HAW HAMBURG

BACHELORTHESIS
Matthias Szykora

Konzeption und Implementierung einer verteilten Supply Chain Management Applikation auf Basis von Smart Contracts und der Ethereum Blockchain

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Matthias Szykora

Konzeption und Implementierung einer verteilten Supply Chain Management Applikation auf Basis von Smart Contracts und der Ethereum Blockchain

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

Betreuender Prüfer: Prof. Dr. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 22. August 2019

Matthias Szykora

Thema der Arbeit

Konzeption und Implementierung einer verteilten Supply Chain Management Applikation auf Basis von Smart Contracts und der Ethereum Blockchain

Stichworte

Blockchain, Ethereum, Smart Contract, Supply Chain Management

Kurzzusammenfassung

Im heutigen digitalen Zeitalter leidet durch die steigende Anonymität besonders das Vertrauen untereinander. Die Blockchain Technologie gilt als mögliche Lösung, um dieses Problem zu beheben. In dieser Arbeit wird mit Hilfe der Ethereum Blockchain und Smart Contracts aufgezeigt, dass sich das gegenseitige Vertrauen stärken lässt, solange man bei der Umsetzung mit gewissen Einschränkungen leben kann. Hierfür wird ein reales Fallbeispiel aus dem Bereich des Supply Chain Managements konzipiert und implementiert. Dafür wird der Prozess von der Endfertigung eines fiktiven Arzneimittels bis zur Produktauslieferung an den Kunden mittels der Blockchain Technologie abgebildet.

Matthias Szykora

Title of Thesis

Conception and implementation of a distributed supply chain management application based on Smart Contracts and the Ethereum Blockchain

Keywords

Blockchain, Ethereum, Smart Contract, Supply Chain Management

Abstract

In today's digital age, trust in one another suffers due to increasing anonymity. Blockchain technology is considered a possible solution to this problem. In this paper, the

Ethereum Blockchain and Smart Contracts demonstrate that mutual trust can be strengthened as long as one can live with limitations in implementation. For this purpose, a real case study from the field of supply chain management is designed and implemented. For this purpose, the process from the final production of a fictitious drug to the delivery of the product to the customer is modeled using Blockchain technology.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **ix**

1 Einleitung **1**
1.1 Motivation 1
1.2 Ziel und Abgrenzung 2
1.3 Zielgruppe 3
1.4 Aufbau der Arbeit 3

2 Grundlagen und Einführung **4**
2.1 Grundlagen der Kryptografie 4
2.1.1 Symmetrische und asymmetrische Kryptografie 4
2.1.2 Kryptografische Hashfunktion 5
2.1.3 Digitale Signatur 6
2.1.4 Hash-Baum 8
2.2 Peer-to-Peer-Systeme 9
2.3 Funktion einer Blockchain 10
2.3.1 Eigenschaften der Blockchain 10
2.3.2 Transaktionen 11
2.3.3 Aufbau eines Blocks 12
2.3.4 Erstellung und Verkettung von Blöcken 14
2.4 Ethereum 16
2.4.1 Ethereum Datenstruktur 17
2.4.2 Zustände und Accounts 17
2.4.3 Ethereum Netzwerke 21
2.5 Organisation einer Blockchain 21
2.5.1 Öffentliche Blockchain 21
2.5.2 Private Blockchain 21

2.6 Smart Contracts 22
2.6.1 Ethereum Virtual Machine 22
2.7 IT-Sicherheit und Blockchain 23

3 Anforderungsanalyse **24**
3.1 Anforderungsanalyse 25
3.1.1 Funktionale Anforderungen 25
3.1.2 Anwendungsfall 29
3.1.3 Nicht-funktionale Anforderungen 34
3.2 Bewertung bestehender Lösungen 35

4 Konzeption und Implementierung **37**
4.1 Architektur 37
4.1.1 Logische Ebene 37
4.1.2 Anwendungsebene 38
4.1.3 Zusammenspiel der Funktionsbereiche 39
4.2 Technologieentscheidungen 40
4.2.1 Programmiersprache Solidity 40
4.2.2 Open Zeppelin 41
4.2.3 Truffle 42
4.2.4 Ganache 42
4.2.5 Flask 43
4.2.6 Web3.py 43
4.2.7 MySQL Datenbank 43
4.2.8 Postman 43
4.2.9 Vagrant und Ansible 44
4.2.10 Mobile Applikation 44
4.2.11 Temperatursensor 44
4.3 Implementierung des Backend Systems 45
4.3.1 Implementierung des Smart Contracts 45
4.3.2 Implementierung der RESTful-Schnittstelle 49
4.3.3 Aufbau der Datenbank 51
4.4 Implementierung der Android Applikation 51
4.4.1 Kommunikation mit dem Temperatursensor 52
4.5 Test der Implementierung 52

5 Fazit und Ausblick **56**
5.1 Ausblick 57

Literaturverzeichnis **59**

A Anhang **64**

Selbstständigkeitserklärung **67**

Abbildungsverzeichnis

2.1 Nachrichtenaustausch mittels asymmetrischer Kryptografie zwischen Bob und Alice. 5
2.2 Hashing des Eingabetextes Blockchain in SHA-256 und Keccak-256 6
2.3 Digitale Signatur zwischen Bob und Alice 7
2.4 Hashbaum mit 8 Einträgen. Text T_5 soll überprüft werden 9
2.5 Client-Server-System vs. Peer-to-Peer-System 10
2.6 Verkettung von einzelnen Blöcken in der Blockchain.(in Anlehnung an [14, S.87] 14
2.7 Beispiel für einen Patricia-Trie 18

3.1 Übersicht der einzelnen Komponenten 26

4.1 Komponentendiagramm der logischen Ebene 38
4.2 Aktivitätsdiagramm für die Nutzergruppe der Hersteller 39
4.3 Laufzeitsicht des Smart Contracts 48
4.4 Integrationstest des Smart Contract mit Hilfe der Truffle Bibliothek . . . 53

A.1 View Übersicht der Android Applikation - 1. Teil 65
A.2 View Übersicht der Android Applikation - 2. Teil 66

Tabellenverzeichnis

2.1 Aufbau eines Blockheader [3] 13

3.1 Registrierung eines neuen Nutzers (UC-001) 29
3.2 Anmelden eines Nutzers im System (UC-002) 30
3.3 Erstellung einer digitalen Kopie (UC-003) 30
3.4 Erwerb eines Produktes (UC-004) 31
3.5 Lieferung und Eingangsprüfung nach Erwerb eines Produktes. (UC-005) . 32
3.6 Produkt wird zum Verkauf vom Händler angeboten (UC-006) 34

4.1 Struktur eines Truffle Projekts 42
4.2 Übersicht der Gaskosten pro Operation 45
4.3 Übersicht der RESTful-API Endpunkte 49

1 Einleitung

1.1 Motivation

Durch die vorwiegende Anonymität im heutigen digitalen Zeitalter werden immer mehr Daten manipuliert, Inhalte von Regierungen zensiert oder Wahlen von Algorithmen beeinflusst. Hierdurch leidet das Vertrauen untereinander.

Als Satoshi Nakamoto 2008 das White Paper *Bitcoin: a peer-to-peer electronic cash system* veröffentlichte, wurde neben der Kryptowährung Bitcoin auch die dazugehörige dezentrale Blockchaintechnologie beschrieben [31], die Vertrauen, Transparenz und Robustheit stärken soll [15].

Bob Greifeld, CEO von Nasdaq, Inc. beschreibt die Blockchain als "biggest opportunity set we can think of over the next decade"[22]. Zwar ist die Blockchain kein Allheilmittel und jeder der für seinen Use-Case auf die Aspekte Transparenz oder Vertrauen verzichten kann, wird mit einer anderen Technologie glücklicher sein. Aber Industrien in denen es genau auf diese Kriterien ankommt, können von der Blockchaintechnologie profitieren. Dazu gehört unter anderem auch die Pharmaindustrie.

Medikamente sind Güter, die dies höchsten Sicherheits- und Qualitätsansprüchen erfüllen müssen. Sie unterliegen strengen gesetzlichen Vorschriften. Schon die kleinste Abweichung in der Qualität kann einen erheblichen Schaden beim Konsumenten verursachen.

Um diesen hohen Ansprüchen gerecht zu werden, wurde im Jahr 2011 ein europaweites Gesetz erlassen, welches die Arzneimittelversorgung sicherer gestalten soll. Dieses Gesetz trat am 09.02.2019 in Kraft und soll den Kunden vor Plagiaten schützen [27]. Des Weiteren wurden im Jahr 2013 die *Good Distribution Practice Guidelines* der EU revidiert. Diese Überarbeitung der Richtlinie beinhaltet, dass die Temperatur von Arzneimitteln bei risikobewerteten Transporten fortan überwacht werden muss [18]. Genau für diese Aspekte kann die Blockchain förderlich sein, da sie eine innovative Lösung für die

Rückverfolgbarkeit von Medikamenten in der Arzneimittelversorgungskette ist. Beteiligte Akteure der Lieferkette, wie Fabrikanten, Großhändler oder Endverbraucher erhalten dadurch mehr Transparenz, da Transaktionen innerhalb der Lieferkette für jedermann einsehbar und nachvollziehbar sind. Auch helfen intelligente Verträge, sogenannte Smart Contracts, welche auf der Blockchaintechnologie basieren, die Vertragsbestandteile zwischen den Parteien zu überwachen. Durch all dies kann zwischen den beteiligten Gruppen ein neues Vertrauen entstehen und ganz nebenbei die Arzneimittelfälschung begrenzt werden.

Denn leider nimmt die Anzahl an gefälschten Medikamenten in vielen Ländern Asiens, Afrikas und Latainamerikas nicht ab. Laut der Weltgesundheitsorganisation (WHO) können immer noch 30 Prozent aller weltweit hergestellten Medikamente gefälscht sein. Diese gelangen immer wieder auf illegale Weise, wie zum Beispiel auf dubiosen Internetportalen, an ahnungslose Kunden [53].

1.2 Ziel und Abgrenzung

In dieser vorliegenden Arbeit soll eine verteilte Supply Chain Management Applikation konzipiert und implementiert werden. Dafür wird als Fallbeispiel der Prozess von der Endfertigung eines fiktiven Arzneimittels bis zur Produktauslieferung an den Kunden mittels Smart Contracts und der Ethereum Blockchain abgebildet. Insbesondere wird viel Wert auf die Einhaltung vorher definierter Vertragsinhalte, wie der Temperatur und Qualität geachtet. Dafür wird u.a. die vorgeschriebene Temperatur des Medikamentes ab dem Zeitpunkt der fertigen Produktion überwacht, wobei unzulässige Temperaturabweichungen die Qualität des Erzeugnisses mindern können. Üblicherweise werden Arzneimittel in großen Chargen vom Hersteller an die Großhandelsunternehmen verschickt. Da die Überwachung der Temperatur jedes einzelnen Paketes aber den Umfang sprengen würde, wird in dieser Arbeit nur die Temperatur eines einzelnen Paketes überwacht.

Es ist nicht das Ziel, den kompletten Pfad der Arzneimittelversorgungskette lückenlos abzubilden. Vielmehr soll mit dieser Arbeit überprüft werden, ob der Einsatz der Blockchaintechnologie aus heutiger Sicht auf das oben beschriebene Fallbeispiel anzuwenden ist.

1.3 Zielgruppe

Grundlegende Kenntnisse aus dem Bereich der Verteilten Systeme, Kryptografie und IT-Sicherheit wird für das Verständnis der Arbeit vorausgesetzt. Daher richtet sich diese vorliegende Arbeit an Studenten aus höheren Fachsemestern. Insbesondere ist diese Arbeit aber für Informatiker aus dem Bereich IT-Sicherheit geeignet, die offen für neue Technologien und deren Einsatzgebiete sind.

1.4 Aufbau der Arbeit

Blockchain und Smart Contracts sind sehr aktuelle Themen in der digitalen Welt. Für das nötige Verständnis dieser Bereiche wird im folgenden 2. Kapitel auf die technischen Grundlagen eingegangen. Hier werden wichtige Grundlagen zum Thema Kryptografie und Peer-to-Peer Systeme erläutert. Zudem wird die Theorie der Blockchain und Smart Contracts erklärt. Insbesondere wird hier auf die Ethereum Blockchain und die Sicherheit einer Blockchain eingegangen.

Das 3. Kapitel beschäftigt sich mit dem aktuellen Zustand der Supply Chain sowie einer detaillierten Analyse der Aufgabe, welche dieser Arbeit zu Grunde liegt. Dies beinhaltet die Ausarbeitung der funktionalen Anforderungen für die zu entwerfende Applikation. Anschließend werden die nicht-funktionalen Anforderungen, wie die Performanz und Nutzbarkeit des Systems hervorgehoben.

Im darauffolgenden 4. Kapitel *Konzeption und Implementierung* wird Bezug auf die eigentliche Systemarchitektur, verwendeten Programmiersprachen, Bibliotheken und Tools genommen. Des Weiteren wird die Umsetzung der Implementierung und Tests aufgezeigt, sowie die Prozessdurchführung beleuchtet.

Im 5. und letzten Kapitel wird das Fazit dieser Arbeit gezogen. Zudem wird ein Ausblick in die Zukunft gegeben.

2 Grundlagen und Einführung

2.1 Grundlagen der Kryptografie

Kryptografische Konzepte sind der Grundbaustein jeder Blockchaintechnologie. Dazu gehören neben dem Public-Key-Algorithmus auch kryptografische Hashfunktionen, die digitale Signatur sowie Hashbäume, die unter anderem für die Integrität und Authentizität innerhalb der Blockchain verantwortlich sind.

2.1.1 Symmetrische und asymmetrische Kryptografie

Im Gegensatz zur symmetrischen Kryptografie, wo nur ein geheimer Schlüssel für das Ver- und Entschlüsseln von Nachrichten verantwortlich ist, werden bei der asymmetrischen Kryptografie zur sicheren Kommunikation zwei unterschiedliche Schlüssel verwendet. Jenes Schlüsselpaar besteht aus einem geheimen, privaten Schlüssel ($K_{private}$) sowie einem allgemein bekannten, öffentlichen Schlüssel (K_{public}). Für den Erhalt einer verschlüsselten Nachricht muss ein Kommunikationspartner, in diesem Fall Alice, nur den privaten und öffentlichen Schlüssel generieren. Zwischen beiden Schlüsseln besteht eine mathematische Beziehung. Der private Schlüssel bleibt beim Besitzer und nur der öffentliche Schlüssel wird an alle anderen Partner verteilt. Mit diesem öffentlichen Schlüssel kann nun eine entsprechende Nachricht N an Alice von einem Kommunikationspartner chiffriert werden. Jene Nachricht N lässt sich im Anschluss mit dem zugehörigen privaten Schlüssel von Alice wieder dechiffrieren, siehe Abbildung 2.1.

Umgangssprachlich wird dieser Algorithmus auch als Public-Key-Algorithmus bezeichnet. Er wurde wesentlich später, als die symmetrischen Verfahren in den 70er Jahren von Diffie und Hellmann [8] sowie Ralph Merkle, entwickelt. Das Ziel der asymmetrischen Kryptografie ist es, das Vertrauen, die Integrität sowie die Authentifizierung für den Austausch von Nachrichten oder Dateien zu stärken.

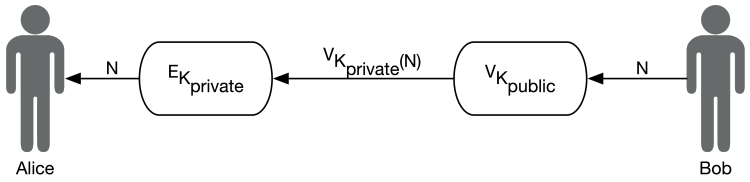

Abbildung 2.1: Nachrichtenaustausch mittels asymmetrischer Kryptografie zwischen Bob und Alice.

2.1.2 Kryptografische Hashfunktion

Kryptografische Hashfunktion, auch unter der Bezeichnung Einweg-Hash-Funktionen bekannt, werden oft für die Verschlüsselung von Passwörtern und der Benutzerauthentifizierung in webbasierten Systemen eingesetzt. Auch innerhalb jeder Blockchain ist dieses Verfahren für die Kontrolle der Integrität von Daten verantwortlich. Hashfunktion bilden einen beliebig langen Klartext auf einem Hash-Wert fester Länge ab. Um die Sicherheit zu gewährleisten, müssen kryptografische Hashfunktionen die Einwegfunktionalität aufweisen und auch kollisionsresistent sein.
Unter einer Einwegfunktion versteht man, wenn es unmöglich oder nur sehr schwer ist, zu einem vorhandenen Ausgabetext b einen passenden Eingabetext a zu finden, den die Hashfunktion h auf b abbildet: $h(a) = b$. Zudem sind Hashfunktionen kollisionsresistent, wenn es nicht trivial ist, zwei unterschiedliche Texte mit dem gleichen Hash-Wert zu finden [10, S. 98]. Bilden jedoch zwei Texte den gleichen Hash-Wert, spricht man von einer Kollision. Leider lassen sich Kollisionen bei dieser Art von Hashfunktion nicht ganz ausschließen.

SHA-256

Eine der kryptografisch stärksten Methoden ist, laut Bundesamt für Sicherheit in der Informationstechnik, (BSI) SHA-256, auch unter SHA-2 bekannt. Jener Algorithmus wird unter anderem in der Bitcoin Blockchain eingesetzt. Die erste Version (SHA-1) wurde 1993 von der National Security Agency (NSA) sowie dem Nationale Institute of Standards and Technology (NIST) entwickelt. Beträgt bei SHA-256 der ausgegebene Hash-Wert 256 Bit, so ist er bei SHA-1 nur 160 Bit lang [10, S.99]. Aktuell wird SHA-1 vom BSI nicht mehr als sicher eingestuft [21].

Um einen Text mit SHA-256 zu berechnen, wird dieser in 512-Bit große Eingabeblöcke

bzw. sechzehn 32-Bit Wörter aufgeteilt. Diese werden kryptografisch verkettet und für die Ausgabe ein 256 Bit langer Hash-Wert generiert.

Eine Besonderheit liegt für die Bitcoin Blockchain vor. Dort wird der Double-SHA-256 berechnet. Das bedeutet, dass der Algorithmus zweimal hintereinander auf das vorherige Ergebnis angewandt wird. Danach werden beide generierten Hashes wieder miteinander vereint [14, S.75].

Keccak-256

Eine weitere Einweg-Hash-Funktion, die in der Ethereum Blockchain eingesetzt wird, ist der Keccak-256 Algorithmus. Dieser wurde in einem von der NIST öffentlich ausgeschriebenen Wettbewerb als Verbesserung von SHA-2 entwickelt und 2012 als SHA-3 getauft. Allerdings ähnelt dieser Algorithmus nicht seinen Vorgängern SHA-1 und SHA-2, was als Vorteil angesehen werden kann, sollte SHA-2 in der Zukunft unsicher werden. Der Keccak-256 Algorithmus basiert auf der sogenannten Sponge-Funktion (Schwamm-Funktion). Dabei wird der Input wie von einem Schwamm aufgesaugt und im Anschluss zur Bildung des Hash-Wertes wieder herausgequetscht [43, S.245-248].

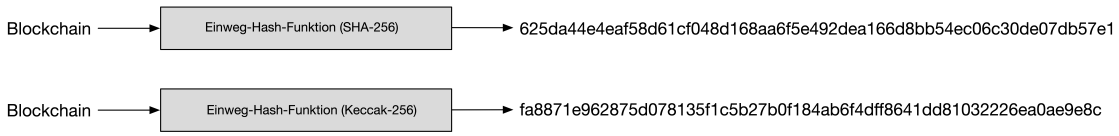

Abbildung 2.2: Hashing des Eingabetextes Blockchain in SHA-256 und Keccak-256

2.1.3 Digitale Signatur

Neben der Integrität soll eine Datei auch authentisch, fälschungssicher, nicht wiederverwendbar, unveränderbar und bindend sein. Diese Ziele kann man durch digitale Signaturen, die ein wertvolles Hilfsmittel der Kryptografie sind, erlangen.

In der Praxis wird der Sender (Alice) einer Nachricht N, einen Hash-Wert[1] H mithilfe einer kryptografischen Hashfunktion von N erstellen und anschließend diesen Wert H mit dem privaten Schlüssel $K_{private,alice}$ von Alice signieren. Diese Signatur S wird zusammen

[1]Ein Hash-Wert wird im Allgemeinen auch als Fingerabdruck bezeichnet.

mit der Nachricht N an den Empfänger (Bob) übermittelt. Dort angekommen, wird aus der übertragenen Nachricht N der Hash-Wert $H^{'}$ abgeleitet. Zusätzlich überprüft Bob die Signatur $S^{'}$ mit Hilfe des öffentlichen Schlüssels von Alice $K_{public,alice}$. Stimmen beide Ergebnisse $H^{'}$ und $S^{'}$ nicht überein, so ist die Nachricht ungültig und wurde manipuliert, siehe auch Abbildung 2.3. Um die Sicherheit dieses Verfahrens zu gewährleisten, ist es wichtig, dass kryptografische Hashfunktion benutzt werden. Nur diese haben den Anspruch an die Kollisionsresistenz [38, S.336 - 337].

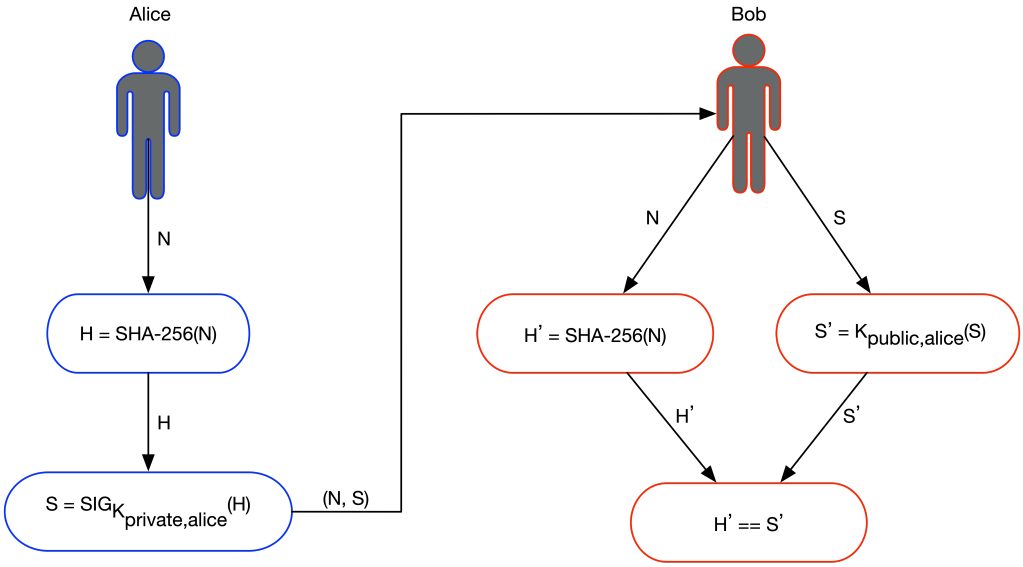

Abbildung 2.3: Digitale Signatur zwischen Bob und Alice

Auch in der Blockchain Technologie werden digitale Signaturen eingesetzt, die sicherstellen, dass Transaktionen vom zulässigen Besitzer stammen. Dieses geschieht durch elliptische Kurven (ECC). Genauer gesagt dem Elliptic Curve Digital Signature Algorithm (ECDSA), der eine Verbindung zwischen ECC und dem Digital Signature Algorithmus (DSA) darstellt. Der Digital Signature Algorithmus ist in [34] beschrieben.

Elliptische Kurven

Bei den elliptischen Kurven handelt es sich um eine Form der asymmetrischen Kryptografie. Sie basieren auf dem Problem des diskreten Logarithmus über die Addition und Multiplikation von Punkten auf einer elliptischen Kurve. Veröffentlicht wurde die vom Bitcoin Protokoll verwendete Kurve *secp256k1* vom NIST. Diese ist durch folgende

Funktion definiert: $y^2 = \left(y^3 + 7\right)$ über (F_p). F_p ist der endliche Körper mit der Primzahl p.
Die Kurve *secp256k1* besitzt eine Schlüssellänge von 256 Bit und ein Sicherheitslevel von 128 Bit. Das bedeutet: Um den privaten Schlüssel von 256 Bit zu finden, sind 2^{128} Operationen erforderlich [2, S.63-67].

2.1.4 Hash-Baum

Hash-Bäume (Merkle-Tree), die bereits 1979 von Ralph Merkle erfunden wurden, sind Bäume, die kryptografische Hash-Werte enthalten. Das Ziel bei Hash-Bäumen ist die Verminderung von großen Datenmengen und die Integritätsgarantie auf die Daten. Ein Merkle-Tree besteht aus mehreren Blattknoten. Diese haben ihren Ursprung aus paarweise angeordneten Hash-Blöcken, die den eigentlichen Text beinhalten. Diese Blattknoten werden paarweise bis zum Ende des Baumes konkateniert. Der oberste Knoten wird als Elternknoten[2] bezeichnet, mit dem sich die digitale Signatur überprüfen lässt. Durch dieses Verfahren benötigt man nicht die gesamte Nachricht, um eine digitale Signatur zu verifizieren.

Wie in Abbildung 2.4 dargestellt, benötigt ein Empfänger, um den Text T_5 korrekt zu verifizieren, nur die Hash-Werte von $Hash\,T_4$, $Hash\,T_6T_7$ und $Hash\,T_0T_1T_2T_3$. Daraus lassen sich die Werte für $Hash\,T_5$, $Hash\,T_4T_5$ $Hash\,T_4T_5T_6T_7$ und $Hash\,T_0T_1T_2T_3T_4T_5T_6T_7$ errechnen. Mit dem berechneten Wert des Elternknotens kann man nun die digitale Signatur kontrollieren [43, S.263-264].

Für einen Angreifer ist es fast unmöglich, den Text nach dem Erstellen des Wurzelknotens zu manipulieren. Um nämlich den ursprünglich identischen Hash-Wert zu erlangen, müsste der Angreifer alle möglichen Kombinationen dementsprechend modifizieren, um den vorherigen Wert zu erzielen [26, S.37].

[2] Root-Knoten oder Wurzelknoten.

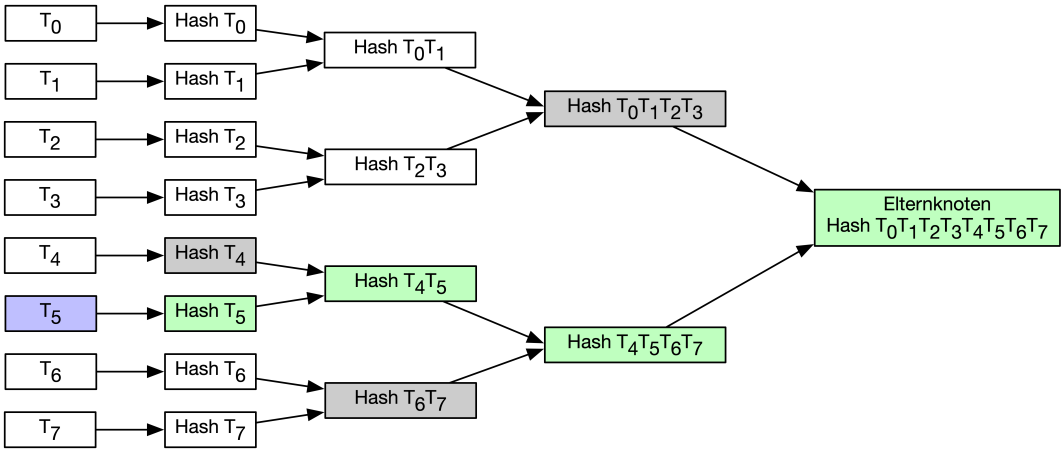

Abbildung 2.4: Hashbaum mit 8 Einträgen. Text T_5 soll überprüft werden

2.2 Peer-to-Peer-Systeme

In einer herkömmlichen Client-Server-Architektur ist der Server ein Prozess, der einen bestimmten Dienst anbietet. Dieser implementierte Dienst kann von einem Client in Anspruch genommen werden, beispielsweise die Abfrage des aktuellen Bitcoin Kurses. Leider besitzt diese Architektur den Nachteil, dass der Server dem Client die Rechte an dem Dienst entziehen kann bzw. der Server ausfällt. Somit können Anfragen vom Client an den Server nicht mehr verarbeitet werden. Der Server ist in diesem System der Single Point of Failure.

Anders verhält es sich in einem dezentralen Peer-to-Peer-System (P2P). Dort ist jeder Rechner gleichberechtigt bzw. ebenbürtig. Darunter versteht man, dass sämtliche Rechner im Peer-to-Peer-System zugleich Client als auch Server sein können. Diese Architektur benötigt keinen zentralen Server und organisiert sich selbst [48].

- Sicherstellung, dass alle Teilnehmer zum System beitragen.
- Alle Teilnehmer besitzen die identische Rolle, auch wenn sie möglicherweise unterschiedliche Ressourcen beitragen.
- Anbieter und Nutzer des Systems können einen gewissen Grad an Anonymität erreichen. Dies ist, in einem Peer-to-Peer-System allerdings nicht gewollt.

Neben der Ethereum oder Bitcoin Blockchain findet die Peer-to-Peer-Technologie Einsatz in Tauschbörsen wie BitTorrent oder Instant Messaging Technologien.

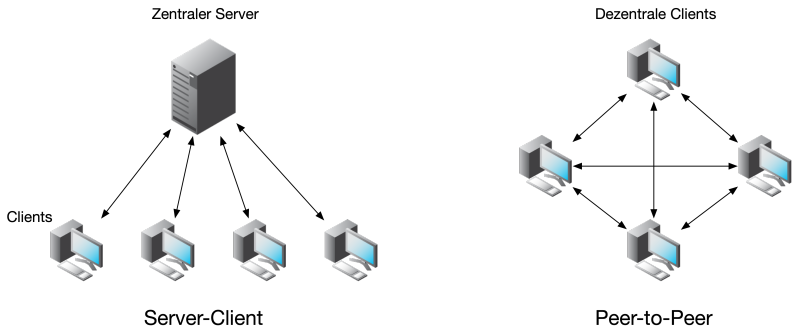

Abbildung 2.5: Client-Server-System vs. Peer-to-Peer-System

2.3 Funktion einer Blockchain

Anhand der Bitcoin Blockchain wird in den kommenden Abschnitten der Aufbau sowie die Struktur einer Blockchain genauer aufgezeigt. Die Blockchain Technologie basiert, wie im vorherigen Abschnitt 2.2 angemerkt, auf dem dezentralen Peer-to-Peer-System. Jeder einzelne vollwertige Knoten[3] kommuniziert miteinander und besitzt dieselben Rechte sowie vollen Zugriff auf alle in der Blockchain gespeicherten Daten. Man bezeichnet die Blockchain auch als eine große verteilte Datenbank. Die Datenstruktur der Blockchain Technologie, die sich auch aus dem Namen ableiten lässt, besteht aus einer verketteten Liste von Blöcken. Jeder Block wird durch seinen eigenen Hashwert eindeutig kenntlich gemacht. Zudem referenziert jeder Block auf seinen vorherigen Block. Das bedeutet, dass ein Block in der Chain den Hash-Wert des vorherigen Blocks beinhaltet [9, S.826]. Der erste Block in der Blockchain wird auch als *Genesis-Block* bezeichnet.

2.3.1 Eigenschaften der Blockchain

Folgende Eigenschaften beinhaltet die klassische öffentliche Blockchain Technologie:

Hohe Stabilität und Konsens
Durch die dezentrale P2P-Struktur gibt es keine zentrale Administration. Auf jedem einzelnen Knoten werden die Daten in replizierter Form gespeichert. Das bedeutet, dass jeder teilnehmende Knoten in der Blockchain die identischen Daten impliziert und somit

[3] Umgangssprachlich auch Full-Nodes genannt.

ein Konsens untereinander herrscht [26, S.29]. Somit ist eine hohe Stabilität und Verfügbarkeit innerhalb der Blockchain gewährleistet, da das System auch bei einem Ausfall von einem oder mehreren Knoten fehlerfrei weiterläuft.

Transparenz
Innerhalb der Blockchain sind Transaktionen von einem zum anderen Nutzer ersichtlich und nicht zurückweisbar. Darum gilt die Blockchain als eine transparente Technologie, in der jeder Beteiligte jegliche Transaktion einsehen kann.

Manipulationssicher
Aufgrund von sequentiell verketteten Blöcken können in einer öffentlichen Blockchain die Transaktionen nachfolgend nicht mehr modifiziert oder entfernt werden [9, S.827]. Im Kontext von Unternehmen werden oft private Blockchains, die eine zentrale Instanz besitzen, eingesetzt. Diese können wiederum zur Schwächung der klassischen Eigenschaften führen, da der zentraler Punkt als Single Point of Failure angesehen werden kann. Des Weiteren kann diese Instanz den Blockchain Mitgliedern die Zugriffsrechte entziehen.

Vertrauen
Dank der Replikation identischer Daten auf verteilten Knoten, der Transparenz von Transaktionen sowie der Manipulationssicherheit kann Vertrauen zwischen Blockchain Mitgliedern aufgebaut werden wo vorher keines vorhanden war.

2.3.2 Transaktionen

Transaktionen gehören in einem Blockchain Netzwerk zu den wichtigsten Bausteinen und werden allesamt in der Blockchain gespeichert. Eine Transaktion kann beispielsweise der Eintrag in ein Wahlregister oder die Stimmenabgabe für eine Hochschulwahl sein. Damit aber eine Transaktion ausgeführt werden kann, muss zuvor eine Bitcoin Adresse erstellt werden. Hierbei wird für jede Adresse mit Hilfe des ECDSA-Algorithmus ein zugehöriger privater und öffentlicher Schlüssel erstellt. Wird eine Transaktion verschickt, so wird sie vom Absender mit einem privaten Schlüssel signiert und an alle beteiligten Knoten im Blockchain Netz verschickt. Empfängt ein Knoten eine noch nicht bekannte und valide Transaktion, so sendet er diese an alle mit ihm verbundenen Knoten weiter. Hiermit wird sichergestellt, dass Transaktionen schnell durch das Peer-to-Peer Netzwerk an alle

aktiven Knoten weitergereicht werden. Diese Technik ist in einem verteilten System auch als Fluten (Flooding) bekannt [2, S.26]. Fehlerhafte Transaktionen werden jedoch sofort vom empfangenen Knoten verworfen. Mit Hilfe sogenannter Miner-Knoten wird im Anschluss die gültige Transaktion bestätigt und in einem Block aufgenommen [9, S.829]. Damit die einzelnen Knoten die Transaktionen erfolgreich verarbeiten können, beinhalten diese weitere Informationen. Dazu gehören u.a. die Versionsnummer, Lock-Zeit, Input- und Output-Liste sowie den In- und Out-Zähler.

Versionsnummer
Die Nummer gibt Auskunft über die verwendete Version der Transaktion [14, S.83].

Lock-Zeit
Diese Zeit legt fest, wann eine Transaktion zum Block hinzugefügt wird [14, S.83].

Input- und Output-Liste
Transaktionen kosten immer Geld. In diesem Fall in Form der Währung Bitcoin. Damit eine neue Transaktion von einem Teilnehmer (Alice) ausgeführt werden kann, muss diese im Besitz von Bitcoin sein. Dazu ist es notwendig, dass man von einem Teilnehmer (Bob) in einer vorherigen Transaktion Bitcoin erhalten hat. Demzufolge steht in der Input-Liste der neuen Transaktion, die Adresse von Alice, welche auf die Output-Liste der früheren Transaktion von Bob referenziert. In der Output-Liste steht somit die Adresse des Empfängers [14, S.83].

In- und Out-Zähler
In dem In- und Out-Zähler steht die Anzahl der Inputs und Outputs einer Transaktion [14, S.83].

2.3.3 Aufbau eines Blocks

Ein Block in der Bitcoin Blockchain besteht aus einem Blockheader, der Transaktionsliste, einer Magic Number, der Blockgröße und dem Transaktionszähler.

Blockheader

Der Blockheader beträgt in der Bitcoin Blockchain 80 Byte und enthält relevante Informationen, siehe auch Tabelle 2.1 .

Tabelle 2.1: Aufbau eines Blockheader [3]

Feld	**Größe**	**Beschreibung**
Versionsnummer	4 Byte	Welche Version der Blockchain zum Zeitpunkt der Blockerstellung genutzt wurde.
Hash-Wert des vorherigen Blockheader	32 Byte	Ein Double-SHA-256-Hash des Headers des vorherigen Blocks. Dies stellt sicher, dass kein vorheriger Block geändert werden kann, ohne auch den Header dieses Blocks zu ändern.
Merkle Tree Hash	32 Byte	Der Wert des Elternknotens des Hash-Baums. Abgeleitet von allen Transaktionen in dem Block befindlichen Transaktionen. Auch zur Sicherheit, dass keine der Transaktionen verändert werden kann.
Timestamp	4 Byte	Wann wurde der Block ungefähr erstellt.
Difficulty	4 Byte	Bestimmung der Schwierigkeit im Netzwerk.
Nonce[4]	4 Byte	Hash-Wert des Blockheaders für den PoW.

Transaktionsliste

Liste aller im Block enthaltenen Transaktionen.

Magic Number

Die Magic Number der Bitcoin Blockchain lautet *f9beb4d9* und zeigt den einzelnen Knoten im Netzwerk, dass der übertragene Block zu der Bitcoin Blockchain gehört [14, S.88].

Blockgröße

Anhand der Blockgröße wird ersichtlich, wie viele Transaktionen in einem einzelnen

[4] engl.: number used once (beliebige Zahl)

Block Platz haben. Die maximale Kapazität eines Blocks in der Bitcoin Blockchain beträgt aktuell 1 MB.

Transaktionszähler
Ein Zähler, der im Block enthaltenen Transaktionen.

2.3.4 Erstellung und Verkettung von Blöcken

Ein Block in der Bitcoin Blockchain besitzt eine Speichergrenze für die Menge aller aufzunehmenden Transaktionen. Wurde eine Transaktion noch nicht in einem Block im Netz hinzugefügt, wird diese in einer entsprechenden Warteschlange eines vollwertigen Knotens[5] zwischengespeichert. Nur Transaktionen, die vom Full-Node validiert und verifiziert wurden, werden in der Warteschlange aufgenommen [14, S.92], siehe auch Abbildung 2.6.

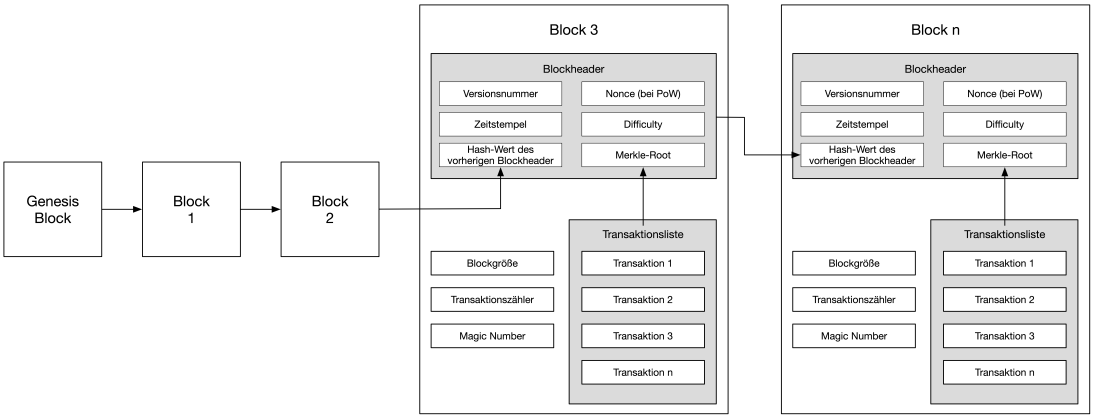

Abbildung 2.6: Verkettung von einzelnen Blöcken in der Blockchain.(in Anlehnung an [14, S.87]

Mining

Neue Blöcke mit Transaktionen werden ungefähr alle 10 Minuten von speziellen Miner-Knoten in der Bitcoin Blockchain erstellt. Nach Fertig [vgl. 14, S.96] wird eine Blockchain mit Hilfe des Minings so zusammengebaut, dass es praktisch unmöglich ist, diese nachträglich zu verändern.

[5] In der Bitcoin Blockchain als Mempool oder Memory Pool bezeichnet.

Ein Mining-Knoten fragt eine Liste des Mempool-Inhalts eines vollwertigen Knotens ab und wählt die entsprechenden Transaktionen aus. Im besten Fall werden alle Transaktionen aus dem Mempool in den neuen Block geschrieben. Es kann aber sein, dass Transaktionen im Mempool verbleiben, da die Zeit oder die maximal aufzunehmende Größe an Transaktionen überschritten wurde.
Bei der Auswahl der Transaktionen aus der Warteschlange gewinnen jene, die die höchste Transaktionsgebühr besitzen. Jedoch muss dies nicht die Regel sein und kann vom Miner-Knoten selbst konfiguriert werden [14, S.96].

Der Miner-Knoten berechnet für jede einzelne Transaktion den Double-SHA-256 Hash-Wert. Mit Hilfe des Merkle-Tree Verfahrens wird für alle beteiligten Transaktionen ein einziger Hash-Wert gebildet und zusammen mit einem Zeitstempel in den Blockheader des neuen Block geschrieben. Nebenbei überprüft der Miner-Knoten die Input- und Output-Liste jeder Transaktion. Wichtig ist dabei, dass neu erstellte Blöcke an alle teilnehmenden Full-Nodes gesendet werden. Nur so kann sichergestellt werden, dass der Konsens unter den teilnehmenden Knoten bestehen bleibt, da sich alle auf eine Wahrheit einigen müssen. Damit die neuen Blöcke auch korrekt und im Konsens in die bestehende Block-Kette aufgenommen werden, nutzt die Bitcoin Blockchain den Proof-of-Work (PoW) [14, S.97-99].

Proof-of-Work

Hierzu muss vom Miner, während der Erstellung des Blocks, ein rechenintensives, kryptografisches Rätsel gelöst werden. Dazu muss jeder Miner seine Rechenleistung dem Netzwerk zu Verfügung stellen und eine entsprechende Miner-Software ausführen. Damit die entsprechenden Miner-Knoten einen Anreiz bekommen dem Blockchain Netzwerk ihre Rechenleistung für das Lösen des Rätsels zur Verfügung zu stellen, wird dem Finder eine Belohnung in Form von Bitcoins in Aussicht gestellt. Das Endergebnis dieses kryptografischen Rätsels ist ein 32 Byte langer Hash-Wert über den gesamten Blockheader. Dieser muss eine festgelegte Anzahl an führenden Nullen besitzen, um kleiner als ein vorgeschriebener Grenzwert zu sein. Nach einer Zeit von zwei Wochen oder 2016 erzeugten Blöcken, wird dieser Wert automatisch neu erzeugt. Der aus dem Header resultierende Hash-Wert ist im Grunde eine Referenznummer für den neuen Block in der Kette. Um einen Wert mit der entsprechenden Anzahl an führenden Nullen zu finden hilft die Nonce, da im Header nur diese beliebige Zahl variabel und veränderbar ist. Alle anderen Felder im Header besitzen eine festgelegte Bedeutung. Sollen die ersten 20 Bits

des Hash-Werts eine Null sein, so benötigt man $2^{20} = 1048576$ Versuche um eine passende Nonce zu finden, die einen entsprechenden Hash-Wert realisiert. Der Grenzwert betrug im Juli 2019 Hexadezimal:
$0000000000000000001f0d9b0000000000000000000000000000000000000000$.
Bei der Umwandlung von Hexadezimal in das Binärsystem würden aus den 18 führenden Nullen 72 Nullen werden. Man benötigt also aktuell 2^{72} Versuche, um eine passende Nonce zu finden [26, S.42-44]. Leider besitzt dieses PoW-Prinzip den Nachteil, dass der Energieverbrauch außordentlich hoch ist. Aktuell liegt, laut Merz [vgl. 26, S.17], der weltweite Verbrauch für das dauerhafte Mining in der Bitcoin Blockchain bei zwei Gigawatt.

2.4 Ethereum

Eine weitere Blockchain Technologie ist die dezentrale Ethereum Blockchain, die 2014 in dem White Paper [4] von Vitalik Buterin und seinem Mitbegründer Dr. Gavin Wood in dem Yellow Paper [57] veröffentlicht wurde. Da der praktische Teil dieser Arbeit, auf der Ethereum Blockchain aufbaut, wird in diesem Abschnitt auf die Theorie und die Unterschiede zur Bitcoin Blockchain eingegangen.

Buterin besitzt die Absicht, mit dem Open-Source Ethereum Projekt, die digitale Welt etwas besser zu machen. Ziel ist es, die zentrale Datenspeicherung im Internet durch das Ethereum Protokoll auszutauschen und zu schützen [19].

Bei der Entwicklung der Ethereum Blockchain lag ein großes Augenmerk auf der Effizienz gegenüber der Bitcoin Blockchain. Während es bei Bitcoin Projekten ungefähr 10 Minuten dauert einen neuen Block zu erschaffen, liegt die Erstellungsdauer bei Ethereum gerade mal bei ca. 7 Sekunden. Außerdem kann jeder Teilnehmer im Ethereum Netzwerk das schon bekannte Mining durchführen. Ein weiterer Unterschied zur Bitcoin Blockchain liegt in der Entwicklung von Anwendungen für die Blockchain. Dies wurde durch die vollständige Unterstützung von mehreren Programmiersprachen deutlich verbessert. Somit gilt aktuell Ethereum als der Standard für diese Anwendungsentwicklung im Bereich Blockchain. Es ist möglich, dezentrale Applikationen (Dapps) oder Smart Contracts für die Blockchain zu entwickeln [46]. Auch Rosenberger [vgl. 42, S.53] sagt, dass Ethereum die Mutter aller Smart Contracts sei.

Die Währung, mit der innerhalb des Netzwerkes bezahlt wird, heißt Ether. Der Ether ist in mehreren kleinen Einheiten aufgeteilt, wohingegen für diese Arbeit die Einheit Wei wichtig ist. 1 Ether sind umgerechnet 1000000000000000000 wei[6].

2.4.1 Ethereum Datenstruktur

Auch in der Ethereum Blockchain sind die einzelnen Blöcke mit dem vorhergehenden Block kombiniert. Aber anders als in der Bitcoin Blockchain wird hier der Hash-Wert des Vorgängers mit einem zusätzlichen Zeitstempel versehen. Ein weiterer Unterschied ist die Zusammenfassung von Transaktionen in einem Block. Während in der Bitcoin Blockchain Transaktionen mit dem Merkle-Tree im Blockheader zusammengefasst werden, nutzt Ethereum den Modified-Merkle-Patricia-Trie (MPT) , beschrieben in [57]. Der Modified-Merkle-Patricia-Trie ist eine Zusammensetzung aus dem Merkle-Tree sowie dem Patricia-Trie , der 1968 von Donald R. Morrison [29] entwickelt wurde. Auch das Ethereum Projekt nutzt für die Hash-Wert Berechnung beim MPT-Verfahren die Keccak-256 Hashfunktion, siehe Abschnitt 2.1.2.

Patricia-Trie

Ein Trie ist wie ein Baum aufgebaut, in dem sehr lange Zeichenketten gespeichert werden können. Die Kanten des durchsuchbaren Trie-Baums sind mit Schlüsseln benannt und bilden einen Key-Value-Speicher ab. Mit dieser Struktur können lange Wörter oder IP-Adressen abgebildet werden. Der Patricia-Trie basiert auf dem Trie-Baum. Allerdings ist der Patricia-Trie wesentlich effizienter und kompakter, da diese Pfade bei identischen Zeichen im Patricia-Trie zusammengefasst werden können [16, S.104-105], anders als beim herkömmlichen Trier, wo z.B. jeder Buchstabe oder jede Zahl einen einzelnen Pfad darstellt. Siehe auch Abbildung 2.1.

2.4.2 Zustände und Accounts

Ein wichtiges Grundkonzept von Ethereum sind die transaktionsbasierten Zustände und Accounts, die für die Adressen der Teilnehmer im Ethereum Netz stehen. Der globale Zustand wird laut Fertig [14, S.119] im State Tier abgebildet und auf jedem Knoten der

[6] http://eth-converter.com

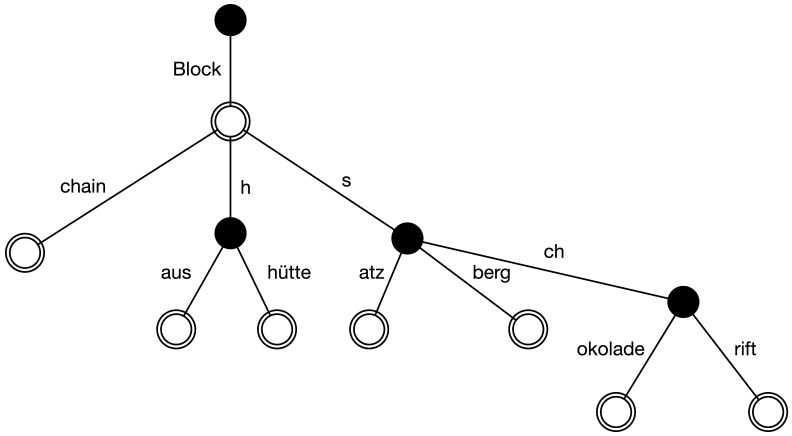

Abbildung 2.7: Beispiel für einen Patricia-Trie

Blockchain gespeichert. Dieser Zustand bildet quasi eine Momentaufnahme der gesamten Ethereum Blockchain ab. Ändert sich der Zustand eines Accounts, verändert sich damit auch der globale Zustand. Ethereum verwendet mehre Typen von Accounts. Dazu gehören u.a. der Contract Account (CA) und der Externally Owned Account (EOA). Beide Accounts sind im State Tier integriert.
Ein Externally Owned Account wird von externen realen Nutzern verwendet und besitzt für die digitale Signatur einen privaten Schlüssel. Wer im Besitz dieses Schlüssel ist, hat das Recht den EOA zu steuern. Der private Schlüssel dient dazu, signierte Transaktionen zu erstellen, womit Nachrichten im Ethereum Netz verschickt werden können [4][11].
Mit dem Contract Account werden dagegen intelligente Verträge, sogenannte Smart Contracts, gesteuert. Erhält der CA eine Nachricht, so wird der Code des Smart Contracts ausgeführt. Contract Accounts können auch Nachrichten an andere Smart Contracts senden bzw. neue erstellen [4][11].
Im Ethereum Projekt setzt sich der Zustand beider Accounts aus vier identischen Feldern zusammen [14, S.118].

- **Account Balance**: Speicherung des aktuellen Kontostands des Accounts.
- **Nonce**: Zähler für Interaktionen mit anderen Accounts beim CA und Anzahl der Transaktionen beim EOA.
- **Storage Root**: Wird nur im Contract Account verwendet. Dient als interner Speicher für Variablen des Contracts.

- **Code Hash**: Hash-Wert des Smart Contract Codes. Wird ebenfalls nur vom CA verwendet.

Transaktionen

Transaktionen sind nach Wood [57] einzelne kryptografisch, signierte Nachrichten, die von einem Externally Owned Account stammen. Diese werden über das Ethereum Netzwerk übertragen und in der Blockchain gespeichert. Durch die Transaktionen wird gewissermaßen der Übergang zwischen Zuständen im Netzwerk vollzogen. Um diese Änderungen nachzuvollziehen besitzt Ethereum die Receips Komponente. Receips sind in der Lage, ausführliche Informationen über vergangene Transaktionen zu offenbaren.

Eine Transaktion beinhaltet in der Ethereum Blockchain folgenden Aufbau:

- **Nonce**: Eine von EOA ausgegebene Sequenznummer, um doppelte Nachrichten zu verhindern.
- **Gas price**: Jede Transaktion in der Ethereum Blockchain kostet Geld mit der Einheit Gas[7], die im Yellow Paper [57] festgelegt ist. Bestimmte Transaktionen kosten mehr oder weniger Gas.
- **Gas limit**: Beinhaltet den Maximalen Gas-Wert, den der EOA bereit ist für seine Transaktionen zu bezahlen.
- **Recipient**: In diesem Feld wird die Zieladresse der Transaktion eingetragen.
- **Value**: Die Menge an Ether, die an die Zieladresse gesendet werden soll.
- **Data**: Enthält die Daten wie z.B. Eingabeparameter in binärer Form.
- **v,r,s**: Diese Variablen werden für das korrekte Signieren einer Transaktion durch den Absender nach dem ECDSA-Algorithmus benötigt.

Die Aufgabe einer Transaktion ist die Anweisung, was ein externer Nutzer von dem Ethereum Netzwerk möchte, wie z.B. Ether zwischen zwei EOAs zu senden, Programmcode eines Smart Contract auszuführen oder einen neuen Smart Contract zu erstellen.

Wie auch bei Bitcoin, wird in einem Block der Ethereum Blockchain eine bestimmte Anzahl von Transaktionen abgelegt. Es passen so viele Transaktionen in einen Block, bis

[7] Laut [14, S.121] ist der Gas Price so etwas wie der an der Tankstelle angegebene Benzinpreis pro Liter.

das festgelegte Block-Gas-Limit erreicht ist. Auch in der Ethereum Blockchain werden Transaktionen in einem Block abgelegt und der Hash-Wert mittels Modified-Merkle-Patricia-Trie gebildet. Das Limit an zu speichernden Transaktionen hängt von der Größe des Block-Gas-Limits ab.

Messages

Im Gegensatz zu Transaktionen, die nur von EOAs initiiert werden können, dienen Messages zur Kommunikation unter den teilnehmenden CAs. Mit einer Message kann ein CA eine Funktion eines weiteren CAs aufrufen. Messages haben das Merkmal, dass sie nur verschickt werden, wenn sie durch eine Transaktion eines externen Nutzers angestoßen werden [14, S.122].

Konsensbildung

Für den Konsens der einzelnen Knoten im Netzwerk, nutzt das Ethereum Projekt aktuell den Ethash-Algorithmus. Dieser ähnelt sehr stark den Proof-of-Work-Algorithmus aus der Bitcoin Blockchain, siehe Abschnitt 2.3.4. Auch führen in der Ethereum Blockchain alle Knoten jede empfangene Transaktion aus, um somit die im Knoten gespeicherten Zustände selbst zu aktualisieren.

Aktuell stellt die Ethereum Organisation jedoch den Konsensmechanismus vom Proof-of-Work-Algorithmus auf den Proof-of-Stake-Algorithmus (PoS) um [23].

Proof-of-Stake

Vorteil des PoS soll die Einsparung der Rechenkapazität und dem damit verbundenen Energieverbrauch sein. Um dieses umzusetzen werden keine kryptografischen Rätsel, wie beim PoW, mehr gelöst. Stattdessen behalten entsprechende Validatoren einen gewissen Anteil der Kryptowährung als Werteinlage, in diesem Fall Ether, ein. Diese Einlage wird auch als *Stake* bezeichnet. Möchte ein Validator an der Bildung eines neues Blocks teilnehmen, so beweist er mit dem einbehaltenen Stake, dass er das Blockchain Netzwerk schützen möchte, da sich der Wert seiner Werteinlage sonst vermindern würde [20, S15-17] [12].

2.4.3 Ethereum Netzwerke

Ethereum verfügt über eine Vielzahl an unterschiedlichen Netzwerken. Dazu gehört das Mainnet, dass als Hauptnetzwerk von Ethereum bekannt ist. Auf diesem Netzwerk ausgeführte Transaktionen kosten tatsächliches Geld. Für Testzwecke gibt es weitere Testnetzwerke. In diesen Netzen werden Smart Contracts der Ethereum Community zur Verfügung gestellt, ohne das mögliche Kosten anfallen. Bekannte Testnetzwerke sind Rinkeby, Ropsten und Kovan. Falls man sich mehr Sicherheit im Netzwerk erhofft, ist es auch möglich, ein privates Ethereum Netzwerk zu installieren.

2.5 Organisation einer Blockchain

Egal auf welcher Technologie die verwendete Blockchain basiert: Es gibt es zwei wesentliche Kategorien, nach denen eine Blockchain organisiert ist. Dazu zählen die öffentliche und die private Blockchain.

2.5.1 Öffentliche Blockchain

Eine öffentlich Blockchain ist, wie in Abschnitt 2.3 beschrieben, ein verteiltes, dezentrales System. Hierbei gibt es keinen bestimmten Eigentümer. Jeder Nutzer kann dem System beitreten und ist somit ein Teil der Blockchain. Alle Teilnehmer besitzen identischen Rechte. Diese Art der Blockchain zeichnet sich besonders durch hohes Vertrauen, Transparenz und Ausfallsicherheit aus.

2.5.2 Private Blockchain

Eine privat betriebene Blockchain ist ein geschlossenes Netzwerk. Der Zugang wird von einer zentralen Instanz gesteuert. Diese kann den Teilnehmern auch gewisse Rechte im System erteilen oder entziehen. Jedoch leidet das Vertrauen, die Transparenz sowie die Ausfallsicherheit darunter. Private Blockchains werden oft von Firmen eingesetzt, die nur die Technologie der Blockchain und nicht deren Eigenschaften nutzen wollen.

2.6 Smart Contracts

Intelligente Verträge oder auch Smart Contracts sind laut Merz [vgl. 26, S.62] Programme, die kompiliert als Bytecode auf die Ethereum Blockchain übertragen werden. Sie sollen, genau wie ein papierbasierter Vertrag, für die Einhaltung der rechtlichen Pflichten zwischen den verschiedenen Vertragsparteien sorgen. Jedoch nutzen Smart Contracts IT-basierte Konzepte, um die Erfüllung von Verträgen zwischen verschiedenen Maschinen zu gewährleisten. Die Idee der Smart Contracts wurde in den 90er Jahren durch Nick Szabo [49] bekannt und erlangte durch die Einführung der Blockchain Technologie, insbesondere der Ethereum Blockchain, einen Aufschwung. Mit dieser Technologie ist es möglich zwischen unterschiedlichen Parteien dezentrale Geschäfte der verschiedensten Güter zu tätigen ohne, dass eine dritte Instanz als Administrator oder Kontrollorgan fungiert.

2.6.1 Ethereum Virtual Machine

Smart Contracts sind Turing-Vollständig und in der Ethereum Blockchain ein Bestandteil der Contract Accounts. Dort wird der Code gespeichert und in der Ethereum Virtual Machine (EVM) ausgeführt.

Für die Ausführung eines Smart Contracts verfügt jeder Knoten im Ethereum Netzwerk über eine eigene EVM. Diese läuft auf dem Knoten komplett abgeschottet von anderen Systemkomponenten und besitzt somit auch einen Zugriff auf weitere Systemprozesse und -komponenten. Mehrere höhere Programmiersprachen, wie Solidity [47] oder Vyper [54], werden aktuell von der EVM unterstützt.

Mittels entsprechender Transaktionen eines externen Nutzers (EOA) lässt sich ein valider Smart Contract auf alle Knoten im Netzwerk hinzufügen. Zusätzlich erstellt die Ethereum Virtual Machine einen entsprechenden Contract Account samt Adresse, worüber der Smart Contract anzusprechen ist und realisiert alle nötigen Abhängigkeiten [14, S.131]. Um mit einem Contract zu interagieren, wird eine Transaktion an den Smart Contract gesendet, der die in der Nachricht übergebenen Parameter ausführt. Die Schnittstelle zwischen einem externen Nutzer und dem Contract bildet das sogenannte Wallet. Es fungiert als elektronische Geldbörse mit der z.B. Ether oder Bitcoin gespeichert, verschickt oder empfangen werden kann. Damit Transaktionen von externen Nutzern an einen Smart Contract auch korrekt umgesetzt werden, müssen genug Ether vorhanden

sein, um den entsprechenden *Gas Price* zu zahlen. Nur so ist sichergestellt, dass durch Transaktion ein neuer Smart Contract erstellt oder der Code eines Contracts ausgeführt wird. Möchte man auf der Blockchain einen gespeicherten Wert auslesen, so ist dies kostenlos. Hierbei wird keine Transaktion gestartet, die durch einen Mining Knoten in einen neuen Block geschrieben wird. Stattdessen wird eine Transaktion gegen den eigenen lokalen Blockchain Knoten ausgeführt, der die Transaktion unverzüglich beantwortet. Hierbei spricht man von einem sogenannten *Call*.

Die redundante Erstellung und Ausführung von Smart Contracts auf allen Knoten führt zu einem sicheren und robusten Geflecht, das ein System mit einem zentralen Server so nicht liefern kann [20].

2.7 IT-Sicherheit und Blockchain

Laut Eckert [9] gilt ein System dann als informationssicher, wenn es keine Zustände annimmt, die zu einer unautorisierten Informationsveränderung oder -gewinnung führen. Sichere Systeme besitzen die Eigenschaft der Integrität, Vertraulichkeit und Verfügbarkeit. Diese Eigenschaften sind auch als CIA-Triade bekannt, die oft zur Beurteilung von informationssicheren Systemen aufgegriffen werden [33]. Im Laufe der Zeit sind weitere Schutzziele, wie Authentizität, Nichtabstreitbarkeit sowie Zurechenbarkeit hinzugekommen.

Die Integrität und Authentizität wird in der Blockchain durch die Anwendung kryptografischer Hashfunktionen, digitaler Signaturen sowie die Bildung des Konsens realisiert. Unter Vertraulichkeit versteht man, Informationen vor unbefugten Personen zu schützen. Dies ist jedoch kein Designziel einer Blockchain [9, S.838]. Denn die Besonderheit einer Blockchain besteht darin, die Transparenz und dadurch das Vertrauen untereinander zu stärken. So sind gerade in einer öffentlichen Blockchain jegliche Transaktionen für alle Teilnehmer einsehbar. Als weiteres Ziel der CIA-Triade wird die Verfügbarkeit durch den Einsatz von Peer-to-Peer Netzwerken umgesetzt. Auch die weiteren Ziele der Nichtabstreitbarkeit sowie der Zurechenbarkeit werden durch die Verkettung der einzelnen Blöcke realisiert, siehe auch Unterabschnitt 2.3.4.

3 Anforderungsanalyse

In dieser Arbeit soll auf Basis von Smart Contracts und der Ethereum Blockchain eine Supply Chain Management Applikation implementiert werden.

Supply Chain Management (SCM) bedeutet laut Winterstein [56, vgl.] die integrierte, prozessorientierte Planung und Steuerung der Waren-, Informations- und Geldflüsse über die gesamte Wertschöpfungs- und Lieferkette (Supply Chain) vom Kunden bis zum Rohstofflieferanten. Ursprünglich wurde diese Lieferkette als eine lineare Kette, die sich vom Hersteller bis zum Endverbraucher ausstreckte, angesehen [6]. Dieses System hat sich aber in den vergangenen Jahre zu einem großen, verteilten Netzwerk verschiedenster Hersteller, Lieferanten und Dienstleister weiterentwickelt [24]. Dabei wurde jedoch der technologische Ausbau vernachlässigt, wodurch es sich zu einem ineffizienten System zurückentwickelt hat. Um dieses System wieder auf den neusten Stand zu überführen, bedarf es den Einsatz neuer Technologien [25].

Anhand der Kombination von Blockchain Technologie und Smart Contracts lassen sich bestimmte Anwendungsfälle in der Lieferkette verbessern. Eine aktuelle Studie der HTWK Leipzig besagt, dass verschiedene Unternehmen und Organisationen aus dem Bereich Supply Chain Management ein großes Potenzial in der Blockchain Technologie sehen [30].

Gerade im Bereich der unverzichtbaren Sendungsverfolgung von Waren, lässt sich der Vorteil einer Blockchain veranschaulichen. Durch die Verfolgung von Waren erhalten jegliche, an diesem Prozess Beteiligte, Informationen über den gegenwärtigen Status. Jedoch weist die derzeitige Umsetzung der Sendungsverfolgung einige Schwachstellen auf. Jegliche Daten sind in diesem System an einem zentralen Knoten gespeichert und sind vor späteren Manipulationen nicht ausreichend geschützt. Hinzu kommt, dass Transaktionen und Zahlungen ebenfalls nur über eine zentrale Instanz möglich sind. Durch den Einsatz der Blockchain lassen sich diese Schwachstellen aber deutlich verbessern. Eine zentrale Instanz kann den dezentralen Aufbau der Blockchain ablösen. Auch ist durch

die sequentielle Verkettung der Blöcke die Manipulationssicherheit der persistenten Daten gewährleistet. Zusätzlich kann ein Smart Contract, wie in Abschnitt 2.6 beschrieben, für die Einhaltung rechtlicher Pflichten unter allen Partnern sorgen und darüber hinaus verschiedene Mechanismen anstoßen.

3.1 Anforderungsanalyse

Mit der Entwicklung der Supply Chain Management Applikation sollen die Vor- und Nachteile für das realitatsnahe Fallbeispiel, beschrieben in Abschnitt 1.4, dargestellt werden. Hierfür wird die Ethereum Blockchain in Kombination mit Smart Contracts angewandt.

Dabei wird für ein Medikament, dass bekanntermaßen die höchsten Sicherheits- und Qualitätsansprüche besitzt, siehe Abschnitt 1.1, eine digitale Kopie aller relevanten Produkt- und Sendungsinformationen mit Hilfe eines Smart Contracts in die Blockchain geschrieben. Wird das Produkt von einem Händler erworben, kann jener es wieder zum Kauf für den Endkonsumenten anbieten. Der Weg der Sendung, vom Absender bis zum Empfänger, wird von einem entsprechenden Versanddienstleister durchgeführt. Während des gesamten Prozesses werden an allen Teilabschnitten die Informationen des Produktes sowie der Status der Sendung überwacht und Änderungen in die Blockchain eingetragen. Dabei prüft ein Smart Contract im Hintergrund, ob vorher definierte Vertragsbestandteile eingehalten werden und führt bei Nichterfüllung ebenfalls zuvor entwickelte Prozesse aus. Ein wichtiger Bestandteil ist die Einhaltung der Temperaturgrenzen während des gesamten Zyklus. Hierfür wird die Sendung, während des gesamten Vorgangs anhand eines im Paket befindlichen Temperatursensors, kontrolliert. Alle Produkt- und Sendungsinformationen können nicht nur während der gesamten Vorgangszeit eingesehen werden, sondern bleiben dauerhaft in der Blockchain abrufbar. Erst wenn die komplette Ethereum Blockchain nicht mehr aktiv ist, wären somit auch die Informationen nicht mehr zugänglich.

3.1.1 Funktionale Anforderungen

Das zu entwerfende System besteht aus mehreren einzelnen Komponenten. Eine davon beinhaltet den Ethereum Knoten, auf dem der Smart Contract, der die rechtlichen Merkmale überwacht, ausgeführt wird. Darüber hinaus gibt es einen Bereich, der eine

Datenbank und eine Schnittstelle für die Interaktion zwischen Client und Smart Contract sowie zwischen Client und Datenbank enthält. Die Datenbank dient für die Speicherung der Nutzerinformationen.

Des Weiteren wird ein Client in Form einer mobilen Applikation entwickelt. Eine Anforderung ist neben der Nutzerverwaltung der Teilnehmer, die Erstellung einer digitalen Kopie aller relevanten Produkt- und Sendungsinformationen für ein Arzneimittel. Auch wird über den Client der Erwerb sowie der Verkauf der Arzneimittel getätigt. Diese Komponente ist zusätzlich in der Lage, die aktuellen Temperaturwerte eines Bluetooth-Sensors zu erfassen. Weiterhin lassen sich über den Client jegliche, in der Blockchain gespeicherten Informationen, die für dieses System relevant sind, abrufen. Einen Überblick über die einzelnen Komponenten vermittelt Abbildung 3.1.

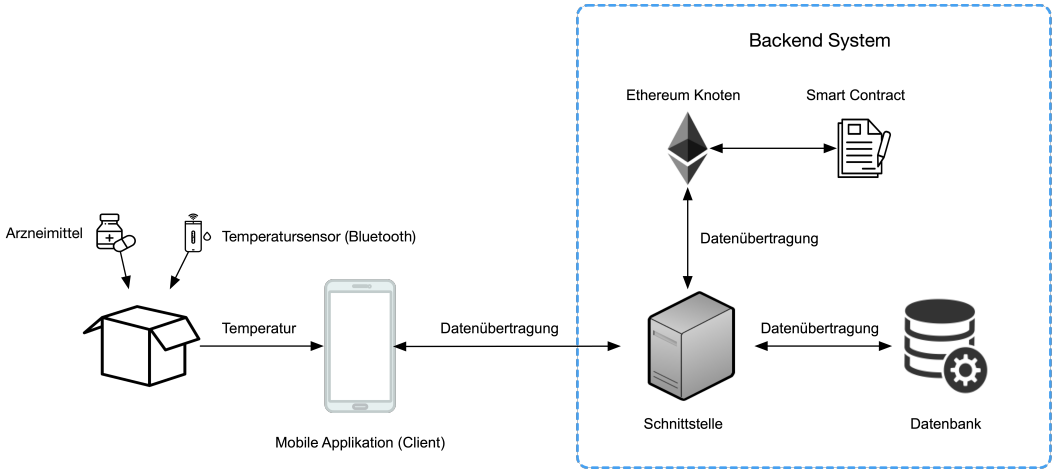

Abbildung 3.1: Übersicht der einzelnen Komponenten

Anforderungen an den Smart Contract

Folgende Vertragsbestandteile müssen von dem Smart Contract dieses Fallbeispiels eingehalten werden.

1. Jedes Produkt muss über eine eindeutige, einmalige ID zu identifizieren sein.
2. Die Temperatur innerhalb der Medikamentenverpackung darf einen festgelegten minimalen Grenzwert von 0,00°C nicht unterschreiten und einen maximalen Grenzwert von 4,00°C nicht überschreiten.

3. Sollte die Temperatur während des Prozesses zwischen den Grenzwerten nicht eingehalten werden, so mindert sich dementsprechend die Qualität des Produktes.
4. Das Ergebnis von Qualitätseinbußen trägt zur Minderung des Verkaufspreises bzw. zur endgültigen Stornierung bei, siehe auch Abschnitt 3.1.1.
5. Trifft ein Produkt beim Empfänger ein und entspricht der angegebenen Qualität, wird dieser dem Absender in Höhe des ausgewiesenen Produktpreises vergütet. Die Bezahlung erfolgt in Form der Währung *wei* und wird automatisiert vom Smart Contract durchgeführt.
    - Besitzt ein Produkt nicht mehr die angegebene Qualität, so wird nur der verminderte Betrag an den Absender gezahlt.
    - Sollte die Qualität nicht mehr den vertraglichen ausgehandelten Anspruch genügen, so wird diese als Ausschussware an den Absender zurückgeschickt.
6. Alle Teilnehmer gehören zu einer entsprechenden Nutzergruppe und besitzen bestimme Rechte.
7. Absender und Empfänger müssen unterschiedlich sein.
8. Nur Eigentümer eines Produktes dürfen dieses versenden.
9. Nur Eigentümer, mit bestimmten Rechten, dürfen den Verkaufspreis eines Produktes ändern.
10. Der Verkaufspreis des Händlers darf nicht kleiner als der Preis vom Produkthersteller sein.
11. Während der Zustellung eines Produktes, muss die aktuelle geografische Position an den Übergabepunkten der Teilnehmer gespeichert werden.

Qualitätsklassifizierung

- **Qualität A**: Aktuelle Temperatur liegt zwischen den vorgegebenen Grenzwerten.
- **Qualität B**: Aktuelle Temperatur ist $\leq$ 25% vom maximalen Grenzwert, so wird das Produkt auf die Qualität B heruntergestuft.

- **Qualität C**: Aktuelle Temperatur ist $\leq$ 50% vom maximalen Grenzwert, so wird das Produkt auf die Qualität C heruntergestuft.
- **Ausschuss Ware**: Temperaturabweichung beträgt mehr als 50% des maximalen Grenzwertes oder Temperatur liegt unter dem minimalen Grenzwert.

Nutzergruppe

1. **Hersteller** sind Eigentümer des Smart Contracts. Sie besitzen das Recht neue Produkte herzustellen und anzubieten sowie neue Hersteller, Versanddienstleister und Händler zu berufen.
2. **Versanddienstleister** befördern das Produkt zwischen Absender und Empfänger. Ein Versanddienstleister darf weitere Versanddienstleister berufen.
3. **Händler** besitzen das Recht Produkte vom Hersteller zu erwerben, den Verkaufspreis zu ändern und das Produkt für den weiteren Verkauf an den Endverbraucher anzubieten. Ebenfalls verfügen sie über das Recht neue Händler dem System hinzuzufügen.
4. **Endverbraucher**: Jeder Teilnehmer kann ein Endverbraucher sein, solange er nicht der aktuelle Besitzer des Produktes ist.

Anforderungen an die Schnittstelle

Durch die Anforderung an den Smart Contract ergeben sich folgende Funktionalitäten, die die Schnittstelle realisieren muss.

1. Alle erstellten Produkte werden durch eine Abfrage zurückgegeben.
2. Die detaillierte Ausspielung für ein einzelnes Produkt ist möglich.
3. Alle abgeschlossenen Phasen lassen sich für ein Produkt ausspielen.
4. Ausgabe aller in dieser Blockchain gespeicherten Blöcke.
5. Das Guthaben, die Nutzerrechte sowie die pro Account ausgeführten Transaktionen sollen ausgespielt werden.

6. Dem Nutzer muss es möglich sein, neue Produkte zu erstellen, erwerben, überprüfen, aktualisieren und den Transport zu veranlassen.

7. Auch lassen sich Accounts mit bestimmten Nutzerrechten erweitern.

Zusätzlich soll die Schnittstelle wesentliche Funktionalitäten für eine Nutzerverwaltung bereitstellen. Dazu gehört die Erstellung eines Nutzer, der in der Datenbank sicher gespeichert wird. Außerdem müssen die Nachrichten, die zwischen der Anwendung und der Schnittstelle ausgetauscht werden entsprechend asuthentifiziert sein.

3.1.2 Anwendungsfall

Mit den formulierten Anforderungen lassen sich folgende Anwendungsfälle beschreiben.

Tabelle 3.1: Registrierung eines neuen Nutzers (UC-001)

Abschnitt	**Inhalt**
Bezeichner	**UC-001**
Name	Registrierung eines neuen Nutzers.
Aktor	Jegliche Teilnehmer wie Hersteller, Versanddienstleister, Händler und Endverbraucher.
Vorbedingungen	**1.** Ein Server auf dem die Schnittelle und der Datenbank-Dienste aktiv ist. **2.** Der Aktor hat den Client geöffnet.
Nachbedingung	Die Nutzerinformationen des Aktor sind in der Datenbank gespeichert und er kann sich erfolgreich sich im System anmelden.
Ergebnis	Der Aktor wurde im System registriert.
Hauptszenario	**1.** Der Aktor gibt die nötigen Informationen in das Registrierungsformular des Clients ein. **2.** Der Client sendet die Informationen an die Schnittstelle. **3.** Diese Schnittstelle prüft ob der Aktor schon im System registriert ist. **4.** Daten werden in die Datenbank geschrieben. **5.** Die Schnittstelle übermittelt das Ergebnis der Registrierung an den Client.

Tabelle 3.1: Registrierung eines neuen Nutzers (UC-001) (Fortsetzung)

Abschnitt	**Inhalt**
Ausnahmeszenario	**3a.** Die gesendeten Daten sind nicht korrekt. **4a.** Daten werden nicht in die Datenbank geschrieben.

Tabelle 3.2: Anmelden eines Nutzers im System (UC-002)

Abschnitt	**Inhalt**
Bezeichner	**UC-002**
Name	Anmelden eines Nutzers im System.
Aktor	Jegliche Teilnehmer wie Hersteller, Versanddienstleister, Händler und Endverbraucher.
Vorbedingungen	**1.** Ein Server auf dem die Schnittelle und das Datenbanksystem aktiv sind. **2.** Der Aktor hat den Client geöffnet. **3.** Der Aktor ist im System registriert.
Nachbedingung	Der Aktor ist im System angemeldet und kann entsprechende Services nutzen.
Ergebnis	Der Aktor wird im System angemeldet.
Hauptszenario	**1.** Der Aktor gibt nötige Informationen in das Anmeldeformular des Clients ein. **2.** Der Client sendet die Informationen an die Schnittstelle. **3.** Diese Schnittstelle prüft ob die übermittelten Nutzerinformationen des Aktor korrekt sind. **4.** Der Aktor wird im System angemeldet. **5.** Die Schnittstelle übermittelt das Ergebnis der Anmeldung an den Client.
Ausnahmeszenario	**3a.** Übermittelte Daten sind nicht korrekt. **4a.** Der Aktor wird nicht im System angemeldet.

Tabelle 3.3: Erstellung einer digitalen Kopie (UC-003)

Abschnitt	**Inhalt**
Bezeichner	**UC-003**

Tabelle 3.3: Erstellung einer digitalen Kopie (UC-003) (Fortsetzung)

Abschnitt	**Inhalt**
Name	Erstellung einer digitalen Kopie aller relevanten Produkt- und Sendungsinformationen.
Aktor	Hersteller
Vorbedingungen	**1.** Ein Server auf dem die Schnittelle und das Datenbanksystem aktiv sind. **2.** Der Smart Contract wurde auf den Ethereum Knoten migriert. **3.** Der Aktor hat den Client geöffnet. **4.** Der Aktor ist im System angemeldet.
Nachbedingung	Produkt wird zum Kauf angeboten und die aktuellen Produkt- und Sendungsinformationen können in der Blockchain von jedermann angerufen werden.
Ergebnis	Produkt- und Sendungsinformationen sind in die Blockchain eingetragen.
Hauptszenario	**1.** Der Aktor gibt nötige Informationen in das Produktformular des Clients ein. **2.** Der Client sendet die Informationen an die Schnittstelle. **3.** Die Schnittstelle leitet die Daten an den Smart Contract weiter. **4.** Der Smart Contract prüft ob die Richtlinien eingehalten werden. **5.** Der Smart Contract schreibt jegliche Informationen für das Produkt in die Blockchain. **6.** Die Schnittstelle übermittelt das Ergebnis des Smart Contract an den Client.
Ausnahmeszenario	**4a.** Die Richtlinien wurden nicht korrekt eingehalten. **5a.** Jegliche Informationen für das Produkt werden nicht in die Blockchain geschrieben.

Tabelle 3.4: Erwerb eines Produktes (UC-004)

Abschnitt	**Inhalt**
Bezeichner	**UC-004**
Name	Erwerb eines Produktes.
Aktor	Händler, Endverbraucher

Tabelle 3.4: Erwerb eines Produktes (UC-004) (Fortsetzung)

Abschnitt	**Inhalt**
Vorbedingungen	**1.** Ein Server auf dem die Schnittelle und das Datenbanksystem aktiv sind. **2.** Der Smart Contract wurde auf den Ethereum Knoten migriert. **3.** Der Aktor hat den Client geöffnet. **4.** Der Aktor ist im System angemeldet. **5.** Produkt wird im Client zum Kauf vom Hersteller ausgewiesen.
Nachbedingung	Eingangsprüfung nach Erhalt des Produktes.
Ergebnis	Der Aktor wartet auf die Lieferung des Produktes.
Hauptszenario	**1.** Der Aktor wählt zu kaufendes Produkt aus. **2.** Der Aktor wählt einen Versanddienstleister aus. **3.** Der Client sendet die Informationen an die Schnittstelle. **4.** Die Schnittstelle leitet die Daten an den Smart Contract weiter. **5.** Der Smart Contract prüft ob die Richtlinien eingehalten werden. **6.** Der Smart Contract schreibt jegliche aktualisierte Informationen für das Produkt in die Blockchain. **7.** Die Schnittstelle übermittelt das Ergebnis des Smart Contract an den Client.
Ausnahmeszenario	**5a.** Die Richtlinien wurden nicht korrekt eingehalten. **6a.** Jegliche aktualisierte Informationen für das Produkt werden nicht in die Blockchain geschrieben.

Tabelle 3.5: Lieferung und Eingangsprüfung nach Erwerb eines Produktes. (UC-005)

Abschnitt	**Inhalt**
Bezeichner	**UC-005**
Name	Lieferung und Eingangsprüfung nach Erwerb eines Produktes.
Aktor	Händler, Endverbraucher
Vorbedingungen	**1.** Ein Server auf dem die Schnittelle und das Datenbanksystem aktiv sind. **2.** Der Smart Contract wurde auf den Ethereum Knoten migriert. **3.** Der Aktor hat den Client geöffnet. **4.** Der Aktor ist im System angemeldet.

Tabelle 3.5: Lieferung und Eingangsprüfung nach Erwerb eines Produktes.(UC-005) (Fortsetzung)

Abschnitt	**Inhalt**
	5. Das Produkt wurde vom Aktor gekauft. **6.** Das Produkte befindet sich auf dem Weg zum Aktor.
Nachbedingung	Das Produkt kann weiter verwendet werden.
Ergebnis	Das Produkt besteht die Eingangsprüfung.
Hauptszenario	**1.** Der Aktor nimmt das Produkt vom Versanddienstleister entgegen. **2.** Der Aktor führt mithilfe des Clients eine Eingangsprüfung des Produktes durch. **3.** Der Client sendet die Informationen an die Schnittstelle. **4.** Die Schnittstelle leitet die Daten an den Smart Contract weiter. **5.** Der Smart Contract prüft ob die Richtlinien eingehalten werden. **6.** Der Smart Contract schreibt jegliche aktualisierte Informationen für das Produkt in die Blockchain. **7.** Der Smart Contract transferiert automatisch den zu zahlenden Ether-Betrag an den Absender. **8.** Die Schnittstelle übermittelt das Ergebnis des Smart Contract an den Client.
Ausnahmeszenario	**5a.** Die Richtlinien wurden nicht korrekt eingehalten. **6a.** Jegliche aktualisierte Informationen für das Produkt werden nicht in die Blockchain geschrieben. **7a.** Der Smart Contract transferiert nicht den zu zahlenden Ether-Betrag an den Absender sondern klassifiziert das Produkt als Storno-Ware und sendet es, über den identischen Versanddienstleister, an den Absender zurück.

Tabelle 3.6: Produkt wird zum Verkauf vom Händler angeboten (UC-006)

Abschnitt	**Inhalt**
Bezeichner	**UC-006**
Name	Produkt wird zum Verkauf vom Händler angeboten.
Aktor	Händler
Vorbedingungen	**1.** Ein Server auf dem die Schnittelle und das Datenbanksystem aktiv sind. **2.** Der Smart Contract wurde auf den Ethereum Knoten migriert. **3.** Der Aktor hat den Client geöffnet. **4.** Der Aktor ist im System angemeldet. **5.** Der Aktor ist im Besitz mindestens eines Produktes. **6.** Das Produkt hat die Eingangsprüfung bestanden.
Nachbedingung	Das Produkt wird von einem Endverbraucher gekauft.
Ergebnis	Neuer Preis wird für das Produkt vom Aktor angegeben.
Hauptszenario	**1.** Der Aktor bestimmt einen neuen Preis für das Produkt. **2.** Der Client sendet die Informationen an die Schnittstelle. **3.** Die Schnittstelle leitet die Daten an den Smart Contract weiter. **4.** Der Smart Contract prüft ob die Richtlinien eingehalten werden. **5.** Der Smart Contract schreibt jegliche aktualisierte Informationen für das Produkt in die Blockchain. **6.** Die Schnittstelle übermittelt das Ergebnis des Smart Contract an den Client.
Ausnahmeszenario	**4a.** Die Richtlinien wurden nicht korrekt eingehalten. **5a.** Jegliche aktualisierte Informationen für das Produkt werden nicht in die Blockchain geschrieben.

3.1.3 Nicht-funktionale Anforderungen

Anforderung an die Sicherheit

Die Blockchain wird als sicheres System angesehen, siehe auch Abschnitt 2.7. Um die Sicherheit für das gesamte System zu gewährleisten, müssen auch die restlichen Komponenten über gewisse Sicherheitsanforderungen verfügen. So werden beispielsweise alle

Nutzerangaben mit einer starken Gewichtung als kryptografischer Hash-Wert in der Datenbank gespeichert. Außerdem wird der Nutzerzugriff zwischen Client und Schnittstelle durch einen geeigneten Authentifizierungsmechanismus geschützt.

Anforderung an die Leistung

Die Anforderung an ein leistungsstarkes System, in dem eine Blockchain Technologie eingesetzt wird, kann infolge der aktuellen Beschaffenheit der Blockchain nicht realisiert werden. Denn, wie in Abschnitt 2.4 beschrieben, dauert es in einer Ethereum Blockchain ca. 7 Sekunden, bis ein neuer Block erstellt wird. Auch der Einsatz der Bitcoin Blockchain kann dieses Leistungsproblem nicht lösen. Da dieses Fallbeispiel aber keine Echtzeitanforderung besitzt, kann dieser erhebliche Nachteil der Blockchain, ignoriert werden. Dennoch darf die Leistungsanforderung für die restlichen Komponenten nicht vernachlässigt werden.

Anforderungen an die Erweiterbarkeit, Integration und Wartbarkeit

Das System ist modular aufgebaut, sodass auch nach Fertigstellung dieser Arbeit eine Weiterentwickelt problemlos möglich ist. Somit kann die Verbesserung einzelner Komponenten beispielsweise als Thema für weitere Abschlussarbeiten dienen.

3.2 Bewertung bestehender Lösungen

Das folgende Kapitel vermittelt eine Auswahl über bestehende Supply Chain Management Applikationen, die die Technologie einer Ethereum Blockchain in Verbindung mit Smart Contracts einsetzen. Da diese Technologie sehr jung ist, existieren leider wenige Firmen, die über den Entwicklungsstand eines Prototypen hinaustreten.

modum.io AG

Eine der ersten Firmen, die die Blockchain Technologie in einer Supply Chain einsetzen, ist die modum.io AG. Sie wurde 2016 in der Schweiz gegründet und entwickelt ebenfalls ein entsprechendes Medikamenten-Tracking-System für die Pharmaindustrie. Dabei liegt der Fokus auf der Fertigung eines eigenes Datenloggers. Dieser soll neben der

Temperatur, die aktuelle Feuchtigkeit sowie die Bewegung für eine Medikamentensendung überwachen und die gewonnenen Daten an einen dazugehörigen Smart Contract weiterleiten. Zur Finanzierung startete diese junge Firma 2018 ihren eigenen Ethereum Token. Dieser, mit der Bezeichnung *MOD*, kann aktuell nur durch die Währung Bitcoin oder Ether erworben werden [28].

CryptoTec AG
Auch diese Kölner Firma nutzt die Blockchain Technologie, um mit deren Sicherheitseigenschaften die zu entwerfenden Software-Systeme der CryptoTec AG noch sicherer zu gestallten. Zu den Kunden gehören neben Firmen das Pharmaindustrie auch Unternehmen aus dem Bereich der Automobilindustrie oder der Versicherungsbranche. Die CryptoTec AG hat beispielsweise ein System entwickelt, dass eine Kombination aus der Blockchain Technologie und GSM-fähigen Sensoren darstellt. Mit diesem lässt sich der Lieferweg von Gütern, wie zum Beispiel Lebensmitteln, Medikamenten oder Blutkonserven in Echtzeit überwachen [7].

Weitere Projekte oder Startups, die bestehende oder neu entwickelte Supply Chain Management Systeme durch die Blockchain-Technologie ergänzen sind:

- **Everlederger**
  - Dieses Startup entwickelt ein System mit dessen Hilfe die Herkunft von Luxusartikeln ermittelt werden kann [13].
- **Shipchain**
  - Auch dieses Unternehmen hat ein Blockchain basiertes System entworfen, um Abläufe in der Logistik transparenter, sicherer und effizienter zu gestalten [45].

4 Konzeption und Implementierung

4.1 Architektur

Um die beschriebenen Anforderungen und Anwendungsfälle für das System umzusetzen, benötigt es eine entsprechende Systemarchitektur. Wie schon in Abschnitt 3.1 beschrieben, enthält das System mehrere Komponenten. Diese werden für einen detaillierteren Ausblick über die gesamte Architektur in zwei Funktionsbereiche, die logische und die Anwendungsebene, unterteilt. Für beide Bereiche werden die einzelnen Komponenten vorgestellt. Zusammengesetzt ergeben diese die Gesamtfunktion der einzelnen Bereiche.

4.1.1 Logische Ebene

Auf der logischen Ebene, Abbildung 4.1, befinden sich, der Wortbedeutung nach, alle Komponenten, die für die grundlegende Logik des Systems verantwortlich sind und auf einem Backend System laufen. Dazu gehört auch der Ethereum Knoten. Dieser führt den Smart Contract aus und interagiert mit allen weiteren, im Ethereum Netzwerk befindlichen, Knoten. Somit wird sichergestellt, dass jegliche Transaktionen, die mit dem Smart Contract in Verbindung stehen, sicher in neue Blöcke der Blockchain geschrieben und auf allen Ethereum Knoten gespeichert werden. Der Contract verifiziert dabei alle in Unterabschnitt 3.1.1 aufgeführten Anforderungen.

Damit das System der Anwendungsebene über ein Netzwerk mit der logischen Ebene kommunizieren kann, wird ein Webserver mit entsprechender Schnittstelle realisiert. Diese basiert auf der oft verwendeten REST-Architektur, die wiederum auf dem Hypertext Transfer Protocol (HTTP) aufbaut. REST basierte Lösungen haben sich zu einem Standard in der Entwicklung von Web-APIs etabliert. Die zustandslose RESTful-Schnittstelle (RESTful-API) ermöglicht einer Client-Anwendung, über erzeugte HTTP-Methoden, die wesentliche Logik des Servers zu konsumieren, indem sie Daten aus der Blockchain oder

der Datenbank abruft, neu erstellt oder modifiziert. Zudem sorgt sie für die Einhaltung des Nachrichtenaufbaus sowie der Authentifizierung ein- und ausgehender Nachrichten. Diese Komponente ist auch wieder in zwei Bereiche aufgeteilt. Der erste Teil verarbeitet und überwacht alle Smart Contract bezogenen Nachrichten, wohingegen der zweite Teil die Authentifizierung der einzelnen Nutzer umsetzt. Dafür ist die Schnittstelle mit einer MySQL Datenbank verbunden, in welcher die anwenderbezogenen Informationen gespeichert werden. Das System wird in Abbildung 4.1 visuell beschrieben.

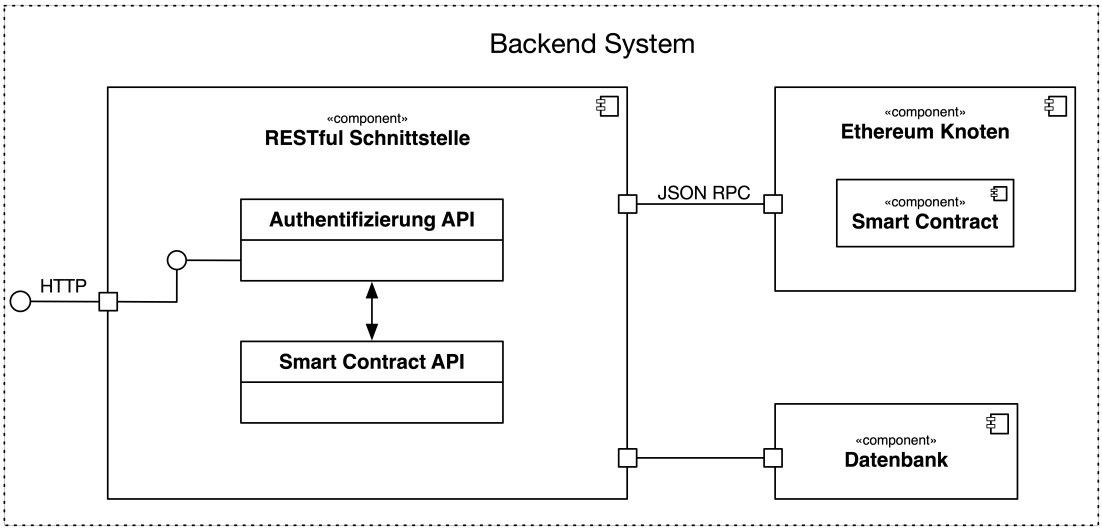

Abbildung 4.1: Komponentendiagramm der logischen Ebene

Der Ethereum Knoten stellt für die Kommunikation mit der Außenwelt eine RPC-Schnittstelle bereit. Durch diese ist die Interaktion zwischen Smart Contract und RESTful-API möglich.

4.1.2 Anwendungsebene

Die Seite der Anwendungsebene beinhaltet eine mobile Applikation (Client) und einen Temperatursensor. Dieser wird vom Hersteller für die Lebenszeit der Lieferung innerhalb der Sendung platziert. An den Übergabepunkten der Lieferkette wird die aktuelle geografische Position sowie die Temperatur der Lieferung an die Applikation übermittelt und von dort über die RESTful-API an den Smart Contract weitergeleitet. Des Weiteren kann die Applikation eine digitale Kopie für ein Produkt erstellen und erwerben. Außerdem ist es möglich den gegenwärtigen und vergangenen Produkt- und Lieferstatus

der Sendung über die Schnittstelle abzufragen. Eine weitere Anforderung ist die Authentifizierung der Nutzer, die über die Applikation an die Schnittstelle weitergeleitet wird. Dabei kann sich ein Nutzer neu im System registrieren oder anmelden. Mit Hilfe des Clients können zudem Nutzerrechte der Teilnehmer vergeben werden. Die Abbildung 3.1 stellt noch einmal die Kombination der Anwendungsebene mit der logischen Ebene dar.

4.1.3 Zusammenspiel der Funktionsbereiche

Anhand des Aktivitätsdiagramms wird das genaue Zusammenspiel der beiden Bereiche aufgezeigt. Das Diagramm zeigt einen detaillierten Ablauf vom Anmeldeprozess bis hin zur Erstellung eines neues Produktes für die Nutzergruppe der Hersteller.

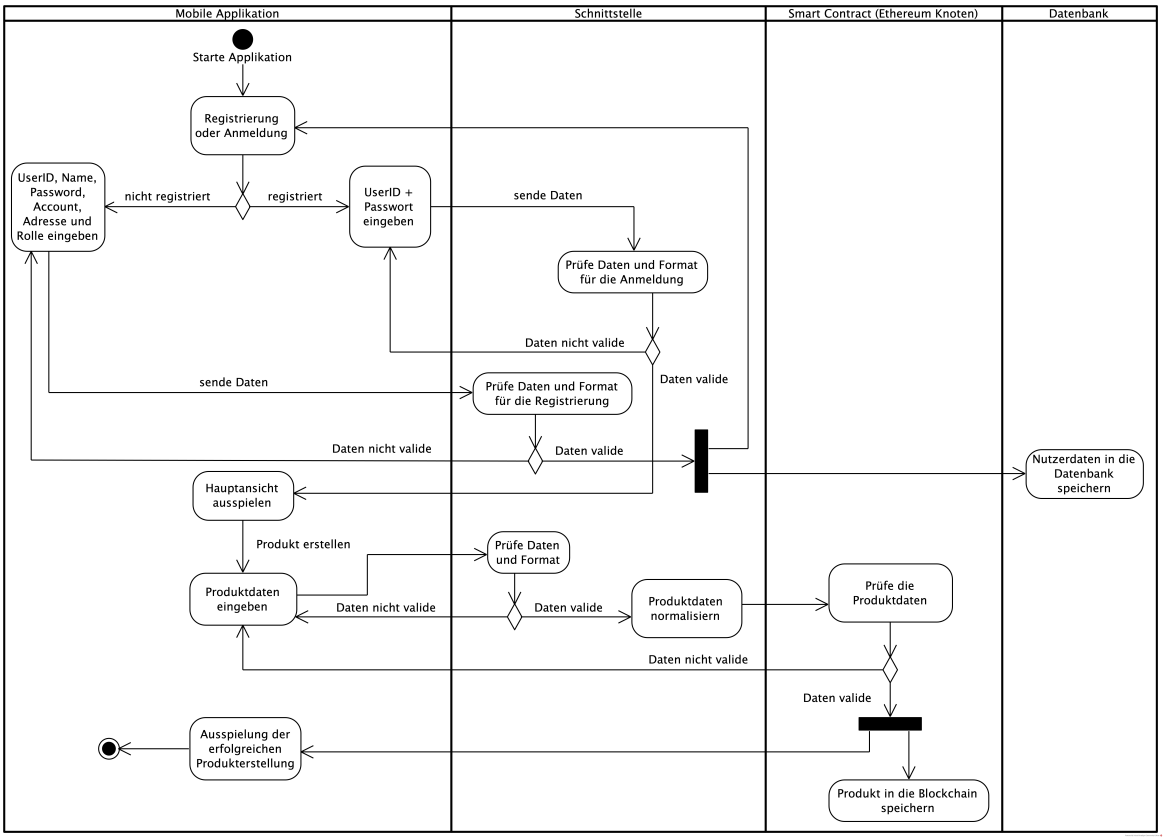

Abbildung 4.2: Aktivitätsdiagramm für die Nutzergruppe der Hersteller

4.2 Technologieentscheidungen

Auf Grundlage der zuvor beschriebenen Anforderungen sowie der aufgezeigten Architektur werden die eingesetzten Technologien, zur Realisierung des Systems, in diesem Abschnitt erläutert. Dabei wird nicht jede Technologie im Detail erklärt, da dies nicht Kern dieser Arbeit ist.

4.2.1 Programmiersprache Solidity

Für die Umsetzung eines Smart Contracts auf der Ethereum Blockchain gibt es aktuell zwei hervorstechende Programmiersprachen. Dies ist zum einem die Python-ähnliche Programmiersprache Vyper und zum anderen Solidity, die Ähnlichkeiten zu Javascript, C++ und ebenfalls Python aufweist. Gegenwärtig zählt Solidity als Allzweckprogrammiersprache für Smart Contracts innerhalb des Ethereum Projekts und wird auch in dieser Arbeit eingesetzt [47].

Eigenschaften von Solidity

Solidity besitzt einige Besonderheiten, die für die Entwicklung eines Smart Contracts bedeutsam sind. So muss man sich bei der Entwicklung aufgrund von Kapazitäts- und Kostengründen entsprechende Gedanken über die Speicherung der verschiedenen Parameter machen. Denn die Ethereum EVM verfügt über drei unterschiedliche Bereiche.

- **Memory**: Speicherung von temporären Variablen. Ist zudem sehr kostengünstig, da am Ende eines Funktionsaufrufes dieser wieder gelöscht wird.
- **Storage**: Enthält den Zustand eines Contracts, in dem Zustandsvariablen abgelegt werden. Dieser Bereich gilt als sehr kostspielig.
- **Stack**: Fast kostenloser Speicherbereich ,aber wenig Fassungsvermögen. Nur sinnvoll für nicht öffentliche Variablen.

Dass die Signatur einer Funktion immer mit dem Keyword *function* anfängt, kann als eine weitere Eigenschaft genannt werden. Darauf folgt die Parameterliste, die Sichtbarkeit (*private*, *public*, *internal* oder *external*) und die Modifier-Liste. Am Ende der Signatur erscheint eine Liste von Rückgabetypen, da sich in Solidity mehr als ein Rückgabetyp

definieren lässt. Die Modifier-Liste wird mit den Keywords *pure*, *view*, keinen der beiden Keywords oder einem eigenen Modifier gekennzeichnet.

- **pure**: Die Funktion besitzt keinem Interaktion mit einem anderen Smart Contract, außer mit einer anderen Funktion, die ebenfalls den Modifier *pure* besitzt. Ein Beispiel hierfür wäre eine Funktion, die zwei Werte miteinander multipliziert.
- **view**: Ist eine Funktion als *view* deklariert, ist diese aussschließlich eine lesende Funktion.
- **Kein Keyword**: Ist keines dieser vorherigen Keywords angegeben, so handelt es sich um eine schreibende Funktion. Bei der Ausführung wird eine Transaktion gegen das Blockchain Netzwerk gestartet, die von den Mining Knoten aufgegriffen und in neue Blöcke gespeichert wird.
- **Eigener Modifier**: Mit einem eigenen Modifier kann der Inhalt bzw. das Verhalten einer Funktion modifiziert und somit die eigentliche Funktion mit dem Inhalt der Modifier Funktion erweitert werden. Beispielsweise lässt sich somit der Zugriff auf eine Funktion auf bestimmte Nutzergruppen einschränken.

Eine weitere Eigenart der Sprache Solidity sind die Fallback-Funktionen. Hierbei handelt es sich um eine anonyme Funktion, die nur einmal pro Contract vorkommen darf. Diese Funktion wird ausgeführt, wenn Ether an die Adresse des Contracts gesendet wird ohne direkt mit einer anderen Funktion zu interagieren. Zudem wird sie ausgespielt, wenn eine nicht im Contract existierende Funktion fälschlicherweise aufgerufen wird.

Weitere Besonderheiten der Sprache Solidity sind, dass aktuell keine Gleitkommazahlen unterstütz werden und Festkommazahlen sich nur deklarieren, aber nicht verwenden lassen. Variablen vom Typ *address* besitzen Funktionen und Eigenschaften. Zudem verfügen jegliche Adressen im Ethereum Netzwerk über einen Kontostand, der als *balance* bezeichnet wird.

4.2.2 Open Zeppelin

Für einen Smart Contract, der für eine öffentliche Blockchain entwickelt wird, müssen gewisse Kriterien eingehalten werden. So ist zu beachten, dass jeder Teilnehmer den Speicher eines Smart Contracts auslesen kann. Außerdem kann das Verhalten durch

einen Funktionsaufruf verändert werden. Um dies zu verhindern, wird oftmals eine entsprechende Zugriffskontrolle für Smart Contracts implementiert. Hierbei wird nur entsprechenden Nutzergruppen der Zugriff auf bestimmte Funktionen gewährt. Durch die Verwendung der Open Zeppelin Bibliothek kann die Anforderung an die Nutzergruppen aus Abschnitt 3.1.1 sehr leicht und einfach umgesetzt werden. Zudem verfügt die Bibliothek über die Funktionalität mathematische Operationen zu überprüfen. Hiermit lassen sich auftretende Over- oder Underflows vermeiden [36].

4.2.3 Truffle

Mit dem Truffle Framework lässt sich die Entwicklung und das Testen des zu entwerfenden Smart Contracts wesentlich verbessern. Ein wichtiger Bestandteil dieses Frameworks ist die Ausführung von Unit- und Integrationstests für den implementierten Contract. Zudem lässt sich mit Truffle ein voll entwickelter Contract auf eine entsprechende Ethereum Blockchain migrieren. Ein Truffle Projekt weist folgende Struktur auf [52].

Tabelle 4.1: Struktur eines Truffle Projekts

Datei oder Ordner	**Beschreibung**
contracts/	In diesem Ordner befinden sich die einzelnen Smart Contracts.
migrations	Dieses Verzeichnis beinhaltet eine ausführbare JavaScript Datei, mit deren Hilfe der Smart Contract auf die Ethereum Blockchain migriert wird.
test	Alle Testfälle werden in diesem Ordner hinterlegt.
truffle-config.js	Allgemeine Truffle Konfigurationsdatei.

4.2.4 Ganache

Ganache kann als ein privates Ethereum Testnetzwerk angesehen werden und ist die einfachste Möglichkeit, das zu entwerfende System auf einem Ethereum Netzwerk zu testen. Dabei simuliert das Programm, welches lokal auf dem Rechner installiert wird, das Verhalten einer realen Ethereum Blockchain. Bei der Verwendung von Ganache werden dem Nutzer zehn Test-Accounts mit jeweils 100 Ether zur Verfügung gestellt. Ein Vorteil ist die komplette Unterstützung des Truffle Frameworks [51].

4.2.5 Flask

Der Webserver und die RESTful-Schnittstelle wird mit Hilfe des Web-Frameworks Flask entwickelt. Flask wurde 2010 veröffentlicht und ist komplett in Python geschrieben. Es gilt als einfach, gut erweiterbar und sehr benutzerfreundlich. Flask besitzt keine native Unterstützung, beispielsweise für die Authentifizierung von Nutzern, den Zugriff auf Datenbanken oder die Validierung von Webformularen. Dadurch wird die Hauptfunktionalität von Flask sehr gering gehalten. Gerade Firmen, die auf eine Microservice-Architektur setzen, nutzen des Öfteren dieses noch junge Framework [41].

4.2.6 Web3.py

Mit Hilfe der Python Bibliothek Web3.py ist über das JSON-RPC Protokoll eine Interaktion zwischen der RESTful-API und dem Smart Contract im Netzwerk möglich. So können alle im Contract implementierten Funktionen aufgerufen werden. Außerdem lassen sich weitere Informationen aus der Blockchain abfragen. Dazu gehört neben der Ausgabe aller, in der Blockchain vorhandenen Ethereum Adressen, auch das zur Verfügung stehende Guthaben pro Account [55].

4.2.7 MySQL Datenbank

Die Speicherung der Nutzerinformationen wird durch das relationale Datenbanksystem MySQL umgesetzt. MySQL ist ein Open Source Projekt und weltweit sehr verbreitet. Das System verfügt über eine hohe Performanz und kann somit eine große Vielzahl von Daten verarbeiten. Anfragen an die Datenbank werden durch die Datenbanksprache SQL ausgeführt [37].

4.2.8 Postman

Mit Hilfe von Postman lässt sich die Funktionalität der RESTful-API testen. Für diese Überprüfung wird die Nutzung einer Client Anwendung nicht benötigt. So kann die gesamte Schnittstelle getestet werden, ohne dass ein Client, der die RESTful-API Endpunkte aufruft, vorhanden sein muss [39]

4.2.9 Vagrant und Ansible

Für die lokale Entwicklung und das Testen müssen alle Komponenten der logischen Ebene korrekt und ohne aufwendige Installationsschritte ausgeführt werden. Um dies zu gewährleisten, wird eine virtuelle Maschine mit Hilfe von Vagrant in Kombination mit Ansible erstellt und verwaltet. Da beide Tools plattformunabhängig sind, entsteht eine identische Infrastruktur für die Entwicklung des Systems. In der Vagrant Community[1] findet man frei zugänglich vorkonfigurierte Maschinen inklusive Betriebsystem, die sich zeitnah auf dem Host-System ausführen lassen. Ansible ist dabei ein Open-Source Orchestrierungstool, das dabei hilft die Konfigurations- und Anwendungsverwaltung in dem System zu automatisieren. So werden jegliche Abhängigkeiten während des Installationsprozesses der Maschine selbständig umgesetzt [17] [1].

4.2.10 Mobile Applikation

Für den mobilen Client wird eine entsprechende Android Applikation entwickelt . Dieses, auf einem Linux-Kernel basierende Betriebssystem, findet z.B. Einsatz auf Smartphones, Fernsehern sowie Wearable Computern und wird als Open-Source-Projekt von der Open Handset Alliance entwickelt [35]. Android unterstützt u.a. auch die Programmiersprache Java, welche über entsprechende Schnittstellen Zugriff auf die benötigten Kernfunktionen und Systemdienste des Betriebssystems erhält. Somit lässt sich mit der Android Applikation die aktuelle geografische Postion an den Übergabepunkten und die Temperatur mittels Bluetooth Schnittstelle vom Sensor ermitteln.

4.2.11 Temperatursensor

Als Temperatursensor wird das SHT31 Smart Gadget Development Kit der Firma Sensirion verwendet. Das Gadget beinhaltet den SHT31 Feuchtigkeits- und Temperatursensor. Zudem ist es BluetoothLE (BLE)-kompatibel [44]. Dadurch ist eine Kommunikation mit der Android Anwendung gewährleistet. Durch den Sensor lassen sich aktuelle und vergangene Messwerte für die Temperatur und die Feuchtigkeit abfragen. So können auftretende Temperaturabweichungen an den Übergabepunkten der Warensendung überprüft werden.

[1] https://app.vagrantup.com/boxes/search

4.3 Implementierung des Backend Systems

Die Umsetzung orientiert sich an dem Komponentendiagramm aus Unterabschnitt 4.1.1. Als Grundsystem für die virtuelle Maschine dient ein Ubuntu 18.04 als Betriebssystem. Dieses Linux System wird oft und erfolgreich im produktiven Einsatz von Backend Servern eingesetzt. Zusätzlich werden durch Ansible weiter benötigte Programmiersprachen, Frameworks und Tools wie Solidity, Python, MySQL, Flask, Truffle und Ganache installiert. Der Smart Contract wird mit Hilfe der Truffle Bibliothek kompiliert und auf dem Ganache Netzwerk ausgeführt.

4.3.1 Implementierung des Smart Contracts

Die größte Herausforderung bestand in der Implementierung des Smart Contracts. Hierbei musste man sich nicht nur in eine neue Programmiersprache einarbeiten, sondern auch die Kosten einzelner Transaktionen beachten. So wurde bei der Entwicklung des Contracts darauf geachtet, die Kosten durch unnötige Implementierungen in Maßen zu halten. Beispielsweise liegen aktuell[2] die Transaktionskosten für eine einfache **for**-Schleife, die von $0 <= 50$ zählt, einem Gas Preis von $2\,Gwei$ und einer durchschnittlichen Bearbeitungsdauer bei $0,03048\,\$$. Würde man die Bearbeitungszeit der Transaktion erhöhen, so würde auch der Gas Preis auf $20\,Gwei$ ansteigen und sich die Transaktionskosten auf $0.30474\,\$$ pro ausgeführter Transaktion erhöhen. Folgende Tabelle 4.2 zeigt einen Ausschnitt der Gaskosten pro Operation [5]. Eine vollständige Auflistung findet man im Ethereum Yellow Paper von Wood [57].

Tabelle 4.2: Übersicht der Gaskosten pro Operation

Operation	**Gaskosten**	**Beschreibung**
ADD/SUB	3	Arithmetische Operation
MUL/DIV	5	Arithmetische Operation
ADDMOD/MULMOD	8	Arithmetische Operation
AND/OR/XOR	3	Bitweise logische Verknüpfung
LT/GT/SLT/SGT/EQ	3	Vergleichsoperation
POP	2	Stack Operation
PUSH/DUP/SWAP	3	Stack Operation

[2] Stand vom 21.08.2019

Tabelle 4.2: Übersicht der Gaskosten pro Operation (Fortsetzung)

Operation	**Gaskosten**	**Beschreibung**
MLOAD/MSTORE	3	Speicheroperation (Memory)
SLOAD	200	Speicheroperation
SSTORE	5000/20000	Speicheroperation
CREATE	32000	Mit *CREATE* ein neues Konto erstellen

Eine weitere Einschränkung lag in der Größ des Contracts. Denn in der Ethereum Blockchain dürfen Smart Contracts wegen des EIP-170[3] eine Gesamtgröße von 24 KB nicht überschreiten. Aus diesem Grund werden für bestimmte Variablen auch effektivere Datentypen eingesetzt. Beispielsweise wird für den Produktnamen und die geografische Position, anstelle des Datentyps *String*, der effiziente Datentyp *byte32* verwendet. In der gesamten Implementierung des Smart Contracts wird der Typ *String* nur für die Beschreibung des Artikels verwendet.

Damit bestimmte Funktionen, wie das Versenden oder die Überprüfung der Ware, nicht fälschlicherweise doppelt ausgeführt werden, wurde ein Mutex-ähnliches Verhalten durch boolesche Kontrollvariablen umgesetzt. Dazu wird die Ausführung für ein Produkt so lange gesperrt, bis sie an andere Stelle wieder freigegeben wird.

Aufgrund der Tatsache, dass Solidity aktuell noch keine Gleit- und Festkommazahlen unterstützt, musste auch für die Teilanforderung des Smart Contracts aus Abschnitt 3.1.1 die Einteilung der Temperaturskala verändert werden. Für diese Umwandlung wurden jegliche arithmetische Berechnungen in der RESTful-API durchgeführt, wodurch sich die eigentlichen Kosten des Smart Contracts minimiert ließen. Der Contract erhält von der Schnittstelle nur eine Ganzzahl, um mit dieser weiterzuarbeiten. Als Ergebnis wird Schlussendlich eine Ganzzahl vom Contract an die Schnittstelle übermittelt, die dort wiederum in eine Dezimalzahl umgewandelt wird. Laut den Anforderungen aus Abschnitt 3.1.1, liegt der gültige Temperaturbereich zwischen 0,00°C und 4,00°C. Da im Contract die Temperatur vom Datentyp *uint* ist und dieser keine negativen Zahlen darstellen kann, wird der eigentlich Nullpunkt um vier ganzzahlige Werte in den positiven Bereich verschoben. Dadurch ergibt sich ein neuer interner Nullpunkt und zugleich minimaler Grenzwert von 4,00°C und ein maximaler Wert von 8,00°C. Damit eine eingehende

[3] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-170.md

Dezimalzahl auch im Contract korrekt verarbeitet und für die Qualitätsberechnung verwendet werden kann, wird das Komma durch eine Multiplikation verschoben.

Beispielrechnung

Der maximale Grenzwert für die A-Qualität des Produktes liegt bei 4,00°C. Der Temperatursensor im Paket liefert eine Wert von 6,89°C. Dies entspricht einen Anstieg von 72.25%. Damit der Smart Contract eine fehlerfreie Qualitätsberechnung, anhand der übermittelten Temperatur, durchführen kann, sind folgende Berechnungen in der RESTful-Api und dem Contract notwendig.

Berechnung in der RESTful-API

$Temp_{max} = 4,00 \quad x = 6,89$

$y = (4,00 + 6,89) * 100000 = 1089000$

Qualitätsberechnung im Contract

$z = \frac{1089000}{40} - 20000 = 7225$

Laufzeitsicht des Smart Contracts

In der Abbildung 4.3 wird die Laufzeitsicht des Smart Contracts grafisch dargestellt. Ein Medikament wird vom Hersteller für den Kauf freigegeben. Dieses kann von einem verifizierten Händler erworben werden. Im Anschluss sendet der Hersteller das Produkt mit der Funktion *transferProduct()* an den entsprechenden Empfänger. Dort angekommen, wird die Qualität des Medikaments überprüft. Bei erfolgreichem Ergebnis bzw. nicht schlechter als Qualität C, siehe Abschnitt 3.1.1, wird der Warenpreis für den Hersteller berechnet und automatisch an dessen Ethereum Adresse gesendet. Zusätzlich kann der Händler einen neuen Verkaufspreis für dass Produkt festsetzen. Wird die Qualität der Ware nicht eingehalten, so wird diese in der Funktion *setStorno()* als Ausschuss deklariert und umgehend an den Hersteller zurückgeschickt.
Das identische Vorgehen spiegelt sich auch zwischen Händler und Endverbraucher wider. Allerdings ist ein Endverbraucher nicht in der Lage einen neuen Verkaufspreis für das erworbene Medikament zu setzen.

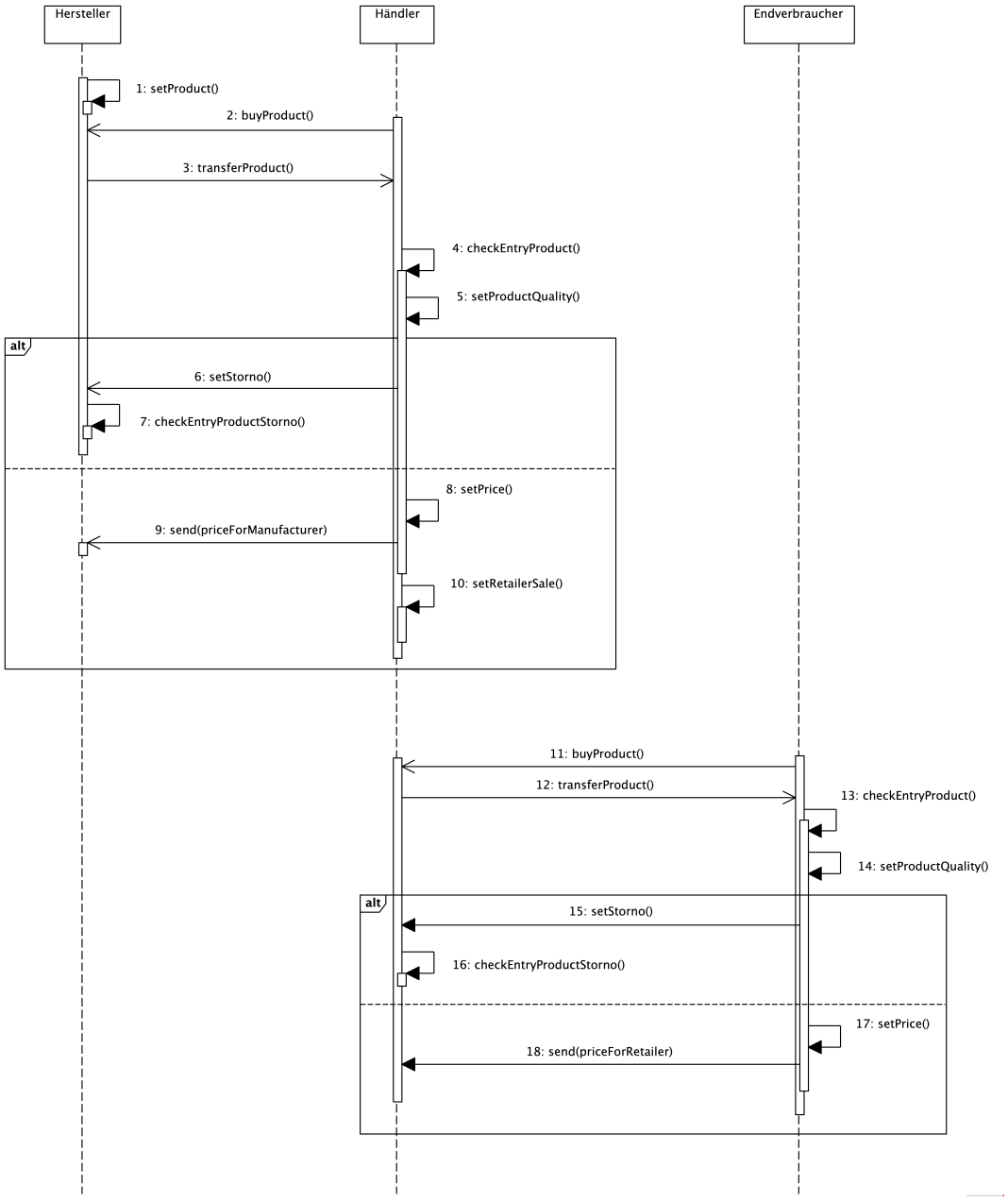

Abbildung 4.3: Laufzeitsicht des Smart Contracts

4.3.2 Implementierung der RESTful-Schnittstelle

Bei der Umsetzung wurde auf die Einhaltung der Eigenschaften einer RESTful-Schnittstelle geachtet. Dazu gehört, dass sich jede Ressource über einen eindeutigen Unique Ressource Identifiers (URI) identifizieren lässt. Eine URI besteht aus einem Schema, in diesem Fall das HTTP oder HTTPS Protokoll, dem Host samt Port, dem Pfad und optionalen Parametern. Durch die HTTP-Methoden GET, POST und PUT wird von der Android Anwendung die benötigte Ressource angefordert, erstellt oder verändert.
Bei einer RESTful-API wird auch immer ein DELETE Befehl implementiert. Jedoch wird in dieser Umsetzung darauf verzichtet. Denn auf der Blockchain geschriebene Informationen können nicht gelöscht werden. Auch ist die Entfernung eines Nutzer in den Anforderungen an das System nicht vorgesehen. Neben diesen Befehlen gibt es noch weitere, wie PATCH, HEAD und OPTIONS. Diese werden aber für die Umsetzung der Schnittstelle nicht benötigt.

Anhand der Anforderungen aus Unterabschnitt 3.1.1 wurden folgende Endpunkte für die Schnittstelle realisiert.

Tabelle 4.3: Übersicht der RESTful-API Endpunkte

Methode	**Ressource**	**Pfad/Parameter**	**Beschreibung**
GET	Smart Contract	/api/accounts	Guthaben und Transaktionen pro Account
GET	Smart Contract	/api/block	Ausgabe der Blöcke
GET	Smart Contract	/api/product/{upc}	Produktinformation
GET	Smart Contract	/api/product	Ausgabe Aller Produkte
GET	Smart Contract	/api/steps/{upc}	Produktphasen für ein Produkt
GET	Smart Contract	/api/{stakeholder}/{address}	Überprüfung Hersteller, Versanddienstleister und Händler
GET	Nutzer	/api/users	Ausspielung aller User
POST	Smart Contract	/api/{stakeholder}	Rechte für den Hersteller, Versanddienstleister und Händler setzen
POST	Smart Contract	/api/product	Produkt erstellen
POST	Nutzer	/api/register	Nutzer erstellen

Tabelle 4.3: Übersicht der RESTful-API Endpunkte (Fortsetzung)

Methode	**Ressource**	**Pfad + Parameter**	**Beschreibung**
POST	Nutzer	/api/login	Anmeldung des Nutzers
POST	Nutzer	/api/refresh	Neuen JWT-Token erstellen
PUT	Smart Contract	/api/buyproduct/{upc}	Produkt erwerben
PUT	Smart Contract	/api/transferproduct{upc}	Produkt versenden
PUT	Smart Contract	/api/{entry}/{upc}	Eingangsprüfung des Produktes
PUT	Smart Contract	/api/retailersale/{upc}	Verkaufspreis beim Händler setzen

Zusätzlich übernimmt die Schnittstelle die Typumwandlung von einem nutzerfreundlichen Format in dem vom Smart Contract geforderten Datentyp und wieder zurück , siehe auch Unterabschnitt 4.3.1. Weitere Inhalte der übermittelten Nachrichten oder Fehlermeldungen werden ebenfalls in ein lesbares Format konvertiert. Beispielsweise werden Fehlermeldungen, die der Contract ausgibt, durch einen regulären Ausdruck umgewandelt und dem Nutzer leserlich ausgespielt.

Für den Austausch und die Darstellung der Ressourcen zwischen Anwendung und API wird das leicht verständliche JSON-Format verwendet. Im Grunde wäre es auch möglich andere Formate wie XML zu wählen. Doch das JSON-Format hat sich zu einem Standart bei der Entwicklung von Web-Schnittstellen entwickelt.

Sicherheit und Authentifizierung der Nutzer

Eine weitere Anforderung an die Schnittstelle besteht in der sicheren Speicherung der Nutzerinformationen sowie der Authentifizierung von übertragenen Nachrichten während der Verwendung des Systems. Für einen Nutzer werden in der MySQL-Datenbank nur die UserID, der Name, das Passwort, der Ethereum Account, die öffentliche Adresse des Accounts und die Nutzerrolle gespeichert. Aufgrund dieser Angaben ist es wichtig, dass im Falle eines Angriffs auf die Datenbank, das Passwort nicht als Klartext, sondern als verschlüsselter Wert in der Datenbank gespeichert wird. Dies wird durch den bcrypt-Algorithmus umgesetzt, der mittels der Python Bibliothek *bcrypt* in das System der Schnittstelle eingebunden wird. Der bcrypt-Algorithmus gilt als relativ sicher. Er

benötigt viel Rechenzeit und erschwert somit Brute-Force-Angriffe. Aktuell gilt dieser Algorithmus als Standard für die Passwort-Verschlüsselung [32].

Die Authentifizierung der Nachrichten wird durch den JSON Web Token (JWT) realisiert. Mit Hilfe dieses Tokens werden die übertragenen Nachrichten zwischen der Android Applikation und der Schnittstelle signiert und verschlüsselt. Dadurch können die Ressourcen, während der Aufrufe an die Schnittstelle, nicht von dritten, unautorisierten Nutzern eingesehen werden. Für die Umsetzung wird die Python Bibliothek *Flask-JWT-Extended* verwendet, die sich einfach in das bestehende System einbinden lässt.

Wie in Unterabschnitt 4.2.6 beschrieben, hilft die Python Bibliothek Web3.py bei der Kommunikation zwischen der RESTful-API und dem Ethereum Knoten. Mit Hilfe dieser Bibliothek werden eingehende Nachrichten an den Contract weitergeleitet und so die Ausgabe der Schnittstelle bereitgestellt.

4.3.3 Aufbau der Datenbank

Die eingesetzte MySQL Datenbank beinhaltet nur die Tabelle für die Speicherung der Nutzerinformationen. Dazu gehört die User-ID, der Name, das Passwort, der verwendete Ethereum Account, die dazugehörige öffentliche Ethereum Adresse sowie die Angabe der Nutzergruppe des Teilnehmers.

4.4 Implementierung der Android Applikation

Infolge der Anforderungen aus Abschnitt 3.1 muss die Android Applikation in der Lage sein alle Endpunkte der Schnittstelle zu verarbeiten, siehe Tabelle 4.3. Für die HTTP-Kommunikation zwischen Anwendung und RESTful-API wird die Netzwerkbibliothek Retrofit eingesetzt. Durch diese typsichere Bibliothek lassen sich Ressourcen an die Schnittstelle abfragen und übermitteln.

Um die aktuelle geografische Position am Übergabepunkt der Sendung zu ermitteln, wird die Google Play Service Bibliothek eingesetzt. Diese baut auf dem Android Location Manager auf, ist aber wesentlich genauer und effizienter als das Android Äquivalent [40].

Die einzelnen Ansichten der Android Applikation sind grafisch im Anhang dargestellt, siehe Abbildung A.1 und Abbildung A.2.

4.4.1 Kommunikation mit dem Temperatursensor

Als Problem hat sich jedoch die Kommunikation zwischen dem Sensirion Smart Gadget und der Android App herausgestellt. Unter Verwendung der offiziellen Sensirion Bibliothek ist es nicht möglich einen Temperaturwert mit dem zu testenden Smartphone[4] auszulesen. Die Verwendung der Sensirion Demo Applikation spielt nicht das erhoffte Ergebnis aus. Somit wird die Android Anwendung für die manuelle Eingabe des Temperaturwertes erweitert. Nebenbei wird eine alternative Lösung, ohne Einbindung der Sensirion Bibliothek, umgesetzt. Die Smart Gadget BLE Dokumentation[5] hat dabei zur Lösung des Problems beigetragen. Das Dokument gibt Auskunft über die vom Sensor eingesetzten UUID Adressen, die zum GATT Profil gehören. GATT steht für Generic Attribute und steuert den Datenfluss zwischen einem BluetoothLE Gerät und dem Client. Um Strom zu sparen ist das GATT Profil hierarchisch aufgebaut. Auf der obersten Ebene sind die Service UUIDs definiert, zu denen verschiedene charakteristische UUIDs gehören [50].

Anhand der UUID Adresse kann der Temperaturwert von der Android Applikation ausgelesen und das zuvor aufgezeigte Problem mit der Sensirion Bibliothek umgangen werden. Dennoch bleibt die manuelle Eingabe des Temperaturwertes in der Applikation als Sicherheit bestehen.

4.5 Test der Implementierung

Die Überprüfung des gesamten Systems in Bezug auf eine korrekte Funktionsweise ist unabdingbar. Anhand der Anforderungen aus Abschnitt 3.1 ergeben sich unterschiedliche Szenarien, die zu testen sind, siehe Abbildung 4.4. Besondere Beachtung, neben der grundlegenden Funktionalität, erklärt dabei das Verhalten zwischen dem Smart Contract und den Ethereum Accounts. So wird beispielsweise die Einhaltung der Nutzerrechte, der Produktqualität und das vom Contract eingeleitete Verhalten bei Nichterfüllung

[4] Samsung Galaxy S mini, Android 6.0.1, API Level 23

[5] https://github.com/Sensirion/SmartGadget-Firmware/blob/master/Simple_BLE_Profile_Description.pdf

der Vorgaben, kontrolliert. Mit Hilfe der Truffle Bibliothek lässt sich eine exemplarische Kommunikation mit dem Smart Contract als Integrationstest im Ganache Netzwerk durchführen. Dabei ist zu beachten, dass jede zu testende Transaktion kostenpflichtig ist.

```
Contract: SupplyChainMain
  ✓ 1. Test: UPC-1: Die Produkt-UPC sollte noch nicht vergeben sein.
  ✓ 2. Test: Rechte der Stakeholder (Hersteller, Haendler, Versanddienstleister) testen (588ms)
  ✓ 3. Test: UPC-1: Product erstellen und Ausgabe ueberpruefen. (121ms)
  ✓ 4. Test: UPC-1: Ausgabe des Produkt Status.
  ✓ 5. Test: UPC-1: Ausgabe des Produkt Flags.
  ✓ 6. Test: UPC-2: Gibt es zwei identische Produkt-UPCs. (202ms)
  ✓ 7. Test: UPC-1: Kauf eines Produktes (Haendler). (165ms)
  ✓ 8. Test: UPC-1: Versende Produkt vom Hersteller zum Haendler. (164ms)
  ✓ 9. Test: UPC-1: Ueberpruefe das eingegangene Produkt beim Haender => Ergebnis A-Ware.  (178ms)
  ✓ 10. Test: UPC-1: Neuen Verkaufspreis beim Haendler fuer das Produkt setzen. (149ms)
  ✓ 11. Test: UPC-1: Kauf eines Produktes (Endverbraucher). (127ms)
  ✓ 12. Test: UPC-1: Versende Produkt vom Haendler zum Endverbraucher. (170ms)
  ✓ 13. Test: UPC-1: Ueberpruefe das eingegangene Produkt beim Endverbraucher => Ergebnis A-Ware. (146ms)
  ✓ 14. Test: UPC-3: Neues Produkt erstellen und an Haendler senden. (224ms)
  ✓ 15. Test: UPC-3: Ueberpruefe B-Ware beim Haendler. (114ms)
  ✓ 16. Test: UPC-4: Neues Produkt erstellen und an Haendler senden. (279ms)
  ✓ 17. Test: UPC-4: Ueberpruefe C-Ware beim Haendler. (111ms)
  ✓ 18. Test: UPC-5: Neues Produkt erstellen und an Haendler senden. (207ms)
  ✓ 19. Test: UPC-5: Ueberpruefe den Storno Artikel beim Haendler. (144ms)
  ✓ 20. Test: UPC-6: Neues Produkt erstellen und an den Haendler senden. (197ms)
  ✓ 21. Test: UPC-6: Ueberpruefe die B-Ware beim Haendler, setze neuen Verkaufspreis, und sende es an den Endverbraucher. (319ms)
  ✓ 22. Test: UPC-6: Ueberpruefe B-Ware beim Endverbraucher. (114ms)
  ✓ 23. Test: UPC-7: Neues Produkt erstellen und an den Haendler senden. (191ms)
  ✓ 24. Test: UPC-7: Ueberpruefe C-Ware beim Haendler, neuen Verkaufspreis setzen und an den Endverbraucher senden. (367ms)
  ✓ 25. Test: UPC-7: Ueberpruefe C-Ware beim Endverbraucher. (118ms)
  ✓ 26. Test: UPC-8: Neues Produkt erstellen und an den Haendler senden. (204ms)
  ✓ 27. Test: UPC-8: Ueberpruefe C-Ware beim Haendler, neuen Verkaufspreis setzen und an den Endverbraucher senden. (287ms)
  ✓ 28. Test: UPC-8: Ueberpruefe Storno-Ware beim Endverbraucher. (140ms)
  ✓ 29. Test: UPC-9: Ueberpruefe die Gleitkommazahl => Temperatur mit 2 Nachkommastellen. (97ms)
  ✓ 30. Test: UPC-10: Überprüfe die Qualitaet => Die Qualitaet des Produktes ist zu schlecht. (41ms)
  ✓ 31. Test: UPC-10: Überprüfe die min. Temperatur (Grenze: (U))  => Die Temperatur muss groesser der min. Temperatur. (39ms)
  ✓ 32. Test: UPC-10: Überprüfe die max. Temperatur (Grenze: (O))  => Die Temperatur darf den max. Wert nicht ueberschreiten. (48ms)
  ✓ 33. Test: UPC-11: Überprüfe die min. Temperatur (Grenze: (O)). (71ms)
  ✓ 34. Test: UPC-12: Überprüfe die max. Temperatur (Grenze: (U)). (74ms)
  ✓ 35. Test: UPC-13: Neues Produkt erstellen und an den Haendler senden. (246ms)
  ✓ 36. Test: UPC-13: Ueberpruefe C-Ware beim Haendler, neuen Verkaufspreis setzen und an den Haendler senden. (296ms)
  ✓ 37. Test: UPC-13: Ueberpruefe Storno Artikel beim Consumer und die Eingangspruefung des stornierten Artikels beim Haendler. (252ms)
  ✓ 38. Test: UPC-14: Neues Produkt erstellen und an den Haendler senden. (182ms)
  ✓ 39. Test: UPC-14: Ueberpruefe Storno Artikel beim Haendler und die Eingangspruefung des stornierten Artikels beim Hersteller. (196ms)

39 passing (7s)
```

Abbildung 4.4: Integrationstest des Smart Contract mit Hilfe der Truffle Bibliothek

Für die korrekte Funktionsweise der RESTful-API, werden die Endpunkte der Schnittstelle mit Hilfe der Postman Anwendung getestet. Dafür wurden vier Szenarien erstellt, die die Anforderungen aus Unterabschnitt 3.1.1 überprüfen.

1. Szenario:

   a) Nutzer wird im System angemeldet (POST).

   b) Die Nutzerrechte für Hersteller, Händler, Endverbraucher und Versanddienstleister werden gesetzt (POST).

   c) Ein neues Produkt wird erstellt (POST).

   d) Händler kauft das erstelle Produkt (PUT).
   e) Das Produkt wird vom Hersteller an den Hersteller verschickt (PUT).
   f) Eingangsprüfung beim Händler (PUT).
   g) Händler setzt einen neuen Verkaufspreis (PUT).
   h) Überprüfung der einzelnen GET-Anfragen.

2. Szenario:
   a) Nutzer wird im System angemeldet (POST).
   b) Ein neues Produkt wird erstellt (POST).
   c) Händler kauft das erstelle Produkt (PUT).
   d) Das Produkt wird vom Hersteller an den Hersteller verschickt (PUT).
   e) Eingangsprüfung beim Händler (PUT).
   f) Händler setzt einen neuen Verkaufspreis (PUT).
   g) Endverbraucher kauft das erstelle Produkt (PUT).
   h) Das Produkt wird vom Händler an den Endverbraucher verschickt (PUT).
   i) Eingangsprüfung beim Endverbraucher (PUT).
   j) Überprüfung der einzelnen GET-Anfragen.

3. Szenario:
   a) Wie Szenario 2a - 2d (POST/PUT).
   b) Eingangsprüfung beim Händler. Die Temperatur des Produktes ist zu hoch, daher wird es sofort als Storno-Ware an den Hersteller zurückgeschickt(PUT).
   c) Eingangsprüfung der Storno-Ware beim Hersteller (PUT).
   d) Überprüfung der einzelnen GET-Anfragen.

4. Szenario:
   a) Wie Szenario 2a - 2h (POST/PUT).

b) Eingangsprüfung beim Endverbraucher. Die Temperatur des Produktes ist zu hoch, daher wird es sofort als Storno-Ware an den Händler zurückgeschickt(PUT).

c) Eingangsprüfung der Storno-Ware beim Händler (PUT).

d) Überprüfung der einzelnen GET-Anfragen.

5 Fazit und Ausblick

Anhand einer Supply Chain Management Applikation wurden in dieser Arbeit die Vor- und Nachteile durch den Einsatz der Ethereum Blockchain und Smart Contracts in untersucht. Hierbei wurde für das reales Fallbeispiel, siehe Abschnitt 1.4, ein System, bestehend aus folgenden Komponenten entwickelt: Ethereum Blockchain, Smart Contract, RESTful-API, Datenbank, Android Applikation und Temperatursensor.

Schon während der theoretischen Einarbeitung in die Blockchain und Smart Contracts, wurden viele Vorteile für ein Supply Chain Management System ersichtlich. Gerade die dauerhafte und nicht veränderliche Speicherung aller ausgeführten Aktionen in einer Lieferkette, spricht für den Einsatz der Blockchain. Ein weiterer Vorteil besteht in der Einhaltung rechtlicher Pflichten unter allen Vertragspartnern, durch die Verwendung von Smart Contracts. Zusätzlich überzeugen weitere Eigenschaften der Blockchain, wie Sicherheit, Transparenz und Ausfallsicherheit. So lässt sich neues Vertrauen unter den Vertragspartnern aufbauen, wo vorher keines vorhanden war.

Aber auch Nachteile wurden durch den Einsatz der neuen Technologie entdeckt. Neben dem hohen Energieverbrauch der Blockchain traten allein bei der Entwicklung des Smart Contracts unerwartete Fallstricke auf. Diese spiegeln sich beispielsweise in der maximalen Größe eines Contracts oder den aufkommenden Transaktionsgebühren wider. Außerdem ist eine einfache Aktualisierung des Contracts auf der Blockchain nicht möglich, da alle gespeicherten Informationen dauerhaft in der Blockchain gespeichert und nicht entfernt werden können. Darüber hinaus ist die Verständlichkeit der Programmiersprache Solidity für Nicht-Informatiker sehr komplex. Bei einem Rechtsstreit zwischen verschiedenen Vertragspartnern der Supply Chain, besteht heutzutage die Herausforderung schon in der einfachen Darstellung schwieriger Vertragsinhalte. Da ein Smart Contract diese komplexen Inhalte widerspiegelt, ist es für Nicht-Informatiker unmöglich Fehler vor Gericht aufzudecken und Rechtsstreitigkeiten niederzulegen.

Während der Einsatz der Blockchain und Smart Contract Technologie in einem Supply Chain Management System für Vor- und Nachteile sorgte, verlief die Verwendung der

RESTful-API ohne Probleme. Selbst erhebliche Nachteile der Programmiersprache Solidity, wie die fehlende Unterstützung von Gleit- und Fließkommazahlen, konnte in der Implementierung der Schnittstelle beseitigt werden. Zudem wurde bei der Entwicklung der API darauf geachtet, das Gesamtsystem weiterhin sicher zu gestalten. Dafür wurde die Kommunikation zwischen der Schnittstelle und der Android Anwendung durch den Einsatz eines JSON Web Tokens signiert und verschlüsselt. Zusätzlich wurden die Benutzerkennwörter in der Datenbank durch den kryptografischen bcrypt-Algorithmus verschlüsselt.

Im Vergleich dazu verlief die Entwicklung der mobilen Android Applikation nicht fehlerfrei. Die korrekte Umsetzung des Temperatursensors bedarf der ausführlichen Einarbeitung in das GATT Profils. Nur über dieses Profil lässt sich der Temperatursensor von der Android Applikation ansprechen und der aktuelle Wert an den Übergabepunkten auslesen. Dabei funktioniert noch nicht die Übermittlung vergangener Werte. So ist es nicht möglich, Temperaturabweichungen für einen zurückliegenden Zeitpunkt der Lieferung zu ermitteln.

5.1 Ausblick

Im Allgemeinen spricht eine Vielzahl von Vorteilen für den Einsatz der Ethereum Blockchain in Verbindung mit Smart Contracts in einem Supply Chain Management System. Jedoch dürfen die Nachteile hierbei nicht außer Acht gelassen werden. Damit diese Technologie den Einsatz in einem realen, produktiven System findet, muss noch bei vielen Anwendern das theoretische Verständnis für diese Technologie geschaffen und Überzeugungsarbeit geleistet werden. Gerade die anfallenden Transaktionskosten sorgen bei vielen Teilnehmern für großes Misstrauen hinsichtlich dieser Technologie. Außerdem schrecken weitere Nachteile wie der hohe Energieverbrauch einer Blockchain und die noch nicht ausgereifte Programmiersprache Solidity viele Akteure vor dem Einsatz ab. Des Weiteren wird der Pluspunkt der Transparenz innerhalb der Blockchain von vielen Anwendern als Nachteil gewertet, da alle Informationen frei zugänglich und dauerhaft in der Blockchain verfügbar sind.

Außerdem ist es sinnvoll, den Einsatz des Temperatursensors zu verbessern. Bis jetzt wird die aktuelle Temperatur lediglich an den Übergabepunkten mit dem definierten Sollwert vom Smart Contract abgeglichen. Dabei macht es Sinn, die Temperatur während der gesamten Lieferung in Echtzeit zu überwachen. Dafür soll der Sensor in der Lage

sein, die Messwerte in einem bestimmten Intervall über die Schnittstelle an den Smart Contract zu senden. Dieser kontrolliert die vertraglichen Inhalte und leitet gegebenenfalls weitere Aktionen ein. Zudem muss ein verbesserter Sensor kostengünstig produziert werden können, um eine Vielzahl an Lieferungen möglich zu machen. Im Übrigen sollte geprüft werden, ob es wirklich sinnvoll ist, jede Sendung mit einem Sensor auszustatten. Denn jeder Sensor, der während der Reise die Daten an die Schnittstelle sendet, vergrößert ganz automatisch das Nachrichtenaufkommen in dem verteilten Peer-to-Peer System.

Literaturverzeichnis

[1] AnsibleWorks, Inc.: *Ansible Documentation*. 2019. – URL https://docs.ansible.com/ansible/latest/index.html. – Zugriffsdatum: 2019-08-21

[2] Antonopoulos, Andreas M.: *Bitcoin & Blockchain*. 2. Auflage. Heidelberg : dpunkt.verlag GmbH, 2018

[3] BitcoinProject: *Bitcoin Developer Reference*. 2019. – URL https://bitcoin.org/en/developer-reference#mempool. – Zugriffsdatum: 2019-08-20

[4] Buterin, Vitalik: Ethereum White Paper A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM. (2014)

[5] Chen, T. ; Li, X. ; Luo, X. ; Zhang, X.: Under-optimized smart contracts devour your money. In: *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb 2017, S. 442–446

[6] Christopher, Martin: *Logistics & Supply Chain Management*. 4. Auflage. Edinburgh : Pearson UK, 2011

[7] CryptoTec: *CryptoTec AG*. 2019. – URL https://www.cryptotec.com/de/supply-chain-security/. – Zugriffsdatum: 2019-08-19

[8] Diffie, W. ; Hellman, M. E.: Privacy and authentication: An introduction to cryptography. In: *Proceedings of the IEEE* 67 (1979), March, Nr. 3, S. 397–427. – ISSN 0018-9219

[9] Eckert, Claudia: *IT-Sicherheit*. 10. Auflage. Berlin/Boston : Walter de Gruyter GmbH, 2019

[10] Ertel, Wolfgang ; Löhmann, Ekkehard: *Angewandte Kryptographie*. 5., überarbeitete und erweiterte Auflage. München : Carl Hanser Verlag, 2018

[11] ETHDOCS.ORG: *Ethereum Homestead Documentation*. 2016. – URL http://www.ethdocs.org/en/latest/. – Zugriffsdatum: 2019-08-21

[12] ETHEREUM.ORG: *Proof-of-Stake*. 2019. – URL https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ. – Zugriffsdatum: 2019-08-21

[13] EVERLEDGER: *Everledger*. 2019. – URL https://www.everledger.io. – Zugriffsdatum: 2019-08-21

[14] FERTIG, Tobias ; SCHÜTZ, Andreas: *Blockchain für Entwickler*. 1. Auflage. Bonn : Rheinwerk Verlag GmbH, 2019

[15] GREENSPAN, Gideon: *Blockchains vs centralized databases*. 05 2016. – URL https://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/. – Zugriffsdatum: 2019-08-21

[16] HÄBERLEIN, Tobias: *Praktische Algorithmik mit Python*. 1. Auflage. München : Oldenbourg Wissenschaftsverlag GmbH, 2012

[17] HASHICORP: *Vagrant Documentation*. 2019. – URL https://www.vagrantup.com/docs/index.html. – Zugriffsdatum: 2019-08-21

[18] HEIDELBERG, CONCEPT: *GDP: Wird für jeden Transport eine Temperaturmessung gefordert?* 06 2015. – URL https://www.gmp-navigator.com/gmp-news/gdp-wird-fuer-jeden-transport-eine-temperaturmessung-gefordert. – Zugriffsdatum: 2019-08-21

[19] HERTIG, Alyssa: *What is Ethereum?* 2019. – URL https://www.coindesk.com/information/what-is-ethereum. – Zugriffsdatum: 2019-08-21

[20] HINCKELDEYN, Johannes: *Blockchain-Technologie in der Supply Chain - Einführung und Anwendungsbeispiele*. 1. Auflage. Wiesbaden : Springer Vieweg, 2019

[21] INFORMATIONSTECHNIK, Bundesamt für Sicherheit in der: BSI - Technische Richtlinie - Kryptographische Verfahren: Empfehlungen und Schlüsssellängen. (2019), 02. – URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=10

[22] JAISWAL, Sweta: *Is Blockchain a Game-Changer for Healthcare?* 04 2018. – URL http://ctan.math.utah.edu/tex-archive/macros/latex/contrib/glossaries/glossariesbegin.pdf. – Zugriffsdatum: 2019-08-21

[23] Lange, Guido: *Ethereum-Erfinder Vitalik Buterin: Technisch ist alles für die Umstellung auf PoS vorbereitet.* 2019. – URL https://block-builders.de/ethereum-erfinder-vitalik-buterin-technisch-ist-alles-fuer-die-umstellung-auf-pos-vorbereitet/. – Zugriffsdatum: 2019-08-21

[24] Lehmacher, Wolfgang: *Die Welt der Logistik.* Wiesbaden : Springer Fachmedien Wiesbaden, 2013. – URL https://doi.org/10.1007/978-3-8349-4296-8_1. – ISBN 978-3-8349-4296-8

[25] Loop, Peter: *Blockchain: The Next Evolution of Supply Chains.* 2016. – URL https://www.mhlnews.com/global-supply-chain/blockchain-next-evolution-supply-chains. – Zugriffsdatum: 2019-08-21

[26] Merz, Michael: *Blockchain im B2B-Einsatz.* 1. Auflage. Hamburg : MM Publishing, 2019

[27] Mihm, Andreas: *Ein elektronischer Pass für jedes Arzneimittel.* 02 2019. – URL https://www.faz.net/aktuell/wirtschaft/wie-securpharm-europaeische-arzneimittel-vor-plagiaten-schuetzen-soll-16031549.html. – Zugriffsdatum: 2019-08-21

[28] modum.io: *modum.io AG.* 2019. – URL https://modum.io/. – Zugriffsdatum: 2019-08-10

[29] Morrison, Donald R.: PATRICIA&Mdash;Practical Algorithm To Retrieve Information Coded in Alphanumeric. In: *J. ACM* 15 (1968), Oktober, Nr. 4, S. 514–534. – URL http://doi.acm.org/10.1145/321479.321481. – ISSN 0004-5411

[30] Mueller, Holger: *Blockchains im Supply Chain Management 4.0.* 2018. – URL https://www.htwk-leipzig.de/no_cache/hochschule/aktuelles/newsdetail/artikel/1486/. – Zugriffsdatum: 2019-08-15

[31] Nakamoto, Satoshi: *Bitcoin: A Peer-to-Peer Electronic Cash System.* 2008. – URL https://bitcoin.org/bitcoin.pdf. – Zugriffsdatum: 2019-08-21

[32] Niels Provos ; David Mazieres: A FutureAdaptable Password Scheme. (1999). – URL https://www.usenix.org/legacy/event/usenix99/provos/provos.pdf

[33] NIST: *Fips Pub 199*. National Institute of Standards and Technology. 2004. – URL http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.199.pdf. – Zugriffsdatum: 2017-09-17

[34] NIST: *Federal Information Processing Standards Publication 186-4*. 2013

[35] Open Handset Alliance: *Industry Leaders Announce Open Platform for Mobile Devices*. 08 2019. – URL http://www.openhandsetalliance.com/press_110507.html. – Zugriffsdatum: 2019-08-21

[36] OpenZeppelin: *OpenZeppelin Contracts*. 2019. – URL https://docs.openzeppelin.com/contracts/2.x/. – Zugriffsdatum: 2019-08-21

[37] Oracle Corporation: *MySQL Documentation*. 2019. – URL https://dev.mysql.com/doc/. – Zugriffsdatum: 2019-08-20

[38] Paar, Christof ; Pelzl, Jan: *Kryptografie verständlich Ein Lehrbuch für Studierende und Anwender*. 1. Auflage. Heidelberg : Springer Vieweg, 2016

[39] Postman, Inc.: *Postman Documentation*. 2019. – URL https://learning.getpostman.com/docs/postman/launching_postman/installation_and_updates/. – Zugriffsdatum: 2019-08-21

[40] Reto Meier ; Ian Lake: *Professional Android*. 4. Auflage. Indianapolis : John Wiley & Sons, Inc., 2019

[41] Ronacher, Armin ; contributors: *Flask Documentation*. 2019. – URL https://flask.palletsprojects.com/en/1.1.x/. – Zugriffsdatum: 2019-08-21

[42] Rosenberger, Patrick: *Bitcoin und Blockchain: Vom Scheitern einer Ideologie und dem Erfolg einer revolutionären Technik*. 1. Auflage. Berlin, Heidelberg : Springer Berlin Heidelberg, 2018

[43] Schmeh, Klaus: *Kryptografie Kryptografie - Verfahren, Protokolle, Infrastrukturen*. 5., aktualisierte Auflage. Heidelberg : dpunkt.verlag GmbH, 2013

[44] Sensirion AG: *SHT31 Smart Gadget Development Kit*. 2019. – URL https://www.sensirion.com/de/umweltsensoren/feuchtesensoren/development-kit/. – Zugriffsdatum: 2019-08-21

[45] Shipchain: *Shipchain*. 2019. – URL https://shipchain.io. – Zugriffsdatum: 2019-08-10

[46] Sixt, Elfriede: *Bitcoins und andere dezentrale Transaktionssysteme: Blockchains als Basis einer Kryptoökonomie*. 1. Auflage. Wiesbaden : Springer Fachmedien Wiesbaden, 2017

[47] solidity.readthedocs.io: *Solidity Documentation*. 2019. – URL https://solidity.readthedocs.io/en/v0.5.2/. – Zugriffsdatum: 2019-08-20

[48] Steinmetz, Ralf ; Wehrle, Klaus: Peer-to-Peer-Networking & -Computing - Aktuelles Schlagwort. In: *Informatik Spektrum* 27 (2004), Nr. 1, S. 51–54. – URL https://doi.org/10.1007/s00287-003-0362-9

[49] Szabo, Nick: *Smart Contracts: Building Blocks for Digital Free Markets*. 1997. – URL https://archive.is/zWbhL#selection-607.412-607.427. – Zugriffsdatum: 2019-08-21

[50] Townsend, Kevin: Introduction to Bluetooth Low Energy. (2019)

[51] Truffle Blockchain Group: *Ganache Overview*. 2019. – URL https://www.trufflesuite.com/docs/ganache/overview. – Zugriffsdatum: 2019-08-21

[52] Truffle Blockchain Group: *Truffle Overview*. 2019. – URL https://www.trufflesuite.com/docs/truffle/overview. – Zugriffsdatum: 2019-08-21

[53] vfa: *Illegale Arzneimittel: Achtung Fälschung!* 2019. – URL https://www.vfa-patientenportal.de/arzneimittel/arzneimittelsicherheit-1/illegale-arzneimittel-achtung-faelschung.html. – Zugriffsdatum: 2019-08-21

[54] Vyper: *Vyper Documentation*. 2019. – URL https://vyper.readthedocs.io/en/latest/index.html. – Zugriffsdatum: 2019-08-21

[55] web3py.readthedocs.io: *Web3.py Documentation*. 2019. – URL https://web3py.readthedocs.io/en/stable/. – Zugriffsdatum: 2019-08-18

[56] Winterstein, Sebastian: *Was ist Supply Chain Management? Definition, Beispiel & Ziele!* 2019. – URL https://www.mm-logistik.vogel.de/was-ist-supply-chain-management-definition-beispiel-ziele-a-614558/. – Zugriffsdatum: 2019-08-21

[57] Wood, Dr. G.: ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER EIP-150 REVISION. (2014)

A Anhang

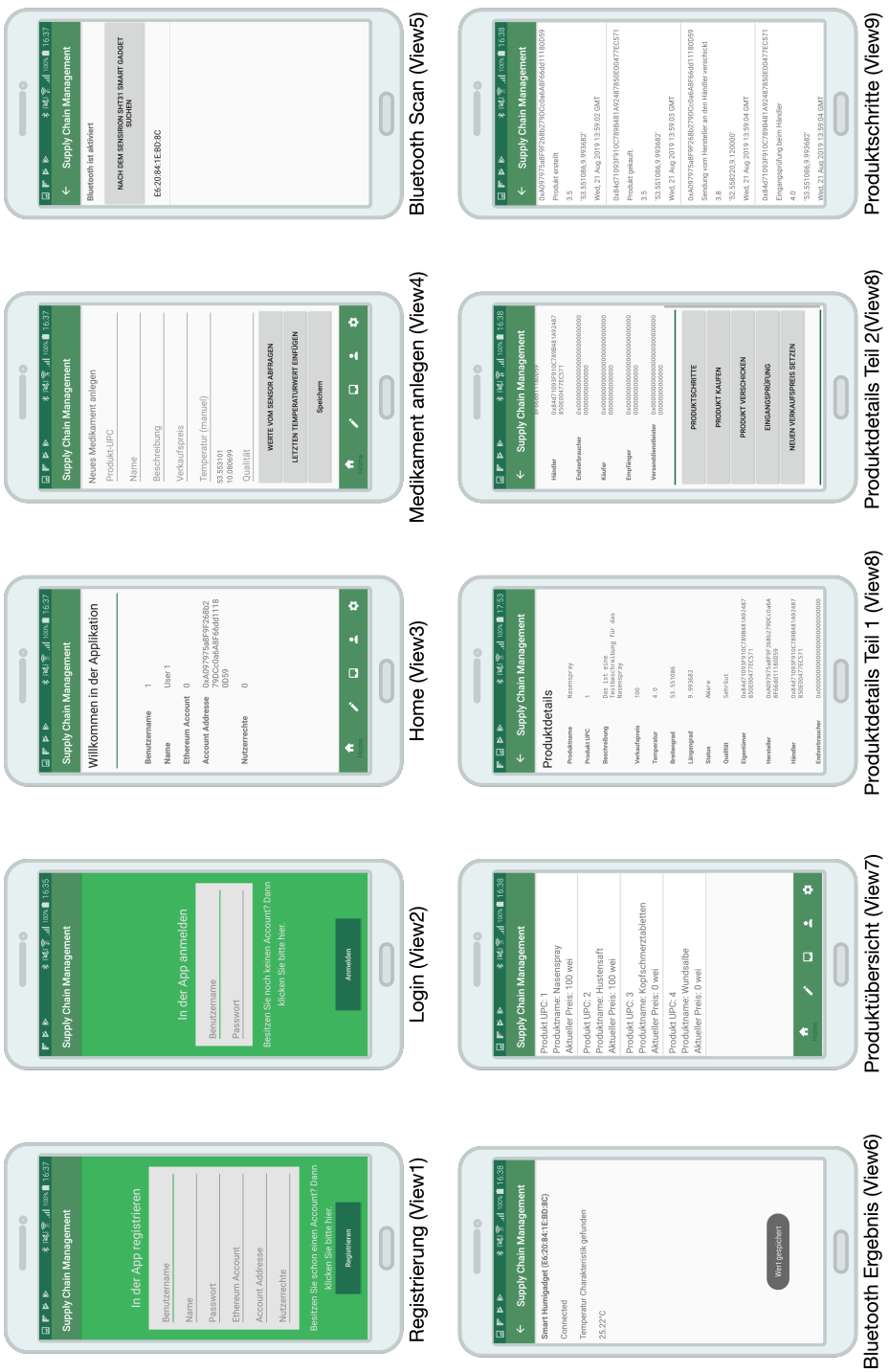

Abbildung A.1: View Übersicht der Android Applikation - 1. Teil

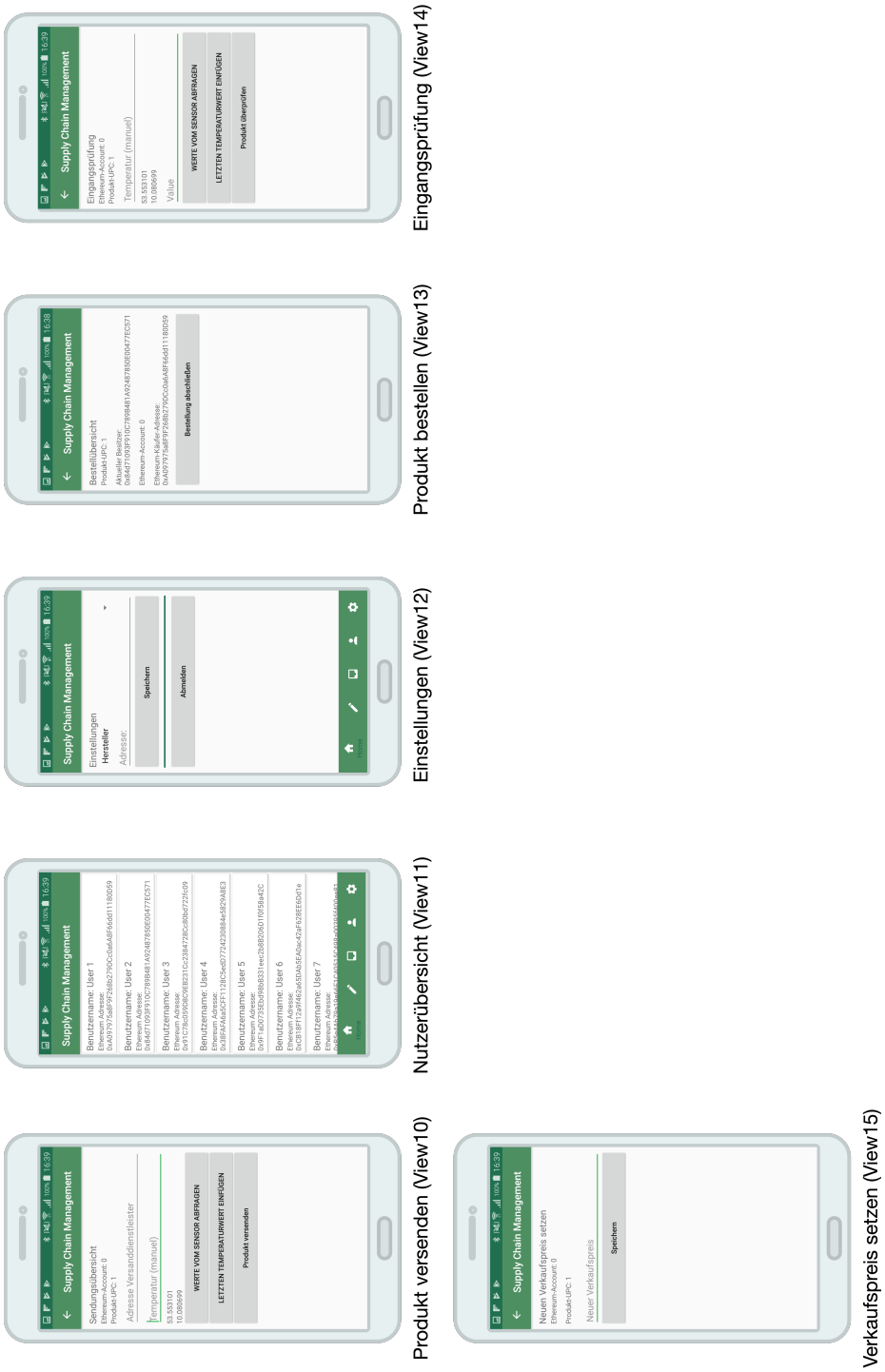

Abbildung A.2: View Übersicht der Android Applikation - 2. Teil

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „ bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§18 Abs. 1 APSO-TI-BM bzw. §21 Abs. 1 APSO-INGI)] ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen.“

Quelle: §16 Abs. 5 APSO-TI-BM bzw. §15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzeption und Implementierung einer verteilten Supply Chain Management Applikation auf Basis von Smart Contracts und der Ethereum Blockchain

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und 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