UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR SOFTWARETECHNIK
UND PROGRAMMIERSPRACHEN

Nahrungsmittelfrische: Absichern der Kontrollkette durch unveränderliche Datenspeicherung

Freshness of Food Consumer Goods: Ensuring a Chain of Custody Through Immutable Data Storage

Bachelorarbeit

verfasst am
Institut für Softwaretechnik und Programmiersprachen

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Max Henning Junghans

ausgegeben und betreut von
Prof. Dr. Martin Leucker

mit Unterstützung von
Mohammad Khodaygani und Tobias Braun

Lübeck, den 01. Oktober 2022

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Max Henning Junghans

Zusammenfassung

In der heutigen Zeit gewinnen Lieferketten immer weiter an Bedeutung. Sie sind in alle Lebensbereiche verwickelt. Besonders relevant sind die Lieferketten für die Nahrungsmittelversorgung. Hier muss vorwiegend die Frische des Essens sichergestellt werden. Dazu gehören unter anderem Kühlketten, die durchweg gewahrt werden müssen.

Diese Bachelorarbeit befasst sich mit der Findung eines geeigneten Systems, um die in der Lieferkette anfallenden Daten vor Veränderungen geschützt zu speichern. Dafür wird ein Vergleichssystem aufgestellt. Evaluiert werden klassische Datenbanken, Amazon Quantum Ledger Database als zentralisierten Ledger, öffentliche Blockchains und BigchainDB. Als besten geeignet hat sich BigchainDB herausgestellt. Es erfüllt die Kriterien, ist ausreichend performant und skalierbar, sowie bezahlbar.

Ein Proof of Concept für dieses System wurde entwickelt. Dieser zeigt die Funktionalität dieses Systems.

Abstract

In today's world, supply chains are becoming increasingly important. They are involved in all areas of life. Supply chains are particularly relevant for the food supply. Here, above all, the freshness of the food must be ensured. This includes, among other things, cold chains that must be maintained throughout.

This bachelor thesis deals with finding a suitable system to store the data accumulating in the supply chain in a way that protects it from changes. For this purpose, a comparative system is set up. Classic databases, Amazon Quantum Ledger Database as a centralized ledger, public blockchains and BigchainDB are evaluated. BigchainDB turned out to be the most suitable. It meets the criteria, is sufficiently performant and scalable, and is affordable.

A proof of concept for this system was developed. This shows the functionality of this system.

Danksagungen

Ich habe vielen Menschen zu danken.
Zunächst danke ich Prof. Leucker für die Gelegenheit, diese Arbeit verfassen zu können. Vom ISP danke ich weiter Mohammad Khodaygani und Tobias Braun, die mich beim Verfassen der Arbeit betreut haben. Auch danke ich meinen Freunden und meiner Familie für ihre moralische Unterstützung. Im Speziellen danke ich für Ideen beim Proof of Concept Sven und für das Korrekturlesen meiner Mutter und meinem Vater, André, meiner Tante Gesa, meinem Onkel Hermann, Prof. Calero Valdez und meinen Freunden Benedikt, Christoph, Cornelia, Dennis, Fabi, Flo, Jennie, Jorge, Magnus, Marie, Marine und Max.

Inhaltsverzeichnis

1

Einleitung

Lieferketten sind ein wichtiger Bestandteil der Lebensmittelindustrie. Über die Zeit sind diese immer größer und komplexer geworden. In der heutigen Zeit können sie um den ganzen Globus gehen und hunderte Parteien umfassen.

Besonders in den letzten Jahren wurden die Lieferketten vor große Herausforderungen gestellt. Durch Corona kamen viele Produktionen zum Stillstand. Dadurch kam es teils zu großen Verzögerungen oder Umstellungen in der Produktion, um die Teile auszutauschen, die nicht geliefert werden konnten. Ein weiterer Vorfall, der sich auf die Lieferketten global ausgewirkt hat, war die Blockade des Suezkanals im März 2021 durch ein Containerschiff. Dadurch konnte der Kanal für sechs Wochen nicht befahren werden.

Außerdem sind strenge Vorgaben einzuhalten. So ist in der Lebensmittelindustrie die Frische der Lebensmittel ein wichtiger Aspekt. Kommt es hier zu Komplikationen, kann es gefährlich werden, wenn die Lebensmittel schlecht werden. Hier eröffnen sich viele Möglichkeiten, die Einhaltung dieser Herausforderungen mit innovativen Ideen und Techniken zu kontrollieren.

1.1 Beiträge dieser Arbeit

Diese Arbeit befasst sich mit der Entwicklung eines geeigneten Systems zur Protokollierung von Daten entlang einer Lieferkette. Die besondere Anforderung ist, dass die Daten unveränderbar gespeichert werden und vor Manipulationsversuchen geschützt sind. Um dies zu erreichen, habe ich ein Vergleichssystem entworfen, nach welchem die verschiedenen Systeme verglichen werden können. Ich habe klassische Datenbanken, den zentralisierten Ledger von AWS namens Amazon Quantum Ledger Database, öffentliche Blockchains und die Blockchain-Datenbank BigchainDB analysiert und bewertet, ob sie für die Problemstellung geeignet sind. In der Arbeit stelle ich ausführlich dar, dass klassische Datenbanken für das System nicht geeignet sind, da sie die Anforderungen der Unveränderlichkeit nicht erfüllen. Des Weiteren stelle ich fest, dass Amazon Quantum Ledger Database die Anforderungen für diesen spezifischen Anwendungsfall nicht erfüllt.

In ähnlichen Situationen mit leicht anderen Anforderungen ist dies jedoch eine geeignete Lösung. Bei der Betrachtung der öffentlichen Blockchains bin ich zu dem Schluss gekommen, dass diese nicht skalieren und aus diesem Grund nicht geeignet sind. Im Vergleich zu den anderen Optionen hat sich herausgestellt, dass BigchainDB alle Anforderungen erfüllt und am ehesten geeignet ist. Danach habe ich eine API entworfen, die es Konsumenten ermöglicht, Daten zu ihrem Lebensmittel abzufragen. Zum Schluss folgte die Umsetzung eines Proof of Concepts mit anschließender Diskussion. Durch diesen zeige ich, dass dieses System in der Realität funktioniert. Er beinhaltet eine Demo-Webseite zum Anzeigen der Daten für einen Konsumenten, implementiert in React. Die Webseite greift dafür auf ein Flask-Backend zurück, in welchem die API realisiert wurde. BigchainDB läuft in einem Docker-Container.

1.2 Verwandte Arbeiten

In dem Bereich der Protokollierung von Lieferketten gibt es viele Arbeiten:

Thume u.a.[67] haben ein Spezifikations-Framework für blockchainbasierte Nachverfolgungssysteme entwickelt. Dieses Framework beinhaltet ein Lieferkettenprozessmodell, Nutzungsvoraussetzungen und eine Zusammenfassung von technischen Voraussetzungen.

Liu u.a.[54] haben ein Nahrungsmittel-Rückverfolgungs-Framework basierend auf der permissioned Blockchain Hyperledger Fabric 2.0 vorgeschlagen. In diesem werden die unterschiedlichen Funktionen in einer Lieferkette verschiedenen Organisationen zugeteilt (Material, Produzent, Verteiler, Verkäufer, Kunde), die eigene Smart Contracts und Knoten aufweisen.

Tian u.a.[69] haben ein Lebensmittellieferkettennachverfolgungssystem basierend auf HACCP, BigchainDB, RFID und dem IoT entwickelt. Diese Arbeit fokussiert sich auf das Risikomanagement.

Kamilaris u.a.[49] haben in ihrer Arbeit einen Überblick über aktuelle (2019) Projekte geschaffen, die auf die Blockchain-Technologie für Lebensmittellieferketten setzen. Dabei haben sie 49 Initiativen untersucht, diskutieren die Probleme und Potenziale dieser Initiativen und kommen zu dem Ergebnis, dass die Blockchain-Technologie vielversprechende Möglichkeiten für dieses Gebiet bietet.

Dey u.a.[32] haben ein blockchainbasiertes Framework entwickelt, welches die Informationen für den Konsumenten mittels der Cloud und eines QR-Codes zur Verfügung stellt. Die Daten werden im QR-Code selbst und zusätzlich in der Blockchain gespeichert. Durch eine ID können die Daten im QR-Code immer mit der Blockchain verglichen werden.

1.3 Aufbau dieser Arbeit

Diese Bachelorarbeit beginnt in Kapitel 2 mit den technischen Grundlagen. Zunächst werden einige wichtige Begriffe in den Bereichen Datenbanken, Ledger und API erklärt. Darauf folgt Kapitel 3 mit der Problemstellung. Dort wird eingeführt, wie mögliche Lieferketten strukturiert sind und wie die Anforderungen an das System aussehen. Dabei wird zusätzlich betrachtet, welche Datenmengen anfallen und was die Folgen der Unveränderlichkeit sind. Im nächsten 4. Kapitel wird die beste Datenstruktur für das System gesucht. Dies beginnt mit der Definition von Vergleichskriterien, nach denen mögliche Datenstrukturen bewertet werden können. Darauf folgt die Erklärung und Bewertung der verschiedenen Datenstrukturen. Betrachtet werden klassische Datenbanken, Amazon Quantum Ledger Database, öffentliche Blockchains und BigchainDB. Das Kapitel endet mit einer Analyse der Bewertungen und der Auswahl der geeignetsten Datenstruktur. Danach folgt Kapitel 5, in dem eine API für das System entworfen wird. In Kapitel 6 wird der Proof of Concept programmiert. Dies wird begleitet von einer Diskussion der Schlüsse, die sich aus dem Proof of Concept ziehen lassen. Die Bachelorarbeit endet mit einer Zusammenfassung der zuvor bearbeiteten Inhalte und rekapituliert hierbei die relevantesten Aspekte.

2

Technische Grundlagen

Diese Arbeit befasst sich insbesondere mit dem Gebiet der Datenspeicherung und Abrufung dieser Daten. Dafür gibt es diverse Möglichkeiten. Das Kapitel Grundlagen befasst sich damit, die Begrifflichkeiten zu klären. Zunächst werden einige Begriffe geklärt, die allgemein relevant für das Gebiet der Datenspeicherung sind. Auf das Allgemeine folgt das Spezifische und die klassischen Datenbanken und die Ledger-Datenbanken werden erklärt. Zum Schluss folgt eine Erklärung von APIs.

2.1 Allgemeine Grundlagen der Datenspeicherung

Im Folgenden wird von *klassischen Datenbanken* und von *Ledger-Datenbanken* gesprochen. Der Begriff *Ledger* kann auf Deutsch mit Kassenbuch übersetzt werden. Ledger-Datenbanken stellen die Integrität der Daten in den Vordergrund. Eine Historie der Datenänderungen wird gespeichert und kryptographisch verifiziert. In der Literatur liegt der Fokus auf einer Unterform, den *Distributed Ledgers* (also verteilten Kassenbüchern) wie zum Beispiel *Blockchain*[24, 60]. Es existieren auch nicht-verteilte Ledger, wie *Amazon QLDB* (Amazon Quantum Ledger Database)[4]. Der Kernaspekt der Distributed Ledger ist, dass ein solches System aus einem Netz von Knoten besteht, die nicht unter der Kontrolle einer Entität stehen (müssen). Mit klassischen Datenbanken werden dementsprechend solche Datenbanken gemeint, die kein Ledger einsetzen und nicht aus dem System selbst heraus vor unvorhergesehenen Datenänderungen geschützt sind. Auch eine klassische Datenbank kann auf mehreren Servern liegen. Hier besitzt nur eine Entität die Kontrolle über die Server. Eine Ledger-Datenbank kann intern als Teil des Systems auch eine klassische Datenbank einsetzen.

SQL und NoSQL

Diese klassischen Systeme lassen sich in *SQL*[27, 23] und *NoSQL*[29] Datenbanken unterteilen. SQL steht für Structured Query Language und ist eine Anfragesprache für Relationale Datenbanken. Dementsprechend sind SQL-Datenbanken solche, die diese Anfragesprache unterstützen. Demgegenüber stehen NoSQL-Datenbanken. Der Begriff wird verschieden

genutzt. Die ursprüngliche Bedeutung war *kein SQL*. Dies bedeutete, dass die Datenbank die Anfragesprache SQL nicht unterstützt und nicht relational ist. Seit einigen Jahren wird die Abkürzung für Not only SQL verwendet[42]. Im Gegensatz zur früheren Nutzung des Begriffs soll klargemacht werden, dass eine NoSQL-Datenbank nicht unbedingt auf SQL verzichten muss. Sie ist dennoch nicht relational. NoSQL-Datenbanken können die Daten auf viele verschiedene Arten speichern[29], zum Beispiel als Key-Value Speicher oder als Graphdatenbank[9].

Vertikale und horizontale Skalierbarkeit

Ergänzend zu den zuvor genannten Punkten gibt es die Unterscheidung zwischen *vertikaler* und *horizontaler Skalierbarkeit*. Vertikal bedeutet hier, dass die Hardware selbst durch bessere ausgetauscht werden kann, also ein besserer Prozessor, ein größerer Arbeitsspeicher, eine schnellere Festplatte und ähnliche technische Verbesserungen. Dies ist in praktisch allen Situationen möglich, die Skalierung ist gleichzeitig begrenzt durch den technischen Stand der Zeit[45]. Die einzige Ausnahme hierzu stellt eine Datenbank dar, welche auf Treiber angewiesen ist, die für neue Hardware nicht existiert. Horizontal meint, dass ein System um mehr Instanzen erweitert werden kann, welche in der Lage sind zusammenzuarbeiten. Diese können im gleichen Gebäude oder auf der ganzen Welt verteilt sein. Eng verbunden mit der horizontalen Skalierbarkeit ist das sogenannte *Sharding*. Dabei wird eine Datenbank in kleinere Teile partitioniert, die dann auf verschiedenen Servern verwaltet werden. Ein horizontal skalierbares System kann sehr viel weiter skaliert werden, hat dennoch seine Grenzen[48].

Es hängt von der Datenstruktur und den Anforderungen an das System ab, wie einfach die Datenbank horizontal skalierbar ist[25]. Sobald die Datenbank verteilt liegt, muss die Synchronisierung der Daten beachtet werden. So wäre es für ein Finanzsystem fatal, wenn auf verschiedenen Servern der Bank verschiedene Kontostände vorliegen würden. Bei einem Dienst wie Wikipedia wäre es wiederum irrelevant, ob eine Änderung an einem Beitrag sofort überall verfügbar ist oder ob verschiedene Nutzer in den Minuten nach der Änderung verschiedene Versionen angezeigt bekommen. Die Lösung dieser Probleme erzeugt einen administrativen Overhead. Bei der Betrachtung dieser Probleme kommt man zu dem Schluss, dass READ und WRITE Operationen bei der Synchronisierung in unterschiedlichen Systemen eine andere Priorität einnehmen.

ACID

ACID steht für die vier Eigenschaften *Atomicity*, *Consistency*, *Isolation* und *Durability*[44]. Diese garantieren bestimmte Sicherheiten in der Datenbank. Im Folgenden wird der Begriff *Transaktion* verwendet. In diesem Kontext meint eine Transaktion eine Kette von Operationen auf einer Datenbank, welche eine logische Einheit bilden. So kann zum Beispiel bei einer Bank eine Überweisung eine Transaktion sein. Diese setzt sich zusammen aus dem Abrufen des aktuellen Kontostands der beiden Konten, dem Abzug des Überweisungsbetrages vom Kontostand des Senders und der Erhöhung des Kontostands des Empfängers um diesen Wert. Die ACID-Eigenschaften sind wie folgt definiert:

Atomarität Die Atomarität einer Transaktion wird so definiert, dass entweder eine Transaktion zur Gänze durchgeführt oder alle Zwischenschritte vollständig rückgängig gemacht werden[44].

Konsistenzerhaltung Ergänzend hierzu bezeichnet die Konsistenzerhaltung eine Eigenschaft, bei der die Datenbank vor der Transaktion einen legalen Zustand haben sollte und dieser nach jeder Transaktion erhalten bleiben muss[44].

Isolation Sofern eine Datenbank nebeneinander laufende Transaktionen, ohne eine gegenseitige Beeinflussung, verarbeiten kann, wird dieses als Isolation bezeichnet[44].

Dauerhaftigkeit Die Dauerhaftigkeit ist die letzte zu nennende Eigenschaft. Die Kernessenz hierbei ist, dass vollständig durchgeführte Transaktionen unter keinen antizipierbaren Umständen unrettbar verloren gehen können[44].

2.2 Grundlagen der klassischen Datenbanken

Die klassischen Datenbanken lassen sich in die folgenden Kategorien einteilen:

Relationale Datenbank Die *Relationale Datenbank*[28] ist eine SQL-Datenbank und das bekannteste Datenbankmodell. Aufgrund der weiten Verbreitung haben sich viele verschiedene Varianten dieses Modells entwickelt, welche sich dennoch zentrale Eigenschaften teilen. Sie nutzen Operatoren der relationalen Algebra[27], um mit Tabellen zu interagieren, deren Zeilen einzigartige Schlüssel aufweisen.

Schlüssel-Werte-Datenbank Eine *Schlüssel-Werte-Datenbank* ist eine einfach aufgebaute NoSQL-Datenbank. Sie besteht aus einer Tabelle von eindeutigen Schlüsseln mit dazugehörigen Werten. Diese Werte können vollkommen beliebig sein.

Dokumentenorientierte Datenbank Bei der *Dokumentenorientierten Datenbank*[26] handelt es sich um eine Unterform der Schlüssel-Werte-Datenbank. Diese speichert die Daten in sogenannten Dokumenten, welche weiter in Kollektionen geordnet werden können. Die Dokumente entsprechen den Werten der Schlüssel-Werte-Datenbank, und sind dementsprechend über einen eindeutigen Schlüssel identifizierbar. Die Daten in den Dokumenten werden meist in einem Format gespeichert, wie XML und JSON, wobei das Format von Dokument zu Dokument verschieden sein kann. Sie müssen jedoch keinem strikten Schema folgen. Ein bekannter Vertreter ist MongoDB[59].

Wide-Column Datenbank Die *Wide-Column Datenbank*[29] ist ebenso eine Unterform der Schlüssel-Werte-Datenbank. Sie ähnelt der Relationalen Datenbank, da sie auch auf eine Tabellenstruktur setzt, einen Key mit mehreren Spalten. Im Unterschied jedoch zur Relationalen Datenbank, bei der jede Zeile einer Tabelle die gleichen Spalten besitzt, kann in einer Wide-Column Datenbank jede Zeile unterschiedliche Spalten aufweisen.

Graphdatenbank Der Letzte der relevanten Datenbanktypen ist eine *Graphdatenbank*[29, 9]. Hier werden die Daten direkt als Knoten eines Graphen gespeichert. Beziehungen zwischen den Daten werden über Kanten zwischen den Datenknoten modelliert.

Hierbei entstehen große Vorteile gegenüber einer Relationalen Datenbank, wenn es um die Beziehungen zwischen den Daten geht. Denn bei den Graphdatenbanken gibt es keine teuren Join-Operationen. Die Beziehungen werden einem durch die Struktur zur Verfügung gestellt.

2.3 Grundlagen der Ledger-Datenbanken

Es gibt *Zentralisierte Ledger* wie *LedgerDB*[74] oder *Amazon QLDB*[4] und Distributed (verteilte) Ledger. Ledgertechnologien setzen intern (häufig) auf eine Blockchain. Eine *Blockchain*[55] ist eine Kette von Datenblöcken. Jeder Block, mit Ausnahme des sogenannten Genesis-Blocks (dem allerersten Block), beinhaltet einen Hash des vorherigen Blocks. Auf diese Art sind die Blöcke miteinander verkettet. Wird einer der vorherigen Blöcke verändert, stimmt der Hash im nächsten Block nicht mehr. Auf diese Weise ist kryptographisch sichergestellt, dass Veränderungen nicht unentdeckt bleiben. Wenn im allgemeinen Sprachgebrauch von Blockchain die Rede ist, ist meistens ein Distributed Ledger gemeint und im Folgenden wird der Begriff auch entsprechend genutzt[51]. Die Technologie Blockchain selbst gibt dies nicht vor, ist aber gut geeignet für den Einsatz als Distributed Ledger. Das prominenteste Beispiel hierfür ist Bitcoin[60].

Blockchains können zur Verwaltung von *Token* und *Assets* genutzt werden. Im Kontext der Blockchain sind Token Marker, die einen Wert repräsentieren. Diese können meist gehandelt werden. Ein Asset bezeichnet hier die digitale Abbildung eines Sachverhaltes in einem einfachen Datenformat. Dadurch kann sowohl ein Haus, ein Nahrungsmittel oder ein abstraktes Konzept wie das Lizenzrecht über ein Werk abgebildet werden. Auch Assets können gehandelt werden.

Arten von Blockchain

Distributed Ledger in der Form einer Blockchain können weiter kategorisiert werden als *öffentliche Blockchain*[51], *hybride Blockchain*[1] und *private* beziehungsweise *permissioned Blockchain*[62]. Nicht immer ist eine klare Zuordnung in einer dieser Kategorien möglich, denn der Übergang ist teilweise fließend. Bei einer öffentlichen Blockchain hat jeder Zugriff auf die Daten und kann an der Validierung von Transaktionen teilnehmen. Das gesamte Netzwerk ist öffentlich. Bei einer hybriden Blockchain wird dies etwas eingeschränkt. Es kann weiterhin jeder auf die Daten zugreifen, Transaktionen zu validieren ist jedoch einer kleineren Gruppe an Nutzern vorbehalten. In einer privaten Blockchain ist der Zugriff auf die Daten limitiert auf die bekannten Nutzer. Hier kann es verschiedene Rollen geben, nach denen der Zugriff begrenzt werden kann.

Dann gibt es noch die sogenannten *Side-Chains*. Eine Side-Chain ist eine Blockchain, die sich von einer übergeordneten Blockchain abgespalten hat. Der Austausch von Assets und Token zwischen der Side-Chain und der übergeordneten Blockchain ist weiter möglich.

Probleme mit Konsens

In einer verteilten Blockchain gibt es keine zentrale Autorität, die entscheiden kann, welche Blöcke hinzugefügt werden. Daher muss ein *Konsensmechanismus* existieren, welcher dafür sorgt, dass die Teilnehmer der Blockchain sich auf einen Block einigen. Bei einem verteilten System kann ein Knoten den anderen Knoten nicht vertrauen. Knoten repräsentieren hierbei die Teilnehmer des Systems. Dies gilt insbesondere für ein System, in dem beliebige Nutzer sich zuschalten können. Die Knoten können ausfallen, fehlerhafte Daten senden oder bösartige Ziele verfolgen. Ein solcher Fehler wird *byzantinischer Fehler* genannt, welcher auf das *Byzantine Generals Problem*[52] zurückzuführen ist.

Im Byzantine Generals Problem haben byzantinische Generäle eine Stadt umzingelt. Diese müssen einstimmig beschließen, wann sie angreifen, um erfolgreich zu sein. Zur Kommunikation stehen ihnen jedoch nur Boten zur Verfügung und einige der Generäle sind Verräter. Diese Verräter versuchen, gezielt falsche Botschaften zu schicken. Hier ist ein byzantinischer Fehler, also das Senden von falschen Botschaften an die anderen Generäle oder das Senden gar keiner Botschaft.

Ein System, welches eine bestimmte Menge dieser byzantinischen Fehler aushalten kann, nennt man *Byzantine Fault Tolerant* (BFT). Für eine Blockchain bedeutet Aushalten hier, dass das System weiterhin korrekt arbeitet. Es dürfen keine falschen Transaktionen in die Blockchain aufgenommen werden. Das Aufnehmen neuer Blöcke darf nicht blockiert werden. Zudem darf kein (dauerhafter) Fork in der Blockchain entstehen. Aushalten bedeutet jedoch nicht, dass das System weiter in der gleichen Geschwindigkeit arbeiten können muss. Bei der Byzantine Fault Tolerance wird meistens mit angegeben, wie viele Knoten einen byzantinischen Fehler aufweisen dürfen. Manche Algorithmen setzen bestimmte Eigenschaften der funktionierenden Knoten voraus. Eine solche Eigenschaft könnte die Fähigkeit sein, die Zeit zu messen.

Eine mögliche Schwachstelle von verteilten Systemen sind sogenannte *Sybill-Attacken*[34]. Bei diesem Angriff erzeugt der Angreifer viele falsche Identitäten, um die echten Knoten im System von falschen Tatsachen zu überzeugen. In einem verteilten System, in welchem die Knoten eine einfache Abstimmung zur Feststellung der Wahrheit durchführen, ist ein solcher Angriff sehr erfolgreich.

Konsensalgorithmen

Für die Konsensbildung gibt es eine Vielzahl an Algorithmen, die Unterformen aufweisen können. In der folgenden Liste werden zunächst die beiden bekanntesten Strategien erklärt. Die letzten beiden Strategien werden in späteren Kapiteln noch von Relevanz sein.

Proof of Work *Proof of Work*[47] ist die bekannteste Strategie. Hierbei werden alle Knoten, die einen neuen Block an die Blockchain hängen wollen, vor ein komplexes, mathematisches Problem gestellt. In vielen Implementierungen soll ein Hash gefunden werden, der mit einer bestimmten Anzahl an Nullen anfängt[60]. Da meistens Ziel ist, eine feste Blockzeit zu erreichen, wächst die Schwierigkeit dieses Rätsels mit der

Rechenleistung aller Teilnehmer. Daher verbraucht eine Blockchain mit Proof of Work vergleichsweise große Mengen an Energie, worin ein großer Kritikpunkt liegt[30]. Es kann nun passieren, dass zwei Knoten gleichzeitig die Lösung für das Rätsel finden. In diesem Fall kommt es kurzfristig zu einem Fork in der Blockchain. Andere Knoten akzeptieren die Kette als Wahrheit, die am längsten ist, da dort die meiste Arbeit investiert wurde. So lösen sich die Forks wieder auf. In hinreichend großen Netzwerken wie Bitcoin bedeutet dies, dass ein Sybill-Angriff auf das Netzwerk kaum tragbar ist. Ein Angreifer würde etwa die Hälfte der Rechenleistung des Netzwerks oder mehr benötigen. In kleinen Systemen stellt dies jedoch eine Schwachstelle dar, da es hier verhältnismäßig leicht ist, eine solche Rechenleistung zu erreichen. Dies bedeutet, dass eine Transaktion niemals als absolut bestätigt angesehen werden kann. Und zumindest die nächsten paar Blöcke werden häufig abgewartet, bis eine Transaktion als hinreichend bestätigt bezeichnet werden kann.

Proof of Stake *Proof of Stake*[63] ist ein Konsensmechanismus, der nur bei Blockchains funktioniert, die auf einer Art Token aufbauen. Hierbei müssen Knoten, die neue Blöcke hinzufügen wollen, einen Teil ihrer Token hinterlegen. Das System wählt einen zufälligen Knoten, gewichtet nach der Menge der hinterlegten Token, aus. Für den Fall, dass der Block nicht den Regeln des Netzes entspricht, können die hinterlegten Token eingefroren werden, wodurch Angreifer mehr verlieren, als gewinnen. Der Angreifer benötigt für einen erfolgreichen Angriff die Hälfte der hinterlegten Token.

Practical Byzantine Fault Tolerance *Practical Byzantine Fault Tolerance* (pBFT)[22] setzt auf eine replizierte State Maschine in einem verteilten System. Die Replika müssen zwei Anforderungen erfüllen: Determinismus und Start im selben State.

Vereinfacht funktioniert der Algorithmus wie folgt: Ein Knoten ist (wechselnd) der primäre Knoten, an den die Klienten Anfragen senden. Die anderen Knoten sind die Backups. Der primäre Knoten sendet nun die Anfrage an alle anderen Knoten. Alle Knoten senden jetzt ihre Antwort an den Klienten. Wenn mehr als ein Drittel der Knoten die gleiche Antwort gesendet haben, akzeptiert der Klient diese Antwort.

Somit ist dieser Konsens-Algorithmus Byzantine Fault Tolerant bis zu einem Drittel der Knoten. Gleichzeitig ist dieser anfällig für einen Sybill-Angriff, wenn er in einem öffentlichen Netzwerk eingesetzt wird. Des Weiteren kommunizieren alle Knoten mit allen anderen Knoten, weshalb der Algorithmus nicht gut skaliert.

Tendermint-Konsens Der *Tendermint-Konsens-Algorithmus* ist inspiriert vom zweiten DLS Algorithmus[35] und vom pBTF SMR Algorithmus[22].[20, 21] Die Kommunikation zwischen den Knoten setzt auf ein Gossip-Protokoll. Der Algorithmus ist in Runden aufgeteilt. Zunächst wird ein neuer Block vorgeschlagen. Über diesen Block wird abgestimmt und zwei Drittel der Knoten müssen zustimmen. Danach folgt eine zweite Wahlphase, in der wieder zwei Drittel der Knoten zustimmen müssen. Geschieht dies, ist der Block akzeptiert. Es werden zwei Wahlgänge benötigt. In einem asynchronen System könnte ein Angreifer durch Koordination mit anderen Knoten das System in einen inkonsistenten Zustand bringen. Der Algorithmus ist Byzantine Fault Tolerant bis zu einem Drittel der Knoten.

2.4 Grundlagen der APIs

Eine *API* ist eine Schnittstelle für ein System, über welche Daten ausgetauscht werden können. Die Abkürzung steht für *Application Programming Interface*. APIs sind ein wichtiger Bestandteil eines Systems. Gute APIs sorgen für einen effizienten Datenaustausch und sind erweiterbar. Sie sind Teil einer guten Kapselung der Systembestandteile. Eine schlechte API ist nicht erweiterbar, offenbart zu viel vom Innenleben des Systems und erschwert den Datenaustausch.

API-Architekturen

Es gibt verschiedene Arten von APIs, die in unterschiedliche Architekturen eingeteilt werden können. Zu den meist genutzten gehören *RPC*, *SOAP* und *REST APIs*. Eine API ist nicht immer einer bestimmten Architektur zugehörig.

RPC *RPC*[72] steht für *Remote Procedure Call*. Hierbei werden Prozeduren / Funktionen aus der Entfernung heraus aufgerufen. Es gibt Implementierungen, die als RFC eingereicht wurden[46, 68, 72], prinzipiell ist RPC jedoch eine allgemeine Klassifizierung.

SOAP *SOAP* ist ein Standard des W3C[71], daher ein definiertes Protokoll und eine Form von RPC. Das Protokoll kann auf einem beliebigen Transportprotokoll aufsetzen und nutzt XML als Datenformat.

REST *REST*[40] steht für *Representational State Transfer*. REST ist kein Protokoll, sondern ein Set an Einschränkungen, die eine API einhalten muss, um RESTful API genannt zu werden. Es ist eine Abstraktion ohne Implementierungsdetails oder Protokollsyntax, um möglichst allgemein Anwendung zu finden. Die Einhaltung dieser Einschränkungen sind von Vorteil für die Skalierbarkeit, Performanz und Ähnliche. Als erste Einschränkung gilt die Client-Server-Trennung. Die Kommunikation zwischen Client und Server muss zustandslos sein. Daher ist es notwendig, dass jede Anfrage des Clients alle notwendigen Informationen beinhaltet. Außerdem sollen die Daten als cachebar oder nicht cachebar markiert werden, sodass der Client einen Cache erstellen kann, um zukünftige Anfragen zu beschleunigen. Eine vierte Einschränkung ist eine einheitliche Schnittstelle. Das bedeutet, dass alle Komponenten auf die gleiche Art miteinander kommunizieren, selbst wenn es an manchen Stellen effizienter wäre, eine spezielle Schnittstelle zu entwickeln. Dies fördert die eindeutige Trennung der Komponenten und simplifiziert die komplette Architektur. Zudem ist ein geschichtetes System gefordert. Eine REST API abstrahiert alle Daten zu Ressourcen, auf die über einen Identifikator zugegriffen werden kann. Dieses Konzept wurde mit Blick auf HTTP entwickelt. Daher ist es gut für Webseiten nutzbar. Eine RESTful API muss aber nicht HTTP nutzen.

API-Klassifizierungen

APIs können weiter als privat, öffentlich, partner und composite API klassifiziert werden. Eine private API ist nur zur internen Verwendung gedacht und häufig nur in einem internen Netz verfügbar. Daher sind hier weniger Anforderungen an die Authentifizierung

und Sicherheit erforderlich. Eine Partner-API steht nur bestimmten anderen Entitäten zur Verfügung, die häufig hierfür zahlen. Öffentlich ist eine API, wenn sie von jedem im Netz genutzt werden kann. Schließlich ist eine composite API eine, die mehrere Anfragen in einem API-Aufruf bündelt.

API-Datenformate

Für das Datenformat gibt es viele Möglichkeiten. Zu den populärsten gehören CSV, XML und JSON.

CSV steht für *Comma Seperated Values* und wird viel in der Businesswelt genutzt. Standard-Software wie Excel exportiert die Daten für gewöhnlich in ein solches Format. Ein Beispiel ist in Programmtext 2.1 zu sehen.

Programmtext 2.1: Daten im CSV-Format

```
Lübeck,23556,456,
Baden,12345,789,
Hamburg,10101,010,
```

XML steht für *Extensible Markup Language* und wurde vor allem in der Vergangenheit viel genutzt. Mittlerweile ist die Nutzung zurückgegangen, da XML viel Overhead im Vergleich zur Nutzlast aufweist. Außerdem werden keine Datentypen unterstützt, es gibt nur Strings. Ein Beispiel ist in Programmtext 2.2 zu sehen.

Programmtext 2.2: Daten im XML-Format

```
<nachricht>
    <an>Max Mustermann</an>
    <von>Max Musterfrau</von>
    <gebühr>€10</gebühr>
    <text>Lebe lang und in Frieden.</text>
</nachricht>
```

JSON steht für *JavaScript Object Notation* und ist der quasi-Standard im Web heutzutage für die Übertragung von Daten zwischen der Webseite und dem Server. Er findet auch in nicht-Web Bereichen Einsatz und kann von vielen Programmiersprachen, nicht nur JavaScript, verarbeitet werden. Ein Beispiel ist in Programmtext 2.3 zu sehen.

Programmtext 2.3: Daten im JSON-Format

```
{
    "id": "A2C4E6G8I",
    "daten": {
        "besitzer": [
            {"name": "Peter", "alter": 20},
            {"name": "Klaus", "alter": 60}
        ],
        "beschreibung": "Kiste mit Wertgegenständen."
    }
}
```

3

Problemstellung

Jährlich werden hunderte Millionen Tonnen Fisch gefangen und durch die ganze Welt geschifft[39]. Jedes Jahr kommen allein in Deutschland fast 12 Millionen Tonnen Lebensmittel in den Abfall[53]. Teile dieser Menge landen im Abfall, weil der Verbraucher sich der Frische der Lebensmittel nicht sicher ist. Besonders bei Fisch sind die Verbraucher aufgrund der schnellen Verderblichkeit sehr vorsichtig. Es wurden viele Projekte ins Leben gerufen, um dieses Problem anzugehen. Daher wird ein Framework benötigt, auf dem derartige Projekte aufbauen können. Ein solches Framework soll mehrere Aspekte vereinen. Um das zu verstehen, wird zuerst der Aufbau von Lieferketten betrachtet. Darauf aufbauend betrachten wir, welche Funktionen sinnvoll für das Framework sind. Danach wird die anfallende Datenmenge analysiert. Zum Schluss wird die Frage betrachtet, was für Folgen die Unveränderlichkeit für das System hat.

3.1 Aufbau von Lieferketten

Lieferketten in der Lebensmittelindustrie können, genauso wie in anderen Industrien, einfach oder komplex strukturiert sein. Der grundlegende Aufbau ist jedoch der Gleiche.

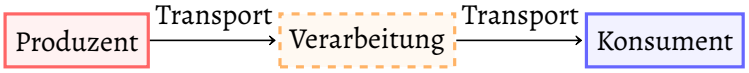

Abbildung 3.1: Grundlegender Aufbau einer Lieferkette. Die Verarbeitung ist optional.

Am Anfang stehen ein oder mehrere Produzenten eines unverarbeiteten Rohstoffes / Lebensmittels. Ein Produzent kann in diesem Fall alles sein, vom Apfelbauer über den Weizenfarmer bis zum Fischereibetrieb. Nach dem Produzenten folgt die Verarbeitung. Dieser Schritt ist optional und kann mehrere Einzelschritte beinhalten. Am Schluss steht der Verkauf an den Konsumenten. Dies ist in Abbildung 3.1 dargestellt.

Die Erzeugnisse können daraufhin unterschiedliche Wege gehen. Ein Produzent beliefert normalerweise mehr als einen Kunden. Betrachtet wird dies einmal am Beispiel eines

Apfelbauern, siehe auch Abbildung 3.2.

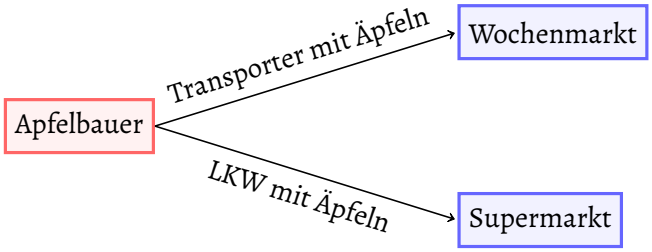

Abbildung 3.2: Hier werden zwei mögliche Lieferketten für einen Apfel dargestellt.

Die Äpfel können etwa direkt beim Bauern verpackt und von ihm zum nächsten Wochenmarkt transportiert werden. Auch ist es denkbar, dass diese an eine Supermarktkette weiterverkauft werden, welche die Äpfel quer durch das Land transportiert und dort verkauft. Während im ersten Fall die Lieferkette für den Kunden offensichtlich ist, ist dies im zweiten Fall nicht mehr so.

Der Weizen wird einen ganz anderen Weg bestreiten.

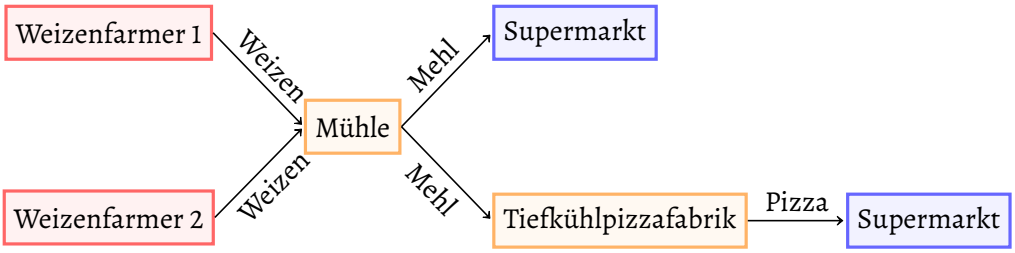

Abbildung 3.3: Ausschnitt aus einer möglichen Lieferkette für Pizza. Die Lieferketten für weitere Pizzazutaten funktionieren ähnlich.

Dieser kann zu einer Mühle transportiert werden. Hier ist es möglich, dass der Müller den Weizen mit dem Weizen anderer Farmer mischt. Dieses Mehl kann dann direkt im Laden verkauft oder zur industriellen Weiterverarbeitung genutzt werden. Spätestens wenn dieses Mehl zu einer Tiefkühlpizza weiterverarbeitet wird, ist es für den Konsumenten nicht mehr ersichtlich, woher die Rohstoffe seiner Ernährung stammen. Dieses Beispiel ist in Abbildung 3.3 dargestellt.

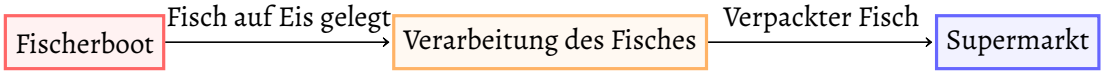

Abbildung 3.4: Hier wird eine mögliche Lieferkette eines Fischs dargestellt. Der Verarbeitungsschritt beinhaltet verschiedene Einzelschritte wie das Entfernen der Gräten und das Verpacken des Fisches. In der gesamten Lieferkette muss die Kühlung sichergestellt werden.

Betrachtet wird als letztes Beispiel Fisch, wie in Abbildung 3.4. Der Fisch wird zunächst im Meer gefangen. Dort wird dieser zur Frischhaltung auf Eis gelegt. Bei Fisch ist das besonders wichtig, da er rasch verderben und lebensmittelbedingte Erkrankungen auslö-

sen kann[10]. Als Nächstes muss er weiterverarbeitet werden. Die Innereien und Gräten müssen entfernt werden. Für diesen Schritt ist es möglich, dass ein im Atlantik gefangener Fisch aus Kostengründen nach China transportiert, dort verarbeitet, verpackt und wieder zurück nach Europa geschifft wird. Dies kann Wochen dauern und die ganze Zeit über muss die Kühlung gewahrt werden.

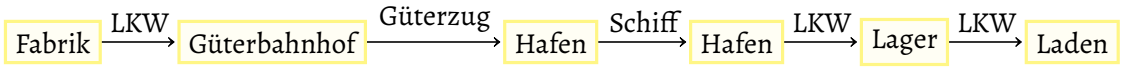

Abbildung 3.5: Hier ist ein möglicher Lieferweg zwischen einer Fabrik und einem Verkaufsort dargestellt. In diesem Beispiel wird ein Lebensmittel von der Fabrik, in der es verarbeitet wurde, mit einem Laster zu einem Güterbahnhof transportiert. Dort wird es auf einen Zug aufgeladen und zum nächsten Hafen transportiert. Im Hafen wird das Lebensmittel auf ein Schiff geladen und quer über die Weltmeere geschifft. Bei der Ankunft im Zielhafen wird es auf einen anderen Laster umgeladen, der das Lebensmittel zu einem Lager transportiert, wo es eine Weile verstaut wird. Nach wochenlanger Lagerung wird das Lebensmittel am Ende noch einmal per Laster zum Discounter transportiert.

Bisher wurde der Transport zwischen den verschiedenen Verarbeitungsorten nicht genauer betrachtet. Am Beispiel in Abbildung 3.5 sehen wir, dass selbst zwischen einer Fabrik und dem Discounter fünf verschiedene Unternehmen für den Transport verantwortlich sein können. Wenn man abschließend eine gesamte Lieferkette betrachtet, kann ein Produkt von dutzenden verschiedenen Entitäten transportiert worden sein.

Zusammenfassung der Probleme in Lieferketten

Zusammenfassend lassen sich mehrere Probleme im aktuellen System identifizieren. Die Lebensmittel gehen im Rahmen der Produktionskette durch viele verschiedene Hände. An erster Stelle steht der Produzent, bzw. bei verarbeiteten Lebensmitteln viele Produzenten für die verschiedenen Bestandteile. Darauf folgt meist die Verarbeitung in industriellen Anlagen. Diese kann aus vielen Zwischenproduzenten bestehen. Am Ende steht der Verkäufer. Und zwischen diesen Entitäten wird das Nahrungsmittel transportiert. Auf einer Strecke kann der Transport von verschiedenen Entitäten durchgeführt werden.

Außerdem müssen für einige Lebensmittel auf dem ganzen Transport besondere Bedingungen sichergestellt werden, wie die Kühlung beim Fisch.

Dabei kann schnell der Überblick verloren gehen. Fallen beim Endprodukt Qualitätsmängel auf, ist hinterher oft nicht feststellbar, an welcher Stelle ein Fehler unterlaufen ist. So kann beim Fisch zwar festgestellt werden, dass die Kühlkette unterbrochen gewesen sein muss, in wessen Verantwortlichkeit die Kühlkette jedoch versagt hat, bleibt jedoch unklar.

Das letzte Problem ist die Intransparenz dem Konsumenten gegenüber. Dieser hat nur die Produktinformationen, die im letzten Schritt der Verpackung aufgedruckt werden. Eine bessere Transparenz ist wünschenswert.

3.2 Notwendige Funktionen des Systems

Es sind einige Probleme aufgefallen, die gelöst werden sollen. Im Kern lassen sich diese Probleme auf eine mangelnde Datenerfassung zurückführen. Daher sollte jedes Lebensmittel einzeln in der Lieferkette protokolliert werden können. Für die Eindeutigkeit wird eine ID für jedes Lebensmittel benötigt. Um in der Industrie tatsächlich sinnvoll eingesetzt werden zu können, wird sich im Folgenden auf den letzten Teil der Lieferkette, zwischen der Verpackung und dem Konsumenten fokussiert. So kann dem Lebensmittel beim Verpacken eine ID zugewiesen werden.

Als Nächstes betrachten wir, welche Daten von dem System erfasst werden sollen. Zum einen sollen allgemeine Daten über das Produkt, die beim Schritt des Verpackens bekannt sind, aufgenommen werden. Dazu gehören insbesondere Name des Produktes, Herkunft und Produktionsdatum. Welche Daten relevant sind, kann sich über die Zeit ändern.

Im Transport fallen zudem Daten an. Dies ist auch heute schon der Fall, jedoch werden diese nicht einheitlich in einem System registriert und teilweise nicht einmal digital. Unter anderem gehören dazu Temperaturmessungen mit Zeitstempel, um die Kühlung zu protokollieren. Auch hier gibt es beliebig viele Möglichkeiten, welche Daten gespeichert werden sollen. Daher sollte das Datenformat arbiträre Strings speichern können. Außerdem können Fotos Teil der Daten sein, welche nicht von jedem Datenformat gespeichert werden können. Eine Möglichkeit besteht darin, diese extern zu speichern und im Datenformat einen URI und Hash zu hinterlegen.

Um Nahrungsmittelverschwendung zu vermeiden, können KI-Modelle eingesetzt werden, welche die Daten aus der Lieferkette nutzen, um den Konsumenten eine Frischevorhersage bereitzustellen. Dementsprechend benötigt das System eine Schnittstelle, um eine Auswertung der Daten zu ermöglichen.

Diese Daten sollen schlussendlich dem Konsumenten zur Verfügung gestellt werden, was über ein gebrauchstaugliches Frontend geschehen sollte.

Die letzte und wichtigste Anforderung ist die Unveränderlichkeit der Daten. Sobald die Daten im System registriert sind, darf sie niemand mehr ändern können, nicht einmal ein Administrator mit direktem Zugriff auf die Datenbank. Dies ist wichtig, um ein Vertrauen in das System herzustellen.

3.3 Anfallende Datenmenge

Um dem System einen Rahmen zu geben, braucht es eine Einschätzung der zu verarbeitenden Datenmenge. Dazu zählt, wie viele *Transaktionen pro Sekunde* (TPS) verarbeitet werden (können) und wie viel Speicherplatz die Datenbank einnimmt. Mit einer Transaktion ist hierbei ein Schreibzugriff gemeint, der ein neues Lebensmittel oder neue Informationen als Update zu einem schon bestehenden Lebensmittel hinzufügt. Im Folgenden wird die Größe einer Transaktion abgeschätzt. Da JSON wenig Overhead hat, wird die Größe einer kleinen und einer mittleren Transaktion im JSON-Format abgeschätzt.

Programmtext 3.6: Kleine Transaktion

```
{
    "id": "dQw4w9WgXcQ",
    "data": {
        "time": "2022-10-01 00:00:00",
        "temp": -10
    },
    "sha256":
       "3da4f5741d8f3b0184aeaafe08401f5e4ab0207a557c94cfde2e3ac64e521353"
}
```

Programmtext 3.7: Mittlere Transaktion

```
{
    "id": "dQw4w9WgXcQ",
    "data": {
        "time": "2022-10-01 00:00:00",
        "temp": -10,
        "pic": {"url": "www.example.com/example/file/
                          5f6b0b4e201f2a7e66927abb5cadeec81624dcc8efe6644b78aa182213f653a2",
                "hash":
                   "5f6b0b4e201f2a7e66927abb5cadeec81624dcc8efe6644b78aa182213f653a2"},
        "gps": "53.83469981433654, 10.696835271012759",
        "transport": "Shipping Company Elizabeth II. GMBH"
    },
    "sha256":
       "63ae99b9de6563931350d66ef63ec1e0741d3687785c14c316f1368a4f366a27"
}
```

Die Nachricht in Programmtext 3.6 ist 189 Bytes und die Nachricht in Programmtext 3.7 ist 518 Bytes groß. Als Codierung wurde UTF-8 genutzt. Es ist vorstellbar, dass eine Transaktion das Vierfache dieser Daten transportiert, somit sind 2 KB eine gute obere Abschätzung für eine Transaktion.

3.4 Die Folgen von Unveränderlichkeit

Es wird noch ein Blick auf die Unveränderlichkeit selbst geworfen. Die Unveränderlichkeit der Daten ist für den Anwendungsfall unerlässlich. Und doch wirft diese Unveränderlichkeit Fragen und Probleme auf.

Nichts, was der Mensch kennt, ist fehlerfrei. Beim Sammeln der Daten können Fehler auftreten. Wenn diese Fehler vor dem Speichern nicht korrigiert werden, bleiben sie im System ewig bestehen. Damit kann auf verschiedenen Wegen umgegangen werden. Der erste Ansatz ist das Ignorieren. Treten Fehler ausreichend selten auf und/oder sind die Fehler nicht gravierend genug, um tatsächlich für Probleme zu sorgen, ist dies ein valider Ansatz. Der zweite Ansatz ist, einen neuen Eintrag hinzuzufügen, der den falschen Eintrag

benennt und richtigstellt. Somit ist der Fehler weiter in der Datenbasis vorhanden, das System kann jedoch mit diesem umgehen.

Das zweite Problem sind Daten, die nicht fehlerhaft sind, aber dennoch nicht gespeichert werden dürfen. So hat jeder EU-Bürger das Recht auf Löschung von personenbezogenen Daten nach der DSGVO[50]. Wenn aus einem Versehen heraus solche Daten in das System gelangen, ist dies ein Problem mit rechtlicher Bedeutung. Diese Arbeit wird auf das datenschutzrechtliche Problem nicht weiter eingehen. Es sollte für jede Art von Daten, die in einem solchen System gespeichert werden, vorher geprüft werden, ob die Daten eventuell gelöscht werden können müssen. Ist dies der Fall, so sollten sie nicht unveränderlich gespeichert werden.

4

Die Wahl der richtigen Datenstruktur

Nun ist bekannt, was die Anforderungen an ein System zur Lösung des Problems sind. Es gibt verschiedene Systeme, die auf diese Anforderungen überprüft werden sollen. Um diese sinnvoll vergleichen zu können, wird ein Vergleichssystem benötigt, dessen Kriterien im Anschluss erläutert werden. Anschließend werden verschiedene Datenstrukturen vorgestellt. Diese werden zunächst dahin gehend überprüft, ob sie die Anforderungen für die gegebene Situation erfüllen. Die Strukturen, die das tun, werden danach mithilfe des Vergleichssystems bewertet. Zum Schluss werden die Ergebnisse analysiert und es wird die am besten geeignete Datenstruktur ausgewählt.

4.1 Definition von Vergleichskriterien

Im Folgenden wird ein Vergleichssystem entworfen. Dafür werden mehrere Kriterien definiert, nach denen jeweils Punkte auf einer Skala von 0 bis 9 vergeben werden. Die Kriterien werden in zwei Gruppen aufgeteilt. Die *primären Kriterien* beinhalten diejenigen, die im Produktiveinsatz direkt relevant sind und bestimmen, wie gut das Framework tatsächlich funktioniert. Die *sekundären Kriterien* betreffen hingegen die nicht funktionalen Aspekte, siehe Tabelle 4.1.

Primäres Kriterium	Sekundäres Kriterium
Performanz	*Dokumentation und Ease of Use*
Skalierung	*Kosten*
Synchronisierung	*Fehleranfälligkeit*

Tabelle 4.1: Einordnung der Kriterien in Primäres Kriterium und Sekundäres Kriterium.

In der Auswertung wird neben der Gesamtpunktzahl auch der Median betrachtet. Zudem wird die niedrigste Punktzahl, die in einer Kategorie erreicht wurde, erfasst und ebenfalls in die Bewertung einbezogen. Dies hat den Hintergrund, dass ein System, welchen in allen anderen Aspekten perfekt geeignet ist, trotzdem nicht eingesetzt werden kann, wenn die Kosten zu hoch sind.

Folgende Kriterien sind Teil des Vergleichssystems:

Performanz Mit der Performanz ist gemeint, wie schnell das System auf Anfragen reagiert und diese verarbeitet. Ein Nutzer des Systems soll nicht erst Sekunden oder sogar Minuten warten müssen, bis angefragte Daten geladen wurden. Die Einspeisung und Verarbeitung muss mindestens so schnell sein, dass bei realistischen Datenmengen kein Backlog entsteht, und das System alle Daten verarbeiten kann.

Skalierung Die Anforderung der Unveränderlichkeit von Daten führt dazu, dass entsprechende Systeme keine Löschung der Daten mehr zulassen. Daher fallen auf lange Sicht sehr große Datenmengen an. Dies darf das System nicht signifikant verlangsamen.

Eine weitere Art der Skalierung bezieht sich auf die Anzahl der Teilnehmer eines Systems. Auch hier soll vermieden werden, dass das System an Geschwindigkeit verliert. Bei einigen Blockchain-Implementierungen ist leider genau dies der Fall, je mehr Knoten beteiligt sind.

Die letzte Form der Skalierung bezieht sich auf die Anzahl der Anfragen an das System pro Sekunde. Anfragen an das System umfassen sowohl Schreib- als auch Leseanfragen. Zum Vergleich: Das VISA-Zahlungsnetzwerk erreicht 65.000 Transaktionen pro Sekunde[70].

Synchronisierung Viele der möglichen Modelle setzen auf eine verteilte Datenspeicherung. Daher ist es wichtig, dass die Daten am Ende alle vorgesehenen Knoten erreichen. Zudem muss dafür gesorgt werden, dass für die Dauer der Synchronisierung Datenkonflikte ausgeschlossen werden. Die Reihenfolge der Datensätze ist hierbei in den meisten Fällen irrelevant, da die Daten mit einem Timestamp versehen sind und nicht aufeinander aufbauen. Je nach Design der Datenstruktur kann es jedoch sein, dass zumindest ein Anlegen des Datensatzes für ein Nahrungsmittel geschehen muss, bevor neue Daten hinzugefügt werden können.

Dokumentation und Ease of Use Die Dokumentation und der sogenannte Ease of Use hängen zwar nicht direkt voneinander ab, sind aber trotzdem eng miteinander verbunden. Um gut mit einem System arbeiten zu können, ist es wichtig, dass es eine gute Dokumentation gibt. Wenn das System Open Source ist, kann man die Implementierung für ein tieferes Verständnis des Systemverhaltens eigenhändig überprüfen. Dies betrifft die Programmierung, die API-Schnittstellen und das Aufsetzen des Systems.

Kosten Kosten können auf viele Weisen anfallen und sind ein wichtiger Faktor. Zum einen gibt es die Anschaffungskosten für nötige Hardware, wie Server. Diese Kosten fallen zwar verhältnismäßig selten an, sind jedoch nicht einmalig. Gründe für die Notwendigkeit neuer Hardware können etwa Komplettausfälle der alten Hardware sein oder dass Software- und Sicherheitsupdates für die alte Hardware eingestellt wurden. Ein weiterer Kostenfaktor sind die laufenden Kosten. Darunter fallen die Strom-, Internet-, Wartungs- und Personalkosten. Sollte das System vollständig in einer Cloud laufen, so fallen die Kosten für die Hardware, den Strom und das Internet weg. Dafür kommen neue Kosten hinzu, die der Cloudprovider in Rechnung stellt. Der Nahrungsmittelpreis sollte nicht durch den Einsatz dieser Technologie steigen.

Um die Kosten gut vergleichen zu können, werden die folgenden zwei Szenarien betrachtet:

1. Im ersten Szenario sollen die Kosten für ein minimal lauffähiges System berechnet werden. Minimal lauffähig bedeutet, dass eine Transaktion pro Sekunde verarbeitet werden kann. Außerdem müssen bei einem verteilten System so viele Knoten eingesetzt werden, wie notwendig sind, um den Ausfall eines Knotens verarbeiten zu können.
2. Im zweiten Szenario sollen die Kosten für ein System berechnet werden, an dem 10 Firmen teilnehmen können, und mit dem 100 Transaktionen pro Sekunde verarbeitet und 100 Lesezugriffe bedient werden können.

Um eine vergleichbare Rechnung zu ermöglichen, werden für die Berechnungen Cloud-Dienste als Kostenbasis herangezogen und weitere Kosten, etwa Personalkosten, nicht mit berücksichtigt. Die Kosten für eine mögliche Anbindung an eine KI, die Speicherung von Bilddaten und das Hosten eines Webservers werden ebenfalls nicht einbezogen, da diese bei jeder Anwendung extern erfolgen und die Kosten somit bei allen Datenstrukturen gleich hoch sind. Die Kosten werden pro Monat berechnet.

Fehleranfälligkeit Wichtig ist die Fehleranfälligkeit und Stabilität des Systems. Es muss bedacht werden, was passiert, wenn ein Teil des Systems ausfällt. Je nach System können dafür Backups eingesetzt werden oder das System ist so konstruiert, dass es in Teilen funktionsfähig ist. Verteilte Systeme sind hier designbedingt weniger anfällig. Hierbei geht es nicht nur um den Schutz vor Datenverlust, sondern auch um die Lauffähigkeit beim Ausfall von Servern.

4.2 Erklärung und Bewertung verschiedener Optionen für die Datenspeicherung des Systems

Im Folgenden werden klassische Datenbanken, Amazon Quantum Ledger Database, öffentliche Blockchains und BigchainDB betrachtet.

Klassische Datenbanken

Klassische Datenbanken weisen große Vorteile auf. Sie können riesige Mengen an Transaktionen verarbeiten und sind horizontal skalierbar. Wenn man mehr Knoten hinzufügt, können weit über 100.000 Transaktionen pro Sekunde (TPS) verarbeitet werden[65]. Eine klassische Datenbank erfüllt jedoch die zentrale Bedingung nicht, die das System benötigt: Die Garantie, dass die Daten nicht verändert werden können. Es gibt keine Mechanismen, die dies sicherstellen können. Daher werden im Folgenden Ledger-Datenbanken betrachtet.

Amazon Quantum Ledger Database

Amazon Quantum Ledger Database[4] ist ein Framework von Amazon, das auf der *AWS Cloud* basiert und einen zentralisierten Ledger einsetzt. Die Untersuchung von Amazon QLDB erfolgt repräsentativ für den Typus zentralisierter Ledger allgemein, da die grundlegenden Vor- und Nachteile vergleichbar sind. Ein ähnliches Angebot stellen *LedgerDB*[74] oder *Azure Confidential Ledger*[56] dar. Amazon QLDB basiert auf einem Ledger in Kombination mit einem *Journal*[5]. Die Übersicht dazu ist in Abbildung 4.2 zu finden. Dieses Journal wird dafür genutzt, alle Datenänderungen zu speichern. Die besondere Eigenschaft dieses Journals ist, dass nur neue Einträge hinzukommen können. Es können keine alten Einträge abgeändert oder gelöscht werden – es ist unveränderlich. Amazon bietet keine API zum Löschen an. Auch erfüllt QLDB die ACID-Kriterien. Die Datenspeicherung erfolgt dokumentenorientiert. Anfragen an das System erfolgen über eine SQL-ähnliche Anfragesprache. Da Amazon QLDB nicht Open Source ist, muss man aber weiterhin einer dritten Partei – Amazon – vertrauen.

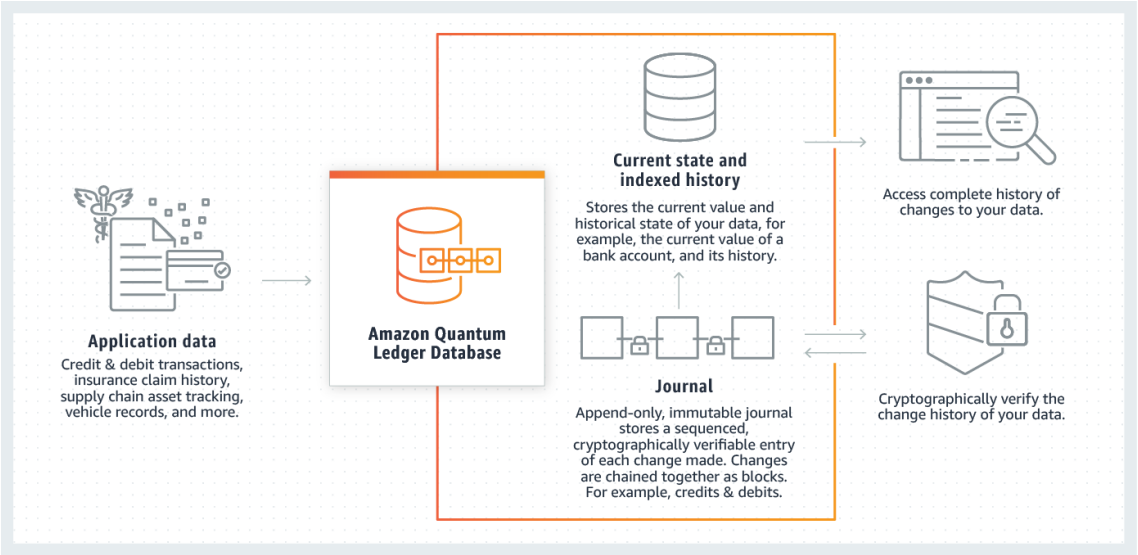

Abbildung 4.2: Zusammenhänge der verschiedenen Bestandteile von QLDB. Die Datenbank ist in Orange abgebildet und die Pfeile stellen die Interaktionen mit dem System dar. Abbildung entnommen aus [6]

Es ist es möglich, den sogenannten *Digest* herunterzuladen und mit den Daten abzugleichen, wodurch sich andere Teilnehmer am System selbst überzeugen können, dass die Daten nicht verändert wurden. Einer der Teilnehmer benötigt einen Account bei AWS und hat damit die Kontrolle über das System. Der Besitzer des Accounts kann zwar die Daten selbst nicht ändern, aber er kann die Datenbank löschen, wodurch alle Daten verloren gehen würden. Die anderen Teilnehmer des Systems können selbstständig Backups erstellen, eine gute Lösung ist dies jedoch nicht, dieses kann nicht wieder eingespielt werden. Das Backup dient nur der Datensicherung, nicht Wiederherstellung. Daher erfüllt dieses System die Anforderungen nicht ausreichend. Für den interessierten Leser folgt dennoch eine Bewertung von Amazon QLDB anhand der Kriterien. In einem anderen System, in welchem die Unveränderlichkeit der Daten vonnöten ist, es aber akzeptabel ist, dass eine

Partei die vollständige Kontrolle über das System hat, ist Amazon QLDB einsetzbar.

Die Bewertung erfolgt in Tabelle 4.3.

Kriterium	**Punktzahl**	**Begründung**
Performanz	8	Da Amazon QLDB in der AWS-Cloud liegt, profitieren alle Anfragen von dieser Infrastruktur. Zur Sicherstellung der Datenverfügbarkeit wird die Datenbank in mehreren Datenzentren repliziert[2]. Davon profitieren Lesezugriffe. Dies bedeutet jedoch, dass ein Schreibvorgang erst endgültig bestätigt ist, wenn er in mehreren Datenzentren vollendet ist, was für eine kurze Verzögerung sorgt. Außerdem führt QLDB Transaktionen parallel aus, sollte dies möglich sein. Weitere Verzögerungen entstehen durch die Überprüfung der Transaktion auf Korrektheit. Dies ist aber bei allen Systemen vonnöten, da keine falschen Daten gespeichert werden sollen. Bestimmte Teile der Daten können indexiert werden[3], wodurch ein schneller Lesezugriff sichergestellt ist.
Skalierung	9	Die Skalierung wird von AWS verwaltet und geschieht im Hintergrund. Die Infrastruktur von AWS ist auf riesige Datenmengen und Anfragen ausgerichtet, weshalb diese keinen limitierenden Faktor darstellt. Es müssen nicht manuell neue Server oder ähnliches eingerichtet werden. Wenn ein neues Unternehmen aufgenommen wird, müssen für dieses lediglich neue API-Keys und die Treiber eingerichtet werden, damit sie Daten in das System einspeisen können. Einer Untersuchung nach erreicht Amazon QLDB bis zu 10.000 Transaktionen pro Sekunde[55]. Dies ist mehr als ausreichend für das System, entsprechend der Anforderungen.
Synchronisierung	9	Die Synchronisierung ist bei Amazon QLDB kein Problem. AWS repliziert die Datenbank aus Verfügbarkeitsgründen automatisch in verschiedene Datenzentren[2], wobei eine Transaktion nur endgültig bestätigt wird, wenn sie in mehreren Datenzentren bestätigt wurde. Dies ist nicht notwendig für ein Funktionieren des Systems, es stellt eine zusätzliche Datensicherheit und Zugriffsmöglichkeit dar.

Weitergeführt auf der nächsten Seite

Tabelle 4.3 – Weitergeführt von der vorherigen Seite

Kriterium	Punktzahl	Begründung
Dokumentation / Ease of Use	7	Die Dokumentation ist sehr detailliert und hat viele Nutzungsbeispiele. Es werden Open Source Treiber für Java, .NET, Go, Node.js und Python bereitgestellt, sodass man eine Auswahl vieler beliebter Programmiersprachen hat und keine eigenen Treiber schreiben muss. Die Nutzung dadurch erleichtert, dass Amazon QLDB nur konfiguriert werden muss, aber keiner Wartung oder manueller Updates bedarf. Dies geschieht automatisch. Das Innenleben von Amazon AWS ist jedoch nicht Open Source und nicht für das Hosten auf eigenen Systemen verfügbar. Somit ist das System in den Anwendungsfällen eingeschränkt.
Kosten	9	Die Kosten sind transparent einsehbar[8] und damit planbar. Für Szenario 1 werden folgende Annahmen getroffen: Die Serverregion ist Europa (Frankfurt). Bei einer Transaktion pro Sekunde erfolgen im Monat 2.678.400 Schreibzugriffe. Die Kosten hierfür liegen bei 0,854 USD pro 1 Mio. Zugriffe, also 2,287 USD. Die gleiche Anzahl an Lesezugriffen kostet 0,166 USD pro 1 Mio. Zugriffe, also 0,444 USD. Weiter wird angenommen, dass ungefähr 126 GB (2 KB pro Transaktion, bei 1 TPS auf 2 Jahre) an Journal-Speicher und 15 GB an indizierten Speicher benötigt werden. Der Journal-Speicher kostet 0,037 USD pro GB/Monat und der indizierte Speicher kostet 0,305 USD pro GB/Monat. Somit belaufen sich die Kosten für den Journal-Speicher auf 4,666 USD und für den indizierte Speicher auf 4,575 USD. Eingehende Datenübertragungen sind kostenlos. Ausgehende Datenübertragungen sind bis zu 100 GB pro Monat kostenlos. Dieses Kontingent wird nicht überschritten. Somit belaufen sich die Kosten pro Monat auf 11,972 USD. Für Szenario 2 verhundertfachen sich die Kosten für Speicher und Schreib/Lesezugriffe. Dies sind etwa 11.972 USD im Monat. Hierzu kommen noch 525,6 GB an ausgehenden Daten. Abzüglich des 100 GB Freikontingents kosten diese Daten bei 0,09 USD pro GB insgesamt 38,304 USD. Somit belaufen sich die Gesamtkosten auf 12.010,304 USD pro Monat.

Weitergeführt auf der nächsten Seite

Tabelle 4.3 – Weitergeführt von der vorherigen Seite

Kriterium	Punktzahl	Begründung
Fehleranfälligkeit	7	Amazon garantiert, dass Datenbankvorgänge den ACID-Eigenschaften unterliegen[5]. Das System ist vollständig in der AWS-Cloud gehostet. Für die Fehleranfälligkeit hat dies sowohl Vorteile als auch Nachteile. Vorteilhaft ist die automatische Replikation der Datenbank in mehrere Datenzentren. Nachteilhaft ist der Single Point of Failure. Selbst wenn das System im Hintergrund über viele Server, die nicht am gleichen Ort sind, verteilt ist[7], schlussendlich hängt alles an einem Unternehmen. Außerdem ist es aktuell nicht möglich, Backups wieder einzuspielen, sollte es doch zu einem Datenverlust kommen[2].

Tabelle 4.3: Bewertung von Amazon QLDB anhand der in Kapitel 4.1 aufgestellten Kriterien.

Öffentliche Blockchains

Eine weitere Idee könnte sein, eine öffentliche Blockchain zu nutzen, etwa die *Bitcoin-* oder *Ethereum-Blockchain*. Diese Netzwerke haben teilweise über einhunderttausend aktive Knoten[33]. Die Daten wären also global repliziert und aufgrund der Funktionsweise einer Blockchain vor Änderungen geschützt. Es sind auch kleinere öffentliche Blockchains mit einem anderen Fokus (Bitcoin und Ethereum sind primär als Währung und nicht als Datenspeicher gedacht) denkbar. Es gibt hier aber viele Dinge zu bedenken. Zunächst sind alle Daten, die in eine öffentliche Blockchain aufgenommen werden, auf ewig für alle verfügbar. Dies stellt ein Problem dar, wenn Daten protokolliert werden sollen, die aber nicht veröffentlicht werden sollen. Um dies zu umgehen, könnten die Daten verschlüsselt werden. Es besteht die Gefahr, dass der Schlüssel auf irgendeine Weise in die Hände von Unbefugten gelangt und somit alle Daten entschlüsselt werden können. In einem solchen Fall kann der Schlüssel nicht mehr geändert werden. Zudem können auf diese Art ohnehin nicht alle Informationen verschlüsselt werden.

Zusätzlich in Betracht gezogen werden kann die Verwendung von Side-Chains. Eine Option ist dabei, schon vorhandene Side-Chains einzusetzen. Diese haben eine kleinere Nutzungsbasis als die Mutterchain. Bei einer Proof-of-Work-Blockchain bedeutet dies weniger Sicherheit. Eine eigene Side-Chain abzuspalten, ist ebenso nicht sinnvoll. Da das System keine Notwendigkeit für Token hat und die anderen Teilnehmer an der übergeordneten Blockchain die Assets aus dem System nicht benötigen, kann man genauso gut eine eigene private Blockchain hosten.

Trotzdem wollen wir betrachten, wie eine öffentliche Blockchain als Datenspeicher genutzt werden könnte: Die anfallenden Daten werden als Payload einer Transaktion an die Blockchain gesendet. Um die Daten auszulesen, hostet der Webserver einen eigenen Knoten und verfügt somit immer über eine aktuelle Kopie der Blockchain.

Die Bewertung des allgemeinen Konzepts öffentliche Blockchain erfolgt in Tabelle 4.4.

Kriterium	Punktzahl	Begründung
Performanz	3	Es kann Sekunden, Minuten oder Stunden dauern, bis eine Transaktion in eine öffentliche Blockchain aufgenommen wird, da die Miner entscheiden, welche Transaktionen sie aufnehmen und welche nicht. Diese Netze sind darauf ausgelegt, Blöcke in bestimmten Intervallen aufzunehmen. Bitcoin strebt ein 10 min Intervall an[60], Ethereum ein 15 s Intervall[58]. Hinzu kommt, dass eine Blockchain aus der Struktur heraus nicht sehr effizient zu durchsuchen ist. Um also Datenabfragen sinnvoll zu verwalten, wird zusätzlich eine normale Datenbank benötigt, die als Index für die Blockchain dient. Außerdem müssen alle Daten, die in der Blockchain anfallen, verwaltet werden, obwohl diese für den Anwendungsfall keine Bedeutung haben.
Skalierung	0	Die Ethereum-Blockchain kann maximal 30 Transaktionen pro Sekunde verarbeiten[73]. Ähnlich sieht es bei anderen Blockchains aus. Bitcoin kann beispielsweise maximal 27 Transaktionen pro Sekunde verarbeiten[43]. Das System wäre dementsprechend begrenzt und würde in einer kleinen Form einen signifikanten Teil eines globalen Netzwerks für sich in Anspruch nehmen.
Synchronisierung	5	Die Synchronisierung der Daten in der Blockchain wird vom Netzwerk selbst verwaltet. Dies geschieht dadurch, dass Miner Transaktionen zu Blöcken hinzufügen. Hier kommt wieder das gleiche Problem zu Tragen wie bei der Performanz – es kann eine sehr lange Zeit dauern, bis eine Transaktion bestätigt wurde. Nachdem die Transaktion einem Block hinzugefügt wurde, muss der Block noch durch das Netzwerk propagiert werden. In der Bitcoin Blockchain haben nach 40 s etwa 95% der Knoten den Block erhalten[31]. Ein weiteres Problem kann durch den Fakt entstehen, dass Proof of Work Blockchains die längste Kette als tatsächlichen Zustand akzeptieren. Dabei kann es kurzzeitig zu Forks kommen, also Aufspaltungen der Kette[64]. Diese lösen sich zwar selbst wieder auf, können aber kurzfristig dazu führen, dass ein falscher Zustand vermittelt wird.

Weitergeführt auf der nächsten Seite

Tabelle 4.4 – Weitergeführt von der vorherigen Seite

Kriterium	Punktzahl	Begründung
Dokumentation / Ease of Use	4	Öffentliche Blockchains sind vollständig Open Source. Daher kann man verständlich die Verhaltensweise nachvollziehen. Da diese Blockchains nicht auf eine Nutzung als Datenspeicher ausgelegt sind, müssen entsprechende Treiber selbst entwickelt werden.
Kosten	0	Für die Kosten wird die öffentliche Ethereum Blockchain als Basis genommen. Diese sind abhängig vom Wechselkurs, welcher aktuell (12. September 2022) bei etwa 1.700 USD pro Ether liegt. Für das Speichern von Daten in der Ethereum-Blockchain fallen Kosten in sogenanntem *Gas* an. 2 KB an Daten (die Größe einer großen Transaktion) benötigen 1.280.000 Gas. Dies entspricht 0,064 Ether. Beim aktuellen Wechselkurs sind dies 108,8 USD. Bei einer Transaktion pro Sekunde wären dies die sekündlichen Kosten für das erste Szenario. Pro Monat also ungefähr 283.824.000 USD. Das zweite Szenario ist nicht möglich, da Ethereum nicht so viele Transaktionen pro Sekunde verarbeiten kann.
Fehleranfälligkeit	6	Eine große öffentliche Blockchain wie Ethereum ist aufgrund der vielen Knoten, die am System teilnehmen, sehr ausfallresistent. Es gibt keinen zentralen Server, der gewartet werden muss. Außerdem ist durch die globale Verteilung der Knoten kein separates Backup nötig. Gleichzeitig ist es in einer öffentlichen Blockchain nicht garantiert, dass eine Transaktion in die Blockchain übernommen wird. Die Miner entscheiden selbst, welche Transaktionen von ihnen aufgenommen werden. Dabei entscheiden diese sich meist für die Transaktionen, welche die höchste Gebühr zahlen.

Tabelle 4.4: Bewertung von einer öffentlichen Blockchain anhand der in Kapitel 4.1 aufgestellten Kriterien.

Die Blockchain Datenbank BigchainDB

Eine weitere Option ist *BigchainDB*[18, 12]. BigchainDB verfolgt das Ziel, die Vorteile einer Datenbank (Hohe Transaktionsrate, geringe Latenz, Indexing, Abfrage strukturierter Daten) mit den Vorteilen einer Blockchain (Dezentralisierung, Unveränderlichkeit, besitzerkontrollierte Assets) zu kombinieren[18].

BigchainDB sieht alles als sogenanntes *Asset* und setzt auf ein Transaktionsmodell. Ein

solches Asset besteht aus Daten, die das Asset beschreiben sollen, und einer ID[14]. Das Asset selbst ist unveränderlich. Um neue Daten mit einem Asset zu verknüpfen, werden Metadaten von Transaktionen genutzt. Dies wird weiter unten ausgeführt. Eine Transaktion besteht aus Inputs, Outputs, dem Asset, Metadaten und einer TransaktionsID. Für die Daten gibt es zwei relevante Transaktionen, die CREATE-Transaktion und die TRANSFER-Transaktion. Zusätzlich gibt es noch die Transaktionen VALIDATOR_ELECTION, CHAIN_MIGRATION_ELECTION und VOTE. Betrachten werden nun die CREATE- und TRANSFER-Transaktionen. Die Inputs und Outputs stellen sicher, dass nur die Person ein Asset transferieren kann, die es können soll. Dies wird mittels Signaturen und einem Public-Key-Verfahren sichergestellt. Im Asset-Feld kann das Asset mit einem assoziativen Array beschrieben werden. In diesem steht bei TRANSFER-Transaktionen anstatt der Daten eine Referenz auf eine Transaktion. Im Metadaten-Feld kann bei jeder Transaktion ein beliebiges assoziatives Array eingefügt werden. Dieses Feld kann dann als Update für das Asset interpretiert werden. Wichtig hierbei ist, dass vorherige Transaktionen niemals überschrieben werden. Es können nur neue Transaktionen ergänzt werden.

Um zu verstehen, wie man dieses Konzept anwenden kann, um ein Lebensmittel entlang der Lieferkette zu protokollieren, folgt ein Beispiel: Ein Lachs wurde gefangen und verpackt. Jetzt kann ein Asset angelegt werden. In das Asset-Feld des Assets können Daten geschrieben werden wie eine ID, Fangdatum und Fanggebiet, Produzent und ähnliche Daten. Fallen in der Lieferkette nun Messdaten wie eine Temperaturmessung an, so kann eine TRANSFER-Transaktion an sich selbst genutzt werden. Die können wieder als assoziatives Array in das Metadaten-Feld geschrieben werden. Sollte beim Transport die Verantwortlichkeit für das Lebensmittel wechseln, so kann hierfür eine TRANSFER-Transaktion genutzt werden. Dadurch kann die alte Partei dem Asset über die Metadaten keine neuen Daten mehr hinzufügen. Dies kann nur der neue Besitzer. Ein vollständiger Lesezugriff besteht weiterhin für alle Beteiligten.

So funktioniert das Modell von außen betrachtet konzeptionell. Es ist ein verteiltes System, welches auf die Nutzung durch ein Konsortium ausgelegt ist. Dementsprechend besteht ein BigchainDB-Netzwerk aus mehreren Knoten. Die einzelnen Knoten in diesem System bestehen primär aus drei Bestandteilen, die in Abbildung 4.5 dargestellt sind.

Das Erste ist eine *MongoDB*-Instanz[59]. MongoDB ist eine dokumentenorientierte Datenbank. Die Daten sind in mehreren Kollektionen gesammelt.

Der nächste Bestandteil ist ein *BigchainDB-Server*[12]. Dieser verwaltet Anfragen an das System über eine HTTP-Schnittstelle. Es können mit GET-Requests bestimmte einfache Daten angefragt werden. Sollten komplexere Anfragen notwendig sein, ist es möglich, die MongoDB Datenbank selbst abzufragen. Mit POST-Requests können Transaktionen in das System eingebracht werden. Der Server überprüft, ob diese Transaktionen valide sind, also alle Bedingungen erfüllen. Transaktionen, welche die Bedingungen erfüllen, sendet der Server an die Tendermint-Instanz, die sich im selben Knoten befindet.

Tendermint[66] übernimmt die Kommunikation zwischen den Knoten. Die Knoten kommunizieren untereinander mit einem *Gossip-Protokoll*. Hier hat jeder Knoten eine Anzahl an Nachbarn und er kommuniziert nur mit diesen. Auf diese Art können Nachrichten im

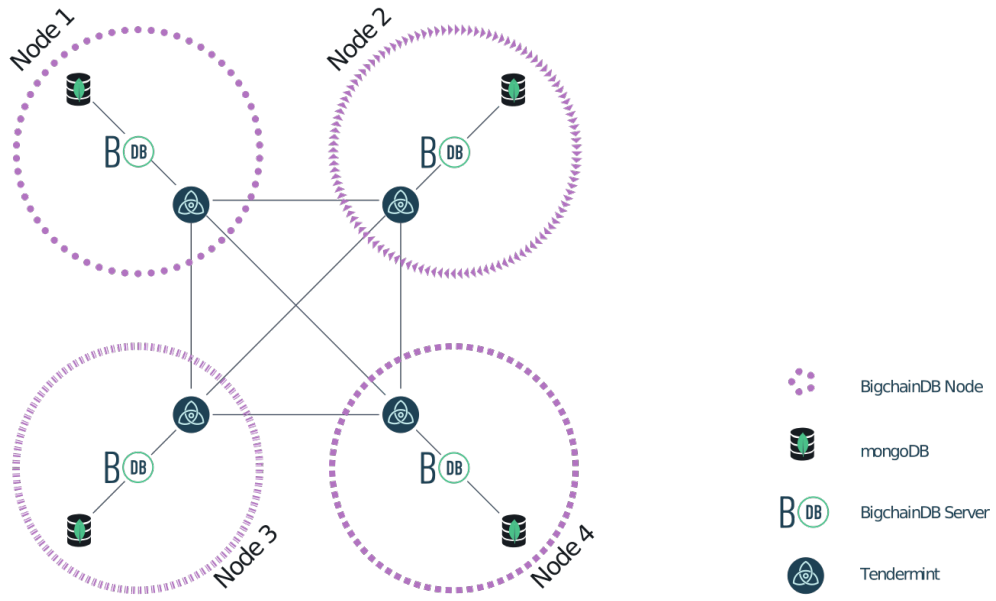

Abbildung 4.5: Ein BigchainDB-Netzwerk mit 4 Knoten. Es ist zu erkennen, dass jeder Knoten seine eigene MongoDB-Instanz hat mit einem BigchainDB Server. Die Kommunikation zwischen den Knoten findet über Tendermint statt. Abbildung entnommen aus [16]

System jeden Knoten erreichen, ohne dass alle Knoten alle anderen kennen müssen. Tendermint nutzt intern eine Blockchain und jeder Knoten besitzt eine vollständige Kopie. Als Konsens-Protokoll wird eine modifizierte Version des DLS-Protokolls[35, 20] eingesetzt. Tendermint ermöglicht es, dass die Knoten ein unterschiedliches Stimmgewicht haben können. Da Tendermint darauf abzielt, als allgemeines Blockchainframework zu fungieren, kann dies in anderen Anwendungsfällen sehr sinnvoll sein. Für BigchainDB wird den Knoten normalerweise ein gleiches Stimmgewicht zugewiesen. Wenn im Folgenden die Rede von x% der Knoten die Rede ist, ist daher x% der Stimmen gemeint. Ein Round-Robin Algorithmus wählt aus der Menge der Validators (Knoten mit Stimmgewicht größer 0) einen Proposer, der den abzustimmenden Block auswählt. Ist ein Block bestätigt, kann er nicht mehr abgeändert werden.

Die Bewertung der BigchainDB erfolgt in Tabelle 4.6.

Kriterium	Punktzahl	Begründung
Performanz	7	BigchainDB kann verschieden eingestellt werden. Diese Einstellungen haben einen großen Einfluss auf die Performanz. Eine wichtige Metrik hierbei ist die Zeit, bis eine Transaktion in einem Block bestätigt wurde. Einer Testreihe zufolge, die mit vier Knoten in Azure durchgeführt wurde, wurden 99,7% der Transaktionen, je nach Einstellung, innerhalb von 4,358 s bis 9,392 s vollständig aufgenommen[15].
Skalierung	7	Die Einstellungen aus der Performanz wirken sich auf die Skalierung aus. Je nach Einstellung konnten zwischen durchschnittlich 298 Transaktionen pro Sekunde und 889 Transaktionen pro Sekunde erreicht werden[15]. In anderen Tests wurde die Skalierung mit unterschiedlicher Anzahl von Clients und Serverknoten getestet. In diesen Tests blieben die TPS bei 4 bis 64 Serverknoten auf dem gleichen Niveau. Die Anzahl an Clients hatte zwischen 4 und 256 keinen Einfluss auf die TPS. Die Verzögerung ist jedoch signifikant angestiegen, um bis zu 1,5 s bei 256 Clients.[41]
Synchronisierung	8	Die Synchronisierung der Knoten erfolgt über Tendermint. Wenn ein neuer Knoten in das Netzwerk aufgenommen wird oder wenn ein Knoten hinterher hängt, fragt der Knoten die Daten bei den anderen Knoten in seiner Nachbarschaft an. Aufgrund des Gossip-Protokolls muss nicht jeder Knoten jeden anderen kennen, dies vereinfacht die Kommunikation. Da die Aufnahme neuer Blöcke über ein Abstimmungssystem erfolgt, ist mindestens der notwendigen 2/3 Mehrheit der aktuelle Zustand bekannt, die restlichen Knoten synchronisieren sich von allein. Die Notwendigkeit zur Synchronisierung stellt aber eine mögliche Quelle für Datenkonflikte dar, die sich auf die Performanz auswirken können.

Weitergeführt auf der nächsten Seite

Tabelle 4.6 – Weitergeführt von der vorherigen Seite

Kriterium	Punktzahl	Begründung
Dokumentation / Ease of Use	6	BigchainDB ist vollständig Open Source[12]. Dementsprechend ist es immer möglich, selbst nachzusehen, wie bestimmte Dinge funktionieren und diese eventuell anzupassen. Es gibt offizielle Treiber für Javascript, Python und Java[13]. Weiter gibt es eine offizielle Dokumentation[11]. Diese ist jedoch in Teilen nur für ältere Versionen verfügbar. Da BigchainDB eine sehr große Änderung in der zweiten Version hatte, ist die ältere Version häufig nicht anwendbar. Es gibt einige Guides, welche die Kernaspekte des Systems erklären und demonstrieren[17].
Kosten	7	Zur Kostenberechnung werden als Grundlage virtuelle Maschinen genommen, die in Azure gehostet werden. Zur Preisberechnung wird der Azure-Preis-Rechner genutzt[57]. Für Kostenbeispiel 1 werden folgende Einstellungen getroffen: 4 VMs des Typs B1s. Es werden 4 BigchainDB Knoten benötigt für ein minimales Setup. Diese werden für 3 Jahre reserviert. Als Speicher werden 4 HDD Standard des Typs S10: 128 GiB gewählt. Bei der ausgehenden Datenübertragung wird mit etwa 5 GB im Monat gerechnet. Die vom Rechner geschätzten monatlichen Kosten belaufen sich auf 47,84 USD im Monat. Für Kostenbeispiel 2 werden folgende Einstellungen getroffen: 10 VMs des Typs F32s v2. Als Speicher werden 10 SSDs Standard des Typs E70: 16.384 GiB gewählt. Bei der ausgehenden Datenübertragung wird mit etwa 500 GB im Monat gerechnet. Die vom Rechner geschätzten monatlichen Kosten belaufen sich auf 19.169,28 USD im Monat.

Weitergeführt auf der nächsten Seite

Tabelle 4.6 – Weitergeführt von der vorherigen Seite

Kriterium	**Punktzahl**	**Begründung**
Fehleranfälligkeit	8	Tendermint ist Byzantine Fault Tolerant mit bis zu einem Drittel Knoten[19, 66, 20], die einen byzantinischen Fehler aufweisen. Da BigchainDB für sämtliche Kommunikation zwischen den Knoten auf Tendermint setzt, ist BigchainDB dies ebenso. Für das Hinzufügen neuer Einträge benötigt es der Zustimmung von mehr als zwei Drittel der Knoten. Bis zu einem Drittel der Knoten können ausfallen, nicht reagieren oder sogar falsche Daten senden, ohne, dass das System dadurch beeinträchtigt wird. Im Gegensatz zu einer öffentlichen Blockchain besteht in einem geschlossenem System wie diesem hier nicht das Problem, dass die Knoten sich für andere Transaktionen entscheiden. Alle am System beteiligten Parteien sind an dessen Betrieb interessiert.

Tabelle 4.6: Bewertung von BigchainDB anhand der in Kapitel 4.1 aufgestellten Kriterien.

Im Juli 2022 veröffentlichte die Interplanetary Database Association (IPDB) e.V eine Weiterentwicklung von BigchainDB namens *PlanetMint*[36]. PlanetMint bietet mehr Funktionen und ist allgemeiner einsetzbar als BigchainDB[37]. Dieses Framework wurde im Bearbeitungszeitraum dieser Arbeit veröffentlicht, daher sind bisher noch nicht ausreichend Quellen und Informationen verfügbar, um es in diesen Vergleich mit aufzunehmen. Daher bleibt es bei dieser Erwähnung.

4.3 Ergebnisanalyse der Bewertungen

Datenbank	QLDB	Blockchain	BigchainDB
Performanz	8	3	7
Skalierung	9	0	7
Synchronisierung	9	5	8
Dokumentation / Ease of Use	7	4	6
Kosten	9	0	7
Fehleranfälligkeit	7	6	8
Durchschnitt	8,2	3	7,2
Median	8,5	3,5	7
Minimum	7	0	6
Maximum	9	6	8

Tabelle 4.7: In dieser Tabelle sind die Ergebnisse aus der Analyse zusammengefasst.

Die tiefgehende Bewertung der drei möglichen Frameworks hat das Ergebnis aus Tabelle 4.7 ergeben. Diese Ergebnisse überraschen nicht. Amazon QLDB hat in allen Kategorien

außer der Fehleranfälligkeit am besten abgeschnitten. Aufgrund der zentralisierten Struktur ist es das effizienteste Framework. Wie aber schon in der Analyse angesprochen wurde, ist QLDB gerade aufgrund der zentralisierten Struktur ungeeignet für diesen spezifischen Anwendungsfall.

Die öffentliche Blockchain hat in jeder Kategorie am schlechtesten abgeschnitten. Sie hat in zwei Kategorien sogar nur 0 Punkte erreicht. Diese Technologie ist dementsprechend ungeeignet, unabhängig davon, wie die Anderen bewertet werden.

Als Letztes bleibt BigchainDB. BigchainDB hat in jeder Kategorie zumindest gut abgeschnitten und erfüllt somit die Anforderungen ausreichend.

Zusammenfassung der Ergebnisse dieses Kapitels

In diesem Kapitel wurden zunächst Vergleichskriterien definiert. Mit den Anforderungen und den Vergleichskriterien wurden klassische Datenbanken, Amazon Quantum Ledger Database, öffentliche Blockchains und BigchainDB untersucht. Dabei wurde herausgefunden, dass BigchainDB am besten für den Zweck geeignet ist.

Die Datenstruktur ist jedoch noch nicht alles. Das Framework benötigt eine API, um sinnvoll und vollständig zu sein. Darum geht es im nächsten Kapitel.

5

API für das System

Nun soll eine API entwickelt werden. Um hier eine Entscheidung zu treffen, muss zunächst klar sein, was gefordert wird und was für Schnittstellen eventuell schon gegeben sind. Die Zusammenhänge des Systems und wie es genutzt werden wird, ist in Abbildung 5.1 dargestellt. Hier müssen zwei Akteure getrennt betrachtet werden. An erster Stelle stehen die Produzenten. Diese füllen das System mit Daten. Dies erfolgt direkt über eine API von BigchainDB, hierfür muss nichts entworfen werden. Sie können über diese API auch Daten abrufen. Ein Benutzer ruft die Daten über eine Webseite ab, auf der die ID des Lebensmittels eingegeben werden kann. Die Schnittstelle zwischen Benutzer und Webseite existiert noch nicht (die Webseite auch noch nicht, siehe folgendes Kapitel 6). Die Webseite ruft die angeforderten Daten dann direkt über MongoDB ab. Alternativ kann sie die Daten direkt über BigchainDB abrufen. Für beide Optionen gibt es Schnittstellen von den entsprechenden Systemen. Diese Daten werden möglicherweise noch von einem KI-Modell ausgewertet und schlussendlich an den Benutzer zurückgeliefert.

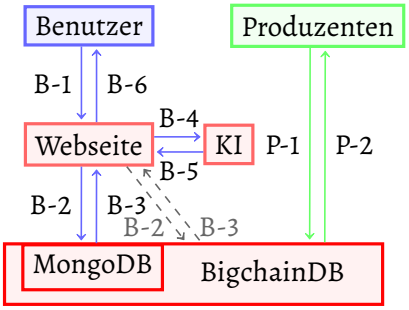

Abbildung 5.1: Darstellung der Schnittstellen des Gesamtsystems: Die Benutzer rufen Daten über die Webseite ab. Diese greift auf die geforderten Daten über MongoDB zu. Sie hat zusätzlich die Möglichkeit, auf die Daten über BigchainDB zuzugreifen. Im nächsten Schritt werden die Daten von einem KI-Modell ausgewertet und an den Nutzer geschickt. Der Produzent sendet neue Daten direkt an BigchainDB zur Verarbeitung.

Es stehen also noch zwei Schnittstellen aus. Einmal die Schnittstelle zwischen dem KI-Modell und der Webseite und einmal die Schnittstelle zwischen der Webseite und dem

Benutzer. Dieses Framework soll möglichst mit beliebigen KI-Modellen funktionieren. Daher ist für diese Schnittstelle im Framework eine Klasse mit einer öffentlichen Funktion vorgesehen. Die Funktion nimmt die Daten als Key-Value-Objekt der Programmiersprache und gibt als Return-Wert die Auswertung des Modells zurück. Somit kann für ein anderes KI-Modell einfach diese Klasse ausgetauscht werden.

Für die Schnittstelle zwischen der Webseite und dem Benutzer ist eine RESTful API sinnvoll. Im System sind die Nahrungsmittel als einzigartiges Asset mit einer ID repräsentiert. Eine RESTful API bildet dies gut ab. So kann auf ein Asset über HTTP zugegriffen werden. Die URL ist wie folgt aufgebaut:

<Name der Webseite>.<TLD>/<api/site>/v<Versionsnummer>/food/<ID des Assets>

Der Zugriff erfolgt mit der HTTP GET Methode. Die weiteren HTTP-Methoden werden erst einmal nicht unterstützt, könnten aber später ergänzt werden. Durch die Angabe von *api* oder *site* kann zwischen einem API-Zugriff und einem Zugriff auf die Webseite unterschieden werden. Bei Ersterem werden die Daten als JSON geliefert, bei Letzterem werden sie visualisiert dargestellt. Es wird *food* spezifiziert, dadurch ist das System in andere Bereiche erweiterbar, ohne mit dem aktuellen Anwendungsfall zu kollidieren. Durch die Versionsnummer in der URL ist es möglich neue Versionen der API einzusetzen, ohne die Funktionsweise alter Links zu ändern.

6

Realisierung eines Proof of Concepts und Diskussion

Aufbauend auf den Erkenntnissen und der Arbeit der vorherigen Kapitel, soll nun ein Proof of Concept zeigen, dass BigchainDB tatsächlich so wie dargestellt zur Datenspeicherung genutzt werden kann. Dieser Proof of Concept wird aus BigchainDB, einem Flask-Server und einer React-Webseite bestehen.

BigchainDB

Für BigchainDB existiert ein *Docker-Image*. In diesem sind Tendermint, MongoDB und BigchainDB-Server installiert und konfiguriert. Läuft der Container, kann mit allen drei Bestandteilen des Systems über drei Ports kommuniziert werden. Es gibt zusätzlich die Möglichkeit, alle Komponenten manuell aufzusetzen und zu konfigurieren. Diese Konfiguration besitzt jedoch sämtliche Funktionen, die für den Proof of Concept notwendig sind. Im Produktiveinsatz kann eine Anpassung zur Erhöhung der Performanz durch Feinjustierung der Einstellungen sinnvoll sein.

Für den Proof of Concept reicht ein Knoten aus. Die Einschränkung auf einen Knoten hat einen Einfluss auf die Performanz. Daher ist diese nicht repräsentativ für einen Produktiveinsatz. Außerdem können bei einem Netz aus nur einem Knoten keine byzantinischen Fehler verhindert werden. Diese beiden Aspekte sind jedoch in Kapitel 4.4 erklärt und in den verlinkten Arbeiten analysiert bzw. bewiesen.

Server

Um eine sinnvolle Wahl für den Server zu treffen, muss zunächst klar sein, was dieser können muss. Dieser stellt den Knotenpunkt zwischen BigchainDB, der Webseite und dem KI-Modell dar. Daher muss er mit allen diesen Komponenten kommunizieren können. Für die Kommunikation mit der Webseite ist dies eine RESTful HTTP Schnittstelle. Das KI-Modell kann beliebig implementiert sein. Daher reicht als Anforderung hier, dass entsprechend ein Modul mit einer definierten Schnittstelle für das System programmiert

werden kann, sodass für das KI-Modell nur dieses Modul getauscht werden muss. Für BigchainDB existieren offizielle Treiber für Python, Javascript und Java[13]. Dementsprechend ist ein Server in einer dieser drei Sprachen sinnvoll. Hier fällt die Wahl auf *Flask*[61], ein Python Web-Framework. Dieses Framework erfüllt die Anforderungen. Außerdem ist es mir persönlich bekannt, weshalb keine neue Einarbeitung erfolgen muss. Für einen Produktiveinsatz ist eine genauere Evaluation dieser und anderer Optionen sinnvoll.

Webseite

Die Webseite soll den Zweck erfüllen, dem Konsumenten die Daten auf eine gebrauchstaugliche Art darzustellen. Dementsprechend ist diese minimalistisch, mit nur einem Eingabefeld entworfen. In diesem kann er die ID des Nahrungsmittels eingeben. Daraufhin werden ihm als Resultat die Daten, die über dieses Nahrungsmittel gespeichert sind, präsentiert. Als weitere Option kann dieses Suchergebnis direkt über die URL erreicht werden. So kann ein QR-Code auf dem Nahrungsmittel platziert werden. Über diesen kann der Konsument sich die Daten für das Nahrungsmittel direkt anzeigen lassen.

Zusätzlich wird das Ergebnis des KI-Modells angezeigt. Für diesen Proof of Concept wird davon ausgegangen, dass das KI-Modell die Haltbarkeit des Nahrungsmittels bewertet und ein Datum zurückliefert. Das Ergebnis ist gefärbt, abhängig davon, wie lange das Nahrungsmittel noch haltbar ist. Liegt die Haltbarkeit in der Vergangenheit, so wird dies durch einen roten Hintergrund zusätzlich signalisiert. Wenn die Haltbarkeit auf bis zu drei Tage in die Zukunft geschätzt wird, wird dies durch einen gelben Hintergrund visualisiert. Ansonsten ist der Hintergrund grün. Für ein anders entworfenes KI-Modell kann dies selbstverständlich angepasst werden.

Für die Webseite wird *React*[38] genutzt. React ermöglicht eine komponentenbasierte Webentwicklung und ist mir bekannt. Daher muss hier keine Einarbeitung erfolgen. Ob besser geeignete Optionen existieren, muss für einen Produktiveinsatz untersucht werden. Für den Proof of Concept sind die Funktionen von React ausreichend.

Implementierung

Jetzt da geklärt ist, wie der Proof of Concept funktionieren soll, muss dieser implementiert werden. Die erste Komponente ist BigchainDB. Für BigchainDB existiert ein all-in-on-Docker-Image. Dieses Image wird beim Starten so konfiguriert, dass die Ports für BigchainDB-Server, Tendermint und MongoDB offen sind.

Als Nächstes folgt der Flask-Server. Dieser besteht aus vier Teilen. Sie Serverapp selbst wird in main.py initialisiert. Die API-Routen sind in routes.py definiert. Diese rufen eine Funktion in bigchaindb.py auf. Bigchaindb.py ist gleichzeitig die Schnittstelle zu BigchainDB und fragt die geforderten Daten an. Die Daten werden dort gefiltert und zur Evaluation an die ai.py übergeben. Danach werden die Daten zurück an die routes.py übergeben, welche diese als HTTP Response an die Webseite schickt.

Die Webseite hat diese Daten vorher angefragt. Sie besteht aus mehreren Komponenten. Die searchbar.js verwaltet die Funktionen der Suche. Dem gegenüber steht die resultbox.js.

Hier werden die Daten zergliedert in ihre Bestandteile und entweder entsprechend direkt visualisiert oder an singleentry.js weitergeleitet. Dort werden die Daten aus dem JSON Format in eine visuelle Darstellung umgewandelt.

Dies alles ist in Abbildung 6.1 dargestellt.

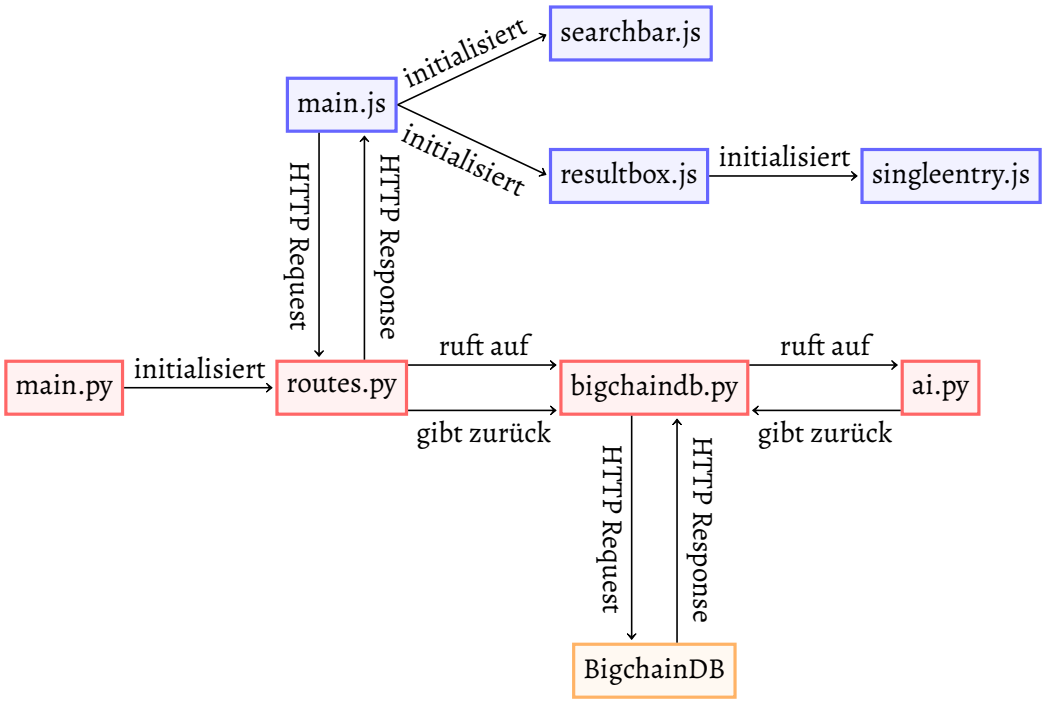

Abbildung 6.1: Hier sind die einzelnen Komponenten des Proof of Concepts visualisiert. Oben sind die Elemente der Webseite in Blau dargestellt. In der Mitte in Rot sind die Abläufe im Server zu sehen. Die Webseite kommuniziert mit der API des Servers, welcher die Anfrage weiter verarbeitet. Der Server selbst fragt die Daten bei BigchainDB an.

Proof of Concept

Nachdem BigchainDB, der Server und die Webseite eingerichtet sind, muss dieser Proof of Concept getestet werden. Dafür wird ein Fisch als Asset angelegt, mit den Daten wie in Programmtext 6.2. In der Lieferkette werden Daten protokolliert. Diese sind beispielhaft dargestellt durch die Daten in den Programmtexten 6.3 und 6.4

Programmtext 6.3: Beispielhafte Protokollierung der Lieferkette im JSON-Format.

```
{
    "timestamp": "2022-09-11",
    "temperature_reading": "150 K",
    "location_gps": "54.02945665096139, 10.958993017386137",
    "shipping_company": "Water Transportation Systems GMBH"
}
```

Programmtext 6.2: Beispielhafte Daten eines Lebensmittels im JSON-Format.

```
{
    "data": {
        "fish": {
            "kind_of_fish": "flying fish",
            "fish_catched_in": "baltic sea"
        },
        "fishing_date": "2022-09-01",
        "id": "A1B2C3D4"
    }
}
```

Programmtext 6.4: Beispielhafte Protokollierung der Lieferkette im JSON-Format.

```
{
    "timestamp": "2022-09-21",
    "temperature_reading": "220 K",
    "location_gps": "52.498659289205754, 13.352006350951045",
    "shipping_company": "Max Schnellfahrer Road Deliveries"
}
```

Wenn nun auf der Webseite des Proof of Concepts oder über die URL nach dem Nahrungsmittel mit der ID A1B2C3D4 gesucht wird, sieht die Webseite wie in Abbildung 6.5. In dieser Abbildung ist die Aufteilung der verschiedenen Einzelteile gut zu erkennen. Oben ist die Suchleiste. Unter dieser befindet sich das Ergebnis der KI-Evaluation.

Diskussion

Nun ist der Proof of Concept entwickelt und getestet. Es kann festgestellt werden, dass sowohl die Speicherung von Daten in BigchainDB als auch das Anzeigen dieser Daten funktioniert. Die Nahrungsmittel werden in BigchainDB als Asset repräsentiert und die Daten in einem JSON-Format gespeichert. Im Nachhinein bestätigt sich, dass dies eine adeäquate Lösung darstellt. Die Daten werden, wie gefordert, unveränderlich gespeichert und sind im Produktiveinsatz auf mehrere Server repliziert.

Für die API gilt Ähnliches. Sie ist simpel designt und erfüllt den Zweck vollkommen. Dadurch ist diese einfach zu verwalten und gut auf als QR-Code auf dem Produkt anbringbar.

Die Webseite erfüllt ihren Zweck. Sie ist einfach zu bedienen und zeigt alle notwendigen Informationen an. Durch minimale Optionen werden Bedienfehler vermieden.

Als Fazit bestätigt dieser Proof of Concept die Viabilität dieses Systems.

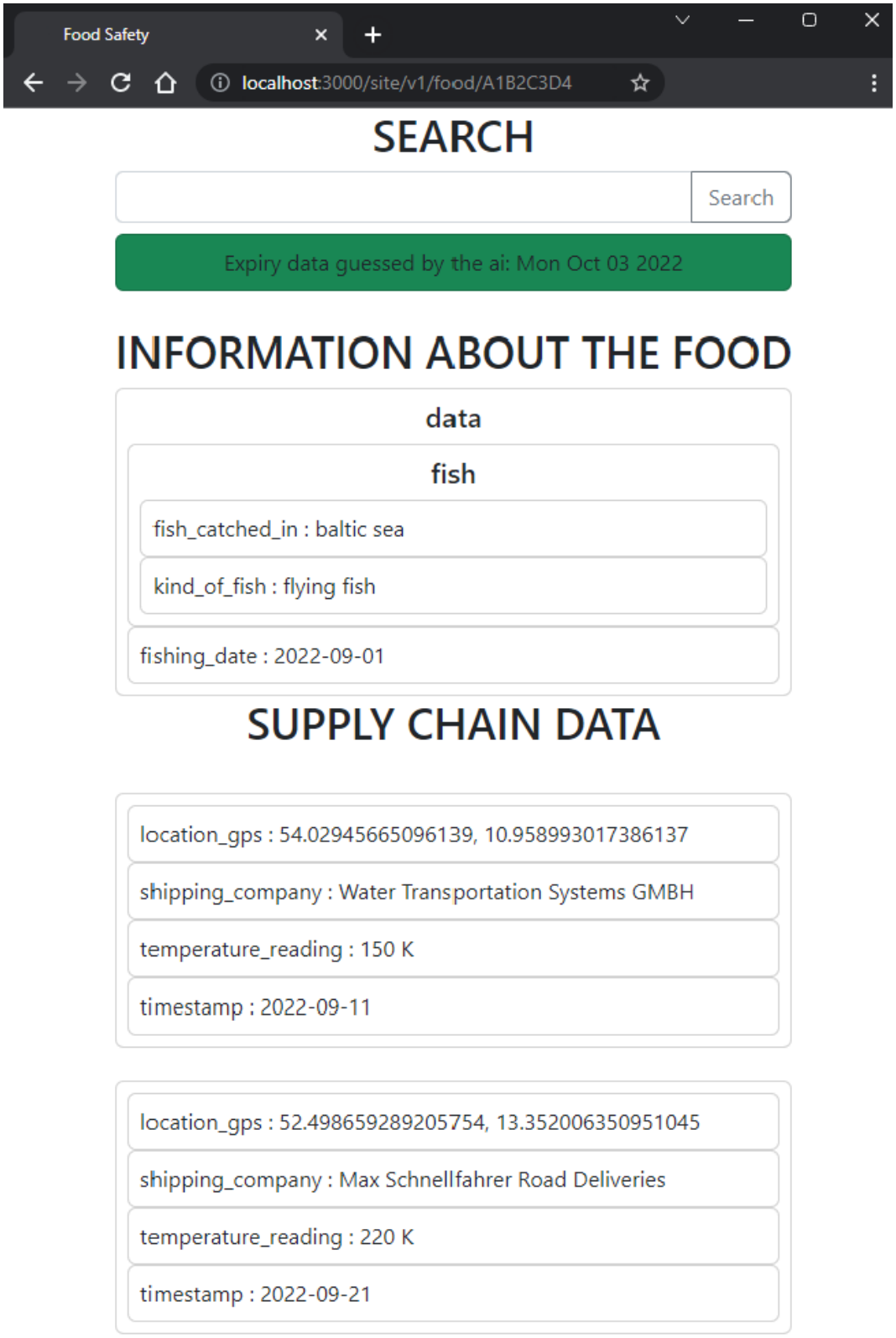

Abbildung 6.5: In diesem Bild ist zu sehen, wie die Webseite des Proof of Concepts aussehen kann. Im oberen Teil werden die Daten des Fisches angezeigt. Im unteren Teil werden die Daten angezeigt, die in der Lieferkette protokolliert wurden. Dieser Screenshot wurde am 28. Oktober 2022 aufgenommen, der 03. Oktober 2022 war mehr als drei Tage in der Zukunft. Daher ist die KI-Vorhersage in Grün.

7

Zusammenfassung und Ausblick

In dieser Arbeit wurden verschiedene Möglichkeiten untersucht, Lieferketten unveränderlich zu protokollieren. Nach einer Motivation in Kapitel 1 wurden die technischen Grundlagen in Kapitel 2 geklärt. Dazu gehören Begriffe aus den Datenbanken, Ledgers und APIs. Danach wurden Lieferketten für Lebensmittel in Kapitel 3 genauer betrachtet. Aus den Hürden, die aus dieser Betrachtung hervorgingen, wurden Anforderungen erstellt. Diese sollte ein System erfüllen.

In Kapitel 4 wurde ein geeignetes System gesucht, welches die Anforderungen erfüllen kann. Dafür wurden in Abschnitt 4.1 zunächst Vergleichskriterien aufgestellt. In Abschnitt 4.2 wurden klassische Datenbanken, Amazon Quantum Ledger Database, öffentliche Blockchains und BigchainDB erklärt und anhand der Kriterien analysiert. In Abschnitt 4.3 wurden die Ergebnisse der Analyse gegenübergestellt. Dabei hat sich ergeben, dass BigchainDB am ehesten geeignet ist. Im 5. Kapitel wurde darauf aufbauend eine API entworfen, welche die verschiedenen Teile des Systems verbinden können.

Danach wurde in Kapitel 6 ein Proof of Concept entworfen und programmiert. Dieser basiert auf BigchainDB, einem Flask-Server und einer React-Webseite. Die Ergebnisse aus dem Proof of Concept wurden diskutiert.

Für eine zukünftige Weiterentwicklung stehen viele Optionen offen. Zum einen werden laufend neue Frameworks entwickelt und möglicherweise eröffnen sich durch diese neue Möglichkeiten. Zum anderen existiert die Weiterentwicklung von BigchainDB namens Planetmint. Diese wurde bei der Erklärung für BigchainDB angesprochen.

Auch kann in Zukunft evaluiert werden, ob die Vorteile eines zentralisierten Ledgers ausreichend hoch sind, um die Anforderungen zu ändern. Ein zentralisierter Ledger wie Amazon QLDB bietet höhere Transaktionsraten, eine bessere Unterstützung und sind einfacher in der Handhabung. Dafür ist das System zentralisiert in der Hand einer Entität.

Eine andere Option, die gut skaliert, sind Datenbanken oder Ledger, die das Sharding unterstützen.

Weiterhin gibt es die Möglichkeit, das hier entworfene System direkt zu verbessern. Durch

andere Einstellungen ist eine eventuell eine höhere Performanz möglich. Die API und Datenformate können auch betrachtet werden. Eventuell ist es hier möglich, gleiche Daten zu deduplizieren und somit an Speicherplatz zu sparen.

Literatur

[1] Alkhateeb, A., Catal, C., Kar, G. und Mishra, A. Hybrid Blockchain Platforms for the Internet of Things (IoT): A Systematic Literature Review. In: *Sensors* 22(4), 2022. ISSN: 1424-8220. DOI: 10.3390/s22041304. URL: https://www.mdpi.com/1424-8220/22/4/1304.

[2] Amazon *Amazon QLDB FAQ*. URL: https://aws.amazon.com/de/qldb/faqs/ (besucht am 25. 08. 2022).

[3] Amazon *Amazon QLDB Index*. URL: https://docs.aws.amazon.com/de_de/qldb/latest/developerguide/working.manage-indexes.html (besucht am 24. 08. 2022).

[4] Amazon *Amazon Quantum Ledger Database*. URL: https://aws.amazon.com/de/qldb/ (besucht am 31. 05. 2022).

[5] Amazon *Amazon Quantum Ledger Database Features*. URL: https://aws.amazon.com/de/qldb/features/ (besucht am 31. 05. 2022).

[6] Amazon *Aufbau von QLDB*. URL: https://d1.awsstatic.com/r2018/h/99Product-Page-Diagram_AWS-Quantum.f03953678ba33a2d1b12aee6ee530e45507e7ac9.png (besucht am 17. 09. 2022).

[7] Amazon *AWS Resiliency*. URL: https://d1.awsstatic.com/whitepapers/compliance/AWS_Operational_Resilience.pdf (besucht am 25. 08. 2022).

[8] Amazon *QLDB Kosten*. 25. Aug. 2022. URL: https://aws.amazon.com/de/qldb/pricing/.

[9] Angles, R. und Gutierrez, C. Survey of Graph Database Models. In: *ACM Comput. Surv.* 40(1), 2008. ISSN: 0360-0300. DOI: 10.1145/1322432.1322433. URL: https://doi.org/10.1145/1322432.1322433.

[10] Ansdell, V. E. *Food poisoning from marine toxins*. 2014.

[11] BigchainDB *BigchainDB Documentation*. URL: https://docs.bigchaindb.com/en/latest/ (besucht am 25. 08. 2022).

[12] BigchainDB *BigchainDB Github*. URL: https://github.com/bigchaindb/bigchaindb (besucht am 16. 08. 2022).

[13] BigchainDB *BigchainDB Treiber für Java, Python und Javascript*. URL: https://docs.bigchaindb.com/projects/server/en/latest/drivers-clients/ (besucht am 16. 08. 2022).

[14] BigchainDB *Key concepts of BigchainDB*. URL: https://www.bigchaindb.com/developers/guide/key-concepts-of-bigchaindb/ (besucht am 25. 08. 2022).

[15] BigchainDB *Performance Study: Analysis of Transaction Throughput in a BigchainDB Network*. URL: https://github.com/bigchaindb/BEPs/tree/master/23 (besucht am 25. 08. 2022).

[16] BigchainDB *Struktur von BigchainDB*. URL: https://docs.bigchaindb.com/en/latest/_images/schemaDB.png (besucht am 17. 09. 2022).

[17] BigchainDB *The Hitchhiker's Guide to BigchainDB*. URL: https://www.bigchaindb.com/developers/guide/ (besucht am 25. 08. 2022).

[18] *BigchainDB 2.0 The Blockchain Database*. Techn. Ber. BigchainDB GmbH, 2018. URL: https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.

[19] Buchman, E. Tendermint: Byzantine fault tolerance in the age of blockchains. Magisterarb. University of Guelph, 2016.

[20] Buchman, E., Kwon, J. und Milosevic, Z. *The latest gossip on BFT consensus*. 2018. DOI: 10.48550/ARXIV.1807.04938.

[21] Cason, D., Fynn, E., Milosevic, N., Milosevic, Z., Buchman, E. und Pedone, F. The design, architecture and performance of the Tendermint Blockchain Network. In: *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. 2021, S. 23–33. DOI: 10.1109/SRDS53918.2021.00012.

[22] Castro, M. und Liskov, B. Practical Byzantine Fault Tolerance and Proactive Recovery. In: *ACM Trans. Comput. Syst.* 20(4):398–461, 2002. ISSN: 0734-2071. DOI: 10.1145/571637.571640. URL: https://doi.org/10.1145/571637.571640.

[23] Chamberlin, D. Early history of SQL. In: *Annals of the History of Computing, IEEE* 34:78–82, Okt. 2012. DOI: 10.1109/MAHC.2012.61.

[24] Chaum, D. L. *Computer Systems established, maintained and trusted by mutually suspicious groups*. Electronics Research Laboratory, University of California, 1979.

[25] Chickerur, S., Goudar, A. und Kinnerkar, A. Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications. In: *Proceedings of the 2015 8th International Conference on Advanced Software Engineering Its Applications (ASEA)*. ASEA '15. USA: IEEE Computer Society, 2015, S. 41–47. ISBN: 9781467398374. DOI: 10.1109/ASEA.2015.19. URL: https://doi.org/10.1109/ASEA.2015.19.

[26] Clifton, C. und Garcie-Molina, H. The Design of a Document Database. In: *Proceedings of the ACM Conference on Document Processing Systems*. DOCPROCS '88. Santa Fe, New Mexico, USA: Association for Computing Machinery, 2000, S. 125–134. ISBN: 0897912918. DOI: 10.1145/62506.62528. URL: https://doi.org/10.1145/62506.62528.

[27] Codd, E. F. A Relational Model of Data for Large Shared Data Banks. In: *Commun. ACM* 13(6):377–387, 1970. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: https://doi.org/10.1145/362384.362685.

[28] Codd, E. F. Relational Database: A Practical Foundation for Productivity. In: *Commun. ACM* 25(2):109–117, 1982. ISSN: 0001-0782. DOI: 10.1145/358396.358400. URL: https://doi.org/10.1145/358396.358400.

[29] Davoudian, A., Chen, L. und Liu, M. A Survey on NoSQL Stores. In: *ACM Comput. Surv.* 51(2), 2018. ISSN: 0360-0300. DOI: 10.1145/3158661. URL: https://doi.org/10.1145/3158661.

[30] de Vries, A. Bitcoin's Growing Energy Problem. In: *Joule* 2(5):801–805, 2018. ISSN: 2542-4351. DOI: https://doi.org/10.1016/j.joule.2018.04.016. URL: https://www.sciencedirect.com/science/article/pii/S2542435118301776.

[31] Decker, C. und Wattenhofer, R. Information propagation in the Bitcoin network. In: *IEEE P2P 2013 Proceedings*. 2013, S. 1–10. DOI: 10.1109/P2P.2013.6688704.

[32] Dey, S., Saha, S., Singh, A. K. und McDonald-Maier, K. FoodSQRBlock: Digitizing Food Production and the Supply Chain with Blockchain and QR Code in the Cloud. In: *Sustainability* 13(6), 2021. ISSN: 2071-1050. DOI: 10.3390/su13063486. URL: https://www.mdpi.com/2071-1050/13/6/3486.

[33] Donet Donet, J. A., Pérez-Solà, C. und Herrera-Joancomartí, J. The Bitcoin P2P Network. In: *Financial Cryptography and Data Security*. Hrsg. von R. Böhme, M. Brenner, T. Moore und M. Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 87–102. ISBN: 978-3-662-44774-1.

[34] Douceur, J. J. The Sybil Attack. In: *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*. 2002. URL: https://www.microsoft.com/en-us/research/publication/the-sybil-attack/.

[35] Dwork, C., Lynch, N. und Stockmeyer, L. Consensus in the Presence of Partial Synchrony. In: *J. ACM* 35(2):288–323, 1988. ISSN: 0004-5411. DOI: 10.1145/42282.42283.

[36] e.V., I. D. A. *Introducing The Blockchain Storage Extension: Planetmint*. 18. Juli 2022. URL: https://ipdb.io/wp-content/uploads/2022/07/Introducing-The-Blockchain-Storage-Extension__-Planetmint-p-.pdf (besucht am 17. 09. 2022).

[37] e.V., I. D. A. *PlanetMint auf GitHub*. URL: https://github.com/planetmint/planetmint (besucht am 17. 09. 2022).

[38] Facebook *ReactJS*. URL: https://reactjs.org/ (besucht am 29. 09. 2022).

[39] FAO *The State of World Fisheries and Aquaculture 2020. Sustainability in action*. Rome, 2020. DOI: 10.4060/ca9229en.

[40] Fielding, R. T. und Taylor, R. N. Architectural Styles and the Design of Network-Based Software Architectures. AAI9980887. Diss. 2000. ISBN: 0599871180.

[41] Ge, Z., Loghin, D., Ooi, B. C., Ruan, P. und Wang, T. Hybrid Blockchain Database Systems: Design and Performance. In: *Proc. VLDB Endow.* 15(5):1092–1104, 2022. ISSN: 2150-8097. DOI: 10.14778/3510397.3510406. URL: https://doi.org/10.14778/3510397.3510406.

[42] George, S NoSQL–NOT ONLY SQL. In: *International Journal of Enterprise Computing and Business Systems* 2(2), 2013.

[43] Georgiadis, E. How many transactions per second can bitcoin really handle? Theoretically. In: *Cryptology ePrint Archive*, 2019.

[44] Haerder, T. und Reuter, A. Principles of Transaction-Oriented Database Recovery. In: *ACM Comput. Surv.* 15(4):287–317, Dez. 1983. ISSN: 0360-0300. DOI: 10.1145/289.291.

[45] Harris, D. J. Practical Issues in Vertical Scaling. In: *Linking and Aligning Scores and Scales*. Hrsg. von N. J. Dorans, M. Pommerich und P. W. Holland. New York, NY: Springer New York, 2007, S. 233–251. ISBN: 978-0-387-49771-6. DOI: 10.1007/978-0-387-49771-6_13. URL: https://doi.org/10.1007/978-0-387-49771-6_13.

[46] Inc., S. M. *RFC1057: RPC: Remote Procedure Call Protocol Specification Version 2*. USA, 1988.

[47] Jakobsson, M. und Juels, A. Proofs of Work and Bread Pudding Protocols(Extended Abstract. In: *Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99)*

September 20–21, 1999, Leuven, Belgium. Hrsg. von B. Preneel. Boston, MA: Springer US, 1999, S. 258–272. ISBN: 978-0-387-35568-9. DOI: 10.1007/978-0-387-35568-9_18. URL: https://doi.org/10.1007/978-0-387-35568-9_18.

[48] Jiang, Q., Lee, Y. C. und Zomaya, A. Y. The Limit of Horizontal Scaling in Public Clouds. In: *ACM Trans. Model. Perform. Eval. Comput. Syst.* 5(1), 2020. ISSN: 2376-3639. DOI: 10.1145/3373356. URL: https://doi.org/10.1145/3373356.

[49] Kamilaris, A., Fonts, A. und Prenafeta-Boldú, F. X. The rise of blockchain technology in agriculture and food supply chains. In: *Trends in Food Science & Technology* 91:640–652, 2019.

[50] Kommission, E. *VERORDNUNG (EU) 2016/679 DES EUROPÄISCHEN PARLAMENTS UND DES RATES zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung)*. 27. Apr. 2016. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32016R0679.

[51] Lai, R. und LEE Kuo Chuen, D. Chapter 7 - Blockchain – From Public to Private. In: *Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2*. Hrsg. von D. Lee Kuo Chuen und R. Deng. Academic Press, 2018, S. 145–177. ISBN: 978-0-12-812282-2. DOI: https://doi.org/10.1016/B978-0-12-812282-2.00007-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128122822000073.

[52] Lamport, L., Shostak, R. und Pease, M. The Byzantine Generals Problem. In: *ACM Trans. Program. Lang. Syst.* 4(3):382–401, 1982. ISSN: 0164-0925. DOI: 10.1145/357172.357176.

[53] *Lebensmittelabfälle in Deutschland - Baseline 2015*. Bd. 71. Thünen-Report. 2019. ISBN: 978-3-86576-198-9. DOI: 10.3220/REP1563519883000. URL: https://nbn-resolving.org/urn:nbn:de:gbv:253-201907-dn061131-9.

[54] Liu, J., Sun, X. und Song, K. A food traceability framework based on permissioned blockchain. Version 2. In: *Journal of Cybersecurity* 2:107–113, 2020. DOI: 10.32604/jcs.2020.011222.

[55] Loghin, D. The Anatomy of Blockchain Database Systems. In: *Data Engineering*:48, 2022.

[56] Microsoft *Azure Confidential Ledger*. URL: https://azure.microsoft.com/en-us/services/azure-confidential-ledger/#overview (besucht am 24. 08. 2022).

[57] Microsoft *Azure VM Preisrechner*. URL: https://azure.microsoft.com/de-de/pricing/calculator (besucht am 26. 08. 2022).

[58] Mohammed, A. H., Abdulateef, A. A. und Abdulateef, I. A. Hyperledger, Ethereum and Blockchain Technology: A Short Overview. In: *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. 2021, S. 1–6. DOI: 10.1109/HORA52670.2021.9461294.

[59] *MongoDB*. URL: https://www.mongodb.com/ (besucht am 16. 08. 2022).

[60] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. In: *Cryptography Mailing list at https://metzdowd.com*, März 2009.

[61] Pallets *Flask*. URL: https://flask.palletsprojects.com/en/2.2.x/ (besucht am 29. 09. 2022).

[62] Polge, J., Robert, J. und Le Traon, Y. Permissioned blockchain frameworks in the industry: A comparison. In: *ICT Express* 7(2):229–233, 2021. ISSN: 2405-9595.

DOI: https://doi.org/10.1016/j.icte.2020.09.002. URL: https://www.sciencedirect.com/science/article/pii/S2405959520301909.

[63] Saleh, F. Blockchain Without Waste: Proof-of-Stake. In: *SSRN Electronic Journal*, Jan. 2018. DOI: 10.2139/ssrn.3183935.

[64] Shahsavari, Y., Zhang, K. und Talhi, C. A Theoretical Model for Fork Analysis in the Bitcoin Network. In: *2019 IEEE International Conference on Blockchain (Blockchain)*. 2019, S. 237–244. DOI: 10.1109/Blockchain.2019.00038.

[65] Shirinbab, S., Lundberg, L. und Casalicchio, E. Performance comparision between scaling of virtual machines and containers using cassandra nosql database. In: *Cloud Computing* 2019:103, 2019.

[66] Tendermint *Tendermint: Consensus without Mining*. Techn. Ber. 2014. URL: https://tendermint.com/static/docs/tendermint.pdf.

[67] Thume, M., Lange, J., Unkel, M., Prange, A. und Schürmeyer, M. *Blockchain-based traceability in the food industry: requirements analysis along the food supply chain*. OSF Preprints uyb64. Center for Open Science, 2021. URL: https://EconPapers.repec.org/RePEc:osf:osfxxx:uyb64.

[68] Thurlow, R. *RFC5531: RPC: Remote Procedure Call Protocol Specification Version 2*. USA, 2009.

[69] Tian, F., Taudes, A. und Mendling, J. A Supply Chain Traceability System for Food Safety Based on HACCP, Blockchain & Internet of Things. In: *AN INFORMATION SYSTEM FOR FOOD SAFETY MONITORING IN SUPPLY CHAINS BASED ON HACCP, BLOCKCHAIN AND INTERNET OF THINGS*:83.

[70] Visa *Visa Fact Sheet*. URL: https://www.visa.co.uk/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf (besucht am 24. 08. 2022).

[71] W3C *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 27. Apr. 2007. URL: https://www.w3.org/TR/soap12/ (besucht am 07. 09. 2022).

[72] White, J. E. *RFC0707: High-Level Framework for Network-Based Resource Sharing*. USA, 1975.

[73] Yang, D., Long, C., Xu, H. und Peng, S. A Review on Scalability of Blockchain. In: *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*. ICBCT'20. Hilo, HI, USA: Association for Computing Machinery, 2020, S. 1–6. ISBN: 9781450377676. DOI: 10.1145/3390566.3391665.

[74] Yang, X., Zhang, Y., Wang, S., Yu, B., Li, F., Li, Y. und Yan, W. LedgerDB: A Centralized Ledger Database for Universal Audit and Verification. In: *Proc. VLDB Endow.* 13(12):3138–3151, 2020. ISSN: 2150-8097. DOI: 10.14778/3415478.3415540. URL: https://doi.org/10.14778/3415478.3415540.