HAW HAMBURG

BACHELOR THESIS
Anit Kumar Rana

Entwicklung eines Prototyps für ein Kassensystem mit einer technischen Sicherheitseinrichtung (TSE)

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Anit Kumar Rana

Entwicklung eines Prototyps für ein Kassensystem mit einer technischen Sicherheitseinrichtung (TSE)

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Martin Hübner

Eingereicht am: 26. Sep 2023

Anit Kumar Rana

Thema der Arbeit

Entwicklung eines Prototyps für ein Kassensystem mit einer technischen Sicherheitseinrichtung (TSE)

Stichworte

KassenSichV, TSE, Kassensystem, ECDSA, digitale Signatur

Kurzzusammenfassung

In dieser Arbeit wurde ein Prototyp für ein Kassensystem mit einer technischen Sicherheitseinrichtung(TSE) entwickelt. Ziel dieser Arbeit ist es, zu überprüfen, wie alle Kassentransaktionen mit einer TSE sichergestellt werden können, um eine nachträgliche Manipulation der Aufzeichnung zu verhindern. Dabei werden die verschiedenen kryptographischen Operationen wie Hash-Funktionen, Elliptic Curve Digital Signature Algorithm (ECDSA), digitale Signaturen näher untersucht. Anhand dieses Prototyps wird deutlich, dass mit einem Secure Element (SE) und einer digitalen Signatur die Kassentransaktionen sicherer gegen Manipulationen gemacht werden können.

Anit Kumar Rana

Title of Thesis

Developing a prototype for Point-of-Sale(POS) with Technical security device (TSE)

Keywords

POS-System, Secure Element, Digital Signature, ECDSA

Abstract

In this work a prototype for a POS-System with a technical device(TSE) was developed. The goal of this work is to analyze how all transactions are secured with a TSE to prevent tampering of the record. The various cryptographic operations such as hash function, Elliptic Curve Digital Signature Algorithm (ECDSA) , digital signatures are examined

in detail. At the end this prototype shows that with the help of a Secure Element(SE) and a digital signature it is possible to make the transactions more secure.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

> „Wer aufzeichnungspflichtige Geschäftsvorfälle oder andere Vorgänge mit Hilfe eines elektronischen Aufzeichnungssystems erfasst, hat ein elektronisches Aufzeichnungssystem zu verwenden, das jeden aufzeichnungspflichtigen Geschäftsvorfall und anderen Vorgang einzeln, vollständig, richtig, zeitgerecht und geordnet aufzeichnet.“(§ 146a AO)

[7]Die erste Registrierkasse wurde 1879 von James Ritty, einem Saloon-Besitzer aus Ohio, erfunden.1.1. Diese Erfindung ermöglicht die fehlerfreie Spicherung der Transaktionen und verbessert auch die Buchführung. Diese wurde in den darauffolgenden Jahren stetig weiterentwickelt. [17] Im Jahr 1986 wurde das erste Kassensystem mit Berührungsbildschirmfunktion von Atari entwickelt und später, in den 1990er- Jahren, das erste Kassensystem mit dem Windows Betriebsystem entwickelt. Im Jahr 1990 hat Microsoft ein Kassensystem für Restaurants entwickelt, der die Finanzdatenbank und Kundeninteraktionen enthält. Heute ist das Kassensystem in vielfältigen Formen in verschiedenen Geschäften im Einsatzt[18] . Es steht sogar auch eine Cloudbasierte Lösung zu Verfügung.

Die Digitalisierung hat in den letzten Jahrzehnten viele Änderungen mit sich gebracht. Mit der Digitalisierung kommt auch die Sammlung von großen Mengen von Datensätzen, die es in früherer Zeiten nicht gab. Früher im alten Kassensystem haben die Händler täglich den Tagesabschluss manuell gemacht, wobei dies heutzutage vom Kassensystem selbst generiert oder erfasst wird. Egal ob es sich um einen kleinen oder größeren Betrieb handelt, gibt es große Mengen an Daten in digitaler Form. Daraus ergeben sich natürlich große Herausforderungen bezüglich Datensicherheit und Vertraulichkeit. Das Finanzamt will auch sicher sein und prüfen, ob die Daten, die an das Finanzamt geschickt worden sind, nicht manipuliert wurden.

Abbildung 1.1: Erste Ritty Registerkasse.
Quelle: https://ohiomemory.org, 2010

Um dieses Problem zu lösen, gibt es eine Konzept, das sich Fiskalisierung nennt. Fiskalisierung ist ein Konzept, bei dem alle Kassentransaktionen korrekt protokolliert werden um die Manipulation von Aufzeichnungen zu verhindern.[30] Eine Fiskalisierung von Registrierkassen ist nicht neu, viele Länder in Europa wie Italien, Schweden, Polen usw. haben sie bereits implementiert. In Italien z.B. gibt es einen Fiskaldrucker, um die Manipulation von Aufzeichnungen zu vermeiden .[26]

Da es viele Möglichkeiten zur Manipulation von Aufzeichnungen gibt, hat auch Deutschland geplant, die Fiskalisierung von Registrierkassen zu implementieren. Laut dem Finanzamt München können die Registrierkassen, die im Zeitraum vom 26.11.2010 bis zum 31.12.2020 besorgt wurden und noch kein TSE System integriert haben, noch bis zum 31.12.2022 verwendet werden[32]. Danach müssen alle Registrierkassen in Deutschland eine sogenannte Technische Sicherheitseinrichtung(TSE) integriert haben.

1.2 Problemstellung

- **Aktueller Stand der POS-System**: In vielen Geschäften werden immer noch keine digitalen Kassensysteme verwendet. Die aktuellen Kassensysteme haben viele Lücken bezüglich Integrität und Sicherheit von Aufzeichnungen der Transaktionen.
- **Notwendigkeit der TSE-Integration**: Die Integration einer TSE in POS-Systeme ist sehr wichtig, um die Sicherheit der Transaktionen zu gewährleisten. Es ist nun auch gesetzlich erforderlich , um die Integrität der Transaktionsaufzeichnung sicherzustellen.

1.3 Zielsetzung

Das Hauptziel dieser Bachelorarbeit ist es, einen benutzerfreundlichen Prototyp eines Kassensystems/ (POS)-Systems zu implementieren, der mit einer Technischen Sicherheitseinrichtung (TSE) integriert ist, mit Hilfe von kryptographischen Operationen. Das Prototyp des Kassensystems wird durch eine Web-Applikation realisiert, sodass es leicht wiederverwendbar ist, sowohl auf Handys, als auch auf Tablets, Laptops usw. Dafür wird das Angular-Framework für das Frontend-GUI und Spring Boot für das Backend verwendet. Für die Implementierung des Prototyps werden verschiedene Softwareentwicklungsthemen wie Microservices, REST-APIs, Socket Server, Secure Element (SE) und Docker aufgegriffen.

Das Hauptziel der Arbeit liegt in der Sicherung der Transaktionensaufzeichnungen. Deshalb werden im Rahmen dieser Arbeit verschiedene kryptographische Operationen, wie die Generierung von Digitalen Signaturen, Secure Element, Hash Funktion usw. eingesetzt. Am Ende sollte eine funktionierender Kassensystem-Prototyp entwickelt worden sein, bei dem ein Benutzer/Kassier die Artikel in den Warenkorb legen, entfernen und andere Kassenfunktionen verwalten kann. Die Integration der TSE und die Verwendung von kryptographischen Operationen sind wichtig, um die Integrität der Transaktionen sicherzustellen, sodass keine Manipulation mehr möglich ist, wenn einmal der Kassenbeleg generiert ist. Es wird versucht, soweit wie möglich, den gesetzliche Anforderungen zu entsprechen.

1.4 Aufbau der Arbeit

Diese Arbeit ist in sieben Kapitel eingeteilt.

Kapitel 1 In diesem Kapitel wird das Thema dieser Arbeit erläutert.

Kapitel 2 Hier werden die Grundlagen im Kontext des Prototyps beschrieben

Kapitel 3 Hier werden die verschiedenen Anforderungen und Spezifikationen an das Kassensystem vorgestellt.

Kapitel 4 In diesem Kapitel wird die Konzeption entwickelt nach den Anforderungen aus Kapitel 3

Kapitel 5 In diesem Kapitel wird nach der Konzeption und Architekur in Kaptiel 4 das Protyp umgesetzt

Kapitel 6 In diesem Kapitel wird die Qualitätssicherung/ der Test beschrieben und bewertet.

Kapitel 7 In diesem Kapitel wird die Arbeit zusammengefasst und ein Ausblick geschrieben.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe der Themen beschrieben, die bei der Entwicklung des Protyps für ein integriertes Kassensystem mit einer technischen Sicherheitseinrichtung(TSE) wichtig sind.

2.1 Registerkasse

2.1.1 Point of Sales(POS)

Ein POS oder eine Registrierkasse ist eine Maschine, die verwendet wird, um die Transaktionen von Geschäften aufzuzeichen.[2] Aktuell gibt es viele Arten von Registrierkassen wie eine physische Hardware in einem Geschäft, eine virtuelle Kasse für Online Shops, eine Cloud-Basierte Lösung usw. Mittlerweile gibt es auch ein Kassensystem, das mit der Warenwirtschaft verbunden ist, das bedeutet, die Überprüfung des Artikelzustands , die Sammlung von Marketingdaten (wie viel an welchem Tag verkauft wurde) usw.

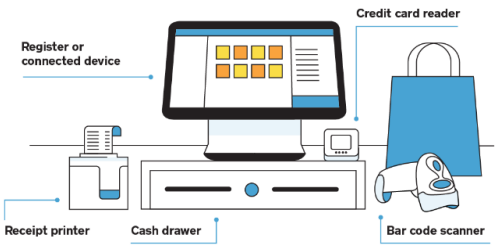

Abbildung 2.1: The Point of Sales (POS)
Quelle: https://squareup.com/us/en/the-bottom-line/operating-your-business/what-pos-system

2.1.2 KassenSichV

KassenSichV ist die Abkürzung für "Gesetz zum Schutz vor Manipulationen an digitalen Grundaufzeichnungen"[13]. Es ist eine neue Verordnung in Deutschland um die Integrität der Kassenaufzeichnung zu gewährleisten.

2.1.3 GoBD

"GoBD bedeutet Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form."[19] Die GoBD legt fest, welche die Unternehmen beim der Aufzeichnung und Buchhaltung einhalten müssen, sodass das Finanzamt die erforderlichen Informationen für die Steuer akzeptiert.

2.2 Datentyp

2.2.1 JSON-Daten

JSON ist die Abkürzung für JavaScript Object Notation. Jason Datentyp ist eine häufig verwendete Struktur für den Datenaustausch z.B bei Web-Applikationen, Konfiguration. Die Datei endet auf *.json. Es handelt sich um ein Schlüssel-Wert Paar, das in einer geschweiften Klammer und dem Schlüssel-Wert Paar in doppelten Anführungszeichnen geschrieben wird. Jedes Paar ist mit einem Komma voneinander getrennt.[3] Es ist sehr einfach lesbar und schnell zu erstellen.

Abbildung 2.2: Beispiel JSON-Format

Wie in Abbildung 2.2 zu sehen ist, beginnt eine JSON-Struktur eine Sammlung mit einer öffnenden Klammer und endet mit einer schließenden Klammer.[31]. Der Inhalt wird mit dem Doppelpunkt voneinander getrennt.

2.2.2 YAML-Dateien

YAML ist eine Abkürzung für YAML Ain't Markup Language. Es ist auch wie JSON eine bekannte Struktur einer Datei. Es wird häufiger für die Konfiguration von Applikationen z.B Server Konfigurationen, Datenbank Konfigurationen usw. verwendet.[28] Die Datei besteht aus Schlüssel-Wert Paaren wie in Abbildung 2.3 zu sehen ist, die durch einen Doppelpunkt getrennt sind und auch hierarchisch geordnet sind. Es ist auch sehr einfach lesbar und einfach zu erstellen.

```yaml
mongodb-service:
  image: mongo
  restart: always
  ports:
    - "27017:27017"
  volumes:
    - ./data:/data/db
  environment:
    MONGO_INITDB_ROOT_USERNAME: admin
    MONGO_INITDB_ROOT_PASSWORD: admin
```

Abbildung 2.3: YAML-Dateien Struktur

2.3 Security

2.3.1 Die Technische Sicherheitseinrichtung(TSE)

[20]Nach BSI Anforderungen soll die Technische Sicherheitseinrichtung(TSE) drei Modulen haben:

- **Sicherheitsmodul**
- **Speichermedium**
- **digitale Schnittstelle**

Ablauf Transaktion:

- Digitale Schnittstelle - Sie ist dafür da, die Kommunikation zwischen TSE-Server und Kassensystem aufzubauen
- Das Sicherheitsmodul- Wird verwendet, um wichtige Protokoll-Informationen zu generieren wie z.B Transaktionsnummer, Start- und Endzeitstempel, Digitale Signatur/ Prüfwert

- Speichermedium - Hier werden die Transaktionsdaten sicher gespeichert.

Überprüfungsschritt:

- Die Transaktionsdaten, die geprüft werden sollen, werden mit Hilfe der Digitalen Schnittstelle aufgerufen
- Public Key wird verwendet, die dem Prüfer bekannt ist, um die Integrität von Signatur zu prüfen
- Der Zeitstempel bei Protokollierung kann verwendet werden, um einzusehen, wann die Transaktion mit TSE abgesichert wurde. Transaktionnummer und Singnaturnummern können verwendet werden um die Manipulation an den Transaktiondaten zu erkennen.

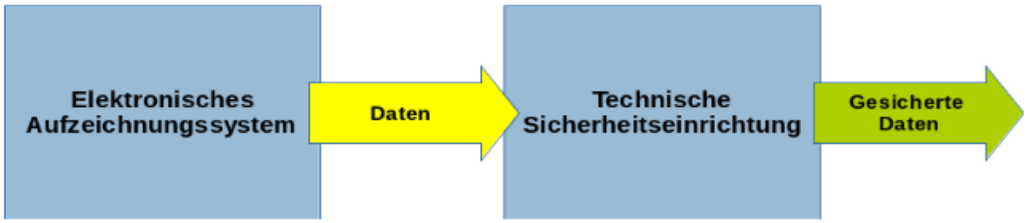

Abbildung 2.4: Absicherung von digitalen Grundaufzeichnungen
Quelle: Bundesamt für Sicherheit in der Infomationstechnik (BSI)

2.3.2 Secure Element

[34]Ein Secure Element ist ein Gerät, das es in verschidene Form wie SD-Karte,USB-Geräte, SIM-Karten usw. gibt, das mehrere Funktionen hat z.B Erkennung von Manipulation von Daten, sichere Speicherung von Schlüsseln, kryptografische Operationen wie Entschlüsselung oder Verschlüsselung, Generierung von Schlüsseln usw.

2.3.3 Kryptographie

Kryptographie ist eine Technik, wie Nachrichten von nicht autorisierte Personen geschützt werden können.[8] Die wichtigsten Punkte der Kryptographie sind Vertraulichkeit, Integrität, Authentizität und Nichtabstreitbarkeit.

2.3.4 Verschlüsselung und Entschlüsselung

Verschlüsselung ist die Umwandlung von Daten in eine bestimmte Form, die als Chiffretext bezeichnet wird. Es vermeidet, dass Daten von nicht autorisierten Personen einfach gelesen werden können. Entschlüsselung ist die Umwandlung von den verschlüsselten Daten in ihre ursprüngliche Form. Siehe2.5. Um den Inhalt von verschlüsselten Nachrichten wiederherzustellen, wird ein richtiger Entschlüsselung-Schlüssel benötigt. Je komplexer der Verschlüsselungsalgorithmus ist, desto schwieriger wird es für nicht autorisierte Personen, die Daten/Kommunikation abzuhören, ohne den Zugriff auf den richtigen Schlüssel. [21]. In Abbildung 2.5 wird der Ablauf der Verschlüsselung und Entschlüsselung visualisiert.

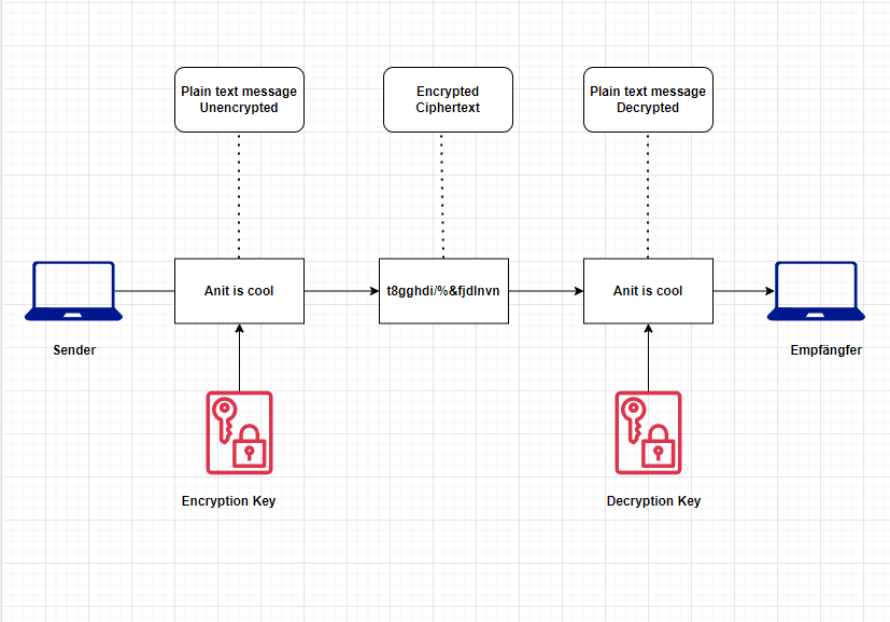

Abbildung 2.5: Encryption Ablauf

2.3.5 Symmetrische Verschlüsselung

Symmetrische Verchlüsselung ist eine Art von Verschlüsselung, bei dem sowohl für die Verschlüsselung als auch für die Entschlüsselung der gleiche private Schlüssel verwendet wird. Um das zu erreichen, wird eine sichere Verbindung zwischen Sender und Empfänger benötigt, um den privaten Schlüssel miteinander zu tauschen. Die symmetrische Verschlüsselung kann in zwei Hauptkategorien geteilt werden, nämlich eine Blockverschlüsse-

lung und die Stromverschlüsselung.[24] Bei der Blockverschlüsselung (Blockchiffre) werden die Daten in einen Block mit einer bestimmten Länge umgewandelt, wohingegen bei der Stromverschlüsselung die Daten als Strom von Bits umgewandelt werden.

2.3.6 Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselung ist ähnlich zur symmetrischen Verschlüsselung, aber sie verwendet zwei unterschiedliche Schlüssel, nämlich einen privaten und einen öffentlichen Schlüssel.[24]. Der private und der öffentliche Schlüssel wird jeweils für die Verschlüsselung und Entschlüsselung verwendet.

2.3.7 Secure Hash Algorithm (SHA)

Eine Hash Funktion ist ein Algorithmus, bei dem aus einer Eingabe beliebiger Länge eine Ausgabe fester Länge in Binärform generiert wird. Der Secure Hash Algorithm ist eine Art von Hash-Algorithmus, der während der Erzeugung einer digitalen Signatur verwendet wird, um die Vertraulichkeit und Integrität digitaler Daten zu überprüfen. Der Secure Hash Algorithmus (SHA) wurde vom „The National Institute of Standards and Technology(NIST)ïm Jahr 1993 erfunden. [27].

2.3.8 Digitale Signatur

Es gibt mehrere Arten von Verschlüsselungstechniken, um die Vertraulichkeit und Integrität von digitalen Daten sicherzustellen. Eine davon ist die Digitale Signatur. Eine digitale Signatur stellt die Vertraulichkeit und Integrität von digitalen Daten sicher, wobei zum einen Mathematische Verfahren wie der Hashing-Algorithmus und zu anderen eine von den verschiedene Verschlüsselungsalgorithmen wie RSA, ECDSA usw. verwendet werden. 2.6.

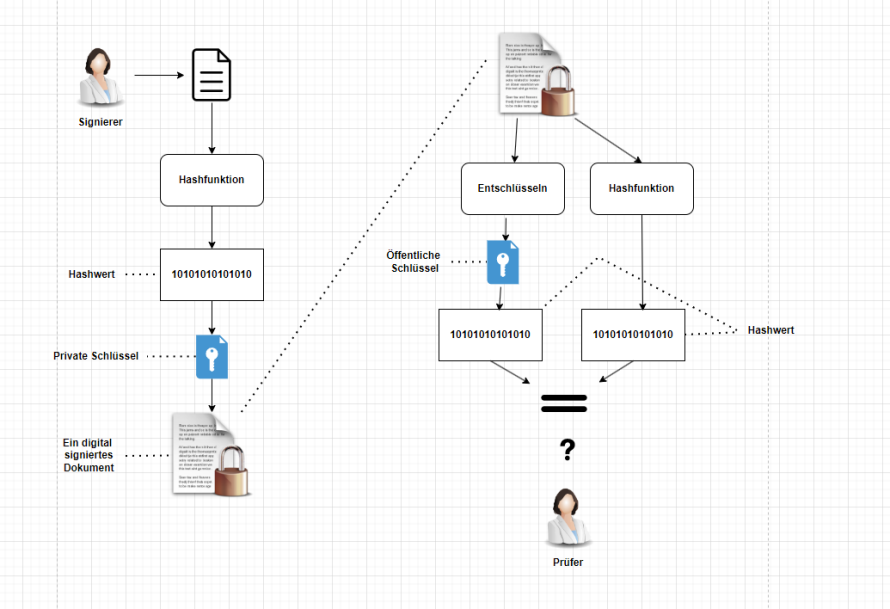

Abbildung 2.6: Digitale Signatur Ablauf

In Abbildung 2.6 wird die Generierung und Prüfung von Digitalen Signaturen angezeigt.

Schritt 1: Signieren und Verschlüsseln

Im ersten Schritt wird das zu signierende Dokument von dem Signierer geschickt und wird mithilfe einer Hash Funktion eine Message digest generiert. Message digest ist eine Einweg-Hashfunktion, die die ursprüngliche Eingabe nimmt und in einer bestimmten Länge in binärer Form ausgibt [12]. Aus dem generierten Hash-Wert oder „Message digest "wird mithilfe vom privaten Schlüssel und Signatur Algorithmus eine Digitale Signatur generiert.[33]. Am Ende wird das ursprüngliche Dokument mit der generierten Digitalen Signatur zusammengestellt.

Schritt 2: Entschlüsselung und Überprüfung

Zuerst werden die zusammen verpackten Daten getrennt (ursprünglich Daten und Signatur). Der Prüfer entschlüsselt die digitale Signatur mithilfe von öffentlichen Schlüsseln

(der öffentliche Schlüssel muss dem Prüfer bekannt sein). Danach wird vom ursprüngliche Dokument mit dem gleichen Hash-Algorithmus ein Hash-Wert generiert. Zum Schluss werden die beiden Hash-Werte verglichen, um die Authentizität des Dokuments zu überprüfen. [33]. Wenn die beiden Hash-Werte unterschiedlich sind, dann kann man feststellen, dass möglicherweise eine Manipulation vorliegt.

2.3.9 ECDSA

ECDSA ist eine Abkürzung für Elliptic Curve Digital Signature Algorithm und wird verwendet um eine Digitale Signatur zu erzeugen. ECDSA wurde zuerst 1992 von Scott Vanstone vorgeschlagen.[1]. Es verwendet ein asymmetrisches Verschlüsselungverfahren auf Basis elliptischer Kurven. Elliptische Kurven haben kürzere Schlüssel als andere Alogrithmus wie RSA.

2.4 Webentwicklung und APIs

2.4.1 Web Applikation

Eine Web Applikation ist eine Anwendung, die mithilfe eines Webbrowsers im Internet funktioniert[35]. Diese wird in einer Websprache (wie HTML, JavaScript, CSS usw.) oder einem Framework wie Angular, React geschrieben. In Abbildung 2.7 ist zu sehen, die erste Website, die entwickelt wurde.

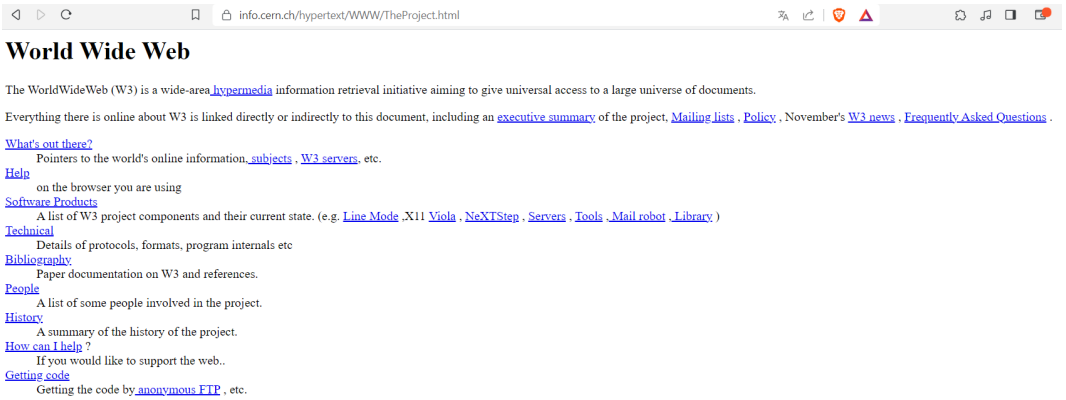

Abbildung 2.7: Die erste Webseite (6. Aug 1991) Quelle: info.cern.ch

2.4.2 Application Programming Interface (API)

Ein API ist eine vordefinierte Regel oder Protokoll, die ein Program mit einem anderen Program eine bestimmte Operation durchführen oder kommunizieren lässt. Mit hilfe von API können die Ressourcen und bestimmte Funktionen von Webdienste zugriefen.

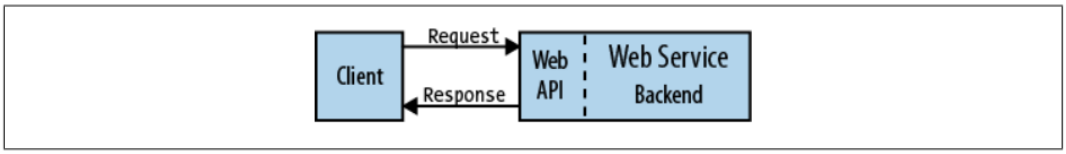

Abbildung 2.8: Web APIs
Quelle: [23]

2.4.3 REST APIs

TODO Ein REST-API (Representational State Transfer Application Programming Interface) ist eine Schnittstelle, die für verschidene Bedürfnisse von Webaplikation unterstützen .Eine Client kann mit Hilfe von APIs mit der Webdienste kommunizeren z.B Verschiedene HTTP-Anfragen wie GET,PUT,POST,DELETE. Webdienste stellt verschiende Funktion und Ressourcen zu verfügung und bearbeitet die Anfragen von Client.Es stellt eine Verbinndungs für eine Datenaustausch zwischen die verschiedene

Programm.REST Architektur ist verwendet um eine APIs des moderne Webdienste zu realisieren.[23].Für die Kommunikation zwischen die verschiendenen Microservices werden auch REST APIs für das Kassenprotyp benutzt.

2.4.4 Client-Server-Architektur

Die Trennung der Anliegen (Seperation of Concerns) ist das wichtigste Punkt der lient-Server-Architektur. Das Web ist ein Client-Server basierendes System, so die Client und Server bestimmte Rolle haben. Es können in eine beliebige Programmiersprache unabhängig von einander implementiert werden [23].Es werden auch später bei der Prototyp für TSE verwendet.

2.4.5 Socket Programmierung

Eine Client-Server-Architektur kann mit Hilfe von Socket Programminierung implementiert werden. Ein Socket ist eine Kommunikationsverbindung Punkte zwischen Client und Server, die in einem Netzwerk laufen.[29] .In Verteilte System wird es häufiger verwendet.Die Datenaustausch wird entweder mit Datagramm oder Stream verwendet. In Java hat auch zwei Klassen nämlich Socket und ServerSocket, um Verbindungen zwischen Clients und Servern herzustellen. Diese Klasse werden auch während der Entwicklung des Kassenprototyps verwendet.

2.5 Microservice

Microservices werden heutezutage sehr verbreitete Software-Architektur die eine Unternehmen verwenden. Im gegenteil zu monolithischen Architektur, in Microservices wird die Softwareanforderungen in kleine Teil geteillt und jede Teil ist verantwortlich für bestimmte Funktion.Größe Unternehmen wie Amazon, Netflix, Uber etc verwenden auch Microservices.[16].Das Vorteil von Microservices sind :

- Agilität - schnell Angorderungen ändern nach bedarf
- Produktivität - effizient arbeiten
- Widerstandsfähigkeit - wenn ein Service Fehler hat dann können andere Services weiterhin laufen

- Skalierbarkeit - Nach bedarf kann die Services schnell skalierbar
- Wartbarkeit
- Die Trennung von Anliegen - Die verschiedene Services können unabhängig von einander entwickelt werden
- Einfach zu Deployen

[4] Für das Kassensystemprotyp wird diese Architektur verwendet.

2.6 Docker

TODO Docker ist eine Technologie, die für die Virtualisierung von Containern verwendet.Es ist sehr leichtgewicht Virtuelle Maschine(VM) .[5] Eine Studie von University of Arizona berichtete, dass weniger als 50% Software wegen „Software Dependency "nicht erfolgreich völlständig gebildet und installiert könnten.[9]. Diese Problem könnte mit Docker lösen da es eine Abhängigkeiten von Software in eine isolierte Umgebung mit Hilfe von Docker Container verwaltet werden kann.

2.7 Bouncy Castle

Bouncy Castle ist eine kryptografische Bibliothek für Java und C#.[11] Die bieten verschiedene kryptografisher Algorithmen und Funktionen. Für das Kassensystemsprototyp wird diese Bibliothek verwendet um eine digitale Signatur sowie auch für die Genierung von Schlüsseln.

2.8 Tools zur Implementation

2.8.1 Angular Framework

Angular ist ein Framework und eine Entwicklungsplattform, die verwendet wird, um effiziente und anspruchsvolle Single-Page-Applicationen (SPA) zu erstellen. Eine SPA ist eine Webapplikation, die den Inhalt auf der aktuellen Webseite dynamisch mit neuen Daten vom Webserver aktualisiert anstatt die komplette neue Seite zu laden.[22] Angular

basiert auf Komponenten basierte Architektur. Das ist eine der wichtigste Punk, wo die einzelne Komponent unabhängig von anderen Komponenten bauen(Seperation of Concern) können, um ein Komponent einfach wieder zu verwenden und modifizieren, weil die Komponente nur für eine spezifische Aufgabe zuständig ist. [6]

2.8.2 Springboot

Springboot ist ein Open-Source-Framework für die Entwicklung von Java basierten enterprise applikation. Es macht die Konfiguration, den Aufbau und den Einsatz von Java-Anwendungen einfacher insbesondere im Bereich der Webanwendungen und Microservices. [10]

2.8.3 MongoDb

MongoDB ist eine Open-Source-Datenbank und gehört zu den NoSQL-Datenbanken. Im Gegensatz zu relationalen Datenbanken gibt es keine vordefinierten Tabellenschema. Die Daten werden als Sammlungen von Dokumenten dargestellt und im Binary JSON (BSON)-Format gespeichert.[15]

3 Anforderungen und Spezifikation an das System

In diesem Kaptel werden alle Anforderungen und die Spezifikationen für die Realisierung von dem Kassensystem-Protyp beschrieben.

3.1 Anforderungen des Kassensystems

1. **Lauffähigkeit auf allen Betriebssystemen** - Der zu entwickelnde Kassensystem Prototyp soll nicht nur auf einem Betriebsystem laufen sondern auf allen Betriebssystemen lauffähig sein.
2. **Nachvollziehbarer und vertraulicher Transaktionsablauf** - Der gesamte Transaktionsablauf soll derart aufgezeichnet und protokolliert werden, dass er nachvollziehbar und vertraulichkeit ist.
3. **Benutzerfreundliche Oberfläche/GUI** - Das Dashboard/GUI für das Kassensystem soll so implementiert sein, dass es benutzerfreundlich ist (z.B passende große Tasten für Tablet oder andere Touchscreen Geräte), um ein gutes Benutzererlebnis zu haben.
4. **Grundlegende Funktionen von einer elektronischen Registrierkasse** - Das Kassensystem soll grundlegende Funktionen wie das Scannen eines Artikels oder die manuelle Eingabe eines Artikels, Hinzufügen von Artikeleinkäufen in einen Warenkorb und das Erstellen eines Kassenbons erfüllen. Hauptziel ist, dass alle notwedigen Kassenfunktionen/ Schritte vorhaden sind, um einen Kassenbon zu generieren.
5. **Benötigte Daten** - Um den Kassensystemprototyp zu realisieren, wird eine Liste von Artikeln mit grundlegenden Attributen wie ID, Preis, Beschreibung usw. benötigt (Schnittstelle für Mock-Artikel um in einer Warenkorb hinzuzufügen).

6. **Technische Anforderungen** - Für das Frontend soll Angular Version 14.1.0 oder höher benutzt werden, das Backend soll Spring Boot Version 2.8.2 oder höher sein, und es soll eine NOSQL Datenbank wie MongoDB verwendet werden, um daen Kassenbeleg zu speichern oder zu verwalten. Das Kassensystem soll der Microservices-Architektur folgen, weil während der Entwicklungsphase recht schnell der Überblick verloren gehen kann.

3.2 Spezifikation

3.2.1 Kassenregister Dashboard

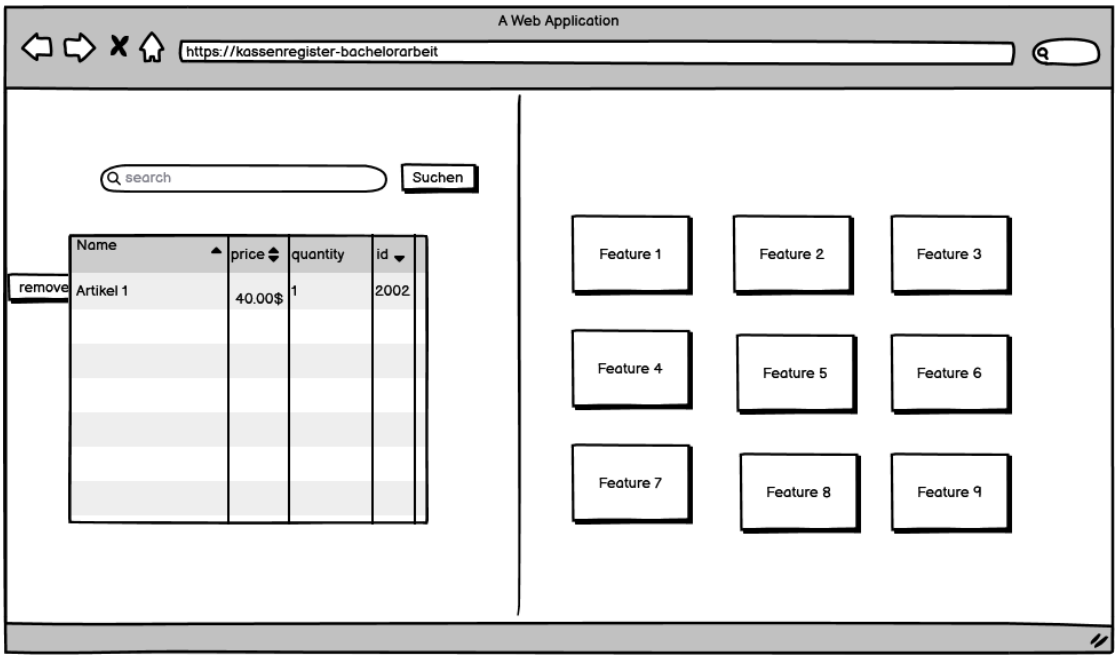

Abbildung 3.1: Kassenregister Dashboard mockup

Wie bei Abbildung 3.1 zu sehen ist, soll der zu entwicklnede Kassensystem-Prototyp mit integrierter TSE ein Dashboard zur Verfügung haben, das in zwei Teile aufgeteilt ist. Auf der linken Seite befindet sich die Suchfunktion für Artikel und das Hinzufügen zum Warenkorb sowie Löschen von Artikel aus dem Warenkorb, aktueller Preis von Artikeln die im Warenkorb liegen, Preis von Artikeln ändern/Rabatt usw. . Auf der rechten Seite des Dashboards sind die Funktionen eines Kassensystems, wie das Erstellen eines

Warenkorbs, die Auswahl der Zahlungsmethoden, die Generierung des Kassenbelegs, Warenkorb löschen, Liste eines offenen Warenkorbs, der noch nicht abgeschlossen ist. Das Kassensystem soll flexibel oder sehr schnell änderbar sein, sodass in Zukunft, wenn eine neue Funktion/ Button hinzugefügt wird, die bestehende GUI weiterhin funktioniert.

3.2.2 Systemanwendungsfälle

Die folgenden Punkte zeigen die Kommunikation zwischen den verschiedenen Microservices für den Kassensystem-Prototyp

1. **Kommunikation zwischen Kassensystem Dashboard mit mock-Produkt-Server** Lieferung von Artikel Information müssen bereitgestellt werden. Dafür wird eine Verbindung zwischen Frondend GUI des Prototyp und mock Server der Artikel vorhanden sein. z.B Bei einem Request durch den Frontend an mock- Produkt-Server soll, der gesuchte Artikel geliefert werden oder nicht gefunden zurückliefern.
2. **Kommunikation zwischen Kassensystem-Dashboard und POS-System-Service** Es soll eine REST-Schnittstelle bereitgestellt werden, um die Kommunikation zwischen dem Kassensystem-Dashboard und dem POS-System-Service zu schaffen. Es ist aber sehr wichtig, dass das zu entwickelnde Prototyp-Dashboard/Frontend nur POS-Sytem-Service kennt, sodass alle Anfragen aus einem Kassensystem zu den anderen Microservices nur mit einem Endpukt durchgeführt wird. Um die Sicherheitslücken zu gewährleisten. z.B Kassensystem-Dasboard kommuniziert mit dem TSE-Server, wobei zu erst die Anfrage an das POS-System-Service (Kassensystem Haupt Backend) schickt und wird von da weitergeleitet.
3. **Kommunikation zwischen POS-System-Service mit TSE-Service** Die Kommunikation zwischen POS-System-Service und TSE-Service soll mit einer Socket Programmierung realisiert werden, sodass eine sichere und schnellere Kommunikation aufgebaut wird, um eine eigenes Kommunikationprotokoll zu erstellen.

3.2.3 Geschäftsanwendungsfälle

Wie in Abbildung Anwendungsfalldiagram 3.2 zu sehen ist, wird in diesem Unterkapitel ein Anwendungsfallsdiagram angezeigt. Mit Hilfe dieser grafischen Darstellung kann

es einen Überblick schaffen, die Verbindung zwischen verschiedenen Komponenten des Kassensystems besser zu verstehen.

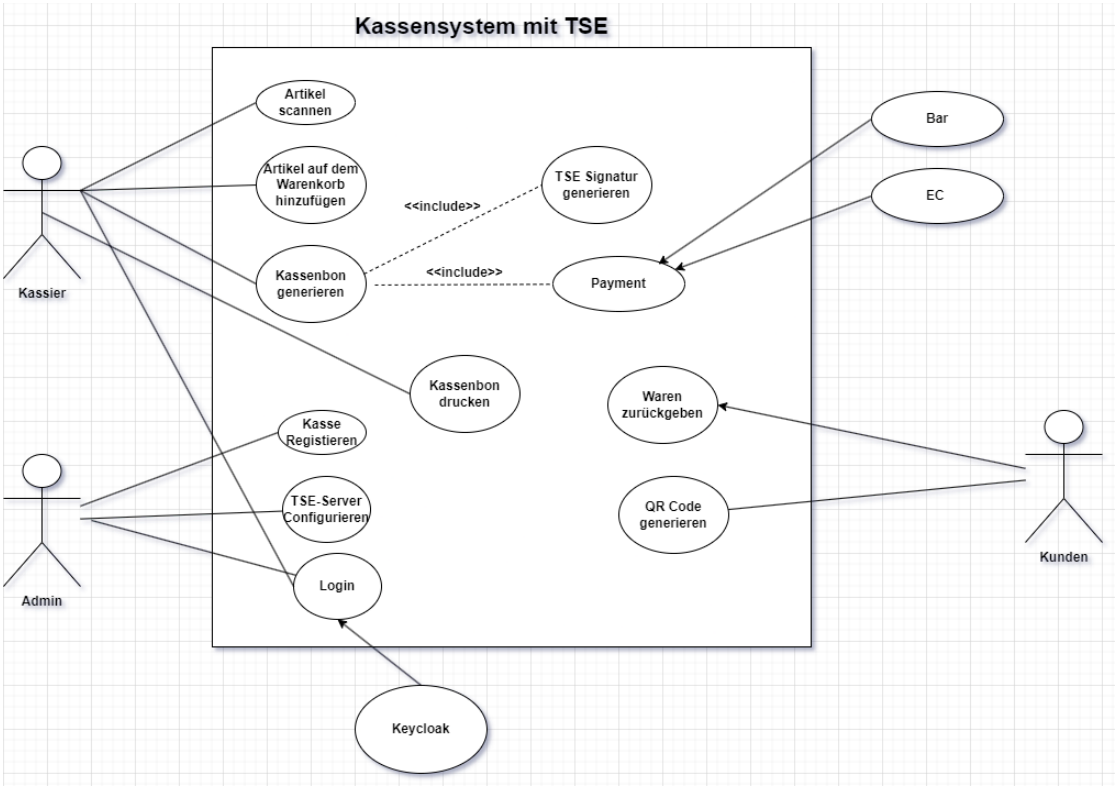

Abbildung 3.2: Anwendungsfallsdiagramm

Hier unten werden alle grundlegenden Use Cases für das Prototyp aufgelistet.

Artikel zum Warenkorb hinzufügen oder entfernen

Artikel im Warenkorb hinzufügen oder entfernen	
Zusammenfassung	Der Benutzer kann Artikel in dem Warenkorb hinzufügen oder aus dem Warenkorb entfernen.
Vorbedingungen	• Der Benutzer hat die Dashboard geöffnet. • Der Benutzer kann die verfügbaren Artikel anfragen.
Beschreibung des Ablaufs	1. Der Benutzer gibt den Artikel-ID manuell oder mit scanner. 2. Wenn der Benutzer ein Arikel auf der Datenbank findet dann kann es in Warenkorb hinzufügen 3. Ein Button ist vorhanden für Entfernen des Artikels aus dem Warenkorb 4. Der Benutzer entfernt das Artikel aus dem Warenkorb mit das Entfernen-Button
Ausnahme	• Falls ein Artikel nicht in der Datenbank verfügbar ist, wird eine Fehlermedung wie Ärtikel nicht gefunden! ängezeigt.
Nachbedingung	• Der Benutzer hat gesuchte Artikel in dem Warenkorb.

Tabelle 3.1: Artikel im Warenkorb hinzufügen oder entfernen

Eine Zahlungsmethode auswählen

Eine Zahlungsmethode auswählen	
Zusammenfassung	Der Benutzer kann aus verschiedenen Zahlungsmethoden(z.B EC,Paypal,Bar usw.) auswählen.
Vorbedingungen	• Der Benutzer hat den Warenkorb mit benötigte Artikel gefüllt.
Beschreibung des Ablaufs	1. Der Benutzer ´clickt das Zahlungsoption-Button in Kassensystem-Dashboard. 2. Auf der Dashboard wird eine Dialog-Fenster mit verschiedene Zahlungsmethoden angezeigt, z.B. Kreditkarte, Barzahlung, mobile Zahlungen. 3. Der Benutzer wählt eine der Zahlungsmethoden aus. 4. Wenn es Bargeld ausgewählt wird dann eine neue Dialog-Fenster mit ein Numpad wird angezeit wo das erhalten Summe von den Kunden eingibt und daraus wird die Enthaltene Summe und ZüruckGeld berechnet und das GUI aktualisiert 5. Bei anderen Zahlungsoptionen werden direkt eine Dialog-Box, ob Kassenbeleg gedrückt werden gefragt
Ausnahme	• Wenn erhaltene Bargeld weniger als zu Sein ist dann wird eine Fehlermeldung angezeigt oder wird die Transktion nicht weitergeführt
Nachbedingung	• Die ausgewählte Zahlungsmethode wurde für die Kassen-Transaktion verwendet.

Tabelle 3.2: Eine Zahlungsmethode auswählen

Initial Warenkorb automatisch Erstellen

Initial Warenkorb automatisch Erstellen	
Zusammenfassung	Wenn ein Dashboard initialisiert wird, dann ist ein neuer Warenkorb für den Benutzer automatisch erstellt. Ein eindeutiger Warenkorb-Identifier wird generiert damit die Artikel in dem Warenkörb hinzufügen kann.
Vorbedingungen	• Das POS-System-Backend ist Erreichbar. • Die MongoDB-Datenbank ist Erreichbar.
Hauptablauf	1. Der Benutzer initialisiert das Kassendasboard mit seinem Zugangsdaten(Wenn vorhanden) 2. Es wird automatsich eine Warenkorb bereitgestellt mit eine Default Warenkorbnamen. 3. Warenkorb name ist auf der Kassendashboard angezeigt. 4. Das Dashboard hat ein Button "Neue Warenkorb erstellen"wenn angeclickt wird , dann eine neue Dialog-Fenster wird geöffnet, mit einem Engabefeld. 5. Der Benutzer gibt eine Warenkorbnamen in das Eingabefeld und wird eine neue Warenkorb erstellt 6. Das Dasboard GUI wird mit neue Warenkorb aktualisiert. 7. Das Warenkorb ist bereit den Aritkel hinzufügen
Nachbedingungen	- Eine neue Warenkorb-Kollektion mit einem eindeutigen Warenkorb-Identifier wird in der Datenbank erstellt. - Das Dashboard zeigt den neu erstellten Warenkorb an.

Tabelle 3.3: Basket erstellen

Artikel zum zugehörigen Warenkorb hinzufügen

Artikel zum Aktuellen Warenkorb hinzufügen	
Zusammenfassung	Der Benutzer kann einen Artikel zu seinem zugehörigen Warenkorb hinzufügen.
Vorbedingungen	• Der Benutzer hat eine Warenkorb-Kollektion in der Datenbank mit einem gültigen Warenkorb-Identifier erstellt. • Mock-Produckt Service ist Erreichbar
Hauptablauf	1. Der Benutzer hat zwei Optionen, um eine Artikel hinzufügen. Wenn mit aktuelle Warenkorb fortfahren dann gehe Schritt 5 sonst Schritt 2 2. Der Benutzer clickt an das neue Warenkorb-Erstellen-Button, eine neue Eingabefeld wird angezeit. 3. Der Benutzer gibt eine Warenkorbnamen ein. 4. Das Kassendashboard GUI wird aktualisiert mit eine neue leere Warenkorb 5. Der Benutzer gibt das Artikel-Id und clickt das Suchen-Button oder Barcode wird eingescannt 6. Wenn das gesuchte Artikel vorhanden ist wird das Artikel sofort in dem Warenkorb hinzugefügt sonst eine Fehlermeldung wird angezeigt. 7. Das Warenkorb GUI wird mit der gesuchte Artikel aktualisiert 8. Das aktuale gesammte Summe wird berechnet und wird in Dasboard angezeigt
Nachbedingungen	- Die Warenkorb-Kollektion in der Datenbank enthält den neu hinzugefügten Artikel. - Gesamtpreis wird berechnet

Tabelle 3.4: Artikel zum Aktuellen Warenkorb hinzufügen

Ausstehende Warenkorb erstellen

Ausstehende Warenkorb erstellen	
Zusammenfassung	Der Benutzer kann die aktuelle Warenkorb als ausstehend markieren, um eine neue Warenkorb zu erstellen
Vorbedingungen	• Ein Warenkorb ist bereit erstellt und mindesten ein Artikel ist schon hinzugefügt
Hauptablauf	1. Der Benutzer hat ein Artikel für Person A in dem Warenkörb hinzugefügt. 2. Der Benutzer möchte eine neue Person B bedinenen. 3. Der Benutzer clickt and das neue Warenkorb-Button, und wird eine Eingabefeld angezeit 4. Der Benutzer gibt eine neue Warenkorb name für Person B und clickt Fertig-Button. 5. Der Warenkörb für Person A wird in den Datenbank gespeichert und wird auf der ausstehende Warenkörbeliste hingefügt. 6. Das Dasboard GUI wird mit eine neue Warenkorb(der leer ist) aktualisiert. 7. Der Benutzer clickt ausstehend Warenkorb-Button und wird die liste der ausstehende Warenkorb angezeigt 8. Der Benutzer wählt ein Option aus der Liste und das Dasboard GUI wird aktualisiert 9. Der Benutzer kann weitere aktion fortfahren
Nachbedingungen	- Ein neuer Warenkorb wurde erstellt. -Der ausstehende Warenkorb mit den hinzugefügte Artikel wurden Erfolgreich in den Datenbank gespeichert. - Der Benutzer kann gleichzeitig mehrere Personen bedinen

Tabelle 3.5: Ausstehende Warenkorb erstellen

Rabatten Optionen

Rabatten Optionen	
Zusammenfassung	Der Benutzer kann Rabatt in den ausgewählte Artikel anwenden. Es kann entweder eine feste Rabatt Option oder manuelle Eingabe verwenden
Vorbedingungen	• Ein Artikel wurde in dem Warenkorb hinzugefügt.
Hauptablauf	1. Es gibt eine Rabatt-Button in Warenkorb für einzelne Artikel die da vorhanden sind 2. Wenn der Benutzer auf den Rabatt-Button anklickt, dann wird eine Dialog-Box mit vershiende Option angezeit. 3. Der Benutzer kann entweder feste Beitrag von Rabatt auswählen oder manuell Eintippen 4. Je nach der Ausgewählte option wird die aktuelle Summe aktualisiert
Nachbedingungen	- Die ausgewählten Rabatte werden auf den Artikel angewendet und der Preis wird entsprechend aktualisiert.

Tabelle 3.6: Rabatten Optionen in Warenkorb

Kasse Registrieren und Zuordnen zu Frontend

Kasse Registrieren und Zuordnen zu Frontend	
Zusammenfassung	Eine Kasse wird im Kassensystem registiert sodass es bereit für Transaktionen ist.
Vorbedingungen	• Das Admin-Dashboard für das Kassensystem ist erreichbar.
Hauptablauf	1. Admin initialisiert das Admin-Dashboard 2. Es gibt ein Eingabefeld wo der Admin eine Kasseninformation eingeben kann. 3. Der Admin klickt den Speicher-Button an 4. Die Kasseninformationstabelle wird mit den akuellen Daten in der Datenbank aktualisiert. 5. Der Admin kann die gespeicherte Information in der Dashoard-Tabelle sehen. 6. Wenn das Kassensystem Dashboard (Warenkorb) initialisiert wird, wird erst überprüft ob die Kasse schon registiert ist. Wenn dies nicht der Fall ist, wird eine Fehlermeldung angezeigt
Nachbedingungen	- Kasseninformation wird erfolgreich in der Datenbank gespeichert

Tabelle 3.7: Kasse Registrieren und Zuordnen zu Frontend

Kassenbelg mit TSE Information erstellen

Kassenbelg mit TSE Information erstellen	
Zusammenfassung	Der Benutzer kann Kassenbeleg zusammen mit TSE Information erstellen
Vorbedingungen	• Der Benutzer hat mindestens einen Artikel zum Warenkorb hinzugefügt. • Der Benutzer hat eine Bezahlungsart ausgewählt.
Hauptablauf	1. Der Benutzer klickt den „Beleg generieren"Button 2. Wenn das Generieren und Speichern vom Kassenbeleg mit zusätzlichen TSE Informationen erfolgreich ist, wird eine Meldung im Dashboard angezeigt.
Nachbedingungen	- Die Kassenbeleg-Kollektion in der MongoDB-Datenbank enthält den neu hinzugefügten Kassenbeleg mit TSE Information. - Das TSE-Protokoll wird erfolgreich generiert und gespeichert

Tabelle 3.8: Kassenbeleg mit TSE Information erstellen

4 Konzeption

Im vorigen Kapitel wurden Anforderungen und die Spezifikationen definiert. In diesem Kapitel werden anhand von Kapitel 3 Lösungsansätze beschrieben.

4.1 Architektur

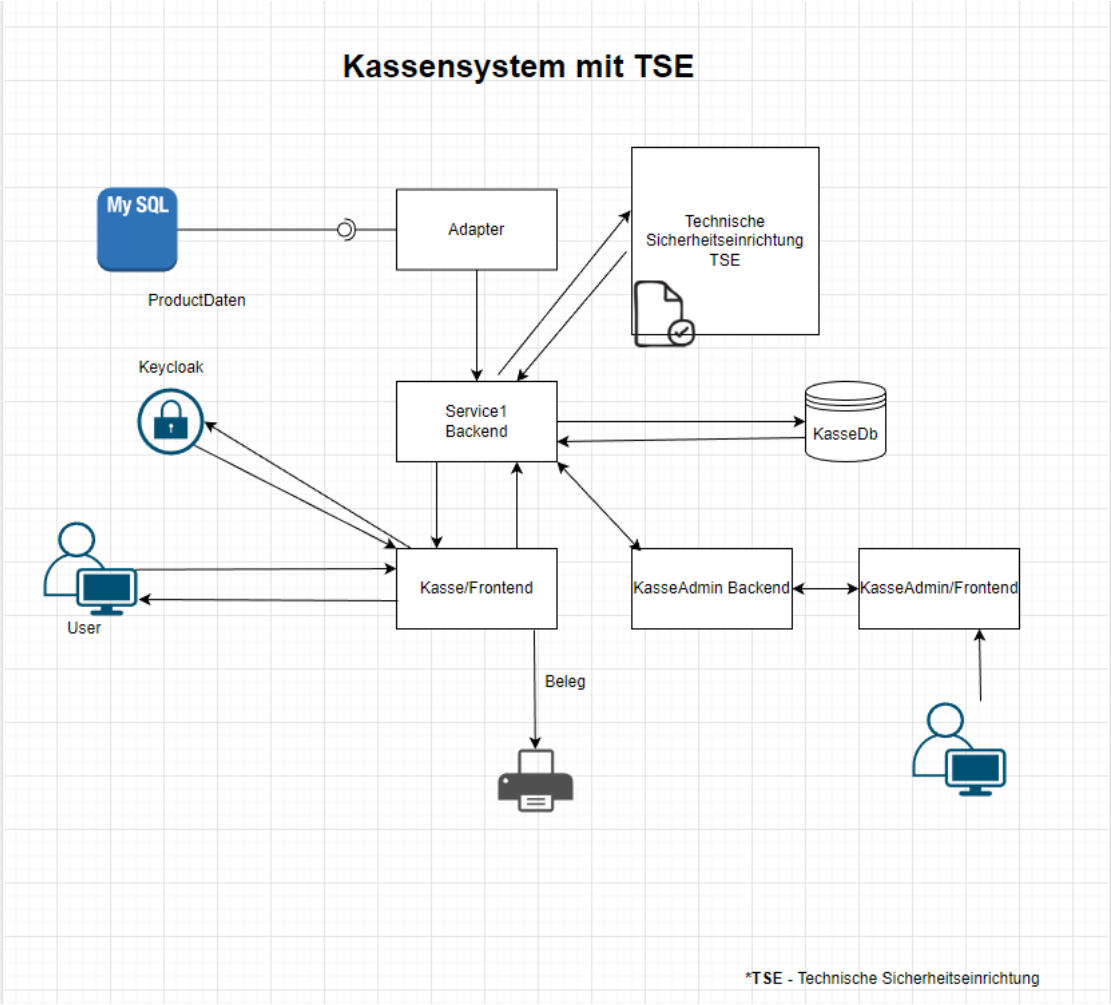

Abbildung 4.1: Entwurf des Kassensystems mit TSE Prototyp

Abbildung 4.1 zeigt einen Überblick des Kassensystem-Prototyps / der Architektur für das Kassensystem mit TSE. Der Protyp hat 6 Mikroservices. Zwei Frontend-Dashboards für jeweils Kassierer und Administrator, und vier Microservices. Das Service1Backend ist die einzige Schnittstelle mit der das Kassen-Dashboard kommuniziert um die gesamte Applikation sicherer zu machen. Alle Services kommunizieren durch Service1 Backend(POS-Service-Backend). Alle Microservices schicken eine Anfrage an den Haupt-POS-Service und von da wird die Anfrage weitergeleitet. Es gibt zwei Datenbanken, eine für die

Speicherung vom Kassensystem und die andere für die Produktverwaltung aus der Warenwirtschaft.

4.2 Lösungsstrategie

In diesem Abschnitt wird eine Übersicht über die gewählte Architekturstrategie gegeben und die Hauptkomponenten des Kassensystems werden vorgestellt. Das Kassensystem Dashboard kann mit der Drei-Schichten-Arkitektur realisiert werden.

- **Präsentationsschicht** - Das Kassensystem-Dashboard/ die Benutzeroberfläche für die Benutzer wird mithilfe des Angular-Frameworks entwickelt. Es ist zuständig für die Verwaltung von Artikeln und die Kassentransaktionen
- **Anwendungsschicht** - Das POS-System-Backend wird mit Spring Boot entwickelt und stellt die erforderlichen Endpunkte zur Kommunikation mit dem Frontend bereit.
- **Datenhaltungsschicht** - MongoDB wird für die Speicherung von Transaktionsdaten oder Warenkorb verwendet.

Für die Realisierung des TSE-Servers wird eine Client-Server-Architektur verwendet. Dafür wird die Socket-Programmierung von Java benutzt.

- **TSE-Server** - Der TSE-Server wird mit Socket-Programmierung für die sichere Kommunikation und die Protokollierung des Ablaufs realisiert. Er bietet verschiedene Dienste für den Client (POS-System-Backend) wie Signieren und Verifizieren von Signaturen usw.
- **TSE-Client** - Das POS-System-Backend(Client) sendet Anfragen an den Server und verarbeitet die Antworten vom Server.

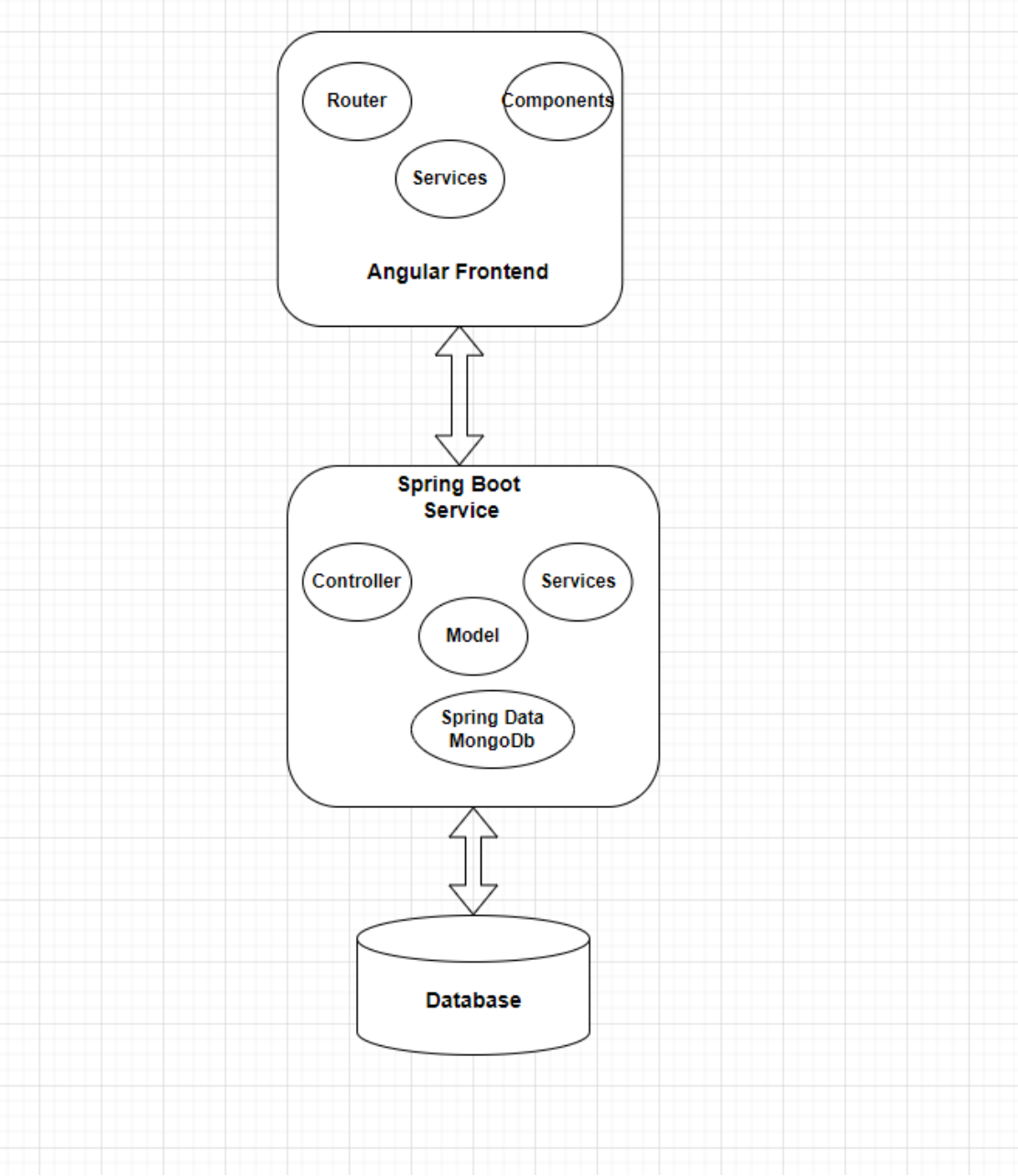

Abbildung 4.2: Drei-Sichten-Architektur

4.3 Laufzeitsicht

In diesem Abschnitt wird beschrieben, wie die verschiedenen Komponenten während der Laufzeit miteinander kommunizieren.

4.3.1 Warenkorb erstellen

In Abbildung 4.3 wird der Ablauf des Warenkorb-Erstellens angezeigt.

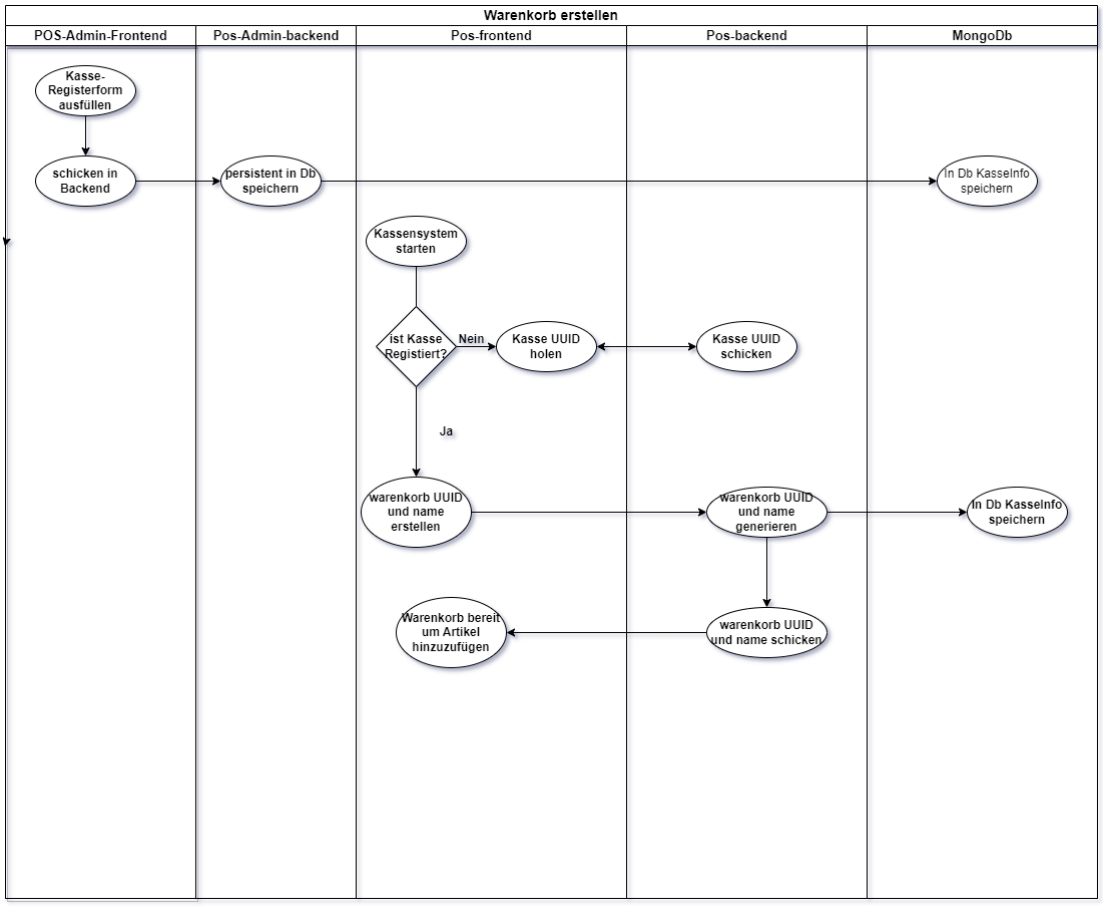

Abbildung 4.3: Ablauf Warenkorb erstellen

4.3.2 Artikel in warenkörb hinzufügen

In Abbildung 4.4 wird der Ablauf des Hinzufügens von Artikeln zum Warenkorb angezeigt.

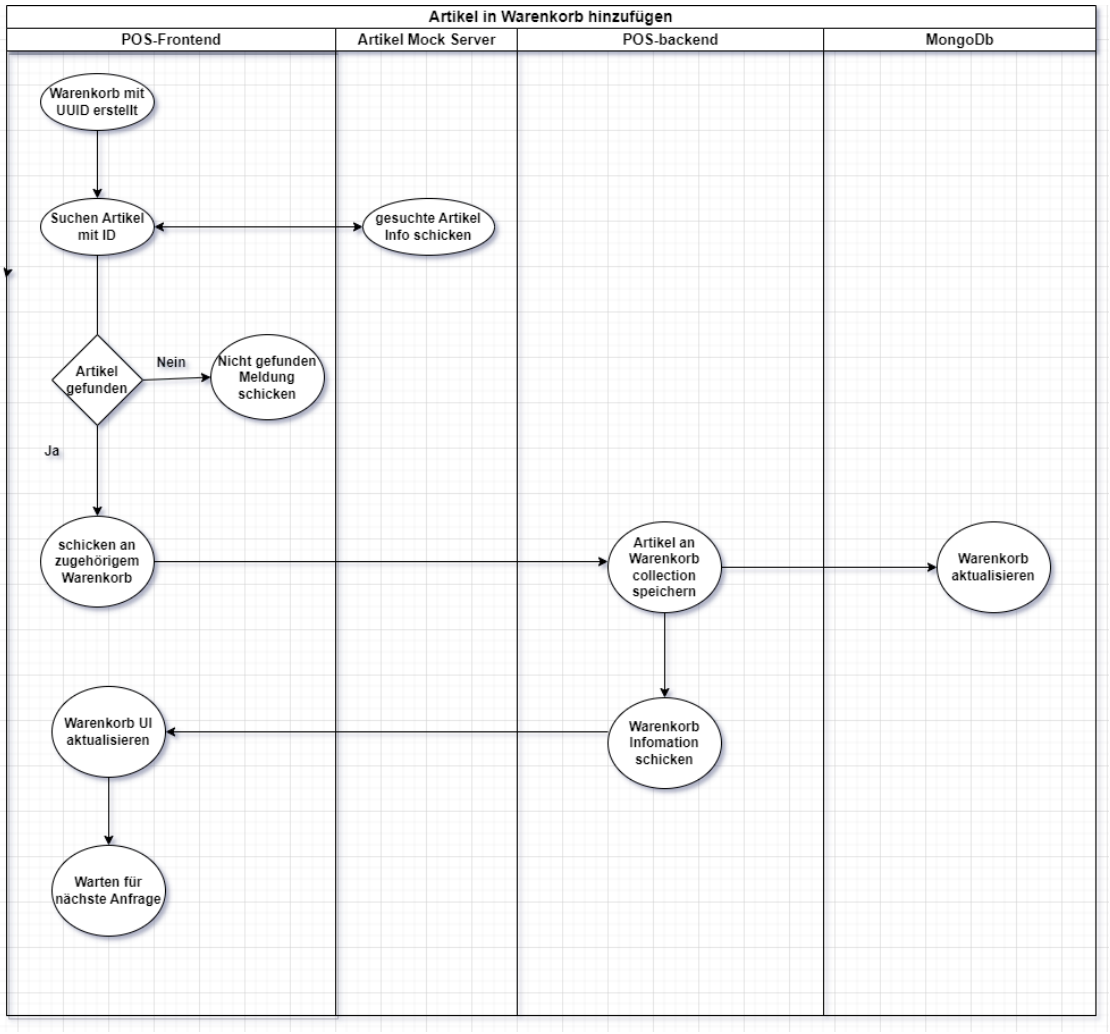

Abbildung 4.4: Ablauf Artikel in Warenkörb hinzufügen

4.3.3 Kassenbeleg mit TSE generieren

In Abbildung 4.5 wird der Ablauf der Kassenbeleggenerierung angezeigt.

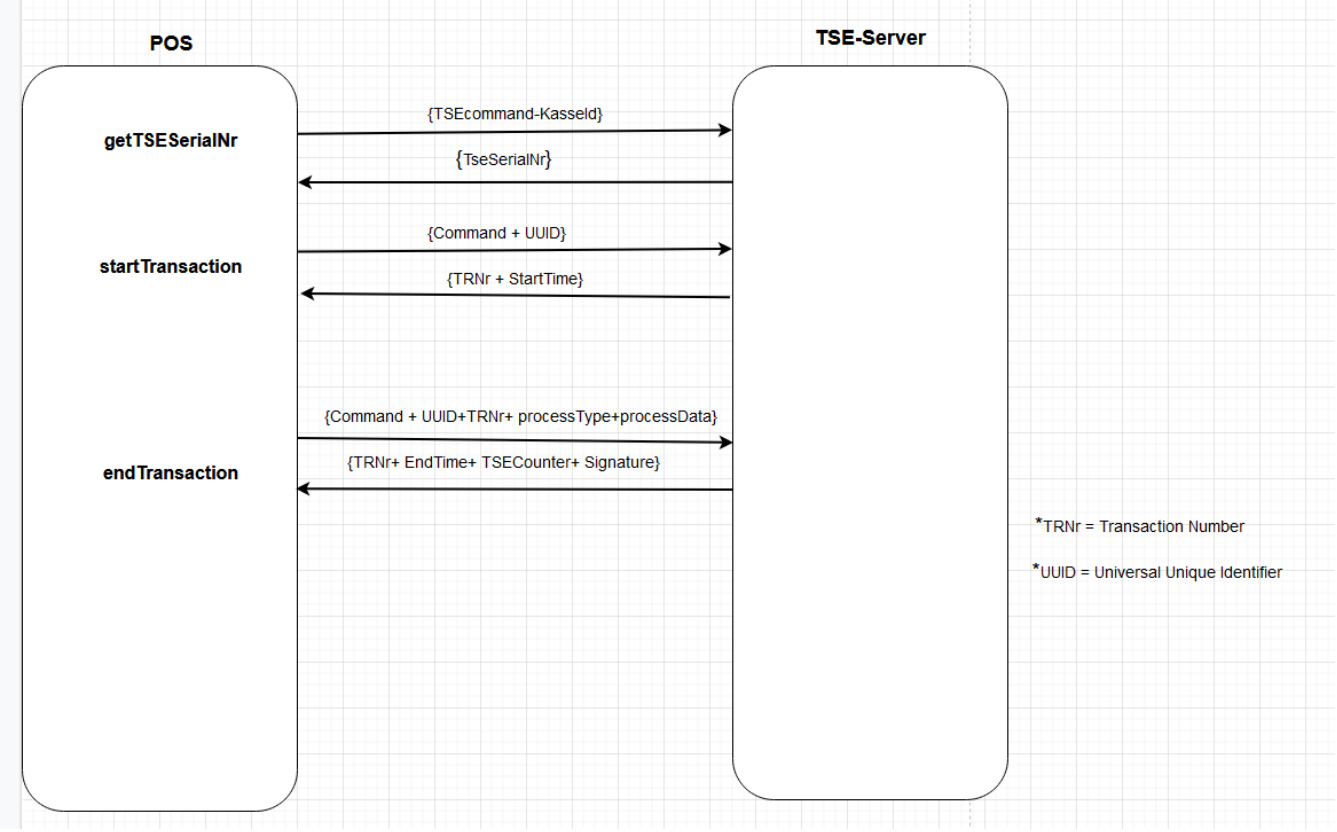

Abbildung 4.5: Signing Ablauf

4.3.4 Schnittstelle zu anderen Systemen

Die Kommunikation zwischen den verschiedenen Microservices erfolgt über HTTP und es wird das JSON-Datenformat verwendet.

4.3.5 Zeitstempel

Da alle Kassenabläufe protokolliert werden müssen, wird das Zeitstempel-Format YYYY-MM-DD verwendet, wobei Jahr, Monat und Tag jeweils angegeben sind.

4.3.6 Mock Json-Server

Der JSON-Server wird in das POS-System integriert, um das echte Verhalten eines Servers nachzuspielen, die für die Produktdatenverwaltung aus der Warenwirtschafft verant-

wörtlich ist. Dadurch ist es möglich, für das Prototyp eine Operation wie das Hinzufügen von Produkten zum Warenkorb usw. durchzuführen.

4.3.7 TSE-Server

Der TSE-Server soll eine Schnittstelle bereitstellen, die für Kryptographische Operationen wie Dokumente signieren, verifizieren, Protokollierung verantwortlich ist.

4.3.8 Mock Secure Element

Da für diese Arbeit kein echtes Secure Element zur Verfügung steht, wird ein Modul als Mock Secure Element erstellt . Das Mock-Secure-Element soll für die Speicherung von privaten Schlüssel/generierung usw. wie das echte Secure Element verantwortlich sein.

5 Umsetzung

Mithilfe der Anforderungen und Spezifikationen die in Kapital 3 definiert sind und den Konzeptionen in Kapitel 4 wird in diesem Kaptitel der Prototyp für das Kassensystem implementiert.

5.1 Mock JSON-Server für die Produkte

In diesem Abschnitt wurde die Implementierung eines Mock JSON Servers zur Unterstützung der Artikelverwaltung in einem Kassensystem beschrieben. Ein Mock JSON Server wurde verwendet, um die Funktionalität eines echten Servers nachzuspielen.

5.1.1 Erstellung der JSON-Datenstruktur

Zunächst wird eine JSON-Datei db.json erstellt, um eine Datenbasis für den Mock-Produkt-Server bereitzustellen. Diese JSON-Datei enthält Beispieldaten für verschiedene Eigenschaften von Artikeln, wie z.B. Artikelnummer, Name, Preis und Verfügbarkeit. Diese dient als Basisstruktur für die Artikel die in den Warenkorb hinzugefügt werden.

```
"products": [
    {
      "id": "123455",
      "code": "254fignf9",
      "name": "T-shit polo",
      "description": "Eine weise Polo Tshirt, große XXL usw.",
      "image": "https://secondella.de/wp-content/uploads/Damen-Schuhe-
Bottega-Veneta-Pumps-5.jpg",
      "price": 30,
      "category": "Accessories",
```

```
      "totalQuantity": 2,
      "inventoryStatus": "INSTOCK"
    }, ...
]
```

Wie oben zu sehen ist, besteht ein Artikel aus verschiedenen Attributen, die Schlüssel-Wert Paaren zugeordnet sind. Es ist eine Sammlung von Arktikeln die in eine Json-Struktur geschrieben sind.

5.1.2 Konfiguration des Mock JSON-Servers

Das Produkt JSON-Server für das Kassensystem wird in einem Docker-Container konfiguriert und gestartet, um eine isolierte Umgebung für die Entwicklung und das Testen zu realisieren. Die json.db Datei wird bei der Konfiguration verwendet. Für detaillierte Informationen wie ein Json-Server in einem Docker-Container startet ist auf der offiziellen Docker Hub-Seite des JSON-Server-Images https://hub.docker.com/r/codfish/json-server#full-example zu finden.

Funktionalitäten des Mock JSON-Servers

Der Mock JSON Server stellt alle notwedigen REST-APIs für die CRUD (Create, Read, Update, Delete) Operationen für die Artikelverwaltung bereit. Dieser Endpunkt dient dazu, Artikelinformationen aus dem Kassendashboard abzurufen für Operationen wie das Hinzufügen neuer Artikel in den Warenkorb, das Aktualisieren von Artikelinformationen (Preis) und das Entfernen von Artikeln aus der Datenbank.

Integration mit dem POS-System

Zuerst wurde der Mock JSON-Server erfolgreich in das Kassensystem integriert, um die Kommunikation zwischen dem Kassendashboard und dem JSON-Server aufzubauen. Es wird ein Service in Angular geschrieben, der eine Anfrage an das hautpt Backend schickt und wird von da aus an den JSON-Server weitergeleitet. Dadurch ist es möglich, für das Kassendashboard, einen Artikel zum Warenkorb hinzuzufügen. Wie es in Abbildung **??** zu sehen ist, wurde der Server erfolgreich durchgeführt.

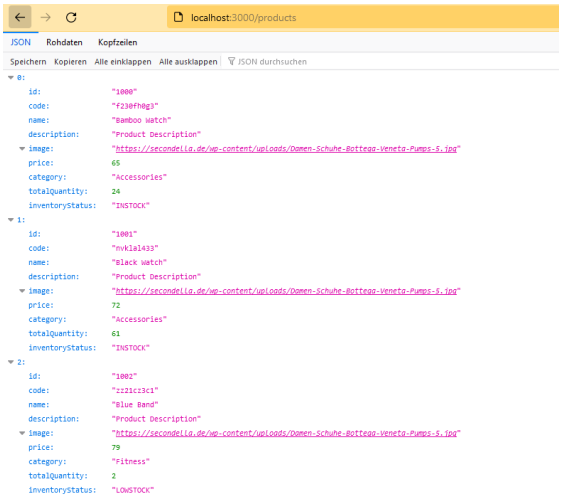

Abbildung 5.1: Ausführen von json-server in Docker

5.2 Implementierung des Kassen-Dashboard

Das GUI Dashboard für die Registrierkasse wird mit dem Angular Framework implementiert. Die wichstigsten Komponenten sind folgende:

- **Dashboard Component** - Wie in Abbildung 5.2 zu sehen ist, enthält das Dashboard Component das Basket Component und das Feature Component.
- **Basket Component** - Hier können Artikel zum Warenkorb hinzugefügt/ entfernt werden, Rabattaktionen durchgeführt werden
- **Feature Component** - Hier können verschiede Operation wie Bezahlungsarten ausgewählt, Kassenbons erstellt, der ausstehende Warenkorb bearbeitet usw. werden.

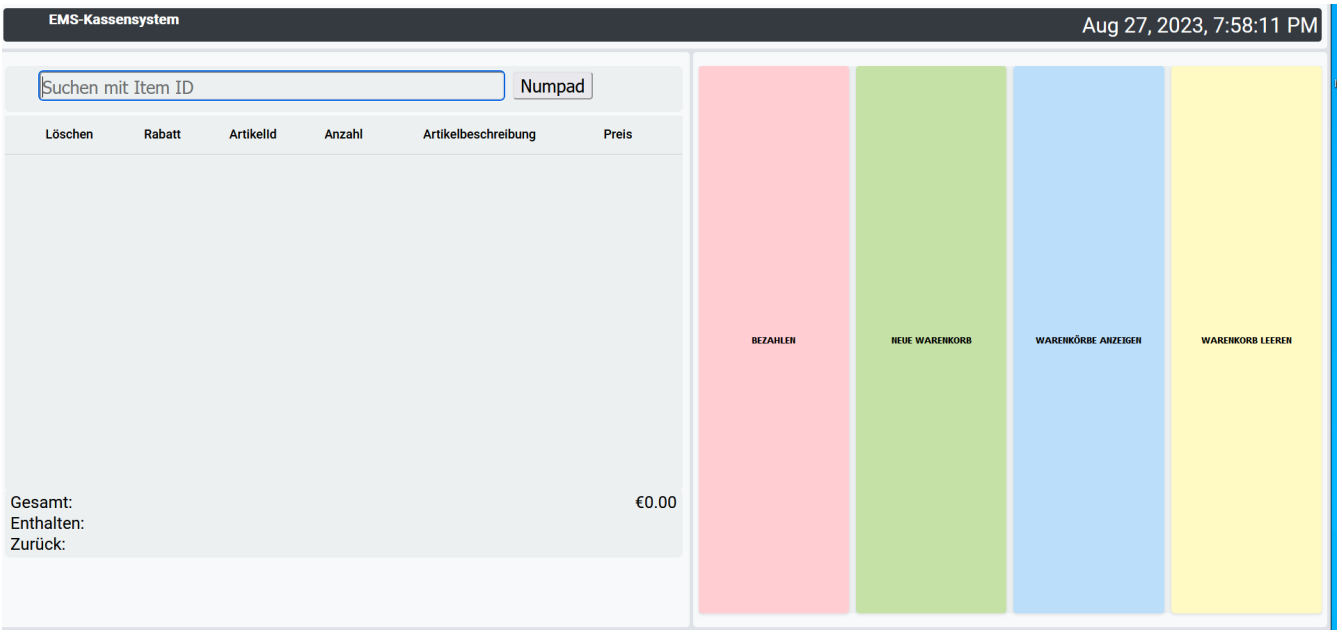

Abbildung 5.2: Kassensystem GUI

5.2.1 Verwendung von CSS Grid

Für die meisten Benutzeroberflächen wurde CSS Grid verwendet. CSS-Grid wurde entschieden um ein flexibles Design des Dashboards zu schaffen, sodass wie in Spezifikation 3.2.1 besschrieben ist, Flexibilität beizubehalten. Das Feature Component ist so implementiert, dass die Entwickler in der Zukunft leicht meherere Buttons hinzufügenn können, ohne das Design ändern zu müssen.

```css
.basket-grid {
  display: grid;
  width: 100%;
  height: calc(100vh - 300px);
  gap: 5px;
  grid-template-areas:
    'search search search search search'
    'content content content content content'
    'total total total total total'
    'numpad numpad numpad numpad numpad'
;
  grid-template-rows: 10% 70% 10% 10% auto;
}

.basker-search {
  grid-area: search;
  background-color: #ecf0f1;
  font-size: 25px;
  height: 100%;
  border-radius: 8px;
}
```

Abbildung 5.3: Grid CSS Beispiel

5.3 Implementation POS-System Service Backend

Das POS-Service-Backend wird in einer Spring Boot Framework Version 14.1.2 implementiert, das als Hauptschnittstelle für die Kommunikation zwischen dem Registrierkasse-Dashboard mit den anderen Microservices dient. Es wurden vierschiedene Controller für die Endpukte definiert, die die verschiedenen Operationen des Kassensystem erfüllen. Im POS-System-Service wurden die folgenden Controller implementiert:

- **Product Controller**
- **Basket Controller**
- **Receipt Controller**
- **RegisterKasse Controller**

5.3.1 Product Controller

Der Produkt-Controller ist zuständig für die Kommunikation zwischen dem Kassendashboard und dem Mock-Produkt JSON-Server. Das Kassendashboard übermittelt über den REST-API Anfragen an den Produkt-Controller und es wird zum JSON-Server weitergeleitet. Es werden alle CRUD Operationen bereitgestellt, um die Artikelinformationen zu verwalten.

5.3.2 Basket Controller

Der Basket-Controller ist verantwortlich für die Verwaltung des Warenkorbs. Das Dashboard kann über den REST-API mit dem Basket Controller kommunizieren. Der Produkte-Controller hat folgende Methoden definiert:

- Warenkorb erstellen - Es wird ein Warenkorb Objekt in der Warenkorb-Kollektion in MongoDb gespeichert mit einem Basket-Indentifier.
- Artikel in Warenkorb hinzufügen - Diese Methode benötigt ein Arikel Objekt mit einem Basket-Indentifier. Wenn das Basket-Identifier in der Datenbank vorhanden ist, dann wird das Artikel Objekt in den zugehörigen Warenkorb hinzugefügt.
- Warenkorb aktualisieren - Diese Methode kann verwendet werden, um die Artikel in dem Warenkorb zu aktualisieren. Sie benötigt die zu ändernden Artikel Objekte mit deren zugehörigen Basket-Identifiern.
- Warenkorb entfernen - Diese Methode dient dazu, um einen bestimmten Warenkorb aus der Dantenbank zu entfernen. Es erwartet einen Basket-Indentifier als Parameter.

5.3.3 Registerkassen Controller

Um ein Kassendashboard in einen betriebsbereiten Zustand zu bringen, muss erstmal die Kasse in dem POS-System registiert werden. Das heißt, dass jede Kasse eine UUID für die Kasse haben. Der Registierungprozess wird von dem Administrator mithilfe des Kasse-Admin-Dashboards verwaltet. Wenn eine Kasse keine UUID hat dann wird erstmal an das POS-System eine Anfrage geschickt. Der Registrierkasse Controller hat eine GET

Methode die überprüft, ob eine freie Kasse in der Datenbank vorhanden ist, wenn dies der Fall ist, dann wird das UUID an die Kasse weitergeleitet.

5.3.4 Receipt Controller

Der Receipt Controller ist zuständig für die Verwaltung von Kassenbelegen. Die wichtgste Aufgabe dieser Controller ist es, einen Kassenbeleg fertig zu stellen mit den notwendige Informationen aus dem TSE-Server, um einen Kassenbeleg zu generieren und in der Datenbank zu speichern. Er ist auch verantwortlich für den Verbindungsaufbau zwischen POS-System und TSE-Server. Grundsätzlich ist dieser Controller verantwortlich für die Verwaltung von Kassenbelegen.

```
@PostMapping(value = "/createNewBasket", produces = "application/json")
public ResponseEntity<Map<String, Object>> createNewBasket(@RequestBody KasseInfo kasseInfo) {
    HttpStatus status = null;
    String response = "";
    try {
        String basketIdentifier = basket1Service.createNewBasket(kasseInfo);
        status = HttpStatus.OK;
        response = basketIdentifier;
    } catch (BasketAlreadyExistsException e) {
        log.error(e.getMessage(), e);
        status = HttpStatus.NOT_FOUND;
        response = e.getMessage();
    }
    Map<String, Object> responseBody = new HashMap<>();
    responseBody.put("status", status);
    responseBody.put("msg", response);
    return ResponseEntity.status(status).body(responseBody);
}
```

Abbildung 5.4: POS Controller Beispiel

5.4 Implementation POS-Admin Service und Frontend

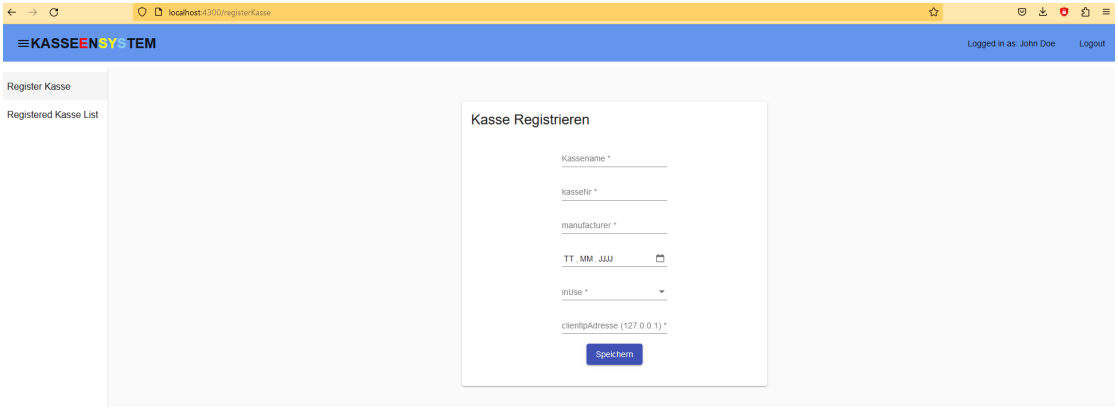

Abbildung 5.5: Kasssensystem-Admin Dashboard

Der POS-Admin-Service ist ein Microservice der in einem Spring Boot implementiert ist, der für die Verwaltung von Kassen-Konfigurationen zuständig ist. Dieser Service ist für die Admins gedacht, wo diese die Möglichkeit haben, die Kassen-Konfigurationen zu verwalten. Es wurde eine Benutzeroberfläche(POS-admin-Frontend) 5.5 mit Angular Framework entwickelt, wo die Admins die erforderlichen Informationen für die Registrierung von Kassen eingeben können oder die vorhandenen Daten in einer Tabelle einsehen können. Über die Frontend-Benutzeroberfläche können Admins folgende Informationen für jede Kasse eingeben, welche als KasseInfo Objekt in der Datenbank gespeichert werden:

- Kassenname
- Kassen UUID
- Hersteller
- Status(In Verwendung oder nicht)
- IP-Adresse der Kasse

Zuerst wird die IP-Adresse freigelassen und der Status(inUse) als ein boolischer Wert = Falsch gespeichert. Wenn eine Kasse registriert werden soll, dann wird erst geprüft, ob eine eine KasseInfo Objekt frei ist(inUse).Wenn eine freie Kasse vorhanden ist, dann

wird die IP-Addresse mit der aktuellen IP des Absender überschrieben und inUse Wert als TRUE gesetzt und das UUID and die Kasse geschickt.

5.5 Implementation TSE Server

Ein TSE Server ist ein Java Socket Server, der Spring Boot Framework verwendet.5.6 Es wurde ein Protokoll für die Kommunikation zwischen dem POS-System und dem TSE-Server definiert. Das Hauptziel dieser Server ist die Generierung und Protokollierung der Transaktionen. Es verwendet ein Secure Element Module um eine digitale Signatur zu generieren und zu verifizieren. Es ist auch zuständig für die Speicherung sowie die Generierung von privaten und öffentlichen Schlüsseln.

```
public SocketServer(@Value("8081") int port,
                    @Value("0.0.0.0") String ipAddress) {
    this.port = port;
    this.ipAddress = ipAddress;
    this.threadPool = Executors.newCachedThreadPool(); // Create a thread pool

    // Initialize the counter from the file or set it to 0 if the file doesn't exist
    int initialValueSignatureCounter = readCounterFromFile(signatureCounterFile);
    signatureCounter = new AtomicInteger(initialValueSignatureCounter);

    // Create a directory for transaction files if it doesn't exist
    File transactionDir = new File( pathname: "transaction");
    if (!transactionDir.exists()) {
        transactionDir.mkdir();
    }
}
```

Abbildung 5.6: TSE Server

5.5.1 Konfiguration

Der Socket-Server liest die Portnummer und die IP-Adresse aus der application.properties 5.7 Datei und initialisiert einen ThredPool, um die Client-Anfragen zu verarbeiten.

```
server.port=8080
# Set the port for the socket server
socket.server.port=8081

# Set the IP address for the socket server
socket.server.ip=0.0.0.0
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration
privatekey.filepath=private_with_ECDSA.key
publickey.filepath=public_with_ECDSA.key
kasseId=3CCEB42E-99CD-49D6-8B8A-536B8C3BE1D2
```

Abbildung 5.7: TSE Server konfiguration

5.5.2 Client-Handler

Für jede Anfrage der Clients wird ein seperater Thread erstellt, der die Anfragen des Clients bearbeitet. Das ermöglicht eine parallele Verarbeitung von mehrerer Anfragen. Es wird mit synchronized 5.8 Block sicher gestellt, dass nur ein Thread gleichzeitig auf die Methode **processMessage** zugreifen.5.8

```
private class ClientHandler implements Runnable {
    4 usages
    private final Socket clientSocket;

    1 usage
    public ClientHandler(Socket clientSocket) { this.clientSocket = clientSocket; }

    @Override
    public void run() {
        try (
                BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true)
        ) {
            String message = in.readLine(); // Read client's message
            System.out.println("Received message: " + message);

            // Process the message and generate a response sequentially
            //only one at a time to avoid having same transaction nr...
            synchronized (SocketServer.this) {
                String response = processMessage(message);
                out.println(response); // Send response to client
            }
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Abbildung 5.8: TSE Client Handler

5.5.3 Kommunikationsprotokoll

Der TSE-Server bietet ein Kommunikationsprotokoll für das Kassensystem, das die verschiedene Aktionen/Anfragen an die TSE-Server schicken kann. Folgende Aufrufcodes sind aktuell definiert:

Aktion A - die Generierung einer TransaktionsID und TSE-Beginn-Zeitstempel

Anfrageformat (Was der Client senden muss):

- **Befehl:** A
- **Parameter:** KasseId (Kassen-ID)
- **Beispielanfrage:** "A::12345"

Antwortformat (Was der TSE-Server zurück schickt):

- **Antwort:** Transaktionsnummer@Startzeit
  - **Transaktionsnummer:** Ein eindeutiger Zähler für den neuen Transaktion.
  - **Startzeit:** Der Zeitstempel, wann die Transaktion begonnen hat.
- **Beispielantwort:** "1@1631234567890"
- Bei Aktion A sendet der Client eine Anfrage an den TSE-Server, um eine neue Transaktion-ID und Transaktion Start-Zeitstempel zu erstellen. Der Client schickt eine eindeutige Kasse-ID als Parameter. Der TSE-Server prüft ob die Kasse-ID stimmt und verarbeitet diese weiter.
- Als nächstes wird eine Datei generiert und mit einem aktuellen Zähler("Transaktion-ID Counter"). Er wird in einer Datei gespeichert, um die TransaktionId persistence zu halten. Danach wird in einem Transaktions Verzeichnis eine neue Datei erstellt mit einem Dateinamen, z. B. Transaktion-1, wo 1 die aktuelle TransaktionId ist und wo der aktuelle Zeitstempel sowie die TransaktionNr gespeichert wird. Diese Datei ist wichtig für die Protokollierung und Überprüfung vom Transaktionsablauf.
- Der TSE-Server sendet eine Antwort an den Client zurück, der die TransaktionsNr und den Startzeitstempel der Transaktion enthält.

Aktion B - Signieren von Transaktionsdaten

Anfrageformat (Was der Client senden muss):

- **Befehl:** "B"
- **Parameter:** KasseId, Transaktionsnummer, Prozesstyp, Prozessdaten
- **Beispielanfrage:** "B::12345::1::Kreditkarte::100.00"

Antwortformat (Was der Server antwortet):

- **Antwort:** Transaktionsnummer@Endzeit@SignaturNr@Signatur
  - **Transaktionsnummer:** Die TransaktionsNr, die vorher von der Server geschickt wurde.
  - **Endzeit:** Der Zeitstempel, wann die Transaktion abgeschlossen wurde.
  - **SignaturNr:** Das eindeutige Signaturzähler
  - **Signatur:** Die digitale Signatur die generiert wurde
- **Beispielantwort:** "1@1631234567890@1@AbCdEFGHIDjgdfjlkdaf"
- Bei Aktion B sendet der Client eine Anfrage zur Signierung von Transaktionsdaten. Die Anfrage enthält die KasseId, die TransaktionNr, den Prozesstyp und die Prozessdaten.
- Zuerst wird wie bei Generierung der TransaktionNr erstmal eine Datei erstellt für Signaturzähler(Um der Zähler persistence zu halten)
- Der Server signiert die angegebenen Transaktionsdaten und erstellt eine neue digitale Signatur.
- Danach werden in dem Transaktions Verzeichnis, wo die Transaktionsprotokoll Datei hinterlegt ist, die richtige Datei mit der TransaktionNr gesucht und die weitere Protokollierungsinformation wie SignaturNr, digitale Signatur und Endezeitstempel konkateniert. Diese Datei ist dann vollständig protokolliert. Siehe Abbildung 5.9 Dieses Protokoll ist dann wichtig, um den Transaktionsablauf zu prüfen.
- Am Ende schickt der TSE-Server die TransaktionsNr, die Endzeit der Transaktion, eine eindeutige SignaturNr und die digitale Signatur im Base64 Format zurück.

- Diese digitale Signatur kann später benutz werden, um die Integrität des Kassenbelegs zu prüfen.

Aktion C - Überprüfen der digitale Signatur

Anfrageformat (Was der Client senden muss):

- **Befehl:** "C"
- **Parameter:** KasseId, digitale Signatur, Öffentlicher Schlüssel
- **Beispielanfrage:** "C::AdfhkErfckdh234Gdk::PublicKey123"

Antwortformat (Was der Server antwortet):

- **Antwort:** boolischer Wert als True oder False.
- Bei Aktion C sendet der Client eine Anfrage zur Überprüfung einer digitalen Signatur. Die Anfrage enthält die KasseId, digitale Signatur (Base64) und den öffentlichen Schlüssel (Er muss den Prüfern bekannt sein).
- Der TSE-Server überprüft die digitale Signatur mit Hilfe von Secure Element module.
- Als Antwort schickt der Server ob die Signatur gültig ist oder nicht.

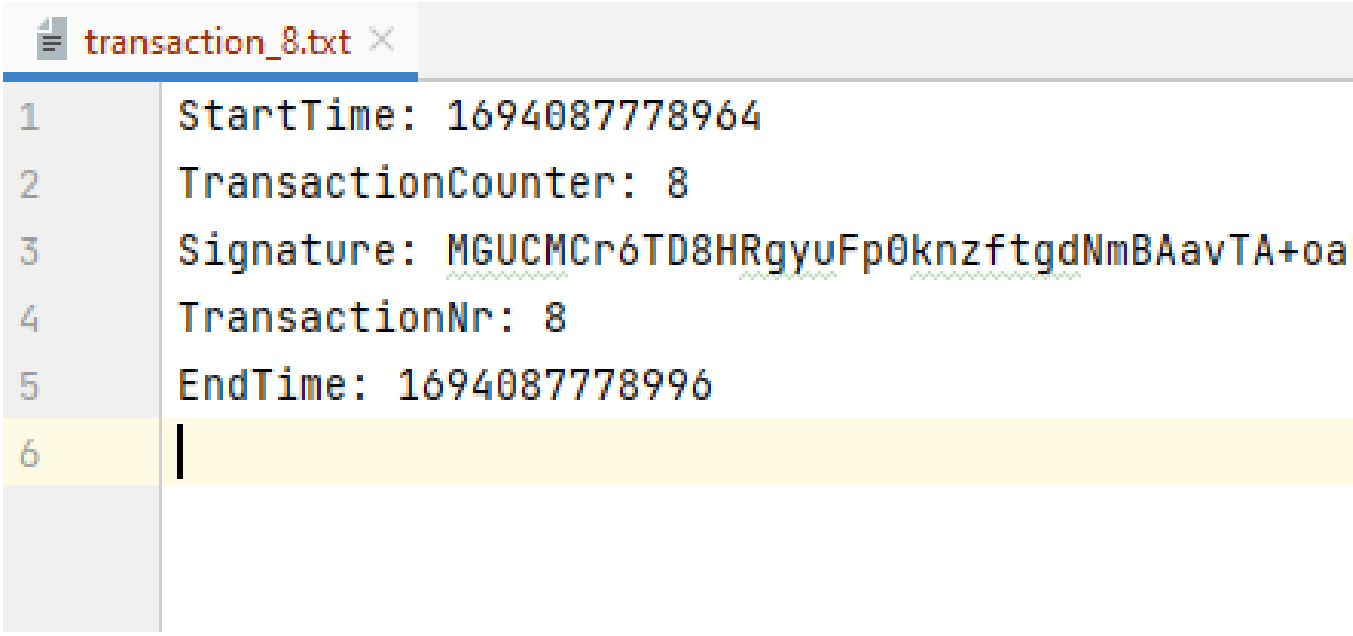

Abbildung 5.9: TSE protokoll

5.6 Implementation Mock Secure Element(SE)

Das Mock-Secure-Element Module wird auch in Java geschrieben um das Verhalten des Secure Element nachzuspielen. Es ist zuständig für die Generierung und die Überprüfung von digitalen Signaturen. Für die Kryptographischen Operationen wie das Generieren von Schlüsseln, Hashfunktion und Signatur Algorithmus wird die BouncyCastle-Provider Bibliothek benutzt. Es werden die notwendigen Parameter angegeben um die verschiedenen Operationen durchzuführen. Es gibt die folgenden Komponenten in diesem Module:

5.6.1 SignReceiptInterface

Diese Klasse hat zwei Methoden, signTheReceipt und verify.

signTheReceipt(String receiptInfo)

Beschreibung: Diese Methode ist verantwortlich für die Generierung der digitalen Signatur. **Parameter:** receiptInfo String die signiert werden soll **Rückgabewert:** Ein digitale Signatur in byte-Array.

```
2 usages
@Override
public byte[] signTheReceipt(String receiptInfo) {
    try {
        byte[] hashValue = hashMessageService.hashTheReceiptData(receiptInfo);
        Signature signature = Signature.getInstance(RequiredAlgorithm.SIGNING_ALGORITHM, new BouncyCastleProvider());
        signature.initSign(extractKey.readPrivateKey());
        signature.update(hashValue);
        return signature.sign();
    } catch (NoSuchAlgorithmException | InvalidKeyException | SignatureException | IOException e) {
        throw new RuntimeException(e);
    }

}
```

Abbildung 5.10: Signieren von Transaktionsdaten

Wie in Abbildung 5.10 zu sehen ist, wird erstmal der String, der als Parameter geschickt wurde, mit Hilfe der hasTheReceiptData Methode in einen Hash-Wert(Message-Digest) umgewandelt (mehr dazu in Unterkapitel 5.6.3). Mit dem vorhandenen Privat Schlüssel und Hash-Wert wird eine digitale Signature generiert. Als signatur-Algorithmus haben wir ECDSA als Parameter eingegeben. Es wird, wie in der Abbildung zu sehen ist, das Bouncy Castle Module verwendet.

verify(String receiptInfo, byte[] signature, PublicKey publicKey)

Beschreibung: Diese Methode prüft, ob die Signature gültig ist oder nicht

Parameter:

- `receiptInfo` - Die Transaktionsdaten, für die die Signatur überprüft werden soll.
- `signature` - Die digitale Signatur
- `publicKey` - Der öffentliche Schlüssel, der für die Überprüfung benötigt wird

Rückgabewert: Ein boolische Wert ob die Signature gültig ist oder nicht

```
@Override
public boolean verify(String receiptInfo, byte[] signature, PublicKey publicKey) {
    try {
        byte[] hashValue = hashMessageService.hashTheReceiptData(receiptInfo);
        Signature signatureInstance = Signature.getInstance(RequiredAlgorithm.SIGNING_ALGORITHM, new BouncyCastleProvider());
        signatureInstance.initVerify(publicKey);
        signatureInstance.update(hashValue);
        return signatureInstance.verify(signature);
    } catch (NoSuchAlgorithmException | InvalidKeyException | SignatureException | IOException e) {
        throw new RuntimeException(e);
    }
}
```

Abbildung 5.11: Überprüfen von Transaktionsdaten

5.6.2 GenerateKeyPair

Die GenerateKeyPair-Klasse ist verantwortlich für die Generierung von Schlüsselpaaren. Es gibt nur eine Methode in dieser Klasse.

generateECDSAKeypair

Beschreibung: Diese Methode generiert ein Schlüsselpaar nach den Parametern. Es wird auch die BouncyCastle Bibliothek verwendet.

Rückgabewert: Es wird eine Schlüsselpaar zurück geschickt, aber hier zu bemerken ist, dass das Schlüsselpaar auch in zwei Dateien gespeichert wird, die später benötigt werden.

```
1 usage
public KeyPair generate_ECDSA_Keypair() throws NoSuchAlgorithmException, InvalidAlgorithmParameterException {

    SecureRandom secureRandom = new SecureRandom();

    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(RequiredAlgorithm.SIGNING_ALGORITHM, new BouncyCastleProvider());
    ECGenParameterSpec ecGenParameterSpec = new ECGenParameterSpec(RequiredAlgorithm.EC_DOMAIN_PARAMETER);
    keyPairGenerator.initialize(ecGenParameterSpec, secureRandom);
    KeyPair keyPair = keyPairGenerator.generateKeyPair();
    PublicKey publicKey = keyPair.getPublic();
    PrivateKey privateKey = keyPair.getPrivate();
    try (FileOutputStream fs1 = new FileOutputStream( name: "public_with_ECDSA.key");
         FileOutputStream fs2 = new FileOutputStream( name: "private_with_ECDSA.key")) {
        fs1.write(publicKey.getEncoded());
        fs2.write(privateKey.getEncoded());
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    return keyPairGenerator.generateKeyPair();
}
```

Abbildung 5.12: Generierung von Schlüsselpaaren

5.6.3 HashMessageInterface

Die HashMessageServiceImp-Klasse ist verantwortlich für die Generierung vom Hash-Wert(Message Digest) in einer bestimmten Länge.

`hashTheReceiptData(String receiptInfo)`

Beschreibung: Bei dieser Methode wird aus einem einen Sting beliebiger Länge als Parameter mit Hilfe vom Hash-Algorithmus ein Byte-Array mit einer bestimmten Länge generiert.

Parameter:

- `receiptInfo` - String die gehasht werden sollen

Rückgabewert: ein Hash-Wert

5.7 Implementation Warenkorb erstellen und Artikel hinzufügen

Nach der Beschreibung der Anforderungen 3.3 und 3.4 wurde das Hinzufügen und Entfernen von Artikeln im Warenkorb erfolgreich implementiert.

```
private createNewBasket(basketName?: string) {

  const updatedBasketName = (basketName !== undefined && basketName !== '') ? basketName : 'Customer' + this.generateRandomUniqueNumber();
  this.paymentMethod = PaymentMethod.NOTSET
  this.sharedVariableService.setReceivedSum(0)
  this.sharedVariableService.setReturnSum(0)
  console.log(updatedBasketName)
  console.log(this.basketService.basketIdentifier.getValue())
  if (this.showUpdateBasketName == true) {
    this.basketService.updateBasketName(this.basketService.basketIdentifier.getValue(), updatedBasketName).subscribe( next: res => {
      console.log("basketname", basketName)
    })
  }
  this.basketService.getKasseInfoWithKassename().pipe(
    switchMap( project: item => {
      this.kasseInfo = item;
      if (item) {
        return this.basketService.createNewBasket().pipe(
          tap( next: response => {
            console.log(response.status)
            this.basketService.basketIdentifier.next(response.msg)
            this.basketService.fetchProducts();

            console.log(response.msg)
          }),
          catchError( selector: error => {
            console.error('HTTP request failed:', error);
            return throwError(error);
          })
        );
      } else {
        return EMPTY;
      }
    })
```

Abbildung 5.13: Warenkorb erzeugen

Beim Initialisieren des Frontends des Dashboards wird zuerst überprüft, ob die Kasse eine eindeutige Kassen-ID hat / ob sie bereits in der Datenbank registriert ist. Falls sie noch nicht registriert ist, wird nach der eindeutigen Kassen-ID gefragt. Die eindeutige Kassen-ID muss vom Admin bereitgestellt werden. Dafür gibt es ein Administrations-Frontend, in dem der Admin manuell die Kassenkonfigurations-Informationen angibt. Siehe Abbildung A.1 und Abbildung A.2. Ein Feld davon ist die IP-Adresse, die anfangs leer ist. Wenn die Registrierkasse eine Anfrage für die Kassen-ID macht, überprüft das POS-Backend, ob eine freie Kasse in der Datenbank vorhanden ist. Die Kasseninformationen enthalten ein Feld ÏnUse", das einen Booleschen Wert hat. Falls eine freie Kasse mit Ïn Verwendung = Falsch"verfügbar ist, wird zuerst die IP-Adresse mit der IP-Adresse des Frontends überschrieben. Dann wird die Kassen-ID an die Kasse gesendet.

Sobald die Registrierkasse bereit ist, kann ein neuer Warenkorb erstellt werden. Falls noch keine eindeutige Warenkorb-Kennung vorhanden ist, wird eine Anfrage an das Backend für die Warenkorb UUID gesendet (basketIdentifier). Wenn der Backend-Dienst die Anfrage erhält, wird eine eindeutige UUID und ein Warenkorbname generiert. Die Warenkorb Information wird in der Datenbank gespeichert, und das Frontend erhält

die eindeutige Warenkorb-UUID. Der Warenkorb ist nun bereit für das Hinzufügen von Artikeln. Siehe Abbildung A.6 und Abbildung A.7

Wie in Abbildung 5.2 zu sehen ist, gibt es ein Suchfeld, in dem nach Artikeln gesucht wird. Falls der Artikel gefunden wird, wird er gleichzeitig zum Warenkorb hinzugefügt. Beim Anklicken des Suchfelds gibt es zwei Optionen: Entweder kann der Barcode direkt gescannt oder die Artikelinformationen manuell eingegeben werden. Es gibt auch ein Costum-Numpad für die Touch Funktion.A.3 Wenn kein Artikel mit der gesuchten Artikel-ID eingegeben wird, erscheint die Meldung "Kein Artikel mit der ID XXX gefunden!"A.4.

5.7.1 Die parallele Generierung mehrerer Warenkörbe (ausstehende Kassen)

Nach der Beschreibung der Anforderung 3.5 wurde die Ausstehende Warenkorbzusammenstellung erfolgreich implementiert. Es gibt einen Button "Neuer Warenkorb", der es ermöglicht, einen neuen Warenkorb parallel zum bestehenden Warenkorb zu generieren. Beim Anklicken des 'Neuer Warenkorb' Buttons wird eine neue Eingabe-Dialogbox geöffnet 5.14, in der der User einen Namen für den aktuellen Warenkorb eingeben muss. Danach wird der Warenkorb in der Datenbank gespeichert und das Dasboard wird mit einer neue Warenkorbsicht aktualisiert. Wie in Abbildung 5.2 zu sehen ist, können die Warenkörbe, die noch nicht geschlossen sind, mit anklicken vom ßhow open Basket"Button wiederhergestellt und weiter bearbeitet werden.

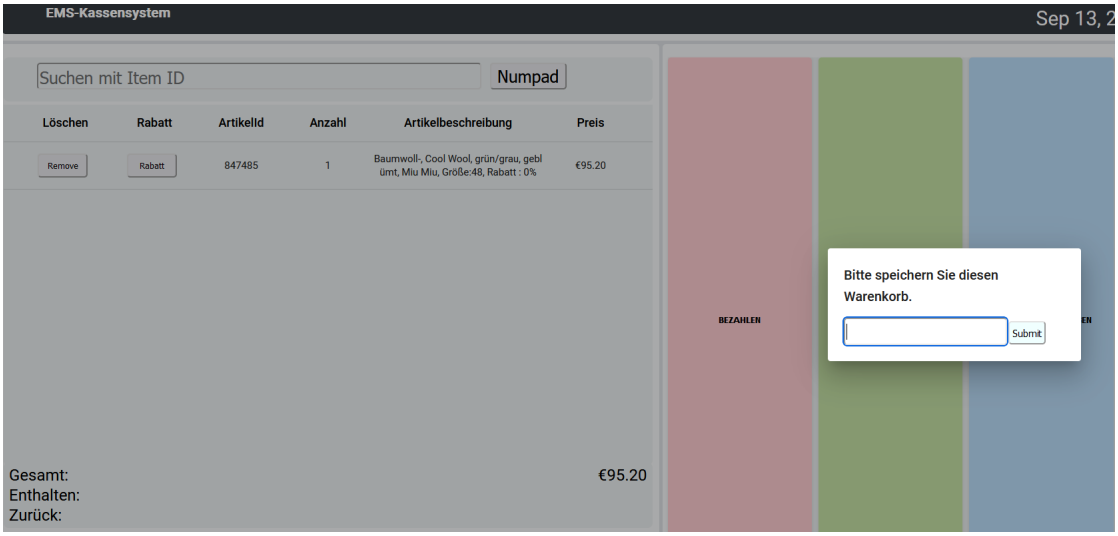

Abbildung 5.14: Die Parallel Generierung mehrerer Warenkörbe

5.7.2 Implementation Bezahlungsmethoden und Rabatt

Nach der Beschreibung der Anforderungen 3.6 und 3.2 wurden erfolgreich implementiert.

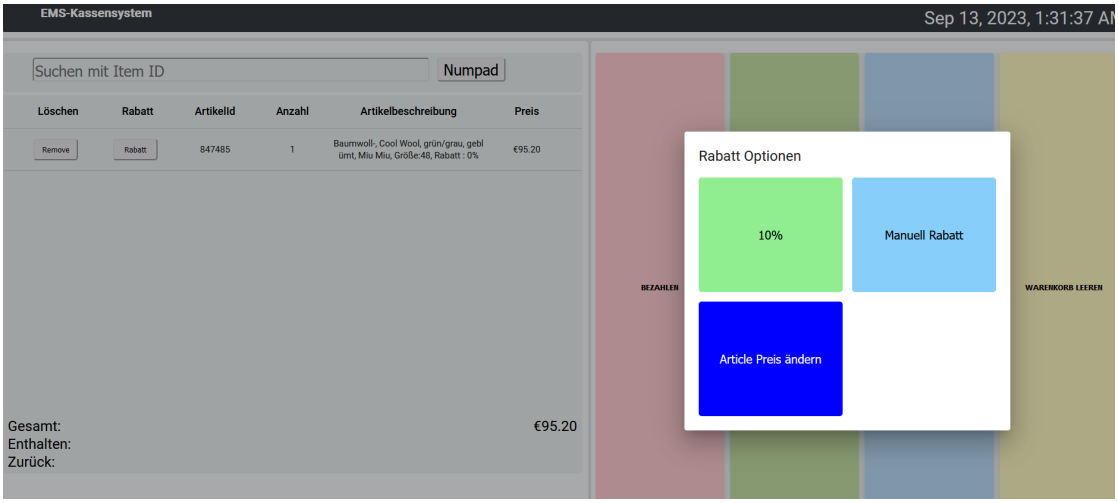

Abbildung 5.15: Rabatt Optionen

Wie in Abbildung 5.15 zu sehen ist, öffnet sich beim Klicken auf den "RabattButton ein Dialogfeld, das drei Optionen anbietet:

- 10 % - Der aktuelle Artikelpreis wird um 10 % reduziert, und die Warenkorbartikel werden mit dem neuen Preis aktualisiert.
- Manueller Rabatt - Hier kann ein Rabatt manuell eingegeben werden, um den aktuellen Artikelpreis zu ändern.
- Artikelpreis ändern - Der Artikelpreis wird direkt nach der Eingabe aktualisiert.

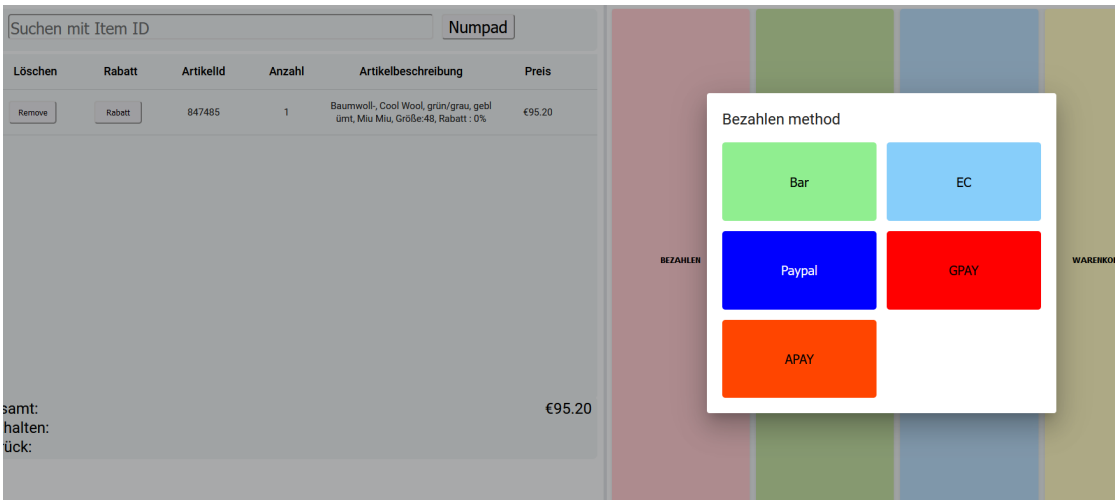

Abbildung 5.16: Bezahlungmethoden

Wie in Abbildung 5.16 zu sehen ist, öffnet sich beim klicken auf den "Bezahlungsmethoden"Button ein Dialogfeld, das mehrere Bezahlungsoptionen anbietet. Insbesondere ist zu beachten, dass beim Anklicken des Barzahlung-Buttons ein Eingabe-Dialogfeld geöffnet wird (siehe Abbildung A.3). Hier kann der vom Kunden gezahlte Bargeldbetrag eingegeben werden, und das Feld für den erhaltenen Betrag sowie das Rückgeld werden aktualisiert. Bei anderen Bezahlungsmethoden werden nur Bezeichnungen wie 'EC' oder 'GPAY' usw. in den Beleginformationen weitergegeben.Wenn eine Barzahlungsoption gewählt wird, kann es weitere Vorgänge geben. Es muss mindesten eine Option ausgewählt werden und beim Bargeld muss der eingegebene Beitrag höher oder gleich der Summe der Warenkorbartikel sein sonst wird eine Fehler angezeigt.

5.8 Implementation Kassenbon erstellen

Nach der Beschreibung der Anforderung 3.8 wurde die Generierung des Kassenbelegs erfolgreich implementiert. Wenn die Artikel erfolgreich zum Warenkorb hinzugefügt wurden und eine der vorhandenen Bezahlungsarten ausgewählt wurde, werden die Warenkorbinformationen an das POS-Service-Backend gesendet, um sie weiterzuverarbeiten und einen Kassenbeleg mit TSE zu signieren.

```
public void startConnection() throws UnknownHostException, IOException {
    clientSocket = new Socket(tseConfig.getIp(), tseConfig.getPort());
    out = new PrintWriter(clientSocket.getOutputStream(), autoFlush: true);
    in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
}
```

Abbildung 5.17: Starten von TSE Server Socket

Wie in Abbildung 5.18 zu sehen ist, gibt es eine Methode, die Informationen über einen Beleg (ReceiptInfo) verarbeitet. Diese Informationen werden als JSON-Objekt im Request-Body übergeben. Der Ablauf der Generierung eines Kassenbons ist unten beschrieben.

- Es wird ein leeres Bon-Objekt erstellt, das später mit vollständigen Informationen ausgefüllt wird.
- Es wird eine TSE-Konfiguration (TseConfig) erstellt, in der die notwendigen Informationen vom TSE-Server konfiguriert werden. Außerdem wird ein TSE-Client initialisiert, um die Kommunikation zwischen dem Kassensystem und dem TSE-Socket-Server aufzubauen, siehe: 5.17.
- Zuerst wird die Transaktion gestartet, wobei der Client eine eindeutige TSE-Transaktionsnummer und Startzeitstempel erhält.
- Sobald der Client die Transaktionsnummer erhalten hat, kann er mit dem Aufruf von 'finishTransaction' den Beleg signieren lassen und die TSE-Signatur sowie die TSE-Endzeit abrufen.
- Das Bon-Objekt wird mit Informationen aus dem Receiptinfo und der TSE-Transaktionsinformation aktualisiert und in der Datenbank gespeichert.

- Am Ende wird eine Meldung im Dashboard erscheinen mit einer "Beleg sucessfully generated"Nachricht.

```
@PostMapping(value = "/item")
public ResponseEntity<Map<String, Object>>  saveReceipt(@RequestBody ReceiptInfo receiptInfo) {
    HttpStatus status = null;
    String msg = "";

    try {
        Bon bon = bonPrinterService.createEmptyBon(receiptInfo.getBelegNr());

        TseConfig tseConfig = new TseConfig();
        TseClient tseClient = new TseClient(tseConfig);
        //tseConfig.setTseSeriennumer(tseClient.getSerialNumber(tseConfig.getKasse()));
        TseTransaction tseTA = tseClient.transactionStart(tseConfig.getKasse());
        tseTA.setTseSeriennummer(tseConfig.getTseSeriennumer());
        tseTA = tseClient.transactionFinish(tseConfig.getKasse(),tseTA);

        bon = bonPrinterService.fillBon(bon, receiptInfo, tseTA);
        receiptInfo.setTseTransaction(tseTA);
        receiptService.saveReceiptInfo(receiptInfo);
        log.info(bon.toString());

        status = HttpStatus.OK;

    } catch (IOException e) {
        log.error(e.getMessage(), e);
        status = HttpStatus.INTERNAL_SERVER_ERROR;
        msg = e.getMessage();
    }
    ApiResponse apiResponse = new ApiResponse(msg,status);
    Map<String, Object> responseBody = new HashMap<>();
    responseBody.put("status", status);
    responseBody.put("msg", msg);
    return ResponseEntity.status(status).body(responseBody);
}
```

Abbildung 5.18: Kassenbeleg generieren und speichern in Datenbank

6 Qualitätssicherung/Testing

In diesem Kapitel geht es um die Qualitätssicherung und Testing, das während der Arbeit gemacht wurde um einen sicheren Protyp zu implementieren.

6.1 Funktionstests

6.1.1 Unit-Tests

Junit Tests sind sehr wichtig um die Codes zu testen und überprüfen, ob die Klasse bzw. die Methoden die implementiert sind, so funktionieren wie es erwartet wird. Für den Kassensystem-Prototyp werden für wichtigste Klassen eine Junit test geschrieben und die einzelnen Methoden getestet.

```
    @BeforeEach
    void init() {
        kasse1 = "3CCEB42E-99CD-49D6-8B8A-536B8C3BE1D2";
        kasse2 = "28DC209D-A3D2-408D-AC7B-D2A16D0523EB";
        String ip = "0.0.0.0";
        int port = 8081;
        tseConfig = new TseConfig(ip, port);
    }
```

```
@Test
    public void runClientTransactionStartTest() throws UnknownHostException, IOException {
        System.out.println("----------------------------------------");

        TseClient client1 = new TseClient(tseConfig);
        client1.startConnection();

        String response = client1.sendMessage( msg: TseServerCommands.TRANSACTION_START +"::"+ kasse1);
        System.out.println("Kasse1: " + response);

        client1.stopConnection();

        int i = response.indexOf("@");
        String transactionNr = response.substring(0, i);
        System.out.println("transactionNr = " + transactionNr);
        System.out.println("Startzeit = " + new Date(Long.parseLong(response.substring( beginIndex: i+1))));
    }
```

Abbildung 6.1: Start transaktion Junit Test

Wie in Abbildung 6.1 zu sehen ist, wird ein Unit Test geschrieben, für die Methode "TransaktionStart". Da wird von einem TSE-Client eine Anfrage an den Server geschickt und als Ausgabe eine TransaktionNr und Zeitstempel (Start des Transakiton) erwartet. Bevor ein TSE-Client eine Abfrage schicken kann, sollte eine Verbindung zu den Server aufgebaut sein. Dafür wird die IP-Adresse und Port-Nr des TSE-Server benötigt. Wenn eine Verbindung erfolgreich ist, kann es nun eine Afrage an den Server schicken. Es wird als Argument ein Serverbefehl und die KasseId getrennt mit :: geschickt. Die Antwort wird auf der Konsole geschrieben. Da es bei jeder Anfrage eine neue TransaktionId und Zeitstempel geliefert wird, ist es schwer wie mit asserTrue zu testen aber bei der Konsolenausgabe kann es festgelegt werden, dass der TSEServer die richtige Antwort zurückschickt.

6.2 Continuous Integration und Continuous Deployment(CI/CD)

```
#stages of the pipeline
stages:
  - build
  - deploy

#image to use in the pipeline
image: docker:latest
#build stage
build:
  image: maven:latest
  stage: build
  script:
    - apk update
    - apk add nodejs npm
    - npm install -g @angular/cli
    - cd Secondella-Frontend
    - npm install
    - ng build
    - mvn clean package
    - docker build -t anitk/secondella-angular-service:latest

    #build for backend
    - cd ..
    - cd secondella-pos-system-service
    - mvn clean package
    - docker build -t anitk/secondella-pos-service:latest

deploy:
  stage: deploy
  script:
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
    - docker push anitk/secondella-angular-service:latest
    - docker push anitk/secondella-pos-service:latest
```

Abbildung 6.2: CI/CD mit gitlab

Wie in Abbildung 6.2 zu sehen ist , während dieser Arbeit wird eine CI/CD-Pipeline mit GitLab eingerichtet. Es ist hilfreich in dem Entwicklungsprozess rechtzeitig Fehler zu entdecken. Bei jedem Git-Push werden die vordefinierten Stages durchgeführt. Und es ist sehr hilfreich auch bei der Produktion und Test Umgebung.

6.3 Leistungstests mit Apache Jmeter

Leistungstests wurde mit Apache Jmeter gemacht. Apache jmeter ist eine open-source-software für Leistungsprüfung, Belastungstests, Skalierungstests usw. von Applikationen.[14]. Für den Kassensystem Prototyp wurde ein Leistungstests gemacht. Im folgenden wird der Ablauf des Leistungstests und das Ergebnis beschrieben. Ablauf des Leistungstests:
Datensatz aus dem Datenbank holen

- Thread Group erstellen - Es wurden 5 Threads im Intervall von 1 sekunde 100 mal konfiguriert
- Datensatz - Es gibt 2220 Kassenbelege in der Datenbank
- Get-Request - Es werden von 5 Threads in einer Sekunden 100 mal der gesamte Datensatz aus der Datenbank geholt.
- Bewertung - Wie in Abbildung 6.3 zu sehen ist, ist die durchschnittliche Zeit um den Datensatz aus der Datenbank zu holen 0.36 Sekunden. Keine Fehler sind aufgetreten.

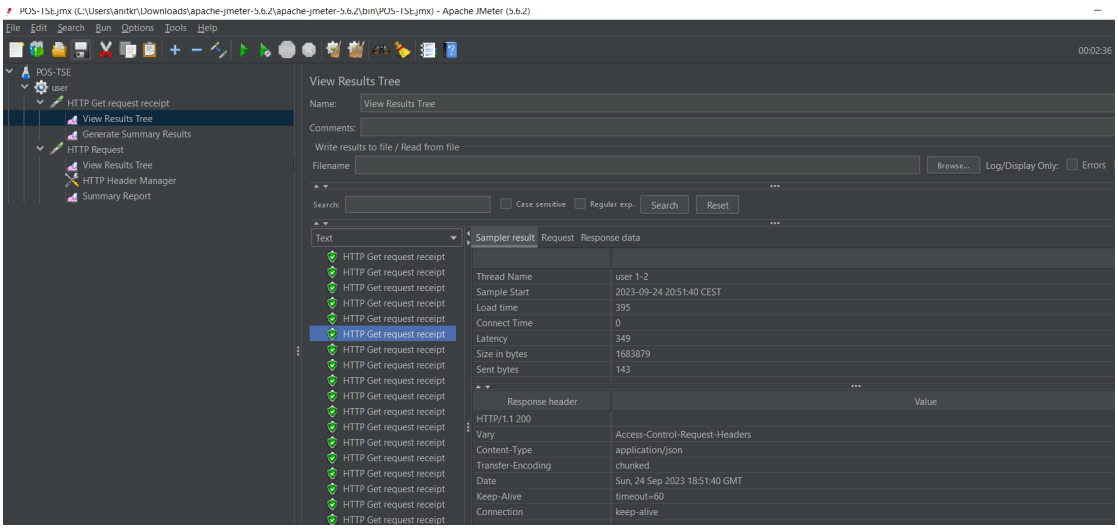

Abbildung 6.3: Leistungstests für alle Kassenbelege aus Datenbank holen

Kassenbeleg generieren Es wurde der gleiche Test für das Generieren vom Kassenbeleg (mit Signatur) gemacht. Es wurden zwei Testfälle geschrieben:

- 1 Thread pro 1 Sekunde 20 mal getestet

- 10 Thread pro 1 Sekunde 100 mal getestet

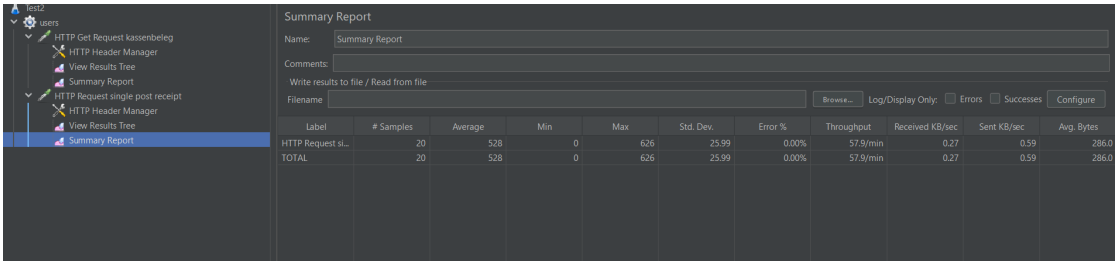

Abbildung 6.4: Kassenbeleg generieren Single Thread

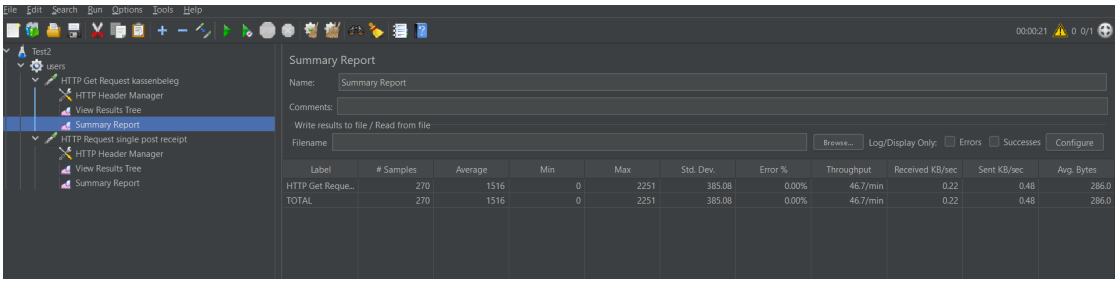

Abbildung 6.5: Kassenbeleg generieren 10 Thread

Bewertung Wie in Abbildung 6.4 und 6.5 zu sehen ist, ist beim Single-Thread die durchschnittliche Zeit, um einen Kassenbeleg zu generieren und zu speichern 0.528 sec, wobei es bei 10 Threads pro 1 sec für 100 Abläufe 1,5 sec dauert. Es liegt daran, dass bei der Generierung von Digital Signature von TSE immer ein Thread zurzeit bearbeit wird (synchronisierter Block 5.8). Es wird im realen Einsatz von Kassensystemen nur jeweils eine TSE pro Kasse geben. Das Ergebnis zeigt auch hier, dass keine Fehler bei dem Testablauf aufgetreten sind.

6.4 Kompatibilitätstests

- **Cross-Browser-Tests** Das Kassensystem wird in verschiedenen Webbrowsern wie Brave, Mozilla, und Google Chrome überprüft, um sicherzustellen, dass es unabhängig vom Webbrowser einwandfrei funktioniert.

- **Geräteübergreifende Tests** Um sicherzustellen, dass der Kassensystem-Prototyp auf allen Geräten wie Tablets und Mobilgeräten funktioniert, wurde die Kompatibilität auf verschiedenen Geräten getestet.

7 Zusammenfassung

In dieser Arbeit wurde ein Prototyp für ein integriertes Kassensystem mit einer technischen Sicherheitseinrichtung (TSE) implementiert. Das Hauptziel dieser Bachelorarbeit ist, die Sicherheit von Kassentransaktionen so zu gewährleisten, dass keine Manipulationen an den Daten möglich ist. Dafür wurden verschiedene Kryptographische Operationen umgesetzt. Während dieser Arbeit wurden die verschiedenen Anforderungen und Spezifikationen, die in Kapitel 3 beschrieben sind, größtenteils mit der Hilfe von Kapitel 4 (Konzeption) erfolgreich umgesetzt. Es wurde im Folgenden Kassenfunktionalität in dem Prototyp implementiert:

- Warenkorb erstellen
- Artikeln im Warenkorb hinzufügen
- Registieren von Kasse
- mehrere Warenkörbe gleichzeitig generieren
- Warenkorb leer machen
- Rabattaktion durchführen
- Artikeln aus dem Warenkorb entfernen
- Verschiedene Bezahlungsarten
- Kassenbeleg generieren mit dem TSE-Protokoll

Die Implementierung beginnt mit der Konfiguration eines Mock Json-Servers, der die Funktionalität eines echten Servers (z.B CRUD) hat, um eine Datenbasis für Artikel bereitzustellen. Dieser Mock Server wird in einer isolierten Entwicklungsumgebung mit

Hilfe von Docker ausgeführt. Das Kassensystem-Dashboard wird mithilfe des Angular-Frameworks realisiert und verwendet CSS-Grid, um ein flexibles Design für das Dashboard zu erstellen. Das Hauptdashboard besteht aus zwei Komponenten 1. Der Warenkorb und 2. Die verschiedenen Kassenfunktionen des Kassensystems. Aufgrund der Verwendung von CSS-Grid können in Zukunft viele weiteren Kassenfunktionen hinzugefügt werden, ohne das Hauptdesign ändern zu müssen. Es können beliebig viele Schaltflächen oder Funktionen auf der rechten Seite hinzugefügt werden(CSS-Grid), die Oberfläche wird automatisch angepasst. Das Backend des Kassensystems wird mithilfe von Spring Boot implementiert. Die Kommunikation zwischen dem Frontend und dem Backend erfolgt über eine REST-Schnittstelle. Da das Kassensystem ein komplexes System ist, wird im Verlauf dieses Projekts eine Microservices-Architektur umgesetzt, um die Anwendung in kleinere Services aufzuteilen. Damit wurden mehrere Services unabhängig voneinander implementiert und getestet. Der TSE-Server wird mithilfe einer Client-Server-Architektur modelliert und es wurde in Java-Socket-Programmierung verwendet. Die Entscheidung, einen Socket-Server statt einer REST-API zu verwenden, basiert auf der Anforderungen, ein eigenes Protokoll zu definieren und eine schnelle Kommunikation mit geringer Latenz zu schaffen. Um die TSE zu implementieren, wird ein Secure Element benötigt, um die digitale Signatur zu generieren und den privaten Schlüssel sicher zu speichern. Normalerweise wird dieser in einer physischen Hardware/Gerät in Form von SD-Karten, USB-Geräten usw. gespeichert sein, wo der Hersteller den privaten Schlüssel während der Herstellung speichert. Da jedoch während dieser Arbeit keine echtes Secure Element zu Verfügung stand, wurde ein simuliertes Secure Element erstellt, das in Java geschrieben wurde, das sich wie ein echtes Secure Element verhält. Das simulierte SE hat Funktionen wie das Generieren von Signaturen, Schlüssel speichern und Singnatur verifizieren.

Am Ende dieser Arbeit wurde erfolgreich ein funktionerender Prototyp eines Kassensystems implementiert, entsprechend allen Anforderugen. Die Integration des TSE-Servers hat gezeigt, dass mit digitaler Signatur und Secure Element eine Sicherheitslücke zur Datenmanipulation in der Transaktionsaufzeichnung verhindert werden kann. Dieser entwickelte Prototyp des Kassensystems entspricht auch den gesetzlichen Anforderungen für die zukünftigen modernen Kassensysteme in Deutschland. Diese Arbeit hat gezeigt, wie vorhandene Kassensysteme mithilfe kryptographischer Verfahren die Sicherheit und Vertraulichkeit der Registerkasse verbessern kann, was für Unternehmen, Kunden und Behörden auch sehr wichtig ist. Es ist auch anzumerken, dass eine solches Sicherheitsverfahren wie TSE nicht nur in einem Kassensystem, sondern auch in anderen Bereichen, die eine digitale Aufzeichnung haben, verwendet werden kann. Für das Prototyp gab es

6 Microservices. Aus der Entwicklung dieses Prototyps lässt sich lernen, dass der Verwaltungsaufwand von Services sehr hoch sein kann. Bei der TSE wurde der ECDSA als digitaler Signaturalgorithmus verwendet, weil er im Vergleich zu anderen Algorithmen wie RSA eine kleinere Schlüsselgröße hat, bei vergleichbarer Sicherheit. Für die Kasse braucht man schnelle Funktionen, wobei ECDSA eine kurzere Schlüssellänge hat d.h eine schnelle Generierung und Verifizierung der Signatur, weniger Speicher. Der Hauptgrund für die Entscheidung von ECDSA ist, dass es der vom National Institute of Standards and Technology (NIST)[25] und dem Bundesamt für Sicherheit in der Informationstechnik (BSI) empfohlene Algorithmus für digitale Signaturen ist. Für die Hash-Funktion bei digitalen Signaturen wird SHA-2 verwendet, da sie sowohl sicher als auch schneller ist. Wenn der Kassenbeleg erfolgreich mit einer digitalen Signatur signiert und gespeichert ist, werden auch gleichzeitig in einem Verzeichnis alle Tranksaktionenprotokolle in einer Datei aufgezeichnet, die dann später nach Bedarf exportiert und überprüft werden kann. Für den Prototyp war es wichtig, dass der generierte Kassenbeleg zu einem späteren Zeitpunkt nicht manipuliert werden kann. Um seine Echtheit zu überprüfen gibt zwei Optioenen

- Die TSE Protokoll Verzeichnis exportieren - da kann man den Kassenbeleg nach TSENr, Zeitstempel, SignaturNr prüfen
- Signatur verifizieren - es kann eine Anfrage an Server gestellt werden ob die Signatur gültig ist.

Für die Überprüfung wird ein öffentlicher Schlüssel benötigt (muss dem Prüfer bekannt sein).

7.1 Ausblick

In Rahmen dieser Arbeit wurde ein Basis-Prototyp eines Kassensystems mit einer technischen Sicherheitseinrichtung entwickelt. Das Prototyp hat noch weitere Verbesserungsmöglichkeiten, die noch weiter entwickelt und erforscht werden können.

- **Authentifizierung und Autorisierung:** Da das Kassensystem von verschiedenen Personen wie z. B. Kassierern und Administratoren genutzt wird, ist es notwendig, zu protokollieren, wer was getan hat. Eine Idee wäre, Keycloak zu verwenden.

- **Protokollierung des Tagesberichts:** Für einen Händler ist es sehr wichtig, einen Überblick über seinen Tages/Monatsablauf zu haben. Eine Tagesbericht wäre hilfreich.
- **wenn der TSE-Server nicht erreichbar ist:** Aufgrund des Zeitrahmens wurde dieser Fall bisher nicht abgedeckt. Es kann natürlich vorkommen, dass der TSE-Server nicht erreichbar ist. Das Kassensystem muss in einem solchen Fall eine Lösung bieten, damit die Arbeit fortgesetzt werden kann.
- **QR-Code für Beleg generieren :** Barcode für Beleg generieren - für den Fall, dass ein Geschäft auf einer Messe oder außerhalb des Ladengeschäfts stattfindet. In solchen Fällen ist es wichtig, dem Kunden den Beleg (digital) zur Verfügung zu stellen- wenn ein Kunde keinen Papierbeleg will, könnte eine Alternative ein QR-Code sein.
- **Cloud-Basierte Lösung:** - Realisierung eines vorhandenen Kassensystems in einer Cloud-Plattform. Langsam steigen die meisten Unternehmen auf Cloud-Basierte Dienste um. Es könnte auch da geforscht werden bezüglich einer Cloud-TSE.

8 Acronyms

CSS Cascading Style Sheets

GUI Graphic User Interface

HTML Hyper Text Markup Language

UI User Interface

TSE Technischen Sicherheitseinrichtung

SE Secure Element

POS Point Of Sale

AO Abgabeordnung

ECR Electronic Cash Register

API Application Programming Interfaces

REST Representational State Transfer

KassenSichV Kassensicherungsverordnung

GoBD Grundsätze zur ordnungsmäßigen Führung und Aufbewahrung von Büchern, Aufzeichnungen und Unterlagen in elektronischer Form sowie zum Datenzugriff

JSON JavaScript Object Notation

YAML YAML Ain't Markup Language

BSI Bundesamt für Sicherheit in der Informationstechnik

SHA Secure Hash Algorithm

NIST The National Institute of Standards and Technology

ECDSA Elliptic Curve Digital Signature Algorithm

RSA Rivest-Shamir-Adleman

VM Virtual Machine

SPA Single-Page-Application

BSON Binary JSON

UUID Universally Unique Identifier

IP Internet Protocol

CI/CD Continuous Integration/ Continuous Deployment

Literaturverzeichnis

[1] Abidi, Abdessalem ; Bouallegue, Belgacem ; Kahri, Fatma: *Implementation of elliptic curve digital signature algorithm (ECDSA)*. 2014. – 1–6 S

[2] Adam Hayes, Timothy l.: What Is a POS System and How Does It Work? In: *Investopedia* (28.04.2023). – URL https://www.investopedia.com/terms/p/point-of-sale.asp

[3] Adams, D.: *Douglas Adams Trilogy/Hitchhiker's Guide to Galaxy/Restaurant at the End of the Universe/Life, the Universe and Everything*. Random House Incorporated, 1995. – URL https://books.google.de/books?id=bMWkPQAACAAJ. – ISBN 9780345397041

[4] Alshuqayran, Nuha ; Ali, Nour ; Evans, Roger: A Systematic Mapping Study in Microservice Architecture. In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2016, S. 44–51

[5] Anderson, Charles: Docker [Software engineering]. In: *IEEE Software* 32 (2015), Nr. 3, S. 102–c3

[6] Angular: *Angular Documentation*. 2023. – URL https://angular.io/guide/what-is-angulars

[7] Arcucci, Isabella: Patent auf die Registrierkasse. In: *Bayerischer Rundfunk* (2014). – URL https://www.br.de/radio/bayern2/sendungen/kalenderblatt/0411-patent-registrierkasse-james-ritty-ncr-100.html#:~:text=James%20Ritty%20hie%C3%9F%20der%20Mann,in%20seiner%20Kasse%20klingelte%20es.

[8] Bhat, Bawna ; Ali, Abdul W. ; Gupta, Apurva: DES and AES performance evaluation. 2015. – Forschungsbericht. Info über AES und DES

[9] Boettiger, Carl: An Introduction to Docker for Reproducible Research. In: *SIGOPS Oper. Syst. Rev.* 49 (2015), jan, Nr. 1, S. 71–79. – URL https://doi.org/10.1145/2723872.2723882. – ISSN 0163-5980

[10] Boot, Spring: *Spring Boot Documentation*. 2023. – URL https://spring.io/projects/spring-framework

[11] Bouncy Castle, The L. of the: Bouncy Castle - A lightweight cryptography API for Java / The Legion of the Bouncy Castle. URL https://www.bouncycastle.org, 2023. – Forschungsbericht

[12] Chernick, C. M.: Federal S/MIME V3 Client Profile. In: *National Institure of Standards and Technology NIST* (November 2002). – URL https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-49.pdf

[13] Finanzen, Bundesministerium der: Gesetzt zum Schutz vor Manipulationen an digitalen Grundaufzeichnungen. In: *Bundesministerium der Finanzen* (2016). – URL https://www.bundesfinanzministerium.de/Content/DE/Gesetzestexte/Gesetze_Gesetzesvorhaben/Abteilungen/Abteilung_IV/18_Legislaturperiode/Gesetze_Verordnungen/2016-12-28-Kassenmanipulationsschutzgestz/0-Gesetz.html

[14] foundation, The A. software: Apache Jmeter. In: *Apache software foundation*. – URL https://jmeter.apache.org/index.html

[15] Gillis, Alexander S.: MongoDB / TechTarget. URL https://www.techtarget.com/searchdatamanagement/definition/MongoDB, 2023. – Forschungsbericht

[16] H, Jeremy: 4 Microservice Example. In: *Dream factory* (2023)

[17] Hausmann, Roger: Die Registrierkasse: von der Diebstahlsicherung zum digitalen Tausendsassa / Swisscom(Schweiz) AG. URL https://www.swisscom.ch/de/b2bmag/data-driven-technologies/kassensysteme-pos-geschichte/, 31.Mai 2021. – Forschungsbericht

[18] Heinig, Ian: The History of the POS System in Better Business Management / Hubworks. URL https://hubworks.com/blog/the-history-of-the-pos-system-in-better-business-management.html, 2020. – Forschungsbericht

[19] Informationsfreiheit, Bundesbeauftragter für den Datenschutz und die: Was bedeutet GoBD / Bundesbeauftragter für den Datenschutz und die Informationsfreiheit. 2018. – Forschungsbericht. https://www.bfdi.bund.de/DE/Buerger/Inhalte/Finanzen-Steuern/ABC_GoBD.html

[20] Informationstechnik (BSI), Bundesamt für Sicherheit in der: Technische Richtlinie BSI TR-03153 / Bundesamt für Sicherheit in der Informationstechnik (BSI). 2018. – Forschungsbericht. Info über TSE

[21] Kaur, Ravneet ; Kaur, Amandeep: *Digital Signature*. Assistant Professors in the Department of Computer Science, SDSPM College For Women, Rayya(Asr), Guru Nanak Dev University, India. Year of publication

[22] Lawson, Katie: What Are Single Page Applications and Why Do People Like Them So Much? bloomreach, 16.09.2022. – Forschungsbericht. – URL https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application

[23] Masse, Mark: *REST API design rulebook: designing consistent RESTful web service interfaces*. O'Reilly Media, Inc., 2011. – URL https://pepa.holla.cz/wp-content/uploads/2016/01/REST-API-Design-Rulebook.pdf

[24] Mohammed N. Alenezi, Hannen Alabdulrazzaq und Nada Q . M.: *Symmetric Encryption Algorithms: Review and Evaluation study*. 2020

[25] NIST: Digital Signatures / NIST. URL https://csrc.nist.gov/Projects/digital-signatures, 2017. – Forschungsbericht

[26] OktoPOS: Kassensicherungsverordnung / OktoPOS. URL https://kassensichv.com/impressum.html. – Forschungsbericht

[27] Penard, Wouter ; Werkhoven, Tim van: On the secure hash algorithm family. In: *Cryptography in context* (2008), S. 1–18

[28] RedHat: What is YAML. In: *Red Hat* (3. März 2023). – URL https://www.redhat.com/en/topics/automation/what-is-yaml

[29] Rolou Lyn R. Maata, Balaji Sudramurthy und Alrence H.: Design and Implementation of Client-Server Based Application using Socket Programming in a Distributed Computing Environment. 2017 IEEE International Conference on Computational Intelligence and Computing Research, 2117. – Forschungsbericht

[30] Schröder, Chris: Fiskalisierung – Was bedeutet das eigentlich? / snabble. 23.03.2022. – Forschungsbericht

[31] Smith, Ben: *Beginning JSON Learn the preferred data format of the Web*. Apress, 2015. – 38–40 S

[32] Steuern, Bayerisches L. für: Besondere Ordnungsvorschriften für die Buchführung und für Aufzeichnungen mittels elektronischer Aufzeichnungssysteme. In: *Finanzamt München*. – URL https://www.finanzamt.bayern.de/Informationen/Steuerinfos/Weitere_Themen/Elektronische_Kassensysteme/default.php?f=LfSt

[33] Subramanya, S.R. ; Yi, B.K.: Digital signatures. In: *IEEE Potentials* 25 (2006), Nr. 2, S. 5–8

[34] Vauclair, Marc: *Secure Element*. S. 1115–1116. In: Tilborg, Henk C. A. van (Hrsg.) ; Jajodia, Sushil (Hrsg.): *Encyclopedia of Cryptography and Security*. Boston, MA : Springer US, 2011. – URL https://doi.org/10.1007/978-1-4419-5906-5_303. – ISBN 978-1-4419-5906-5

[35] Volle, Adam: Web application. In: *Britannica* (2022). – URL https://www.britannica.com/topic/Web-application

A Anhang

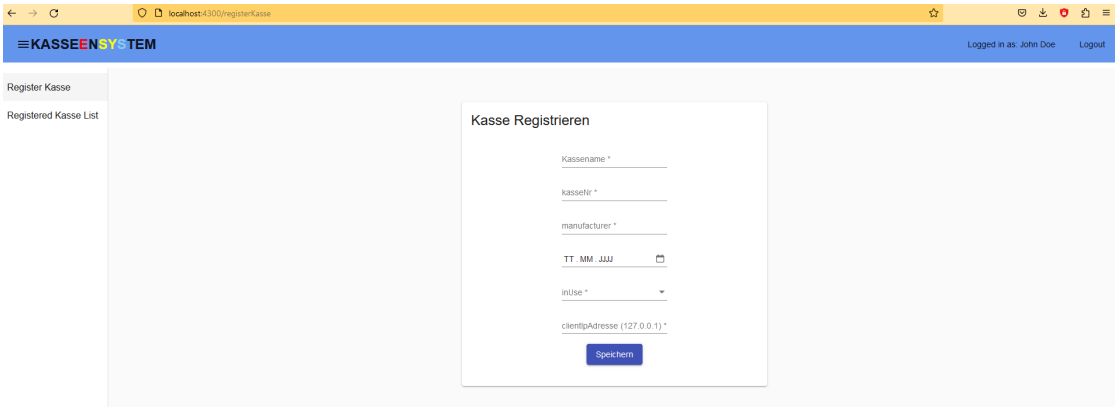

Abbildung A.1: Admin GUI

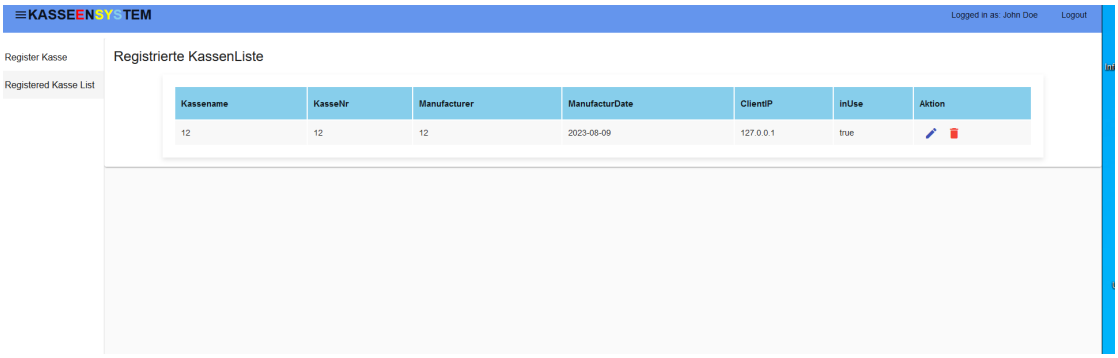

Abbildung A.2: Liste der registrierten Kassen

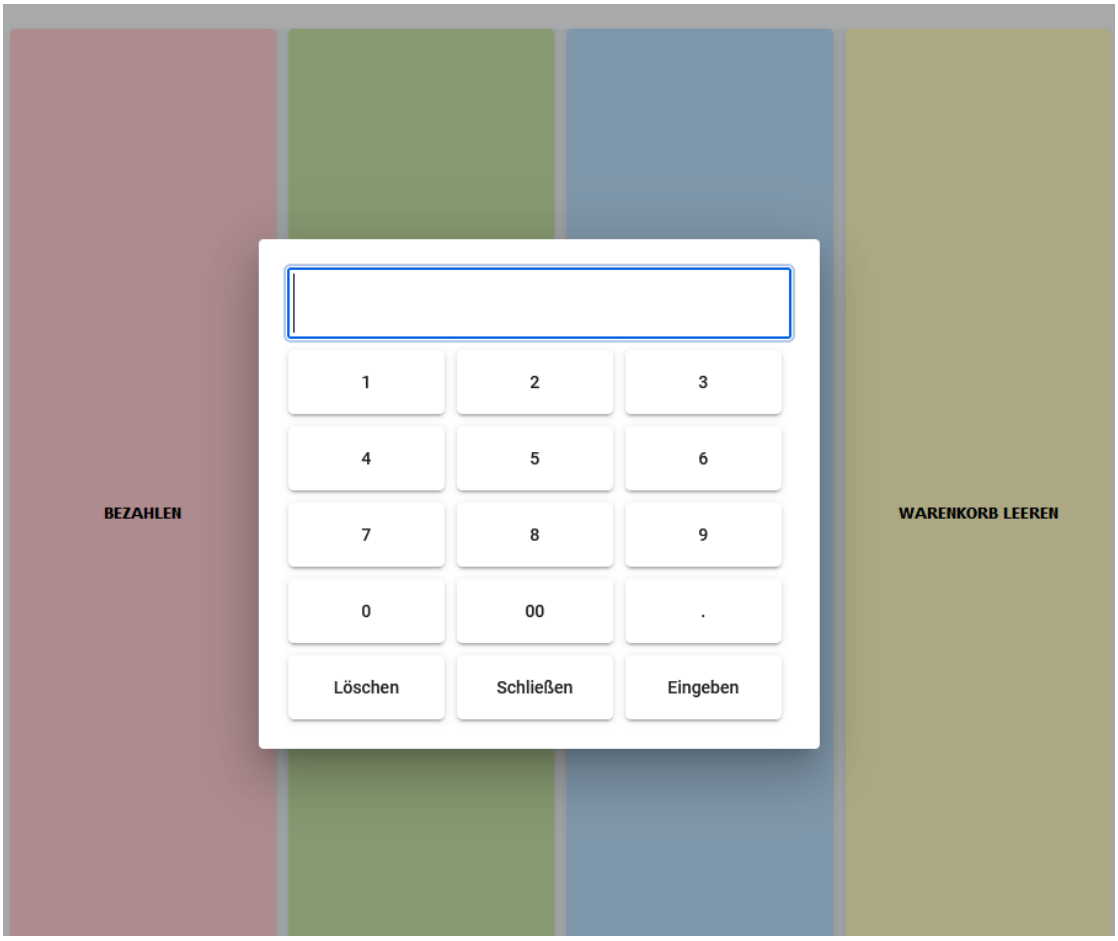

Abbildung A.3: Numpad

Abbildung A.4: Artikel nicht gefunden Meldung

```
onAddToBasket(basketItem: Products) {

  const obj: BasketItem = {
    //code=id for a while?? TODO
    articleId: basketItem.id,
    category: basketItem.category,
    name: basketItem.name,
    totalQuantity: basketItem.totalQuantity,
    price: basketItem.price,
    description: basketItem.description,
    removedFromBasket: false
  }
  if (this.basketService.basketIdentifier.getValue() === '') {
    this.basketService.createNewBasket().pipe(
      switchMap( project: response => {
        return this.basketService.addBasketItem(obj, response.msg);
      })
    ).subscribe( next: res => {
      this.basketService.fetchProducts();
      setTimeout( callback: () => {
        this.scrollToBottom();
      }, ms: 100);
    });
  } else {
    this.basketService.addBasketItem(obj, this.basketService.basketIdentifier.getValue()).subscribe( next: res => {
      this.basketService.fetchProducts();
      setTimeout( callback: () => {
        this.scrollToBottom();
      }, ms: 100);
    });
  }
}
```

Abbildung A.5: onAddToBasket

```
@Override
public void updateBasket(String basketIdentifier, BasketItem basketItem) throws BasketNotFoundException {
    BasketEntity basketEntity = basket1Repository.findBasketByBasketIdentifier(basketIdentifier);
    if (basketEntity == null) {
        throw new BasketNotFoundException("Basket with basketIdentifier " + basketIdentifier + " not found!!");
    }

    BasketItemEntity basketItemEntity = fromBasketItemModelToEntity(basketItem);

    // Generate a unique internal ID for the basketItemEntity using ObjectId
    ObjectId internalId = new ObjectId();
    basketItemEntity.setId(internalId.toHexString());
    basketEntity.getBasketItemEntityList().add(basketItemEntity);
    basket1Repository.save(basketEntity);
}
```

Abbildung A.6: Warenkorb aktualisieren

```
1 usage
@Override
public String createNewBasket(KasseInfo kasseInfo) throws BasketAlreadyExistsException {
    BasketEntity entity = this.fromModelToEntity(kasseInfo);

    // Check if a basket with the same basketIdentifier exists
    if (basket1Repository.existsByBasketIdentifier(entity.getBasketIdentifier())) {
        throw new BasketAlreadyExistsException("Basket with basketIdentifier: " + entity.getBasketIdentifier() + " already exists!");
    }

    // Check if there is already a basket with an empty basketItemEntityList
    BasketEntity existingBasket = basket1Repository.findFirstByEmptyBasketItemEntityList();
    if (existingBasket != null) {
        return existingBasket.getBasketIdentifier();
    } else {
        basket1Repository.save(entity);
        return entity.getBasketIdentifier();
    }
}
```

Abbildung A.7: Neue Warenkorb generieren

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original