgesehen, die Datenbank-ID der Lieferung zu verwenden, da der Benutzer diesen so im Link des QR-Codes sehen würde.

```
shipmentToInsert.qrid = crypto.randomUUID();
```

Abbildung 4.10: Generierung der QR-ID

4.8 CI / CD

Als CI/CD Tool wurde GitLab gewählt. Für das Backend sowie das Frontend wurde jeweils ein GitLab Repository erstellt. Dabei wurde jeweils auf dem **develop** Branch entwickelt.

Auf GitLab wurde zusätzlich eine CI/CD Pipeline eingerichtet, der den Code der Projekte nach jedem Commit automatisch mit ESLint sowie Prettier überprüft. So schlägt dieser fehl, wenn der eingecheckte Code nicht die Richtlinien von ESLint erfüllt oder nicht mit Prettier formatiert wurde.

Auf ein Continuous Deployment wurde hier verzichtet, da dies eine weitere Umgebung eigens dafür bräuchte. Im Rahmen dieses kleinen Projektes und auch um unserem Partner AdaptIT unnötige Kosten auf DigitalOcean zu verursachen, haben wir es dabei belassen.

4.9 Deployment

Wie bereits im Abschnitt 4.1 erwähnt, wurde die Applikation auf DigitalOcean deployed. Dabei wurden das Backend sowie das Frontend auf separaten Apps deployed. Das Backend hat zusätzlich eine Anbindung zum PostgreSQL Datenbank-Cluster. Für das Frontend wurde uns die Domäne **ddn.adaptit.ch** von unserem Partner AdaptIt zur Verfügung gestellt. Dies konnte auf DigitalOcean so hinzugefügt werden.

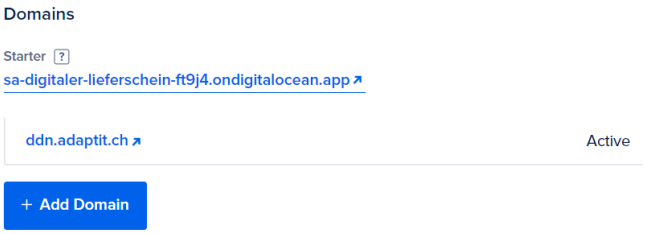


Abbildung 4.11: Hinzufügen einer Domäne auf DigitalOcean

Beim Aufsetzen der jeweiligen Apps wurde entsprechend das dazugehörige GitLab Repository verlinkt. Dies ermöglicht es DigitalOcean, per Mausklick den aktuellen Code des spezifizierten Branches im Repository zu pullen und zu deployen.

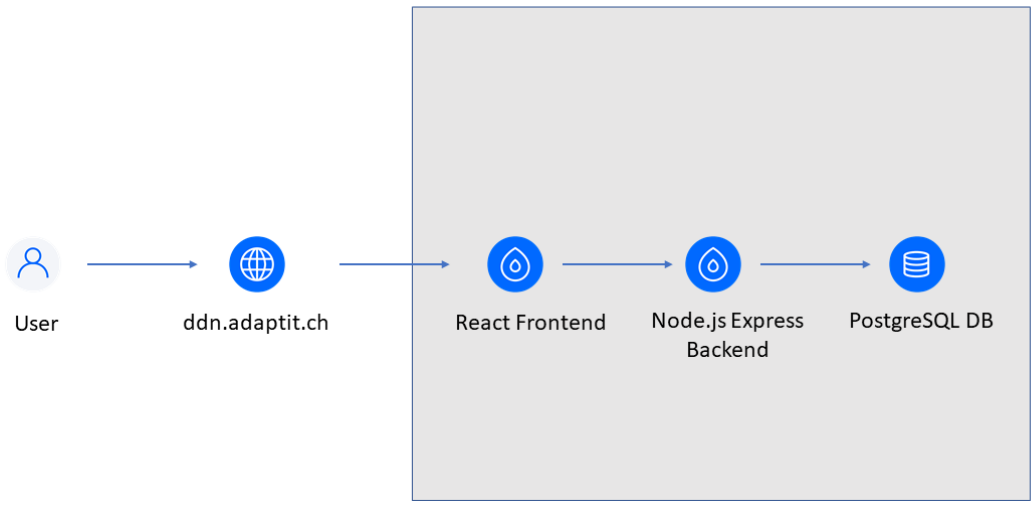


Abbildung 4.12: DigitalOcean Deployment-Diagramm der Applikation

Während der Entwicklung war hier jeweils der **develop**-Branch verlinkt und es wurde jeweils von Hand ein Neudeployment angestossen, sobald wichtige Features implementiert waren.

DigitalOcean bietet hierbei auch die Option, ein automatisches Neudeployment bei jedem Commit anzustossen. Dies wurde anfangs so belassen, verursachte jedoch mit der Zeit nur Probleme, da beispielsweise einerseits das Backend auf einem neuen Stand war, das Frontend jedoch die nötigen Anpassungen noch nicht beinhaltete, um zu funktionieren.

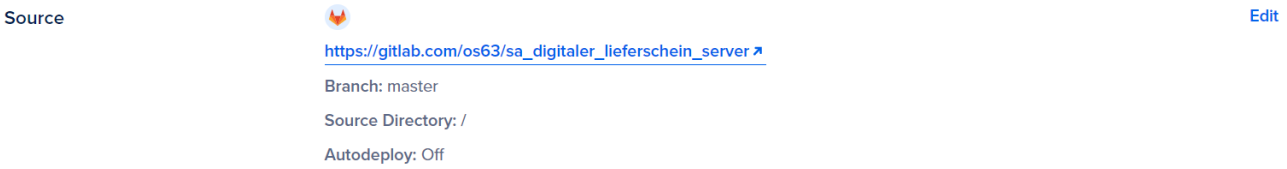


Abbildung 4.13: Konfiguration Repository auf DigitalOcean App

Beim Abschluss-Deployment wurde jeweils der aktuelle Code-Stand des **develop**-Branch in den **master**-Branch gemerged und anschliessend ein Tag **ddn-1.0.0** erstellt. Danach wurden die Apps neu jeweils auf den **master**-Branch verlinkt und neu deployed.

CORS Policy Backend

Da das Frontend die API Calls von einem anderen Origin aus auf das Backend macht, benötigte es die Konfiguration einer Cross-Origin Resource Sharing (CORS) Policy auf dem Backend, damit nicht beliebige Origins die Ressourcen des Backends laden konnten.

DigitalOcean bot hier die Möglichkeit, die CORS Policy direkt auf der App einzurichten. Folgende CORS Policy wurde entsprechend auf der App des Backends eingerichtet:

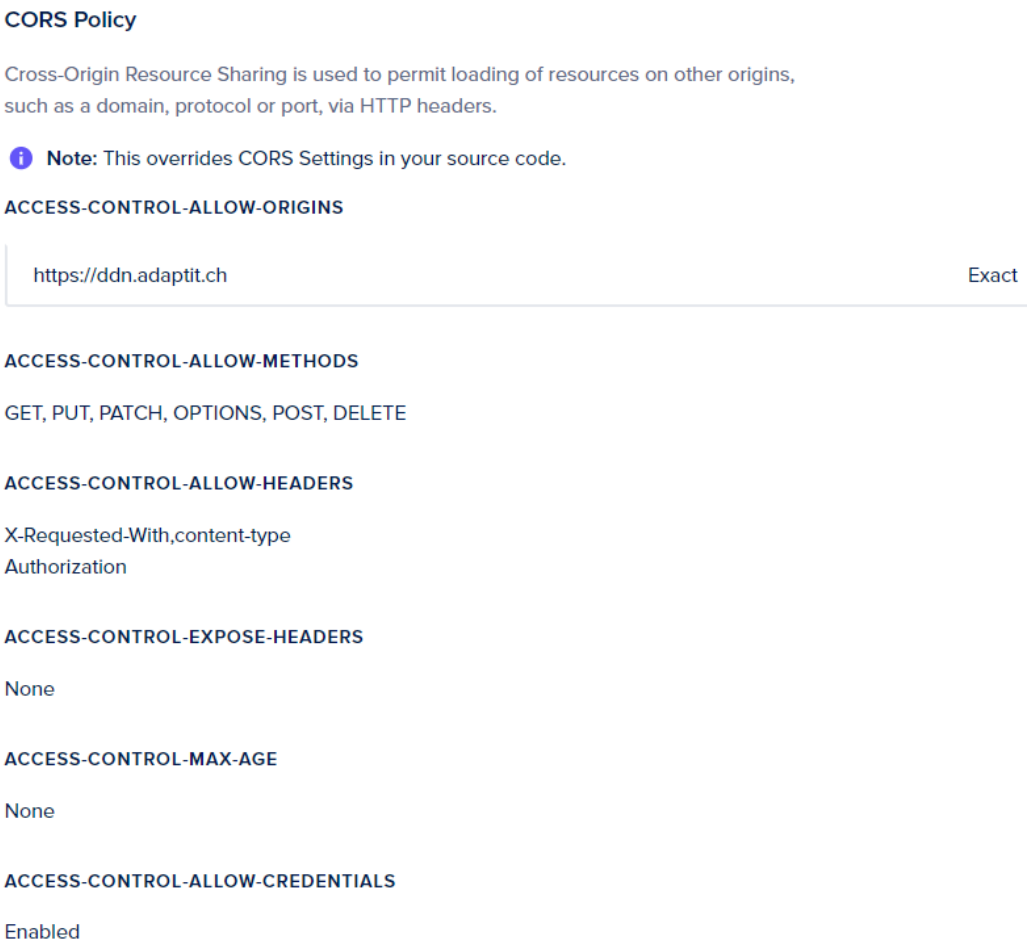


Abbildung 4.14: Konfiguration CORS Policy auf DigitalOcean App

Anpassen Run Command

Während des Deployments auf DigitalOcean kam es einige Male zu Problemen, wodurch gewisse Dependencies nicht korrekt geladen wurden. Dies hatte damit zu tun, dass die Environment-Variable **NODE_ENV** auf **production** gesetzt wurde für die Produktion. Sobald diese gesetzt war, lud DigitalOcean gewisse Dependencies nicht mehr korrekt.

Aus diesem Grund musste der Run Command des Backends auf DigitalOcean wie folgt angepasst werden, so dass diese Environment-Variable beim Starten gesetzt wird.

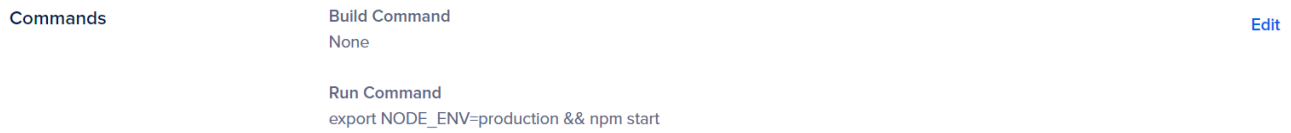


Abbildung 4.15: Konfiguration Run Command auf DigitalOcean App

App-Level Variabeln Backend

Auf DigitalOcean wurden folgende App-Level Variabeln auf dem Backend konfiguriert. Dies sind die Ablaufzeiten für den JWT Token, Refresh-Token sowie die benötigten Secrets für deren Signierung. Die benötigten Secrets wurden durch eine Option auf DigtalOcean verschlüsselt, damit sie nicht einfach so eingesehen werden können.

Abbildung 4.16: Konfiguration der App-Level Variabeln auf DigitalOcean App

5 Qualitätssicherung

Testing Backend

Im Rahmen der Entwicklung wurden die Test-Cases fortlaufend erweitert. Da das Backend hauptsächlich aus der REST API besteht, welche nur im Rahmen von effektiven API Calls sinnvoll getestet werden können, wurden hier nur API Tests implementiert.

Diese wurden basierend auf Mocha, Chai und Supertest umgesetzt. Dabei diente Supertest dem Absetzen der Requests auf die Testumgebung. Chai ist die Assertion Library und Mocha diente als Test-Framework um die Tests durchzuführen.

Die API Tests des Backends wurden nicht automatisiert, da sie eine laufende Umgebung voraussetzen. Hierbei wurde jeweils der Localhost als laufende Instanz vorausgesetzt, um die Tests jeweils manuell laufen zu lassen. Es wäre eine Möglichkeit gewesen, eigens eine Testumgebung auf DigitalOcean aufzusetzen, auf welcher die Tests automatisiert ablaufen. Im kleinen Rahmen des Projekts wurde bewusst darauf verzichtet, da eine weitere Umgebung zusätzliche Kosten verursacht hätten.

Zusätzlich wurde die REST API lokal jeweils auch mit Postman getestet.

System Testing

Zwischendurch wurden System Tests durchgeführt, worin alle Funktionalitäten ausgehend von den Use Cases und den funktionalen Anforderungen getestet wurden. Mit dem abschliessenden Systemtest konnten alle Testfälle erfüllt werden. Die Spezifikation und die Protokolle sind im Anhang unter Abschnitt **??** zu finden. Für diesen Abschnitt wesentliche Kenntnisse wurden im Modul SE2 [5] erarbeitet.

5.1 Usability Testing

Die Bedienbarkeit der Applikation ist sehr wichtig, weil sie schnell und intuitiv bedienbar sein soll. Dies hat den Vorteil, das Mitarbeiter schnell eingearbeitet werden können und in einer

Demonstration schnell begriffen werden kann, wie was funktioniert. Für die Usability Tests wurde das Wissen aus der Vorlesung TODO eingesetzt.

Dafür wurden Usability Tests mit vier Personen durchgeführt. Dadurch, dass die Entwicklung langsamer vonstatten ging als erwartet, konnten die Tests erst spät im Projekt durchgeführt werden. Dadurch war das Ziel nicht, danach noch grossartige Änderungen zu vollziehen, sondern in erster Linie die gemachte Arbeit zu prüfen, kleine Verbesserungen zu machen und Ideen für Empfehlungen zu sammeln, die für die weitere Entwicklung eingesetzt werden könnten.

Im Rahmen der Usability Test stellten wir einige kleine Probleme mit der der Benutzeroberfläche fest, als auch Themen, die eher den Kontext dieser Arbeit sprengen würden. Unter den kleineren Problemen waren unklare Benennung von einigen Buttons und andererseits der Bestätigungsprozess der Kommissionierung. Da die kleineren Probleme bereits bei den ersten drei Testpersonen autftraten, wurden diese für den vierten Test ausgebessert. Der Prozess war dann für die Testperson spürbar verständlicher.

Die grösseren Themen waren Integration und Automation. Diese sind für das Projetk natürlich out of Scope, aber für den Anwender trotzdem wichtig. Autovervollständigung bei den Waren oder sogar Anbindung an SAP sind für den Arbeitsalltag unentbehrlich, sobald ein gewisses Volumen erreicht wird. Auch für den Spediteur ist Automation sehr wichtig, damit nicht bei jedem Paket eine Eingabe von Hand nötig ist, sondern im besten Fall der reine Scan ausreicht.

Die gesamten Dokumente sind im Anhang, **??** zu finden. Für diesen Abschnitt wesentliche Kenntnisse wurden im Modul HCD [7] erarbeitet.

6 Ergebnisse

6.1 Evaluation der nichtfunktionalen Anforderungen (NFA)

Die in der Tabelle 6.1 beschriebenen nicht-funktionalen Anforderungen (NFA) wurden in der Aufgabenstellung vorgegeben. Die Evaluation hat ergeben, dass alle Anforderungen mit Ausnahme von Anforderung Nr. 4 erfüllt wurden. Die Anforderungen werden daher mit einer Quote von 94% erfüllt.

Bei Anforderung Nr. 4 ergab ein erster Test, dass die QR-Code Seite ca. 600ms zum Laden braucht. Das kann je nach Internetverbindung aber auch schneller sein.

Die genaue Evaluation mit Beschreibung ist im Anhang unter Abschnitt **??** zu finden.

6.2 Bezug zur Aufgabenstellung

Durch die Bestätigung durch die Evaluation Abschnitt 6.1 und den erfolgreichen letzten Systemtest kann ein überwiegender Teil der Anforderungen als erfüllt angesehen werden. Noch fehlende Funktionalen Anforderungen sind die folgenden, die hier noch einzeln betrachtet werden:

- **(FR.2)** *Eingeschränkter Zugriff auf die Applikation / Daten (Versender / Spediteur / Zoll / Empfänger)* Ja, dies ist gegeben, wenn auch nicht zu 100%. Grundsätzlich ist es nicht möglich, auf Bestellungen zuzugreifen, bei denen man nicht beteiligt ist. Der QR-Code Scan verhindert den unerlaubten Zugriff auch. Es ist jedoch stellenweise möglich, innerhalb des Frontends durch Manipulation der URL andere Views als vorgesehen aufzurufen und damit Aktionen auf der Lieferung vorzunehmen, die nicht vorgesehen sind. Auch wird durch das Backend nicht eingeschränkt, welcher Akteur einer Bestellung welchen Status erfassen darf.

Nr.	Nichtfunktionale Anforderung	Erfüllt?
1	Das Entwicklerteam implementiert die Features gemäss der abgesprochenen Priorität mit dem Kunden	Ja
2	Wenn ein QR-Code gescannt wird, sollte es nicht länger als 1sek. dauern, bis ein Resultat angezeigt wird	Ja
3	Das Backend sollte 1000 Requests pro Minute handeln können	Ja
4	Jede Seite sollte nicht länger als 200ms für das Laden benötigen	Nein
5	Die Web-Applikation sollte auf Firefox, Chrome und Safari auf Windows Surface Tablets, Android und Apple Handys laufen	Ja
6	Via Internet sollte auf eine vom Kunden zur Verfügung gestellte Domain zugegriffen werden können.	Ja
7	Drei von vier Test-user sollten das UI (Kategorien: layout, responsiveness, colour, content) der Applikation mit einem Tablet mit einer Note von mindestens 8 von 10 bewerten, wobei 10 das beste ist.	Ja
8	Die Datenbank soll bis zu 10'000 Lieferscheine managen können.	Ja
9	Errors sollen keine Systemfehler erzeugen, aber eine Error Nachricht Zeigen und das System auf den vorherigen Zustand zurücksetzen.	Ja
10	Jeder Error soll im System geloggt werden	Ja
11	Jede Kommunikation zwischen Front- und Backend soll mit einem SSL-Zertifikat verschlüsselt werden.	Ja
12	Daten welche in Eingabefelder abgefüllt werden, sollen zuerst validiert werden, bevor diese durch das System verarbeitet werden. SQL Injection test der Eingabefelder sollte keine Verletzlichkeiten zeigen.	Ja
13	User-Passwörter werden nicht in plain-text in der Datenbank gespeichert.	Ja
14	Wenn sich ein User in die Web-Applikation einloggt, werden ihm auch nur seine Daten / auf Daten die er Zugriff haben soll, angezeigt.	Ja
15	Businesslogik im Backend soll modular aufgebaut werden, so dass sie erweitert werden kann.	Ja
16	Die Backend-API soll durch API-testing Tools getestet werden.	Ja
17	Datenbank, Backend und Frontend sollen auf unterschiedlichen Instanzen deployed werden.	Ja

Tabelle 6.1: Resultate der Evaluation der Nicht-Funktionalen Anforderungen

- **(FR.6)** *Verfolgung des Paketes (wer hat es zuletzt gescannt)* Während nicht geloggt wird, wer eine Lieferung wann scannt, wird im Statuslog genau festgehalten, wer wann welche Aktion auf einer Lieferung vornimmt. Eine Verfolgung der Lieferung ist also gegeben. Leider kann dieses Functional Requirement zum aktuellen Zeitpunkt und ohne weitere Klärung nur als teilweise erfüllt angesehen werden.

- **(FR.7)** *Deployment in die Cloud (DigitalOcean)* Die Applikation ist wie vorgegeben bei DigitalOcean eingerichtet.

Abschliessend kann gesagt werden, dass die Anforderungen zu 95% erfüllt werden. Von den optionalen funktionalen Anforderungen konnten leider keine erfüllt werden.

6.3 Empfehlungen

Es gibt verschiedene Richtungen, in die die Anwendung weiter entwickelt werden kann. Es wäre daher sinnvoll, mit dem aktuellen Stand der Anwendung zuerst auf Industriepartner zuzugehen und herauszufinden, ob Interesse an der Dienstleistung besteht und was es dazu noch für den praktischen Einsatz brauchen würde. Die grösste Herausforderung ist vermutlich das Zusammenbringen von mehreren Kunden und Spediteuren die sich auf diese Plattform einlassen. Ein möglicher Ansatz wäre es, zu Beginn die Lieferkette zwischen zwei Firmen umzusetzen, und von da aus weitere Verbindungen aufzubauen.

Danach gibt es wie gesagt verschiedenste Richtungen, die von diesem MVP aus einschlagen werden können. Darum wurde eine Liste zusammengestellt, die diese Richtungen abdeckt und Möglichkeiten für die Weiterentwicklung der Applikation aufzeigt. In die nachfolgenden Vorschläge sind zudem auch die optionalen Anforderungen und Features eingewoben.

- Verbesserung der bestehenden Funktionen
  - Integration eines QR-Code Scanners in die Webapplikation (zum Beispiel mit dem Package *react-qr-reader*)
  - Vollständige Implementation der status- und rollenspezifischen Berechtigungen auf einer Bestellung
  - Evaluation, ob die aktuelle Authenitfizierungsmethodik und -implementation mit JWT in der Praxis praktikabel ist
  - Ausbau des Registrationsverfahrens mit Sicherheitsmassnahmen und Authentifizierungsprozessen wie Mailvalidierung
  - Ausbau des Logins mit Passwortwiederherstellung, 2FA-Authentifizierung, Mailbenachrichtigungen und Sessiontracking

  - Architektur und Funktionen auf Sicherheitslücken prüfen
  - Integration von Schnittstellen von bekannten Speditionen zur Sendungsverfolgung, um die Applikation schnell deutlich attraktiver werden zu lassen
  - Integration des Status *Verloren*: Lieferungen können als Verloren markiert werden oder werden automatisch als verloren markiert. Dies ermöglicht es auch, einen Recovery Prozess zu implementieren.
  - Ausbau der Funktionen, so dass auch Firmen die nicht bei der Software registriert sind, als Empfänger fungieren können und auch nach Empfang auf die Warenliste einer Lieferung zugreifen können
  - Restriktivere Zugriffsrechte für Spediteur oder Zoll: durch den Scan des QR-Codes wird eine temporäre Zugriffsberechtigung auf die Lieferung erteilt

- Ausbau für den Betreiber
  - Entwicklung eines Verwaltungs-Frontend für den Support von Kunden und anderen Partnern
  - Entwicklung eines Preismodells, Businessplans und alles weitere, was mit einem Serviceangebot kommt
  - Statistiken und Auswertungen über Verwendung der Applikation, kann auch für Kunden implementiert werden

- Ausbau für Kunden
  - Bestellungen auf mehrere Lieferungen aufteilbar machen
  - Integration von Etikettendruckern für den bequemen Druck von Etiketten
  - Automatische Vervollständigung der Waren in der Bestellaufnahme durch individuell hinterlegte Warenliste
  - Ausbau in Richtung SAP oder mit Schnittstelle zu Onlineshops oder anderen ERP Anbietern
  - Einbau von einem Prozess zur Unterstützung von Auflösung von Problemen mit der Lieferung
  - Ausbau des Dashboards mit Such- und Filterfunktionen für Lieferungen
  - Stornierung der Lieferung unterwegs ermöglichen

  - Einstellungen für den Benutzer wie Sprache, Schriftgrösse, Darkmode und andere ermöglichen. Denkbar wäre auch ein Fokusmodus auf einen bestimmten Bereich des Dashboards

  - Verwaltungsmöglichkeit für die Benutzerkonten der Mitarbeiter einer Firma via einem Administratorkonto

- Ausbau für Spediteure

  - Evaluation, wie der Standortupdateprozess automatisiert werden kann, damit der Aufwand für Spediteure so gering wie möglich ist und dadurch die Verwendung akzeptabel wird

  - Evaluation ob eine Verwaltungsoberfläche für Spediteure notwendig/nützlich ist

- Ausbau für den Zoll

  - Evaluation der nötigen Unterlagen zur Entwicklung eines Prozesses für die Zollabwicklung via Applikation

7 Projektauswertung

Die Unterlagen zum Projekt sind im **??** zu finden. Für diesen Abschnitt wesentliche Kenntnisse wurden im Modul PmQm [3] erarbeitet.

Als Projektmethode hat das Projektteam Scrum+ verwendet. Dies hat sich grundsätzlich bewährt. Mit den zweiwöchigen Iterationen konnten die Aufgaben gut geregelt und verteilt werden. Jedoch wurde Scrum bei weitem nicht vollständig eingesetzt, sondern nur Sprintplanung und Sprints. Da bereits bei zwei Personen ein spürbarer Mehraufwand für die Koordination entsteht, wurden die Artefakte eingeschränkt.

Im Verlauf des Projekts gab es verschiedene Hindernisse. Die erste Phase *Inception* und die zweite Phase *Elaboration* konnten nach Plan abgeschlossen werden. Die Resultate der Phase *Elaboration* waren im wesentlichen die Use Cases, das Datenmodell, die Wireframes und das Berechtigungskonzept als Entwurf. Jedoch wurde noch nicht mit der Implementation begonnen.

In der dritten Phase gab es dann Schwierigkeiten. Diese wurde durch die Verzögerung des Meilensteins Alpha um zwei Wochen und des Meilensteins Beta um eine Woche geprägt. Die Gründe dafür sind im Timetracking im **??** und in den Systemtest-Protokollen im **??** zu finden. In den ersten beiden Phasen wurde deutlich weniger Zeit investiert als das Wochen-Soll (ca. 17h). Dann erst wurde mit der Implementation gestartet. Danach gab es Verzögerungen, da Frontend und Backend abwechselnd aufeinander warten mussten. Druch den späten start der Implementation und der genannten Verzögerungen ergaben sich Verspätungen und zum Schluss musste beim Umfang des Testing Kompromisse eingegangen werden.

Die verlorene Zeit der ersten zwei Phasen wurde in der zweiten Hälfte des Projekts nachgeholt. Somit konnte trotz allem das Projekt erfolgreich abgeschlossen werden.

8 Zusammenfassung

8.1 Schlusswort

Im Rahmen dieses Studienarbeit wurde eine Applikation entwickelt, die den Verarbeitungsprozess rund um den Lieferschein digitalisiert. Die Resultate der Systemtests und Usability Tests als auch Analyse der Anforderungen bestätigen die weitgehende Erfüllung der Anforderungen und bestärken gleichzeitig die Qualität der Applikation.

Aktuell gibt es noch Probleme mit der Ladezeit, die statt der vorgegebenen 200ms bei 300-400ms liegt. Daneben werden zur Zeit Scans von QR-Codes nicht geloggt, nur Aktionen auf einer Lieferung werden festgehalten. Durch den Fokus auf Benutzbarkeit, Design und Funktionalität gibt es zudem Defizite in anderen Bereichen, wie zum Beispiel der Sicherheit.

Für die Weiterentwicklung wird eine Analyse der Marktsituation empfohlen und anschliessend eine gezielte Entwicklung für potentielle erste Kunden. Die grösste Herausforderung wird hier sein, ein vielseitig einsetzbares und flexibles Angebot zu gestalten, das trotzdem schlank und rank ist. Die Möglichkeiten für den Ausbau sind vorhanden und können abhängig von der weiteren Analyse priorisiert und ausgebaut werden.

8.2 Persönliche Reflexionen

Mathias Lenz

Die Studienarbeit war eine sehr wertvolle Erfahrung für mich. Innerhalb des Projekts habe ich mich stark in der Führung des Projekts engagiert. Daneben konnte ich viel Konzeptarbeit leisten als auch mein Können im Bereich Frontend Engineering mit der Entwicklung des Frontends und den Usability Tests verbessern. Zudem habe ich das erste Mal mit Latex gearbeitet, was auch eine sehr interessante Erfahrung war.

Mir hat es Spass gemacht, die Wireframes zu entwickeln und ein ansprechendes Design für die Applikation zu gestalten und gleichzeitig zu versuchen, die Benutzerfreundlichkeit so gut wie möglich zu machen. Herausfordernd war die Arbeit mit React, da ich React bisher nur aus einer Vorlesung kannte.

Bei der Projektorganisation werde ich beim nächsten Projekt auf jeden Fall früher mit der einer Implementation beginnen, um gezielt nach jeder Iteration ein MVP zu produzieren und so die Applikation agil aufzubauen. Hier bin ich aber auch noch unsicher, wie man die agilen Prinzipien am besten mit dem mir unflexibel vorkommenden Requirement Engineering verheiratet. Ich werde auch früher im Projekt und mehr Zeit in gezieltes kreatives und gemeinsames Ausarbeiten der Konzepte investieren.

Die Zusammenarbeit mit Khoa war jederzeit angenehm. Ich war froh, dass er sich beim Deployment und im Backend ins Zeug gelegt hat, und sich mit der Datenbank herumgeschlagen hat. Dies sind definitv nicht meine Stärken.

Zum Ende des Projekt hin war es recht arbeitsintensiv, der technische Bericht schreibt sich ja nicht von alleine. Allgemein hätte es sich sicher gelohnt, die aufgewendete Zeit etwas gleichmässiger im Projektverlauf zu verteilen. Abschliessend bin ich zufrieden mit der geleisteten Arbeit und der entwickelten Applikation, auch wenn mir immer wieder kleine Details auffallen, die man noch verbessern könnte.

Khoa Tran

Während dieser Arbeit habe ich gelernt, mit Technologien wie React.js und Node.js zu arbeiten. Bisher hatte ich noch wenig mit ihnen zu tun und hatte somit kaum Erfahrung.

Vor allem das Deployment der Applikation in die Cloud DigitalOcean erforderte Einarbeit in das Thema, da speziell die neuere App-Plattform noch weniger Benutzererfahrungen bietet.

Durch die Zusammenarbeit zusammen mit Mathias konnte ich lernen, wie man als kleines Team effizient eine Webapplikation von Grund auf bauen kann. Dies hat mir viel Spass gemacht, da Mathias vor allem im Design der Applikation sehr gute Arbeit geleistet hat und viel Erfahrung im Handling der Arbeit mitbrachte.

Rückblickend habe ich die Arbeitszeit am Anfang des Projekts nicht sehr gut eingeplant. Da ich nebenbei Vollzeit gearbeitet habe, fiel es mir schlecht, die Arbeitszeiten pro Woche einzuhalten. Ich hätte mein Arbeitspensum reduzieren sollen, um genügend Zeit wöchentlich in das Projekt investieren zu können. Trotz allem bin ich dennoch froh, dass Mathias und ich die Arbeit in dieser Zeit abschliessen konnten und alle Hauptfunktionalitäten umsetzen konnten.

9 Anhang

9.1 Glossar

MVP Minimum Viable Product - Erste minimal funktionsfähige Iteration eines Produkts

SAP Software von der gleichnamigen Firma zur Verwaltung von Businessprozessen und Kundenbeziehungen

DigitalOcean Cloud Hosting Plattform

Prisma Object-relational Mapping Tool

CRUD Abkürzung für Create, Read, Update und Delete

JWT JSON Web Token - Offener Standard, um geheime Informationen zwischen zwei Parteien zu übermitteln

CORS Cross-Origin Resource Sharing - Mechanismus, um Clients Cross Origin Requests zu erlauben.

Abbildungsverzeichnis

1 Formular zur Bestellerfassung und QR-Code auf Versandetikette 3
2 Das Dashboard mit einigen exemplarischen Lieferungen 4

2.1 Domain Model 14
2.2 Use Case Diagramm 18

3.1 Statusdiagramm 19
3.2 Übersicht über die wichtigsten Bestandteile des User Interface 23
3.3 Legende zu User Flows 24
3.4 Das Dashboard 25
3.5 second figure 25
3.6 Detailansicht Wireframe 26
3.7 QR-Code Verarbeitung 27
3.8 User Flows Warenausgang 28
3.9 User Flow Wareneingang 28
3.10 Data Model 30

4.1 Architekturdiagramm 33
4.2 Prismaschema 34
4.3 Programmstruktur im Frontend 35
4.4 Routing im Frontend 36
4.5 Winston Logging 37
4.6 Tägliche Logs 38
4.7 Error-Klassen 38
4.8 Code Error-Klasse 38
4.9 Error Middleware 39
4.10 Generierung QR-ID 42
4.11 Add Domain DigitalOcean 42
4.12 Deployment DigitalOcean 43
4.13 Konfiguration Repo DigitalOcean 43
4.14 Konfiguration CORS Policy 44
4.15 Konfiguration Run Command 45
4.16 App-Level Variabeln 45

Tabellenverzeichnis

3.1 Änderbarkeit des Status einer Bestellung 22
3.2 Sichtbarkeit einer Bestellung im Dashboard 22
3.3 Bereiche des Dashboard 25

6.1 Evaluation der NFA 49

Literaturverzeichnis

[1] Prisma sequence documentation. https://www.prisma.io/docs/reference/api-reference/prisma-schema-reference#sequence. Accessed: 2022-04-10.

[2] Prisma website. https://www.prisma.io/. Accessed: 2022-03-24.

[3] Claudio Fuchs. Modul projekt- und qualitätsmanagement. OST, Vorlesung im HS21.

[4] Bobby Iliev. DigitalOcean ssl for digitalocean apps. https://www.digitalocean.com/community/questions/ssl-for-digital-ocean-apps. Accessed: 2022-04-30.

[5] Thomas Kälin. Software-engineering 2. OST, Vorlesung im FS21.

[6] Mirko Stocker. Web engineering + design 3. OST, Vorlesung im FS21.

[7] Markus Stolze. Human computer interaction design. OST, Vorlesung im FS21.

[8] Markus Stolze. Verwendung jwt bei authentifizierung. Modul WED2 an der OST, Vorlesung im HS21.