



Fachbereich Informatik

## Bachelorarbeit

# Prototypische Implementierung eines Preisvergleichs für Kryptowährungen auf unterschiedlichen Exchanges zur Identifikation von Handelspotentialen

Verfasser:

Marvin Ludwig

Jahnstraße 2A

26935 Stadland

Studiengang: Informatik

Matrikelnummer: 909130

Abgabetermin: 30. November 2023

Prüfer:

Sebastian Lothary, M. Sc.

# Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>iv</b>
<b>Abkürzungsverzeichnis</b>	<b>v</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problembeschreibung . . . . .	2
1.3 Forschungsfrage . . . . .	3
1.4 Forschungsmethode- und ziele . . . . .	3
1.5 Vorgehensweise und Aufbau . . . . .	5
<b>2 Stand der Technik und Wissenschaft</b>	<b>7</b>
2.1 Digitale Vermögenswerte . . . . .	7
2.1.1 FIAT Währung . . . . .	8
2.1.2 Kryptowährungen . . . . .	8
2.1.3 Transaktionskosten . . . . .	9
2.1.4 Handelsgröße . . . . .	10
2.2 Exchange . . . . .	10
2.2.1 Kraken . . . . .	11
2.2.2 Bitstamp . . . . .	12
2.3 Datenbanken . . . . .	13
2.4 Verteilte Datenbank . . . . .	15
2.5 Blockchain . . . . .	16
2.6 Application Programming Interface . . . . .	17
2.7 CryptoCurrency eXchange Trading Library . . . . .	17
2.8 Python . . . . .	18
2.9 Nachrichten Broker . . . . .	18
2.10 Go . . . . .	19
2.11 Zeitreihendatenbank . . . . .	20
2.12 In-Memory-Datenbank . . . . .	20

2.13 Frontend . . . . .	21
2.14 Next.js . . . . .	21
2.15 Websocket . . . . .	22
2.16 Webserver . . . . .	22
2.17 Docker . . . . .	23
<b>3 Konzeptionelle Modellierung und Design</b>	<b>25</b>
3.1 Konzeptionelles Anwendungsfallmodell . . . . .	25
3.2 Konzeptionelles Informationsmodell . . . . .	29
3.3 Konzeptionelles Architekturmodell . . . . .	33
<b>4 Prototypische Implementierung</b>	<b>36</b>
4.1 Die Datenbanken und den Nachrichten Broker hochfahren . . . . .	36
4.2 Der Stream Service . . . . .	38
4.2.1 Konfigurationsdaten laden und validieren . . . . .	39
4.2.2 Verbindung zur RabbitMQ herstellen . . . . .	39
4.2.3 Ticker-Informationen abrufen und in die Queue schreiben . . . . .	41
4.3 Der Storage-Service . . . . .	41
4.3.1 Konfigurationsdaten laden und eine Verbindung zu RabbitMq und InfluxDB herstellen . . . . .	42
4.3.2 Golang-Routine für das Lesen und Speichern von Daten . . . . .	42
4.4 Der Websocket . . . . .	43
4.4.1 Server Instanz . . . . .	44
4.4.2 Websocket Funktionen . . . . .	44
4.4.3 Einrichtung und Hochfahren des WebSocket-Servers . . . . .	46
4.4.4 Illustration der WebSocket-Funktionsweise . . . . .	46
4.5 Der Refresher Service . . . . .	47
4.5.1 Den JobController definieren . . . . .	48
4.5.2 Datenabfrage und -verarbeitung . . . . .	48
4.5.3 Hauptausführung und Initialisierung . . . . .	50
4.6 Die Application Programming Interface (API) Schnittstelle . . . . .	51
4.6.1 Initialisierung und hochfahren der API . . . . .	51
4.6.2 API-Abfrage Testen . . . . .	52
4.7 Das Frontend . . . . .	53
4.7.1 Initialisierung und Start des Next.js Servers . . . . .	53
4.7.2 Eine Kryptowährungspreis-Tabelle erstellen . . . . .	54
4.7.3 Einbindung der Daten über die API . . . . .	55
4.8 Bereitstellung der Infrastruktur . . . . .	57

<b>5 Evaluation des Prototypen</b>	<b>62</b>
5.1 Cognitive Walkthrough . . . . .	62
5.1.1 Die Startseite . . . . .	63
5.1.2 Die Detailseite . . . . .	64
5.1.3 Die Potentialen Seite . . . . .	66
5.2 Das Evaluationsergebnis . . . . .	68
5.2.1 Das Evaluationsergebnis von der Startseite . . . . .	69
5.2.2 Das Evaluationsergebnis von der Detailseite . . . . .	69
5.2.3 Das Evaluationsergebnis von der Potentialen Seite . . . . .	70
<b>6 Fazit</b>	<b>73</b>
6.1 Ergebnis . . . . .	73
6.2 Ausblick . . . . .	76
<b>Quellenverzeichnis</b>	<b>79</b>

# Abbildungsverzeichnis

1.1	Rahmenwerk von Nunamaker, Chen und Purdin . . . . .	4
2.1	Kraken Gebührenplan . . . . .	12
2.2	Bitstamp Gebührenplan . . . . .	13
3.1	Konzeptionelles Anwendungsfallmodell als UML Use Diagramm modelliert . . . . .	26
3.2	Konzeptionelles Informationsmodell als UML Klassendiagramm modelliert . . . . .	30
3.3	Konzeptionelles Architekturmodell in Anlehnung an das UML Deployment Diagramm . . . . .	34
4.1	Illustration des Websockets . . . . .	47
4.2	API Abfrage testen . . . . .	53
4.3	Basis Tabelle . . . . .	55
4.4	Final Tabelle . . . . .	56
5.1	Darstellung der Startseite . . . . .	63
5.2	Hell-Dunkel Modus auswählen . . . . .	64
5.3	Dunkel-Modus Darstellung . . . . .	64
5.4	Die Detailseite von BTC/USD . . . . .	65
5.5	Kryptobörsen auf der Detailseite auswählen . . . . .	65
5.6	Die Chart analyse . . . . .	66
5.7	Die Potentialen Seite . . . . .	67
5.8	Potentiellen Handel berechnen . . . . .	67
5.9	Gewinn des potentiellen Handel berechnen . . . . .	68

# Abkürzungsverzeichnis

<b>API</b>	Application Programming Interface (siehe S. 7, 17, 18, 22, 31, 35, 47, 50–53, 55, 56, 60, 73)
<b>CCXT</b>	CryptoCurrency eXchange Trading Library (siehe S. 17–19, 24, 31, 41, 73, 76)
<b>CLI</b>	Command Line Interface (siehe S. 53)
<b>CSR</b>	Client Side Rendering (siehe S. 21)
<b>DB</b>	Datenbank (siehe S. 7, 14, 15, 20, 24, 34, 37, 42)
<b>DBMS</b>	Datenbankmanagementsystem (siehe S. 15, 20)
<b>DBS</b>	Datenbanksystem (siehe S. 15)
<b>FF</b>	Forschungsfrage (siehe S. 3, 5, 29, 75)
<b>FZ</b>	Forschungsziel (siehe S. 3, 5–7, 25, 35, 36, 61, 62, 71, 75)
<b>HTTP</b>	Hypertext Transfer Protocol (siehe S. 44)
<b>IMDB</b>	In-Memory-Datenbank (siehe S. 20, 21, 32, 35, 36, 74)
<b>ISR</b>	Incremental Static Regeneration (siehe S. 21)
<b>PB</b>	Problembereich (siehe S. 2, 3)
<b>SSG</b>	Static Site Generation (siehe S. 21)
<b>SSR</b>	Server Side Rendering (siehe S. 21)
<b>TSDB</b>	Zeitreihendatenbank (siehe S. 20, 31, 32, 34, 35, 73)
<b>VDB</b>	Verteilte Datenbank (siehe S. 15, 16, 24)

# Kapitel 1

## Einleitung

In den letzten Jahrzehnten hat sich das Geldsystem ständig weiterentwickelt, von Münzen über Banknoten hin zu digitalen Geldtransaktionen. Mit der Entstehung von Kryptowährungen ist eine neue Möglichkeit der finanziellen Transaktionen eingeläutet worden. Alles begann mit Satoshi Nakamoto [1], der 2008 das Bitcoin-Whitepaper [1] veröffentlichte und damit den Weg für die erste Kryptowährung ebnete. Seitdem hat sich die Kryptowährungslandschaft rasant entwickelt. Es sind viele neue Kryptowährungen entstanden, darunter Ethereum, Litecoin und viele weitere sogenannter Altcoins<sup>1</sup>. Mit dem explosionsartigen Wachstum der Kryptowährungen sind auch unzählige Exchanges<sup>2</sup> entstanden, die den Handel mit Kryptowährungen ermöglichen (vgl. [2]). Mit dieser Vielfalt an Kryptowährungen und Exchanges eröffnen sich viele Möglichkeiten Handel zu betreiben. Aufgrund der Menge an Kryptowährungen und Exchanges wurde es jedoch für Benutzer zunehmend komplexer und zeitaufwendiger die unterschiedlichen Preise zu vergleichen.

### 1.1 Motivation

Mit der großen Anzahl an Kryptowährungen und Exchanges ergibt sich ein breites Spektrum an Handelsmöglichkeiten. Für die Anleger ist das manuelle Überprüfen von Preisen auf den unterschiedlichen Exchanges zeitaufwendig und wird zudem durch die Volatilität des Marktes noch erschwert. Dadurch wird es zunehmend komplexer, einen Überblick zu erhalten und sinnvolle Handelsentscheidungen zu treffen.

Die Forschung kann hier eine wichtige Rolle spielen, indem sie Systeme und Technologien schafft, die den Handel mit Kryptowährungen effizienter und weniger komplex

---

<sup>1</sup> Altcoins sind alternative Kryptowährungen zu Bitcoin.

<sup>2</sup> Exchanges werden auch Kryptobörsen genannt.

gestalten. Die Automatisierung eines Preisvergleichsprozesses ist hier ein wichtiger Ansatzpunkt. Technologien wie Webanwendungen und automatisierte Systeme können genutzt werden, um Kryptowährungsinformationen schneller und effektiver zu sammeln und zu analysieren. Dadurch können Anleger fundierte Entscheidungen treffen und potentielle Handelsstrategien effektiv bewerten.

Die Entwicklung einer solchen Lösung soll dazu beitragen, das Verständnis und die Nutzung von Kryptowährungen zu verbessern. Die Entwicklung soll zeigen, wie komplexe Prozesse mithilfe der Möglichkeiten der Digitalisierung und Automatisierung vereinfacht darstellt werden können.

## 1.2 Problembeschreibung

Der Kryptomarkt, der aus mehreren Exchanges besteht, ist sowohl vielfältig als auch komplex. Dies zeigt sich unter anderem darin, dass die gleiche Kryptowährung auf verschiedenen Exchanges zu unterschiedlichen Preisen gehandelt wird (vgl. [2]). Auf den ersten Blick scheint es also eine einfache Möglichkeit für profitablen Handel zu geben, indem man günstige Kryptowährung an einem Exchange kauft und sie an einem anderen Exchange zu einem höheren Preis verkauft. Doch diese scheinbar einfache Strategie wird durch die unterschiedlichen Kostenstrukturen der jeweiligen Exchanges stark erschwert. Durch diese Intransparenz auf dem Markt wird es Händlern erschwert, fundierte Entscheidungen zu treffen. Hinzu kommt die hohe Dynamik des Kryptomarktes, die durch schnell veränderte Preise der Kryptowährungen entstehen. Aufgrund dessen ist es für Händler praktisch unmöglich, alle Preise und Kosten manuell zu überprüfen.

Aus dieser Problembeschreibung wird der Problembereich 1 (PB1) definiert: *Die Intransparenz der Kostenstrukturen auf dem Kryptomarkt und die daraus resultierenden Herausforderungen bei der Identifizierung profitabler Handelsmöglichkeiten.*

Ein weiteres Problem stellen die Laufzeiten der Kryptowährungen für den Transfer von einem Exchange zum anderen dar. Dieser Faktor kann Einfluss auf die Rentabilität der Handelsstrategie haben. In dieser Arbeit werden sie jedoch nicht berücksichtigt, da dies zu analysieren den Rahmen der Arbeit sprengen würde.

Schließlich soll in dieser Arbeit stets davon ausgegangen werden, dass Kryptowährungen zum aktuellen, am Markt angegebenen Preis gekauft und verkauft werden können. Dies ist eine wichtige Vereinfachung, die es ermöglicht, den Fokus auf die Analyse der Kostenstrukturen zu legen. In der Realität kann jedoch auch dies eine Herausfor-

derung darstellen, insbesondere bei Kryptowährungen mit geringem Handelsvolumen oder hoher Volatilität.

## 1.3 Forschungsfrage

Aus dem in der Problembeschreibung definierten PB1 ergibt sich die zentrale Forschungsfrage 1 (FF1):

*Wie kann ein automatisiertes System entworfen werden, das die unterschiedlichen Kostenstrukturen auf den Exchanges berücksichtigt, um eine profitable Handelsmöglichkeit darzustellen?*

Die Beantwortung von FF1 wird dazu beitragen, die Intransparenz der Kostenstrukturen auf dem Kryptomarkt zu verringern und ermöglicht es Anlegern, fundierte Entscheidungen auf der Grundlage genauer und aktueller Informationen zu treffen. Die zentrale Herausforderung besteht darin, eine Möglichkeit zu schaffen, die Preise der verschiedenen Exchanges effizient zu vergleichen, dabei die Kostenstrukturen der Exchanges zu berücksichtigen und das Resultat übersichtlich darzustellen. Darüber hinaus sollte diese Methode in der Lage sein, mit der hohen Dynamik des Kryptomarktes umzugehen und stets aktuelle Informationen zu liefern.

## 1.4 Forschungsmethode- und ziele

In diesem Kapitel werden die Forschungsmethoden und die daraus abgeleiteten Forschungsziele (FZ) vorgestellt, welche zur Beantwortung der FF1 eingesetzt werden. Die ausgewählten Forschungsmethoden orientieren sich am Modell „Systems Development in Information Systems Research“ von Nunamaker, Chen und Purdin [3]. Aus diesem Modell leiten sich vier spezifische FZ ab, die darauf ausgerichtet sind, die FF1 zu beantworten.

### *Forschungsmethoden*

Folgende vier Hauptforschungsmethoden werden verwendet: Beobachtung, Theorie Aufbau, Systementwicklung und Experimente. Diese Methoden sind miteinander verbunden und werden iterativ im Verlauf der Arbeit verwendet, um eine Lösung für das in der Forschungsfrage identifizierte Problem zu finden.

- 1. Beobachtung [O]:** In der ersten Phase (engl. Observation) werden die Kenntnisse in einem noch relativ unbekannten Forschungsgebiet erlangt.

2. **Theorie Aufbau [T]:** In der zweiten Phase, dem Theorie Aufbau (engl. Theory Building), liegt der Fokus auf der Entwicklung neuer Ideen und Konzepte. In diesem Schritt werden konzeptionelle Rahmenbedingungen festgelegt und neue Methoden entwickelt, die das in der Beobachtungsphase aufgebaute Wissen nutzen.
3. **Systementwicklung [S]:** Die dritte Phase, die Systementwicklung (engl. System Development), beinhaltet den Übergang von Theorie zur Praxis. Hier wird ein Prototyp des Systems entwickelt, um die Machbarkeit der geplanten Lösungen zu demonstrieren. Erfüllt der Prototyp die Erwartungen, wird das System mit vollem Funktionsumfang entwickelt und in eine reale Umgebung vorbereitet.
4. **Experimente [E]:** Die vierte und letzte Phase ist das Experimentieren (engl. Experimentation). In dieser Phase wird das entwickelte System validiert und evaluiert. Die Ergebnisse dieser Experimente werden zur Verbesserung des Systems verwendet.

In Abbildung 1.1 wird das Rahmenwerk von Nunamaker, Chen und Purdin [3] dargestellt, welches die Forschungsmethodik und die Verknüpfungen der einzelnen Phasen des Forschungsprozesses visualisiert.

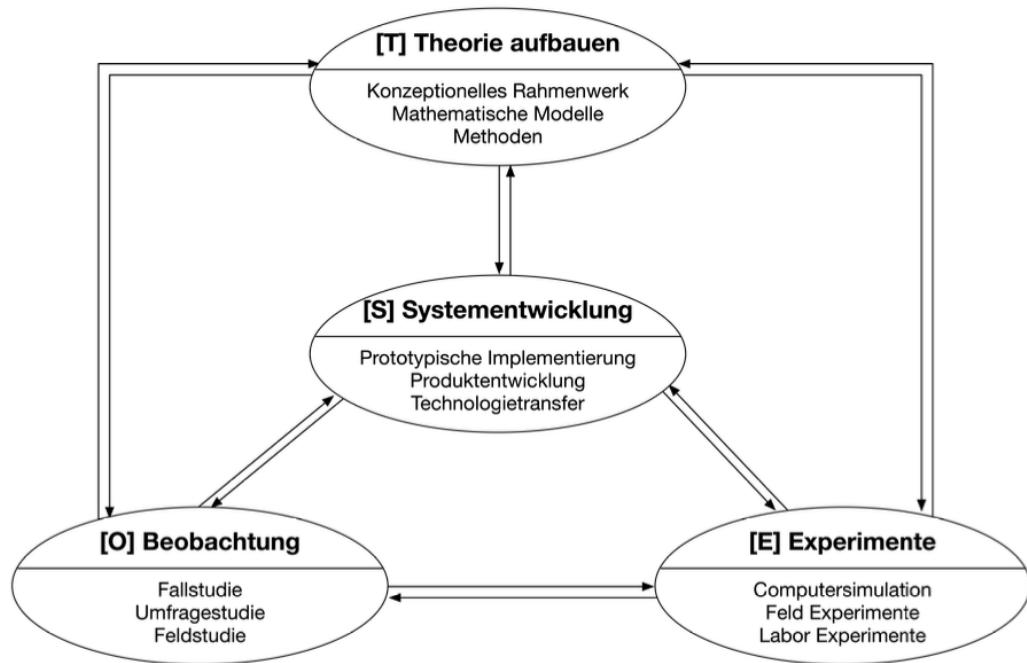


Abbildung 1.1: Rahmenwerk von Nunamaker, Chen und Purdin (aus [3, Abbildung 2  
Übersetzt ins Deutsche])

Jede dieser Phasen steht nicht für sich allein, sondern ist Teil eines iterativen Prozes-

ses, der durch die Pfeile im Modell von Nunamaker, Chen und Purdin [3] dargestellt wird. Die Pfeile repräsentieren die Möglichkeit, vom aktuellen Stand der Forschung zu einem früheren Punkt zurückzukehren, um das gewonnene Wissen zu verfeinern und anzupassen. So kann zum Beispiel nach dem Experimentieren eine Rückkehr zur Theorie Aufbau-Phase erforderlich sein, um die Theorien zu modifizieren oder zu ergänzen. Ebenso kann nach der Systementwicklung zurück zur Beobachtungsphase gegangen werden, um neue Daten zu sammeln und das System weiter zu verbessern. Dieser iterative Ansatz ermöglicht eine ständige Anpassung und Verbesserung des Forschungsprozesses.

#### *Forschungsziele*

1. **FZ1O - Identifikation geeigneter Konzepte, Methoden und Technologien:** Als erstes FZ wird FZ1O definiert, dass durch die Recherche geeignete Konzepte, Methoden und Technologien identifiziert, um ein automatisiertes System zu entwickeln. Dieses System soll in der Lage sein, die unterschiedlichen Preise von Kryptowährungen effizient zu erfassen und zu vergleichen. Das Ziel legt mit der Identifikation den Grundstein für die Konzeptionierung eines Tools.
2. **FZ2T - Entwicklung eines Konzepts:** Das FZ2T konzentriert sich auf die Erstellung eines Konzepts, das die in FZ1O identifizierten Konzepte, Methoden und Technologien verwendet. Dieses Konzept bildet das Design für die Entwicklung des Systems.
3. **FZ3S - Erstellung eines Prototypen:** Im FZ3S soll ein prototypisches System erstellt werden, das das entwickelte Konzept implementiert. Dieser Prototyp soll die Machbarkeit der entwickelten Konzepte und Methoden demonstrieren.
4. **FZ4E - Evaluation der Implementierung:** Bei FZ4E handelt es sich um die Evaluation der prototypischen Implementierung. Dabei wird das System auf seine Effektivität hin überprüft, indem seine Leistung evaluiert wird. Basierend auf den Ergebnissen dieser Evaluation sollen dann mögliche Verbesserungen des Systems identifiziert werden.

## 1.5 Vorgehensweise und Aufbau

Diese Bachelorarbeit ist nach dem methodischen Ansatz von Nunamaker, Chen und Purdin [3] strukturiert. Die folgenden Kapitel geben einen Überblick über den Aufbau und den Ansatz der Arbeit, wobei sie die zentrale FF1 in vier Kernbereiche unterteilen und sich auf die Erreichung der in Abschnitt 1.4 definierten FZ konzentrieren.

In Kapitel 2 „Stand der Wissenschaft und Technik“ wird die Beobachtungsphase dargestellt und *FZ1O* durchgeführt. In diesem Abschnitt wird der Stand der Technik und Wissenschaft in der Kryptowährungsbranche sowie die verschiedenen Technologien und Methoden, die für die Implementierung und Durchführung erforderlich sind, recherchiert und aufgezeigt. Die gewonnenen Erkenntnisse bilden die Grundlage für die nachfolgenden Forschungsphasen.

In Kapitel 3 „Konzeptionelle Modellierung“ wird die Theorie Aufbau-phase durchgeführt, um *FZ1T* zu erfüllen. In diesem Kapitel wird eine Strategie basierend auf den in Kapitel 2 erzielten Ergebnissen entwickelt. Es werden die Konzepte, Methoden und Technologien definiert und zusammengeführt, die zur Entwicklung der Anwendung und der automatisierten Handlungsempfehlungen verwendet werden.

Kapitel 4 „Prototypische Implementierung“, repräsentiert die Systementwicklungsphase und dient der Umsetzung des *FZ1S*. In diesem Abschnitt wird ein Prototyp des Systems entwickelt, um die Machbarkeit der in den vorherigen Kapiteln erarbeiteten Konzepte und Theorien zu demonstrieren.

In Kapitel 5 „Evaluation“ wird die Experimentierphase durchgeführt, um *FZ1E* zu erreichen. Das entwickelte System wird in dieser Phase validiert und evaluiert, um dessen Effektivität zu überprüfen. Die Ergebnisse der Evaluation werden dazu verwendet, Verbesserungen am System zu identifizieren.

Die Arbeit schließt ab mit Kapitel 6 „Fazit und Ausblick“. In diesem letzten Kapitel werden die Ergebnisse zusammengefasst und ein Ausblick gegeben.

# Kapitel 2

## Stand der Technik und Wissenschaft

In diesem Kapitel wird das FZ1O bearbeitet, indem geeignete Konzepte, Methoden und Technologien erforscht werden. Diese sind notwendig für die Entwicklung eines automatisierten Systems zur Erfassung und zum Vergleich der unterschiedlichen Preise von Kryptowährungen.

Die Fortschritte in der Technik und Wissenschaft haben die Entwicklung und Implementierung innovativer Systeme, wie Kryptowährungen ermöglicht. Diese Form digitaler Vermögenswerte hat durch die Exchanges eine weitreichende Verfügbarkeit und Bekanntheit erlangt. Im Zentrum der Informationsspeicherung dieser digitalen Währungen steht die Blockchain-Technologie – eine verteilte, dezentralisierte Datenbank (DB), die durch ihre Transparenz und Sicherheit revolutioniert hat.

Für das effektive Agieren auf dem Markt der Kryptowährungen ist es unerlässlich, ein Konzept zu entwerfen, das nicht nur das Sammeln und Speichern, sondern auch das leicht zugängliche Bereitstellen von Handelsdaten ermöglicht. An dieser Stelle kommen Application Programming Interfaces (APIs) ins Spiel, die es erlauben, relevante Informationen direkt von den Exchanges abzurufen. Programmiersprachen wie Python [4], Golang [5] und TypeScript [6] werden dabei verwendet, um die Entwicklung einer Webanwendung zu ermöglichen, welche die gesamte Applikation integriert und über einen Webserver der Öffentlichkeit im Internet zugänglich macht.

### 2.1 Digitale Vermögenswerte

Nach Schlaffer, Schmeing und Kerber [7] sind Digitale Vermögenswerte eine Kategorie von Vermögenswerten, die digital existieren und daher keinen physischen Gegenstand darstellen. Ermöglicht werden digitale Vermögenswerte durch verteilte Technologie, wie die Blockchain. Sie stellen eine neuartige Form von Werten dar, die weder von

einer Zentralbank noch von einer öffentlichen Institution ausgegeben oder garantiert werden und nicht den gesetzlichen Status einer Währung oder eines Geldes besitzen. Digitale Vermögenswerte können als Tauschmittel, Zahlungsmittel oder zur Investition verwendet werden. Sie sind digital übertragbar, speicherbar und handelbar (vgl. [7]). Ein besonderer Typ von digitalen Vermögenswerten sind Kryptowährungen. Um jedoch ihren Kontext und ihre Besonderheiten vollständig zu verstehen, wird im Unterabschnitt 2.1.1 zuerst der Begriff der Fiat-Währung erklärt. Anschließend folgt im Unterabschnitt 2.1.2 die Erklärung der Kryptowährungen. Zuletzt wird im Unterabschnitt 2.1.3 auf die Transaktionskosten und im Unterabschnitt 2.1.4 auf die Handelsgröße eingegangen, die für den Handel von entscheidener Bedeutung sind, da sie die Profitabilität von Trades beeinflussen.

### **2.1.1 FIAT Währung**

In der Praxis ist Fiatgeld das alltägliche physische Zahlungsmittel, womit wir unsere täglichen Transaktionen wie den Einkauf im Supermarkt oder das Bezahlen einer Dienstleistung vollziehen. Die Zentralbank [8] definiert Fiat-Währung, oder Fiatgeld, als ein von staatlichen Institutionen festgelegtes und kontrolliertes Zahlungsmittel, das im Unterschied zu Warengeld, nicht an den Preis eines Rohstoffes wie Gold oder Silber gebunden ist. Diese Unabhängigkeit ermöglicht es Zentralbanken, das Geldwesen stabiler zu steuern, da sie nicht von Preisschwankungen bestimmter Rohstoffe betroffen sind. Außerdem wird Fiatgeld international akzeptiert, was den Welthandel vereinfacht. Allerdings ist der Wert des Fiatgeldes stark von der wirtschaftspolitischen Führung eines Landes abhängig. Schlecht geführte Wirtschaftspolitiken können zu Inflation oder sogar Hyperinflation führen, wie Nagel [9, S.139, 140] diskutiert. Während Fiatgeld in Form von Münzen und Banknoten in unseren Geldbörsen greifbar sind, existieren Kryptowährung ausschließlich in digitaler Form.

### **2.1.2 Kryptowährungen**

Kryptowährungen stellen eine besondere Form digitaler Vermögenswerte dar. Sie können privat durch Verschlüsselungstechniken erzeugt werden und sind dabei unabhängig von staatlichen Institutionen.

Nagel [9, S. 17] hebt hervor, dass Kryptowährungen im Gegensatz zu Fiatgeld keine staatliche Zentralbank haben und weder auf gesetzlichen Grundlagen noch auf staatlichen Regulierungen basieren. Der Wert von Kryptowährungen wird lediglich durch Angebot und Nachfrage bestimmt, was sie äußerst volatil macht, wie Behringer und Follert [10] anmerken. Diese Volatilität, kombiniert mit dem Handel auf verschiede-

nen Exchanges, führt oft zu Preisunterschieden und schafft Handelsmöglichkeiten, die den Bedarf für einen Preisvergleich zwischen den verschiedenen Exchanges unterstreichen.

Die bekannteste und am weitesten verbreitete Kryptowährung ist der Bitcoin, dessen Konzept 2008 zum ersten Mal von Nakamoto [1] veröffentlicht wurde. Allerdings gibt es noch viele weitere Kryptowährungen, die alle durch eine hohe Volatilität gekennzeichnet sind. Hönig [11, S. 34] hat festgestellt, dass die hohe Volatilität von Kryptowährungen ihre Alltagstauglichkeit als Zahlungsmittel einschränkt, sie jedoch gleichzeitig zu hochriskanten und interessanten Anlageobjekten macht.

Ebenso relevant in der Praxis des Handels mit Kryptowährungen sind die dabei entstehenden Kosten, insbesondere die Transaktionskosten. Diese können das Handelspotential signifikant beeinflussen und sollten daher in jeder Überlegung bezüglich Handelsstrategien für Kryptowährungen berücksichtigt werden.

### 2.1.3 Transaktionskosten

Geroldinger [12, S. 8] definiert Transaktionskosten als jene Kosten, die direkt oder indirekt mit einer Transaktion verbunden sind. Im Kontext von Kryptowährungen unterscheiden man zwischen On-Chain und Off-Chain Transaktionskosten. On-Chain Transaktionskosten entstehen bei dezentralisierten Transaktionen, die direkt in der Blockchain durchgeführt werden, wie beispielsweise der Transfer einer Kryptowährung von einem Exchange zum anderen. Off-Chain Transaktionskosten hingegen entstehen bei Transaktionen, die außerhalb der Blockchain durchgeführt werden, zum Beispiel wenn ein Benutzer eine Kryptowährung innerhalb eines Exchanges kauft und die Transaktionsinformation lediglich in dem Exchange erfasst wird (vgl. [12, S. 28]). Darüber hinaus fallen beim Handel mit Kryptowährungen weitere Arten von Gebühren an, einschließlich:

1. **Maker- und Taker-Gebühren:** Maker-Gebühr sind Kosten, die beim Platzieren einer Order anfallen, die das Orderbuch des Exchanges erweitert (z. B. eine nicht sofort durchführbare Limit-Order), während eine Taker-Gebühr beim Platzieren einer Order entsteht, die eine bestehende Order aus dem Orderbuch ausführt (z. B. eine Market-Order, die sofort ausgeführt wird). Die entsprechende Höhe der Gebühr unterscheidet sich zwischen den Exchanges und richtet sich nach dem effektiven Handelsvolumen der letzten 30 Tage, bezogen auf das Tagesdatum.
2. **Deposit und Withdrawal Fees:** Dies sind Gebühren, die beim Ein- und Auszahlen von Geldern oder auch Kryptowährung aus dem Börsenkonto anfallen.

Diese Gebühren variieren je nach der Börse und der Zahlungsmethode (z.B. Fiatgeld mit Sepa oder Bitcoin über die Blockchain) (vgl. [12, S. 8-11, 45–47]).

Neben den Transaktionskosten spielt auch die Dimensionierung eines Trades, bekannt als Handelsgröße, eine zentrale Rolle im Kryptohandel. Diese bestimmt nicht nur das finanzielle Risiko, sondern kann auch die Rentabilität einer Handelsempfehlung beeinflussen.

### 2.1.4 Handelsgröße

Die Handelsgröße ist die Menge einer bestimmten Kryptowährung, die in einem Exchange gehandelt wird. Diese kann abhängig vom Preis der jeweiligen Kryptowährung stark variieren. Zum Beispiel könnte bei Bitcoin, mit einem aktuellen Preis von etwa 26.800 € pro Bitcoin<sup>3</sup>, wenige Trades im Bereich von 0,x Bitcoin ausreichen, um Gewinne zu erzielen. Im Gegensatz dazu könnten bei Kryptowährungen mit einem geringeren Wert Tausende von Coins benötigt werden, um rentabel zu sein. Es ist daher wichtig, die Handelsgröße im Kontext des spezifischen Marktwertes einer Kryptowährung zu betrachten, da dies die Rentabilität des Handels beeinflussen. Natürlich hängt die Effektivität solcher Trades nicht nur von der Größe oder Transaktionskosten ab, sondern auch von der Plattform, auf der sie durchgeführt werden: den Exchanges.

## 2.2 Exchange

Ein Exchange ist eine digitale Handelsplattform, auf der Nutzer Kryptowährungen kaufen, verkaufen und tauschen können. Ähnlich wie klassische Online-Börsen bieten auch Exchanges 24-Stunden Handelsmöglichkeiten an. Die Auswahl eines vertrauenswürdigen Exchange ist essentiell, da dort die Kryptowährungen aufbewahrt werden. In der Vergangenheit haben bereits einige Betreiber von Exchanges das Vertrauen ihrer Kunden missbraucht. Dies führte zu bedeutenden Verlusten – bekannte Beispiele hierfür sind Mt Gox [13] und FTX [14].

Bevor Anleger nun auf diesen Exchanges handeln können, benötigen sie ein digitales Wallet<sup>4</sup>, um ihre Kryptowährungen sicher aufzubewahren. Die Anmeldung für einen Exchange erfolgt normalerweise durch die Erstellung eines Kontos mit einer E-Mail Adresse und Passwort (vgl. [11, S. 65]).

Exchanges gelten im Vergleich zu klassischen Online-Börsen oft als weniger benutzerfreundlich. Dies kann an der großen Vielfalt der angebotenen Funktionen liegen und

---

<sup>3</sup> Entnommen am 23.07.23 von <https://www.finanzen.net/devisen/bitcoin-euro-kurs>

<sup>4</sup> Ein digitales Wallet ist ein Portemonnaie, welches nur online besteht (vgl. [11, S. 41]).

an der Tatsache, dass der Sitz vieler Exchanges oftmals im Ausland liegt. Das hat zur Folge, dass der Kundenservice nicht immer erreichbar ist und die Handelsoberfläche nicht auf Deutsch verfügbar ist (vgl. [15]).

Nachfolgend werden im Unterabschnitt 2.2.1 Kraken [16] und im Unterabschnitt 2.2.2 Bitstamp [17] vorgestellt, welche zwei bekannte Exchanges sind, auf die sich im weiteren Verlauf dieser Arbeit spezialisiert wird.

### 2.2.1 Kraken

Kraken [16] ist eine der größten Exchanges der Welt<sup>5</sup>. Gegründet 2011 mit Sitz in den USA ist sie bekannt für ihr hohes Handelsvolumen zwischen Digitalwährungen und dem Euro (vgl. [11, S.66]). Um ihren Benutzern eine sichere Handelsumgebung zu bieten, legt Kraken [16] einen großen Wert auf Sicherheitsstandards. Der Exchange bietet außerdem über 200 verschiedene Kryptowährungen an, darunter auch verschiedene Altcoins<sup>6</sup>. Kraken [16] akzeptiert viele Fiat-Währungen für den Kauf von Kryptowährungen, darunter Euro und US-Dollar. Im Allgemeinen ist die Nutzung von Kraken [16] kostenlos, für den Handel mit Kryptowährungen fallen jedoch Gebühren an. Kraken [16] erhebt für die Einzahlung von Kryptowährungen, bis auf die Kryptowährung MINA, keine Einzahlungsgebühren<sup>7</sup>, jedoch fallen Auszahlungsgebühren an, die je nach Kryptowährung variieren können<sup>8</sup>. Darüber hinaus werden unterschiedliche Maker- und Taker-Gebühren berechnet, wie in Abbildung 2.1 dargestellt.

---

<sup>5</sup> <https://coinmarketcap.com>

<sup>6</sup> Altcoins sind alternative Kryptowährungen zu Bitcoin

<sup>7</sup> Stand 16.09.2023, entnommen von <https://support.kraken.com/hc/en-us/articles/360000292886-Cryptocurrency-deposit-fees-and-minimums>

<sup>8</sup> Stand 16.09.2023, entnommen von <https://support.kraken.com/hc/en-us/articles/360000767986-Cryptocurrency-withdrawal-fees-and-minimums>

30-Day Volume (USD)	Maker	Taker
\$0 - \$50,000	0.16%	0.26%
\$50,001 - \$100,000	0.14%	0.24%
\$100,001 - \$250,000	0.12%	0.22%
\$250,001 - \$500,000	0.10%	0.20%
\$500,001 - \$1,000,000	0.08%	0.18%
\$1,000,001 - \$2,500,000	0.06%	0.16%
\$2,500,001 - \$5,000,000	0.04%	0.14%
\$5,000,001 - \$10,000,000	0.02%	0.12%
\$10,000,000 - \$100,000,000	0.00%	0.10%
\$100,000,000 - \$250,000,000 **	0.00%	0.08%
\$250,000,000 - \$500,000,000 **	0.00%	0.06%
\$500,000,000+ **	0.00%	0.04%

Abbildung 2.1: Kraken [16] Gebührenplan<sup>9</sup>

Nach der Vorstellung von Kraken [16] folgt nun die Betrachtung von Bitstamp [17].

## 2.2.2 Bitstamp

Auch Bitstamp [17] ist einer der führenden Exchanges weltweit [2]. Bitstamp [17] hebt sich von vielen anderen Plattformen dadurch ab, dass es der erste Exchange war, der eine staatliche Lizenz erhalten hatte. Ursprünglich in Slowenien anlässig, verlegte Bitstamp [17], nach Erhalt der EU-Lizenz als Zahlungsinstitut im Jahr 2016, seinen Sitz nach Luxemburg (Vgl. [18]). Durch diese Lizenz hat Bitstamp die Möglichkeit, seine Handelsdienste legal in der gesamten Europäischen Union anzubieten. Bitstamp [17] erhebt für die Einzahlung von Kryptowährungen auch keine Gebühren, jedoch fallen für Auszahlungen Gebühren an, die je nach Kryptowährung variieren können<sup>10</sup>. Des Weiteren werden in Abbildung 2.4 die Maker- und Taker-Gebühren dargestellt.

<sup>10</sup> Stand 16.09.2023, entnommen von <https://www.bitstamp.net/fee-schedule/>

30 days USD volume	Standard Maker fee %	Standard Taker fee %	FX Maker fee %	FX Taker fee %
< \$1,000	0.00%	0.00%	0.000%	0.000%
> \$1,000	0.30%	0.40%	0.060%	0.080%
> \$10,000	0.20%	0.30%	0.040%	0.060%
> \$100,000	0.10%	0.20%	0.020%	0.040%
> \$500,000	0.08%	0.18%	0.016%	0.036%
> \$1,500,000	0.06%	0.16%	0.012%	0.032%
> \$5,000,000	0.03%	0.12%	0.006%	0.024%
> \$20,000,000	0.02%	0.10%	0.004%	0.020%
> \$50,000,000	0.01%	0.08%	0.002%	0.016%
> \$100,000,000	0.00%	0.06%	0.000%	0.012%
> \$250,000,000	0.00%	0.05%	0.000%	0.010%
> \$1,000,000,000	0.00%	0.03%	0.000%	0.006%

Abbildung 2.2: Bitstamp [17] Gebührenplan <sup>11</sup>

Nach der detaillierten Betrachtung dieser beiden Exchanges lenkt sich der Fokus nun auf die technologischen Aspekte der Datenverarbeitung und -speicherung, die im Kontext von Kryptowährungen und der zugrundeliegenden Blockchain-Technologie von zentraler Bedeutung sind.

## 2.3 Datenbanken

Bei der Analyse von Kryptowährungen und deren Handelsbewegungen entstehen immense Datenmengen. Diese Daten können Informationen über Preisschwankungen, Handelsvolumen und viele andere relevante Aspekte enthalten. In Kryptowährungsmärkten ist die effiziente Verarbeitung dieser Daten besonders kritisch, da sie die Grundlage für Handelsentscheidungen und Marktübersicht bilden. Um solche Mengen an Informationen sinnvoll zu nutzen, speichern und analysieren zu können, bedarf es effiziente Speichermethoden.

### Was sind Daten?

Bevor überhaupt Speicherlösungen betrachtet werden können, muss geklärt werden, was genau unter „Daten“ zu verstehen ist. In Kontext von Kryptowährungen können diese Daten von einfacher Marktpreisinformation bis zu komplexen Transaktionsdaten auf der Blockchain reichen. Gemäß der Definition des Duden Rechtschreibung [19]

sind Daten durch Beobachtungen, Messungen und andere Methoden gewonnene Werte, oft in Form von Zahlen. Nach Wuttke [20] sind in der Informatik Daten codierte Informationen in Binärkode<sup>12</sup> die elektronisch verarbeitet, übertragen und gespeichert werden können.

### Möglichkeiten der Datenspeicherung

Schneider [21, S. 1, 2] beschreibt, dass in den Anfängen der Computertechnologie die Datenverwaltung hauptsächlich auf von Betriebssystemen unterstützten Dateisystemen basierte. Hierbei definierte jeder Anwendungsprogrammierer die Dateien, die er für seine spezifische Anwendung benötigte, oft ohne Kenntnis von Dateien anderer Anwendungen. Dieser unkoordinierte Ansatz hatte mehrere Nachteile:

- **Redundanz:** Die Daten könnten in mehreren Dateien wiederholt auftreten, was zu Speicherverschwendungen und erhöhtem Verwaltungs- und Verarbeitungsaufwand führt.
- **Inkonsistenz:** Bei Änderungen in einem Datensatz mussten oft mehrere Dateien angepasst werden, was die Konsistenz der Daten gefährdete.
- **Daten-Programm-Abhängigkeit:** Bei Änderungen an der Datenstruktur mussten oft auch die zugehörigen Programme angepasst werden.
- **Inflexibilität:** Da Daten anwendungsgebunden und nicht gesamtheitlich organisiert waren, war es schwierig, sie für neue Anwendungen oder Analysen zu nutzen (vgl. [21, S. 2]).

Als Alternative zu diesen Dateisystemen haben sich DB als effizienter erwiesen, beschreibt Schneider [21, S. 3] weiter. Dies ist insbesondere im Bereich der Kryptowährungen von Bedeutung, wo Transaktionsgeschwindigkeit und Datenintegrität von großer Relevanz sind. Dabei ist eine DB definiert als eine integrierte und strukturierte Sammlung persistenter Daten, die für alle Benutzer als gemeinsame Informationsquelle dient. Sie ist:

- **Integriert:** Sie bietet eine konsistente und anwendungsunabhängige Sicht auf Daten.
- **Strukturiert:** Sie organisiert Daten logisch und zusammenhängend und vermeidet Redundanz.
- **Persistent:** Daten werden dauerhaft auf externen Speichermedien gespeichert.
- **Geteilt und verlässlich:** Mehrere Benutzer können gleichzeitig und sicher darauf zugreifen (Vgl. [21, S. 3]).

---

<sup>12</sup> Als Zahl "0" oder "1"

Nach Dr. Schneider, erhalten die Anwendungsprogramme und Benutzer den Zugriff auf die DB über das sogenannte Datenbankmanagementsystem (DBMS). Ein DBMS ist somit eine Software, die die Interaktion zwischen den Benutzern und der DB ermöglicht und dabei Betriebssystem- und Hardware-Details verbirgt. Darüber hinaus unterstützt das DBMS den Benutzer bei der Definition, Konstruktion und Manipulation von DB. Wenn man dieses DBMS mit den eigentlichen Daten kombiniert, erhält man ein Datenbanksystem (DBS). Das DBS hat die Hauptaufgabe, umfangreiche Mengen strukturierter Informationen zu speichern, zu verwalten und bei Bedarf bereitzustellen (vgl. [21, S. 3, 4]).

Obwohl DB effiziente Speicher- und Abfragemöglichkeiten bieten, sind sie nicht die einzige verfügbare Technologie zur Verwaltung großer, komplexer Datenmengen. Gerade im Kontext von Kryptowährungen, wo die Transparenz und Unveränderlichkeit von Daten eine zentrale Rolle spielen, bieten andere Ansätze wie Verteilte Datenbanken (VDB) zusätzliche Vorteile. Diese Technologien bilden die Grundlage für fortschrittlichere Systeme wie die Blockchain, die im folgenden Abschnitt erläutert wird.

## 2.4 Verteilte Datenbank

Rahm, Saake und Sattler [22, S. 5] erörtern, dass zentralisierte DB bei großen Datenmengen und Nutzerzahlen an ihre Leistungsgrenzen stoßen und im Falle eines Serverausfalls eine geringere Verfügbarkeit bieten. Diese Beschränkungen führten zur Entwicklung von VDB, bei denen die Daten über mehrere Rechner hinweg verteilt gespeichert bzw. koordiniert verarbeitet werden.

Eine VDB besteht, wie Rahm, Saake und Sattler [22, S. 6, 7] weiter ausführen, aus einem Netzwerk von miteinander verbundenen Rechnerknoten, von denen jeder Knoten eine eigene Instanz des DBMSs ausführt und einen Teil der DB verwaltet. Dabei kooperieren die DBMS miteinander, um dennoch auf die Daten aller Knoten zugreifen zu können. Durch diese Verteilung von Daten wird der Ausfall eines beliebigen Rechners das Netz nicht vollständig lahmlegen. Bis auf die Daten des ausgefallenen Rechners, bleibt der Zugriff auf alle weiteren Daten gewährleistet, was eine hohe Ausfallsicherheit garantiert. Schicker [23, S. 302, 303] fügt noch hinzu, dass VDB tausende parallele Abfragen auf verschiedenen Rechnern und DB lenken können, wodurch die Antwortzeiten niedrig bleiben.

Zusammenfassend bieten VDB eine effiziente und ausfallsichere Lösung für die Speicherung und Verwaltung großer Datenmengen. Allerdings ist in speziellen Anwen-

dungsfällen, wie dem der Kryptowährungen, nicht nur die Skalierbarkeit und Ausfallsicherheit von Bedeutung. Hier spielen auch andere Faktoren wie Transparenz, Unveränderlichkeit und Dezentralität eine entscheidende Rolle. Genau diese Anforderungen können mit der Blockchain-Technologie erfüllt werden, die als eine Erweiterung oder Variation des Konzepts der VDB betrachtet werden kann.

## 2.5 Blockchain

Blockchain ist seit 2008 mit der Veröffentlichung des Papers „Bitcoin: A peer-to-peer Electronic cash system“ [1] von Satoshi Nakamoto [1] erstmals publiziert worden. Tönnissen und Teuteberg [24, S. 1171] definieren die Blockchain als eine Kette von Datensätzen, die mittels kryptographischer Verfahren zu einem Datenblock verbunden werden und auf allen teilnehmenden Rechnern verteilt werden. Das bedeutet, dass jeder Knoten im Netzwerk eine vollständige Kopie der gesamten Blockchain enthält. Tönnissen und Teuteberg [24, S. 1171, 1172] erörtern weiterhin, dass die Integration eines neuen Blocks in die Blockchain über komplexe Konsensmechanismen wie das Proof-of-Work-Verfahren erfolgt, welche umfangreiche Rechenoperationen erfordern. Nachdem diese Prozesse erfolgreich abgeschlossen sind, wird der Block in die Kette eingefügt und mithilfe kryptographischer Methoden sicher mit dem vorherigen Block verbunden, wodurch eine lückenlose, chronologische Sequenz von Blöcken entsteht. Zuletzt definieren Tönnissen und Teuteberg [24, S. 1171, 1172] die Datensätze der Blockchain als transparent und für alle Netzwerkteilnehmer einsehbar. Dieses wird durch das Peer-to-Peer-Netzwerk ermöglicht. Die Dezentralisierung der Datenhaltung gewährleistet die Unveränderbarkeit und Konsistenz der Datenblöcke. Um eine Manipulation durchzuführen, wäre es erforderlich, den betreffenden Block und alle nachfolgenden Hashwerte zu modifizieren und diese Änderungen an das gesamte Netzwerk zu verteilen. Dieses Vorgehen würde jedoch an der fehlenden Übereinstimmung mit dem bestehenden Netzwerkconsens scheitern, da das Netzwerk die modifizierten Blöcke nicht annehmen würde. Die Integrität der Daten auf der Blockchain ist somit gewährleistet.

Die Verwendung von Blockchain für Kryptowährungen bietet viele Vorteile. Neben der Unveränderbarkeit der Transaktionshistorie sorgt es auch für Transparenz, da alle Transaktionen in der Blockchain sichtbar sind. Darüber hinaus ermöglicht die Peer-To-Peer Übertragung einen dezentralen Vermittler, wodurch sie unabhängig von zentralen Autoritäten und widerstandsfähig gegen Zensur sind. Schließlich bietet die Blockchain eine Zugänglichkeit für alle Menschen mit einem Internetzugang, somit auch für Menschen, die sonst keinen Zugang zu traditionellen Bankdienstleistungen

hätten.

Während die Blockchain-Technologie die Grundlagen für den sicheren und transparenten Handel von Kryptowährungen schafft, sind für den praktischen Handel und die Analyse von Kryptomärkten spezialisierte Tools notwendig. Hier kommen APIs ins Spiel. Sie ermöglichen den Zugriff auf spezifische Marktinformationen wie Preise und Handelsvolumen, die für die Generierung fundierter Handlungsempfehlungen unerlässlich sind.

## 2.6 Application Programming Interface

RedHat [25] beschreibt, dass eine API eine Schnittstelle ist, die es unabhängigen Anwendungen ermöglicht, miteinander zu kommunizieren und Daten auszutauschen. APIs bieten eine einfache Möglichkeit, auf bestimmte Funktionen einer Software zuzugreifen, ohne die internen Funktionsweisen kennen zu müssen.

In Bezug auf Kryptowährungen ermöglichen APIs den Zugriff auf Echtzeitdaten von Exchanges. Diese APIs bieten eine Vielzahl von Funktionen, einschließlich der Abfrage von Handelsinformationen und Markdaten. Darüber hinaus ermöglichen sie den automatisierten Handel, da sie die Durchführung von Kauf- und Verkaufsorders erlauben. Allerdings können die zahlreichen und unterschiedlich gestalteten APIs verschiedener Exchanges die Komplexität für den Anwender erhöhen. Hier setzt die CryptoCurrency eXchange Trading Library (CCXT)-Bibliothek [26] an. Sie standardisiert den Zugriff auf die verschiedenen Exchanges und ist daher besonders für Anwender interessant, die Handlungsempfehlungen auf der Grundlage einer breiten Datenbasis generieren möchten.

## 2.7 CryptoCurrency eXchange Trading Library

Die CCXT-Bibliothek [26], wie in der Dokumentation von ccxt beschrieben, ist eine herunterladbare Software-Bibliothek, die Entwicklern vorgefertigte Funktionen bietet, um sich mit Exchanges weltweit zu verbinden und dort zu handeln. Durch die Nutzung von CCXT [26] können Anwender unmittelbar auf die APIs der verschiedenen Kryptowährungsbörsen zugreifen, ohne diese von Grund auf neu programmieren zu müssen. Dadurch kann ein schneller standardisierter Zugriff auf die Marktdaten hergestellt werden.

Zu den aktuellen Funktionen gehören:

- Vollständig implementierte öffentliche und private APIs.
- Unterstützung für vieler Exchanges.
- Eine sofort einsatzbereite, vereinheitlichte API, die einfach zu integrieren ist.
- Kompatibilität mit Node 10.4+, Python 3, PHP 8.1+, netstandard2.0/2.1 und Webbrowern (vgl. [26]).

Die CCXT-Bibliothek [26] vereinfacht den Zugang zu den APIs der verschiedenen Exchanges und unterstützt damit die Generierung datenbasierter Handlungsempfehlungen. Während die Bibliothek selbst einfach in eine Anwendung importiert werden kann, stellt sich für Entwickler die Frage nach der geeigneten Programmiersprache, die zur Implementierung und zum Betreib der Anwendung verwendet werden soll. In diesem Zusammenhang erweist sich Python [27] als eine gute Option, da die CCXT-Bibliothek [26] explizit eine Integration mit Python [27] unterstützt.

## 2.8 Python

Python [27] ist eine hochrangig, interpretierte Sprache, die für ihre saubere Syntax und Code-Lesbarkeit geschätzt wird. Die einfache Syntax von Python [27] fördert einen klaren Schreib- und Leseprozess des Codes, was die Effizienz in der Softwareentwicklung deutlich erhöht (vgl. [4]).

Python [27] mit der CCXT-Bibliothek [26], ermöglicht einen vereinfachten und direkten Zugriff auf die Exchanges durch wenige, klar strukturierte Codezeilen. Dies erlaubt es, automatisiert auf Echtzeitdaten zuzugreifen und diese zu verarbeiten. Angesichts der enormen Datenmengen, die bei der Analyse von Kryptowährungen anfallen, ist es jedoch ratsam, eine effiziente Datenverwaltungsmethode zu implementieren. Hier kommt das Konzept des Nachrichten Brokers ins Spiel.

## 2.9 Nachrichten Broker

Ein Nachrichten Broker ist ein System, bei dem Nachrichten von einem Punkt (Publisher) zu einem anderen (Consumer) gesendet werden, wobei eine Warteschlangentechnologie verwendet wird, um die Nachrichten zu speichern, bis sie vom Empfänger abgerufen und verarbeitet werden. Dieses Konzept dient der effizienten Verwaltung von Nachrichten zwischen Anwendungen, wodurch die Lastverteilung optimiert und die Skalierbarkeit verbessert wird.

Nach IBM [28] spielen in einem Nachrichten Broker Publisher und Consumer grundlegende Rollen. Der Publisher ist der Sender der Nachrichten. Seine Hauptaufgabe

besteht darin, Nachrichten in die Warteschlange zu stellen. Während der Consumer hingegen der Endpunkt ist, der tatsächlich Nachrichten aus der Warteschlange verarbeitet. Er nimmt eine Nachricht aus der Warteschlange, führt eine bestimmte Operation durch, etwa die Verarbeitung eingehender Daten und entfernt die Nachricht anschließend aus der Warteschlange.

Angesichts der kontinuierlich hohen Datenströme der Kryptowährungspreise, welche durch die Exchanges bereitgestellt werden und der Notwendigkeit, diese Daten nahezu in Echtzeit zu analysieren, wird die Verwendung eines Nachrichten Broker für die effiziente Verarbeitung dieser Daten helfen. Mit den Daten, die von der CCXT-Bibliothek [26] bereitgestellt werden, fungiert die Python [27] Anwendung hier als Publisher, der die Marktdaten in den Nachrichten Broker einbringt.

Als nächsten Schritt wird eine Anwendung benötigt, welche das Consumer-Verhalten implementiert. Da eignet sich besonders gut eine Programmiersprache, die eine schnelle und ressourceneffiziente Verarbeitung der Daten ermöglicht. Hier kann Go [5] eine Schlüsselrolle spielen. Mit seiner Leistungsfähigkeit und seine einfache Handhabung von asynchronen Operationen, bietet Go [5] die Möglichkeit, als effizienter Consumer zu agieren, der die Daten aus dem Nachrichten Broker abruft und verarbeitet.

## 2.10 Go

Go [5], auch bekannt als „Golang“, ist eine von Google-Mitarbeitern entwickelte statisch typisierte und kompilierte Programmiersprache, die sich durch eine einfache Syntax auszeichnet. Dies ist insbesondere für die parallele Verarbeitung und die Ausführung in hochbelasteten Systemumgebungen von Bedeutung. Gemäß dem Buch *The Go Programming Language* [29, S. xi] wurde Go [5] mit dem Ziel entwickelt, „einfach, zuverlässig und effizient Software zu programmieren“. Go [5] zeichnet sich durch seine Fähigkeit aus, ausdrucksstarken, leistungsfähigen und effektiven Code zu schreiben, welcher sowohl bei der Kompilierung als auch in der Ausführung Zuverlässigkeit und Robustheit bietet. Ein besonderes Merkmal von Go [5] ist die integrierte Unterstützung für Nebenläufigkeit, realisiert durch Goroutinen [5] und Kanäle. Dies ist vor allem für Back-End-Dienste von großer Bedeutung, die eine hohe Anzahl von Anfragen handhaben und dabei Daten schnell und verlässlich verarbeiten müssen.

In Bezug auf die Verarbeitung von Kryptowährungsdaten stellt Go [5] eine geeignete Lösung dar, um als Consumer zu agieren. Es kann die umfangreichen, durch die Exchanges bereitgestellten Datenströme effizient und schnell verarbeiten. Wichtig dabei ist aber auch die Persistenz dieser zeitbezogenen Daten, da diese später zur Analyse

bereitstehen müssen. Daher erfordert die Handhabung dieser Daten eine geeignete Speicherlösung, die eine effiziente Datenaufzeichnung und -abfrage unterstützt, um sowohl die Echtzeit-Analyse als auch die historische Datenforschung zu ermöglichen.

## 2.11 Zeitreihendatenbank

Deri, Mainardi und Fusco [30, S. 143] erklären, dass eine Zeitreihendatenbank (engl. Time Series Database, TSDB) speziell für die Handhabung von Zeitreihendaten konzipiert ist, also für Daten, die im Zeitverlauf gesammelt wurden. Im Gegensatz zu herkömmlichen relationalen DB optimieren TSDB die Speicherung und Abfrage von sequentiell angeordneten Datenpunkten. Diese Eigenschaften machen TSDBs besonders geeignet für die Aufnahme von Kryptowährungsdaten, da sie effiziente Schreiboperationen, Datenreduktionstechniken und Abfragefunktionen bieten, die für die Verarbeitung großer Mengen von chronologischen Daten optimiert sind.

Die effiziente Struktur von TSDB erlaubt es, umfangreiche Mengen von Kryptowährungsdaten zu speichern und systematisch zu organisieren, wodurch eine zeitnahe und zuverlässige Datenabrufbarkeit für zukünftige Analysen gewährleistet wird. Diese Persistenz gewährleistet den Zugang zu historischen Daten, die für die Prognose von Markttrends wichtig sind.

Allerdings reicht es nicht aus, nur auf persistente historische Daten zuzugreifen. Es müssen regelmäßig umfangreiche Berechnungen durchgeführt werden, um den gesamten Markt zu analysieren und Handlungsempfehlungen abzugeben. In solchen Szenarien, in denen wiederholte und ressourcenintensive Abfragen erforderlich sind, erweist sich die Verwendung einer In-Memory-Datenbank (IMDB) als vorteilhaft. Indem Berechnungen durchgeführt und ihre Ergebnisse zur schnellen Abrufung in einer IMDB zwischengespeichert werden, kann die Effizienz gesteigert und die Antwortzeit reduziert werden. Dies ist für die Bedienung mehrerer Benutzer entscheidend ist.

## 2.12 In-Memory-Datenbank

Nach Luber und Litzel [31], ist eine IMDB ein DBMS, das Datenstrukturen hauptsächlich im Ram-Speicher eines Computers hält, um die Zugriffszeiten auf die Daten stark zu reduzieren. Im Gegensatz zu DB, die ihre Daten auf Festplatten speichert, ermöglicht die IMDB den schnellen Zugriff und die Verarbeitung von Daten. Dies ist besonders bei Anwendungen mit intensiven Lese-/Schreiboperationen und einem Bedarf an geringer Latenzzeit von Vorteil ist.

Angesichts der ressourcenintensiven Berechnungen, die für TSDB erforderlich sind,

bietet sich die IMDB als gute Lösung an. Durch die Zwischenspeicherung der Ergebnisse dieser Berechnungen in der IMDB können Benutzern unmittelbar Ergebnisse bereitgestellt werden, ohne dass diese Berechnungen für jeden einzelnen Nutzer separat durchgeführt werden müssen. Dies spart nicht nur wertvolle Rechenzeit, sondern ermöglicht auch eine schnellere und effizientere Benutzererfahrung. Doch selbst mit dieser Optimierung in der Datenverarbeitung ist es von entscheidender Bedeutung, wie diese Daten dem Endbenutzer präsentiert werden. Hier kommt das Frontend ins Spiel.

## 2.13 Frontend

Das Frontend einer Anwendung bezeichnet den visuellen Bereich, mit dem der Anwender interagiert. Es ist das Gesicht der Software. Hier werden Informationen aufbereitet und dem Nutzer präsentiert, während das Backend im Hintergrund die Datenverarbeitung und -bereitstellung übernimmt. Für den Anwender ist der nahtlose Übergang zwischen Datenabfrage und visueller Darstellung entscheidend, ohne dass die Komplexität des Backends in Erscheinung tritt. Next.js [32] ist dabei ein populäres Framework, das für die Erstellung solch interaktiver und benutzerorientierter Frontends genutzt wird und den Zugriff über Webbrowser ermöglicht.

## 2.14 Next.js

Next.js [32] ist ein vielseitiges Framework, das für seine Flexibilität und Leistungsfähigkeit geschätzt wird, insbesondere durch die Unterstützung verschiedener Rendering-Methoden. Es bietet:

- Client Side Rendering (CSR) für dynamisch abgerufene Inhalte, was eine interaktive Nutzererfahrung ermöglicht.
- Server Side Rendering (SSR) für optimierte Ladezeiten und verbesserte SEO, indem Inhalte bereits vollständig gerendert beim Nutzer ankommen.
- Static Site Generation (SSG) für schnelle Ladezeiten bei statischen Inhalten, die keine häufigen Aktualisierungen benötigen.
- Incremental Static Regeneration (ISR) kombiniert die Vorteile von SSG für statische Inhalte, die nur gelegentlich aktualisiert werden müssen, ohne eine komplette Neugenerierung der Seite (vgl. [32]).

Next.js [32] erlaubt es zudem, diese Strategien zu kombinieren, um eine optimierte

Nutzererfahrung zu schaffen, bei der statische Elemente schnell geladen und dynamische Inhalte effizient integriert werden.

Next.js [32] wird typischerweise für die Erstellung moderner Webanwendungen verwendet und kann sowohl in JavaScript [33] als auch TypeScript [6] programmiert werden. TypeScript ist eine typsichere Erweiterung von JavaScript [33] und erhöht die Sicherheit des Codes, was besonders bei komplexen Anwendungen von Vorteil ist. Die Nutzung von TypeScript [6] in Next.js-Projekten [32] fördert die Entwicklung zuverlässiger Anwendungen durch die Bereitstellung starker Typisierungen.

Mit der Darstellung der Möglichkeiten von Next.js [32] und der Typsicherheit durch Typescript [6] wurde nun aufgezeigt, wie Anwendungen visuell dargestellt und interaktiv gestaltet werden können. Um Daten effektiv darzustellen, bedarf es einer zuverlässigen Kommunikationsstelle zwischen Frontend und dem Backend. APIs, welche bereits in Abschnitt 2.6 vorgestellt wurden, bieten hier eine Standardmethode für solche Datenanfragen. Um den Nutzern aber kontinuierliche Preisaktualisierungen zu liefern, wird eine Websocket Verbindung benötigt.

## 2.15 Websocket

Ein Websocket ist dafür zuständig, eine nahtlose Kommunikation zwischen dem Nutzer und dem Backend zu gewährleisten. Websockts erlauben es dabei, eine Verbindung dauerhaft für bidirektionale Kommunikation zu öffnen, was für die Implementierung von Echtzeitfunktionalitäten wie dem Aktualisieren von Kryptowährungspreisen in der Nutzeroberfläche von entscheidender Bedeutung ist. Diese Technologie ergänzt die API-Integration, indem sie einen direkten und ständigen Kommunikationsfluss ermöglicht, der für Anwendungen im Kryptowährungsbereich, wo Marktdaten sich sekündlich ändern können, essenziell ist.

Die Rolle von WebSockets in der Bereitstellung von Echtzeitdaten zeigt, wie entscheidend die Wahl der richtigen Kommunikationstechnologie für die Funktionalität und Benutzererfahrung einer Anwendung ist. Diese Überlegungen führen zur zentralen Rolle des Webservers, der als Bindeglied zwischen den verschiedenen Technologien und dem Endnutzer fungiert.

## 2.16 Webserver

In einer Webanwendung ist der Webserver die primäre Komponente, mit der der Benutzer interagiert. Er fungiert als Eingangstor für den Benutzer, indem er Benutzeranfragen entgegennimmt, diese an externe Systeme weiterleitet und schließlich die

erhaltenen Antworten an den Benutzer zurückgibt (vgl. [34]). Der Webserver leitet die Nutzeranfragen in der Regel basierend auf der URL an die entsprechenden Dienste weiter. Bei Anfragen, die das Frontend betreffen, greift der Webserver entweder selbst auf statische Daten zu und stellt diese dem Benutzer bereit, oder er fragt diese Daten von einem spezialisierten Frontend-Server ab. Backend-Daten werden hingegen über einen API-Server oder WebSocket-Server weitergeleitet. Die Ergebnisse dieser Anfragen werden dann dem Anwender zurückgegeben. Diese Arbeitsweise ermöglicht dem Webserver, als effektive Schnittstelle zwischen der Anwendung und dem Endnutzer zu dienen und einen zuverlässigen sowie kontinuierlichen Zugriff auf die Anwendung zu gewährleisten.

Nach einer umfassenden Betrachtung der für die Entwicklung einer Webanwendung erforderlichen Technologien wird nun abschließend eine Technologie vorgestellt, die die Bereitstellung der gesamten Infrastruktur in eigenständigen Containern ermöglicht. Diese Methode garantiert Konsistenz über verschiedene Umgebungen hinweg und trägt gleichzeitig zur Verbesserung der Portabilität und Skalierbarkeit bei.

## 2.17 Docker

Um die Bereitstellung und den Betrieb der Anwendung zu vereinfachen und gleichzeitig eine hohe Zuverlässigkeit zu gewährleisten, kann Docker [35] als ein wichtiges Werkzeug eingesetzt werden. Docker [35] ermöglicht es, alle benötigten Services wie Nachrichten Broker, Consumer, Webserver und weitere in jeweils separate leichtgewichtige Container zu packen. Diese Container können dann miteinander interagieren, wodurch eine modulare und gut organisierte Infrastruktur geschaffen wird. Das Resultat ist nicht nur eine erleichterte Deployment-Prozedur, sondern auch eine erhebliche Verbesserung der Skalierbarkeit und Wartbarkeit der gesamten Anwendungslandschaft (vgl. [36]).

## Zusammenfassung

In dem vorgestellten wissenschaftlichen Diskurs wurden zunächst die Grundlagen digitaler Vermögenswerte behandelt, wobei ein besonderer Fokus auf Kryptowährungen gelegt wurde. Diese sind insbesondere dadurch charakterisiert, dass sie unabhängig von staatlichen Institutionen agieren und eine hohe Volatilität aufweisen. Dabei spielen Faktoren wie Transaktionskosten und Handelsgröße eine signifikante Rolle im Krypto-Handel. Anschließend wurden Kryptobörsen, exemplarisch Kraken [16] und Bitstamp [17], vorgestellt, die sich durch ihre EU-Lizenzen und die Distanz zu umstrittenen

Steueroasen in ihrer Authentizität und im Vertrauen der Nutzer abheben.

Folgend richtet sich der Blick auf die technologischen Aspekte, die den Handel und die Verwaltung von Kryptowährungen ermöglichen. Hierbei ist zunächst die Relevanz von DB zu nennen, insbesondere von VDB und der Blockchain-Technologie, welche den dezentralen und sicheren Austausch von Kryptowährungen fundiert. Die Implementierung von APIs, speziell durch die CCXT-Bibliothek [26], standardisiert den Zugriff auf verschiedene Kryptobörsen und vereinfacht die Integration in Handelsanwendungen.

Abschließend wurde die Bedeutung von Technologien für die Entwicklung und den Betrieb von Webapplikationen untersucht. Im Gesamten wurden alle notwendigen Technologien analysiert, die für die Realisierung einer solchen Webapplikation erforderlich sind. Diese umfassende Untersuchung bildet das Fundament für die anschließende Konzeption und Umsetzung der Anwendung.

Mit dem Abschluss von Kapitel 2 wurde das Forschungsziel FZ1O erreicht, indem geeignete Konzepte, Methoden und Technologien für die Entwicklung eines automatisierten Systems zur effizienten Erfassung und zum Vergleich von Kryptowährungspreisen identifiziert wurden.

# Kapitel 3

## Konzeptionelle Modellierung und Design

In diesem Kapitel wird die konzeptionelle Modellierung und das Design auf der Grundlage des *User Centered System Design: New Perspectives on Human-computer Interaction* von Norman und Draper [37] strukturiert. Dieser Ansatz beinhaltet ein konzeptionelles Anwendungsmodell, ein konzeptionelles Informationsmodell und ein konzeptionelles Architekturmodell. Diese Modelle sind darauf ausgerichtet, die Softwareentwicklung an den Anforderungen und Bedürfnissen der Benutzern sicherzustellen. Das Hauptziel ist die Gestaltung eines Systems, welches die Unterschiede in den Kostenstrukturen der Exchanges berücksichtigt und dem Nutzer profitable Handlungsmöglichkeiten transparent aufzeigt. Dieser Ansatz entspricht dem FZ2T, bei dem es um die Entwicklung eines Konzepts geht, das die identifizierten Konzepte, Methoden und Technologien in das System integriert. Die konzeptionellen Modelle, die die Basis für das Design des Systems bilden, werden in den nachfolgenden Kapiteln 3.1, 3.2 und 3.3 dargestellt.

### 3.1 Konzeptionelles Anwendungsfallmodell

Das konzeptionelle Anwendungsfallmodell dient dazu, das geplante System aus der Perspektive potentieller Benutzer zu betrachten und somit die Interaktionen und Anforderungen zu identifizieren, die für eine effiziente und effektive Nutzung erforderlich sind.

Zur Visualisierung dieser Interaktionen und Anforderungen wird das konzeptionelle Anwendungsfallmodell als UML Use Case Diagramm [38] modelliert. Dieses Modell wird in Abbildung 3.1 dargestellt und bietet einen Überblick über die Beziehungen und Abhängigkeiten zwischen den einzelnen Anwendungsfällen.

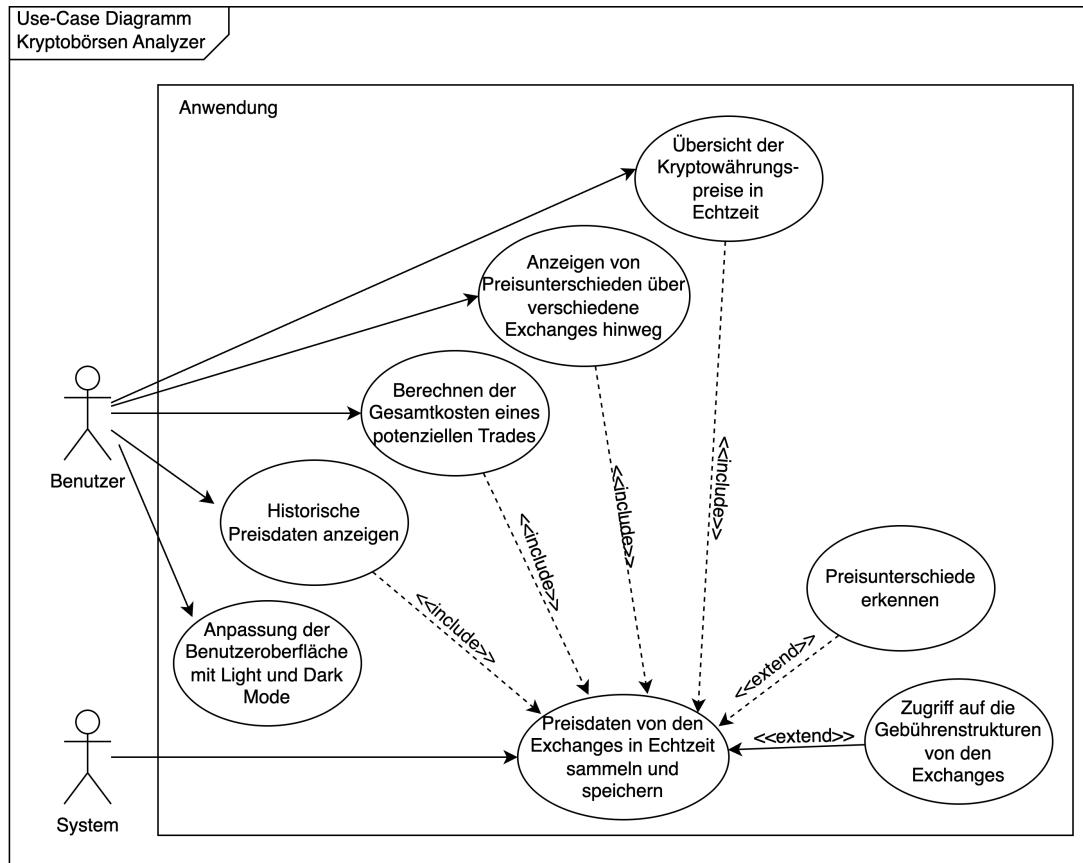


Abbildung 3.1: Konzeptionelles Anwendungsfallmodell als UML Use Diagramm modelliert [38]

Nachdem das Modell in Abbildung 3.1 dargestellt wurde, werden nun die 6 Use-Cases beschrieben. Diese Use-Cases bilden das Fundament für das konzeptionelle Informationsmodell und sind entscheidend für die Entwicklung der Datenstrukturen und der technischen Architektur des Systems.

### Use-Case 1: Übersicht der Kryptowährungspreise in Echtzeit

- **Ziel:** Dem Benutzer wird eine kontinuierlich aktualisierte Ansicht der Marktpreise verschiedener Kryptowährungen zur Verfügung gestellt. Dies soll dem Benutzer helfen, schnell auf Marktveränderungen zu reagieren.
- **Voraussetzungen:** Das System hat Zugriff auf aktuelle Preisdaten von verschiedenen Exchanges und ist in der Lage, diese in Echtzeit zu verarbeiten und anzuzeigen.
- **Hauptablauf:**

1. Der Benutzer navigiert zur Seite der Echtzeit-Kursübersicht.
2. Das System ruft die aktuellen Kryptowährungspreise über die verbundenen

Datenquellen ab.

3. Die Informationen werden in einer übersichtlichen Tabelle dargestellt, welche eine schnelle Visualisierung und Verständnis ermöglicht.
4. Die Preisdaten in der Übersicht werden kontinuierlich (z.B. alle 5 Sekunden) aktualisiert, ohne dass eine Benutzeraktion erforderlich ist.

- **Alternativer Ablauf:** Verbindungs- oder Datenempfangsprobleme: Das System zeigt eine Benachrichtigung oder Warnung an, die auf die aktuelle Unterbrechung hinweist, und gibt gegebenenfalls den Zeitpunkt der letzten erfolgreichen Datenaktualisierung an.
- **Nachbedingungen:** Der Benutzer bleibt über die neuesten Preisentwicklungen informiert und kann auf Basis der neuesten Daten Handelsentscheidungen treffen.

#### **Use-Case 2: Anzeigen von Preisunterschieden über verschiedene Exchanges hinweg**

- **Ziel:** Dem Benutzer eine Übersicht über die Preisunterschiede einer bestimmten Kryptowährung auf verschiedenen Exchanges bieten.
- **Voraussetzungen:** Das System hat Zugriff auf aktuelle Preisdaten von verschiedenen Exchanges.
- **Hauptablauf:**
  1. Der Benutzer wählt eine Kryptowährung aus.
  2. Das System sammelt Preisdaten von verschiedenen Exchanges.
  3. Das System zeigt die Preisunterschiede in einer übersichtlichen Tabelle an.
- **Alternativer Ablauf:** Keine aktuellen Daten verfügbar: Das System zeigt eine Fehlermeldung und bietet dem Benutzer an, später erneut zu versuchen.
- **Nachbedingungen:** Der Benutzer hat eine klare Vorstellung von den Preisunterschieden auf verschiedenen Exchanges.

#### **Use-Case 3: Berechnen der Gesamtkosten eines potentiellen Handels**

- **Ziel:** Dem Benutzer eine Schätzung der Gesamtkosten für einen potentiellen Handel bieten, einschließlich aller Exchanges Gebühren.
- **Voraussetzungen:** Das System hat aktuelle Informationen über die Gebührenstruktur der jeweiligen Exchanges.
- **Hauptablauf:**
  1. Der Benutzer gibt die Details des potentiellen Handels ein (Kryptowährung, Menge, beteiligter Exchange).

2. Das System berechnet die Gesamtkosten basierend auf den Gebührenstrukturen und zeigt sie dem Benutzer an.
- **Alternativer Ablauf:** Keine aktuellen Daten verfügbar: Das System zeigt eine Fehlermeldung und bietet dem Benutzer an, später erneut zu versuchen.
  - **Nachbedingungen:** Der Benutzer ist über die Gesamtkosten des Handels informiert und kann eine fundierte Entscheidung treffen.

#### Use-Case 4: Historische Preisdaten anzeigen

- **Ziel:** Dem Benutzer eine Übersicht über historische Preisdaten bieten, um Trends zu analysieren und fundierte Entscheidungen für zukünftige Handlungen zu treffen.
  - **Voraussetzungen:** Das System hat Zugriff auf archivierte Preisinformationen der verschiedenen Exchanges.
  - **Hauptablauf:**
    1. Der Benutzer wählt eine Kryptowährung und einen Zeitraum aus.
    2. Das System sammelt historische Daten für den angegebenen Zeitraum.
    3. Das System zeigt dem Benutzer einen Chart mit den historischen Daten an.
- **Alternativer Ablauf:** Keine historischen Daten für den angegebenen Zeitraum verfügbar: Das System zeigt eine Fehlermeldung.
  - **Nachbedingungen:** Der Benutzer hat eine klare Vorstellung von historischen Preisentwicklung und kann diese Informationen für zukünftige Handelsentscheidungen nutzen.

#### Use-Case 5: Automatische Identifizierung von Preisunterschieden

- **Ziel:** Das System identifiziert automatisch Preisunterschiede zwischen verschiedenen Exchanges für eine ausgewählte Kryptowährung.
  - **Voraussetzungen:** Das System hat Zugang zu den aktuellen Preisinformationen von den verschiedenen Exchanges.
  - **Hauptablauf:**
    1. Das System fragt in regelmäßigen Abständen die Preise der Kryptowährungen ab.
    2. Die abgefragten Preisinformationen werden kontinuierlich verarbeitet und Preisunterschiede berechnet und abgespeichert.
- **Alternativer Ablauf:** Bei einem Verbindungs- oder Abfragefehler versucht das System, die Daten erneut abzurufen oder setzt die Abfrage im nächsten geplanten Intervall fort.

- **Nachbedingungen:** Die aktuellen Preisunterschiede wurden erfolgreich abgespeichert und die Daten stehen für spätere Vergleiche zur Verfügung.

#### Use-Case 6: Anpassung der Benutzeroberfläche mit Light und Dark Mode

- **Ziel:** Dem Benutzer die Möglichkeit bieten, zwischen einem Light Mode (heller Modus) und einem Dark Mode (dunkler Modus) zu wechseln, um die Benutzeroberfläche je nach persönlicher Vorliebe anzupassen.
- **Voraussetzungen:** Das System unterstützt sowohl einen Light als auch einen Dark Mode und kann zwischen diesen wechseln.
- **Hauptablauf:**
  1. Der Benutzer hat im Header die Möglichkeit zwischen einem Light und Dark Mode zu wechseln.
  2. Das System wendet den gewählten Modus auf alle Oberflächenelemente an.
  3. Die Auswahl des Benutzers wird gespeichert, sodass der Modus bei der nächsten Sitzung beibehalten wird.
- **Nicht unterstützter Modus:** Sollte ein Modus nicht unterstützt werden, wird der Benutzer darüber informiert und auf den Standardmodus zurückgesetzt.
- **Nachbedingungen:** Der Benutzer genießt eine visuell angenehme und augenschonende Erfahrung, die sich an seine individuellen Bedürfnisse anpasst.

Das vorgestellte Anwendungsfallmodell stellt die Interaktionen des Benutzers mit der Anwendung dar. Durch diese detaillierte Betrachtung der Use-Cases wird sichergestellt, dass das endgültige System in der Lage ist, die FF1 zu beantworten. Das Modell trägt dazu bei, dass die entwickelte Lösung die realen Bedürfnisse und Interaktionen der Benutzer berücksichtigt. Im Hinblick darauf ist es auch wichtig, die zugrundeliegenden Datenstrukturen zu verstehen, die im folgenden Kapitel durch das konzeptionelle Informationsmodell näher beleuchtet werden.

## 3.2 Konzeptionelles Informationsmodell

Ein konzeptionelles Informationsmodell beschreibt die logischen Entitäten (Klassen) der Anwendung sowie ihre Beziehungen zueinander. In diesem Kontext dient es dazu, die grundlegenden Datenstrukturen zu definieren, mit denen das System arbeitet und deren Beziehungen untereinander.

Das konzeptionelle Informationsmodell wird als UML Klassendiagramm [38] modelliert und in Abbildung 3.2 dargestellt.

### 3. Konzeptionelle Modellierung und Design

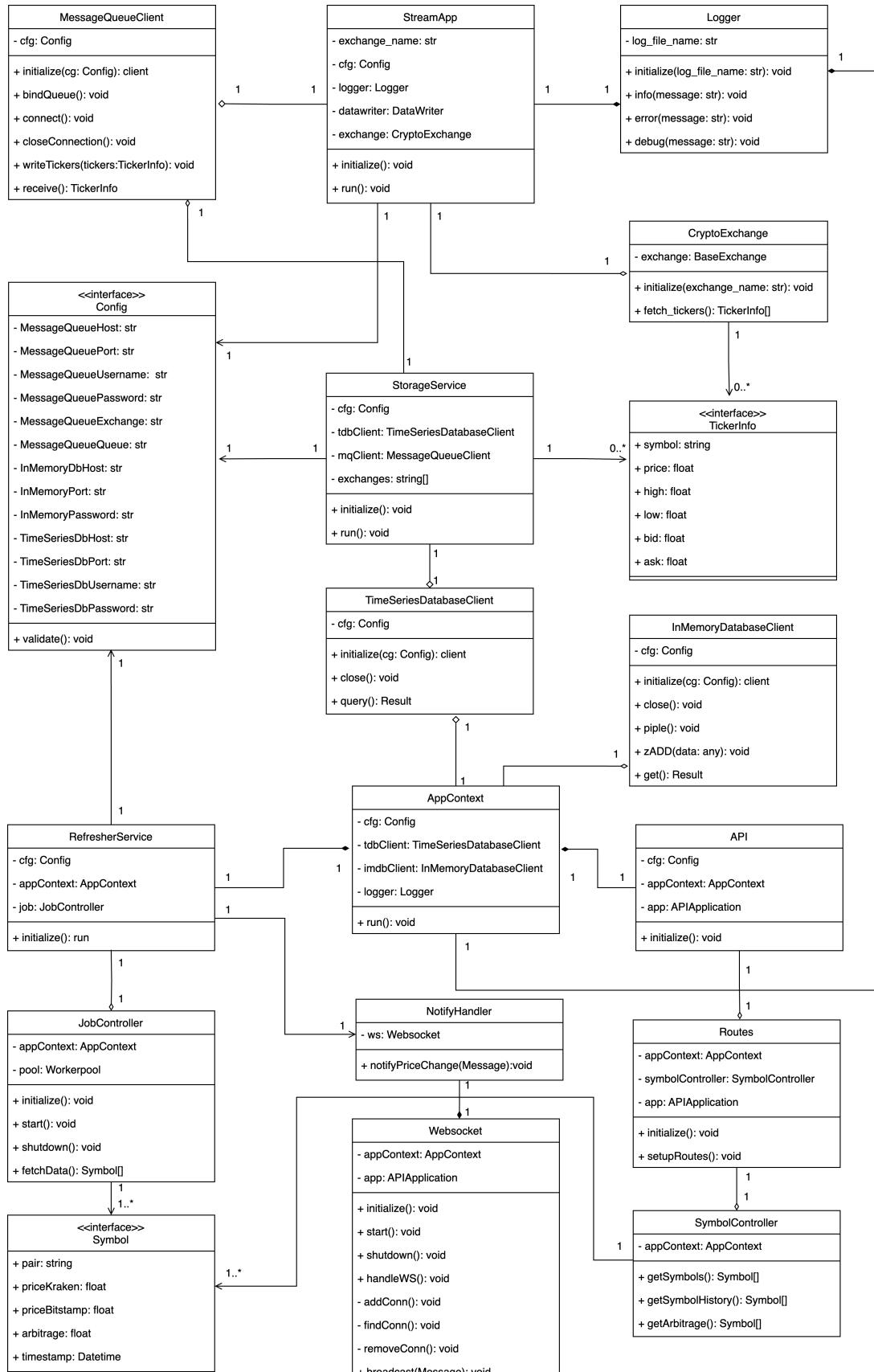


Abbildung 3.2: Konzeptionelles Informationsmodell als UML Klassendiagramm modelliert [38]

Wie in Abbildung 3.2 ersichtlich, umfasst das Modell eine Reihe von Klassen, die für die Umsetzung der im Abschnitt 3.1 identifizierten Use-Cases erforderlich sind. Im Folgenden werden die einzelnen Klassen und ihre Beziehungen zueinander genauer erläutert.

Identifizierung der Klassen:

- **CryptoExchange:** Die CryptoExchange Klasse ist dafür zuständig, die Kryptowährungspreise von den Exchanges über die CCXT-Bibliothek [26] abzurufen. Die CryptoExchange Klasse ist die primäre Schnittstelle für die Datenbeschaffung.
- **StreamApp:** Die StreamApp Klasse greift über den CryptoExchange auf die Kryptowährungspreise zu und legt diese Daten in eine Warteschlange vom Nachrichten Broker ab.
- **MessageQueueClient:** Damit Daten in eine Warteschlange vom Nachrichten Broker eingereicht werden können, wird eine Schnittstelle benötigt, welche mit dem Nachrichten Broker interagiert. Das ist die Aufgabe des MessageQueueClient. Der MessageQueueClient dient dazu, Nachrichten in eine Warteschlange zu senden als auch daraus zu lesen.
- **StorageService:** Die kontinuierliche und schnelle Datenbeschaffung bringt die Herausforderung mit sich, diese Daten auch ebenso effizient zu speichern. Der StorageService ist verantwortlich für das Lesen von Daten aus der Warteschlange und übernimmt die Aufgabe, diese Daten in einer geeigneten Datenbank abzulegen.
- **TSDB-Client:** Wie der Name schon sagt, interagiert diese Klasse mit einer TSDB.
- **API & Routes:** Die Implementierung von API und Routes stellt sicher, dass das System die notwendigen Endpunkte für die Anwendung bietet. Dies beinhaltet die Abfrage der aktuellen Kryptowährungspreise, das Anzeigen von Preisunterschieden zwischen den Exchanges und die Berechnung der Gesamtkosten eines potentiellen Trades.
- **SymbolController:** Dieser Controller dient der Handhabung der spezifischen Kryptowährungs Paare. Wenn ein Benutzer beispielsweise die Preisdaten einer bestimmten Kryptowährung anfordert, wird dieser Controller aktiv und bearbeitet die entsprechende Anfrage.
- **Websocket:** Die Websocket Klasse ist dafür zuständig, die Websocket Verbindungen mit den Benutzern herzustellen.
- **NotifyHandler:** Diese Klasse ist dafür zuständig, über die Websocket Verbindungen mit den Benutzern zu kommunizieren.

dungen Nachrichten an den Benutzer zu senden, sobald neue Daten verfügbar sind. Damit soll die Echtzeitanzeige ermöglicht werden.

- **RefresherService:** Diese Klasse ist verantwortlich für das Abrufen leistungintensiver Berechnungen aus der TSDB, insbesondere im Hinblick auf die Preisanalysen wie in den Use-Cases beschrieben. Die Ergebnisse werden dann in der IMDB zwischengespeichert, um sicherzustellen, dass die Anwendung schnell auf aktuelle Daten zugreifen kann, ohne die Berechnungen erneut durchführen zu müssen.
- **InMemoryClient:** Dieser Client dient dem Schreiben und Lesen von Daten in der IMDB für einen schnellen Zugriff.
- **Config:** In diesem Interface werden die Konfigurationsdetails wie z.B. die Datenbankverbindungen gespeichert.
- **TickerInfo & Symbol:** Diese Klassen bilden das Interface für die spezifische Informationen zu den Kryptowährungen.
- **Logger:** Ein essentieller Bestandteil jeder Anwendung. Der Logger protokolliert alle wichtigen Ereignisse, Fehler und Aktivitäten in der Anwendung. Dies erleichtert das Debuggen und die Überwachung des Systems.
- **AppContext:** Dient als zentraler Knotenpunkt für die Anwendung. Es verwaltet und steuert den Zugriff auf verschiedene Dienste wie den Logger, die Datenbankclients und die Konfiguration, um eine konsistente Anwendungslogik zu gewährleisten.

Beziehungen zwischen den Klassen:

- Die StreamApp interagiert mit dem CryptoExchange, um mittels der *fetch\_tickers* Methode Tickerinformationen von den Exchanges zu erhalten. Anschließend werden die Daten mittels MessageQueueClient in die Warteschlange vom Nachrichten Broker eingereiht. Der StorageService ruft diese Daten mit dem MessageQueueClient auf und speichert sie in einer TSDB mittels TSDB-Client ab.
- Die API-Schicht, mit der Route-Klasse und dem SymbolController, stellt die erforderlichen Endpunkte zur Verfügung, über die Benutzer Anfragen stellen und Daten abrufen können.
- Der NotifyHandler arbeitet direkt mit der WebSocket-Klasse zusammen, um Echtzeit-Updates zu ermöglichen. Der NotifyHandler hört auf Daten, die über einen spezifischen Endpunkt kommen, und verwendet die Broadcast-Funktion der WebSocket-Klasse, um Nachrichten an alle verbundenen Clients zu senden.
- Der RefresherService holt sich Job-Details vom JobController, um Datenaktualisierungen durchzuführen. Dabei interagiert der RefresherService mit dem

AppContext um die benötigten Daten abzurufen und abzuspeichern. Wichtig ist, dass der RefresherService eine aktive Rolle in der Kommunikation mit dem NotifyHandler spielt. Sobald neue Datenupdates erkannt werden, ist es die Aufgabe des RefresherService, diese Informationen an den NotifyHandler zu senden, der dann über die WebSocket-Verbindung Echtzeit-Updates an alle verbundenen Clients schickt.

Das konzeptionelle Informationsmodell bietet einen Überblick über die zentralen Datenstrukturen des Systems und deren Beziehungen. Es dient als Grundlage für die anschließende technische Implementierung und gewährleistet, dass alle für die Umsetzung der Use-Cases erforderlichen Daten und Beziehungen berücksichtigt werden. Im nächsten Abschnitt wird das konzeptionelle Architekturmodell, welches den Fokus auf die technische Umsetzung legt, vorgestellt..

### 3.3 Konzeptionelles Architekturmodell

Das in diesem Kapitel vorgestellte Architekturmodell veranschaulicht die technische Struktur und die Kommunikation der verschiedenen Komponenten des geplanten Systems. Dieses Modell wurde mit besonderem Augenmerk auf Skalierbarkeit und Effizienz entwickelt.

Das Architekturmodell orientiert sich an dem UML Deployment Diagramm und wird in Abbildung 3.3 dargestellt. Diese Darstellung bietet einen Einblick in die Verteilung der Softwarekomponenten und zeigt, wie diese Komponenten miteinander interagieren.

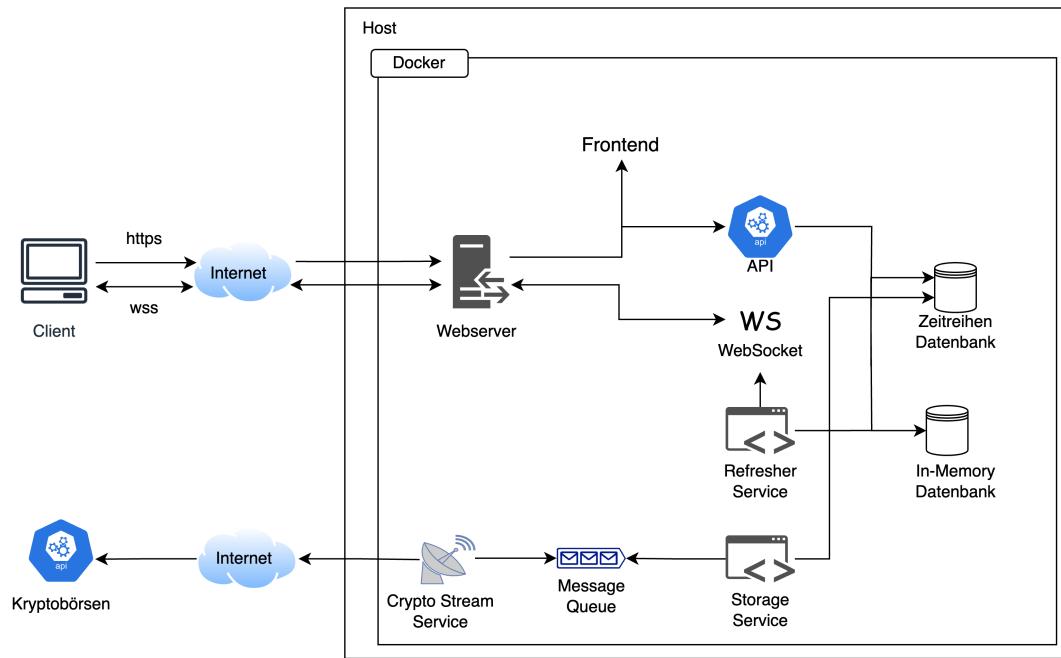


Abbildung 3.3: Konzeptionelles Architekturmodell in Anlehnung an das UML Deployment Diagramm [38]

Die nachfolgende Beschreibung erläutert die einzelnen Elemente des Architekturmodells und deren Zusammenspiel.

Der Hauptbereich des Architekturmodells besteht aus dem Host, in dem alle wesentlichen Services und DB über Docker-Container [35] ausgeführt werden. Docker [35] ermöglicht es, die Services und Abhängigkeiten in isolierten Umgebungen auszuführen, wodurch die Sicherheit und Reproduzierbarkeit des Systems erhöht wird. Im Folgenden wird die Architektur des Systems detailliert beschrieben:

- **Frontend:** Das Frontend ist die Benutzeroberfläche der Web-Anwendung.
- **Crypto Stream Service:** Dieser Dienst ist dafür verantwortlich, Ticker-Informationen von den Exchanges in Echtzeit abzurufen. Durch die Isolierung dieser Funktion kann sich der Service auf das schnelle und zuverlässige Abrufen von Daten konzentrieren. Die abgerufenen Daten werden anschließend in eine Warteschlange vom Nachrichten Broker eingereiht.
- **Storage Service:** Dieser Service liest die Daten aus der Warteschlange aus und speichert sie in einer TSDB.
- **Nachrichten Broker:** Hier werden Daten temporär gespeichert, um sicherzustellen, dass kein Datenverlust auftritt und der Crypto Stream Service nicht blockiert wird.

- **TSDB:** Diese TSDB speichert die Kryptowährungspreise in chronologischer Reihenfolge ab. Sie eignet sich besonders für die Analyse von Daten über einen bestimmten Zeitraum, wie z.B. Preishistorien von Kryptowährungen.
- **IMDB:** Hier werden Berechnungsergebnisse und andere zeitkritische Daten gespeichert, die für einen schnellen Zugriff benötigt werden.
- **API:** Die API ermöglicht es dem Frontend, Daten aus der TSDB oder der IMDB abzurufen. Sie stellt sicher, dass die Daten in einem strukturierten und konsistenten Format bereitgestellt werden.
- **WebSocket:** Dieser Dienst stellt eine Verbindung zum Client her und ermöglicht das Senden von Echtzeit-Updates, wie z.B. Preisänderungen der Kryptowährungen, direkt an die Benutzeroberfläche.
- **Refresher Service:** In regelmäßigen Abständen werden Analysen und Berechnungen auf den Daten in der TSDB durchgeführt. Ergebnisse dieser Berechnungen werden in einer IMDB gespeichert. Dies beschleunigt die Datenabfrage, da aufwendige Berechnungen nicht bei jeder API Anfrage wiederholt werden müssen.
- **Webserver:** Der Webserver leitet eingehende Anfragen an den entsprechenden Service weiter.

Das konzeptionelle Architekturmodell verdeutlicht die technische Umsetzung und Interaktion der verschiedenen Systemkomponenten. Dabei wurde besonderer Wert darauf gelegt, eine modulare und skalierbare Architektur zu schaffen, die sowohl aktuelle als auch zukünftige Anforderungen effizient unterstützt.

## Zusammenfassung

Kapitel 3 hat einen detaillierten Blick auf die konzeptionelle Modellierung und das Design geworfen. Angefangen beim konzeptionellen Anwendungsfallmodell, welches die Bedürfnisse und Interaktionen der Benutzer in den Mittelpunkt stellt, über das konzeptionelle Informationsmodell, das die Funktionen und Datenstrukturen des Systems klärt, bis hin zum konzeptionellen Architekturmodell, das die technische Umsetzung und Interaktion der Systemkomponenten beleuchtet. Dieser ganzheitliche Ansatz gewährleistet, dass das FZ2T erfüllt wird und das System sowohl aus Benutzer- als auch aus technischer Perspektive gut durchdacht und auf zukünftige Anforderungen vorbereitet ist.

# Kapitel 4

## Prototypische Implementierung

In diesem Kapitel erfolgt die Darstellung der prototypischen Implementierung des Projekts, die im Einklang mit FZ3S steht. Die Realisierung dieses Ziels beinhaltet die Entwicklung eines Prototypen, der die konzeptionellen Designs und Methoden aus Kapitel 3 praktisch umsetzt und deren Machbarkeit demonstriert. Die Implementierung beginnt mit der Einrichtung der Entwicklungsinfrastruktur, gefolgt von detaillierten Erläuterungen zu jedem Service, der für das System entwickelt wurde und endet mit der Bereitstellung der Anwendung.

Zu Beginn der Implementierungsphase wird eine robuste Entwicklungsinfrastruktur aufgebaut, damit ein effizientes Arbeiten gewährleistet ist und die verschiedenen Komponenten des Systems nahtlos miteinander integrieren können.

### 4.1 Die Datenbanken und den Nachrichten Broker hochfahren

Für die Verwaltung von Zeitreihendaten wird InfluxDB [39] verwendet, als IMDB-Datenbank dient Redis [40] und als Message Queue wird RabbitMQ [41] eingesetzt. Die Einrichtung dieser Dienste erfolgt mit Docker [35] und wird durch eine Docker-Compose-Datei [35] automatisiert (siehe Listing 4.1). In der Docker-Compose-Datei [35] werden drei Services definiert: InfluxDB [39], Redis [40] und RabbitMQ [41]. Bei InfluxDB [39] und Redis [40] wird jeweils das neueste (latest) Docker-Image [35] verwendet, was bedeutet, dass stets die aktuellste verfügbare Version dieser Software genutzt wird. RabbitMQ [41] hingegen verwendet das spezielle Management-Image, welches neben den Standardfunktionen eine grafische Benutzeroberfläche und zusätzliche Verwaltungstools bereitstellt.

Für die Netzwerkkommunikation wird in jedem Service eine Portbindung konfiguriert. Dabei wird Datenverkehr der internen Ports an den externen Port gebunden. Diese externen Ports können über Umgebungsvariablen festgelegt werden und ermöglichen den Zugriff auf die Services auch außerhalb des Docker-Netzwerks [35]. Dieser Schritt ist nur in der Entwicklungsumgebung notwendig, in der Produktionsumgebung kommunizieren die Services nur direkt über das interne Netzwerk.

Die Definition von Volumes in der Docker-Compose-Datei [35] ist entscheidend für die Datenpersistenz. Durch diese Volumes werden Daten dauerhaft gespeichert und die Daten bleiben auch nach einem Neustart der Container erhalten.

Schließlich wird ein spezielles Netzwerk namens „CryptoTracker“ aufgebaut. Dieses Netzwerk ermöglicht die Kommunikation zwischen den verschiedenen Services innerhalb des Netzwerkes. Somit müssen keine Ports nach außen hin geöffnet werden. Dies erhöht die Sicherheit des Systems (vgl. [35]).

Listing 4.1: Docker Compose-Datei [35] zur Einrichtung der DB und des Nachrichten Broker

```

1 services:
2   influxdb:
3     image: influxdb:latest
4     container_name: influxdb
5     restart: always
6     ports:
7       - ${INFLUXDB_HOST}:${INFLUXDB_PORT}:8086
8     volumes:
9       - $PWD/influxdb:/etc/influxdb2
10      - influxdb:/var/lib/influxdb2
11      - /etc/timezone:/etc/timezone:ro
12      - /etc/localtime:/etc/localtime:ro
13     networks:
14       - cryptoTracker
15
16   redis:
17     image: redis:latest
18     container_name: redis
19     restart: always
20     networks:
21       - default
22     ports:
23       - ${REDIS_HOST}:${REDIS_PORT}:6379
24     volumes:
25       - redis:/data
26       - /etc/timezone:/etc/timezone:ro
27       - /etc/localtime:/etc/localtime:ro
28     environment:
```

```

29      - REDIS_PASSWORD=${REDIS_PASSWORD}
30    command: ["redis-server", "--requirepass", "${REDIS_PASSWORD}"]
31
32 rabbitmq:
33   image: rabbitmq:management
34   container_name: rabbitmq
35   environment:
36     - RABBITMQ_DEFAULT_USER=${RABBITMQ_USERNAME}
37     - RABBITMQ_DEFAULT_PASS=${RABBITMQ_PASSWORD}
38   volumes:
39     - rabbitmq:/var/lib/rabbitmq/
40     - $PWD/rabbitmq/log/:/var/log/rabbitmq
41   ports:
42     - ${RABBITMQ_HOST}:${RABBITMQ_PORT}:5672
43     - 127.0.0.1:15672:15672
44   networks:
45     - cryptoTracker
46
47
48 volumes:
49 influxdb:
50   external: false
51 redis:
52   external: false
53 rabbitmq:
54   external: false
55
56 networks:
57 cryptoTracker:
58   driver: bridge

```

Vor der Ausführung des Befehls `docker-compose up -d` ist es wichtig sicherzustellen, dass alle Umgebungsvariablen korrekt gesetzt sind. Durch den Befehl werden anschließend die Container heruntergeladen und gestartet.

In den folgenden Abschnitten werden die Entwicklung und Implementierung der verschiedenen benötigten Services im Detail beschrieben. Dabei wird sowohl auf die Verwendung spezifischer Bibliotheken und Frameworks als auch auf die Konnektivität und Integration der einzelnen Services eingegangen.

## 4.2 Der Stream Service

Der Streamservice nimmt eine zentrale Position im System ein, da er in Echtzeit Ticker-Informationen von den Exchanges abruft. Er wird in Python [27] entwickelt, um eine nahtlose Integration mit der CCXT-Bibliothek [26] zu gewährleisten (vgl. [26]).

Zunächst wird in Unterabschnitt 4.2.1 beschrieben, wie der Stream Service die Konfigurationsdaten lädt und validiert, was eine Grundvoraussetzung für den sicheren und zuverlässigen Betrieb darstellt. Anschließend wird im Unterabschnitt 4.2.2 der Verbindungsaufbau zu RabbitMQ [41] gezeigt, die für die Übermittlung der Ticker-Informationen an die Queue wichtig ist. Schließlich wird im Unterabschnitt 4.2.3 gezeigt, wie die Ticker-Informationen abgerufen und in die Queue geschrieben werden, womit der Kreislauf der Datenerfassung und -verarbeitung geschlossen wird.

### 4.2.1 Konfigurationsdaten laden und validieren

Zunächst werden die erforderlichen Konfigurationsdaten aus den Umgebungsvariablen geladen und validiert (siehe Listing 4.2). Damit wird sichergestellt, dass im späteren Verlauf ein korrekter Zugriff auf diese Daten möglich ist. Ein praktisches Beispiel hierfür ist die Nutzung der Login-Informationen, um eine Verbindung zur RabbitMQ [41] herzustellen.

Listing 4.2: Konfiguration laden und validieren

```
1 from streamservice.config import Config
2 Config.validate()
```

Da nun die Konfigurationsdaten verarbeitet werden können, wird als nächstes die Verbindung zu RabbitMQ [41] hergestellt.

### 4.2.2 Verbindung zur RabbitMQ herstellen

Die Verbindung zu RabbitMQ [41] wird benötigt, um die Ticker-Informationen in eine Warteschlange zu schreiben. Für die Herstellung dieser Verbindung wird die pika [42] Bibliothek eingesetzt. Dazu wird eine Klasse MessageQueueClient erstellt, welche die Verbindung zu RabbitMQ [41] herstellt. Die Funktion connect wird verwendet, um die Verbindung zu RabbitMQ herzustellen (siehe Listing 4.3). Dabei wird ein Exchange und eine Warteschlange (Queue) erstellt. Die Queue wird dann an den Exchange gebunden, wobei ein spezifischer Routing-Schlüssel die Weiterleitung von Nachrichten an die richtige Queue steuert. Nachrichten werden an den Exchange gesendet, der dann basierend auf dem Routing-Schlüssel entscheidet, in welche Queue jede Nachricht geleitet wird.

Listing 4.3: Verbindung zur RabbitMQ herstellen

```
1 import pika
2 class MessageQueueClient:
3     def __init__(self, config: Type[Config]):
```

```

4     self.config = config
5     self.connection: Optional[BlockingConnection] = None
6     self.channel: Optional[BlockingChannel] = None
7
8     def connect(self) -> None:
9         credentials = pika.PlainCredentials(
10             self.config.RABBITMQ_USERNAME, self.config.RABBITMQ_PASSWORD
11         )
12         self.connection = pika.BlockingConnection(
13             pika.ConnectionParameters(
14                 self.config.RABBITMQ_HOST,
15                 self.config.RABBITMQ_PORT,
16                 "/",
17                 credentials,
18             )
19         )
20         self.channel = self.connection.channel()
21         self.channel.exchange_declare(
22             exchange=self.config.RABBITMQ_EXCHANGE,
23             exchange_type="topic",
24             durable=True,
25         )
26         self.channel.queue_declare(queue="kraken_queue", durable=True)
27         self.channel.queue_bind(
28             queue="kraken_queue",
29             exchange=self.config.RABBITMQ_EXCHANGE,
30             routing_key="kraken_routing_key",
31         )
32
33
34 mq = MessageQueueClient(config=Config)
35 mq.connect()

```

Zur Klasse `MessageQueueClient` wird jetzt noch eine Methode `write_tickers` hinzugefügt. Mit dieser Methode werden die abgerufenen Ticker-Informationen in ein JSON-Format geparsed und anschließend in die deklarierte RabbitMQ-Queue [41] geschrieben (siehe Listing 4.4).

Listing 4.4: Klassenmethode um die Ticker Informationen in die Queue zu schreiben

```

1 class MessageQueueClient:
2     ...
3     def write_tickers(self, tickers: Ticker) -> None:
4         for _, ticker_data in tickers.items():
5             message = json.dumps(ticker_data)
6             self.channel.basic_publish(
7                 exchange=self.config.RABBITMQ_EXCHANGE,
8                 routing_key="kraken_routing_key",
9                 body=message)

```

Die Verbindung zu RabbitMQ [41] ist nun hergestellt und die Ticker-Informationen können in die Queue geschrieben werden. Jetzt muss nur noch die Verbindung zu einem Exchange hergestellt werden und die Ticker-Informationen abgerufen werden.

### 4.2.3 Ticker-Informationen abrufen und in die Queue schreiben

Mit Hilfe der CCXT-Bibliothek [26] wird eine Verbindung zu einem spezifischen Exchange, in diesem Fall Kraken [16], hergestellt. Anschließend werden die aktuellen Ticker-Informationen von dem Exchange abgerufen und in die Queue eingereiht.

Listing 4.5: Ticker-Informationen über die CCXT-Bibliothek abrufen

```

1 import ccxt
2 class CryptoExchange:
3     def __init__(self, exchange_name: str):
4         self.exchange = getattr(ccxt, exchange_name)()
5
6     def fetch_tickers(self) -> Ticker:
7         return self.exchange.fetch_tickers()
8
9 kraken = CryptoExchange(exchange_name="kraken")
10 tickers = kraken.fetch_tickers()
11 mq.write_tickers(tickers=tickers)

```

Nun stehen die Nachrichten der Ticker-Informationen in der Queue und warten darauf, von dem StorageService abgerufen zu werden.

## 4.3 Der Storage-Service

Der Storage-Service ist nun für das effiziente abspeichern von Ticker-Informationen aus der Queue in die InfluxDB [39] zuständig. Um diese Aufgabe zu erfüllen, werden die erforderlichen Konfigurationsdaten geladen und die Verbindung zu RabbitMQ [41] und InfluxDB [39] hergestellt, wie im Unterabschnitt 4.3.1 dargestellt. Anschließend wird eine Goroutine [5] eingesetzt, um kontinuierlich die Daten aus der Warteschlange zu lesen und in InfluxDB [39] abzuspeichern, wie im Unterabschnitt 4.3.2 dargestellt. Dieser Service und alle nachfolgenden Backend Services werden in Go [5] entwickelt.

### 4.3.1 Konfigurationsdaten laden und eine Verbindung zu RabbitMq und InfluxDB herstellen

Zu Beginn werden die benötigten Konfigurationsdaten geladen, damit anschließend auf die benötigten Daten zugegriffen werden kann. Danach kann der Service eine Verbindung zu RabbitMQ [41] und InfluxDB [39] herstellen, um Nachrichten aus der Queue zu lesen und in die InfluxDB [39] zu speichern, wie in Listing 4.6 dargestellt.

Listing 4.6: Konfigurationsdaten laden und eine Verbindung zu RabbitMQ [41] und InfluxDB [39] herstellen

```

1 def main(){
2     cfg, err := config.Read()
3     if err != nil {
4         log.Fatalf("%s: %s", "Failed to read configuration", err)
5     }
6     influxClient, err := influxdb.NewClient(*cfg.InfluxDB)
7     rmq, err := messaging.NewRabbitMQ(*cfg.RabbitMQ)

```

Da nun die Verbindung zur RabbitMQ [41] steht, muss der Service als nächstes die Nachrichten aus der Queue lesen.

### 4.3.2 Golang-Routine für das Lesen und Speichern von Daten

Listing 4.7 zeigt, wie eine Goroutine [5] kontinuierlich überprüft, ob neue Nachrichten in der RabbitMQ-Queue [41] vorhanden sind. Diese Nachrichten werden gelesen, von ihrem JSON-Format in ein Go-Struktur [5] umgewandelt und dann werden sie für die Speicherung in InfluxDB [39] vorbereitet. Dies beinhaltet das Setzen von Tags und Feldern und das Schreiben der Daten in die DB.

Listing 4.7: Nachrichten lesen und in die InfluxDB [39] speichern

```

1 go func() {
2     deliveries, err := rmq.Receive("kraken_queue")
3
4     for d := range deliveries {
5         // Nachrichtenverarbeitung...
6         var msg model.Message
7         json.Unmarshal(d.Body, &msg)
8
9         // Konvertieren des Timestamps aus der Nachricht
10        timestamp := time.Unix(0, msg.Timestamp*time.Millisecond)
11
12        // Tags und Felder für InfluxDB vorbereiten
13        tags := map[string]string{
14            "exchange": "kraken",
15            "pair":     msg.Symbol,

```

```

16     }
17     fields := map[string]interface{}{
18         "high": msg.High,
19         "low": msg.Low,
20         "bid": msg.Bid,
21         "ask": msg.Ask,
22         "last": msg.Last,
23     }
24
25     // Daten in InfluxDB schreiben
26     influxClient.WriteData("crypto_data", tags, fields, timestamp)
27 }
28 }()

```

Um sicherzustellen, dass der Storage-Service ununterbrochen läuft und ständig auf eingehende Nachrichten wartet, wird ein forever-Kanalobjekt erstellt, damit der Hauptthread in einem Wartezustand hält, wie in Listing 4.8 dargestellt. Dies verhindert, dass das Programm vorzeitig beendet wird und erlaubt der Goroutine [5], kontinuierlich im Hintergrund zu arbeiten.

Listing 4.8: Den Hauptthread im Wartemodus halten

```

1 forever := make(chan struct{})
2 log.Printf(" [*] Waiting for messages. To exit press CTRL+C")
3 <-forever

```

Durch diese Implementierung kann der Service effektiv laufen und auf Nachrichten warten, ohne jemals zu stoppen, es sei denn, er wird manuell durch den Benutzer beendet (z.B. durch Drücken von **CTRL** + **C**). Dadurch wird sichergestellt, dass keine Ticker-Informationen übersehen werden und alle Daten korrekt in InfluxDB [39] gespeichert werden.

Als nächster Schritt wird die Implementierung des Websockets dargestellt, damit die Ticker-Informationen an die Clients gesendet werden können.

## 4.4 Der Websocket

Der WebSocket spielt eine entscheidende Rolle in der Echtzeitkommunikation zwischen dem Server und den verbundenen Clients. Der Websocket ist dafür zuständig, Preisaktualisierung an den verbundenen Clients zu senden. Um den Websocket zu implementieren wird zuerst eine Server Instanz benötigt, wie im Unterabschnitt 4.4.1 dargestellt. Im Unterabschnitt 4.4.2 werden anschließend die WebSocket-Funktionen definiert, die für die Handhabung der Client-Kommunikation und das Senden von Preisaktualisierungen verantwortlich sind. Sobald die grundlegenden Funktionen und

die Serverinstanz eingerichtet sind, wird im Unterabschnitt 4.4.3 beschrieben, wie der WebSocket-Server eingerichtet und hochgefahren wird. Zuletzt wird im Unterabschnitt 4.4.4 die Funktionsweise des Websockets illustriert.

#### 4.4.1 Server Instanz

Der erste Schritt besteht darin, eine Instanz des Servers zu erstellen, der sowohl die Verbindungen `conns` als auch die Abonnements `subscriptions` speichert (siehe Listing 4.9). Durch diese Struktur kann der Server jederzeit feststellen, welche Clients verbunden sind und welche Preis-Updates sie abonniert haben.

Listing 4.9: Websocket Server Instanz

```

1 type Server struct {
2     conns      map[*websocket.Conn]bool
3     subscriptions map[string]map[*websocket.Conn]bool
4 }
5
6 func NewServer() *Server {
7     return &Server{
8         conns:      make(map[*websocket.Conn]bool),
9         subscriptions: make(map[string]map[*websocket.Conn]bool),
10    }
11 }
```

Da nun das Grundgerüst des Servers erstellt wurde, können die Funktionen des Websockets implementiert werden.

#### 4.4.2 Websocket Funktionen

Die `handleWS`-Funktion ist dafür verantwortlich, eine normale Hypertext Transfer Protocol (HTTP)-Verbindung auf einen WebSocket-Upgrade zu aktualisieren, wie in Listing 4.10 zu sehen ist. Nach dem Upgrade initiiert sie die Lese-Schleife `readLoop`, die fortlaufend die von den Clients gesendeten Nachrichten liest.

Listing 4.10: Websocket Handler

```

1 func (s *Server) handleWS(w http.ResponseWriter, r *http.Request) {
2     upgrader := websocket.Upgrader{}
3     ws, err := upgrader.Upgrade(w, r, nil)
4     if err != nil {
5         fmt.Println("upgrade err:", err)
6         return
7     }
8     fmt.Println("new incoming connection from client", ws.RemoteAddr())
9     s.readLoop(ws)}
```

In der `readLoop`-Funktion werden die Nachrichten gelesen, die der Client sendet, um sich beim WebSocket anzumelden, dargestellt in Listing 4.11. Hierbei gibt der Client die Währungssymbole an, für die der Client Preis-Updates erhalten möchte. Der Server fügt dann den WebSocket des Clients zur Liste der Abonnenten für jedes angegebene Währungssymbol hinzu.

Listing 4.11: Abonnements verwalten

```

1 func (s *Server) readLoop(ws *websocket.Conn) {
2     for {
3         _, msg, _ := ws.ReadMessage()
4
5         var data map[string]interface{}
6         json.Unmarshal(msg, &data)
7
8         params := data["symbols"].([]interface{})
9
10        for _, p := range params {
11            currency := p.(string)
12            if s.subscriptions[currency] == nil {
13                s.subscriptions[currency] = make(map[*websocket.Conn]bool)
14            }
15            s.subscriptions[currency][ws] = true
16            fmt.Printf("Subscribed to %s \n", currency)
17        }
18    }
19 }
```

Listing 4.12 zeigt nun, wie der Server mit der `broadcast`-Funktion Nachrichten an alle Abonnenten eines bestimmten Währungssymbols senden kann.

Listing 4.12: Nachrichten an alle Clients senden

```

1 func (s *Server) broadcast(currency string, data []byte) {
2     for ws := range s.subscriptions[currency] {
3         go func(ws *websocket.Conn) {
4             ws.WriteMessage(websocket.TextMessage, data)
5         }(ws)
6     }
7 }
```

Die `notifyPriceChange`-Funktion ist schließlich dafür gedacht, Nachrichten im JSON Format zu erhalten und sie an die `broadcast`-Funktion weiterzuleiten, sodass alle Abonnenten benachrichtigt werden (siehe Listing 4.14).

Listing 4.13: Mitteilungen versenden

```

1 func (s *Server) notifyPriceChange(w http.ResponseWriter, r *http.Request) {
2     bodyBytes, _ := io.ReadAll(r.Body)
3 }
```

```

4 var js map[string]interface{}
5 json.Unmarshal(bodyBytes, &js)
6 // Pair aus dem validierten JSON extrahieren fr den Broadcast.
7 symbol := js["symbol"].(string)
8
9 s.broadcast(symbol, bodyBytes)
10 }

```

Der Websocket ist soweit fertig implementiert und kann folglich gestartet werden.

#### 4.4.3 Einrichtung und Hochfahren des WebSocket-Servers

In der `main`-Funktion wird ein neuer WebSocket-Server mit der Methode `NewServer` initialisiert. Anschließend werden zwei Routen definiert: „/ws“, welche den WebSocket-Handler verwendet, und „/notify“, welche den Handler für Echtzeitbenachrichtigungen verwendet. Schließlich wird der Server gestartet und hört auf dem Port 3008 auf eingehende Anfragen (siehe Listing 4.14).

Listing 4.14: Start des Websockers

```

1 func main() {
2     server := NewServer()
3     http.HandleFunc("/ws", server.handleWS)
4     http.HandleFunc("/notify", server.notifyPriceChange)
5     http.ListenAndServe(":3008", nil)
6 }

```

Um die Funktionsweise des Websockets zu veranschaulichen, wird im folgenden Abschnitt eine Verbindung zum WebSocket hergestellt sowie ein Währungspaar abonniert und anschließend eine Nachricht an alle Abonnenten gesendet.

#### 4.4.4 Illustration der WebSocket-Funktionsweise

Nachdem der WebSocket-Server erfolgreich gestartet wurde, kann nun eine Verbindung dazu hergestellt werden. Zur Darstellung wird hier mithilfe des Python [27] Interpreter und der `websocket`-Bibliothek eine Verbindung zum lokalen WebSocket unter dem Pfad `/ws` aufgebaut (siehe Listing 4.15).

Listing 4.15: Websocket Verbindung herstellen

```

1 >>> import json
2 >>> from websocket import create_connection
3 >>> ws = create_connection("ws://localhost:3008/ws")
4 >>> ws.send(json.dumps({"symbols": ["BTC/USD"]}))
5 >>> ws.recv()

```

Nach dem Senden der Abonnement-Nachricht informiert der WebSocket-Server über die neu eingegangene Verbindung und den abonnierten Währungskanal. Um jetzt beispielsweise eine Preisaktualisierung für BTC/USD zu simulieren und an alle Abonnenten dieses Symbols zu senden, wird ein curl-Befehl verwendet, der eine JSON-Nachricht an den /notify-Handler sendet (siehe Listing 4.16).

Listing 4.16: Preisaktualisierung simulieren

```
1 curl --location 'http://localhost:3008/notify' \
2 --header 'Content-Type: application/json' \
3 --data '{ "symbol":"BTC/USD", "price":25000 }'
```

Sobald die Nachricht gesendet wird, empfängt der Python-Interpreter die Preisaktualisierung und zeigt sie, wie in Abbildung 4.1 zu sehen ist, an.

The screenshot shows two terminal windows. The left window shows a Go server running with the command `go run cmd/thesis_script/websocket/main.go`. It receives a connection from a client and subscribes to the "BTC/USD" channel. The right window shows a curl command being run to send a JSON message to the server. The message contains the symbol "BTC/USD" and a price of 25000. The Go server then prints the received message to the console.

```
~/Bachelorarbeit/app/services > python
Python 3.11.5 (main, Aug 24 2023, 15:09:45) [Clang 14.0.3 (clang-1400.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information
>>> import json
>>> from websocket import create_connection
>>> ws = create_connection("ws://localhost:3008/ws")
>>> ws.send(json.dumps({'symbol': "BTC/USD"}))
>>> ws.recv()
'{"symbol": "BTC/USD", "price": 25000}'
>>>
```

```
~/projekte/Bachelorarbeit/app > curl --location 'http://localhost:3008/notify' \
--header 'Content-Type: application/json' \
--data '{ "symbol":"BTC/USD",
          "price":25000
}'
```

Abbildung 4.1: Illustration des Websockets

Da nun der Websocket implementiert ist, kann nun der Refresher Service programmiert werden, welcher den Websocket verwendet um die Preisaktualisierungen an die verbundenen Clients zu senden.

## 4.5 Der Refresher Service

Der RefresherService dient der periodischen Aktualisierung von Daten, insbesondere im Hinblick auf die Preisanalysen. Durch die Verwendung dieses Services wird sichergestellt, dass umfangreiche Berechnungen nur einmalig ausgeführt und die Ergebnisse dann in Redis [40] zwischengespeichert werden. Dies beschleunigt die Datenabfrage erheblich, da diese Berechnungen nicht bei jeder API-Anfrage erneut durchgeführt werden müssen.

Im Unterabschnitt 4.5.1 wird zuerst ein JobController definiert, der für das zeitliche Management der Datenaktualisierung zuständig ist. Weiterführend wird im Unterabschnitt 4.5.2 die Datenabfrage und -verarbeitung behandelt, die für die Bereitstellung der aktuellen Preisinformationen erforderlich ist. Schließlich wird im Unterabschnitt 4.5.3 die Hauptausführung und Initialisierung des Refresher Services beschrieben.

### 4.5.1 Den JobController definieren

Der Kern des RefresherService ist der JobController, welcher die Kontrolle über das Abrufen von Daten in regelmäßigen Intervallen besitzt. Mit Hilfe der Funktion FetchDataWithInterval, wie in Listing 4.17 dargestellt, kann ein beliebiger Job in einem definierten Intervall ausgeführt werden.

Listing 4.17: Jobs im Intervall ausführen

```

1 type JobController struct {
2     applicationContext *appcontext.AppContext
3 }
4
5 func NewJobController(applicationContext *appcontext.AppContext) *JobController {
6     return &JobController{
7         applicationContext: applicationContext,
8     }
9 }
10
11 func (s *JobController) FetchDataWithInterval(job func(), interval time.Duration) {
12     for {
13         start := time.Now()
14         job()
15         elapsed := time.Since(start)
16         s.applicationContext.Logger.Println("Finishing Fetching data at", time.Now(), " took",
17             elapsed)
18         if elapsed < interval {
19             s.applicationContext.Logger.Println("Waiting for", interval-elapsed)
20             time.Sleep(interval - elapsed)
21         }
22     }
23 }
```

Als nächstes können jetzt Jobs definiert werden, welche in regelmäßigen Abständen ausgeführt werden sollen.

### 4.5.2 Datenabfrage und -verarbeitung

Als erstes wird eine zeitintensive InfluxDB-Abfrage [39] benötigt, die regelmäßig durchgeführt werden soll. Für eine bessere Verständlichkeit wird in Listing 4.18 eine vereinfachte und kurze Abfrage präsentiert, welche den Ablauf verdeutlicht. Die gegebene Abfrage ruft den letzten Preis aus der InfluxDB [39] für jedes Symbol ab und berechnet anschließend den Durchschnittspreis der Exchanges.

Listing 4.18: InfluxDB [39] Abfrage

```

1 from(bucket: "CryptoTrackerBucket")
2   |> range(start: -1m)
3   |> filter( fn: (r) => r._measurement == "crypto_data" and (r._field == "last" ), )
4   |> last()
5   |> group(columns: ["pair"])
6   |> mean()
7   |> yield(name:"prices")

```

Nachdem die Abfrage formuliert ist, wird eine Funktion definiert, um diese zu verarbeiten (siehe Listing 4.19). Dabei wird die Abfrage an die InfluxDB [39] gesendet, interpretiert und das Ergebnis in der Variable result gespeichert. Anschließend wird für Redis [40] ein Context initiiert, damit gewährleistet ist, dass alle Ressourcen und Prozesse während der Lebensdauer korrekt gehandhabt werden. Danach wird eine Pipe für Redis erzeugt. Diese Pipe agiert als Puffer, in denen mehrere Redis Abfragen zwischengespeichert werden können und sie anschließend alle gleichzeitig an Redis übermittelt werden. Dadurch wird der Job schneller durchgeführt und die Netzwerkauslastung verringert.

Die Ergebnisse der InfluxDB [39] Abfrage werden nun systematisch durchlaufen. Hierbei werden jeweils das Kryptowährungspaar und der zugehörige Preis extrahiert. Daraus wird ein Symbolobjekt generiert und in das JSON-Format umgewandelt. Diese Daten werden im Anschluss über den WebSocket versendet.

Zum Abschluss des Ablaufs wird die vorbereitete Pipe für Redis ausgeführt. Dies führt zur Speicherung aller in der Pipe bereitgestellten Daten in Redis.

Listing 4.19: Job Funktion

```

1 func (c *JobController) FetchSymbolPriceChanges() {
2   // Logic to fetch symbols and data from InfluxDB
3   result, _ := c.appContext.QueryInfluxDB("queries/prices.flux")
4   // Create a context.Context with a timeout of 2 seconds
5   ctx, cancel := context.WithTimeout(context.Background(), 2*time.Second)
6   // Make sure to clean up the context at the end
7   defer cancel()
8
9   // Create a pipeline for redis HSet
10  pipe := c.appContext.Redis.Pipeline()
11
12  // Iterate over the QueryTableResult and extract the data
13  for result.Next() {
14    pair, _ := result.Record().ValueByKey("pair").(string)
15    price := result.Record().ValueByKey("_value").(float64)
16    symbolData := symbol.Symbol{
17      Pair: pair,
18      Price: price,

```

```

19 }
20
21 // Convert symbolData to json
22 symbolJsonData, _ := json.Marshal(symbolData)
23
24 // Add data to the pipeline. It will not be executed until pipe.Exec() is called.
25 pipe.Set(ctx, pair, symbolJsonData, 0)
26
27 http.Post("http://127.0.0.1:3008/notify", "application/json", bytes.NewBuffer(
28     symbolJsonData))
29 }
30 // Execute all buffered commands at once.
31 pipe.Exec(ctx)
32 }
```

Hiermit ist der erste Job definiert und es kann die Hauptfunktion des RefresherService implementiert werden.

#### 4.5.3 Hauptausführung und Initialisierung

Die Hauptfunktion initialisiert die Konfiguration, erstellt den ApplicationContext für die Verbindung mit Redis [40] und InfluxDB [39] und startet den JobController (siehe Listing 4.20). Der Job wird dann im vorgegebenen Intervall ausgeführt.

Listing 4.20: Hauptausführung des Refresher Service

```

1 func main() {
2     cfg, _ := config.Read()
3
4     applicationContext, _ := appcontext.NewApplicationContext(cfg)
5     defer applicationContext.Close()
6
7     job := jobs.NewJobController(applicationContext)
8     go job.FetchDataWithInterval(job.FetchSymbolPriceChanges, 5*time.Second)
9
10    // Hauptgoroutine am Leben halten, um die Hintergrundgoroutine weiter auszuführen
11    forever := make(chan struct{})
12    log.Printf(" [*] Executing Jobs. To exit press CTRL+C")
13    <-forever
14 }
```

Im nächsten Schritt wird die API entwickelt, welche die Schnittstelle für die Nutzer darstellt.

## 4.6 Die API Schnittstelle

Für die Entwicklung der API, die den Zugriff auf die gespeicherten Daten ermöglicht, wird GoFiber [43] als Framework eingesetzt. Dies gewährleistet eine schnelle und effiziente Handhabung der Daten. Im Unterabschnitt 4.6.1 wird die Initialisierung und das Hochfahren der API beleuchtet und im Unterabschnitt 4.6.2 wird die API getestet, um die korrekte Implementierung sicherzustellen.

### 4.6.1 Initialisierung und hochfahren der API

Zu Beginn wird diesmal zuerst die Hauptfunktion definiert, die für die Initialisierung und den Start der API zuständig ist, wie in Listing 4.21 dargestellt. Es werden die Konfigurationsdaten geladen und der bekannte ApplicationContext erstellt. Mit `fiber.New()` wird eine neue Instanz des GoFiber [43] Web-Frameworks erstellt, um den Aufbau und die Handhabung des Web-Servers zu erleichtern. Die Methode `app.Get("/symbols", func(c *fiber.Ctx) error ...)` definiert eine Route, auf die, über einen HTTP GET-Request, zugegriffen werden kann und legt die zugehörige Logik fest, die beim Zugriff auf dieser Route ausgeführt wird. Schließlich sorgt `app.Listen(":3003")` dafür, dass der Webserver auf dem angegebenen Port, in diesem Fall 3003, gestartet wird und auf eingehende Anfragen wartet.

Listing 4.21: Initialisierung und Start der API

```

1 func main(){
2     cfg, _ := config.Read()
3     appContext, _ := appcontext.NewApplicationContext(cfg)
4     defer appContext.Close()
5
6     app := fiber.New()
7     app.Get("/symbols", func(c *fiber.Ctx) error {
8         return getSymbols(c, appContext)
9     })
10    log.Fatal(app.Listen(":3003"))
11 }
```

Jetzt muss nur noch die Funktion `getSymbols` definiert werden, welche die Logik für den `/symbols` Endpoint enthält (siehe Listing 4.24). Diese Funktion holt alle Symbolinformationen aus Redis [40] und gibt sie als JSON zurück.

Diese Funktion führt mehrere Schritte aus:

1. Erzeugung eines Kontexts mit einem Timeout von 100 Millisekunden.
2. Erstellen einer Pipeline für Redis-Abfragen [40].

3. Abfrage aller Schlüssel in Redis [40].
4. Durchlaufen der Schlüssel und Abruf der entsprechenden Werte mittels der Pipeline.
5. Umwandlung der abgerufenen Daten in Go-Strukturen [5] und Erzeugung eines JSON-Responses.

Listing 4.22: Symbole aus Redis [40] abfragen

```

1 func getSymbols(c *fiber.Ctx, applicationContext *appcontext.AppContext) error {
2     // Create a context.Context with a timeout of 100 milliseconds
3     ctx, cancel := context.WithTimeout(context.Background(), 100*time.Millisecond)
4     // Make sure to clean up the context at the end
5     defer cancel()
6     pipe := applicationContext.Redis.Pipeline()
7
8     keys, _ := applicationContext.Redis.Keys(ctx, "*").Result()
9     // Create a slice to hold your data points
10    for _, key := range keys {
11        pipe.Get(ctx, key)
12    }
13
14    cmd, _ := pipe.Exec(ctx)
15    data := make([]symbol.Symbol, 0, len(cmd))
16    for _, cmd := range cmd {
17        val, _ := cmd.(*redis.StringCmd).Bytes()
18        var symbolData symbol.Symbol
19        json.Unmarshal(val, &symbolData)
20        data = append(data, symbolData)
21    }
22
23    return c.JSON(data)
24 }
```

Anschließend kann die API bereitgestellt werden.

## 4.6.2 API-Abfrage Testen

Hier noch ein praktisches Beispiel, zu sehen in Abbildung 4.3, wie man nun die API mit einem einfachen curl-Command testen kann.

```

go run cmd/thesis_script/api/main.go
Fiber v2.49.2
http://127.0.0.1:3003
(bound on host 0.0.0.0 and port 3003)
Handlers ..... 2 Processes ..... 1
Prefork ..... Disabled PID ..... 38657

curl -s http://localhost:3003/symbols | jq '.[] | {pair, price}' | head -n 20
[
  {"pair": "SPEL/EUR", "price": 0.000428},
  {"pair": "AMPEUR", "price": 0},
  {"pair": "FTM/USD", "price": 0.2239499999999998},
  {"pair": "POWR/EUR", "price": 0.2117},
  {"pair": "AAVE/BTC", "price": 0.00272735}
]

```

Abbildung 4.2: API Abfrage testen

Nachdem die Funktionsweise der API detailliert betrachtet und somit alle Backend Services beschrieben wurden, wendet sich die Betrachtung dem nächsten wichtigen Baustein zu: dem Frontend, welches die Sicht für den Benutzer bereitstellt.

## 4.7 Das Frontend

Das Frontend des Projekts wurde mithilfe des Frameworks Next.js [32] in TypeScript [6] realisiert. Im Unterabschnitt 4.7.1 wird beschrieben, wie das Frontend initialisiert und anschließend der Next.js [32] Server hochgefahren wird, welches die Basis für die Interaktion mit dem Benutzern darstellt. Danach wird im Unterabschnitt 4.7.2 gezeigt, wie eine Tabelle mit Kryptowährungspreisen statisch generiert wird. Anschließend wird im Unterabschnitt 4.7.3 die Einbindung der Daten über die API beschrieben, damit die Daten dynamisch aufgerufen und dargestellt werden können.

### 4.7.1 Initialisierung und Start des Next.js Servers

Der einfachste Weg, mit Next.js [32] zu beginnen, stellt die Nutzung von `create-next-app` dar. Das ist ein Command Line Interface (CLI)-Tool mit dem ein schneller Start in die Erstellung einer neuen Next.js Anwendung initialisiert wird (siehe Listing 4.23).

Listing 4.23: Erstellung einer Next.js [32] Anwendung

```
1 yarn create next-app web
```

Im nächsten Schritt erfolgt die Integration aller notwendigen externen Bibliotheken. Unter anderem dient “shadcn/ui” [44] als Komponentenbibliothek, während “react-query” [45] und “axios” [46] für die Abwicklung von HTTP-Anfragen zuständig sind. Diese Bibliotheken können mit dem Befehl `yarn add <Library>` heruntergeladen werden. Nach der Installation befinden sich die Pakete im Verzeichnis “no-

de\_modules“ und die Referenzen dazu werden automatisch in die Dateien “package.json“ und “yarn.lock“.

#### 4.7.2 Eine Kryptowährungspreis-Tabelle erstellen

Zur Darstellung der Kryptowährungspreise wird zunächst die Datei @/app/kryptopreise/page.tsx im Root-Verzeichnis des Projekts erstellt. Dies ermöglicht den Zugriff auf die Seite unter der Adresse domain/kryptopreise.

Zu Beginn wird die Direktive “use client“ angegeben, um Client-seitiges Rendering (CSR) zu ermöglichen (vgl. [32]). Dies ist notwendig, da API-Anfragen durchgeführt werden und die Daten erst auf dem Client verarbeitet und gerendert werden sollen. Anschließend werden Komponenten für die Tabelle importiert, die mithilfe der zuvor installierten shadcn/ui-Bibliothek [44] zur Verfügung stehen (siehe Listing 4.24).

Listing 4.24: Basis Einer Tabelle

```

1 "use client";
2 import {
3   Table,
4   TableBody,
5   TableCaption,
6   TableCell,
7   TableHead,
8   TableHeader,
9   TableRow,
10 } from "@/components/ui/table";
11
12 export default function KryptoPreise() {
13   return (
14     <div className="max-w-xs bg-gray-300">
15       <Table className="bg-sky-200">
16         <TableCaption className="bg-blue-500 text-white">
17           Kryptowährungspreise
18         </TableCaption>
19         <TableHeader className="bg-sky-200">
20           <TableRow>
21             <TableHead>Symbol</TableHead>
22             <TableHead>Preis</TableHead>
23           </TableRow>
24         </TableHeader>
25         <TableBody className="bg-sky-400">
26           <TableRow>
27             <TableCell>Bsp</TableCell>
28             <TableCell>2.22</TableCell>
29           </TableRow>
30         </TableBody>
31       </Table>

```

```

32     </div>
33   );
34 }
```

Listing 4.24 zeigt die grundlegende Struktur und das Design der Tabelle, wobei TailwindCSS [47] für das Styling eingesetzt wird. Abbildung 4.3 zeigt das Ergebnis.

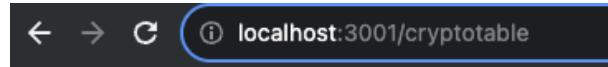


Abbildung 4.3: Basis Tabelle

Hiermit wurde eine erste Tabelle erstellt, wobei die Werte noch statisch sind. Als nächstes wird die Tabelle dynamisch gestaltet, sodass die Daten aus der API abgerufen werden.

### 4.7.3 Einbindung der Daten über die API

Zur Einbindung der Daten aus der API wird axios [46] zur Durchführung von HTTP GET-Anfragen und react-query [45] zur Datenabfrage und -verarbeitung verwendet. Bei Aufruf der Seite wird automatisch eine Anfrage an die API unter <http://127.0.0.1:3003/symbols> gesendet und die erhaltenen Daten in das data-Objekt geladen (siehe Listing 4.25).

Listing 4.25: Einbindung der API

```

1 import { Data } from "@types/symbol";
2 import { useQuery } from "@tanstack/react-query";
3 import axios from "axios";
4
5 export default function KryptoPreise() {
6   const { data } = useQuery({
7     queryKey: ["symbol"],
8     queryFn: async () => {
9       const { data } = await axios.get("http://127.0.0.1:3003/symbols");
10      return data as Data[];
11    },
12  });
}
```

Um zu gewährleisten, dass die Tabelle bereits beim ersten Laden Inhalte zeigt, wird ein Platzhalter-Array verwendet. Dieses füllt die Tabelle zunächst mit Platzhalterdaten, bis die echten Daten aus der API geladen wurden. Innerhalb der Funktion wird anschließend entschieden, ob die echten Daten oder die Platzhalterdaten in der Tabelle dargestellt werden. Mithilfe einer Iteration über das cryptoData-Array werden dann die jeweiligen Werte in die Tabellenzellen eingegeben (siehe Listing 4.26).

Listing 4.26: API-Daten in der Tabelle anzeigen

```

1 const placeholderData = Array(100).fill({
2   pair: "LOADING...",
3   price: 0,
4 });
5
6 export default function KryptoPreise() {
7   ...
8   const cryptoData = data || placeholderData;
9   return (
10     ...
11     <TableBody className="bg-sky-400">
12       {cryptoData.map((d, i) => (
13         <TableRow key={i}>
14           <TableCell>{d.pair}</TableCell>
15           <TableCell>{d.price}</TableCell>
16         </TableRow>
17       )));
18     </TableBody>
19     ...
20   )
21 }
```

Mit diesen Schritten wurde eine dynamische Tabelle erstellt, welche die Kryptowährungsdaten aus einer API dynamisch darstellt, wie Abbildung 4.4 zeigt.

Symbol	Preis
SPELL/EUR	0.000423
AMPEUR	0
FTM/USD	0.2261449999999998

Abbildung 4.4: Final Tabelle

Da nun das Frontend und das Backend implementiert wurde, muss jetzt nur noch die Infrastruktur bereitgestellt werden.

## 4.8 Bereitstellung der Infrastruktur

In diesem Abschnitt geht es um die Implementierung und Bereitstellung (Deployment) der gesamten Infrastruktur auf einen Server. Dabei wird jeder einzelne Service in einer eigenen Docker-Instanz [35] ausgeführt. Die Kommunikation zwischen den Containern erfolgt dann über ein Docker-Netzwerk [35]. Für die Bereitstellung der Infrastruktur wird Docker-Compose [35] verwendet.

Für den Stream Service wurde ein Container erstellt, der auf der Python [27] Laufzeitumgebung basiert. Hier wird das Projekt in den Container kopiert und die Abhängigkeiten mit pip install installiert. Nach der Installation kann der Container gestartet werden und die Kryptowährungsbörse als Argument übergeben werden (siehe Listing 4.27).

Listing 4.27: Dockerfile /citedockerdocs Stream Service

```

1 FROM python:3.11
2
3 # Set work directory
4 WORKDIR /app
5 RUN mkdir /app/logs
6
7 # Copy project
8 COPY . .
9
10 # Install dependencies
11 RUN pip install --upgrade pip
12 RUN pip3 install -r requirements.txt
13
14 # Command to run on container start
15 ENTRYPOINT [ "python", "-m", "streamservice.main" ]
16 CMD [ "bitstamp" ]
```

Der Storage-Service, Websocket Refresher Service nutzen ein mehrstufiges Build-Verfahren, dargestellt in Listing 4.28. Dies ermöglicht die Erstellung von schlanken Containern. Anfänglich wird ein Golang-Container [5] verwendet, um die Services zu kompilieren. Anschließend werden diese kompilierten Services in einen leichtgewichtigen Alpine-Container [35] verlagert und dort ausgeführt. Der Vorteil dieses Ansatzes ist nicht nur eine reduzierte Bildgröße und effizientere Ressourcennutzung, sondern auch eine verbesserte Sicherheit durch die Verwendung minimaler Images wie Alpine [35].

Listing 4.28: Dockerfile Golang Service

```

1 FROM golang:1.20 as builder
2
3 WORKDIR /app
4
5 # Copy the Go Mod and Sum files and download the dependencies separately
6 # This step will be cached if we won't change mod/sum files
7 COPY ../go.mod .
8 COPY ../go.sum .
9 RUN go mod download
10
11 COPY . .
12
13 # Install dependencies and build the applications
14 RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o /out/apiservice ./cmd/
   api/main.go
15 RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o /out/
   datarefresherservice ./cmd/datarefresherservice/main.go
16 RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o /out/storageservice ./cmd/
   storageservice/main.go
17 RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o /out/websocketservice
   ./cmd/websocket/main.go
18
19 # Run with Alpine image
20 FROM alpine:latest as api-runner
21 COPY --from=builder /out/apiservice /app/
22 COPY --from=builder /app/config.yaml /
23 COPY --from=builder /app/queries/ /queries/
24 CMD ["/app/apiservice"]
25
26 FROM alpine:latest as refresher-runner
27 COPY --from=builder /out/datarefresherservice /app/
28 COPY --from=builder /app/config.yaml /
29 COPY --from=builder /app/queries/ /queries/
30 CMD ["/app/datarefresherservice"]
31
32 FROM alpine:latest as storage-runner
33 COPY --from=builder /out/storageservice /app/
34 COPY --from=builder /app/config.yaml /
35 CMD ["/app/storageservice"]
36
37 FROM alpine:latest as websocket-runner
38 COPY --from=builder /out/websocketservice /app/
39 COPY --from=builder /app/config.yaml /
40 CMD ["/app/websocketservice"]

```

Der Web Service benötigt einen Node-Container [32], da node als Laufzeitumgebung für nextjs [32] erforderlich ist. Der Web Service wird in diesen Container importiert

und anschließend mit den Befehlen `yarn build` und `yarn start` ausgeführt, wie in Listing 4.29 dargestellt. `yarn build` kompiliert und optimiert den Code für die Produktion, während `yarn start` die Anwendung startet [32].

Listing 4.29: Dockerfile vom Web Service

```

1 FROM node:alpine
2
3 WORKDIR /app
4 COPY package.json yarn.lock /app/
5 ENV YARN_CACHE_FOLDER=/dev/shm/yarn_cache
6 RUN yarn install
7 COPY . /app
8 RUN yarn build
9 CMD [ "yarn", "start" ]

```

Zum Abschluss des Setups wird die Orchestrierung aller Container durch eine `docker-compose`-Datei ermöglicht. Es ist anzumerken, dass spezifische Dienste wie Redis [40], InfluxDB [39] und RabbitMQ [41] bereits in Abschnitt 4.1 erläutert wurden. Die hier dargestellte Konfiguration in Listing 4.30 bezieht sich auf die Einrichtung von Stream-Services, die exemplarisch für die Implementierung der anderen Services in der Anwendung stehen. Diese anderen Services folgen einer ähnlichen Struktur und Konfiguration. Außerdem wird für das Weiterleiten der Client-Anfragen noch ein Webserver, in diesem Fall Nginx [48], verwendet.

Listing 4.30: Docker Compose Infrastruktur

```

1 version: "3.8"
2
3 services:
4   stream-app-kraken:
5     build: ./services/
6     container_name: stream_kraken
7     networks:
8       - cryptoTracker
9     restart: always
10    environment: *common_environment
11    depends_on:
12      - rabbitmq
13    command: kraken
14
15  nginx:
16    image: nginx:latest
17    container_name: nginx
18    restart: always
19    ports:
20      - "80:80"
21    volumes:

```

```

22     - ./nginx/:/etc/nginx/conf.d/
23 depends_on:
24     - api-service
25     - web-service
26     - websocket-service
27 networks:
28     - cryptoTracker
29
30 networks:
31     cryptoTracker:
32         driver: bridge

```

Listing 4.31 zeigt die Nginx Konfiguration welche im Listing 4.30 als volume übergeben wurde. Diese Konfiguration hört erstmal nur auf Port 80. Die Anfragen von Clients werden entsprechend ihrer Pfade entweder an das Frontend, die API oder den Websocket weitergeleitet. Außerdem ist es so konfiguriert, dass es Websocket Verbindungen upgraden kann. Es gilt aber zu beachten, dass Cloudflare [49] als TLS-Provider eingesetzt wird, dass ermöglicht es, dass die Kommunikation zwischen Client und Cloudflare [49] verschlüsselt wird. Jedoch bleibt die Kommunikation zwischen Cloudflare und dem Server unverschlüsselt. Dies ist für einen produkten Einsatz nicht zu empfehlen, soll aber hier beim Prototypen nicht weiter beachtet werden.

Listing 4.31: Nginx Konfiguration

```

1 server {
2     listen 80;
3     # Frontend
4     location / {
5         proxy_pass http://web-service:3000;
6         proxy_set_header Host $host;
7         proxy_set_header X-Real-IP $remote_addr;
8         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
9     }
10    # Backend API
11    location /api {
12        proxy_pass http://api-service:3003;
13        proxy_set_header Host $host;
14        proxy_set_header X-Real-IP $remote_addr;
15        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
16    }
17    # Backend Notification Endpoint
18    location /notify {
19        proxy_pass http://websocket-service:3008;
20        proxy_set_header Host $host;
21        proxy_set_header X-Real-IP $remote_addr;
22        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
23    }
24    # Backend Websocket

```

```

25   location /ws {
26     proxy_pass http://websocket-service:3008;
27     proxy_http_version 1.1;
28     proxy_set_header Upgrade $http_upgrade;
29     proxy_set_header Connection "upgrade";
30     proxy_set_header Host $host;
31     proxy_set_header X-Real-IP $remote_addr;
32     proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
33   }
34 }
```

Das beschriebene Deployment sorgt für eine schnelle und effiziente Bereitstellung der Anwendung und ihrer Dienste. Es zeigt auch die Flexibilität und Effizienz von Containern und wie verschiedene Technologien zusammenarbeiten können.

## Zusammenfassung

In Kapitel 4 wurde die prototypische Implementierung des Projekts vorgestellt, wodurch das Forschungsziel FZ3S erfüllt wurde. Dies umfasst den gesamten Prozess von der Einrichtung einer robusten Entwicklungsinfrastruktur bis hin zur abschließenden Bereitstellung der Infrastruktur. Die Kernkomponenten des Systems wurden detailliert und unter Einbeziehung relevanter Technologien und Frameworks wie Python [27], Golang [5], RabbitMQ [41], InfluxDB [39] und Docker [35] beschrieben. Die Implementierung jedes Dienstes wurde detailliert erörtert, wobei der Fokus auf der Funktionsweise und der Integration der einzelnen Komponenten lag.

Die erfolgreiche Integration aller Komponenten in einer Docker basierten Infrastruktur demonstriert die Leistungsfähigkeit und Flexibilität des entwickelten Systems. Die Anwendung zeigt sich damit als robustes und effizientes Tool, das die verschiedenen Anforderungen der Kryptowährungsanalyse und -darstellung erfüllt.

Nach der detaillierten Betrachtung der Implementierung des Prototyps, richtet sich der Fokus in Kapitel 5 auf die Evaluation des Systems. Hierbei wird untersucht, wie das entwickelte System in der Praxis funktioniert, wie es auf unterschiedliche Anwendungsszenarien reagiert und inwiefern es die gestellten Anforderungen erfüllt. Diese Evaluation ist entscheidend, um die Effektivität und Zuverlässigkeit des Systems zu validieren und mögliche Verbesserungsbereiche zu identifizieren.

# Kapitel 5

## Evaluation des Prototypen

In Kapitel 5 steht die Evaluation des entwickelten Prototypen im Vordergrund, was der Bearbeitung des FZ4E entspricht. Das vierte und letzte FZ zielt darauf ab, das System auf seine Effektivität hin zu überprüfen, indem der Prototyp evaluiert wird.

Der Prozess beginnt mit einem Cognitive Walkthrough, bei dem die Anwendung Schritt für Schritt dargestellt sowie die implementierten Funktionalitäten im Detail vorgestellt werden. Dieser Schritt beinhaltet eine präzise Beschreibung des User Interfaces und der Navigation innerhalb der Anwendung, begleitet von Screenshots, um die Benutzerführung und -interaktionen zu veranschaulichen.

Anschließend an den Cognitive Walkthrough erfolgt die Bewertung der Funktionalitäten. Die Evaluation wird von dem Experten Herr Lothary, M. Sc. durchgeführt. Er wird die Effektivität, Effizienz und Benutzerfreundlichkeit des Prototypen kritisch bewerten, um festzustellen, inwieweit der Prototyp den gestellten Anforderungen und den Erwartungen der Benutzer entspricht. Durch diese systematische Beurteilung sollen Stärken und Schwächen des Systems identifiziert und Ansatzpunkte für zukünftige Verbesserungen aufgezeigt werden.

### 5.1 Cognitive Walkthrough

Der Cognitive Walkthrough führt durch den gesamten Prototypen. Der Aufruf der Website führt zur Startseite der Anwendung, welche im Unterabschnitt 5.1.1 gezeigt wird. Anschließend wird im Unterabschnitt 5.1.2 die Detailseite der Anwendung beleuchtet, die historische Preisdaten und detaillierte Analysen der Kryptowährungen präsentiert. Zuletzt wird im Unterabschnitt 5.1.3 die Seite der potentiellen Handelsmöglichkeiten (Potentialen Seite) vorgestellt, die Nutzern eine Übersicht über die aktuellen Handelschancen bietet.

### 5.1.1 Die Startseite

Im oberen Bereich jeder Seite findet sich die Navigationsleiste (rote Umrandung in Abbildung 5.1). Die Navigationsleiste bietet folgende Interaktionsmöglichkeiten:

- Durch anklicken des Logos oder des „Coins“-Buttons gelangen Nutzer zurück auf die Startseite (1 und 2 in Abbildung 5.1).
- Ein klick auf „Potentialen“ führt zu der Seite mit den potentiellen Handlungsempfehlungen (3 in Abbildung Abbildung 5.1).
- Über das Github-Icon können Nutzer direkt zum Github Repository<sup>13</sup> des Projekts navigieren (4 in Abbildung Abbildung 5.1).
- Der Hell- und Dunkelmodus der Anwendung lässt sich durch klicken auf die Sonnen-Icon umstellen (5 in Abbildung Abbildung 5.1).

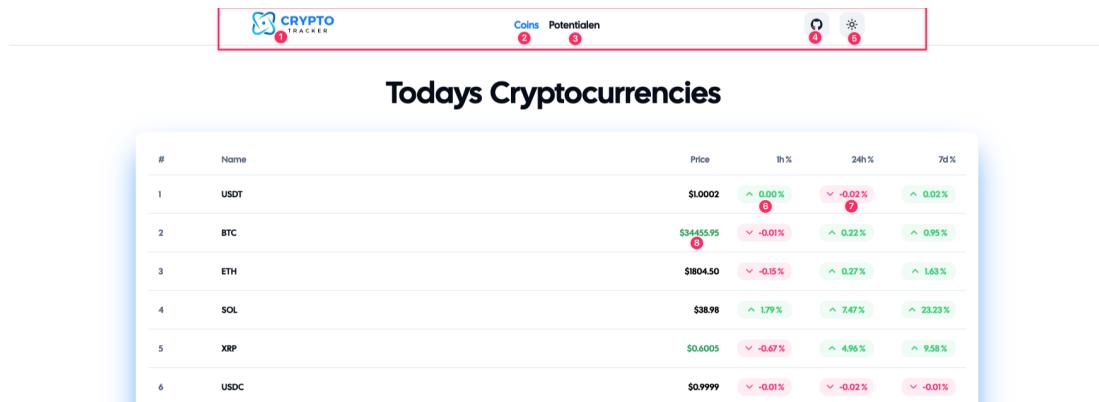


Abbildung 5.1: Darstellung der Startseite

Unterhalb der Navigationsleiste präsentiert die Startseite eine Übersicht der Kryptowährungspreise in Tabellenform. Angezeigt werden der Kürzel der Kryptowährung, der aktuelle Preis in Echtzeit sowie der Preisverlauf der letzten Stunde, der letzten 24 Stunden und der letzten 7 Tage. Positive Preisentwicklungen werden in Grün und negative in Rot dargestellt (6 und 7 in Abbildung 5.1).

Die Preise der Kryptowährungen aktualisieren sich alle 5 Sekunden automatisch. Bei einer Preissteigerung leuchtet der Preis kurz grün auf, während ein Preisrückgang in rot dargestellt wird (8 in Abbildung 5.1). Sortiert werden die Kryptowährungspreise nach dem höchsten Handelsvolumen der letzten 24 Stunden..

Durch einen klick auf das Sonnen-Icon (1 in Abbildung 5.2) wird ein Dropdown-Menu angezeigt. Dort kann der Hell- oder Dunkelmodus ausgewählt werden (2 und 3 in

<sup>13</sup> Erreichbar unter <https://github.com/Marv963/CryptoTracker>

Abbildung 5.1).

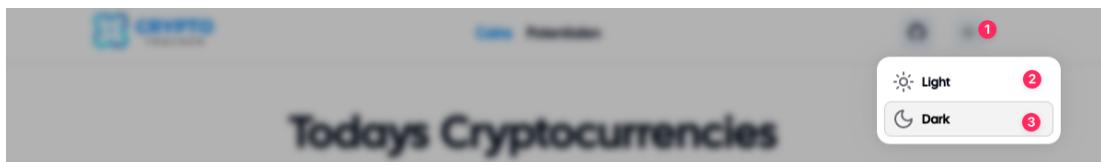


Abbildung 5.2: Hell-Dunkel Modus auswählen

Klickt man nun auf den Button „Dark“ (3 in Abbildung 5.2) wird der Dunkelmodus aktiviert (siehe Abbildung 5.3). Dieser Modus bleibt solange aktiv, bis der Hell-Modus wieder aktiviert wurde oder der Cookie gelöscht wird.

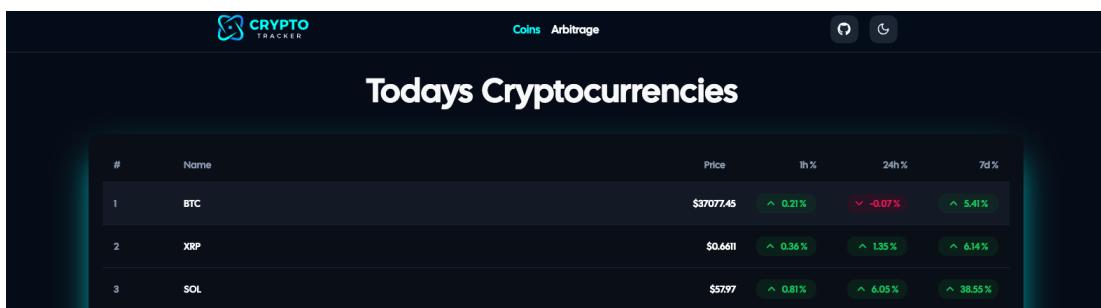


Abbildung 5.3: Dunkel-Modus Darstellung

Wenn man nun mit der Maus über einen Tabelleneintrag geht, wird ein Hover-Effekt sichtbar (1 in Abbildung 5.3). Klickt man nun auf diesen Eintrag wird man zur Detailseite der Kryptowährung weitergeleitet. Alle weiteren Seiten werden zur besseren Darstellung im Hellen Modus gezeigt.

### 5.1.2 Die Detailseite

Die Detailseite präsentiert die historischen Preisdaten einer Kryptowährung in einem interaktiven Chart, wie in Abbildung 5.4 dargestellt. Diese Funktion ermöglicht es Nutzern, die Preisentwicklung einer Kryptowährung über verschiedene Zeiträume und Exchanges hinweg zu verfolgen und zu analysieren. Standardmäßig wird im Chart der durchschnittswert der Exchanges verwendet, welche in der oberen Linken Ecke unter „Exchanges=average“ zu sehen ist (siehe 1 in Abbildung 5.4)

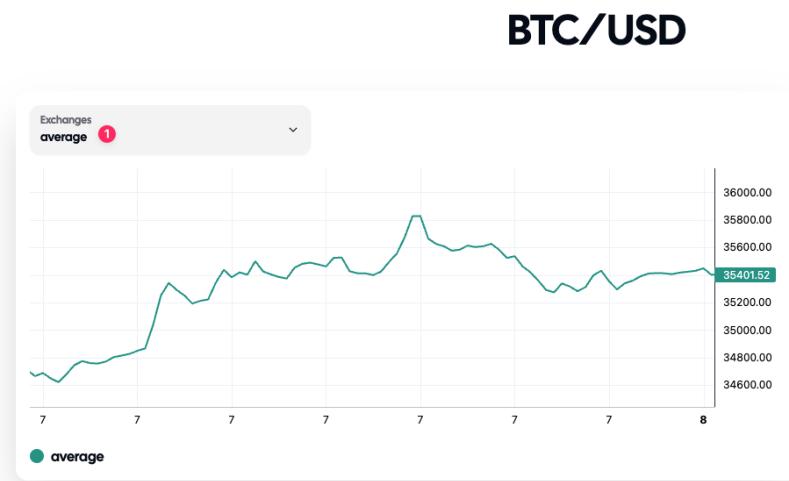


Abbildung 5.4: Die Detailseite von BTC/USD

Durch anklicken des „Exchanges“ Feld öffnet sich ein Dropdown-Menü, in dem die Nutzer zwischen verschiedenen Exchanges wählen können, wie in Abbildung 5.5 illustriert.



Abbildung 5.5: Kryptobörsen auf der Detailseite auswählen

Bei Auswahl einer bestimmten Kryptobörse aus dem Dropdown-Menü (siehe 1 und 2 Abbildung 5.5) wird diese in den Chart integriert und die entsprechenden Preisinformationen werden angezeigt, wie in Abbildung 5.6 zu sehen.

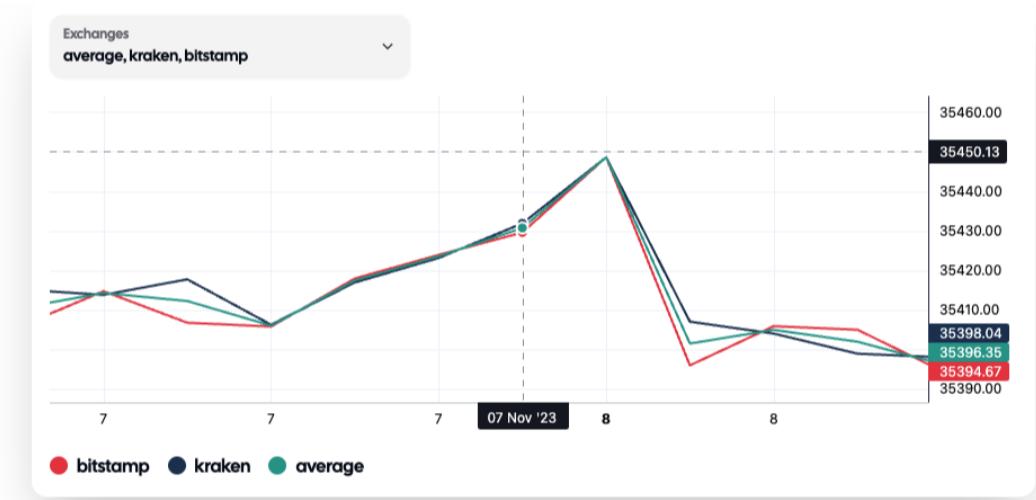


Abbildung 5.6: Die Chart analyse

Wenn Nutzer den Mauszeiger über den Chart bewegen, erscheint ein Koordinatenkreuz (siehe Abbildung 5.6), das präzisere Analysen ermöglicht. Durch das Drücken der linken Maustaste oder durch `Shift`+`Scrollen` können Nutzer in der Zeitachse navigieren, um spezifische Zeitpunkte und deren zugehörige Daten genauer zu betrachten. Das Zoomen in den Chart hinein oder heraus erfolgt durch das Scrollen mit der Maus. Als nächstes wird die Potentialen Seite vorgestellt, welcher über die Navigationsleiste erreicht werden kann (siehe 3 in Abbildung 5.1).

### 5.1.3 Die Potentialen Seite

Die Potentialen Seite gibt den Nutzern eine Übersicht über die Kryptowährungspaare, die aktuell die größten Handelschancen bieten. Diese Kryptowährungspaare werden in einer Tabelle präsentiert, sortiert nach dem höchsten potentiellen Gewinn in absteigender Reihenfolge, wie in Abbildung 5.7 dargestellt. In der Tabelle werden der Kürzel der Kryptowährung, der Durchschnittspreis über alle Exchanges hinweg, der niedrigste Preis an einem spezifischen Exchange, der höchste Preis an einem anderen Exchange, die prozentuale Differenz zwischen den Preisen sowie ein „Calculate“-Button für weitere Berechnungen angezeigt.

#	Name	Average Price	Lowest Price	Highest Price	Differenz in %	Calculated Arbitrage
1	RAD/EUR	\$1.4125	1.3250	1.5000	13.21	<button>Calculate 1</button>
2	CVX/EUR	\$2.8000	2.6300	2.9700	12.93	<button>Calculate</button>
3	ENJ/EUR	\$0.2505	0.2390	0.2621	9.6527	<button>Calculate</button>
4	ALPHA/EUR	\$0.0753	0.0719	0.0787	9.4258	<button>Calculate</button>
5	IMX/EUR	\$0.6330	0.6060	0.6600	8.9109	<button>Calculate</button>

Abbildung 5.7: Die Potentialen Seite

Durch anklicken des „Calculate“-Button (siehe 1 in Abbildung 5.7), öffnet sich ein Formular im Vordergrund, wie in Abbildung 5.8 am Beispiel des Paares RAD/EUR illustriert. Dieses Formular ermöglicht es Nutzern, den geschätzten Profit für einen potentiellen Trade zu berechnen.

Berechne Potentiellen Trade

**RAD/EUR**

**Lowest Price**  
**1.3250**  
at kraken ①

+13%

**Highest Price**  
**1.5000**  
at bitstamp ②

---

**Amount in EUR**  
 Amount of EUR to Trade ③

**Trade Volume on kraken last 30 Days**  
Trade Volume on kraken last 30 Days ④

---

Trade Volume on bitstamp last 30 Days  
Trade Volume on bitstamp last 30 Days ⑤

**Estimated Profit: EUR -4.65**

Step 1: Buy RAD/EUR at kraken for 1.325.  
 Step 2: Sell at bitstamp for 1.5.

[Close](#)

Abbildung 5.8: Potentiellen Handel berechnen

In diesem Beispiel liegt der niedrigste Preis für RAD/EUR bei Kraken [16] mit 1,325 € (1 in Abbildung 5.8), während der höchste Preis bei Bitstamp [17] mit 1,5 € festgestellt wurde (2 in Abbildung 5.8). Im Formular darunter können Nutzer den Betrag in Euro eingeben, den sie investieren möchten (3 in Abbildung 5.8). Zudem

müssen die Handelsvolumina der letzten 30 Tage auf Kraken [16] und Bitstamp [17] angegeben werden, da diese für die Berechnung der Handelsgebühren relevant sind (4 und 5 in Abbildung 5.8).

Nach der Eingabe dieser Daten, wie in Abbildung 5.9 dargestellt, wird der geschätzte Profit berechnet und angezeigt (siehe 1 in Abbildung 5.9). In diesem Szenario würde der Nutzer auf Kraken [16] 10.000 € in RAD tauschen, diese dann zu Bitstamp [17] transferieren und anschließend dort verkaufen. Der berechnete Profit unter Berücksichtigung der Ein- und Auszahlungskosten auf beiden Exchanges beträgt in diesem Beispiel 1.241,52 €. Es ist jedoch zu beachten, dass Transaktionskosten in dieser Berechnung noch nicht berücksichtigt sind und gegebenenfalls vom Profit abgezogen werden müssen.

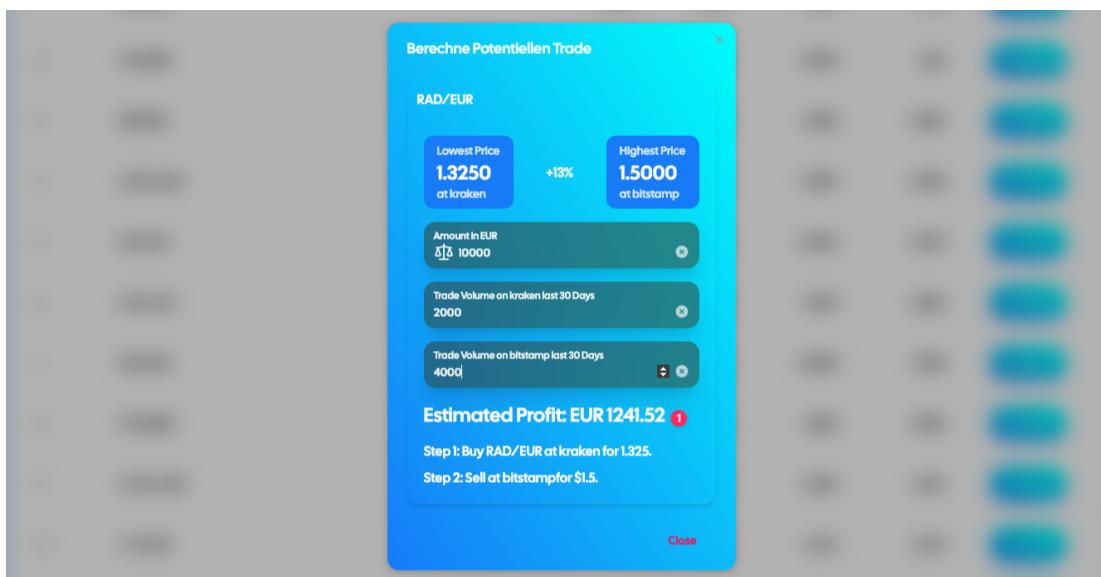


Abbildung 5.9: Gewinn des potentiellen Handel berechnen

Mit der Darstellung der Potentialen Seite ist der Cognitive Walkthrough des Prototypen abgeschlossen und es folgt das Evaluationsergebnis.

## 5.2 Das Evaluationsergebnis

Die Evaluation des Prototypen wurde vom Experten Herrn Sebastian Lothary, M. Sc., durchgeführt. In dieser Phase wurden die Funktionalitäten und die Benutzererfahrung des Systems einer kritischen Prüfung unterzogen. Im Unterabschnitt 5.2.1 werden die Evaluationsergebnisse der Startseite des Prototypen dargestellt, die sowohl Stärken als auch Verbesserungsmöglichkeiten des ersten Kontaktpunkts mit der Anwendung aufzeigen. Anschließend wird im Unterabschnitt 5.2.2 das Ergebnis der Evaluation

von der Detailseite und im Unterabschnitt 5.2.3 die Ergebnisse der Evaluation von der Potentialen Seite vorgestellt.

### 5.2.1 Das Evaluationsergebnis von der Startseite

Die Evaluationsergebnisse der Startseite des Prototypen haben sowohl Stärken als auch Verbesserungsmöglichkeiten aufgezeigt:

1. **Anzeige der Kryptowährungen:** Derzeit werden nur die ersten 100 Kryptowährungen auf der Startseite angezeigt. Dies führt dazu, dass einige Kryptowährungen fehlen.
2. **Sortierung der Daten:** Die aktuelle Sortierung der Kryptowährungen ist für den Benutzer nicht unmittelbar ersichtlich. Es wäre hilfreich, die Sortierkriterien deutlicher zu kennzeichnen, um die Transparenz und Nachvollziehbarkeit für die Nutzer zu verbessern.
3. **Ästhetik und Performance:** Die Startseite wirkt optisch ansprechend und lädt sehr schnell.
4. **Responsives Design:** Die Webseite ist responsiv und passt sich gut an verschiedene Bildschirmgrößen an, was die Benutzerfreundlichkeit auf mobilen Geräten und Tablets erhöht.
5. **Browser-Kompatibilität:** Die Seite zeigt in den drei gängigen Webbrowsern (Safari, Firefox, Chrome) eine konsistente Erscheinungsbild.
6. **Informationsanzeige bei Hovern:** Wenn man mit der Maus über ein Kryptowährungskürzel „hovert“ wird der vollständige Name der Kryptowährung nicht angezeigt.
7. **Realtime-Daten:** Es ist positiv hervorzuheben, dass die Preise der Kryptowährungen in Echtzeit aktualisiert werden.

Diese Ergebnisse zeigen, dass der Prototyp in vielen Aspekten wie Design, Performance und Benutzerfreundlichkeit gut funktioniert, während gleichzeitig Raum für Verbesserungen in der Informationsdarstellung und Marktdeckung besteht. Im nächsten Abschnitt werden die Ergebnisse der Evaluation der Detailseite vorgestellt.

### 5.2.2 Das Evaluationsergebnis von der Detailseite

Die Detail Seite des Prototypen zeichnet sich durch eine Reihe von positiven Aspekten aus, die während der Evaluation hervorgehoben wurden:

1. **Visuelle Anziehungskraft und Schnelligkeit:** Die Seite ist optisch ansprechend gestaltet und überzeugt durch eine schnelle Ladezeit. Diese Eigenschaften tragen zu einer positiven Benutzererfahrung bei.
2. **Responsives Design:** Die Webseite passt sich effektiv an verschiedene Bildschirmgrößen und Geräte an, wodurch ihre Zugänglichkeit und Benutzerfreundlichkeit auf mobilen Geräten und Tablets erhöht.
3. **Browser-Kompatibilität:** Die gängigen Webbrowser (Safari, Firefox, Chrome) zeigen ein konsistentes Erscheinungsbild.
4. **Attraktive Chart-Darstellung:** Der auf der Seite integrierte Chart ist nicht nur funktional, sondern auch visuell ansprechend.
5. **Verbesserungspotential bei der Touchpad-Bedienung:** Die Benutzerfreundlichkeit des Charts mit einem Touchpad müsste optimiert werden, um die Handhabung zu erleichtern und die Zugänglichkeit zu verbessern.
6. **Fehlende Erläuterungen:** Eine Erklärung oder Anleitung auf der Seite könnte den Nutzern helfen, die Funktionsweise und den Nutzen der Detail Seite besser zu verstehen und effizienter zu nutzen.
7. **Automatischer Reset im Chart:** Es wurde festgestellt, dass der Chart nach längerem Verbleiben in der Zoomansicht automatisch zurücksetzt. Das erschwert die Analyse und sollte behoben werden.
8. **Handelsvolumen:** In dem Chart fehlt die Angabe des Handelsvolumen für die jeweilige Zeit. Diese wird benötigt um die Nachfrage einer Kryptowährung auf einer Kryptobörse nachzuvollziehen.

Diese Punkte verdeutlichen, dass die Detail Seite ansprechend und funktional umgesetzt wurde, während gleichzeitig einige Aspekte vorhanden sind, die für eine optimierte Benutzererfahrung angepasst werden könnten. Im nächsten Abschnitt wird das Evaluationsergebnis der Potentialen Seite vorgestellt.

### 5.2.3 Das Evaluationsergebnis von der Potentialen Seite

Die Potentialen Seite des Prototypen hat in der Evaluation diverse Stärken und einige Verbesserungsmöglichkeiten aufgezeigt:

1. **Visuelle Anziehungskraft und Geschwindigkeit:** Die Seite ist ansprechend designet und überzeugt mit einer hohen Ladegeschwindigkeit.
2. **Responsives Design:** Die Seite passt sich an verschiedene Bildschirmgrößen und Endgeräte an.
3. **Browser-Kompatibilität:** Die Seite zeigt in gängigen Webbrowsern (Safari, Firefox, Chrome) ein konsistentes Erscheinungsbild.

4. **Format der Daten:** Das Format der Daten ist nicht unmittelbar klar ist, insbesondere bei der Dezimaltrennung (Punkt statt Komma). Eine Anpassung an das lokal übliche Format könnte die Verständlichkeit für Nutzer verbessern.
5. **Präzision der Preisangaben:** Bei Werten unter 10 kann die Anzeige von vier Nachkommastellen verwirrend wirken. Eine Reduzierung der Dezimalstellen könnte hier für mehr Klarheit sorgen.
6. **Profitkalkulation:** Die Seite ermöglicht es, nachzuvollziehen, wie viel Gewinn bei einer bestimmten Investitionssumme erzielt werden kann, welches eine wertvolle Funktion für Nutzer darstellt.
7. **Fehlende Gebührenstruktur:** In der aktuellen Profitberechnung fehlt die Anzeige der Gebührenstruktur.
8. **Text unter der Profitberechnung:** Der Text unter der Profitberechnung kann auf den ersten Blick irreführend sein und zur Annahme führen, es handele sich um Gebühren. Eine klarere Kennzeichnung könnte Missverständnisse vermeiden.
9. **Tradevolumen in Berechnungen:** Das Tradevolumen in den Berechnungen könnte vernachlässigt werden und stattdessen könnte man von den höchsten Handelsgebühren ausgehen, da das Tradevolumen keinen signifikanten Einfluss auf den Profit hat.

Diese Ergebnisse deuten darauf hin, dass die Potentialen Seite insgesamt gut funktioniert und in vielen Bereichen eine hohe Benutzerfreundlichkeit und Effektivität bietet. Gleichzeitig gibt es klare Möglichkeiten für Verbesserungen, insbesondere in Bezug auf die Präzision und Verständlichkeit der dargestellten Informationen. Im nächsten Schritt wird eine zusammenfassende Bewertung des gesamten Prototypen vorgenommen.

## Zusammenfassung

Die Evaluation des Prototypen hat gezeigt, dass der Prototyp in mehreren Aspekten gut funktioniert und somit FZ4E erfüllt. Besonders sind dabei die schnellen Ladezeiten und das ansprechende Design hervorzuheben. Durch die konsistente Darstellung in verschiedenen Webbrowsern sowie das responsive Design wird eine breite Zugänglichkeit und Benutzerfreundlichkeit über verschiedene Geräte hinweg gewährleistet. Es wurde jedoch auch deutlich, dass der Prototyp in seiner aktuellen Form sehr erklärbungsbedürftig ist. Insbesondere bei komplexeren Funktionen und Darstellungen, wie beispielsweise der Profitkalkulation, fehlen oft klare Erklärungen, die den Nutzern helfen, die Funktionsweise der Seite besser zu verstehen und effizient zu nutzen. Zudem

besteht Bedarf an weiteren Informationen und Verbesserungen in der Datenpräsentation.

Abschließend lässt sich feststellen, dass der Prototyp eine solide Grundlage für weitere Entwicklungen bietet. Im nächsten und letzten Abschnitt folgt das Fazit.

# Kapitel 6

## Fazit

Im Abschluss dieser Arbeit wird ein Fazit über die erzielten Ergebnisse gezogen und ein Ausblick auf mögliche Verbesserungen und Weiterentwicklungen des Prototypen gegeben.

Diese Arbeit hat das Ziel verfolgt, einen Prototypen zu entwickeln, der die unterschiedlichen Kostenstrukturen auf den verschiedenen Exchanges berücksichtigt und profitable Handelsmöglichkeiten aufzeigt. Der entwickelte Prototyp hat in der Evaluation gezeigt, wie ein automatisiertes System die Preise verschiedener Kryptowährungen effizient erfassen und vergleichen kann. Trotz einiger Herausforderungen, insbesondere im Hinblick auf die Transparenz der Kostenstrukturen und die Dynamik des Kryptomarktes, liefert der Prototyp Einblicke und Handlungsempfehlungen für Anleger. Gleichzeitig besteht aber noch der Bedarf, den Prototypen mit mehr Informationen zu füttern.

### 6.1 Ergebnis

Zunächst werden die technische Umsetzung und die Infrastruktur des entwickelten Prototyps beleuchtet, gefolgt von einer Diskussion über die während der Entwicklung aufgetretenen Herausforderungen und Verbesserungspotenziale. Abschließend wird der Bezug zur Forschungsfrage geklärt, um zu bewerten, inwieweit diese durch den Prototypen beantwortet werden konnte.

#### Technische Umsetzung und Infrastruktur

Die Infrastruktur des Systems hat sich als leistungsfähig erwiesen, welches sich in schnellen Ladezeiten widerspiegelte. Insbesondere die Verwendung von Docker [35] spielte eine zentrale Rolle, da es das einfache Hochfahren und Verwalten der be-

nötigten Services wie RabbitMQ [41], Redis [40], InfluxDB [39] und Nginx [48] ermöglichte. Dadurch konnte die Entwicklung des Prototypen sofort starten, ohne die Services manuell installieren zu müssen. Außerdem ergab sich daraus ein großer Vorteil beim späteren Bereitstellen in der Produktionsumgebung, da die Services bereits lokal konfiguriert waren und somit die gleiche Infrastruktur auf der Produktionsumgebung genutzt werden konnte. Diese Vorbereitung und Gestaltung der Entwicklungs- und Produktionsumgebung trug wesentlich zur Beschleunigung des gesamten Entwicklungsprozesses bei und stellte eine reibungslose Übertragung des Systems in den Live-Betrieb sicher.

Mit Python [27] und der CCXT [26] konnte erfolgreich ein System entwickelt werden, um die Kryptowährungsinformationen einheitlich abzurufen. Die standardisierte Ausgabe durch die CCXT-Bibliothek [26] hat es möglich gemacht, einfach auf die APIs der Exchanges zuzugreifen und die Daten zu verarbeiten. Die CCXT-Bibliothek [26] und Python [27] haben sich für diese Aufgabe gut geeignet.

Mit der RabbitMQ [41] konnten die Kryptowährungsdaten effizient zwischen den Services ausgetauscht werden. Dadurch konnten sich jeder Service auf seine Aufgabe konzentrieren und musste nicht auf die anderen Services warten.

Die Programmiersprache Go [5] zeichnete sich durch die schnelle Geschwindigkeit für die Datenverarbeitung und -bereitstellung aus. Trotz des Vorteils der Effizienz sollte jedoch beachtet werden, dass die Entwicklung in Go [5] tendenziell mehr Zeilen Code erfordert als in Python [27]. Da die Services des Systems aber voneinander entkoppelt waren und unabhängig funktionierten, hätte theoretisch auch eine einfachere Sprache wie Python [27] für die Backend-Entwicklung verwendet werden können. Dennoch hat die Wahl von Go [5] aufgrund der höheren Verarbeitungsgeschwindigkeit und der Ressourcenschonung zu einer effizienteren Backend-Implementierung beigetragen. Go [5] hat sich in diesem Kontext als eine leistungsstarke und effiziente Programmiersprache für Backend-Entwicklungen bewährt.

InfluxDB [39] als TSDB hat sich für die Speicherung zeitbasierter Daten als gut erwiesen. Die Daten konnten einfach in der Datenbank gespeichert, analysiert und abgerufen werden. Allerdings ergab sich aufgrund der Strategie, Daten alle 5 Sekunden zu speichern, ein beträchtlicher Speicherverbrauch. Nach einer Woche Dauerbetrieb wurden bereits 10 GB an Daten gespeichert. Für eine langfristige Datenspeicherung und -verwaltung wäre es daher sinnvoll, die Datenspeicherungsstrategie zu überden-

ken. Eine mögliche Lösung könnte sein, die Daten nach einem Tag in Minutenwerte zu aggregieren, nach einer Woche in Stundenwerte und so weiter. Diese Methodik würde es ermöglichen, den Speicherverbrauch zu reduzieren und dennoch relevante historische Daten für längere Zeiträume effizient zu erfassen und zu nutzen.

Das Frontend mit Next.js [32] mit den verschiedenen Rendering-Methoden hat sich als schnell und effizient für die Erstellung einer reaktionsfähigen und optisch ansprechenden Benutzeroberfläche erwiesen. Die Nutzung durch Redis [40] als IMDB zur Speicherung der aktuellen Kryptowährungspreise trug wesentlich zur schnellen Datenbereitstellung für die Start- und Potentialen-Seite bei. Durch die Implementierung einer WebSocket-Verbindung konnten zudem die Kryptowährungspreise regelmäßig aktualisiert und den Nutzern ein dynamisches Echtzeiterlebnis geboten werden. Diese Kombination aus schnellen Frontend-Technologien und effektiver Datenverarbeitung und -bereitstellung ermöglichte es, eine leistungsstarke und benutzerfreundliche Plattform zu schaffen, die die Anforderungen der Nutzer in Bezug auf Schnelligkeit und Aktualität erfüllte.

### **Herausforderungen und Verbesserungspotenziale**

Obwohl der Prototyp technisch robust und leistungsfähig ist, zeigte die Evaluation auch Bereiche, in denen Verbesserungen erforderlich sind. Insbesondere die Benutzeroberfläche erwies sich als erklärbungsbedürftig, was die Zugänglichkeit für weniger technisch versierte Nutzer einschränkte. Eine Vereinfachung und Integration von Hilfetexten könnten hier Abhilfe schaffen.

Um eine fundierte Handelsentscheidungen zu treffen, sind weitere umfassendere Informationen erforderlich. Derzeit basieren die Handlungsempfehlungen hauptsächlich auf Preisunterschiede. Für eine Entscheidungsgrundlage sollten zusätzliche Faktoren wie Angebots- und Nachfragepreise sowie Handelsvolumen integriert werden.

Die Nichtberücksichtigung der On-Chain Transaktionskosten kann zu falschen Schlussfolgerungen über die Rentabilität von Trades führen. Eine Integration dieser Kosten in die Berechnungen würde zu realistischeren und praktikableren Handelsempfehlungen führen. Beispielsweise liegen die On-Chain Transaktionskosten bei Bitcoin aktuell bei knapp 14€<sup>14</sup>. Das macht Bitcoin als Währungspaar für geringere Handlungssummen direkt unrentabel.

---

<sup>14</sup> Entnommen am 16.11.2023 von <https://bitinfocharts.com/comparison/bitcoin-transactionfees.html>

Ein weiterer Aspekt, der während des Projekts aufgefallen ist, sind die Handels- und Einzahlungsstops auf den Exchanges. Hohe Preisunterschiede zwischen zwei Exchanges kommen meistens deshalb zu stande, weil auf einem Exchange der Handel oder die Einzahlung für eine Kryptowährung gestoppt wurde. Dieser Aspekt macht natürlich eine profitable Handelsmöglichkeit zu nichts. Daher sollte dieser Aspekt in zukünftigen Versionen berücksichtigt werden.

### **Bezug zur Forschungsfrage**

Die Ergebnisse zeigen, dass der Prototyp bereits eine wertvolle Grundlage für Handelsentscheidungen bietet und das Potential aufzeigt, durch weitere Entwicklungen zu einem leistungsfähigeren Werkzeug für den Kryptohandel zu werden. Für eine fundierte Handelsentscheidung sind jedoch weitere Beobachtungen und Informationen erforderlich. Die Identifizierung und Integration dieser zusätzlichen Aspekte in zukünftige Versionen des Prototyps wird entscheidend sein, um dessen Nützlichkeit und Effektivität weiter zu steigern.

Im Hinblick auf die definierten FZ (FZ1O, FZ2T, FZ3S, FZ4E) zeigt sich, dass der Prototyp die Kriterien für eine effiziente Preis- und Kostenanalyse erfüllt (FZ1O) und das entwickelte Konzept umsetzt (FZ2T). Seine prototypische Implementierung (FZ3S) wurde erfolgreich evaluiert (FZ4E), wobei die Ergebnisse aufzeigen, dass weitere Entwicklungen für eine fundierte Handelsentscheidung erforderlich ist. Zusammenfassend lässt sich festhalten, dass die in dieser Arbeit gestellte FF1s teilweise beantwortet werden konnte und das angestrebte FZ, die Entwicklung eines Prototyps, erreicht ist. Der entwickelte Prototyp stellt einen wesentlichen Schritt dar, um die unterschiedlichen Kostenstrukturen auf Kryptowährungsbörsen zu analysieren und profitable Handelsmöglichkeiten aufzuzeigen. Allerdings hat diese Arbeit auch gezeigt, dass weitere Forschung und Entwicklung erforderlich sind, um ein vollständig funktionsfähiges System zu schaffen, das alle Aspekte der Forschungsfrage umfassend adressiert.

## **6.2 Ausblick**

In diesem Ausblick wird die Zukunft des Prototypen beleuchtet. Dazu werden mögliche Erweiterungen und Verbesserungen des Prototypen vorgestellt.

In der aktuellen Version zeigt der Prototyp bereits vielversprechende Ergebnisse, doch

es gilt zu beachten, dass er als Instrument für bedeutende Handelsentscheidungen dient. Anwender sollten grundsätzlich ein fundiertes Verständnis des Kryptomarktes besitzen und die Chancen erkennen können. Nichtsdestotrotz ist es wichtig, den Prototypen durch die Anreichung weiterer Informationen zu optimieren, mit dem Ziel, eine vollautomatisierte und fundierte Handelsentscheidung zu treffen. Die Herausforderung besteht darin, so viel Informationen bereitzustellen, sodass alle unbekannten Kosten berücksichtigt werden. Wenn es gelingt, alle Unsicherheiten zu beseitigen und Handelsentscheidungen zuverlässig zu treffen, dann besteht das Potential, profitable Handelsentscheidungen automatisiert zu treffen. Dies beinhaltet auch die genaue Berücksichtigung jeglicher Gebühren, da selbst geringfügige Kosten den Gewinn schmälern. Sollte es aber gelingen, auch nur einen geringen, aber konstanten Profit automatisiert zu erzielen, so könnte das System potentiell für Tausende von Handlungen pro Tag skaliert werden. Dies könnte beispielsweise durch die Integration weiterer Exchanges realisiert werden, wofür die Infrastruktur und das Design der Anwendung bereits ausgelegt sind. Das Hinzufügen weiterer Exchanges erfordert lediglich die Aktivierung eines zusätzlichen Docker Container im Streamservice, die Anpassung des Consumers im Storageservice und die Erweiterung der Flux-Abfragen.

Allerdings muss auch beachtet werden, dass der Kryptomarkt bereits sehr gesättigt ist und insbesondere auf den größeren Exchanges die Chancen auf profitable Handlungen begrenzt sind. Jüngere und kleinere Exchanges könnten hier zwar größere Gewinnmöglichkeiten bieten, allerdings ist hier das Risiko eines Totalverlustes tendenziell höher. Ein weiterer interessanter Ansatz für zukünftige Entwicklungen könnte darin bestehen, anstatt eine Kryptowährung zwischen zwei Exchanges zu handeln, den Handel verschiedener Kryptowährung innerhalb eines einzigen Exchange zu betrachten. Dies würde bedeuten, dass man kontinuierlich nach drei Währungspaaren sucht, die zusammen einen Gewinn generieren können. Dabei tauscht man alle drei Paare miteinander und nutzt die Differenz der jeweiligen Handelspaare aus. Beispielsweise tauscht man Euro mit Bitcoin, Bitcoin mit Ethereum und Ethereum wieder mit Euro, um einen Gewinn zu erzielen. Der Schlüssel zum Erfolg liegt dabei in der Geschwindigkeit einen profitablen Handel zu erkennen. Hierfür wäre es von Vorteil nicht die CCXT Bibliothek zu verwenden, sondern eine direkte Verbindung über einen WebSocket mit dem Exchange herzustellen, um Preisänderungen noch schneller zu erfassen und zu analysieren. Dabei entstehen aber höhere Handelskosten, da man 3 Trades anstatt 2 Trades durchführt, welche berücksichtigt werden müssen.

Die vorliegenden Ergebnisse und Erkenntnisse bieten eine solide Grundlage für die weitere Entwicklung des Prototypen. Durch die Integration zusätzlicher Informationen und die Automatisierung des Handelsprozesses kann der Prototyp zu einem effektiven

Instrument für Kryptowährungshändler weiter ausgebaut werden.

# Quellenverzeichnis

- [1] S. Nakamoto, „Bitcoin: A Peer-to-Peer Electronic Cash System,“ 2008. Adresse: <https://bitcoin.org/bitcoin.pdf>.
- [2] CoinMarketCap. „Kryptowährungen gelistet nach Börsenwert.“ (2023), Adresse: <https://coinmarketcap.com> (besucht am 24.07.2023).
- [3] J. F. Nunamaker, M. Chen und T. D. M. Purdin, „Systems Development in Information Systems Research,“ *Journal of Management Information Systems*, Jg. 7, Nr. 3, S. 89–106, 1990. Adresse: <http://www.jstor.org/stable/40397957> (besucht am 26.07.2023).
- [4] A. Sharma, F. Khan, D. Sharma und S. Gupta, „Python: The Programming Language of Future,“ *International Journal of Innovative Research in Technology (IJIRT)*, Jg. 6, Nr. 12, S. 115, 2020.
- [5] Google Inc. „Go Documentation.“ (o. D.), Adresse: <https://go.dev/doc/> (besucht am 02.11.2023).
- [6] Microsoft Corporation. „TypeScript.“ Zugegriffen am 25.09.2023. (o. D.), (besucht am 15.09.2023).
- [7] W. Schlaffer, J. Schmeing und P. Kerber. „Digitale Assets – Was ist das? Definition und Entwicklung eines zukünftigen Milliardenmarktes.“ (04/2023), Adresse: <https://bankinghub.de/themen/digitale-assets> (besucht am 22.07.2023).
- [8] E. Zentralbank. „Was ist Geld?“ (11/2015), Adresse: [https://www.ecb.europa.eu/ecb/educational/explainers/tell-me-more/html/what\\_is\\_money.de.html](https://www.ecb.europa.eu/ecb/educational/explainers/tell-me-more/html/what_is_money.de.html) (besucht am 22.07.2023).

- [9] D. J. Nagel, „Geld und Geldpolitik,“ *Deutsche Bundesbank*, 2022. Adresse: <https://www.bundesbank.de/resource/blob/606038/79786120337268ad14bddbb8afbb187b/mL/geld-und-geldpolitik-data.pdf>.
- [10] S. Behringer und F. Follert, „Controlling von Kryptowährungen,“ in *Digitale Transformation im Controlling: Praxisorientierte Lösungsansätze und Chancen für Unternehmen*, J. Hastenteufel, S. Weber und T. Röhm, Hrsg. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, S. 183–195. DOI: 10.1007/978-3-658-38225-4\_12. Adresse: [https://doi.org/10.1007/978-3-658-38225-4\\_12](https://doi.org/10.1007/978-3-658-38225-4_12).
- [11] M. Höning, *ICO und Kryptowährungen*. Abraham-Lincol-Str. 46, 65189 Wiesbaden, Deutschland: Springer Gabler, 2020.
- [12] J. Geroldinger, „Transaktionskosten am Markt für Krypto-Assets,“ Magisterarb., Universität Graz, Universitätspl. 3, 8010 Graz, Österreich, 02/2021. Adresse: <https://unipub.uni-graz.at/obvugrhs/download/pdf/5878330?originalFilename=true>.
- [13] Econotimes. „Mt Gox CEO Re-Arrested For Fraud And Mis-Use Of Funds.“ (05/2015), Adresse: <https://www.econotimes.com/Mt-Gox-CEO-Re-Arrested-For-Fraud-And-Mis-Use-Of-Funds-108625#:~:text=The%20Mt%20Gox%20debacle%20is,6m%29%20of%20customer> (besucht am 20.11.2023).
- [14] J. Hamilton. „FTX Bankruptcy Team Says the Exchange Owed Customers 8.7B.“ (06/2023), Adresse: <https://www.coindesk.com/policy/2023/06/26/ftx-bankruptcy-team-says-the-exchange-owed-customers-87b/#:~:text=About%20%246, the%20report%20filed%20on%20Monday> (besucht am 20.11.2023).
- [15] H. o.V. „Bitcoin und Co.: Die besten Krypto Börsen 2023 im Vergleich.“ (06/2023), Adresse: <https://www.handelsblatt.com/vergleich/krypto-boersen-vergleich/#:~:text=Eine%20Krypto%20B%C3%B6rse%20ist%20eine,%20%2C%20Crypto.com%20oder%20Kraken.> (besucht am 22.07.2023).
- [16] I. Payward. „Kraken.“ (o. D.), Adresse: <https://www.kraken.com/> (besucht am 12.09.2023).

- [17] Bitstamp Ltd. „Bitstamp.“ (o. D.), Adresse: <https://www.bitstamp.net/> (besucht am 12.09.2023).
- [18] A. Kannenberg. „Bitstamp: Erste Bitcoin-Börse erhält EU-Lizenz.“ (04/2016), Adresse: <https://www.heise.de/news/Bitstamp-Erste-Bitcoin-Boerse-erhaelt-EU-Lizenz-3185763.html> (besucht am 22.07.2023).
- [19] Dudenredaktion. „Duden Rechtschreibung.“ (07/2023), Adresse: <https://www.duden.de/rechtschreibung/Daten> (besucht am 22.07.2023).
- [20] L. Wuttke. „Daten: Definition, Arten und Speicherung.“ (06/2022), Adresse: <https://datasolut.com/wiki/daten-definition/> (besucht am 10.08.2023).
- [21] D. M. Schneider, *Implementierungskonzepte für Datenbanksysteme*, 1. Aufl. Springer, 2004.
- [22] E. Rahm, G. Saake und K.-U. Sattler, *Verteiltes und Paralleles Datenmanagement*. Springer Berlin Heidelberg, 2015. DOI: 10.1007/978-3-642-45242-0. Adresse: <http://dx.doi.org/10.1007/978-3-642-45242-0>.
- [23] E. Schicker, *Datenbanken und SQL*. Springer Fachmedien Wiesbaden, 2014. DOI: 10.1007/978-3-8348-2185-0. Adresse: <http://dx.doi.org/10.1007/978-3-8348-2185-0>.
- [24] S. Tönnissen und F. Teuteberg, „Abbildung von Intercompany-Verträgen auf der Blockchain durch Smart Contracts – eine Fallstudie am Beispiel von IT-Services,“ *HMD Praxis der Wirtschaftsinformatik*, Jg. 55, Nr. 6, S. 1167–1184, 2018. DOI: 10.1365/s40702-018-00445-x. Adresse: <https://doi.org/10.1365/s40702-018-00445-x>.
- [25] RedHat. „Was ist eine API?“ (02/2023), Adresse: <https://www.redhat.com/de/topics/api/what-are-application-programming-interfaces> (besucht am 23.07.2023).
- [26] ccxt. „CCXT – CryptoCurrency eXchange Trading Library.“ (2023), Adresse: <https://github.com/ccxt/ccxt> (besucht am 15.10.2023).
- [27] Python Software Foundation. „Python.“ (o. D.), Adresse: <https://www.python.org/> (besucht am 14.09.2023).

- [28] IBM. „Übersicht über Publish/Subscribe.“ (08/2022), Adresse: <https://www.ibm.com/docs/de/integration-bus/10.0?topic=applications-publishsubscribe-overview> (besucht am 15.10.2023).
- [29] A. A. A. Donovan und B. W. Kernighan, *The Go Programming Language* (Addison-Wesley Professional Computing). Addison-Wesley Professional, 2015.
- [30] L. Deri, S. Mainardi und F. Fusco, „tsdb: A Compressed Database for Time Series,“ in *Traffic Monitoring and Analysis*, A. Pescapè, L. Salgarelli und X. Dimitropoulos, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 143–156.
- [31] S. Luber und N. Litzel. „Was ist eine In-Memory-Datenbank?“ (10/2017), Adresse: <https://www.bigdata-insider.de/was-ist-eine-in-memory-datenbank-a-655470/>.
- [32] Vercel, *Next.js Documentation*, o. D. Adresse: <https://nextjs.org/docs> (besucht am 28.10.2023).
- [33] Mozilla Foundation, *JavaScript Guide*, o. D. Adresse: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide> (besucht am 03.11.2023).
- [34] F. Karlstetter. „Was ist ein Webserver?“ (12/2019), Adresse: <https://www.cloudcomputing-insider.de/was-ist-ein-webserver-a-884026/> (besucht am 16.09.2023).
- [35] Docker, Inc. „Docker Documentation.“ Zugriffen am 20.10.2023. (o. D.), Adresse: <https://docs.docker.com/> (besucht am 20.09.2023).
- [36] IBM. „Was ist Docker?“ (2023), Adresse: <https://www.ibm.com/de-de/topics/docker>.
- [37] D. A. Norman und S. W. Draper, *User Centered System Design: New Perspectives on Human-computer Interaction*, 1. Aufl. CRC Press, 1986.
- [38] OMG. „Unified Modeling Language, Version 2.5.1,“ Object Management Group. (12/2017), Adresse: <https://www.omg.org/spec/UML/2.5.1/PDF>.
- [39] InfluxData. „InfluxDB Documentation.“ (o. D.), Adresse: <https://docs.influxdata.com/influxdb/> (besucht am 29.09.2023).

- [40] Redis Labs. „Redis Documentation.“ (o. D.), Adresse: <https://redis.io/docs/> (besucht am 29.09.2023).
- [41] RabbitMQ. „RabbitMQ Documentation.“ (o. D.), Adresse: <https://www.rabbitmq.com/documentation.html> (besucht am 02.10.2023).
- [42] Pika Contributors. „Pika Documentation.“ (o. D.), Adresse: <https://pika.readthedocs.io/en/stable/> (besucht am 03.11.2023).
- [43] Gofiber. „Go Fiber Documentation.“ (o. D.), Adresse: <https://docs.gofiber.io/> (besucht am 10.10.2023).
- [44] Shadcn/ui, *Shadcn/ui Documentation*, o. D. Adresse: <https://ui.shadcn.com/docs> (besucht am 28.10.2023).
- [45] TranStack Query, *TanStack Query Documentation*, o. D. Adresse: <https://tanstack.com/query/v3/docs/react/overview> (besucht am 28.10.2023).
- [46] AXIOS. „AXIOS Documentation.“ (o. D.), Adresse: <https://axios-http.com/docs/intro> (besucht am 28.10.2023).
- [47] Tailwind Labs Inc. „Tailwind CSS Documentation.“ (o. D.), Adresse: <https://tailwindcss.com/docs> (besucht am 30.10.2023).
- [48] Nginx, Inc. „Nginx Documentation.“ (o. D.), Adresse: <https://nginx.org/en/docs/> (besucht am 01.11.2023).
- [49] Cloudflare, Inc. „Cloudflare Documentation.“ (o. D.), Adresse: <https://developers.cloudflare.com> (besucht am 01.11.2023).

# Eidesstattliche Erklärung

Studierender: Marvin Ludwig  
Matrikelnummer: 909130

Hiermit erkläre ich an Eides statt, dass ich diese Arbeit, das Druckexemplar sowie alle nachfolgenden Exemplare selbstständig abgefasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinne nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Ich bin mit einer Plagiatsprüfung einverstanden.

**Digitales Exemplar und Druckexemplar sind identisch.**

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

**29.11.2023, Stadland**

Ort, Abgabedatum

*M. Ludwig*

Unterschrift (Vor- und Zuname)

