Hochschule Emden/Leer

Praxisprojekt

im Studiengang Medieninformatik

Entwicklung einer Client-Server-Architektur am Beispiel einer Anwendung zur Verwaltung von Selbstbedienungsständen

Autorin: Christine Dall
MatNr. 7005906

Version vom: 14. Oktober 2021

1. Betreuer: Dipl.-Inform. Andreas Wilkens
2. Betreuerin: Dipl.-Dok. Inessa Stanke

Zusammenfassung

Die vorliegende Arbeit, beschäftigt sich mit der Entwicklung einer Client-Server-Architektur am Beispiel einer Anwendung zur Verwaltung von Selbstbedienungsständen.

Im ersten Teil dieser Arbeit werden die Grundlagen, die zum späteren Verständnis wichtig sind, erläutert. Es wird erklärt, was ein Selbstbedienungsstand ist, außerdem werden einige gesetzliche Grundlagen vorgestellt. Anschließend wird etwas genauer beleuchtet, was unter einer Client-Server-Architektur zu verstehen ist. Im Anschluss werden unterschiedliche Frameworks dargestellt.

Das zweite Kapitel widmet sich der Anforderungsanalyse, dort werden die wichtigsten Anforderungen genauer betrachtet. Die einzelnen Anforderungen werden kurz erörtert und etwas näher beschrieben.

Darauf aufbauend, wird im dritten Kapitel mit der Konzeption der Anwendung begonnen. Zunächst wird der allgemeine Architekturentwurf präsentiert. Danach folgt im Anschluss der Entwurf des Servers und der Datenbank. Hiernach wird auf die Konzeption der Schnittstellen eingegangen. Der letzte Teil des Kapitels beschäftigt sich mit der Konzeption des Clients.

Im Fokus des vierten Kapitels steht die Realisierung und Implementierung der Anwendung. Zunächst wird die Datenbank implementiert, es werden einige Datenbankabfragen vorgestellt. Nachfolgend wird auf die Implementierung des Servers eingegangen. Näher werden einige Beispielcodeausschnitte erläutert. Zuletzt wird auf die Implementierung des Angular-Clients eingegangen. Dabei wird das Kapitel aufgesplittet in Implementierung der grafische Benutzeroberfläche(Frontend) und Implementierung des Hintergrundbereiches (Backend).

Zum Schluss folgt eine Zusammenfassung und ein Ausblick.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Einleitung

Verbraucher achten beim Kauf von Lebensmitteln verstärkt darauf, dass diese aus der Region stammen, belegt eine Umfrage [umf21b] aus dem Jahr 2020. In dieser Umfrage gaben ca. 33 Prozent an, dass Sie nach Möglichkeit Produkte aus der Region kaufen. Gleichzeitig bieten immer mehr Erzeuger Ihre Ware in sogenannten Selbstbedienungsständen an. Dort können Verbraucher regionale Lebensmittel kaufen. Es werden beispielhaft Eier, Kartoffeln, Kürbisse, Honig usw. angeboten. Es finden sich aber auch immer mehr Selbstbedienungsstände, in denen keine Lebensmittel, sondern selbst hergestellte Produkte verkauft werden.

Die entwickelte Anwendung soll dabei helfen, die Verwaltung eines Selbstbedienungsstandes zu vereinfachen.

Das Ziel der Projektarbeit ist die Entwicklung einer Client-Server-Architektur am Beispiel einer Anwendung zur Verwaltung von Selbstbedienungsständen.

Die Arbeit beschreibt nicht den gesamten Entwicklungsablauf, da dieses den Rahmen sprengen würde, es werden nur einige Entwicklungsschritte genauer erläutert.

Die wichtigste Literatur, die verwendet wird ist das Buch „Java ist auch eine Insel“ [Ull17] sowie die Dokumentation von Angular [Ang21a]. Des Weiteren werden einige unterschiedliche Onlinequellen und Bücher eingesetzt.

2 Grundlagen

Das erste Kapitel beschäftigt sich mit den Grundlagen, diese sind zum Verständnis der nachfolgenden Kapitel notwendig.
Zuerst wird erläutert, was ein Selbstbedienungsstand ist. Anschließend wird es einen kleinen Überblick über einige gesetzliche Grundlagen geben.
Im nächsten Abschnitt werden Begriffsdefinitionen eingeführt und näher erläutert.
Das anschließende Unterkapitel liefert einen Überblick über Frameworks, die zur Realisierung einer Client-Server-Architektur genutzt werden können.

2.1 Selbstbedienungsstand

Immer mehr Landwirte oder andere Anbieter bieten ihre Produkte in Direktvermarktung an. Eine Studie [dir20] aus dem Jahr 2020 belegt, das 47 Prozent der konventionellen Betriebe und 70 Prozent der Bio-Betriebe in Zukunft die Direktvermarktung weiter ausbauen möchten.
Es werden dabei unterschiedliche Vermarktungswege gewählt, z. B. Hofladen, Wochenmarkt, Abo-Kiste, Onlineshop oder Selbstbedienungsstand. [Reg20]. Jeder dieser Wege bietet unterschiedliche Vorteile und Nachteile. Im Nachfolgenden wird auf den Selbstbedienungsstand mit offener Ladenkasse näher eingegangen.

Abbildung 1: „Kartoffelkiste“ [Kar14]

Der Begriff Selbstbedienung wird von [Sel] wie folgt definiert „Verkaufsprinzip im Handel, bei dem der Kunde aus einem frei zugänglichen und griffbereit ausgestellten Warensortiment ohne Mitwirkung des Verkaufspersonals die von ihm gewählten Artikel zu den betrieblichen Inkassostellen transportiert.“

Abbildung 2: Eierhaus

Bei einem Kauf in einer Selbstbedienungshütte bezahlt der Kunde seine Ware selbstständig, ohne Verkaufspersonal und wirft das Geld in einen dafür vorgesehenen Behälter. Dieses kann zum Beispiel eine abschließbare Geldkassette mit Münzeinwurf sein.
Der Stand bzw. der Ort, an dem die Waren angeboten werden, kann die unterschiedlichsten Bauweisen oder Formen aufweisen.
Auf dem Bild 1 und dem Bild 2 sind zwei unterschiedliche Bauweisen zu sehen. Das Bild 1 zeigt einen Straßenstand, auf dem zweiten Bild 2 ist eine Blockhütte zu sehen.

2.2 Gesetzliche Grundlagen

In diesem Kapitel werden einige gesetzliche Grundlagen vorgestellt. Es ist nicht möglich auf alle gesetzlichen Vorschriften einzugehen, da es zu viele unterschiedliche Reglungen gibt. Außerdem werden die hier vorgestellten Vorschriften und Reglungen nicht im Detail erläutert. Für ausführlichere Informationen muss der Betreiber sich bei den zuständigen Stellen erkundigen.

Als Erstes wird geklärt, was Direktvermarktung landwirtschaftlicher Erzeugnisse eigentlich bedeutet. [ges17] schreibt „Unter Direktvermarktung versteht man die direkte Abgabe landwirtschaftlicher Produkte durch den Erzeuger auf dem Hof, auf dem Markt, an der Tür oder über eigene Hofläden an den Verbraucher.“

Werden nur „selbsterzeugte unverarbeitete landwirtschaftliche Produkte, die als Urprodukte gelten vermarktet“, ist dieses „kein Gewerbe im Sinne der Gewerbeordnung“ erwähnt [ges10a]. Außerdem heißt es dort weiter „werden (Ur-)Produkte für den Verkauf gereinigt, sortiert und hergerichtet (sogenannte erste Verarbeitungsstufe), so ist dies unschädlich.“

Tabelle: Schritte der Weiterverarbeitung

Urproduktion	***erste Verarbeitungsstufe*** *(= Nebenbetrieb d. Landwirtschaft)*	***zweite Verarbeitungsstufe*** *(= Gewerbe)*
Schweine, Schafe, Ziegen, Wild, Rinder	schlachten und zerlegen in Hälften (Schweine, Schafe, Ziegen, Wild) bzw. Viertel (Rinder)	weitere Zerlegung, bratfertige Stücke, Herstellung von Wurst, Schinken etc.
Nebenprodukte	Häute, Felle, Wolle	Strickwaren, Kleidung
Puten, Gänse	schlachten und zerlegen in Hälften	weitere Zerlegung und Verarbeitung
sonstiges Geflügel	Verkauf ganzer Tiere	weitere Zerlegung und Verarbeitung
Fische	Fischfilet (auch geräuchert)	weitere Verarbeitung
Milch und Milchprodukte	Milch, Butter, Quark, Käse, Joghurt, andere Erzeugnisse mit mind. 75% Milchanteil	Kondensmilch, Speiseeis, Milchpulver
Eier	kochen, färben	Nudeln, Eierlikör
Getreide	Mehl, Schrot, Flocken	Brot, Backwaren, Kuchen, Müsli
Obst	schälen, zerkleinern, trocknen, einlegen, Säfte, Obstwein	Liköre, Schnäpse, Konfitüren, Fruchtaufstriche
Gemüse, Kartoffeln	schälen, zerkleinern, einlegen, konservieren, Säfte	Fertiggerichte
Weintrauben	Most, Wein, Winzersekt, Branntwein aus Wein (Roh-/Feinsprit)	Branntweinerzeugnisse, Weinbrand

Tabelle 1: Urproduktion, „Schritte der Weiterverarbeitung“ [ges15]

In der Tabelle 1 sind einige Urprodukte der ersten und zweiten Verarbeitungsstufe abgebildet.
Folgende Punkte führen zur Anzeigepflicht eines Gewerbes:[ges15]

- Der Verkauf von weiterverarbeiteten Produkten (zweite Verarbeitungsstufe) erzielt einen Umsatz von 10 Prozent des Gesamtumsatzes oder mehr des Betriebes.
- Verkauf von zugekaufter Ware, deren Umsatz 10 Prozent oder mehr des Betriebsumsatzes ausmachen.
- Verkauf und Urproduktion sind räumlich und personell getrennt. Ausgenommen hiervon ist ist der Marktstand.

Eine Vertrauenskasse wird laut [gesc] wie folgt definiert „Kasse, in die der Käufer selbständig den Kaufbetrag für die gekaufte Ware oder Leistung einzahlt, ohne dass in irgendeiner anderen Form der Verkäufer kassiert“.

[Bun18] schreibt „Als offene Ladenkasse gelten eine summarische, retrograde Ermittlung der Tageseinnahmen sowie manuelle Einzelaufzeichnungen ohne Einsatz technischer Hilfsmittel (BMF-Schreiben v. 19.6.2018).“ Der Begriff offene Ladenkasse bezieht sich nicht auf die Bauform der Kasse, sondern auf die Art, wie die Tageseinnahmen ermittelt werden.

Ein Landwirt, der direkt seine Produkte vermarktet, muss alle Einnahmen aufzeichnen. Diese müssen „in einem Kassenbericht vollständig, richtig, zeitgerecht und geordnet aufgezeichnet werden“, erwähnt [gesb]. Dort heißt es weiter „Laut Gesetz kommt bei Bargeldgeschäften der Grundsatz der Einzelauszeichnung zur Anwendung, [...] Da dieser Grundsatz in einem Hofladen [...] nicht praktikabel ist, gelten bestimmte Vereinfachungen.“ Diese Vereinfachungen besagen das es reicht, täglich die Summer der verkauften Waren und deren Einnahmen zu protokollieren.

Tabelle 2: Übersicht, Versteuerung Direktvermarkter [ges10b]

In der Tabelle 2 sind mögliche Versteuerungen zu sehen. Wichtig zu wissen, dies ist nur eine Übersicht, es kann möglicherweise sein, dass es im Steuerrecht abweichende Grenzen und Regeln gibt.

Nachfolgend werden ein paar weitere gesetzliche Reglungen aufgezählt, die eventuell beachtet werden müssen: Handwerksordnung, Gewerbeordnung, Steuerrecht, Baurecht, Lebensmittelhygienerecht, Qualitätskennzeichen, Verpackungsgesetz usw..

2.3 Begriffsdefinitionen und Begriffserklärung

Dieses Kapitel soll einen Überblick über verwendete Begriffsdefinitionen geben, zusätzlich werden einige Begriffe etwas genauer beschrieben.

2.3.1 Client-Server-Architektur

Zunächst wird dargestellt, was ein Server ist, anschließend wird auf den Client eingegangen.

Laut [cli20] sind Server und Client keine reinen Hardwarekomponenten, sondern der Begriff „Server und Client“ beschreibt Rollen.

[cli20] stellt fest das die Aufgabe eines Servers daraus besteht „einen bestimmten Dienst lokal oder über das Netzwerk bereitzustellen.“ Ein Server kann zum Beispiel Daten, Anwendungen, Dienste usw. bereitstellen.
[Win08] [S.116] schreibt „Eine Hardware- oder Softwarekomponente, der Dienste von einem Server in Anspruch nehmen kann (Client-Server-Prinzip) wird Client (Kunde) genannt.“ Dieses könnte zum Beispiel ein Computer sein, der die Dienste eines Servers in Anspruch nimmt.
Der Autor [cli20] erwähnt „Das Client-Server-Modell ist ein Architekturkonzept zur Verteilung von Diensten und Aufgaben in einem Netzwerk. Dienste werden von Servern bereitgestellt und können von Clients genutzt werden.“.

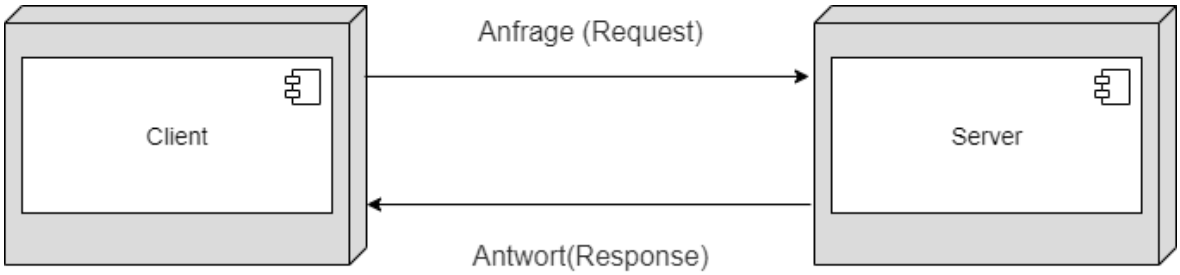

Abbildung 3: Vereinfachte Darstellung einer Kommunikation zwischen Client und Server

Die Abbildung 3 stellt eine vereinfachte Darstellung einer Kommunikation zwischen Client und Server dar. Ein Client sendet eine Anfrage (Request) an einen Server, dieser verarbeitet die Anfrage und schickt eine Antwort (Response) zurück an den Client. Durchaus mehrere Clients können auf einen Server zugreifen. Außerdem kann sowohl der Client und der Server auf den selben Rechner ausgeführt werden.
[cli20] schreibt, dass die Kommunikation und der Informationsaustausch über Protokolle geregelt wird. Bekannte und häufig verwendete Protokolle sind laut [cli20] das „FTP (File Transfer Protokoll), HTTP (Hypertext Transfer Protocol) und das SNMP (Simple Network Management Protocol)“.
Ein Server baut nicht selbstständig eine Verbindung auf, sondern wartet auf eine Verbindungsanfrage.
Ein typisches Beispiel einer Client-Server-Architektur ist ein Webbrowser, der auf einem Client läuft, dieser sendet eine Anfrage an einen Webserver und möchte eine bestimmte Internetseite aufrufen. Der Server prüft diese Anfrage und sendet die gewünschten Daten zurück.
Einer der am häufigsten genutzten Webservern, ist der Apache HTTP Server[1].

[1] https://httpd.apache.org

2.3.2 REST-API

REST-API ist die Abkürzung von „Representational State Transfer - Application Programming Interface “ beschreibt [RES18]. Eine REST-API ist eine Programmierschnittstelle, mithilfe einer Schnittstelle ist es möglich, Daten und Informationen zwischen unterschiedlichen Systemen auszutauschen. Der Autor [RES17] erklärt „Der als REST (oder auch ReST) bezeichnete Architekturansatz beschreibt, wie verteilte Systeme miteinander kommunizieren können.“. Gemäß [RES18] werden HTTP-Anfrage genutzt um auf Informationen zuzugreifen. Die bekanntesten oft genutzte HTTP-Anfragen sind: [RES17]

- GET - Der Client fordert Daten vom Server an.
- POST - Der Server sendet die angeforderten Daten an den Client.
- PUT/PATCH - Der Server ändert vorhandene Daten.
- DELETE - Es sollen Daten vom Server gelöscht werden.

Schreibt das [Hor07] alternativ auch das Nachrichtenprotokoll SOAP (Simple Object Access Protocol) verwendet werden kann.

„Die Antworten werden oft im JSON-Format (JavaScript Object Notation) geliefert. Das Format ist sowohl für Menschen als auch Maschinen lesbar.“ berichtet [RES].

```
{
  "Benutzer": {
    "Name": [
    {
      "id": 1,
      "Vorname": "Christine",
      "Nachname": "Dall",
      "Administrator": true,
      "Adresse": null
    },
    {
      "id": 2,
      "Vorname": "Christin",
      "Nachname": "Ball",
      "Administrator": false,
      "Adresse": "Neuer Weg 6, 12345 Ort"
    },
    {
      "id": 3,
      "Vorname": "Christina",
      "Nachname": "Fall",
      "Administrator": false,
```

```
23          "Adresse": null
24      }
25      ]
26    }
27 }
```

Listing 1: Beispiel für Daten im JSON Format

Im Listing 1 zu sehen ist eine beispielhafte Datenstruktur. Dieses könnten zum Beispiel Daten, sein die der Client über eine Get-Anfrage erhält.

Außerdem wird vom Server auf jede Anfrage ein HTTP-Statuscode mitgesendet. Bekannte Statuscodes sind unter anderem: [Sta21]

- 200, OK
- 201, Erzeugt
- 400, Ungültige Anfrage
- 403, Verboten
- 404, Nicht gefunden
- 408, Anfrage-Zeitüberschreitung
- 409, Konflikt
- 500, Interner Server-Fehler

2.3.3 Frameworks zur Realisierung einer Client-Server-Architektur

Ein Framework ist ein Programmiergerüst, dieses wird überwiegend in der objektorientierten Programmierung eingesetzt, dieses bildet den Rahmen und Anwendungsarchitektur einer Anwendung definiert [Fraa]. Weiter heißt es dort „Das Framework umfasst Bibliotheken, Komponenten und Laufzeitumgebungen und stellt die Designgrundstruktur für die Entwicklung zur Verfügung.“.

Die Arbeit mit Frameworks bietet einige Vorteile. [Frab] erwähnt folgenden Vorteil, „Wiederkehrende Aufgaben können schneller abgewickelt werden,“ bei der Verwendung von Frameworks spart sich der Programmierer Zeit und Aufwand.

Nachfolgend werden ein paar Frameworks und eine Bibliothek vorgestellt.

Der Unterschied zwischen Bibliotheken und Frameworks ist „Bei einer Bibliothek handelt es sich um eine Sammlung von Programmroutinen, die oft einen thematischen Zusammenhang aufweisen.“ schreibt [Ang21c]. Bibliotheken ergänzen Programmiersprachen um Funktionen. Ein Framework ist ein universelles Rahmenwerk und „Um dieses rasch und übersichtlich einsetzen zu können, verwendet es nicht selten eine ergänzte oder variierte Syntax beziehungsweise eine eigene Programmiersprache und Struktur.“ wird noch ergänzend geschrieben von [Ang21c].

Vorgestellt werden Angular, React, Vue.js und Spring Boot.

	Angular
Typ	Web-Framework
Veröffentlichung	2016 entwickelt von Google Inc. [Ang18a]
Aktuelle Version	Angular 12.1.1 (Stand: 30.06.2021)
Lizenz	MIT-Lizenz
Betriebssystem	Plattformunabhängig
Programmiersprache	TypeScript
Webseiten die Angular nutzen	Youtube, PayPal

Bemerkung
Angular ist der Nachfolger von AngularJS und wurde komplett neu geschrieben.
Der Autor [Ang] erwähnt das mit Angular komponentenbasiert entwickelt wird.
Eine Komponente besteht aus: [Ang]

- Template, enthält das HTML-Grundgerüst.
- Klasse, enthält die Logik die das Verhalten der Komponente steuert.
- Stylesheet, enthält das Layout.

Vorteile

- Große Community [Ang20b]
- Große Typsicherheit, da TypeScript verwendet wird. [Ang17]
- Erfordert keine zusätzlichen Bibliotheken [Ang20b].
- „Komponentenbasierte Architektur“ [Ang]
- Große Wiederverwendbarkeit der Komponenten [Ang21b]
- „Zukunftssicher“ [Ang]
- Stabiles und verlässliches Framework [Ang20c]
- Hohe Performance. [Ang]
- Gut bei anspruchsvollen Backend-Anwendungen. [Ang20b]
- Verwendet HTML, CSS, Javascript, dieses ist vielen Entwicklern bereits bekannt. [Ang17]
- Enthält viele Features („Dependency Injection, Templates, Routing, Ajax Requests“ [Ang18a])
- Gute Dokumentation

Nachteile

- Hohe Einstiegshürde [Ang20b]
- Komplexes Framework [Ang21b]
- Abnehmender Community Support [Ang21b]
- Angular Versionen sind wenig kompatibel untereinander. [Ang20c]
- Geringere Flexibilität [Ang18a]

Tabelle 3: Vorstellung des Frameworks Angular

	React
Typ	JavaScript-Bibliothek
Veröffentlichung	2013 entwickelt von Facebook Inc. [Ang18a]
Aktuelle Version	17.0.02 (Stand: 30.06.2021)
Lizenz	MIT-Lizenz
Betriebssystem	Plattformunabhängig (Ab Version 3)
Programmiersprache	JavaScript, ab Version 3 auch TypeScript
Webseiten die React nutzen	Facebook, Instagram, Discord, Pinterest

Bemerkung
[Ang18a] schreibt React besteht nur aus der View-Komponente. Diese Komponente beinhaltet „den UI-Teil und die Logik der Applikation“.

Vorteile

- Virtuelles DOM (Document Object Model), Veränderungen am eigentlich DOM werden dadurch klein gehalten. [Ang17]
- Große Flexibilität [Ang21b]
- „schlank, klein und elementar gehalten“ am Anfang sind nur Grundfunktionen verfügbar. [Ang20a]
- Beliebt, dadurch große Community vorhanden[Ang21b]
- Keine so hohe Einstiegshürde wie Angular. [Ang17]
- Gute Unterstützung interaktiver Elemente. [Ang20b]
- Gute Dokumentation
- Es werden regelmäßig Updates veröffentlicht.

Nachteile

- Erfordert eventuell zusätzliche Bibliotheken. [Ang18a]
- Wenig Futures, benötigte Packages müssen nachinstalliert werden. [Ang18a]
- Template wird mit JSX geschrieben, Sprache muss eventuell gelernt werden. [Ang20a]
- Der Einsatz lohnt sich nur für Seiten die viele Interaktive Elemente enthalten. [Ang17]

Tabelle 4: Vorstellung der Bibliothek React

	Vue.js
Typ	Web-Framework
Veröffentlichung	2014 entwickelt Evan You [Ang18a]
Aktuelle Version	17.0.02 (Stand: 30.06.2021)
Lizenz	MIT-Lizenz
Betriebssystem	Plattformunabhängig (Ab Version 3)
Programmiersprache	JavaScript, ab Version 3 auch TypeScript
Webseiten die Vue.js nutzen	Gitlab, Nintendo, Shopware
Bemerkung Bei Vue.js„handelt es sich hier um eine monolithische Lösung, die dem Entwickler alle nötigen Tools für eine umfassende Applikation anbietet.“ [Ang20a]	
Vorteile • Leichter Einsieg [Ang18b] • Modularer Aufbau [Ang18c] • Schlanker Aufbau, unterschiedliche Module können hinzugefügt werden.[Ang19] • „50 % weniger Renderzeit: im Vergleich zu anderen Frameworks.“[Ang20d] • Virtuelles DOM (Document Object Model), Veränderungen am eigentlich DOM werden dadurch klein gehalten. [Ang18c] • Ist nur 10 kB groß.(Version 3) [Ang20d]	
Nachteile • Kleine Community [Ang21b] • Kleines Entwicklerteam [Ang18c] • Wenig Erweiterungen. [Ang21c]	

Tabelle 5: Vorstellung des Frameworks Vue.js

	Spring Boot
Typ	Application Framework
Veröffentlichung	2002 entwickelt von Rod Johnson, ab September 2009 VMware
Aktuelle Version	5.3.8 (Stand: 30.06.2021)
Lizenz	Apache-Lizenz
Programmiersprache	Java, Kotlin, Groovy
Bemerkung [Spr19b] definiert „Das Spring Framework ist ein schlankes Open-Source-Framework für Java. Mittels Dependency Injection und aspektorientierter Programmierung soll es einen insgesamt leichteren und besser wartbaren Programmcode ermöglicht.“	
Vorteile • MVC-Architektur • Dependency Injection • Aspektorientierte Programmierung • Flexible Modulsammlung • Große Community • Ausführliche Dokumentation • Flache Lernkurve [Spr18] • Unterstützt viele Datenbanken [Spr18]	
Nachteile • Höhere Code-Ausführungszeit • Erfordert Einarbeitungszeit	

Tabelle 6: Vorstellung des Frameworks Spring Boot

2.4 SQL-Datenbank

Datenbanken werden in fast jeder größeren Softwareanwendung in der IT verwendet. In einer Datenbank werden Daten bereitgestellt und gespeichert, die für den Betrieb der Software nötig sind. Das können zum Beispiel folgende Daten sein: Benutzername, Passwort, Artikelinformationen usw..

Gemäß [Win08] [S.55] ist eine Datenbank, eine strukturierte, dauerhaft elektronisch gespeicherte Sammlung von Datenmengen. Eine Datenbank kann mithilfe eines Datenbankmanagementsystems (DBMS) verwaltet werden. Bekannte DBMS sind unter anderem MySQL[2] und Oracle[3].

In der Praxis werden unterschiedliche Datenbankmodell verwendet, eines der am häufigsten genutzten Systeme ist das relationale Datenbankmodell. [t3r17] schreibt, dass eine relationale Datenbank aus unterschiedlich vielen Tabellen bestehen kann. Es werden logisch zusammenhängende Daten miteinander verknüpft (in Relation gesetzt). [Ora] beschreibt eine relationale Datenbank als „ein Typ von Datenbanken, der die Speicherung und den Zugriff auf miteinander verbundener Datenpunkte ermöglicht.“.

Mithilfe einer Datenbanksprache lassen sich unter anderem Daten einfügen, löschen oder verändern. Diese Datenbankoperationen sind auch unter dem Begriff CRUD bekannt. [db219] erklärt das die Abkürzung sich aus den Begriffen Create, Read, Update, Delete zusammensetzt.

In dieser Praxisarbeit wird SQL verwendet, die Abkürzung steht für Structured Query Language. Gemäß [dat] ist SQL „eine Datenbanksprache zur Erstellung von Datenbankstrukturen in relationalen Datenbanken sowie zum Bearbeiten und Abfragen der darauf basierenden Datenbeständen.“.
[Lub17] erwähnt das SQL keine vollwertige Programmiersprache ist, das heißt hiermit können keine kompletten Anwendungen erstellt werden, aber SQL lässt sich gut mit Programmiersprachen kombinieren. [bit] schreibt das SQL, durch die weite Verbreitung zum Standard in Deutschland geworden ist.

Laut [Lub17] lassen sich die SQL-Befehle in drei unterschiedliche Kategorien aufteilen. Die drei Kategorien sind, DML-Befehle (Data Manipulation Language Befehle), DDL-Befehle (Data Definition Language Befehle) und DCL-Befehle (Data Control Language Befehle).
DML-Befehle werden zum Bearbeiten, Löschen und Einfügen verwendet. Bekannte und oft verwendete Befehle werden in der Tabelle 7 dargestellt.
DDL-Befehle sind dazu geeignet, die Struktur der Datenbank zu steuern. Bekannte DDL-Befehle sind in der Tabelle 8 abgebildet.
Außerdem gibt es noch die DLC-Befehle, diese Befehle helfen bei der Rechteverwaltung. Beispiele hierfür sind in der Tabelle 9 zusehen.

[2] https://www.mysql.com/de/
[3] https://www.oracle.com/database/

In den nachfolgenden Tabellen werden einige Beispielbefehle, entnommen aus [SQL21], aufgezeigt.

Befehl	**Beschreibung**
SELECT * FROM Tabellenname	Mit diesem Befehl werden Daten aus Tabellen ausgelesen.
INSERT INTO Tabellenname (Spaltenname1, Spaltenname2, ...) VALUES (Wert1, Wert 2, ...)	Dieser Befehl fügt Daten in die Tabelle ein.
DELETE FROM Tabellenname WHERE Bedienung	Mithilfe dieses Befehls werden Daten aus der Tabelle gelöscht.
UPDATE Tabellenname SET (Spalte1 = Wert1, Spalte2 = Wert2, ...) WHERE Bedienung	Dieser Befehl sorgt dafür, dass Daten in der Tabelle aktualisiert werden.

Tabelle 7: DML-Befehle

Befehl	**Beschreibung**
CREATE DATABASE Datenbankname	Mithilfe dieses Befehls wird eine neue Datenbank erstellt.
CREATE TABLE Tabellenname (Spaltenname1 Datentyp1, Spaltenname2 Datentyp2, ...)	Dieser Befehl sorgt dafür, dass eine neue Tabelle angelegt wird.
DROP TABLE Tabellenname	Soll eine Tabelle gelöscht werden, kann dieser Befehl genutzt werden.
ALTER TABLE Tabellename ADD Spaltenname Datentyp	Es wird eine neue Spalte in der Tabelle eingefügt, sobald dieser Befehl genutzt wird.

Tabelle 8: DDL-Befehle

Befehl	**Beschreibung**
GRANT Privileg ON Datenbankname TO {Benutzername |PUBLIC |Rollenname}[WITH GRANT OPTION]	Die Zuggriffrechte werden mithilfe dieses Befehls gewährt.
REVOKE Privileg ON Datenbankname FROM {Benutzername |PUBLIC |Rollenname}	Dieser Befehl entzieht dem Benutzer Zugriffsrechte.

Tabelle 9: DLC-Befehle

3 Anforderungsanalyse

In diesem Kapitel wird eine Anforderungsanalyse durchgeführt, Ziel ist es, die Anforderungen für die zu erstellte Anwendung zu ermitteln. Zunächst muss allerdings eine kleine Problemanalyse durchgeführt werden. Diese Analyse dient dazu, das Problem näher zu erläutern.

Unterschieden wird zwischen zwei Anforderungsarten.[Fun].

- Mithilfe funktionaler Anforderungen wird festgelegt was das System tun soll.
- Nichtfunktionale Anforderungen legen fest, wie das System etwas tun soll.

3.1 Problemanalyse

Zuerst wird erläutert, welches Problem die Anwendung lösen soll.
Es muss jeden Tag, an dem Ware verkauft wird, eine Abrechnung erstellt werden. Diese Anwendung soll dabei helfen, diesen Vorgang zu vereinfachen. Genauer gesagt soll die Anwendung helfen, einen Überblick über die verkaufte Ware zu erhalten. Diese soll an einem Fallbeispiel verdeutlicht werden.

In einer Hütte werden unter anderem Eier verkauft. Am Tagesbeginn befinden sich bereits 254 Eier in der Hütte. Im Laufe des Tages werden insgesamt 480 Eier in die Hütte gebracht. Die Eier werden von unterschiedlichen Personen in unterschiedlicher Anzahl in die Hütte gestellt. Jeder dieser Personen muss immer irgendwo die genaue Stückzahl festhalten. Am Ende des Tages wird eine Abrechnung erstellt. Dazu werden die in der Hütte vorhanden Eier gezählt, dieses sind 142 Stück, anschließend wird folgende Rechnung aufgestellt.

Anzahl der vorhandenen Eier vom Vortag + Summe der Eier aus Zugängen - Anzahl der aktuell gezählten und noch in der Hütte vorhanden Eier = Summe der theoretisch verkauften Eier.

254 Eier + 480 Eier - 142 Eier = 592 verkaufte Eier

Hierbei sind Diebstähle usw. nicht berücksichtigt. Anschließend wird das Geld in der Kasse gezählt und mit der Summe verglichen, die es theoretisch sein sollte.

Aufwendiger wird dieser Vorgang, sobald mehr als eine Warensorte angeboten wird. Einige Erzeuger betreiben auch mehrere Selbstbedienungsstände, für jeden davon muss eine separate Abrechnung erstellt werden.

3.2 Funktionale Anforderungen

Aus der Problemstellung lassen sich folgende funktionale Anforderungen ableiten.

Funktion: **Benutzer und Selbstbedienungsstand neu anlegen**
Beschreibung: Es wird ein neuer Benutzer angelegt. Der Benutzer erhält Adminrechte. Es wird überprüft, ob der Benutzername vergeben ist, ebenso ob das Passwort den Anforderungen entspricht. Außerdem wird überprüft, ob der Name des Selbstbedienungsstandes bereits genutzt wird.

Funktion: **Benutzername oder Passwort ändern**
Beschreibung: Der Benutzername bzw. das Passwort des Benutzers wird geändert. Es wird überprüft, ob der Benutzername bereits vergeben ist bzw. ob das Passwort den Anforderungen entspricht.

Funktion: **Benutzer löschen**
Beschreibung: Der Benutzer wird gelöscht.

Funktion: **Mehrbenutzer**
Beschreibung: Einem Selbstbedienungsstand können mehrere Benutzer zugeordnet werden.

Funktion: **Selbstbedienungsstand anlegen**
Beschreibung: Es wird ein neuer Selbstbedienungsstand angelegt. Zuvor wird überprüft, ob bereits ein Selbstbedienungsstand mit gleichen Namen existiert.

Funktion: **Selbstbedienungsstand, Name ändern**
Beschreibung: Der Name des Selbstbedienungsstands wird geändert. Nur der Admin des Selbstbedienungsstandes kann diesen ändern. Es wird überprüft, ob bereits ein Selbstbedienungsstand mit diesem Namen erstellt worden ist.

Funktion: **Selbstbedienungsstand löschen**
Beschreibung: Der Selbstbedienungsstand wird gelöscht. Nur der Admin des Selbstbedienungsstandes kann diesen löschen.

Funktion: **Selbstbedienungsstand, Benutzer hinzufügen**
Beschreibung: Der Admin fügt weitere Benutzer hinzu, diese Nutzer gehören dann ebenfalls zu dem Selbstbedienungsstand.

Funktion: **Selbstbedienungsstand, neuen Benutzer erstellen**
Beschreibung: Der Admin legt einen weiteren Benutzer an, diese Nutzer gehören dann ebenfalls zu dem Selbstbedienungsstand.

Funktion: **Ware anlegen**
Beschreibung: Der Admin kann neue Ware anlegen. Folgende Wareninformationen sollen hinterlegt sein: Warenname, Einheit, Preis, Bemerkung, aktuell im Verkauf. Sobald eine neue Ware angelegt wird, erhält diese automatisch den Status im Verkauf.

Funktion: **Ware löschen**
Beschreibung: Der Admin kann eine Ware löschen.

Funktion: **Warendaten ändern**
Beschreibung: Ein Administrator kann die Warendaten einer Ware ändern.

Funktion: **Warenbewegung eingeben**
Beschreibung: Ein Anwender kann für jede Ware den Wareneingang oder Warenabgang festhalten, und dieser wird gespeichert.

Funktion: **Tagesabschluss erstellen**
Beschreibung: Es wird eine Tagesabrechnung erstellt. Aus dieser geht hervor, wie viele Waren verkauft worden sind und wie hoch die Einnahmen sind. Dabei werden Diebstahl usw. nicht berücksichtigt.

3.3 Nichtfunktionale Anforderungen

Nachfolgend werden Nichtfunktionale Anforderungen vorgestellt.

- Technische Anforderungen
  - Das System sollte erweiterbar sein (App, Desktop-Anwendung).
  - Es sollte nach Möglichkeit auf jeder Plattform lauffähig sein.
  - Die Anwendung soll Standortunabhängig sein.
  - Die Anwendung sollte von möglichst vielen unterschiedlichen Geräten (z.B. Smartphone, Destop-Pc usw.) bedienbar sein. (Responsive-Webdesign)
  - Tagesabschluss als PDF-Dokument.
- Sicherheit
  - Es soll JSON Web Token genutzt werden.
  - Passwort soll verschlüsselt in der Datenbank gespeichert werden.

- Ergonomische Anforderungen
  - Leichte Bedienbarkeit.
  - Übersichtliches Design.
  - Deutsche Textausgabe.

3.4 Zusammenfassung der Anforderungen

In diesem Kapitel sind nachfolgend alle Anforderungen nochmals kurz aufgezählt.

Nr.	Anforderung
1	Benutzer und Selbstbedienungsstand neu anlegen
2	Benutzername oder Passwort ändern
3	Benutzer löschen
4	Mehrbenutzer
5	Selbstbedienungsstand anlegen
6	Selbstbedienungsstand, Name ändern
7	Selbstbedienungsstand löschen
8	Selbstbedienungsstand, Benutzer hinzufügen
9	Selbstbedienungsstand, neuen Benutzer erstellen
10	Ware anlegen
11	Ware löschen
12	Warendaten ändern
13	Warenbewegung eingeben
14	Tagesabschluss erstellen
15	JSON Web Token
16	Verschlüsseltes Passwort in Datenbank speichern
17	Erweiterbares System
18	Plattformunabhängig
19	Standortunabhängig
20	Responsive-Webdesign
21	Tagesabschluss als PDF-Dokument
22	Leichte Bedienbarkeit
23	Übersichtliches Design
24	Deutsche Textausgabe

Tabelle 10: Zusammenfassung der Anforderungen

4 Konzeption

In diesem Kapitel werden einige Konzeptionsentwürfe dargestellt und erläutert. Zuerst wird der allgemeine Architekturentwurf präsentiert. Im Anschluss folgen die Entwürfe für Server, Datenbank, Client und Schnittstellen.

4.1 Allgemeiner Architekturentwurf

In diesem Unterkapitel wird der allgemeine Architekturentwurf erläutert.

Für das Projekt wird eine Client-Server-Architektur gewählt, da aus der Anforderungsanalyse hervorgeht, dass die Anwendung auf möglichst vielen Geräten standortunabhängig lauffähig sein soll. Außerdem soll in Zukunft die Möglichkeit bestehen, dass eine App eingebunden werden kann.

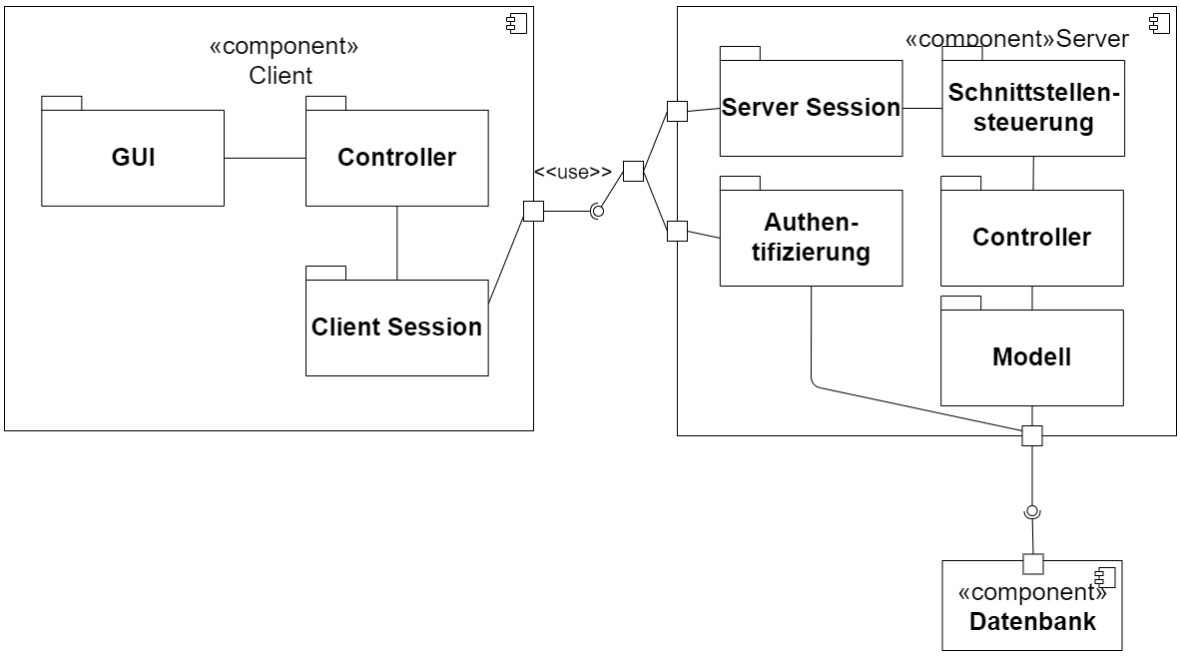

Abbildung 4: Komponentendiagramm

Die Abbildung 4 zeigt ein Komponentendiagramm, mit folgenden drei Komponenten Client, Server und Datenbank. Die Darstellung ist eine vereinfachte Abbildung, diese dient dazu einen ersten Überblick über das zu entwerfende System zu erhalten.

Der Client enthält drei Pakete GUI, Controller und Client Session.
GUI (Graphical User Interface) beinhaltet die Grafische Benutzeroberfläche.
Sobald ein Benutzer eine Eingabe tätigt, wird diese an den Controller gesendet. Der Controller ist dafür zuständig, die Eingaben zu prüfen, weiterzuleiten, zu ändern usw..

Sollte der Controller Daten vom Server benötigen, wird eine Session (Verbindung) zum Server aufgebaut, dafür ist das Paket Client Session zuständig.

Der Server beinhaltet fünf Pakete Server Session, Authentifizierung, Schnittstellensteuerung, Controller und Modell.
Sollte der Benutzer bislang noch kein Account angelegt haben oder die Zugriffsberechtigung (Token) ist abgelaufen, muss eine neue Authentifizierung durchgeführt werden. Hierfür ist das Paket Authentifizierung zuständig.
Ansonsten wird mithilfe einer Server Session eine Verbindung aufgebaut.
Das Paket Schnittstellensteuerung verwaltet eingehende Anfragen und leitet diese an den Controller weiter. Das Paket Controller steuert die eingehenden Anfragen. Sobald Anfragen Daten ändern, löschen, hinzufügen oder abfragen, sendet der Controller diese an das Paket Modell. Die Pakete Authentifizierung und Modell können eine Verbindung zur Datenbank aufbauen.

4.2 Entwurf der Datenbank

In diesem Kapitel werden die einzelnen Entwurfsschritte, die zum Erstellen der Datenbank notwendig sind, erläutert. Wie bereits in Kapitel 2.4 erwähnt, wird eine rationale Datenbank umgesetzt.

4.2.1 Normalisierung der Datenbank

Es liegt nach der Anforderungsanalyse folgende Ausgangssituation vor. Nachfolgende Daten sollen unter anderem gespeichert werden: Benutzername, Passwort, Benutzerrolle, Selbstbedienungsstandname, Datum, Warenname, Preis(€), Wareneingang, gezählte Waren, verkaufte Waren, Einnahmen(€).

Aus diesen Daten wird ein Entity-Relationship-Modell (ER-Modell) erstellt, dieses beinhaltet Entitäten, Attribute, Beziehungen, Primärschlüsseln, zusammengesetzte Primärschlüssel und Fremdschlüssel. Aus dem ER-Modell werden im Anschluss Tabellen erstellt.
Eine Entität (Rechteck) beschreibt ein Objekt, daraus wird später der Tabellenname. Die Attribute (Ellipsen) werden in Tabellenspalten überführt.
Ein Primärschlüssel (unterstreichendes Attribut) dient dazu, die einzelnen Datensätze auch Tupel genannt eindeutig zu identifizieren. Bei einem zusammengesetzten Primärschlüssel werden zwei oder mehr Attribute zur eindeutigen Erkennung eines Tupels verwendet. Der Fremdschlüssel verweist auf ein anderes Attribute, dieses muss ein Primärschlüssel sein. Dadurch können Tabellen miteinander in Beziehung (Raute) gebracht werden.

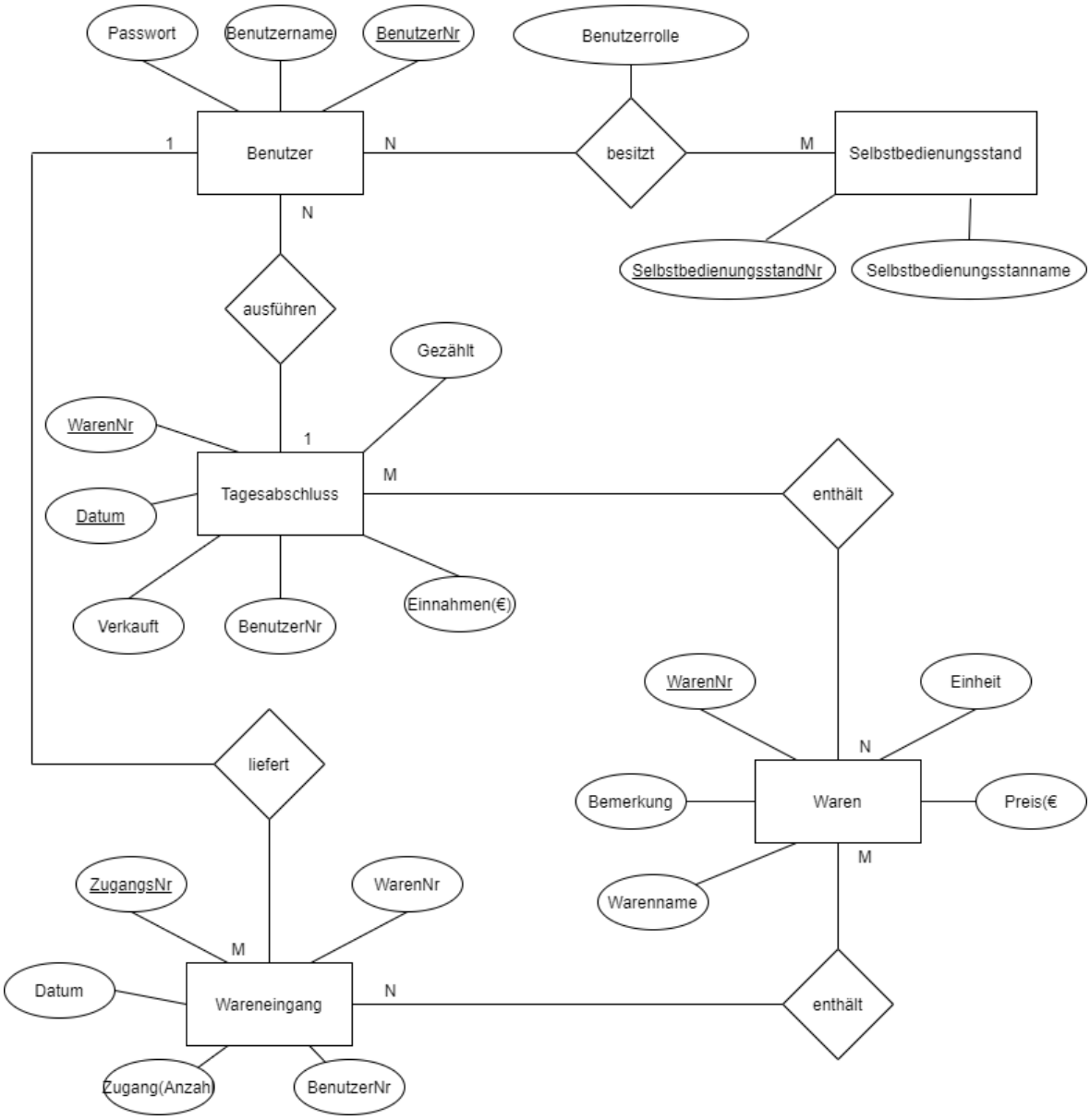

Abbildung 5: ER-Modell

In der Abbildung 5 ist das erstellte ER-Modell abgebildet. Daraus werden die Tabellen abgeleitet.
[DBR19] empfiehlt es, eine Datenbank so zu gestalten das „Redundanzen, Inkonsistenzen und Anomalien“ vermieden werden. Dieses soll mithilfe der Normalformen umgesetzt werden. Es existieren bis zu fünf Normalformen, in der Praxis werden oftmals nur die ersten drei Normalformen gebildet, erwähnt [DB218]. Es sollen nachfolgend drei aufeinander aufbauende Normalisierungschritte durchgeführt werden.
[rel17] zufolge sollten die folgenden Datenbankanforderungen eingehalten werden:

- Eindeutige Struktur, jeder Datensatz muss eindeutig zu identifizieren sein.
- Redundanzfrei, nach Möglichkeit sollten keine Daten mehrfach vorhanden sein.
- Keine Inkonsistenzen, keine Widersprüchlichkeit zwischen den Daten.

Die Voraussetzung für die erste Normalform ist, dass alle Tabellenspalten gleichartige Werte enthalten und alle Daten atomar sind. Das heißt, eine Tabellenspalte darf keine unterschiedlichen Typen enthalten, z.B. Betrag in Euro und Betrag in Cent. Außerdem müssen die Daten atomar sein, es ist nicht erlaubt mehrere Werte in einem Feld zu haben z.B. Straße, Postleitzahl, Ort.

Der Autor [DB218] schreibt, sobald eine Tabelle in der ersten Normalform und jedes Nichtschlüsselattribut vom Primärschlüssel funktional abhängig ist, befindet sich die Tabelle in der zweiten Normalform. Dieses trifft bereits zu, z. B. der Benutzername ist abhängig von der BenutzerNr..

Bei der Erstellung der Tabellen wird darauf geachtet das diese direkt in der 2. Normalform vorliegen.

BenutzerNr	Benutzername	Passwort
1	Christine	12345
2	ADE	asdfg
3	Hugo	yxcvb

Tabelle 11: Die Tabelle Benutzerdaten in der 2. Normalform.

SelbstbedienungsstandNr	Selbstbedienungsstandname
1	Eierhütte
2	Kartoffelhaus
3	Stand1

Tabelle 12: Die Tabelle Selbstbedienungsstand in der 2. Normalform.

SelbstbedienungsstandNr	BenutzerNr	Benutzerrolle
1	1	Admin
2	1	User
3	2	Admin
3	3	Admin

Tabelle 13: Die Tabelle Selbstbedienungsstand-Nutzer in der 2. Normalform.

Die Tabellen 11, 12 und 13 beinhalten die Benutzerdaten und Selbstbedienungsstanddaten. Es wird davon ausgegangen das ein Selbstbedienungsstand mehrere Benutzer haben kann, gleichzeitig kann ein Nutzer mehrere Selbstbedienungsstände betreiben, dieses wird mithilfe der Tabelle 13 umgesetzt.

WarenNr	Warenname	Einheit	Preis(€)	Bemerkung
1	Ei	Stück	0,20	
2	Kartoffeln	KG	5	Sorte Annabelle

Tabelle 14: Die Tabelle Waren in der 2. Normalform.

WareneingangsNr	Datum	Wareneingang(Anzahl)	WarenNr	BenutzerNr
1	25.07.2021	+100	2	1
2	25.07.2021	+80	1	1
3	26.07.2021	+60	2	1
4	26.07.2021	+5	2	2

Tabelle 15: Die Tabelle Wareneingang in der 2. Normalform.

Datum	WarenNr	BenutzerNr	Gezählt	Verkauft	Einnahmen(€)
25.07.2021	1	1	231	251	50,20
26.07.2021	1	1	52	239	47,80
26.07.2021	2	1	3	2	10

Tabelle 16: Die Tabelle Tagesabschluss in der 2. Normalform.

In dieser Tabelle 16 ist die Spalte BenutzerNr nachträglich hinzugefügt worden, dadurch lässt sich eindeutig erkennen wer den Tagesabschluss ausgeführt hat.

Die Tabelle 13 und Tabelle 14 befinden sich noch nicht in der dritten Normalform, die restlichen Tabellen befinden sich in der dritten Normalform, aus diesem Grund werden die Tabellen nachfolgend nicht mehr aufgeführt.

Eine Tabelle ist in der dritten Normalform, sobald diese sich in der zweiten Normalform befindet und "kein Nichtschlüsselattribut transitiv von einem Kandidatenschlüssel abhängt" [DB3]. Das heißt, indirekte Abhängigkeiten sollen vermieden werden.

SelbstbedienungsstandNr	BenutzerNr	Benutzerrolle
1	1	1
2	1	2
3	2	1
3	3	1

Tabelle 17: Die Tabelle Selbstbedienungsstand-Nutzer in der 3. Normalform.

RollenNr	Benutzerrollename
1	Admin
2	User

Tabelle 18: Die Tabelle Benutzerrollen-Nutzer in der 3. Normalform.

Die Tabelle 18 beinhaltet die aktuellen Benutzerrollen, aktuell die Rolle Admin und User, allerdings lassen sich leicht neue Rollen hinzufügen und bestehende Rollen verändern.

Im Anhang 7.2 sind alle Tabellen nochmals in der dritten Normalform aufgeführt.

4.3 Entwurf des Servers

Anhand eines Paketdiagramms wird der Entwurf des Servers erläutert. [Sera] beschreibt ein Paket, als eine Sammlung an inhaltlich zusammengehörigen Klassen. Klassen sind Vorlagen, „Sie sind Vorlagen, aus denen Objekte erzeugt werden. Objekte haben Eigenschaften und Methoden.“ schreibt [Serb].

Aus den zuvor erstellten Tabellen lassen sich bereits einige Pakete ableiten. Beispielweise das Paket, Repository, in diesem Paket werden alle Klasse gebündelt die für Datenbankenabfragen zuständig sind.

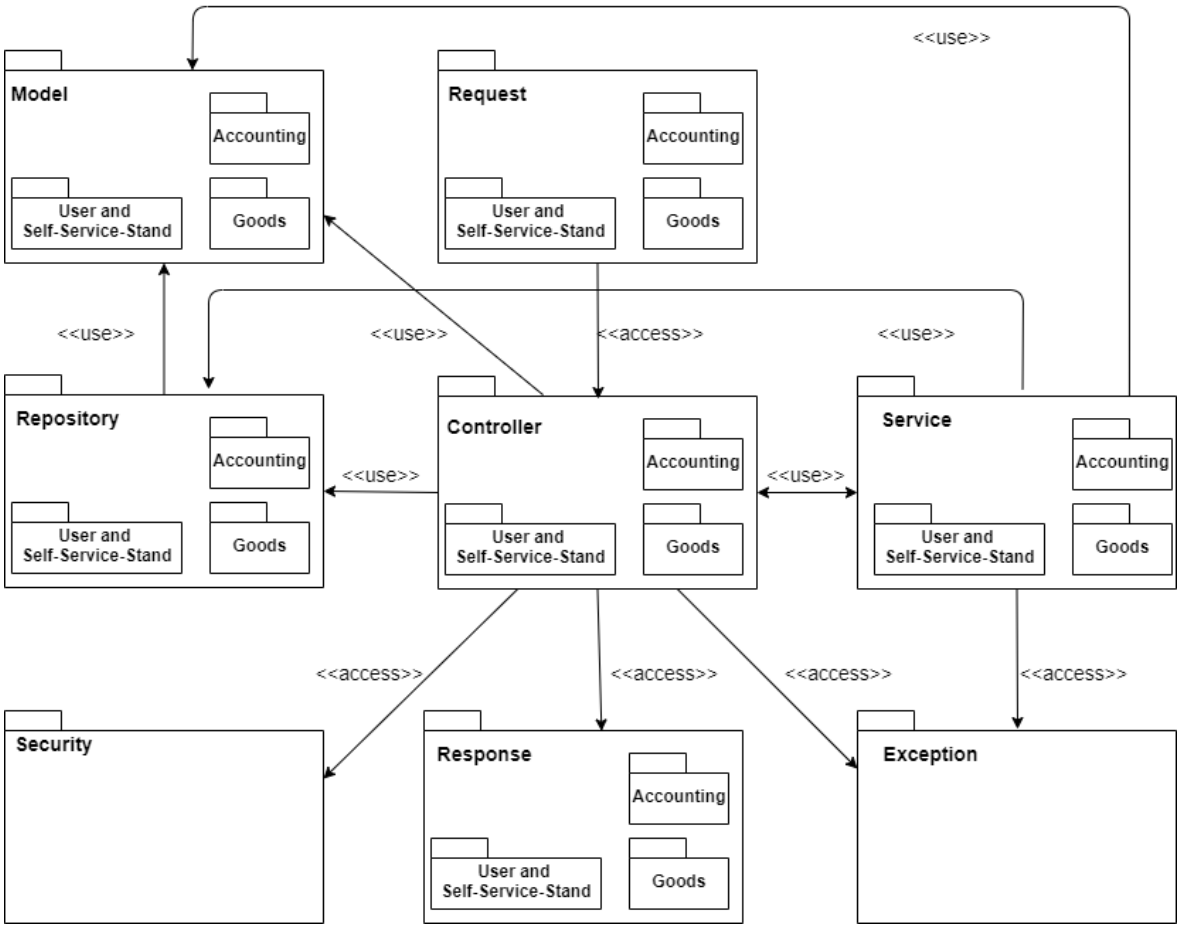

Abbildung 6: Paketdiagramm des Servers

In der Abbildung 6 ist das erstellte Paketdiagramm abgebildet, anhand des Paketdiagramms wird der Aufbau des Servers erläutert.

Das Paket Request (Anfrage) enthält Klassen, die die Struktur der Anfragen an eine Schnittstelle festlegen. Beispielhaft enthält die Klasse Goods im Unterpaket goods unter anderem folgende Variablen goodsName, goodsPrice usw.. Sobald eine Anfrage an eine Schnittstelle gesendet wird, muss die vorgebende Struktur eingehalten werden. Das heißt zum Beispiel, gleiche Namen und vorgebende Datentypen (int, String usw.).

Eines der wichtigsten Pakete ist das Controller Paket. Alle eingehenden Anfragen an die REST-API werden von einer Klasse im Paket Controller entgegengenommen. Der Controller entscheidet, was mit einer eingehenden Anfrage geschehen soll, zuerst prüft der Controller die eingehenden Daten auf Korrektheit. Anschließend steuert der Controller den Datenfluss und leitet die Daten weiter an die zuständigen Serviceklassen, Securityklassen oder Exeptionklassen.

Im Paket Response(Antwort) sind Klassen enthalten, die die Struktur der Ausgabennachrichten, die an einen Anfragenden zurückgesendet wird, festlegen.

Das Paket Service enthält Klassen und Methoden die die Anwendungslogik für den Controller bereitstellen. Im Unterpaket Good, werden z. B. Waren hinzufügen, geändert und gelöscht.

Sobald eine Exception (Ausnahme) ausgelöst wird, ist die zugehörige Klasse, die die Exceptions verwaltet, im Paket Exception zu finden sein. Exceptions werden von den Klassen in den Paketen Service und Controller ausgelöst. Eine Exception ist ein Programmfehler, der ohne Behandlung zu Abstürzen führen kann.

Verwaltet werden im Paket Repository Datenbankabfragen. Dieses kann z. B. eine Abfrage sein, ob eine bestimmte Ware bereits existiert oder die Suche nach einem Datensatz. Abfragen werden von Klassen aus den Paketen Service und Controller durchgeführt.

Im Paket Modell sind die Klassen vorhanden, die für die Datenbankmodellierung vonnöten sind. In diesen Klassen wird die Struktur der Datenbank abgebildet. Wie z. B. Primärschlüssel, Fremdschlüssel, Attribute und Datentypen.

Enthalten sind im Paket Security Klassen, die für die Sicherheit der Anwendung relevant sind.

Viele der dargestellten Pakete enthalten die drei Unterpakete Goods, User and Self-Service-Stand und Accounting in diesen sind die Klassen der entsprechenden Themen zusammengefasst.
So enthält das Unterpaket Goods, Klassen die für die Verwaltung der Waren zuständig sind.
Zusammengefasst werden die Benutzerklassen und Selbstbedienungsstandklassen, da sich einige Funktionen der Klassen in beiden Kategorien wiederfinden.
Zuletzt existiert noch das Unterpaket Accounting, hier finden sich Klassen, die für die Verwaltung, den Tagesabschluss usw. zuständig sind.

4.3.1 Entwurf der Rest-Schnittstellen

Nachfolgend sind alle REST-API 2.3.2 aufgeführt. Damit der HTTP-Client die Anfrage an die korrekte Schnittstelle senden kann, ist eine eindeutige Adressierung notwendig. Anfragen werden an eine URL gesendet, z. B. an http://localhost:8080/role/, zusammen mit der HTTP-Anfragemethode (GET, POST usw.). In der nachfolgenden Aufzählung, ist der Übersichtlichkeit halber zuerst die HTTP-Anfragemethode und anschließend der URL Pfad angeben z. B. POST /endOfDay/.
Die Datenübertragung via HTTP kann über den HttpBody oder über den HttpHeader erfolgen. Beispielhaft wird bei POST /goodsName/{goodsName} der Warenname über die URL übertragen.

Schnittstelle Tagesabschluss(EndOfTheDayController)
Der EndOfTheDayController stellt die folgenden Schnittstellen zur Verfügung:

- POST /endOfDay/
- PUT /endOfDay/
- GET /endOfDay/

Diese Schnittstellen dienen zur Verwaltung des Tagesabschlusses, für jede Ware kann pro Tag nur eine Endabrechnung erstellt werden.
Die Schnittstelle POST/endOfDay/ sorgt dafür das ein neuer Eintrag erstellt werden kann.
Mithilfe der Schnittstelle PUT/endOfDay/ kann dieser Eintrag editiert werden.
Die Schnittstelle GET/endOfDay/ übermittelt eine Auflistung, darin enthalten sind unter anderem die Angaben der theoretisch verkauften Waren und theoretische Einnahmen (ohne Diebstahl, beschädigte Ware usw.).

Schnittstelle Waren(GoodsController)
Der GoodsController stellt die folgenden Schnittstellen zur Verfügung:

- POST /goods/
- PUT /goods/
- DELETE /goods/
- GET /goods/
- GET /goods/{selfServiceStandName}
- GET /goods/{selfServiceStandName}/{goodsName}

Mithilfe dieser Schnittstellen lassen sich neue Ware anlegen (POST/goods/), verändern (PUT/goods/), oder auch löschen (DELETE/goods/). Eine Ware kann nur gelöscht

werden, wenn diese nicht genutzt wird. Die beiden Schnittstellen GET/goods/ geben eine Übersicht der Waren zurück, dieses kann eine Liste oder eine einzelne Ware sein.

Schnittstelle Warenname(GoodsNameController)
Der GoodsNameController stellt die folgenden Schnittstellen zur Verfügung:

- POST /goodsName/{goodsName}
- PUT /goodsName/
- DELETE /goodsName/
- GET /goodsName/

Die Schnittstellen der Klasse GoodsNameController enthalten Schnittstellen, die dafür verantwortlich sind, Warennamen hinzuzufügen (POST/goodsName/{goodsName}) zu ändern (PUT/goodsName/) und zu löschen (DELETE/goodsName/).
Außerdem kann eine Liste mit allen verfügbaren Namen ausgegeben werden. Jeder Warenname kann nur einmalig vergeben werden.
Der Warennamen wird in einer separaten Tabelle gespeichert, dieses hat den Vorteil, dass Redundanzen vermieden werden. Außerdem wird so verhindert, das unterschiedliche Schreibweisen für ein und dieselbe Ware existieren.

Schnittstelle Wareneingang(ReceiptController)
Der ReceiptController stellt die folgenden Schnittstellen zur Verfügung:

- POST /receipt/
- PUT /receipt/
- DELETE /receipt/{receiptGoodsNr}
- GET /receipt/one{receiptGoodsNr}
- GET /receipt/all/{selfServiceStandName}
- GET /receipt/variety/{selfServiceStandName}/{goodsName}

Mithilfe dieser Schnittstellen wird der Wareneingang und Warenausgang protokolliert. Mithilfe der Schnittstelle POST/receipt/ wird ein neuer Wareneingang oder Warenausgang hinzugefügt. PUT/receipt/ ändert diesen und DELETE/receipt/{receiptGoodsNr} löscht diesen Eintrag.
Die Schnittstelle GET/receipt/one/{receiptGoodsNr} übergibt einen einzelnen Datensatz. Eine Liste mit allen Eingängen und Ausgängen einer bestimmten Ware überliefert die Schnittstelle GET /receipt/variety/{selfServiceStandName}/{goodsName}.
Die Schnittstelle GET/receipt/all/{selfServiceStandName} liefert ebenfalls eine Liste, mit allen Ein-und Ausgängen.

Schnittstelle Benutzerrollen(RoleController)
Der RoleController stellt die folgenden Schnittstellen zur Verfügung:

- GET /role/

Die Schnittstelle GET/role/ gibt eine sortierte Liste mit allen verfügbaren Benutzerrollen zurück.

Schnittstelle Selbstbedingungsstand (SelfServiceStandController)
Der SelfServiceStandController stellt die folgenden Schnittstellen zur Verfügung:

- POST /selfServiceStand/{userName}/{selfServiceStandName}
- POST /selfServiceStand/addUser/
- DELETE /selfServiceStand/{removeUserName}/{userName}/{selfServiceStandName}
- GET /selfServiceStand/{selfServiceStandName}

Diese Schnittstellen helfen bei der Verwaltung eines Selbstbedienungsstandes. Ein neuer Selbstbedienungsstand wird mit der Schnittstelle POST/selfServiceStand/{userName}/{selfServiceStandName} erstellt, der Benutzer, der diesen erstellt hat, erhält Administratorrechte.
Die Schnittstelle POST /selfServiceStand/addUser/ fügt einen bereits existierenden Benutzer einen Selbstbedienungsstand hinzu, dabei kann die Benutzerrolle frei gewählt werden. Sobald ein Benutzer entfernt werden soll, hilft die Schnittstelle DELETE/selfServiceStand/{removeUserName}/{userName}/{selfServiceStandName}.
Eine Liste mit allen Benutzern, die dem Selbstbedienungsstand zugeordnet sind, wird mithilfe der Schnittstelle GET/selfServiceStand/{selfServiceStandName} ausgeben.

Schnittstelle Einheiten(UnitController)
Der UnitController stellt die folgenden Schnittstellen zur Verfügung:

- POST /unit/{unitName}
- PUT /unit/{oldUnitName}/{newUnitName}
- DELETE /unit/{unitName}
- GET /unit/{unitName}

Einheiten wie z. B. Stück KG, Sack usw. können mit der Schnittstelle POST/unit/ erstellt werden. Verändert werden können diese mit der Schnittstelle PUT /unit/{oldUnitName}/{newUnitName}. Soll eine Einheit gelöscht werden, ist dieses mit der Schnittstelle DELETE/unit/{unitName} möglich.

Allerdings können Änderung und Löschungen nur durchgeführt werden, wenn die Einheit noch nicht verwendet wird. Eine Liste der verfügbaren Einheiten gibt die Schnittstelle GET/unit/{unitName} aus.

Schnittstelle Benutzer(UserController)
Der UserController stellt die folgenden Schnittstellen zur Verfügung:

- POST /user/newAdmin
- POST /user/newUser
- PUT /user/
- GET /user/{username}

Die Benutzerverwaltung kann mithilfe nachfolgender Schnittstellen durchgeführt werden. Ein neuer Benutzer und ein neuer Selbstbedienungsstand werden mit der Schnittstelle POST/user/newAdmin erstellt. Der neu erstellte Benutzer wird automatisch dem erstellten Selbstbedienungsstand als Administrator zugeordnet. Der Name des Selbstbedienungsstandes und des Benutzers darf noch nicht vergeben sein.
Ein Administrator eines Selbstbedienungsstandes kann einen neuen Benutzer zu diesem hinzufügen, der Benutzer wird dabei neu erstellt, die Benutzerrolle kann frei gewählt werden, dieses geschieht mit der Schnittstelle POST/user/newUser.
Sobald ein neuer Benutzer erstellt wird, wird zuerst überprüft, ob der Benutzername bereits vergeben ist, sollte dieses der Fall sein, kann der Benutzer nicht erstellt werden. Die Schnittstelle PUT/user/ ist zum Editieren des Benutzers gedacht. Ein einzelner Benutzer kann mit der Schnittstelle GET/user/{username} ausgegeben werden.

4.4 Entwurf des Clients

Im nachfolgenden Kapitel wird genauer auf den Entwurf des Clients eingegangen. Das Kapitel wird unterteilt in Entwurf der grafischen Benutzeroberfläche und Entwurf des Hintergrundbereiches.

4.4.1 Entwurf der grafischen Benutzeroberfläche (Frontend)

Die Zielsetzung bei der Oberflächengestaltung dieser Anwendung ist, dass diese möglichst übersichtlich und schlicht gestaltet sein soll.
[gui20] nennt Tipps zur Gestaltung, unter anderem “Reduziere auf das Wesentliche“ es sollen nur die nötigsten und wichtigsten Informationen und Schaltflächen dargestellt werden.
Die Anwendung soll nur zwei Hauptbereiche umfassen, das sind der Anmeldebildschirm und das Hauptmenü. Auf diese wird nachfolgend kurz eingegangen.

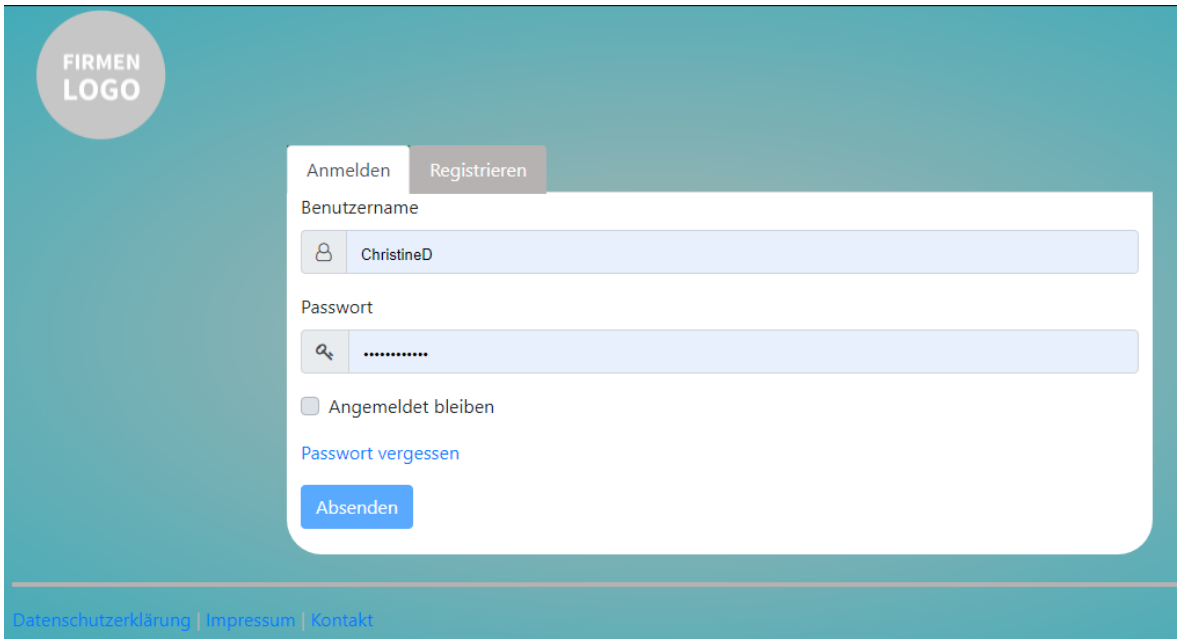

Abbildung 7: Anmeldebildschirm

Die 7 Abbildung zeigt den Anmeldebildschirm, dort können sich die Benutzer Anmelden und Registrieren.
Ein weiterer Tipp von [gui20] wird hier umgesetzt, dieser heißt „Orientiere dich an bereits erstellten Oberflächen“. Vielen Anwendern ist die Struktur des Anmeldebildschirmes bereits von anderen Anwendungen bekannt, dadurch fällt Ihnen die Bedienung leichter, da sie vieles wiedererkennen.
Sobald der JSON Web Token ungültig ist oder abgelaufen, wird der Nutzer immer auf den Anmeldebildschirm umgeleitet. Ein Token dient zur gegenseitigen Authentifizierung von Client und Server, erklärt [tok20]. Sobald ein Benutzer sich zum ersten mal anmeldet, wird eine Anfrage (Request) an den Server gestellt. Dieser sendet eine Antwort (Response) zurück, sind die Anmeldedaten gültig, wird zusätzlich der Token mitgesendet. Der Token wird bei jeder erneuten Kommunikation mit dem Server mitgesendet.

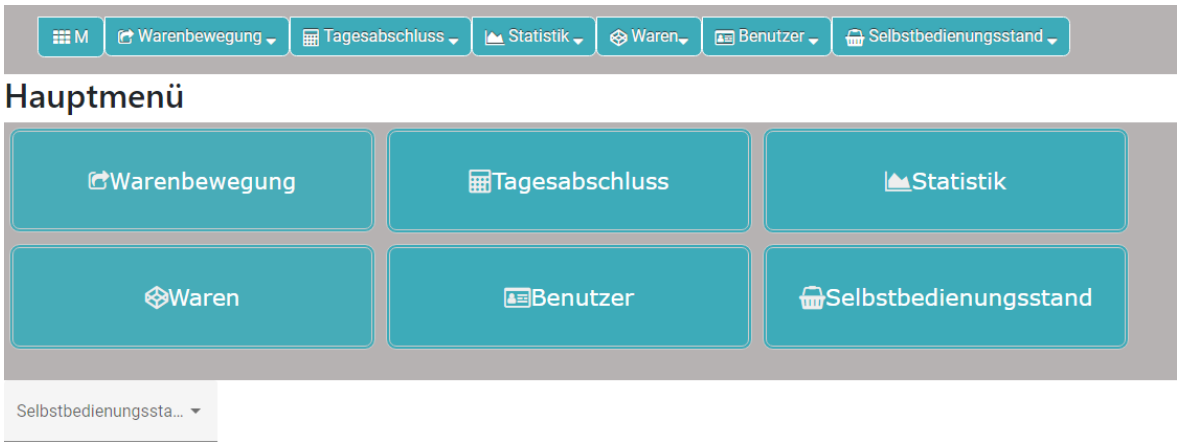

Abbildung 8: Hauptmenü

Die Abbildung 8 stellt das Hauptmenü da. Im oberen Bereich ist eine Navigationsleiste abgebildet, diese wird dauerhaft angezeigt. Der mittlere Bereich wird je nach Bedarf ausgetauscht. Im unteren Sektor kann der Anwender zwischen bestehenden Selbstbedienungsständen wechseln.

Sobald der Nutzer den Punkt Warenbewegung auswählt, kann er dort die Warenbewegungen (Eingänge und Ausgänge) und das Datum eintragen.
Ein Tagesabschluss wird erstellt, sobald der Anwender auf Tagesabschluss geht.
Unter dem Menüpunkt Statistik sind Monats- und Jahresstatistiken zu finden.
Mithilfe der Menüpunkte Waren, Benutzer und Selbstbedienungsstand können Waren, Benutzer und der Selbstbedienungsstand geändert, gelöscht oder hinzugefügt werden.

4.4.2 Entwurf des Hintergrundbereiches (Backend)

Die Ausgangsidee ist, die angefertigt Skizze des Hauptmenüs und des Anmeldebildschirmes, aus dem vorigen Kapitel, in einzelne Bestandteile bzw. Komponenten zu zerlegen. Die Abbildung 9 stellt dieses dar.

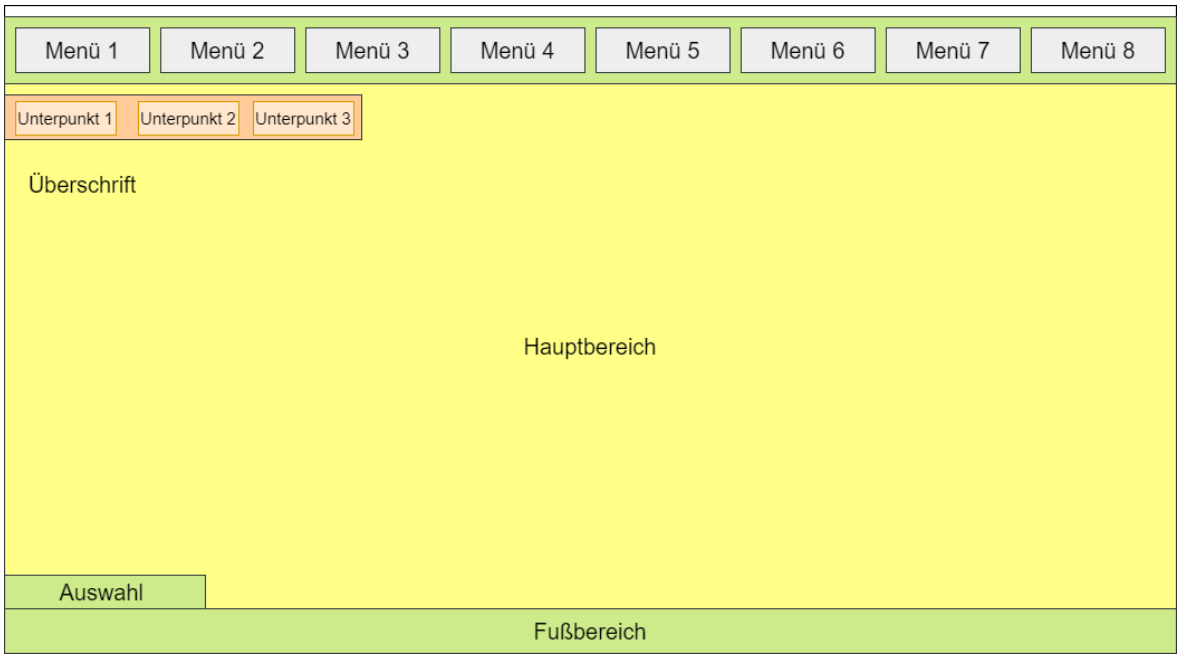

Abbildung 9: Darstellung der Komponenten des Clients

Zu sehen ist der Hauptbereich 9 unterteilt in einzelne Komponenten die jeweils farbig dargestellt sind. Die grün dargestellten Komponenten werden dauerhaft eingeblendet, dieses sind der Fußbereich, Hauptmenü und Auswahl des Selbstbedienungsstandes.

Je nachdem in welchem Menü der Anwender sich befindet, wird im Hauptbereich (Gelb) situationsabhängig unterschiedliche Unterpunkte (Orange) angezeigt. Je nachdem, was der Benutzer davon auswählt, wird dieses im Hauptbereich eingeblendet.

Dadurch ergibt sich folgende Struktur.

- Hauptmenü
- Hauptbereich
    - Tagesabschluss
        + Eintrag bearbeiten oder anlegen
        + Übersicht Tagesabschluss
    - Warenverwaltung
        + Ware anlegen oder ändern
        + Übersicht Warenverwaltung
    - Warenbewegung
        + Warenbewegung hinzufügen
        + Übersicht Warenverwaltung
        + Warenoptionen
    - Benutzerverwaltung
        + Benutzerdaten bearbeiten
        + Übersicht Warenverwaltung
    - Selbstbedienungsstandverwaltung
        + Selbstbedienungsstanddaten bearbeiten
        + Benutzer zum Selbstbedienungsstand hinzufügen
        + Benutzer anlegen und Selbstbedienungsstand hinzufügen
    - Statistik
        + Monatsstatistik anzeigen
        + Jahresstatistik anzeigen
- Fußbereich
- Auswahl des Selbstbedienungsstandes

Zusätzlich werden unterschiedliche Serviceklassen angelegt, diese beinhalten Inhalte die von mehren unterschiedlichen Komponenten genutzt werden.

5 Realisierung und Implementierung

Diese Kapitel thematisiert die Realisierung und Implementierung. Zunächst wird die Implementierung der SQL-Datenbank vorgestellt, danach folgt die Implementierung des Servers und des Clients. Es werden jeweils ein paar Quellcodeausschnitte näher beleuchtet.

5.1 Implementierung der SQL Datenbank

Zuerst wird XAMPP[4] heruntergeladen, installiert und eingerichtet. [DB521] schreib das XAMPP nur eine lokale Testumgebung simuliert, XAMPP sollte mit der Grundkonfiguration nicht als Webserver im Internet eingesetzt werden, da es Sicherheitslücken aufweist. XAMPP bündelt unterschiedliche freie Software, darunter auch einen lokalen Webserver mit der Datenbank MariaDB und phpMyAdmin.
Im Anschluss wird eine Datenbank für das Projekt mit dem Namen db_SB_V1 angelegt.

Anschließend umgesetzt und realisiert werden die entworfenen Tabellen aus dem Kapitel 4.2.1. Im Zug des Implementierungsprozesses, werden die Tabellen- und Spaltennamen ins Englische übertragen. Folgendes Schema zur Namensgebung wird verwendet.

- Datenbankname: db_Datenbankname
- Tabellenname: tb_Tabellenname
- Spaltenname: cl_Spaltenname

Beispielhaft wird nachfolgend die Tabelle 27 implementiert.

```
CREATE TABLE `tb_end of-the-day` (
  `cl_graduation-date` date NOT NULL,
  `cl_goods-nr` int(10) UNSIGNED NOT NULL,
  `cl_user-nr` int(10) UNSIGNED NOT NULL,
  `cl_graduation-count` int(10) UNSIGNED NOT NULL,
  `cl_graduation-sold` int(10) UNSIGNED NOT NULL,
  `cl_graduation-revenue` decimal(6,2) UNSIGNED NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Listing 2: Erstellung der Tabelle tb_end of-the-day mit Hilfe des SQL-Behelfes CREATE

In Listing 2 ist zu sehen wie mithilfe des Befehls CREATE eine neue Tabelle mit den Namen tb_end of-the-day angelegt wird.
Folgende Datentypen werden eingesetzt, int (Zahl, von -2.147.483.648 bis 2.147.483.647), VARCHAR (Zeichenkette, bis zu 65.535 Zeichen), decimal (Vorkommastelle Anzahl,

[4]https://www.apachefriends.org/de/index.html

Nachkommastelle Anzahl)und DATE (Datum, im Format YYYY-MM-DD).
Durch die Option NOT NULL wird sichergestellt, dass bei Erstellung eines neuen Tupels das Feld einen Wert hat. Die Option UNSIGNED sorgt dafür, das kein Vorzeichen (+,-) eingeben werden kann, diese hat den Vorteil das nur positive Zeichen eingeben werden können.

```
1  ALTER TABLE `tb_end of-the-day`
2  ADD PRIMARY KEY (`cl_graduation-date`,`cl_goods-nr`),
3  ADD KEY `cl_goods-nr` (`cl_goods-nr`),
4  ADD KEY `cl_user-nr` (`cl_user-nr`);
5
6
7  ALTER TABLE `tb_end of-the-day`
8  ADD CONSTRAINT `tb_end of-the-day_ibfk_1`
9  FOREIGN KEY (`cl_goods-nr`) REFERENCES `tb_goods` (`cl_goods-nr`),
10 ADD CONSTRAINT `tb_end of-the-day_ibfk_2`
11 FOREIGN KEY (`cl_user-nr`) REFERENCES `tb_user` (`cl_user-nr`);
12
```

Listing 3: Veränderung der Tabelle tb_receiptgoods mit Hilfe des SQL-Behelfes ALTER

Mit der Anweisung ALTER TABLE können nachträglich unter anderem Optionen hinzugefügt werden, zu sehen ist dieses in Listing 3. Es wird ein zusammengesetzter Primärschlüssel festgelegt mit der Option ADD PRIMARY KEY, die beiden Spalten cl_graduation-date und cl_goods-nr bilden das Schlüsselpaar.
Anschließend wird die Anweisung ALTER TABLE erneut durchgeführt, es wird ein Fremdschlüssel (FOREIGN KEY) für die Spalten cl_goods-nr und cl_user-nr generiert. In diesem Beispiel, werden die Spalte cl_goods-nr mit einem Datensatz in der Tabelle tb_goods, Spalte cl_goods-nr verknüpft.

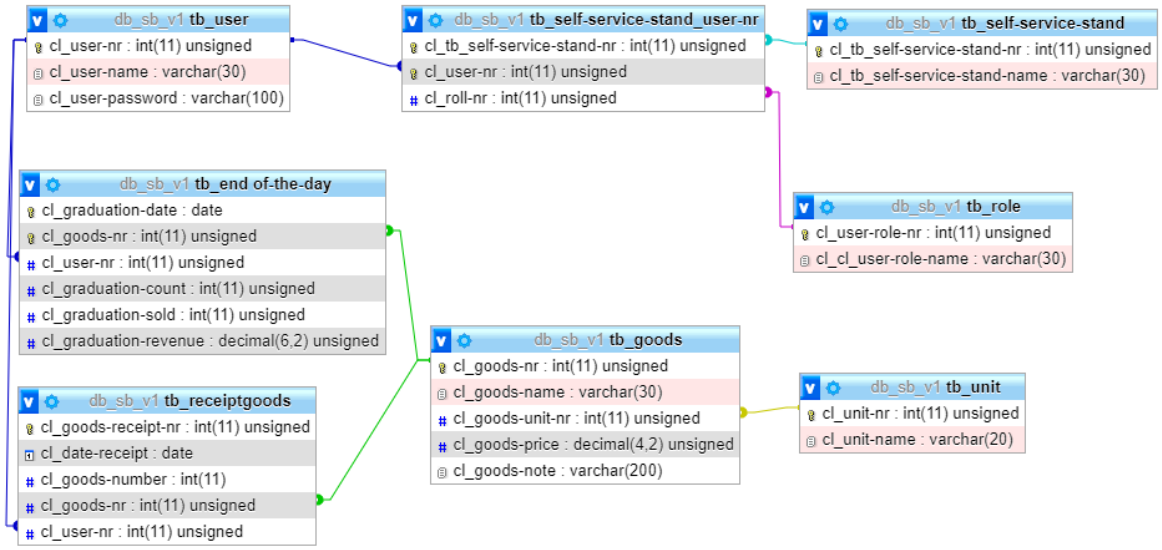

Abbildung 10: Übersicht, erstellte Tabellen

Die Abbildung 10 stellt die erstellten Tabellen grafisch dar. Aus der Abbildung lassen sich die einzelnen Tabellen, mit den Spalten und den verwendeten Datentypen ablesen. Außerdem können die Primärschlüssel, erkennbar am gelben Schlüssel, leicht identifiziert werden. Die genutzten Fremdschlüssel lassen sich ebenfalls erkennen, dargestellt durch die farbigen Verbindungslinien.

5.1.1 MariaDB und phpMyAdmin

MariaDB ist ein bekanntestes Datenbankmanagementsystem (DBMS), dieses wird in diesem Projekt genutzt. In diesem Kapitel wird erläutert, warum dieses System verwendet wird.

MariaDB[5] wurde im Jahr 2009 veröffentlicht. Das DBMS unterliegt einer GPL2 Open-Source Lizenz, das heißt sowohl private als auch kommerzielle Nutzung ist erlaubt. Entwickelt wurde MariaDB von Michael Widenius, er entwickelte ebenfalls MySQL, aus diesem Grund gibt es sehr viele Gemeinsamkeiten.
MariaDB hat eine gute Geschwindigkeit und Performance, bei der Durchführung von Datenbankabfragen. Die Seite [DB4] schreibt “Features wie Hochverfügbarkeit, Security, Interoperabilität und Performanceverbesserungen aufweist.“ bietet MariaDB.
Damit die Verwaltung der Datenbank komfortabler ist, wird die Software phpMyAdmin[6] verwendet. PhpMyAdmin ist weitverbreitet und die Standartsoftware zum Verwalten von Datenbanken.

[5]https://mariadb.com
[6]https://www.phpmyadmin.net

5.2 Implementierung des Spring Boot Servers

Für die Entwicklung des Servers wurde Spring Boot gewählt. Spring Boot ist eine Open Source Software und kann kostenlos genutzt werden. Außerdem existiert eine große Community, die bei Fragen weiterhelfen kann. Spring Boot bietet außerdem einen großen Funktionsumfang. Es lassen sich leicht neue Funktionen usw. integrieren. Allerdings erfordert Spring Boot etwas Einarbeitungszeit.

Nachfolgend werden einige Quellcodeausschnitte etwas genauer erläutert. Der komplette Quellcode der Klasse ist im Anhang 7.2 zu finden. Zuerst werden zwei Klassen aus dem Paket model dargestellt, anschließend werden aus den Paketen controller, service und repoitory ebenfalls Klassen vorgestellt.

Um die objektrelationale Abbildung der Datenbank zu erleichtern, wird JPA(Java Persistence API) verwendet, JPA ist im Spring Boot Framework enthalten. [JPA20] schreibt "Die Java Persistence API (JPA) des Java Community Process ist ein Standard für das Objekt-relationale Mapping (ORM) von Java-Objekten. ORM ist ein Verfahren zur Speicherung von Objekten in Datenbanken - Klassen und Objekte werden auf Tabellen und Zeilen abgebildet.".

5.2.1 Beispielcodeausschnitte aus Model-Klassen

Die Klasse EndOfTheDay 16 befindet sich im Paket model, die Klasse bildet die Datenbankstruktur der Tabelle tb_end of_the_day ab. Die Datenbanktabelle enthält unter anderem einen zusammengesetzten Primärschlüssel, sowie drei Fremdschlüssel.

```
1  @Table(name = "tb_end of_the_day", indexes = {
2    @Index(name = "cl_user_nr", columnList = "cl_user_nr"),
3    @Index(name = "cl_goods_nr", columnList = "cl_goods_nr"),
4    @Index(name = "cl_self_service_stand_nr", columnList = "
       cl_self_service_stand_nr")
5  })
6  @Setter
7  @Getter
8  @Entity
9  public class EndOfTheDay {
10
11   /**
12   * ID End of the Day.
13   */
14   @EmbeddedId
15   private EndOfTheDayId id;
16
17   /**
```

```
18   * Refers to a user ID.
19   */
20   @ManyToOne
21   @JoinColumn(name = "cl_user_nr", nullable = false)
22   private User userNr;
```

Listing 4: Das Listing zeigt einen Ausschnitt aus der Klasse EndOfTheDay

Im Codeausschnitt 4 Zeile 1, ist die Zuordnung einer bereits bestehenden Tabelle in einer Datenbank mithilfe der JPA-Annotationen @Table zu sehen. Anschließend wird in den Zeilen 2 bis 5 die Fremdschlüsselverknüpfung durchgeführt, dabei wird auf eine Spalte in einer anderen Tabelle verwiesen.

Die Annotationen @Setter und @Getter in Zeile 6 und 7 sorgen dafür, das die Getter und Setter Methoden nicht mehr implementiert werden müssen. Sondern mithilfe des Java-Framework Lombok automatisch intrigiert werden.

Die JPA-Annotationen @Entity in der 8 Zeile bewirkt das automatisch alle Attribute in der Klasse auf gleichnamige Datenbankspalten (Mapping) abgebildet werden. Es kann der Fall auftreten, dass die Attribute und Spaltennamen unterschiedlich heißen, wie z. B. in Zeile 22 Mit der Annotation @JoinColumn(name = “cl_user_nr“) in Zeile 21 wird der Tabellenspaltname zugewiesen.

Da es sich bei dem Attribute User um eine Fremdschlüsselverknüpfung handelt, wird in Zeile 20 eine 1:n Beziehung hinzugefügt, es wird ein User-Objekt deklariert.

Bei der ID handelt es sich um einen zusammengesetzten Primärschlüssel, siehe Zeile 13 und 19. Es wird ein EndOfTheDayId-Objekt erstellt. Im Listing 5 ein Codeausschnitt der Klasse EndOfTheDayId abgebildet. In Zeile 8 und 14 sind die Attribute zu sehen aus denen sich der zusammengesetzte Primärschlüssel definiert. Die Annotationen in Zeile 1 deutet darauf hin, dass in dieser Klasse ein zusammengesetzter Primärschlüssel vorhanden ist.

```
1  @Embeddable
2  public class EndOfTheDayId implements Serializable {
3
4    /**
5    * serialVersionUID
6    */
7    private static final long serialVersionUID = 4548732131340178144L;
8
9    /**
10   * End of the day date.
```

```
11   */
12   @Column(name = "cl_graduation_date", nullable = false)
13   private LocalDate graduationDate;
14
15   /**
16   * The commodity number.
17   */
18   @Column(name = "cl_goods_nr", nullable = false)
19   private Integer goodsNr;
```

Listing 5: Das Listing zeigt einen Ausschnitt aus der Klasse EndOfTheDayId

5.2.2 Beispielcodeausschnitte aus Repository-Interfaces

Im Paket repository sind Interfaces enthalten, die für Datenbankoperationen zuständig sind. Es handelt sich nicht um Standard JPA, sondern es wird Spring Data JPA verwendet. Genutzt wird die Kurzschreibweise für Abfragen.

```
1  @Repository
2  public interface GoodsNameRepository extends JpaRepository<GoodsName,
       Integer> {
3
4    //Query list of all goods names, sorted by unit name.
5    List<GoodsName> findByOrderByGoodsNameAsc();
6
7    //Query, goods name with the name exists.
8    boolean existsByGoodsNameIs(String goodsName);
9
10 }
```

Listing 6: Das Listing zeigt einen Ausschnitt aus dem Interface GoodsNameRepository

Im Listing 6 ist das Repository des Interfaces GoodsNameRepository abbildet. Das Interface erbt vom Interface JpaRepository Zeile 2.
In der Zeile 5 ist eine Datenbankabfrage zu sehen die eine sortierte Liste mit allen vorhanden Einträgen in der Tabelle zurückliefert.
Eine weitere Abfrage ist in Zeile 8, diese Abfrage liefert ein True zurück, sobald ein Eintrag mit der eingebenden Zeichenkette übereinstimmt.

```
1  List<ReceiptGoods> findByDateReceiptAndGoodsNr_GoodsId(LocalDate
       dateReceipt, Integer goodsId);
```

Listing 7: Das Listing zeigt einen Ausschnitt aus dem Interface ReceiptGoodRepository

Das Listing 7 liefert ein Beispiel für eine Verknüpfte Und-Abfrage. Außerdem wird in der Tabelle nach einem Fremdschlüsseltribut (GoodsNr_GoodsId) gesucht.

5.2.3 Beispielcodeausschnitte aus einer Controller-Klasse

```
1  @RestController
2  @RequestMapping("/unit")
3  public class UnitController {
4
5    /**
6    * unitRepository to handle unit information.
7    */
8    private final UnitRepository unitRepository;
9
10   /**
11   * goodsRepository to handle goods information.
12   */
13   private final GoodsRepository goodsRepository;
14
15   /**
16   * Service to handle self-service stand information logic.
17   */
18   private final UnitService unitService;
19
20   public UnitController(UnitRepository unitRepository, GoodsRepository
       goodsRepository, UnitService unitService) {
21     this.unitRepository = unitRepository;
22     this.goodsRepository = goodsRepository;
23     this.unitService = unitService;
24   }
```

Listing 8: Das Listing zeigt einen Ausschnitt aus der Klasse unitController

In Zeile 1 wird festgelegt, dass es sich bei der Klasse um einen Rest-Controller handelt. Mapping wird in der 2. Zeile durchgeführt.[map17] beschreibt, das es beim Mapping „um die Konversion von zwei verschiedenen Adressräumen.„ geht.„Dabei werden die Daten eines Adressraums auf einen zweiten Adressraum abgebildet.„
Alle Klassen mit der Annotation @Repository, @Controller, @Service oder @Component werden automatisch als Bean erkannt und registriert. In der Spring Dokumentation [spr21] steht das ein Bean, ein Objekt ist, das von einem Spring IoC-Container instanziiert, assembliert und verwaltet wird.

Es wird das Endwurstmuster Dependency Injektion verwendet. Dieses besagt, das Komponenten und Klassen soweit möglich unabhängig von anderen Klassen sein sollen, bekannt auch als lose Kopplung, erklärt [Dep21]. “Als Dependency Injection bezeichnet man in der objektorientierten Programmierung eine externe Instanz, die die Abhängigkeiten von Objekten im Vorhinein regelt “ [Spr19a].
In diesem Projekt wird mithilfe von Constructor-Injection (Konstruktor-Injektion) das

Objekt injiziert. Dieses ist in den Zeilen 8 bis 24 zu erkennen, es werden sowohl Repositoryklassen und eine Serviceklasse eingebunden, Zeilen 8 bis 18. Diese werden im Konstruktor injizierten, Zeile 20 bis 24.

```
1  /**
2  * Method returns a sorted list with all units.
3  *
4  * @return json list with all units
5  */
6  @GetMapping(path = "/", produces = MediaType.APPLICATION_JSON_VALUE)
7      private ResponseEntity<List<Unit>> getUnits() {
8
9    //Return sorted list, sorted by unit name.
10   return new ResponseEntity<>(unitService.getUnitsList(), HttpStatus.
       OK);
11 }
```

Listing 9: Das Listing zeigt eine Methode aus der Klasse unitController

Im Listing 9 ist eine Methode aus der Klasse UnitController abgebildet. Diese Methode wird durch die REST-Schnittstelle GET /unit angesprochen (8). Der Pfad /unit wurde bereits am Klassenanfang festgelegt 8 Zeile 1. Da in dieser Klasse nur eine Get-Schnittstelle angesprochen wird, kann diese eindeutig infiziert werden, eine weitere Pfad Angabe ist nicht nötig.
Die Methode gibt eine Json-Liste und einen HttpStatus zurück. Diese ist in Zeile 10 implementiert. Außerdem ist zu sehen, wie eine Service Bean aufgerufen wird, (unitService.getUnitsList()). Dadurch wird die entsprechende Methode in der Serviceklasse ausgelöst.

5.3 Implementierung des Angular-Clients

Diese Kapitel erläuterte die Implementierung des Angular-Clients. Zuerst wird auf die Implementierung der grafischen Benutzeroberfläche eingegangen. Danach folgt die Implementierung des Hintergrundbereiches.

5.3.1 Implementierung der grafischen Benutzeroberfläche (Frontend)

In diesem Kapitel wird auf die Implementierung der grafischen Benutzeroberfläche eingegangen, nachfolgend werden einige Quellcodeausschnitte gezeigt. Zuerst wird kurz auf das Grid Layout eingegangen.

Eines der Hauptmerkmale von Angular ist, das dieses komponentenbasiert arbeitet. Da bereits bei der Konzeption die Oberflächen in Komponenten eingeteilt worden sind, bietet es sich an, Angular zu verwenden.

Jede dieser Komponenten enthält eine CSS-Datei, HTML-Datei und eine Typscript-Datei. In diesem Kapitel wird auf die CSS-Datei und HTML-Datei eingegangen.

Das Gestaltungsraster wird mithilfe CSS Grid Layout gestaltet. Mithilfe von Grid lässt sich eine Seite in einzelne Raster aufteilen, die Flächen müssen nicht die gleiche Größe haben, sondern nur rechteckig sein. Die Komponenten werden den Rasterzellen zugewiesen.

```
1  .container {
2    display: grid;
3    grid-template-columns: 0.5fr 2fr 8fr 2fr 0.5fr;
4    grid-template-rows: 0.4fr 3.4fr 0.4fr 0.4fr;
5    gap: 0px 0px;
6    grid-auto-flow: row;
7    grid-template-areas:
8    "navbar navbar navbar navbar navbar"
9    "main main main main main"
10   "selectSelf selectSelf . . ."
11   "footer footer footer footer footer";
12 }
13
14 .navbar {
15   grid-area: navbar;
16   background: var(--colorGray);
17 }
18
```

Listing 10: Umsetzung des Grid-Layout

Im Listing 10 ist ein Ausschnitt aus einer CSS-Datei zusehen. In dieser, wird die Vorlage aus dem Kapitel 4 Entwurf der grafischen Benutzeroberfläche 8 mit Grid umgesetzt.
In den Zeilen 3 und 4 ist zu sehen, wie Größe und Anzahl der Spalten und Zeilen festgelegt wird, es gibt fünf Spalten und vier Zeilen. In Zeile 7 bis 11 werden den Rasterbereichen Namen zugewiesen, der Bereich navbar (Hauptmenü) ist beispielsweise in Zeile eines, Spalte zwei bis vier zu finden.
Die Zeilen 14 bis 17 zeigen, wie einer CSS-Klasse der Grid-Bereich navbar zugewiesen wird. In Zeile 16 wird die Farbe des Hintergrundbereiches festgelegt, die Farbe ist in einer Variabel gespeichert. Dieses hat den Vorteil, dass die festgelegt Farbe schnell und leicht änderbar ist.

```
<div class="container">
  <div class="navbar">
    <app-navbar></app-navbar>
  </div>
  <div class="main">
    <router-outlet></router-outlet>
  </div>
  <div class="footer">
    <app-footer></app-footer>
  </div>
```

Listing 11: Zuweisung der Komponenten in HTML

Im Listing 10 ist ein Teilausschnitt einer HTML-Datei abgebildet. Das Listing 11 veranschaulicht die Zuweisung der CSS-Klassen z. B. in Zeile 2.
Außerdem ist zu sehen wie erstellte Angular-Komponenten eingebunden werden, Zeile 3 und 9.
In Zeile 6 wird das Router-Outlet im Hauptbereich implementiert. Angular sorgt automatisch dafür dass die richtige Komponente eingefügt wird. Mithilfe von vordefinierten internen Routinezuweisungen wird die passende Komponente ausgewählt. Mehr dazu im nachfolgenden Kapitel.

```
<tr (click)="setValues(goods.goodsId)" *ngFor="let goods of
    searchResultList">
  <td>{{goods.goodsId}}</td>
  <td>{{goods.goodsName}}</td>
  <td>{{goods.unitName}}</td>
  <td>{{goods.goodsPrice}}</td>
  <td>{{goods.goodsNote}}</td>
  <div *ngIf=goods.goodsActive>
    <td>Ja</td>
  </div>
  <div *ngIf=!goods.goodsActive>
    <td>Nein</td>
  </div>
</tr>
```

Listing 12: Schleifen und Abfragen in HTML

Im Listing 12 ist eine forEach-Schleife und zwei IF-Abfragen zu sehen. Die forEach-Schleife befindet sich in der Zeile 1,(*ngFor=“let goods of searchResultList) es wird eine Liste (searchResultList), diese wurde in der dazugehörigen TypeScript-Datei initialisiert, durchlaufen. Der nachfolgende HTML-Code wird dabei so oft ausgeführt wie es der Länge der Liste entspricht. Für jeden Schleifendurchlauf wird ein Element in die Variabel goods gespeichert. Diese wird nachfolgende ausgeben.

In Zeile 7 wird eine If-Abfrage durchgeführt, dabei wird überprüft, ob goods.goodsActive mit dem Wert True initialisiert worden ist. Ist dieses der Fall werden alle Elemente im nachfolgenden Div-Element ausgeführt.

5.3.2 Implementierung des Hintergrundbereichs (Backend)

Es werden in diesem Kapitel einige TypeScript-Quellcodeausschnitte gezeigt.

```
1  const myRoutes: Routes = [
2  {
3    path: '',
4    component: MainScreenComponent,
5    children: [
6    {
7      path: 'goods',
8      component: GoodsComponent,
9      children: [
10     {
11       path: 'overview',
12       component: OverviewGoodsComponent
13     },
14     {
15       path: 'new',
16       component: NewGoodsComponent
17     },
18     {
19       path: 'change',
20       component: ChangeGoodsComponent
21     }, {
22       path: 'settings',
23       component: SettingsGoodsComponent
24     }
25     ]
26   },
```

Listing 13: Festlegung der internen Routen in Angular

Im ersten Listing 13 dieses Kapitels ist zu sehen, wie die internen Routen in der Konstante myRoutes gespeichert werden, Zeile 1. Mithilfe von internen Routen ist es möglich innerhalb von Angular zu navigieren oder die Ansichten zu ändern. Dieses wird im Quellcode Listing 12 Zeile 6 genutzt.
Im Listing 14 ist HTML-Code abgebildet, in der Zeile 4 ist ein interner Link zu sehen, dieser leitet weiter zum Pfad /goods/overview. Es können Eltern-Pfad-Routen und deren Kinderelemente festgelegt werden, dargestellt im Listing 13 Zeile 5 bis 26.
Dass RouterModule welches importiert wird, übernimmt das Routing für den Entwickler.

```
1
2 <div class="btn btn-default link-menu">
3   <i class="fa fa-id-card-o"></i>
4   <a routerLink="/goods/overview">Waren</a>
5 </div>
6
```

Listing 14: Festlegung der internen Routen in Angular

Folgender Quellcode sogt dafür, dass eine Get-Anfrage an den Server gesendet wird.

```
1
2   private loadGoods() {
3
4   //set url
5   let url = 'goods/' + localStorage.getItem('key_selfServiceStand_name
     ');
6
7   //communication Server
8   this.communication
9   .serverCommunication(url, '', 'get', true)
10  .subscribe((response: any) => {
11
12    //save entry in list
13    for (var entry of response) {
14      this.goodsList.push({
15        goodsId: entry.goodsId,
16        goodsName: entry.goodsNameNr.goodsName,
17        unitName: entry.goodsUnitNr.unitName,
18        goodsPrice: entry.goodsPrice,
19        goodsNote: entry.goodsNote,
20      });
21    }
22
23  }, error => {
24    console.log(error);
25  }
26  );
27 }
```

Listing 15: Get-Anfrage an den Server

Im Listing 15 ist zu sehen wie eine Anfrage an den Server gesendet wird, es wird eine Liste mit allen verfügbaren Waren angefordert. Zuerst wird ein Teil der URL in der Variabel url gespeichert. Anschließend wird eine Verbindung zum Server aufgebaut (Zeile 8 bis 10), die restlichen erforderlichen Informationen sind in der Service Klasse server-communication.service hinterlegt, unter anderem der Token, URL usw..

Bei erfolgreicher Übermittlung werden die anforderten Daten in der mehrdimensionalen Liste goodsList gespeichert (Zeilen 13 bis 21). Ansonsten wird eine Fehlermeldung ausgeben (Zeile 24).

6 Testen der Anwendung

Zum Abschluss folgt eine Tabelle, aus dieser ist zu entnehmen, welche der definierten Anforderungen aus Kapitel 3 erfüllt oder teilweise erfüllt sind.

Nr.	Anforderung	Erfüllt	Teilweise erfüllt	Nicht erfüllt	Unerfüllbar
1	Benutzer und Selbstbedienungsstand neu anlegen	X			
2	Benutzername oder Passwort ändern		X*		
3	Benutzer löschen		X*		
4	Mehrbenutzer		X*		
5	Selbstbedienungsstand anlegen		X*		
6	Selbstbedienungsstand, Name ändern		X*		
7	Selbstbedienungsstand löschen		X*		
8	Selbstbedienungsstand, Benutzer hinzufügen		X*		
9	Selbstbedienungsstand, neuen Benutzer erstellen		X*		
10	Ware anlegen	X			
11	Ware löschen	X			
12	Warendaten ändern	X			
13	Warenbewegung eingeben	X			
14	Tagesabschluss erstellen	X			
15	JSON Web Token	X			
16	Verschlüsseltes Passwort in Datenbank speichern	X			
17	Erweiterbares System	X			
18	Plattformunabhängig	X			
19	Standortunabhängig		X		
20	Responsive-Webdesign		X		
21	Tagesabschluss als PDF-Dokument	X			
22	Leichte Bedienbarkeit		X		
23	Übersichtliches Design		X		
24	Deutsche Textausgabe	X			

X* Funktion ist bereits auf dem Server implementiert.

Tabelle 19: Überprüfung der Anforderungen

Die Tabelle 19 listet nochmals die Anforderungen auf. In einigen Spalten X* zu sehen (Zeilen 2 bis 9), diese beutetet das die Funktion bereits auf dem Server implementiert worden sind.
Nach Fertigstellung der Funktion auf dem Server wurden diese getestet. Getestet wurde mit dem Programm Postman.[7]. Damit ist es möglich unterschiedliche HTTP-Anfragen durchzuführen.

[7]https://www.postman.com

Die Anforderung Nr. 19 kann nue Teilweise erfüllt werden, da für die nutzung der Anwendung ein Internetzugang erforderlich ist. Abhilfe könnte eine App schaffen, die die Daten lokal speichert und sobald Internetempfang vorhanden ist diese überträgt.

Die Anforderung Nr. 20 Responsive-Webdesign wurde nur teilweise erfüllt, die Navigationsleiste erinnert an klassische Desktopanwendungen. Ein Hamburger-Menü ist hingegen besser bedienbar von mobilen Endgeräten.

Die Anforderungen 22 und 23 lassen sich schwer überprüfen, da dieses jeder Benutzer anderes wahrnimmt. Dieses lässt sich am besten in einem Praxistest mit unterschiedlichen Anwendern überprüfen.

7 Fazit und Ausblick

In diesem letzten Kapitel folgt ein kurzes Fazit und es wird ein kurzer Ausblick geben.

7.1 Fazit

Bei der Entwicklung der Anwendung hat sich gezeigt, dass eine gute und ausführliche Planung der Anwendung wichtig ist. Vornherein muss genau überlegt werden, was für Funktionen die Anwendung beinhalten soll. Eine genaue Auflistung ist hier vom Vorteil.
Danach muss überlegt werden, welche Daten brauche ich und wie speicher ich diese am besten in der Datenbank. Dabei ist es wichtig, gründlich und genau zu arbeiten, ich habe die Erfahrung machen müssen, dass es im Nachhinein schwierig ist, noch weitere Tabellenspalten hinzuzufügen.

Im Anschluss folgt die Konzeption des Servers, hierbei ist es wichtig, die Pakete und Klassen möglichst geordnet anzulegen. Außerdem sollte darauf geachtet werden, das bekannte Architekturmuster umgesetzt werden. Diese haben sich häufig in der Praxis bewährt grade bei größeren Projekten helfen diese dabei, eine Struktur beizubehalten.

Als besonders wichtig hat sich die genaue Planung der Schnittstellen herausgestellt. Es sollte eine Schnittstellenbeschreibung angefertigt werden. Dieses erleichtert die Entwicklung des Clients ungemein. Da dort übersichtlich alle Schnittstellen dargestellt werden können.

Es hat sich herausgestellt, dass sich die Entwicklung eines Angular-Clients vereinfachen lässt, in dem vorab eine Skizze der Benutzeroberfläche angefertigt wird. Diese lässt sich in einzelne Bestandteile zerlegt und somit gut in Komponenten aufteilen.

Abschließend lässt sich sagen, dass es wichtig ist, die Software gut zu planen, dadurch lassen sich viele Fehler vermeiden, die sonst erst bei der Realisierung der Software auffallen. Es hat sich gezeigt, dass sich grade kleine Fehler, die am Anfang des Projektes gemacht werden, am Ende des Projektes zu immer größeren Fehlern führen.

7.2 Ausblick

Die Anwendung kann und soll noch weiter entwickelt werden, dieses ist leicht möglich, da diese modular aufgebaut worden ist. Folgende Funktionen sind noch denkbar:

- Erstellung eines Warenverkaufsschildes im PDF Format
- Erstellung unterschiedlicher Statistiken (Jahresstatistik, Monatsstatistik)
- Einbindung eines Kassenzählprotokolls
- Berechnung des Verlustes, durch Diebstahl usw.
- App-Entwicklung

Außerdem soll ein Praxistest erfolgen, mit Besitzern von Selbstbedienungsständen, dabei soll sich herausstellen, ob diese Anwendung in der Paxis erfolgversprechend eingesetzt werden kann.

Literaturverzeichnis

[Abt10] Abts, Dietmar: *Masterkurs Client/Server-Programmierung mit Java.* 3., erweiterte Auflage. Wiesbaden : Vieweg +TeubnerVerlag I Springer Fachmedien Wiesbaden GmbH, 2010

[Ang] *Webentwicklung mit dem JavaScript-Framework Angular von Google.* https://bmu-verlag.de/webentwicklung-mit-dem-javascript-framework-angular-von-google/. – Eingesehen am 08.07.2021

[Ang17] *React vs. Angular – wann ist was besser?* https://t3n.de/news/react-vs-angular-besser-845241/. Version: 2017. – Eingesehen am 08.07.2021

[Ang18a] *Angular vs. React - Ein ausführlicher Vergleich. Welche Bibliothek passt zu Ihrem Projekt?* https://www.ventzke-media.de/blog/angular-vs-react-vergleich.html. Version: 2018. – Eingesehen am 08.07.2021

[Ang18b] *Der neue Stern am JavaScript-Himmel: Vue.js gewinnt den Beliebtheitswettbewerb.* https://entwickler.de/online/javascript/javascript-vue-579827096.html. Version: 2018. – Eingesehen am 10.07.2021

[Ang18c] *Was ist eigentlich Vue.js?* https://t3n.de/news/vuejs-1097879/. Version: 2018. – Eingesehen am 10.07.2021

[Ang19] *Warum wir auf Vue.js setzen.* https://www.wigital.de/blog/warum-vue-als-javascript-framework.html. Version: 2019. – Eingesehen am 10.07.2021

[Ang20a] *Angular vs. React vs. Vue.js – JavaScript Frontend Frameworks im Vergleich.* https://tappert-it.de/2020/01/01/angular-react-vue-javascript-frontend-frameworks-vergleich/. Version: 2020. – Eingesehen am 08.07.2021

[Ang20b] *React vs Angular: was ist besser?* https://www.yuhiro.de/react-vs-angular-was-ist-besser/. Version: 2020. – Eingesehen am 08.07.2021

[Ang20c] *Webapps mit Angular entwickeln.* https://zeix.com/durchdacht/2019/11/12/webapps-mit-angular-entwickeln/. Version: 2020. – Eingesehen am 08.07.2021

[Ang20d] *Welche Vorteile bietet Vue.js 3 für Entwickler?* https://vuejs.de/artikel/vuejs-developer-benefits/. Version: 2020. – Eingesehen am 10.07.2021

[Ang21a] *Angualr Dokumentation.* https://angular.io. Version: 2021

[Ang21b] *SPA Frameworks - ein Vergleich zwischen Angular, React und Vue.* https://www.snipclip.com/blog/spa-frameworks-ein-vergleich-zwischen-angular-react-und-vue/. Version: 2021. – Eingesehen am 08.07.2021

[Ang21c] *Welche Vorteile bietet Vue.js 3 für Entwickler?* https://www.hosttest.de/artikel/angular-vs-react-vs-vue-js-was-ist-das-beste-framework. Version: 2021. – Eingesehen am 10.07.2021

[bit] *SQL, MySQL und MS-SQL.* https://www.bitfarm-archiv.de/dokumentenmanagement/glossar/sql-mysql.html. – Eingesehen am 01.07.2021

[Bun18] *Neufassung des § 146 Abs. 1 AO durch das Gesetz zum Schutz vor Manipulationen an digitalen Grundaufzeichnungen vom 22. Dezember 2016; Anwendungserlass zu § 146 AO.* https://www.bundesfinanzministerium.de/Content/DE/Downloads/BMF_Schreiben/Weitere_Steuerthemen/Abgabenordnung/AO-Anwendungserlass/2018-06-19-aenderung-anwendungserlass-abgabenordnung-Einzelaufzeichnungspflicht.pdf?__blob=publicationFile&v=2. Version: 2018. – Eingesehen am 11.07.2021

[cli20] *Was ist das Client-Server-Modell?* https://www.ip-insider.de/was-ist-das-client-server-modell-a-940627/. Version: 2020. – Eingesehen am 05.07.2021

[dat] *SQL Einführung – Was ist SQL ?* https://www.datenbanken-verstehen.de/sql-tutorial/sql-einfuehrung/#google_vignette. – Eingesehen am 01.07.2021

[DB218] *Weniger Redundanz dank Datenbank-Normalisierung.* https://www.ionos.de/digitalguide/hosting/hosting-technik/normalisierung-von-datenbanken/. Version: 2018. – Eingesehen am 26.07.2021

[db219] *CRUD.* https://mindsquare.de/knowhow/crud/. Version: 2019. – Eingesehen am 07.10.2021

[DB3] *Vergleich der Systemeigenschaften MariaDB vs. MySQL vs. Oracle.* https://www.datenbanken-verstehen.de/datenmodellierung/normalisierung/dritte-normalform/. – Eingesehen am 28.07.2021

[DB4] *Dritte Normalform (3NF).* https://db-engines.com/de/system/MariaDB%3BMySQL%3BOracle. – Eingesehen am 26.07.2021

[DB521] *XAMPP.* https://www.heise.de/download/product/xampp-10929. Version: 2021. – Eingesehen am 29.07.2021

[DBR19] *Was ist Normalisierung?* https://www.bigdata-insider.de/was-ist-normalisierung-a-840412/. Version: 2019. – Eingesehen am 26.07.2021

[Dep21] *Dependency-Injektion in Spring Boot.* https://www.dev-insider.de/dependency-injektion-in-spring-boot-a-1005234/. Version: 2021. – Eingesehen am 18.09.2021

[dir20] *Neue und innovative Formen der Direktvermarktung landwirtschaftlicher Produkte - Analyse und Erarbeitung von Handlungsempfehlungen.* https://orgprints.org/id/eprint/37311/1/37311-15NA192-ecozept-boehm-2020-innodirekt.pdf. Version: 2020. – Eingesehen am 07.10.2021

[Fraa] *Definition Framework.* https://www.datacenter-insider.de/was-ist-rest-api-a-714434/. – Eingesehen am 07.07.2021

[Frab] *Was ist ein Framework?* https://www.bsh-ag.de/it-wissensdatenbank/framework/. – Eingesehen am 07.07.2021

[Fun] *Funktionale und Nicht-Funktionale Anforderungen.* https://deutsche-projekt-akademie.de/2020/03/pm-verstehen-funktionale-und-nicht-funktionale-anforderungen/. – Eingesehen am 14.07.2021

[gesa] *Fachliche ERläuterungen zu Kasse: Vertauenskasse.* https://www.verfahrensdokumentation.services/files/erlaeuterungen/kasse-vertrauenskasse.pdf. – Eingesehen am 12.07.2021

[gesb] *Kassenführung im Hofladen.* https://www.lw-heute.de/kassenfuehrung-hofladen. – Eingesehen am 12.07.2021

[gesc] *Vertrauenskasse (Deutsch).* https://www.wortbedeutung.info/Vertrauenskasse/. – Eingesehen am 12.07.2021

[ges10a] *Definition: Landwirtschaftliche Direktvermarktung.* http://norddeutsche-direktvermarkter.de/fileadmin/Dokumente/PDF_Unterlagen/Def_DV.pdf. Version: 2010. – Eingesehen am 12.07.2021

[ges10b] *Die neuen Steuer-Regeln für Direktvermarkter.* Ausgabe 5/2010. Münster : top agrar, Landwirtschaftsverlag GmbH, 2010

[ges15] *Infoservice Direktvermarktung/Einkommensalternativen, Rechtsbestimmungen in der Direktvermarktung.* https://www.lwk-rlp.de/fileadmin/lwk-rlp.de/Beratung/EA/PDF/DV/DV_01.1_Recht_Uebersicht.pdf. Version: 2015. – Eingesehen am 12.07.2021

[ges17] *Merkblatt Direktvermarktung landwirtschaftlicher Erzeugnisse.* https://www.landkreis-ansbach.de/media/custom/2238_2539_1.PDF?1468233256. Version: 2017. – Eingesehen am 12.07.2021

[gui20] *Erfolgreiche UI-Gestaltung - 5 Einsteiger-Tipps, die du bei erstellen einer Benutzeroberfläche beachten solltest.* https://softwarekontor.de/2020/10/28/ui-design-basics/. Version: 2020. – Eingesehen am 01.10.2021

[Hor07] Horn, Torsten: *SQL-Grundlagen.* https://www.torsten-horn.de/techdocs/sql.htm. Version: 2007. – Eingesehen am 01.07.2021

[JPA20] *Java persistence API (JPA).* https://www.itwissen.info/JPA-Java-persistence-API.html. Version: 2020. – Eingesehen am 16.09.2021

[Kar14] *HALLO, ICH BIN DIE KARTOFFELKISTE!* https://www.potatis-hof.de/s/selbstbedienungsstand/. Version: 2014. – Eingesehen am 11.07.2021

[Lub17] Luber, Stefan: *Was ist SQL?* https://www.dev-insider.de/was-ist-sql-a-586264/. Version: 03 2017

[map17] *Abbildung.* https://www.itwissen.info/mapping-Abbildung.html. Version: 2017. – Eingesehen am 17.09.2021

[Ora] Oracle: *Was ist eine relationale Datenbank?* Markt + Technik Verlag https://www.oracle.com/de/database/what-is-a-relational-database/. – Eingesehen am 30.06.2021

[Reg20] *Direktvermarktung :Der Einkauf beim Bauern liegt im Trend.* https://www.bzfe.de/nachhaltiger-konsum/einkaufsorte-finden/direktvermarktung/. Version: 2020. – Eingesehen am 11.07.2021

[rel17] *Was ist eine relationale Datenbank?* https://www.bigdata-insider.de/was-ist-eine-relationale-datenbank-a-643028/. Version: 2017. – Eingesehen am 02.07.2021

[RES] *REST-API: Definition, Funktionen und Bedingungen.* https://www.talend.com/de/resources/was-ist-rest-api/. – Eingesehen am 06.07.2021

[RES17] *Was ist REST-API?* https://www.cloudcomputing-insider.de/was-ist-eine-rest-api-a-611116/. Version: 2017. – Eingesehen am 06.07.2021

[RES18] *Was ist REST-API?* https://www.datacenter-insider.de/was-ist-rest-api-a-714434/. Version: 2018. – Eingesehen am 06.07.2021

[Sel] *Selbstbedienung (SB).* http://www.wirtschaftslexikon24.com/e/selbstbedienung-sb/selbstbedienung-sb.htm. – Eingesehen am 11.07.2021

[Sera] *Was ist ein Paket.* https://www.java-tutorial.org/objektorientierung-pakete.html. – Eingesehen am 14.09.2021

[Serb] *Was ist ein Paket.* https://www.a-coding-project.de/ratgeber/java/klassen-und-objekte. – Eingesehen am 15.09.2021

[Sin18] *Was ist eigentlich eine Single-Page-Webanwendung?* https://t3n.de/news/single-page-webanwendung-1023623/. Version: 2018. – Eingesehen am 10.07.2021

[Spr18] *Spring Boot: Vom Hype zur etablierten Basistechnologie?* https://m.heise.de/developer/artikel/Spring-Boot-Vom-Hype-zur-etablierten-Basistechnologie-4247771.html?seite=all. Version: 2018. – Eingesehen am 10.07.2021

[Spr19a] *Spring – das Framework für komplexe Java-Applikationen.* https://www.ionos.de/digitalguide/websites/web-entwicklung/spring-framework-das-steckt-in-dem-java-grundgeruest/. Version: 2019. – Eingesehen am 10.07.2021

[Spr19b] *Was ist das Spring Framework?* https://www.dev-insider.de/was-ist-das-spring-framework-a-829846/. Version: 2019. – Eingesehen am 10.07.2021

[spr21] *Core Technologies.* https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-introduction. Version: 2021. – Eingesehen am 18.09.2021

[SQL21] SQL, W3Schools: *SQL Tutorial.* https://www.w3schools.com/sql/default.asp. Version: 2021. – Eingesehen am 30.06.2021

[Sta21] *HTTP/Statuscodes*. https://wiki.selfhtml.org/wiki/HTTP/Statuscodes. Version: 2021. – Eingesehen am 07.10.2021

[t3r17] *Was sind eigentlich relationale Datenbanken?* https://t3n.de/news/eigentlich-relationale-datenbanken-683688/. Version: 2017. – Eingesehen am 02.07.2021

[tok20] *Was ist JWT?* https://www.egovernment-computing.de/was-ist-jwt-a-984685/. Version: 2020. – Eingesehen am 01.10.2021

[Ull17] Ullenboom, Christian: *Java ist auch eine Insel*. 12. Auflage. Bonn, 2017

[Umf21a] *Aktuelle Umfrage bestätigt Ernährungspolitik*. https://www.bmel.de/SharedDocs/Pressemitteilungen/DE/2021/089-ernaehrungsreport.html. Version: 2021. – Eingesehen am 11.07.2021

[umf21b] *Bevölkerung in Deutschland nach Einstellung zur Aussage Ïch kaufe nach Möglichkeit Produkte hier aus der Regionïm Geschlechtervergleich im Jahr 2020*. https://de.statista.com/statistik/daten/studie/826841/umfrage/umfrage-in-deutschland-zum-kauf-von-produkten-aus-der-region-im-geschlechtervergleich/. Version: 2021. – Eingesehen am 05.10.2021

[Win08] Winkler, Peter: *Computer Lexikon 2009, Die ganze digitale Welt zum Nachschlagen*. Vollständig überarbeitete Neuauflage. München : Markt + Technik Verlag, 2008

Anhang

Tabellen in der 3. Normalform

BenutzerNr	Benutzername	Passwort
1	Christine	12345
2	ADE	asdfg
3	Hugo	yxcvb

Tabelle 20: Die Tabelle Benutzerdaten in 3. Normalform.

SelbstbedienungsstandNr	Selbstbedienungsstandname
1	Eierhütte
2	Kartoffelhaus
3	Stand1

Tabelle 21: Die Tabelle Selbstbedienungsstand in der 3. Normalform.

SelbstbedienungsstandNr	BenutzerNr	RollenNr
1	1	1
2	1	2
3	2	1
3	3	1

Tabelle 22: Die Tabelle Selbstbedienungsstand-Nutzer in der 3. Normalform.

RollenNr	Benutzerrollename
1	Admin
2	User

Tabelle 23: Die Tabelle Benutzerrollen in der 3. Normalform.

WarenNr	Warenname	EinheitNr	Preis(€)	Bemerkung
1	Ei	1	0,20	
2	Kartoffeln	2	5	Sorte Annabelle

Tabelle 24: Die Tabelle Waren in der 3. Normalform.

Einheit-Nr	Name der Einheit
1	Stück
2	KG

Tabelle 25: Die Tabelle Einheit in der 3. Normalform.

WareneingangsNr	Datum	Wareneingang(Anzahl)	WarenNr	BenutzerNr
1	25.07.2021	+100	2	1
2	25.07.2021	+80	1	1
3	26.07.2021	+60	2	1
4	26.07.2021	+5	2	2

Tabelle 26: Die Tabelle Wareneingang in der 3. Normalform.

Datum	WarenNr	BenutzerNr	Gezählt	Verkauft	Einnahmen(€)
25.07.2021	1	1	231	251	50,20
26.07.2021	1	1	52	239	47,80
26.07.2021	2	1	3	2	10

Tabelle 27: Die Tabelle Tagesabschluss in der 3. Normalform.

Quellcode Server (Ausschnitt)

```
import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;

/**
* Class represents the table "tb_end of_the_day".
*/
@Table(name = "tb_end of_the_day", indexes = {
  @Index(name = "cl_user_nr", columnList = "cl_user_nr"),
  @Index(name = "cl_goods_nr", columnList = "cl_goods_nr"),
  @Index(name = "cl_self_service_stand_nr",
  columnList = "cl_self_service_stand_nr")
})
@Setter
@Getter
@Entity
public class EndOfTheDay {

  /**
  * ID End of the Day.
  */
  @EmbeddedId
  private EndOfTheDayId id;

  /**
  * Refers to a user ID.
  */
  @ManyToOne
  @JoinColumn(name = "cl_user_nr")
  private User userNr;

  /**
  * Refers to a self-Service-Stand name.
  */
  @ManyToOne
  @JoinColumn(name = "cl_self_service_stand_nr")
  private SelfServiceStand selfServiceStandName;

  /**
  * The number of goods counted.
  */
  @Column(name = "cl_graduation_count")
  private Integer graduationCount;

  /**
```

```
47    * The number of computationally determined goods sold.
48    */
49    @Column(name = "cl_graduation_sold")
50    private Integer graduationSold;
51
52    /**
53    * The number of mathematically determined income in euros.
54    */
55    @Column(name = "cl_graduation_revenue")
56    private Double graduationRevenue;
57  }
```

Listing 16: Das Listing zeigt die Klasse EndOfTheDay im Paket model

```
1   /**
2   * Class represents the table ID "tb_end of_the_day".
3   */
4   @Getter
5   @Setter
6   @Embeddable
7   public class EndOfTheDayId implements Serializable {
8
9     /**
10    * serialVersionUID
11    */
12    private static final long serialVersionUID = 45487321313401781 44L;
13
14    /**
15    * End of the day date.
16    */
17    @Column(name = "cl_graduation_date", nullable = false)
18    private LocalDate graduationDate;
19
20    /**
21    * The commodity number.
22    */
23    @Column(name = "cl_goods_nr", nullable = false)
24    private Integer goodsNr;
25
26    /**
27    * Overwritten hash code method.
28    *
29    * @return Objects
30    */
31    @Override
32    public int hashCode() {
33      return Objects.hash(goodsNr, graduationDate);
34    }
```

```
35
36    /**
37    * Overwritten equals method.
38    *
39    * @param o goodsNr and graduationDate
40    * @return Objects
41    */
42    @Override
43    public boolean equals(Object o) {
44      if (this == o) return true;
45      if (o == null || Hibernate.getClass(this) != Hibernate.getClass(o)
       ) return false;
46      EndOfTheDayId entity = (EndOfTheDayId) o;
47      return Objects.equals(this.goodsNr, entity.goodsNr) &&
48      Objects.equals(this.graduationDate, entity.graduationDate);
49    }
50  }
```

Listing 17: Das Listing zeigt die Klasse EndOfTheDayId im Paket model

```
1  /**
2  * This class contains the database queries of the table
       tb_receipt_goods.
3  */
4  @Repository
5  public interface ReceiptGoodRepository extends JpaRepository<
       ReceiptGoods, Integer> {
6
7    /**
8    * Query, finds a record with the incoming ID.
9    * @param receiptGoodId The receipt goods Id.
10   * @return record receiptGoods
11   */
12   ReceiptGoods findByReceiptGoodId(Integer receiptGoodId);
13
14   /**
15   * Query, provides a list of all self-service booths that have the ID
       entered.
16   * @param selfServiceStandNId The self-service-stand Id.
17   * @return list receiptGoods
18   */
19   List<ReceiptGoods>
       findBySelfServiceStandNr_SelfServiceStandNIdOrderByDateReceiptAsc(
       Integer selfServiceStandNId);
20
21   /**
22   * Query, provides a list of all self-service booths that have the ID
       and product name entered.
```

```
23   * @param goodsName The goods name.
24   * @param selfServiceStandNId The self-service-stand Id.
25   * @return list receiptGoods
26   */
27   List<ReceiptGoods>
      findByGoodsNr_GoodsNameNr_GoodsNameAndSelfServiceStandNr_SelfServiceStandNId
      (String goodsName, Integer selfServiceStandNId);
28
29   /**
30   * Query, finds a list of all dates and goods In that match.
31   * @param dateReceipt The receipt date.
32   * @param goodsId The goods ID.
33   * @return list receiptGoods
34   */
35   List<ReceiptGoods> findByDateReceiptAndGoodsNr_GoodsId(LocalDate
      dateReceipt, Integer goodsId);
36 }
```

Listing 18: Das Listing zeigt das Interface ReceiptGoodRepository im Paket repository

```
1  /**
2  * The role Controller class outputs a list of all existing units.
3  * It is possible to create new units.
4  * Units can also be changed and deleted, but only if they are not yet
       used in another table.
5  *
6  * <p>
7  * Provides the following endpoints: <br>
8  * - GET /role/ <br>
9  * - POST /role/{unitName} <br>
10 * - DELETE /role/{oldUnitName} <br>
11 * - PUT /role/{oldUnitName}/{newUnitName} <br>
12 */
13 @RestController
14 @RequestMapping("/unit")
15 public class UnitController {
16
17   /**
18   * Logger of the unitController class.
19   */
20   private static final Logger LOGGER = LoggerFactory.getLogger(
       UnitController.class);
21
22   /**
23   * unitRepository to handle unit information.
24   */
25   private final UnitRepository unitRepository;
26
```

```
27  /**
28  * goodsRepository to handle goods information.
29  */
30  private final GoodsRepository goodsRepository;
31
32  public UnitController(UnitRepository unitRepository, GoodsRepository
      goodsRepository) {
33    this.unitRepository = unitRepository;
34    this.goodsRepository = goodsRepository;
35  }
36
37  /**
38  * Method returns a sorted list with all units.
39  *
40  * @return json list with all units
41  */
42  @GetMapping(path = "/", produces = MediaType.APPLICATION_JSON_VALUE)
43  private ResponseEntity<Object> getUnits() {
44
45    //Query sorted list, sorted by unit name
46    return new ResponseEntity<>(unitRepository.
     findByOrderByUnitNameAsc(), HttpStatus.OK);
47  }
48
49  /**
50  * Method creates a new unit, but only if it does not yet exist.
51  *
52  * @param unitName Name of the unit
53  * @return Http Status code: Create if a new unit has been created,
     otherwise no content
54  * and created unit.
55  */
56  @PostMapping(path = "/{unitName}", produces = MediaType.
     APPLICATION_JSON_VALUE)
57  private ResponseEntity<Unit> createUnit(@PathVariable String
     unitName) {
58
59    //excess whitespace is removed
60    unitName = unitName.trim();
61
62    Unit newUnit = null;
63
64    //It is checked whether the unit already exists.
65    if (!unitRepository.existsByUnitNameIs(unitName)) {
66
67      //save the new unit
68      newUnit = unitRepository.save(new Unit(unitName));
69
```

```
70        LOGGER.info("The unit " + unitName + " has been created.");
71      } else {
72        LOGGER.info("The unit " + unitName + " is already there.");
73      }
74
75      // Set the http status by unit; CREATED if it is null, NO_CONTENT
        otherwise
76      final HttpStatus httpStatus = null != newUnit
77      ? HttpStatus.CREATED
78      : HttpStatus.NO_CONTENT;
79
80      return new ResponseEntity<>(newUnit, httpStatus);
81    }
82
83    /**
84    * The method only deletes a unit from the list if it has not yet
        been used.
85    *
86    * @param unitName unit name
87    * @return message in Json format and Http status code
88    */
89    @DeleteMapping(path = "/{unitName}", produces = MediaType.
        APPLICATION_JSON_VALUE)
90    private ResponseEntity<JSONObject> deleteUnit(@PathVariable String
        unitName) {
91
92      //create new json object
93      JSONObject message = new JSONObject();
94
95      //excess whitespace is removed
96      unitName = unitName.trim();
97
98      //It is checked whether the unit is not used in the "goods" table.
99      if (!goodsRepository.existsByGoodsUnitNr_UnitName(unitName)) {
100
101       Unit deleteUnit = new Unit();
102       deleteUnit.setUnitId(unitRepository.findByUnitName(unitName).
        getUnitId());
103
104       //Delete Unit at the database
105       unitRepository.delete(deleteUnit);
106
107       LOGGER.info("The unit " + unitName + " has been deleted.");
108       message.put("Message", "The unit " + unitName + " has been
        deleted.");
109       return new ResponseEntity<>(message, HttpStatus.OK);
110     } else {
111       LOGGER.info("The unit " + unitName + " is already in use.
```

```
   Deletion is no longer possible.");
112        message.put("Message", "The unit " + unitName + " is already in
   use. Deletion is no longer possible.");
113        return new ResponseEntity<>(message, HttpStatus.CONFLICT);
114     }
115  }
116
117  /**
118  * The method only changes the name of a unit if it has not yet been
    used in the Goods table.
119  *
120  * @param oldUnitName old unit name
121  * @param newUnitName new unit name
122  * @return message in Json format and Http status code
123  */
124  @PutMapping(path = "/{oldUnitName}/{newUnitName}", produces =
    MediaType.APPLICATION_JSON_VALUE)
125  private ResponseEntity<JSONObject> updateUnit(@PathVariable String
    oldUnitName, @PathVariable String newUnitName) {
126
127     //create new json object
128     JSONObject message = new JSONObject();
129
130     //excess whitespace is removed
131     oldUnitName = oldUnitName.trim();
132     newUnitName = newUnitName.trim();
133
134     //It is checked whether the unit is not used in the "goods" table.
135     if (!goodsRepository.existsByGoodsUnitNr_UnitName(oldUnitName)) {
136
137        Unit changeUnit = new Unit();
138        changeUnit.setUnitId(unitRepository.findByUnitName(oldUnitName).
   getUnitId());
139
140        changeUnit.setUnitName(newUnitName);
141
142        //change unit at the database
143        unitRepository.save(changeUnit);
144
145        LOGGER.info("The name of the unit has been changed from " +
   oldUnitName + " to " + newUnitName + ".");
146        message.put("Message", "The name of the unit has been changed
   from " + oldUnitName + " to " + newUnitName + ".");
147        return new ResponseEntity<>(message, HttpStatus.OK);
148     } else {
149        LOGGER.info("The unit " + oldUnitName + " is already in use.
   Change is no longer possible.");
150        message.put("Message", "The unit " + oldUnitName + " is already
```

```
    in use. Change is no longer possible.");
151        return new ResponseEntity<>(message, HttpStatus.CONFLICT);
152      }
153    }
154 }
```

Listing 19: Das Listing zeigt die Klasse UnitController im Paket controller

Quellcode Client (Ausschnitt)

```
1  .container {
2    display: grid;
3    grid-template-columns: 0.5fr 2fr 8fr 2fr 0.5fr;
4    grid-template-rows: 0.4fr 3.4fr 0.4fr 0.4fr;
5    gap: 0px 0px;
6    grid-auto-flow: row;
7    grid-template-areas:
8    "navbar navbar navbar navbar navbar"
9    "main main main main main"
10   "selectSelf selectSelf . . ."
11   "footer footer footer footer footer";
12 }
13
14 .navbar {
15   grid-area: navbar;
16   background: var(--colorGray);
17 }
18
19 .main {
20   grid-area: main;
21   background: white;
22 }
23
24 .footer {
25   grid-area: footer;
26   background: var(--colorGray);
27 }
28
29 .selectSelf {
30   grid-area: selectSelf;
31   background: white;
32 }
33
34 html,
35 body {
36   height: 100%;
```

```
37   background: radial-gradient(var(--colorBackground2), var(--
      colorBackground1));
38   overflow: hidden;
39 }
40
41 .container-main-screen {
42   background: white;
43 }
```

Listing 20: Das Listing zeigt die CSS-Datei main-screen.component

```
1  const myRoutes: Routes = [
2  {
3    path: '',
4    component: MainScreenComponent,
5    children: [
6    {
7      path: 'goods',
8      component: GoodsComponent,
9      children: [
10     {
11       path: 'overview',
12       component: OverviewGoodsComponent
13     },
14     {
15       path: 'new',
16       component: NewGoodsComponent
17     },
18     {
19       path: 'change',
20       component: ChangeGoodsComponent
21     }, {
22       path: 'settings',
23       component: SettingsGoodsComponent
24     }
25     ]
26   },
27   {
28     path: 'receipt',
29     component: ReceiptComponent,
30     children: [
31     {
32       path: 'new',
33       component: NewReceiptComponent
34     },
35     {
36       path: 'overview',
37       component: OverviewReceiptComponent
```

```
38      },
39      ]
40    },
41    {
42      path: 'user',
43      component: UserComponent,
44      children: [
45      {
46        path: 'change',
47        component: ChangeUserComponent
48      },
49      {
50        path: 'create',
51        component: CreateUserComponent
52      },
53      {
54        path: 'overview',
55        component: OverviewUserComponent
56      }
57      ]
58    },
59    {
60      path: 'statistics',
61      component: StatisticsComponent,
62      children: [
63      {
64        path: 'month',
65        component: MonthComponent
66      },
67      {
68        path: 'year',
69        component: YearComponent
70      }
71      ]
72    },
73    {
74      path: 'self',
75      component: SelfServiceStandComponent,
76      children: [
77      {
78        path: 'add',
79        component: AddUserSelfComponent
80      },
81      {
82        path: 'change',
83        component: ChangeSelfComponent
84      },
85      {
```

```
      path: 'overview',
      component: OverviewSelfComponent
    }
    ]
  },
  {
    path: 'end',
    component: EndOfDayComponent,
    children: [
    {
      path: 'overview',
      component: OverviewEndComponent
    },
    {
      path: 'new',
      component: NewEndComponent
    },
    {
      path: 'create',
      component: EndOfDayPdfComponent
    }
    ]
  },
  {
    path: '',
    component: MainMenuComponent
  },
  {
    path:'**',
    component: PageNotFoundComponent,
  },
  ]
}
]

@NgModule({
  declarations: [],
  imports: [
  CommonModule,
  RouterModule.forChild((myRoutes))
  ],
  exports: [RouterModule]
})
export class MainScreenRoutesModule {
}
```

Listing 21: Das Listing zeigt die CSS-Datei main-screen.component

```
1
2  /**
3  * Class provides an overview of all goods.
4  */
5  export class OverviewGoodsComponent implements OnInit {
6
7    //empty list goods
8    goodsList: Goods[] = [];
9
10   //Heading of the table
11   header: string[] = ['Warenname', 'Einheit', 'Preis', 'Bemerkung', '
       im Verkauf'];
12
13   //Saves whether a result was found
14   searchResult: boolean = true;
15
16   //List of search results
17   searchResultList: Goods[] = [];
18
19
20   constructor(private client: HttpClient,
21   private communication: ServerCommunicationServiceService
22   ) {
23   }
24
25   ngOnInit(): void {
26     this.loadGoods();
27     this.searchResultList = this.goodsList;
28   }
29
30   //Filter the list of goods according to the search results and save
       them in the list
31   search(event: Event) {
32
33     //Old search results are deleted
34     this.searchResultList = [];
35
36     //Saves the keystrokes
37     const filterValue = (event.target as HTMLInputElement).value;
38
39     //Searches the list for the word
40     for (let key in this.goodsList) {
41       if (this.goodsList.hasOwnProperty(key)) {
42         if (
43         this.goodsList[key].goodsName.includes(filterValue) ||
44         this.goodsList[key].unitName.includes(filterValue) ||
45         this.goodsList[key].goodsPrice.toString().includes(filterValue
       ) ||
```

```
46          this.goodsList[key].goodsNote.includes(filterValue)
47          ) {
48            //Saves all found experiences in the list
49            this.searchResultList.push((this.goodsList[key]));
50          }
51        }
52        //If not found, the variable is set to false.
53        this.searchResult = this.searchResultList.length > 0;
54      }
55    }
56
57    // Loads a list of goods from the server
58    private loadGoods() {
59
60      //set url
61      let url = 'goods/' + localStorage.getItem('
       key_selfServiceStand_name');
62
63      //communication Server
64      this.communication
65      .serverCommunication(url, '', 'get', true)
66      .subscribe((response: any) => {
67
68        //Saves the goods in a list
69        for (var entry of response) {
70          this.goodsList.push({
71            goodsId: entry.goodsId,
72            goodsName: entry.goodsNameNr.goodsName,
73            unitName: entry.goodsUnitNr.unitName,
74            goodsPrice: entry.goodsPrice,
75            goodsNote: entry.goodsNote,
76            goodsActive: entry.goodsActive
77          });
78        }
79      }, error => {
80        console.log(error);
81      }
82      );
83    }
84  }
```

Listing 22: Das Listing zeigt die TypeScript-Datei change-goods.ts

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Projektarbeit

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Unterschrift :

Ort, Datum :