BCCM

HOCHSCHULE MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Herr
André Schild

Entwicklung einer modularen Software für den einfachen und sicheren Bitcoin-Handel via Kassensystem und mobiler App

Mittweida, September 2024

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Entwicklung einer modularen Software für den einfachen und sicheren Bitcoin-Handel via Kassensystem und mobiler App

Autor:
André Schild

Studiengang:
Blockchain & Distributed Ledger Technologies

Seminargruppe:
BC22w1-M

Erstprüfer:
Prof. Dr.-Ing. Andreas Ittner

Zweitprüfer:
M.Sc. Tim Käbisch

Einreichung:
Mittweida, 22.09.2024

Verteidigung/Bewertung:
Mittweida, 2024

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Modular software development for simple and secure Bitcoin trading via POS system and mobile app

Author:
André Schild

Course of Study:
Blockchain & Distributed Ledger Technologies

Seminar Group:
BC22w1-M

First Examiner:
Prof. Dr.-Ing. Andreas Ittner

Second Examiner:
M.Sc. Tim Käbisch

Submission:
Mittweida, 22.09.2024

Defense/Evaluation:
Mittweida, 2024

Bibliografische Beschreibung

Schild, André:
Entwicklung einer modularen Software für den einfachen und sicheren Bitcoin-Handel via Kassensystem und mobiler App. – 2024. – 83 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften, Masterarbeit, 2024.

Kurzreferat

Diese Arbeit beschäftigt sich mit der Entwicklung einer modularen Softwarearchitektur für die serverseitige Anwendung der „Cash2Coin“-Plattform, die den Kauf und Verkauf von Bitcoins in Partnerfilialen ermöglicht. Ziel ist es, eine skalierbare, sichere und benutzerfreundliche Lösung zu schaffen, die den Zugang zu Bitcoins vereinfacht und deren Verwahrung transparent und sicher gestaltet.

Im Rahmen eines strukturierten Entwicklungsprozesses werden eine umfassende Anforderungsanalyse durchgeführt, geeignete Architekturlösungen identifiziert und in einem prototypischen System implementiert. Die Softwarearchitektur basiert auf einem schichtenbasierten und modularen Ansatz, bei dem unabhängige Microservices spezifische Aufgaben wie die Benutzerauthentifizierung, die Auftragsabwicklung und die Datenspeicherung übernehmen. Die Microservices werden in einer containerisierten Umgebung bereitgestellt, mit Kubernetes orchestriert und in einer hochverfügbaren und ausfallsicheren Cloud-Umgebung betrieben.

Nutzer und Partnerfilialen können über standardisierte Schnittstellen auf das System zugreifen und Bitcoins kaufen und verkaufen. Zur Einhaltung der regulatorischen Anforderungen werden ein externer Zahlungs- und ein KYC-Dienstleister in die Plattform integriert und alle Transaktionen überwacht.

Diese Arbeit gibt einen praktischen Einblick in die Entwicklung einer technischen Lösung für eine Bitcoin-Handelsplattform und dient als Referenz für zukünftige Softwareprojekte im Bereich des Kryptowährungshandels. Die prototypische Implementierung der Kernkomponenten demonstriert die Eignung der gewählten Architekturlösungen für den produktiven Einsatz und bildet eine solide Basis für die Weiterentwicklung der Plattform.

Abstract

This thesis focuses on the development of a modular software architecture for the server-side application of the “Cash2Coin” platform, which enables the buying and selling of Bitcoin in partner stores. The goal is to create a scalable, secure, and user-friendly solution that simplifies access to Bitcoin and ensures transparent and secure custody.

As part of a structured development process, a comprehensive requirements analysis is conducted, appropriate architectural solutions are identified, and a prototype system is implemented. The software architecture is based on a layered and modular approach, with independent microservices performing specific tasks such as user authentication, order processing and data storage. The microservices are deployed in a containerized environment, orchestrated with Kubernetes, and operated in a highly available and resilient cloud environment.

Users and partner stores can access the system through standardized interfaces to buy and sell Bitcoin. To comply with regulatory requirements, an external payment service provider and a KYC provider are integrated into the platform, and all transactions are monitored.

This thesis provides practical insights into the development of a technical solution for a Bitcoin trading platform and serves as a reference for future software projects in the cryptocurrency trading space. The prototype implementation of the core components demonstrates the suitability of the chosen architectural solutions for production use and provides a solid foundation for further platform development.

Inhaltsverzeichnis

1 Einleitung

Die fortschreitende Digitalisierung hat in den letzten Jahrzehnten zu tiefgreifenden Veränderungen in nahezu allen Wirtschaftsbereichen geführt, die sich insbesondere auch auf den Finanzsektor auswirken. Die internationale Finanzkrise 2008 führte zu einem Vertrauensverlust in das traditionelle Finanzsystem [Rot09]. Fast die Hälfte der deutschen Bevölkerung verlor infolge der Krise das Vertrauen in die Banken [Bau14]. Satoshi Nakamoto veröffentlichte im Oktober 2008 das *Bitcoin-Whitepaper* [Nak08a; Nak08b] während die US-Aktienkurse den größten Wochenverlust in der Geschichte der Wall Street erlitten [Aut+08].

Die Kernidee von Bitcoin ist die Schaffung eines bankenunabhängigen Peer-to-Peer-Zahlungsnetzwerks und einer digitalen Währung (*Kryptowährung*) auf Basis einer dezentralen Datenbank, der sogenannten Blockchain [Nak08a]. Sechzehn Jahre später, im Jahr 2024, ist Bitcoin mit einer Marktkapitalisierung von 1,2 Billionen US-Dollar die weltweit größte Kryptowährung [Coi24a]. Bitcoins können zentral über Kryptobörsen oder dezentral über Peer-to-Peer-Plattformen und Bitcoin-Geldautomaten gehandelt werden.

Diese Arbeit beschäftigt sich mit der Entwicklung einer modularen Softwarearchitektur für eine Plattform, die den Kauf und Verkauf von Bitcoins in Partnerfilialen wie Supermärkten und kleinen Verkaufsstellen ermöglicht. Der Fokus liegt auf der Schaffung einer benutzerfreundlichen Lösung, die den Zugang zu Bitcoins vereinfacht und die Verwahrung transparent und sicher gestaltet.

1.1 Motivation

Trotz der zunehmenden Verbreitung digitaler Währungen besitzen laut einer Umfrage aus dem Jahr 2023 lediglich 13 Prozent der deutschen Bevölkerung Kryptowährungen [Bra23]. Die Mehrheit der Befragten hat noch nie in Kryptowährungen investiert. Die Hauptgründe für die geringe Akzeptanz sind mangelndes Wissen über die Funktionsweise von Bitcoin sowie Sicherheitsbedenken und unsichere regulatorische Rahmenbedingungen [Blo22]. Eine internationale Umfrage aus dem Jahr 2022 hat ergeben, dass 22 Prozent der Befragten nicht wissen, wie sie Zugang zu Bitcoins erhalten und daher noch keine Bitcoins gekauft haben [Blo22].

Für unerfahrene Nutzer stellt der Erwerb von Bitcoins eine Herausforderung dar. Der Anmeldeprozess bei einer Kryptobörse und die Verifizierung der Zahlungsmethode sind oft zeitaufwändig und komplex [Bin19]. Peer-to-Peer-Plattformen erfordern ein hohes Maß an Vertrauen in den Handelspartner und stellen ein hohes Betrugsrisiko dar [TCC24]. Bitcoin-Geldautomaten sind in Deutschland nur vereinzelt verfügbar und erheben häufig hohe Gebühren [Coi24b]. Es besteht somit ein Bedarf an einer benutzerfreundlichen Lösung, die den Bitcoin-Handel vereinfacht und die anschließende Verwahrung transparent und sicher gestaltet.

Das Projekt „Cash2Coin“ soll diese Lücke schließen und eine Brücke zwischen der traditionellen Finanzwelt und der Welt der Kryptowährungen schlagen. Dies soll durch eine benutzerfreundliche Plattform erreicht werden, die den Kauf und Verkauf von Bitcoins in den Partnerfilialen ermöglicht. Die Plattform besteht aus einer mobilen Kundenanwendung und einem Kassensystem für die Partnerfilialen. Die mobile Anwendung soll eine benutzerfreundliche Oberfläche bieten und den Nutzer durch den gesamten Kauf- und Verkaufsprozess führen. Die Partnerfilialen werden mit einem Kassensystem ausgestattet, das das Verkaufspersonal durch den Kauf- und Verkaufsprozess leitet.

Im Rahmen dieser Arbeit wird ein systematisch strukturierter und modularer Aufbau des zugrundeliegenden Softwaresystems – die Softwarearchitektur – entwickelt [Den13]. Aufgrund der Komplexität und der hohen Anforderungen an die Sicherheit und Zuverlässigkeit der Plattform ist eine sorgfältige Planung der Softwarearchitektur erforderlich. Mögliche Risiken und Schwachstellen müssen frühzeitig identifiziert und durch geeignete Maßnahmen adressiert werden. Die Plattform soll einen einfachen und sicheren Zugang zu Bitcoin ermöglichen und die Verbreitung und Akzeptanz von Bitcoin als digitale Währung fördern.

1.2 Methodik

Die Entwicklung der Softwarearchitektur orientiert sich an bewährten Prinzipien der Softwaretechnik und folgt einem strukturierten Entwicklungsprozess. Zunächst erfolgt eine umfassende Anforderungsanalyse, in der sowohl die funktionalen als auch die nicht-funktionalen Anforderungen an das System präzise definiert werden. Dabei werden insbesondere Aspekte der Skalierbarkeit, Sicherheit und Benutzerfreundlichkeit berücksichtigt.

Die funktionalen Anforderungen beschreiben das erwartete Verhalten und die Kernfunktionen der Software [Bal10]. Diese Anforderungen werden in Anwendungsfällen beschrieben und in Use-Case-Diagrammen visualisiert. Die nicht-funktionalen Anforderungen umfassen die Qualitätsmerkmale des Systems [Bal10]. Die *ISO/IEC 25010* definiert acht Qualitätsmerkmale zur Bewertung von Softwareprodukten [Hao+17], die in der Anforderungsanalyse berücksichtigt und an die projektspezifischen Anforderungen angepasst werden.

Auf Grundlage der Ergebnisse der Anforderungsanalyse erfolgt der Architekturentwurf. Wichtige Einflussfaktoren werden identifiziert und ihr Einfluss auf die Architektur untersucht. Zur Adressierung von Risiken und Schwachstellen der Architektur werden geeignete Lösungsstrategien entwickelt. Basierend auf der Anforderungsanalyse und den Lösungsstrategien werden konkrete Architekturentscheidungen getroffen.

Die Softwarearchitektur soll eine hohe Flexibilität und Erweiterbarkeit gewährleisten und wird daher modular und mehrschichtig aufgebaut. Die Software wird in unabhängige Module unterteilt, von denen jedes eine klar definierte Aufgabe erfüllt. Die Kopplung zwischen den Modulen soll dabei gering und die Kohäsion innerhalb der Module hoch sein. Die Kommunikation zwischen den Modulen erfolgt über standardisierte Schnittstellen und Protokolle. Es wird eine Drei-Schichten-Architektur implementiert, die aus einer Präsentationsschicht, einer Anwendungsschicht und einer Persistenzschicht besteht [DH13]. Die Präsentationsschicht stellt die Benutzeroberfläche der mobilen Kundenanwendung, des Kassensystems und weiterer

Systemkomponenten dar. Die Anwendungsschicht realisiert die fachlichen Funktionalitäten des Systems und bildet die Geschäftsprozesse ab [DH13]. Die Persistenzschicht realisiert die dauerhafte Speicherung der Anwendungsdaten in einer Datenbank [DH13]. Die Entwicklung der Präsentationsschicht ist nicht Gegenstand dieser Arbeit.

Aufbauend auf dem Architekturentwurf werden ausgewählte Kernkomponenten der Software implementiert und die Bitcoin-Kauffunktion in einer realitätsnahen Umgebung getestet. Der Fokus liegt dabei auf der technischen Umsetzung der Anforderungen und der Integration der Schnittstelle zu Drittanbietern. Wirtschaftliche und regulatorische Aspekte werden daher weniger berücksichtigt. Die Ergebnisse dieser Arbeit bilden die Grundlage für die Entwicklung und den Betrieb einer technisch funktionsfähigen Anwendung und sollen als Referenz für zukünftige Projekte dienen.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in sechs Kapitel, die den Entwicklungsprozess der Softwarearchitektur systematisch darstellen. Das Strukturdiagramm in Abbildung 1.1 veranschaulicht diesen Prozess und wird im Verlauf der Arbeit schrittweise vervollständigt. Das folgende Kapitel behandelt die theoretischen Grundlagen der Softwaretechnik und der Blockchain-Technologie, die für das Verständnis der Arbeit von zentraler Bedeutung sind. Im dritten Kapitel erfolgt die Anforderungsanalyse, in der die funktionalen und nicht-funktionalen Anforderungen an das System definiert werden. Aufbauend auf den Anforderungen erfolgt im vierten Kapitel der Architekturentwurf, der die Identifikation von Einflussfaktoren, die Entwicklung von Lösungsstrategien und die darauf basierenden Architekturentscheidungen umfasst. Das fünfte Kapitel beschreibt die prototypische Implementierung von der Einrichtung der Entwicklungsumgebung bis zur Integration der API des Zahlungsdienstleisters. Abschließend werden im letzten Kapitel die Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf mögliche Weiterentwicklungen der Plattform und die damit verbundenen Herausforderungen gegeben. Diese Struktur ermöglicht eine klare und nachvollziehbare Darstellung des Entwicklungsprozesses und der Ergebnisse der Arbeit.

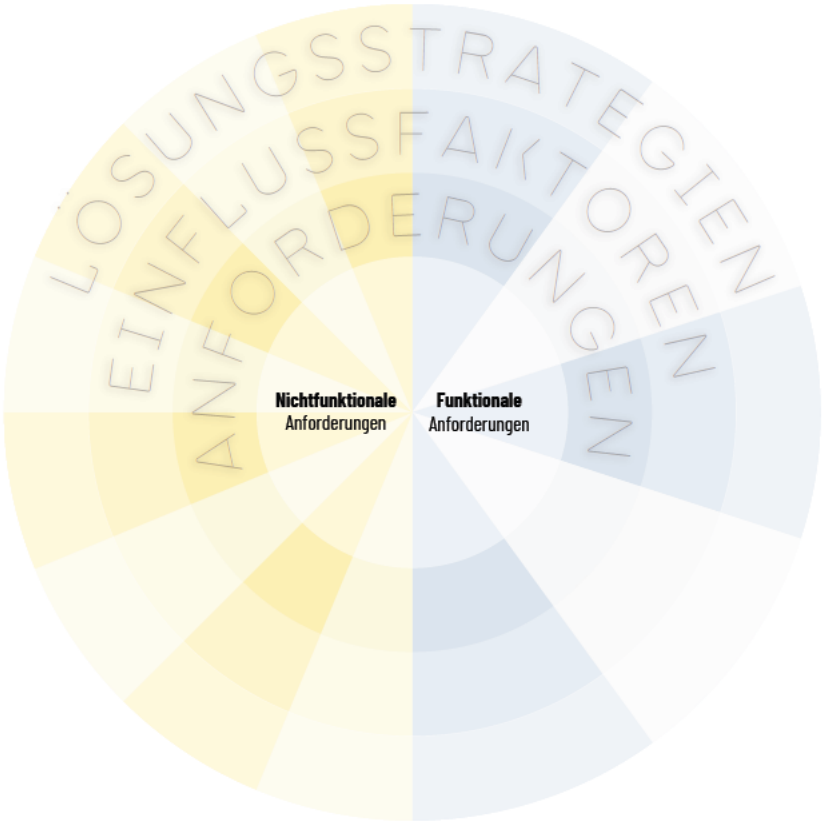

Abbildung 1.1: Strukturierter Entwicklungsprozess

2 Grundlagen

Im Folgenden werden grundlegende Begriffe und Konzepte im Zusammenhang mit der Softwaretechnik und den eingesetzten Technologien erläutert.

2.1 Softwaretechnik

Die Softwaretechnik (*Software Engineering*) umfasst die systematische Anwendung von Prinzipien, Methoden und Werkzeugen zur ingenieurmäßigen Entwicklung komplexer Softwaresysteme [Bal96; Bal10]. Ein Softwaresystem ist ein System, das aus einer Vielzahl von Computerprogrammen, Konfigurationsdateien und weiteren Softwarekomponenten besteht, die zur Erfüllung der Anforderungen an ein bestimmtes Projekt erforderlich sind [Som11; Den13].

2.1.1 Entwicklungsprozess

Der Entwicklungsprozess beschreibt das systematische Vorgehen bei der Entwicklung eines Softwaresystems. In einem Vorgehensmodell wird der Entwicklungsprozess spezifiziert und die Reihenfolge der durchzuführenden Aktivitäten [Den13] festgelegt. Man unterscheidet zwischen klassischen linearen Vorgehensmodellen wie dem *Wasserfallmodell* und agilen Vorgehensmodellen wie *Scrum*. *DevOps-Praktiken* unterstützen eine effiziente und automatisierte Umsetzung des Entwicklungsprozesses [CLT23].

Wasserfallmodell

Das *Wasserfallmodell* ist ein klassisches lineares Vorgehensmodell (definiert 1970 von Royce [Roy70]), das den Entwicklungsprozess in sequentielle Phasen unterteilt, die nacheinander durchlaufen werden (vgl. Abb. 2.1) [AA13]. In der ersten Phase – der *Anforderungsanalyse* – werden die Anforderungen an das System ermittelt und dokumentiert. Nach Abschluss der Anforderungsanalyse wird die *Softwarearchitektur* entworfen [Den13; RP97]. Anschließend erfolgen Implementierung und Test sowie Inbetriebnahme und Wartung des Systems [Den13; RP97].

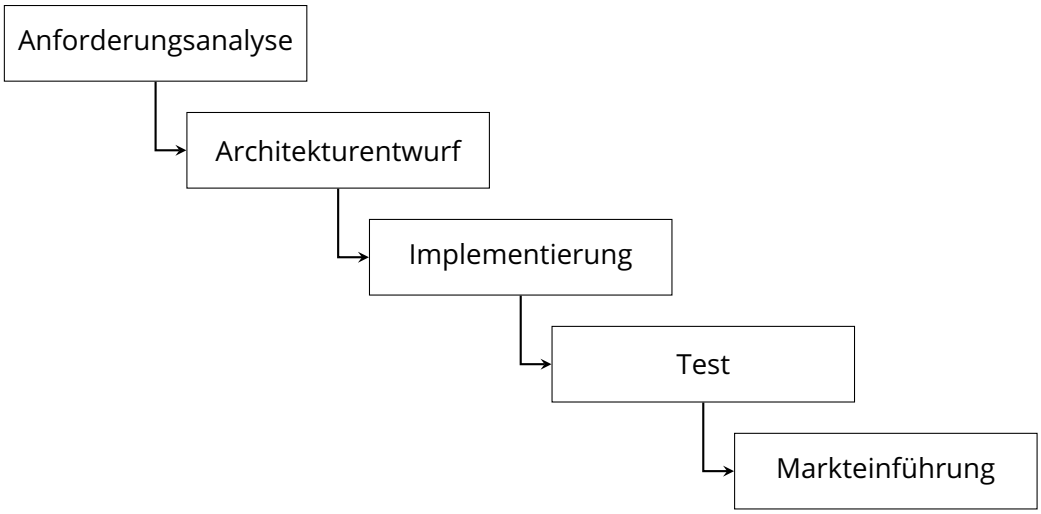

Abbildung 2.1: Lineares Vorgehensmodell: Wasserfallmodell [AA13]

Das Wasserfallmodell eignet sich für Projekte mit fest definierten Anforderungen und klaren Zielen [AA13; ZS20]. Während des Entwicklungsprozesses können nur sehr wenige Änderungen an den Anforderungen vorgenommen werden [ZS20].

Scrum

Für komplexe Projekte mit dynamischen Anforderungen empfiehlt sich ein *agiles Vorgehensmodell* [ZS20; Hem+20]. *Scrum* ist ein agiles Vorgehensmodell für die Entwicklung komplexer Softwaresysteme in größeren Teams. Scrum basiert auf einem iterativen und inkrementellen Entwicklungsprozess, der in mehreren kurzen Entwicklungszyklen organisiert ist (vgl. Abb. 2.2) [ZS20]. Ein Entwicklungszyklus wird als *Sprint* bezeichnet und dauert in der Regel zwei bis vier Wochen [ZS20].

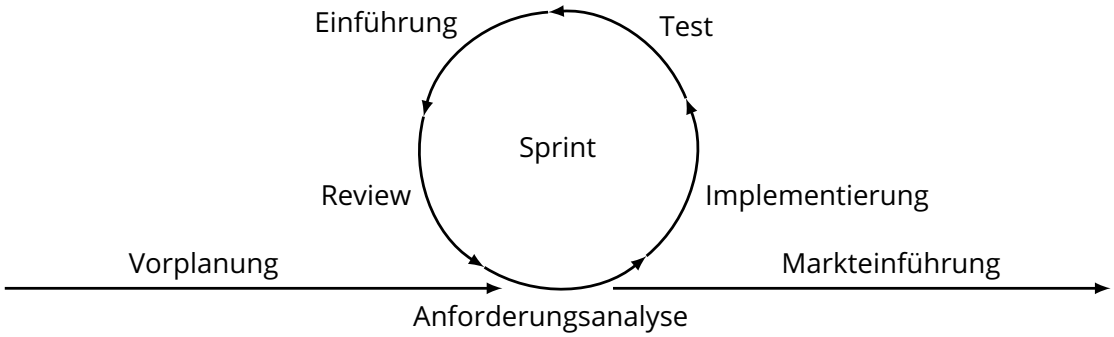

Abbildung 2.2: Agiles Vorgehensmodell: Scrum

Scrum ermöglicht eine flexible Anpassung an sich ändernde Anforderungen und fördert die kontinuierliche Integration von Kundenfeedback in den Entwicklungsprozess [Hem+20]. Es steigert die Zusammenarbeit und Selbstorganisation im Team, was zu einer effizienteren Projektabwicklung und einer höheren Zufriedenheit aller Beteiligten führt [Hem+20].

DevOps

DevOps ist eine Sammlung von Praktiken und Werkzeugen, um die Zusammenarbeit zwischen Entwicklung (***Dev****elopment*) und Betrieb (***Op****erations*) zu verbessern und den Entwicklungsprozess zu automatisieren. Entwicklungsaufgaben wie die Implementierung, Test und Bereitstellung von Software werden automatisiert und in einem kontinuierlichen Prozess durchgeführt [CLT23]. Die Automatisierung des Entwicklungsprozesses ermöglicht eine schnellere und zuverlässigere Bereitstellung von Software und reduziert das Risiko menschlicher Fehler [CLT23].

Ein zentrales Element von DevOps ist die kontinuierliche Integration (*Continuous Integration*) und Bereitstellung (*Continuous Deployment*) von Software (kurz *CI/CD*). Softwareänderungen werden regelmäßig in ein gemeinsames *Repository* (Versionsverwaltungssystem) integriert und automatisiert getestet [CLT23]. Nach erfolgreichem Abschluss der Tests wird die Software automatisch in einer Produktionsumgebung bereitgestellt [CLT23]. CI ermöglicht eine frühzeitige Fehlererkennung und eine kontinuierliche Verbesserung der Codequalität. CD ermöglicht eine schnelle und zuverlässige Auslieferung von Softwareänderungen und trägt zu einer hohen Verfügbarkeit des Softwaresystems bei. Die *CI/CD Pipeline* umfasst die Automatisierung des gesamten Entwicklungsprozesses, von der Integration der Software in ein gemeinsames Repository, über automatisierte Tests bis hin zur Bereitstellung in einer Produktionsumgebung [CLT23]. Durch die Automatisierung des Entwicklungsprozesses können Softwareänderungen häufiger und in kleineren Schritten bereitgestellt werden, was zu einer höheren Codequalität, geringeren Kosten und einer schnelleren und zuverlässigeren Bereitstellung führt [CLT23].

GitLab ist eine DevOps-Plattform mit integrierten Werkzeugen zur Unterstützung von CI/CD Pipelines [CLT23]. GitLab bietet eine einheitliche Benutzeroberfläche für die Zusammenarbeit zwischen Entwicklung und Betrieb und ermöglicht die Integration externer Werkzeuge zur Automatisierung des Entwicklungsprozesses [CLT23]. GitLab unterstützt die Entwicklung von *Cloud-nativen Anwendungen* durch die Integration von *GitLab Runners* in *Kubernetes*. *GitLab Runners* sind spezielle Agenten, die CI/CD Pipelines in einer skalierbaren und hochverfügbaren *Kubernetes-Umgebung* ausführen [CLT23].

2.1.2 Softwarearchitektur

Die Softwarearchitektur bildet die Schnittstelle zwischen den definierten Anforderungen und der konkreten Implementierung eines Softwaresystems. Die Architektur definiert den grundsätzlichen strukturellen Aufbau des Softwaresystems, die wesentlichen Systemkomponenten und deren Beziehungen untereinander [Gar08].

Modellierung

Die UML (*Unified Modeling Language*) ist eine standardisierte Notation zur Modellierung von Softwaresystemen [Med+02]. Die UML umfasst verschiedene Diagrammtypen zur Visualisierung und Dokumentation unterschiedlicher Aspekte der Softwarearchitektur. Die Diagramme

können auf verschiedenen Abstraktionsebenen erstellt werden, um den Entwicklungsprozess von der Anforderungsanalyse bis hin zur konkreten Implementierung zu unterstützen [Med+02].

Use-Case-Diagramme beschreiben die Interaktionen mit dem Softwaresystem gemäß den definierten Anforderungen. *Klassendiagramme* modellieren die statische Struktur des Softwaresystems und zeigen die Klassen und ihre Beziehungen. *Komponentendiagramme* beschreiben die physische Struktur des Softwaresystems, indem sie die Komponenten und ihre Schnittstellen darstellen. *Sequenzdiagramme* modellieren die dynamische Struktur des Softwaresystems und zeigen die zeitliche Abfolge der Interaktionen zwischen den Komponenten.

Entwurfsprinzipien

Entwurfsprinzipien sind allgemeine, aus der Praxis der Softwareentwicklung abgeleitete Grundsätze zur Schaffung einer robusten und anpassungsfähigen Softwarearchitektur. Zu den zentralen Entwurfsprinzipien gehören die Förderung der Erweiterbarkeit und Wartbarkeit durch die Schaffung redundanzfreier, kohärenter und lose gekoppelter Systemkomponenten [DH13]. Diese Prinzipien minimieren die Abhängigkeiten zwischen Systemkomponenten und führen zu einer robusten Softwarearchitektur [DH13].

Das Prinzip der *Separation of Concerns* (SoC) beschreibt den Ansatz, ein komplexes Problem in kleinere, voneinander unabhängige Teilprobleme zu zerlegen und diese getrennt zu betrachten [MEM04]. Handelt es sich bei den Teilproblemen um bekannte Problemklassen, können standardisierte Entwurfsmuster zur Lösung herangezogen werden [DH13]. Die Anwendung des SoC-Prinzips ist auf verschiedenen Ebenen der Softwarearchitektur möglich – beispielsweise durch die Einführung einer *Schichtenarchitektur* oder die Zerlegung des Softwaresystems in einzelne Module (vgl. *Entwurfsmuster*) [DH13]. Dieses Vorgehen sorgt für eine klare Trennung der Verantwortlichkeiten und fördert die Verständlichkeit, die Wartbarkeit und die Wiederverwendbarkeit des Softwaresystems [MEM04].

Das *Geheimnisprinzip* (*Information Hiding*) beschreibt den Ansatz, die Implementierungsdetails einer Systemkomponente vor dem Zugriff durch andere Komponenten zu verbergen [LS18]. Durch das Verbergen der internen Struktur einer Komponente wird die Abhängigkeit von anderen Komponenten reduziert und die Wartbarkeit und Erweiterbarkeit des Softwaresystems erhöht.

Entwurfsmuster

Entwurfsmuster bieten Lösungen für wiederkehrende Entwurfsprobleme in der Softwareentwicklung und werden auf verschiedenen Ebenen der Softwarearchitektur eingesetzt. Man unterscheidet zwischen *Architekturmustern* und *Designmustern*. Architekturmuster dienen der Strukturierung und Zerlegung von Softwaresystemen auf einer höheren Abstraktionsebene – der sogenannten *Makroarchitektur* – und helfen, grundlegende Ordnungsprinzipien zu etablieren [DH13]. Designmuster adressieren spezifische Entwurfsprobleme auf einer niedrigeren Abstraktionsebene – der sogenannten *Mikroarchitektur* – und unterstützen die Implementierung von Systemkomponenten [DH13].

Eine *Schichtenarchitektur* ist ein Architekturmuster zur Strukturierung eines Softwaresystems in mehrere unabhängige Schichten, die jeweils spezifische Aufgaben erfüllen [DH13]. Jede Schicht stellt ihre Funktionalität über eine definierte Schnittstelle zur Verfügung und kennt nur die Schnittstellen der benachbarten Schichten. Eine *Drei-Schichten-Architektur* gliedert das Softwaresystem in eine *Präsentationsschicht* (Benutzeroberfläche), eine *Anwendungsschicht* (Geschäftslogik) und eine *Persistenzschicht* (Datenzugriff) [DH13]. Diese Umsetzung ermöglicht eine klare Trennung der Verantwortlichkeiten (SoC-Prinzip) und eine effiziente Verteilung der Schichten auf mehrere physische Rechner.

Eine *modulare Architektur* ist ein Architekturmuster, das ein Softwaresystem in unabhängige Module zerlegt, von denen jedes eine bestimmte Funktionalität bereitstellt (SoC-Prinzip). Durch diese Zerlegung wird ein komplexes Softwaresystem leichter verständlich und handhabbar [BC00]. Die Elemente eines Moduls sind intern stark (hohe Kohäsion) und mit den Elementen anderer Module schwach verbunden (lose Kopplung) [BC00; DH13]. Die Funktionalität eines Moduls wird durch eine schmale Schnittstelle nach außen zur Verfügung gestellt, während die Komplexität im Inneren verborgen bleibt (Geheimnisprinzip).

Eine *ereignisgesteuerte Architektur* (*Event-Driven Architecture*, EDA) ist ein Architekturmuster für die Entwicklung eines Softwaresystems, das aus einer Gruppe von stark entkoppelten, autonomen Komponenten besteht, die asynchron auf Ereignisse reagieren [Lai+20]. Ein Ereignis (*Event*) ist eine Zustandsänderung oder ein Signal, das von einer Komponente erzeugt und an andere Komponenten weitergeleitet wird [Lai+20]. Anstatt direkt miteinander zu kommunizieren (synchron), reagieren die Komponenten auf eine vordefinierte Menge von Ereignissen und lösen entsprechende Aktionen aus (asynchron) [Lai+20]. Ein *Event Consumer* kann sich für bestimmte Events registrieren und erhält automatisch Aktualisierungen, sobald ein Event eintritt [Ama24; Lai+20]. Ein *Event Producer* erzeugt ein Event und leitet es an einen *Event Router* weiter, der die Events filtert und an den entsprechenden *Event Consumer* weiterleitet [Ama24]. Event Producer und Event Consumer sind somit entkoppelt und können unabhängig voneinander entwickelt, bereitgestellt und skaliert werden [Ama24; Lai+20]. Die EDA ist insbesondere für Softwaresysteme relevant, die in Echtzeit auf Ereignisse reagieren müssen. In Kombination mit einer *Microservice-Architektur* ermöglicht eine EDA eine hohe Skalierbarkeit und Wartbarkeit des Softwaresystems [Lai+20].

Eine *API* (*Application Programming Interface*) ist eine Schnittstelle für den Zugriff auf einzelne Systemkomponenten über definierte Methoden und Protokolle [MSS18]. Interne Implementierungsdetails werden durch die API verborgen und nur definierte Schnittstellen für den Zugriff auf die Funktionalität bereitgestellt (Geheimnisprinzip). Die Verwendung von APIs ermöglicht die Entkopplung von Systemkomponenten und deren Wiederverwendung in unterschiedlichen Kontexten [MSS18].

Architekturstil

Ein *Architekturstil* charakterisiert eine Klasse von Softwaresystemen durch gemeinsame strukturelle und semantische Eigenschaften. Es werden spezifische Designelemente, Regeln für deren Zusammensetzung und semantische Bedeutungen (Funktionalität und Verhalten der Systemkomponenten zur Erreichung des gewünschten Systemverhaltens) definiert [Mon+97].

Dies fördert die Verständlichkeit und Wartbarkeit der Softwarearchitektur und unterstützt die Wiederverwendung von Systemkomponenten [Mon+97]. Die Umsetzung eines Architekturstils in einem konkreten Softwaresystem erfolgt durch die Auswahl geeigneter Entwurfsmuster und die Einhaltung definierter Entwurfsprinzipien.

2.1.3 Verteilte Softwaresysteme

Verteilte Softwaresysteme bestehen aus einer Menge unabhängiger Systemkomponenten, die auf mehreren autonomen Rechnern in einem Netzwerk verteilt sind [DH13]. Eine verteilte Architektur ermöglicht eine horizontale Skalierung und trägt zu einer hohen Verfügbarkeit des Softwaresystems bei [DH13]. Die Kommunikation zwischen den Systemkomponenten erfolgt mittels Interprozesskommunikation über das Netzwerk [DH13]. Die Entwicklung verteilter Softwaresysteme erfordert die Berücksichtigung von Latenzzeiten, Netzwerkfehlern und die Auswahl geeigneter Architekturstile [Boo04]. Zwei gängige Architekturstile für verteilte Softwaresysteme sind die *serviceorientierte Architektur* und die *Microservice-Architektur* [Sta24]. *Webservices* bieten standardisierte Schnittstellen und Kommunikationsprotokolle für die Interaktion und den Datenaustausch in verteilten Softwaresystemen [Boo04]. Ein *API Gateway* bietet eine zentrale Schnittstelle für den Zugriff auf verteilte Systemkomponenten und ermöglicht die einfache Erweiterung und Skalierung des Gesamtsystems [TLP18]. *Cloud-native Anwendungen* sind speziell für die Ausführung in einer Cloud-Umgebung entwickelte Softwaresysteme, die die Vorteile der Cloud-Infrastruktur nutzen [GBS17].

Serviceorientierte Architektur

Eine *serviceorientierte Architektur* (*SOA*) ist ein Architekturstil für die Entwicklung verteilter Softwaresysteme auf der Basis wiederverwendbarer *Services* [CDT18]. Ein Service kapselt die Funktionalität einer Systemkomponente und stellt diese über eine standardisierte Schnittstelle zur Verfügung [Boo04]. Die interne Struktur eines Services bleibt dem aufrufenden System verborgen (Geheimnisprinzip) [Boo04]. Durch die lose gekoppelte Architektur können Services unabhängig voneinander entwickelt, getestet und in unterschiedlichen Kontexten wiederverwendet werden [Boo04]. Einzelne Services werden zu komplexen Anwendungen kombiniert, die auf mehreren physischen Rechnern in einem Netzwerk verteilt sind [Boo04]. Die Steuerung der Services erfolgt über ein zentrales System, das die Interaktion und den Datenaustausch zwischen den Services koordiniert und die Geschäftsprozesse des Gesamtsystems überwacht [CDT18]. Die zentrale Steuerung führt jedoch zu einer hohen Komplexität und einer monolithischen Struktur des Gesamtsystems [CDT18].

Microservice-Architektur

Eine *Microservice-Architektur* ist ein Architekturstil für die Entwicklung verteilter Softwaresysteme auf der Basis kleiner, unabhängiger *Microservices* [CDT18]. Die Microservice-Architektur ist eine Weiterentwicklung der SOA und zielt auf eine höhere Flexibilität und Skalierbarkeit ab [CDT18]. Im Gegensatz zur SOA, bei der die Steuerung der Services zentral erfolgt, folgt die Microservice-Architektur dem Prinzip der Dezentralität und Autonomie der Services [CDT18]. Jeder Microservice läuft in einem eigenen Prozess und kommuniziert über leichtgewichtige

Mechanismen wie *REST* oder *GraphQL* [CDT18; BHJ16]. Einzelne Services können unabhängig voneinander und in unterschiedlichen Programmiersprachen entwickelt, getestet und bereitgestellt werden [TLP18; BHJ16]. Die Verteilung der Services auf mehrere physische Rechner ermöglicht eine horizontale Skalierung und erhöht die Ausfallsicherheit des Gesamtsystems [CDT18]. Die Microservice-Architektur ermöglicht die kontinuierliche Integration und Bereitstellung von Softwareänderungen (CI/CD) und die Entwicklung von *Cloud-nativen Anwendungen* [TLP18].

Webservices

Ein *Webservice* ist eine spezielle Form einer API, die über ein Netzwerk zugänglich ist und die Interaktion und den Datenaustausch in verteilten Softwaresystemen realisiert [Boo04]. Webservices bieten eine plattform- und programmiersprachenunabhängige Möglichkeit, Netzwerkdienste bereitzustellen und zu nutzen [Boo04; BV20]. Ein Architekturstil definiert, wie Ressourcen zugänglich gemacht und Daten zwischen Systemen ausgetauscht werden. Dabei wird die Einhaltung definierter Schnittstellen und Kommunikationsprotokolle festgelegt. Zwei gängige Architekturstile für Webservices sind *REST* und *GraphQL* [Pos23; Kam+23].

REST (*Representational State Transfer*) ist ein Architekturstil für verteilte Softwaresysteme, der auf dem HTTP-Protokoll basiert [BV20]. *RESTful Webservices* verwenden standardisierte HTTP-Methoden (GET, POST, PUT, DELETE) zum Zugriff auf Ressourcen über dedizierte Endpunkte [Boo04]. RESTful Webservices sind zustandslos und erfordern keine Sitzungsinformationen zwischen den Anfragen [Boo04]. Eine *RESTful API* ist eine spezifische Implementierung des REST-Architekturstils, die einen einfachen Zugriff auf Ressourcen in einem verteilten System über standardisierte HTTP-Methoden ermöglicht.

GraphQL (*Graph Query Language*) ist ein flexibles Abfragesystem und eine Laufzeitumgebung für APIs, die es ermöglicht, Daten als Graphen zu modellieren und präzise abzurufen [The24]. Im Gegensatz zu REST, wo Daten über mehrere Endpunkte abgerufen werden, bietet GraphQL einen einzigen Endpunkt, über den präzise Abfragen definiert und Daten effizient abgerufen werden können [BV20; The24]. GraphQL ermöglicht die einfache Weiterentwicklung von APIs, indem neue Informationen hinzugefügt werden können, ohne bestehende Funktionen zu beeinträchtigen [The24]. Es stellt sicher, dass die Struktur der API klar und nachvollziehbar bleibt und im Fehlerfall verständliches Feedback gegeben werden kann [The24]. GraphQL kann in einem Architekturstil eingebettet werden, um spezifische Anforderungen an die Interaktion und den Datenaustausch zu erfüllen.

API Gateway

Ein *API Gateway* fungiert als zentraler Einstiegspunkt in ein verteiltes Softwaresystem [TLP18]. Es leitet Anfragen an einen oder mehrere Microservices weiter, aggregiert die Ergebnisse und stellt den Clients eine einheitliche Schnittstelle zur Verfügung [TLP18]. Das API Gateway kann auch als Lastverteiler eingesetzt werden, indem es die Anfragen auf Basis der Verfügbarkeit und des Standorts der Services optimiert [TLP18]. API Gateways ermöglichen eine einfache Erweiterung des Systems, verbessern die Skalierbarkeit und erleichtern die Interaktion zwischen Clients und Microservices [TLP18]. Allerdings erhöht das API Gateway die Komplexität der Architektur und kann einen potenziellen Flaschenhals darstellen [TLP18].

Cloud-native Anwendungen

Eine *Cloud-native Anwendung* ist ein Softwaresystem, das speziell für die Ausführung in einer Cloud-Umgebung entwickelt wird [GBS17]. Cloud-native Anwendungen basieren in der Regel auf Microservices, die in *Containern* ausgeführt und *dynamisch orchestriert* werden, um eine flexible und effiziente Ressourcennutzung zu gewährleisten [GBS17]. Diese Anwendungen sind darauf ausgelegt, in einer verteilten Umgebung zu arbeiten und trotz möglicher Ausfälle der Infrastruktur einen unterbrechungsfreien Betrieb zu gewährleisten [GBS17]. Sie nutzen die Vorteile der Cloud-Umgebung: Hochverfügbarkeit, globale Skalierbarkeit und flexible Ressourcennutzung [GBS17].

2.1.4 Virtualisierungstechnologien

Virtualisierungstechnologien ermöglichen die Ausführung mehrerer Betriebssysteme und Anwendungen auf einer einzelnen physischen Maschine [Pot+20]. Zwei gängige Virtualisierungstechnologien sind die *Hypervisor-basierte Virtualisierung* und die *Containerisierung* [Pot+20]. Die Hypervisor-basierte Virtualisierung ermöglicht die Ausführung mehrerer isolierter Betriebssysteme auf einer gemeinsamen Hardwareplattform (vgl. Abb. 2.3 links) [Pot+20]. Die Containerisierung ermöglicht die Ausführung mehrerer isolierter Anwendungen auf einem gemeinsamen Betriebssystem (vgl. Abb. 2.3 rechts) [Pot+20].

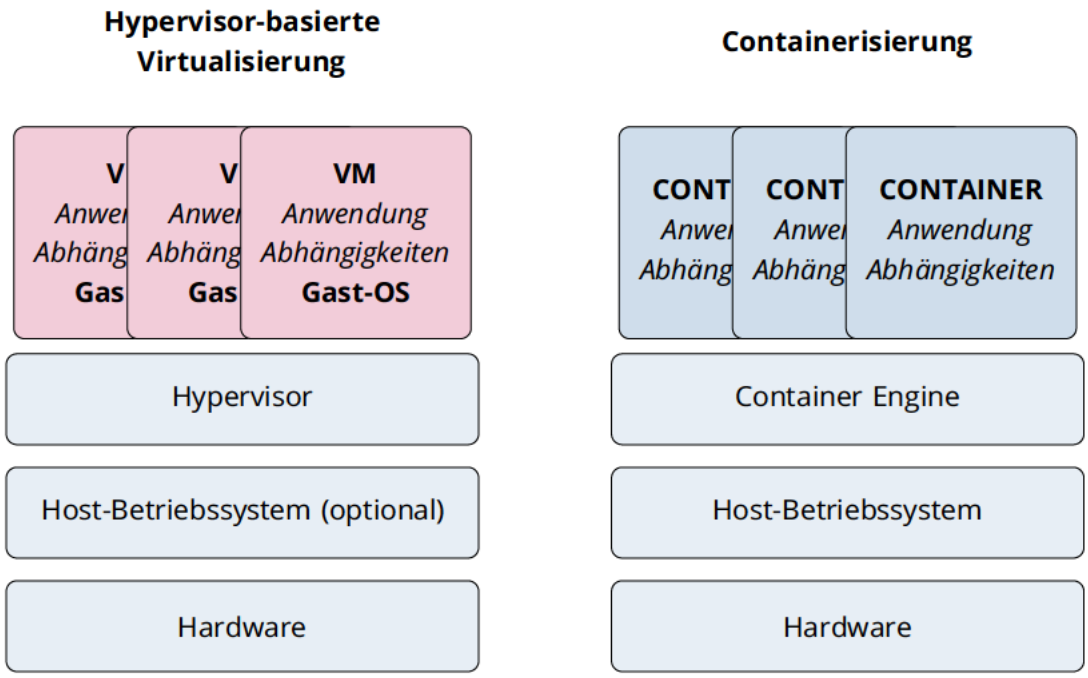

Abbildung 2.3: Virtualisierungstechnologien im Vergleich

Hypervisor-basierte Virtualisierung

Die *Hypervisor-basierte Virtualisierung* ist eine Form der Virtualisierung, bei der ein *Hypervisor* als Software- oder Hardware-Schicht zwischen dem Betriebssystem oder der Hardware der Host-Maschine und den virtuellen Maschinen (*VMs*) fungiert [Pot+20]. Eine VM ist eine

vollständige Emulation einer physischen Maschine und enthält ein voll funktionsfähiges Betriebssystem (*Gast-OS*) [Pot+20]. Der Hypervisor verwaltet die Ressourcenzuweisung und den Zugriff auf die Hardware für die VMs. Die Emulation eines vollständigen Betriebssystems führt im Vergleich zu einer *Containerlösung* zu einem erhöhten Ressourcenverbrauch und einer geringeren Performanz [Pot+20]. VMs eignen sich zur Ausführung komplexer Anwendungen, die eine isolierte Betriebssystemumgebung benötigen [Pot+20]. Für die Integration in Cloud-native Anwendungen sind Containerlösungen aufgrund ihrer Effizienz und Skalierbarkeit jedoch besser geeignet [Pot+20].

Containerisierung

Die *Containerisierung* ist eine leichtgewichtige Form der Virtualisierung, bei der Anwendungen und ihre Abhängigkeiten in isolierten *Containern* ausgeführt werden [GBS17; Pot+20]. Container teilen sich den Kernel des Host-Betriebssystems und erhalten nur die für ihre Ausführung notwendigen Ressourcen [GBS17; Pot+20]. Da Container kein eigenes Betriebssystem emulieren, sind sie effizienter und schneller als VMs [Pot+20]. Container ermöglichen eine schnelle Bereitstellung und Skalierung von Microservices und eignen sich für die Entwicklung von Cloud-nativen Anwendungen [GBS17; Pot+20]. Die Verwaltung und Bereitstellung von Containern erfolgt über eine Container-Engine wie *Docker* [Pot+20].

Docker ist ein Open-Source-Werkzeug (*Apache-Lizenz*) zur automatisierten Bereitstellung von Anwendungen in einer containerisierten Umgebung [Pot+20; Doc24a]. *Docker Engine* ist die Kernkomponente von Docker und basiert auf einer Client-Server-Architektur [Pot+20; Doc24a]. Docker Engine besteht aus einem *Docker Daemon*, einem *Docker Client* und einer API [Pot+20]. Der Docker Daemon (*serverseitige Komponente*) läuft als Hintergrundprozess auf dem Host-Betriebssystem und ist für die Erstellung und Ausführung von Containeranwendungen zuständig [Pot+20]. Der Docker Client (*clientseitige Komponente*) stellt eine Benutzerschnittstelle für die Interaktion mit dem Docker Daemon zur Verfügung und ermöglicht die Steuerung der Containeranwendungen über die Kommandozeile [Pot+20]. Die API (*Docker Engine API*) realisiert die Kommunikation zwischen dem Docker Daemon und dem Docker Client über eine REST-Schnittstelle (RESTful API). Containeranwendungen werden in *Container Images* (*Docker Images*) definiert und in Containern (*Docker Container*) ausgeführt. Docker Container sind isolierte Umgebungen, die eine effiziente und konsistente Ausführung von Anwendungen in unterschiedlichen Umgebungen ermöglichen [Pot+20]. Docker Images sind unveränderliche Schnappschüsse einer Anwendung und ihrer Abhängigkeiten, die als Vorlage für die Erstellung von Containern dienen [Pot+20]. Mit Hilfe einer *Dockerfile*-Konfigurationsdatei können Docker Images automatisiert erstellt und konfiguriert werden [Pot+20]. Docker bietet auch die Möglichkeit, Docker Images in einem zentralen Container-Repository (*Docker Registry*) zu verwalten und bereitzustellen [BHJ16]. Docker hat sich als Standardlösung für die Containerisierung von Anwendungen in Cloud-nativen Umgebungen etabliert [Pot+20; BHJ16]. Die Verwaltung und Orchestrierung von Containern in verteilten Umgebungen erfolgt über spezielle *Container-Orchestrierungswerkzeuge* [Pan+19].

2.1.5 Container-Orchestrierung

Die *Container-Orchestrierung* bezeichnet den Prozess der Verwaltung und Koordination mehrerer Container, um die Skalierbarkeit und Ausfallsicherheit verteilter Softwaresysteme zu gewährleisten [Pou20; Pan+19]. Zu den grundlegenden Aufgaben der Orchestrierung gehören die Bereitstellung, Skalierung und Überwachung von Containern auf einem oder mehreren physischen Rechnern [Pou20]. Das Ziel ist es, die Bereitstellung von Anwendungen zu automatisieren, die Ressourcennutzung zu optimieren und die Ausfallsicherheit des Gesamtsystems zu erhöhen. Zur effizienten und automatisierten Verwaltung der Container werden zentrale *Container-Orchestrierungswerkzeuge* wie *Docker Compose*, *Docker Swarm* oder *Kubernetes* eingesetzt [Pan+19].

Docker Compose

Docker Compose ist ein einfaches Orchestrierungswerkzeug von Docker, das die automatisierte Bereitstellung und Verwaltung von Multi-Containeranwendungen auf einer Maschine ermöglicht [Pan+19]. Mit Docker Compose können mehrere Containeranwendungen in einer Konfigurationsdatei (*YAML*-Skript) definiert und über die Kommandozeile gesteuert werden [Pan+19]. Allerdings bietet Docker Compose keine Möglichkeit, Container über mehrere Maschinen hinweg zu überwachen und zu skalieren [Pan+19]. Für anspruchsvollere Aufgaben werden leistungsfähigere Orchestrierungswerkzeuge wie *Docker Swarm* oder *Kubernetes* benötigt [Pan+19].

Docker Swarm

Docker Classic Swarm – die erste Version von *Docker Swarm* – wird nicht mehr aktiv weiterentwickelt und durch *Docker Swarm mode* ersetzt [Doc24b]. *Docker Swarm mode* ist ein in die Docker Engine integriertes Orchestrierungswerkzeug, das die automatisierte Bereitstellung und Verwaltung von Containeranwendungen in einem *Swarm* ermöglicht [Doc24c]. Ein Swarm besteht aus mehreren *Manager Nodes* und *Worker Nodes*, die jeweils eine Instanz der Docker Engine ausführen und Teil eines *Clusters* (Rechnerverbund) sind [Doc24c]. Manager Nodes sind für die Orchestrierung der Container und die Verwaltung des Clusters zuständig, während Worker Nodes die Containeranwendungen ausführen [Doc24c]. Die Knoten eines Clusters können über mehrere physische Maschinen und über eine Cloud-Umgebung verteilt sein [Doc24c].

Manager Nodes bestehen aus einer API, einem *Orchestrator*, einem *Allocator*, einem *Scheduler* und einem *Dispatcher* [Pan+19]. Die API (*Docker Engine API*) nimmt Befehle über die CLI (*Swarm mode CLI*) oder API-Endpunkt entgegen und erstellt einen *Service* [Doc24b; Pan+19]. Ein Service definiert sogenannte *Tasks*, die auf den Knoten im Swarm ausgeführt werden [Doc24c]. Ein Task enthält die Definition eines Docker Containers und die Befehle, die innerhalb des Containers ausgeführt werden sollen [Doc24c]. Der Orchestrator überprüft den Zustand des Clusters (über einen internen verteilten Zustandsspeicher) und die Service-Definitionen und erstellt Tasks für die Worker Nodes entsprechend dem gewünschten Zustand des Clusters [Pan+19]. Der Allocator weist den Worker Nodes Ressourcen (IP-Adressen etc.) gemäß den

Service-Definitionen zu [Pan+19]. Der Scheduler prüft die Verfügbarkeit der Worker Nodes und weist die Tasks den am besten geeigneten Knoten zu [Pan+19]. Der Dispatcher leitet die Tasks an die ausgewählten Worker Nodes weiter und überwacht deren Ausführung [Pan+19]. Worker Nodes führen die ihnen zugewiesenen Tasks aus und melden ihren Status regelmäßig an die Manager Nodes zurück [Pan+19].

Docker Swarm ermöglicht die einfache Skalierung von Containeranwendungen durch das Hinzufügen und Entfernen von Worker Nodes und die automatische Verteilung der Tasks auf die verfügbaren Ressourcen. In einem Swarm können mehrere Manager Nodes parallel betrieben werden [Pan+19]. Fällt ein führender Manager Node aus, übernimmt ein anderer Knoten automatisch die Führung und verhindert so einen Ausfall des Clusters [Pan+19].

Docker Swarm ist eine einfache und leichtgewichtige Lösung für die Container-Orchestrierung und eignet sich besonders für kleine und mittelgroße Projekte [Pan+19]. Die Funktionalität von Docker Swarm ist durch die API von Docker begrenzt und bietet nur eine eingeschränkte Fehlertoleranz [Pan+19]. Für komplexe verteilte Softwaresysteme mit hohen Anforderungen an Skalierbarkeit und Ausfallsicherheit empfiehlt sich *Kubernetes* [Pan+19; Pou20].

Kubernetes

Kubernetes (*K8s*) ist eine portable, erweiterbare Open-Source-Plattform für die automatisierte Bereitstellung, Verwaltung und Skalierung von Containeranwendungen in einer verteilten Umgebung [Kub23a; Pou20]. K8s wurde von Google entwickelt, steht unter der Apache-Lizenz und wird von der *CNCF* (*Cloud Native Computing Foundation*) verwaltet [Kub23a; Ora24a]. K8s realisiert die Container-Orchestrierung über eine *Control Plane*, die einen Cluster von *Nodes* verwaltet, auf denen die Containeranwendungen ausgeführt werden [Kub24a].

Die Hauptkomponenten der Control Plane sind der API-Server, der *Cluster Store*, der *Controller Manager* und der *Scheduler* (vgl. Abb. 2.4 links) [Kub24a]. Der API-Server (*kube-apiserver*) stellt eine RESTful API für die gesamte Kommunikation innerhalb des Clusters zur Verfügung [Pou20]. Kube-apiserver ist für eine horizontale Skalierung ausgelegt und kann mehrere Instanzen zur Lastverteilung bereitstellen [Kub24a]. Der Cluster Store (*etcd*) ist eine hochverfügbare verteilte Schlüssel-Werte-Datenbank (*Key Value Store*) zur Speicherung aller Konfigurationsdaten und des Clusterzustandes [Kub24a]. Etcd sollte in einer replizierten Konfiguration mit 3 bis 5 Knoten betrieben werden, um eine hohe Verfügbarkeit zu gewährleisten [Pou20]. Der Controller Manager (*kube-controller-manager*) überwacht den Zustand des Clusters und stellt sicher, dass die gewünschte Konfiguration eingehalten wird [Pou20]. Kube-controller-manager führt mehrere *Controller Loops* (Steuerungsschleifen) aus, die kontinuierlich den Zustand des Clusters überwachen und bei Bedarf Anpassungen vornehmen [Pou20]. Eine dieser Steuerungsschleifen ist der *Node Controller*, der die Verfügbarkeit und den Zustand der Nodes überwacht und bei Ausfällen entsprechend reagiert [Kub24a]. Der *Scheduler* (*kube-scheduler*) ist verantwortlich für die Zuweisung von *Pods* an die Nodes im Cluster [Pou20]. Ein Pod ist die kleinste Einheit in K8s und enthält einen oder mehrere Container, die in Gruppen bereitgestellt und verwaltet werden [Pan+19]. Kube-scheduler wählt die Knoten anhand eines komplexen Rankingsystems aus, das Faktoren wie Ressourcenver-

fügbarkeit, Policy-Einstellungen und Benutzerpräferenzen berücksichtigt [Kub24a]. Wird kein geeigneter Knoten gefunden, wartet der Scheduler mit der Zuweisung, bis die benötigten Ressourcen verfügbar sind [Pou20].

Ein *Node* ist eine physische oder virtuelle Maschine im Cluster, auf der die Containeranwendungen ausgeführt werden. Die Hauptkomponenten eines Nodes sind das *kubelet*, die *Container runtime* und der *kube-proxy* (vgl. Abb. 2.4 rechts) [Kub24a]. Das kubelet verbindet den Node mit der Control Plane und registriert seine Ressourcen im Cluster [Pou20]. Das Kubelet fordert offene Arbeitsaufträge vom API-Server an, führt diese aus und meldet den Ausführungsstatus an die Control Plane zurück [Pou20]. Kubelet arbeitet auf Basis von *PodSpecs*, die eine *YAML*- oder *JSON*-Konfiguration (Serialisierung von Daten in Schlüssel-Werte-Paaren) der Containeranwendungen (Pods) enthalten [Kub24b]. Kubelet stellt sicher, dass die Pods gemäß der Spezifikation ausgeführt werden und überwacht ihren Zustand [Kub24b]. Die Container runtime verwaltet die Ausführung und den Lebenszyklus der Container auf dem Knoten [Kub24a]. Sie ist verantwortlich für die Bereitstellung von Images und das Starten und Stoppen von Containern [Pou20]. Das *Container Runtime Interface* (*CRI*) bietet eine standardisierte Schnittstelle für die Integration von Drittanbieter-Container-Runtimes wie *containerd* und *cri-dockerd* [Pou20; Kub24c]. Die *cri-dockerd*-Implementierung ermöglicht die Integration der Docker Engine als Container-Runtime in K8s [Kub24c]. Die *containerd*-Implementierung ist eine Weiterentwicklung der Docker Engine durch die CNCF [Pou20]. Der *Kube-proxy* ist für das lokale Netzwerkmanagement im Cluster verantwortlich [Pou20]. Er sorgt dafür, dass jeder Knoten eine eindeutige IP-Adresse erhält und implementiert lokale Netzwerkregeln [Pou20]. Diese Netzwerkregeln steuern den Datenverkehr zwischen den Pods innerhalb des Clusters und regeln den Zugriff von außen [Kub24a].

Kubectl ist das Kommandozeilenwerkzeug zur Cluster-Verwaltung in K8s und ermöglicht die Interaktion mit der Control Plane [Pou20]. Kubectl nimmt Befehle über die Kommandozeile entgegen, generiert entsprechende API-Anfragen und sendet diese an den API-Server [Pou20]. Eine Konfigurationsdatei enthält Informationen über den Cluster, den Nutzer und die Zugriffsrechte [Pou20]. Das Kommandozeilenwerkzeug ist für Linux, Mac und Windows verfügbar [Pou20].

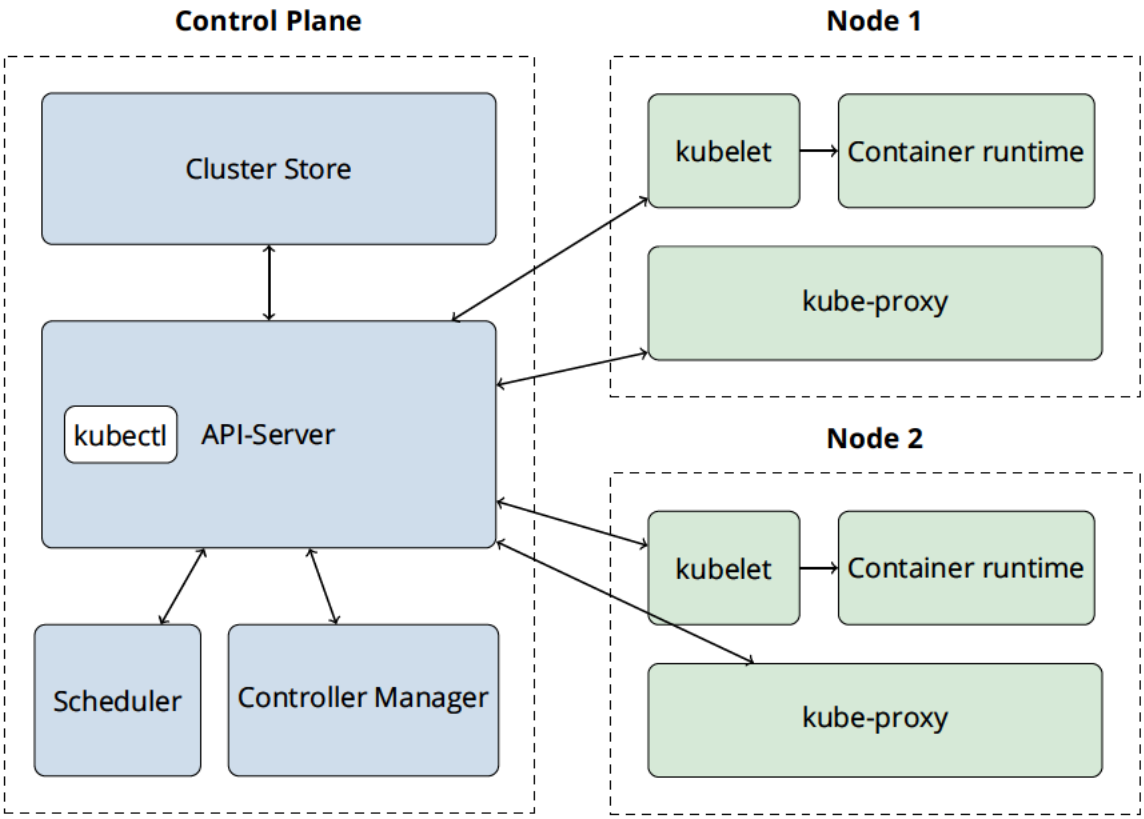

Abbildung 2.4: Kubernetes Cluster

Ein *Service* definiert eine Netzwerkadresse für eine Gruppe von Pods und stellt die Anwendung als Netzwerkdienst bereit [Kub24d]. *Ingress* bietet eine Möglichkeit, HTTP(S)-Routen für den Zugriff auf die Services im Cluster bereitzustellen und Regeln für den Netzwerkverkehr zu konfigurieren [Kub24d]. Auf diese Weise können die Pods über eine virtuelle IP-Adresse erreicht und eine *Lastverteilung* durchgeführt werden [Kub24d].

Um eine hohe Verfügbarkeit und Ausfallsicherheit in Produktionsumgebungen zu gewährleisten, wird die Control Plane in der Regel auf mehrere Knoten repliziert und die Nodes in einem Cluster aus mehreren Rechnern betrieben [Kub24a]. K8s bietet eine hochverfügbare, skalierbare und ausfallsichere Container-Orchestrierungslösung für Cloud-native Anwendungen und wird weltweit von zahlreichen Unternehmen eingesetzt [Pan+19; Vai23]. Der *cloud-controller-manager* ermöglicht den Betrieb von K8s-Clustern in Cloud-Umgebungen wie *Google Cloud Platform* (GCP), *Amazon Web Services* (AWS) und *Microsoft Azure* [Kub24a; Sy +24]. GCP stellt mit der *Google Kubernetes Engine* (GKE) einen verwalteten K8s-Dienst zur Verfügung, der die einfache Bereitstellung und Verwaltung von K8s-Clustern in der Google Cloud ermöglicht [Goo24a].

Lastverteilung

Ein *Load Balancer* verteilt den eingehenden Netzwerkverkehr gleichmäßig auf mehrere Serverknoten, um eine optimale Ressourcennutzung und hohe Verfügbarkeit in verteilten Systemen zu gewährleisten. In K8s werden Services und Ingress-Ressourcen verwendet, um den eingehenden Netzwerkverkehr auf die Pods im Cluster zu verteilen [Kub24d]. Ingress selbst bietet

keine Lastverteilung, sondern definiert Regeln, die von einem *Ingress Controller* umgesetzt werden. Der Ingress Controller ist für die Erfüllung des Ingress verantwortlich und konfiguriert einen Load Balancer entsprechend der Ingress-Regeln [Kub24d]. Der Ingress Controller ist nicht Bestandteil des kube-controller-managers und muss separat bereitgestellt werden [Kub24d].

NGINX Ingress Controller ist eine auf *NGINX* basierende Implementierung eines Ingress Controllers zur Lastverteilung in K8s-Clustern auf OSI-Schicht 7 (Anwendungsschicht) und OSI-Schicht 4 (Transportschicht) [F5 24]. NGINX unterstützt die Protokolle HTTP(S), TCP und UDP und kann daher für eine Vielzahl von Diensten wie Webanwendungen, Drittanbieter-APIs und Datenbanken (z.B. MySQL) eingesetzt werden [Ami22]. Der NGINX Ingress Controller wird als Pod innerhalb des Clusters bereitgestellt und fungiert als Reverse Proxy für den eingehenden Netzwerkverkehr [Ami22]. Der Ingress Controller empfängt Anfragen von externen Clients und leitet diese an die entsprechenden Services im Cluster weiter [Ami22]. Die NGINX-Software setzt die Ingress-Regeln um und bietet zusätzliche Funktionen, die über die herkömmlichen Möglichkeiten des Ingress-Ressourcentyps hinausgehen [F5 24]. Der NGINX Ingress Controller unterstützt eine Ende-zu-Ende-Verschlüsselung zwischen dem Ingress Controller und den Pods und bietet verschiedene Betriebsmodi wie *TLS Passthrough*, *TLS Termination*, *Traffic Control* und *Traffic Splitting* [Ami22]. TLS Passthrough ermöglicht die Lastverteilung von TLS-verschlüsselten Datenverbindungen, ohne dass der Ingress Controller die Daten entschlüsseln muss [Ami22]. TLS Termination entschlüsselt den Datenverkehr auf dem Ingress Controller und leitet ihn unverschlüsselt an die Pods weiter [Ami22]. Traffic Control beinhaltet Methoden wie Rate Limiting, um einzelne Services vor Überlastung zu schützen [Ami22]. Traffic Splitting ermöglicht es, neue Versionen einer Anwendung ohne Ausfallzeiten einzuführen und den Datenverkehr zwischen einer Produktions- und einer Testumgebung aufzuteilen (*Blue/Green-Deployment*) [Ami22].

Wird der K8s-Cluster in der Cloud betrieben, erfolgt die Lastverteilung des eingehenden Netzwerkverkehrs über Cloud-interne Load Balancer wie *GKE Ingress* (GKE) [Goo24b]. Innerhalb eines GKE-Clusters ist der *GKE Ingress Controller* die bevorzugte Methode zur Lastverteilung des HTTP(S)-Verkehrs [Goo24b]. Der GKE Ingress Controller erstellt einen *Google Cloud-HTTP(S)-Load-Balancer* und konfiguriert diesen entsprechend den Informationen in den Ingress- und Service-Objekten [Goo24b].

Monitoring

Das *Monitoring* ist ein wesentlicher Bestandteil in der Verwaltung von Cloud-nativen Anwendungen. Das Monitoring umfasst die Erfassung und Analyse von Metriken, die einen Einblick in den Zustand und die Leistung des Systems geben [MUS22]. Es ermöglicht die Überwachung von Ressourcenverbrauch, Anwendungsleistung und Sicherheitsereignissen in Echtzeit [MUS22]. Durch die kontinuierliche Überwachung dieser Metriken können potenzielle Probleme frühzeitig erkannt und behandelt werden [MUS22]. Das Monitoring ist somit entscheidend für die Aufrechterhaltung der Systemstabilität und die Einhaltung vereinbarter Leistungsziele (*Service Level Agreements*) [MUS22].

In K8s wird das Monitoring durch spezialisierte Werkzeuge zur Erfassung, Speicherung und Visualisierung von Metriken (*Metrics*) realisiert [MUS22]. Zu den gängigen Monitoringwerkzeugen in K8s gehören *Prometheus* und *Grafana*, die in der Regel in Kombination eingesetzt werden. *Prometheus* ist das zentrale Werkzeug zur Erfassung von Metriken in K8s [MUS22]. Prometheus ist ein Open-Source-Werkzeug zur Überwachung und Alarmierung in verteilten Systemen [MUS22]. Durch die nahtlose Integration in K8s kann Prometheus Dienste und Endpunkte automatisch erkennen und überwachen [MUS22]. Es sammelt Metriken aus dem K8s-Cluster, speichert diese in Form von Zeitreihen und ermöglicht detaillierte Abfragen über seine flexible Abfragesprache *PromQL* [MUS22]. Aufgrund seiner Robustheit und Effizienz eignet sich Prometheus besonders für die Überwachung großer und komplexer Softwaresysteme [MUS22]. *Grafana* ist ein leistungsstarkes Visualisierungswerkzeug, das eine interaktive und anpassbare Darstellung von Metriken ermöglicht [MUS22]. Es integriert sich nahtlos in Prometheus und unterstützt die Kombination von Metriken aus verschiedenen Datenquellen [MUS22]. Grafana bietet auch umfangreiche Alarmfunktionen, die visuelle und proaktive Ansätze für das Monitoring ermöglichen [MUS22]. Mit Grafana können interaktive Dashboards erstellt werden, die die wichtigsten Leistungsindikatoren und Trends im System visualisieren [MUS22].

2.1.6 Datenbankarchitektur

Für die dauerhafte Speicherung von Daten in einem Softwaresystem stehen verschiedene Datenbankmodelle und -technologien zur Verfügung. Die Datenbankmodelle lassen sich in zwei Hauptkategorien unterteilen: Relationale Datenbanken und nicht-relationale Datenbanken. Die Erstellung und Verwaltung von Datenbanken erfolgt durch Datenbankmanagementsysteme. Die Transformation von Anwendungsdaten in relationale Datenbanken erfolgt durch sogenannte *ORM-Werkzeuge*.

Relationale Datenbanken

Relationale Datenbanken basieren auf einem relationalen Datenmodell, das Daten in Tabellen (Relationen) organisiert und Beziehungen zwischen den Tabellen herstellt [Arn23]. Diese Struktur ermöglicht eine effiziente und konsistente Speicherung und Verwaltung von Daten [Arn23]. *SQL* (*Structured Query Language*) ist die Standardsprache für die Abfrage und Manipulation von Daten in relationalen Datenbanken [Arn23]. Relationale Datenbanken bieten durch die Einhaltung der *ACID*-Eigenschaften (*Atomicity*, *Consistency*, *Isolation*, *Durability*) ein hohes Maß an Datenkonsistenz und Transaktionsintegrität [Arn23]. Eine Transaktion ist eine Gruppe von Datenbankoperationen, die als eine Einheit entweder vollständig oder gar nicht ausgeführt werden [Arn23]. Die Atomarität (Atomicity) gewährleistet, dass eine Transaktion unteilbar ist und entweder vollständig oder gar nicht ausgeführt wird. Die Konsistenz (Consistency) stellt sicher, dass sich die Datenbank nach Abschluss einer Transaktion in einem konsistenten Zustand befindet. Die Isolation (Isolation) verhindert, dass sich Transaktionen, bei gleichzeitigem Zugriff auf die Datenbank, gegenseitig beeinflussen. Die Dauerhaftigkeit (Durability) stellt sicher, dass die Daten nach Abschluss einer Transaktion dauerhaft gespeichert bleiben. Relationale Datenbanken sind ideal für transaktionale Anwendungen, die eine hohe Datenkonsistenz und -integrität bei der Ausführung von Transaktionen erfordern.

Nicht-relationale Datenbanken

Nicht-relationale Datenbanken verzichten auf das relationale Datenmodell und damit auf die strengen Konsistenzgarantien, die relationale Datenbanken bieten [Arn23]. Nicht-relationale Datenbanken sind auf hohe Verfügbarkeit und Skalierbarkeit ausgelegt und bieten ein vereinfachtes Datenmodell [Arn23]. Sie folgen den *BASE*-Eigenschaften (*Basically Available*, *Soft State*, *Eventually Consistent*), die eine hohe Verfügbarkeit und Skalierbarkeit ermöglichen [Arn23]. Das System garantiert die Verfügbarkeit der Daten (Basically Available), erlaubt einen inkonsistenten Zwischenzustand der Daten (Soft State) und strebt eine spätere Konsistenz der Daten an (Eventually Consistent). Nicht-relationale Datenbanken werden in der Regel als Key-Value-Stores, dokumentenorientierte Datenbanken oder spaltenorientierte Datenbanken implementiert [Arn23]. Nicht-relationale Datenbanken eignen sich für verteilte Softwaresysteme, die ein hohes Maß an Skalierbarkeit und Verfügbarkeit erfordern, bei denen jedoch weniger Wert auf die Konsistenz der Daten gelegt wird.

Datenbankmanagementsysteme

Ein Datenbankmanagementsystem (*DBMS*) ist eine Software zur Erstellung und Verwaltung von relationalen oder nicht-relationalen Datenbanken. Ein DBMS bietet Funktionen zum Speichern, Abrufen, Aktualisieren und Löschen von Daten sowie zum Sichern und Wiederherstellen von Datenbanken. Für eine Datenbankinfrastruktur in einer Cloud-nativen Umgebung werden spezielle verteilte Datenbankmanagementsysteme eingesetzt, die die Anforderungen an Skalierbarkeit, Verfügbarkeit und Ausfallsicherheit in verteilten Umgebungen erfüllen [Arn23].

MySQL ist eines der bekanntesten Open Source DBMS zur Verwaltung relationaler Datenbanken [Arn23]. Für den Einsatz als verteiltes Datenbankmanagementsystem in Cloud-nativen Anwendungen unterstützt MySQL Replikationsmechanismen, Sharding-Techniken und Clustering-Lösungen. Diese Mechanismen werden in der Regel in Kombination eingesetzt, um die Anforderungen von Cloud-nativen Anwendungen zu erfüllen [Arn23].

Object-Relational Mapping

Object-Relational Mapping (ORM) ist eine Technik zur Transformation von Daten zwischen objektorientierten Programmiersprachen und relationalen Datenbanken. ORM-Werkzeuge ermöglichen die Abbildung von Objekten (Klassen) auf Tabellen (Relationen) und umgekehrt. Das ORM vereinfacht die Datenbankinteraktion und gewährleistet die Konsistenz zwischen dem objektorientierten Modell und der relationalen Datenbank.

2.1.7 Verteilte Datenbanken

Eine verteilte Datenbank ist eine Datenbank, bei der die Daten oder Teile der Daten auf mehrere physische Rechner verteilt sind. Die zentrale Verwaltungssoftware wird als verteiltes Datenbankmanagementsystem (*DDBMS*) bezeichnet und ermöglicht die Erstellung und

Verwaltung verteilter Datenbanken. Ein DDBMS bietet Funktionen für die Replikation (Replication), die Aufteilung (Sharding) und die Clusterbildung (Clustering) von Daten in verteilten Umgebungen. Für MySQL stehen hochverfügbare und skalierbare Clustering-Lösungen wie *InnoDB Cluster* und *NDB Cluster* zur Verfügung [Arn23].

Replication

Replication ist ein Verfahren zur Replikation von Daten von einem Datenbankserver (*Primärserver*) auf einen oder mehrere Datenbankserver (*Sekundärserver* oder *Replikate*) [Arn23; Ora24b]. Je nach Konfiguration können alle Datenbanken, ausgewählte Datenbanken oder ausgewählte Datenbanktabellen repliziert werden [Ora24b]. Die Replikation erhöht die Verfügbarkeit und Ausfallsicherheit der Datenbank, da die Daten redundant auf mehreren Servern gespeichert werden [Arn23]. Durch die Verteilung der Daten auf mehrere Server können Lese- und Schreibzugriffe auf die Datenbank parallel ausgeführt und die Last bei Lesezugriffen auf die einzelnen Server verteilt werden [Arn23].

Es gibt verschiedene Konfigurationsmöglichkeiten, wie die *asynchrone*, *semi-synchrone* und *synchrone Replikation* [Arn23]. In MySQL erfolgt die Replikation standardmäßig asynchron [Ora24b]. Bei einer asynchronen Replikation werden Transaktionen auf dem Primärserver ausgeführt und später an die sekundären Server weitergeleitet [Arn23]. Die Sekundärserver erhalten die Änderungen zeitversetzt und können daher eine gewisse Latenz aufweisen. Da die Clients nicht auf den Abschluss der Replikation warten müssen, kann die asynchrone Replikation eine hohe Flexibilität und Performanz bieten [Arn23]. Die asynchrone Replikation kann jedoch zu Inkonsistenzen führen, wenn der Primärserver ausfällt, bevor alle Änderungen repliziert wurden [Arn23]. Eine synchrone Replikation stellt sicher, dass alle Änderungen sofort auf die Sekundärserver repliziert werden, bevor eine Transaktion als abgeschlossen gilt [Arn23]. Dies gewährleistet, dass die Daten auf allen Replikaten vorhanden und konsistent sind, verringert jedoch die Performanz, da die Clients auf den Abschluss der Replikation warten müssen [Arn23]. Die semi-synchrone Replikation ist ein Kompromiss zwischen synchroner und asynchroner Replikation. Die Transaktion gilt als abgeschlossen, wenn mindestens ein sekundärer Server die Änderungen bestätigt hat oder die Änderungen durch eine Gruppe (*Quorum*) von sekundären Servern bestätigt wurden [Arn23; Ora24b].

Sharding

Sharding ist eine Technik zur horizontalen Skalierung, bei der große Datenbanken in kleinere, handhabbare Teile (*Shards*) aufgeteilt werden [Arn23]. Dabei wird nicht die gesamte Datenbank repliziert, sondern ein Teil der Daten auf mehrere Datenbanken oder Tabellen verteilt. Jeder Shard kann auf einem separaten Server oder einer Gruppe von Servern gespeichert werden [Arn23]. Dadurch wird die Last auf die einzelnen Server verteilt und die Performanz und Skalierbarkeit der Datenbank erhöht [Arn23]. Durch die Reduzierung der Last auf den einzelnen Servern können die vorhandenen Ressourcen effizienter genutzt werden. Allerdings erhöht Sharding auch die Komplexität der Datenbankarchitektur, da die Daten über mehrere Server hinweg konsistent gehalten werden müssen [Arn23]. Sharding ist ideal für Anwendungen, die mit großen Datenmengen arbeiten und hohe Anforderungen an die Skalierbarkeit stellen [Arn23].

MySQL bietet keine integrierte Sharding-Funktionalität. Es stehen aber verschiedene Werkzeuge wie *Vitess* oder *ShardingSphere* als Open-Source-Middleware zur Verfügung [Arn23].

Clustering

Clustering ist eine Technik zur Verteilung von Datenbanken auf mehrere Server, um die Ausfallsicherheit und Skalierbarkeit zu erhöhen [DF06]. Beim Clustering werden mehrere Server zu einem *Cluster* zusammengeschlossen, die als ein einheitliches System agieren und gemeinsam eine Datenbank betreiben [DF06]. Durch die Verteilung der Datenbank auf mehrere Server wird die Last auf die einzelnen Server verteilt und eine Redundanz geschaffen, um Ausfälle einzelner Server zu kompensieren [DF06]. Es gibt verschiedene Arten von Clustering-Techniken, darunter *Shared-Disk Clustering* und *Shared-Nothing-Clustering* [DF06]. Beim Shared-Disk Clustering greifen mehrere Server auf dasselbe Speichermedium zu und verwenden einen *Lock Manager*, der den Zugriff auf die Daten koordiniert [DF06]. Beim Shared-Nothing-Clustering verwaltet jeder Server seine eigenen Ressourcen und repliziert die Daten auf andere Server, um die Ausfallsicherheit zu gewährleisten [DF06]. MySQL bietet integrierte Shared-Nothing-Clustering-Lösungen, die eine hohe Verfügbarkeit und Skalierbarkeit in verteilten Umgebungen ermöglichen – darunter *InnoDB Cluster* und *NDB Cluster* [Arn23; Ora24c].

MySQL Group Replication ermöglicht die Verteilung von Datenbankoperationen auf mehrere Server, die als eine einheitliche Gruppe agieren [Ora24d]. Dabei koordinieren sich die Server automatisch, um als Gruppe zu arbeiten und die Daten konsistent zu halten [Ora24d]. Die Server können im *Single-Primary-Modus* (automatische Auswahl eines primären Servers, der Änderungen verarbeitet) oder im *Multi-Primary-Modus* (mehrere Server verarbeiten Änderungen parallel) betrieben werden [Ora24d].

InnoDB Cluster

InnoDB Cluster ist eine semi-synchrone Shared-Nothing-Clustering-Lösung für MySQL, die auf der *InnoDB Storage Engine* basiert [Arn23; Ora24e; Ora24c]. InnoDB Cluster verwendet MySQL Group Replication für die automatische Replikation und Verwaltung der Clusterknoten [Ora24e]. Änderungen an den Datenbanken werden automatisch auf alle Knoten repliziert, um eine hohe Konsistenz und Verfügbarkeit zu gewährleisten [Arn23]. Bei Ausfall eines Primärservers wird automatisch ein neuer Primärserver ausgewählt [Arn23]. InnoDB Cluster lässt sich nahtlos in bestehende MySQL-Umgebungen integrieren und bietet eine hohe Verfügbarkeit in verteilten Umgebungen [Arn23]. Die Integration in K8s wird offiziell unterstützt und ermöglicht die Bereitstellung eines hochverfügbaren MySQL-Clusters in einer containerisierten Umgebung [Arn23].

NDB Cluster

NDB Cluster ist eine synchrone Shared-Nothing-Clustering-Lösung für MySQL, die auf der *NDB Storage Engine* basiert [Arn23; Ora24f; Ora24c]. Die NDB Storage Engine ist eine hochverfügbare, verteilte Speicherlösung, die auf der Shared-Nothing-Architektur basiert [Ora24f]. Die Daten werden primär im Arbeitsspeicher gehalten und durch eine persistente Speicherung

auf Festplatten ergänzt [Arn23]. Dies ermöglicht eine schnelle Verarbeitung von Lese- und Schreibzugriffen auf die Datenbank [Arn23]. NDB Cluster bietet eine Lösung für Anwendungen, die eine hohe Verfügbarkeit, Skalierbarkeit und Ausfallsicherheit sowie eine geringe Latenz erfordern [Arn23]. Die Integration in K8s wird ebenfalls offiziell unterstützt und ermöglicht die Bereitstellung eines hochverfügbaren und hochperformanten MySQL-Clusters in einer containerisierten Umgebung [Arn23].

2.2 Blockchain-Technologie

Da das zu entwickelnde Softwaresystem ein Bitcoin-Handelssystem darstellt, ist ein Verständnis der zugrundeliegenden Blockchain-Technologie von zentraler Bedeutung. Zunächst wird die Blockchain-Technologie definiert und die technischen Grundlagen der Blockchain erläutert. Anschließend wird die praktische Anwendung der Blockchain-Technologie anhand der Kryptowährung Bitcoin dargestellt und ein Überblick über die regulatorischen Aspekte im Bereich der Kryptowährungen gegeben. Die Einbeziehung technologischer und regulatorischer Aspekte ermöglicht eine ganzheitliche Betrachtung der für das Handelssystem relevanten Rahmenbedingungen.

2.2.1 Definition & Konzept

Die Blockchain-Technologie ermöglicht eine sichere, transparente und dezentrale Datenverarbeitung und bildet die technologische Grundlage für Kryptowährungen wie Bitcoin. Die Blockchain ist ein dezentrales, verteiltes Hauptbuch, das Transaktionen in Form von Blöcken – einer kontinuierlich wachsenden Kette von Datensätzen – organisiert [Nak08a]. Jeder Block enthält Transaktionsdaten sowie einen kryptographischen Hash des vorhergehenden Blocks. Die Verkettung der Blöcke stellt eine unveränderliche chronologische Aufzeichnung aller Transaktionen im Netz dar und gewährleistet die Integrität und Transparenz der Daten.

Die Dezentralität der Blockchain wird durch Mechanismen zur Konsensfindung sichergestellt. Die Teilnehmer im Netzwerk einigen sich auf einen gemeinsamen Zustand der Blockchain, ohne dass eine zentrale Instanz notwendig ist. Dies reduziert das Risiko von Manipulationen und Ausfällen und schafft eine vertrauenswürdige Umgebung für die Durchführung von Transaktionen [TS16].

Blöcke & Transaktionen

Blöcke sind die elementaren Bestandteile der Blockchain, in denen die Transaktionsdaten aus einer Reihe von Transaktionen zusammengefasst werden. Ein Block besteht aus einem *Block Header* und einer Liste von Transaktionen. Der Block Header enthält wichtige Metadaten wie einen Zeitstempel, eine *Nonce*, einen *Merkle-Root-Hash* und den Hash des vorhergehenden Blocks. Die Nonce ist ein Zufallswert, der bei der Berechnung des Blockhashes für die Konsensfindung verwendet wird. Der Merkle-Root-Hash ist ein kryptographischer Hash, der die Transaktionen in einem Block zusammenfasst und eine effiziente Verifizierung der Transaktionen im Block ermöglicht [Mer88]. Der Hash des vorhergehenden Blocks wird zur Verkettung der Blöcke verwendet. Die so entstehende Kette von Blöcken wird als Blockchain bezeichnet.

Transaktionen repräsentieren den Austausch von Informationen oder Vermögenswerten zwischen den Teilnehmern im Netzwerk. Jede Transaktion verweist auf eine oder mehrere vorangegangene Transaktionen, so dass ein lückenloser Verlauf aller Transaktionen im Netzwerk – vom Ursprung des Wertes bis zum aktuellen Zustand – möglich ist. Diese Struktur gewährleistet die Integrität und Transparenz der Transaktionen und verhindert doppelte Ausgaben und Manipulationen [Bon+15].

Hashing & Blockverkettung

Ein kryptographischer Hash ist eine mathematische Funktion, die eine Eingabe variabler Größe in eine Ausgabe fester Länge umwandelt. Wichtig für die Blockchain-Technologie ist die Eigenschaft der Einwegfunktion und der Kollisionsresistenz. Die Berechnung des Hashwertes einer Eingabe ist einfach und effizient, während die Umkehrung des Prozesses nicht mit vertretbarem Aufwand möglich ist (Einwegfunktion). Die Kollisionsresistenz stellt sicher, dass zwei unterschiedliche Eingaben nicht denselben Hashwert erzeugen [Sta17].

Die Integrität der Blockchain basiert auf der kryptographischen Unveränderlichkeit der Hashwerte, die alle Blöcke über die Hashes im Block Header miteinander verknüpfen. Durch die Verkettung der Blöcke entsteht eine unveränderliche, chronologische Aufzeichnung aller Transaktionen im Netzwerk. Eine nachträgliche Manipulation von Transaktionen in einem Block würde den Hashwert dieses Blocks und damit auch den Hashwert aller nachfolgenden Blöcke verändern [TS16].

Konsensfindung

Die dezentrale Struktur der Blockchain erfordert einen effizienten Mechanismus zur Konsensfindung, der sicherstellt, dass alle teilnehmenden Knoten im Netzwerk eine einheitliche Sicht auf den Zustand der Blockchain haben. Im Bitcoin-Netzwerk wird der Konsens durch den *Proof-of-Work*-Algorithmus (*PoW*) realisiert [Nak08a]. Dieser Mechanismus basiert auf dem Wettbewerb zwischen den sogenannten *Minern*, die komplexe kryptographische Rechenaufgaben lösen, um das Recht zu erhalten, einen neuen Block in die Blockchain aufzunehmen. Die Lösung dieser Aufgaben erfordert erhebliche Rechenressourcen und den Einsatz von Energie und dient als Nachweis für die erbrachte Rechenleistung. Als Anreiz für die Bereitstellung dieser Rechenleistung erhalten die Miner neu generierte Bitcoins sowie Transaktionsgebühren für die im Block enthaltenen Transaktionen [Nak08a; DN92].

Wallet

Eine Wallet ist eine digitale Geldbörse, die eine sichere Speicherung und Verwaltung der für die Transaktionen notwendigen kryptographischen Schlüssel ermöglicht. Die kryptographischen Schlüssel selbst repräsentieren nicht den eigentlichen Wert (z.B. Bitcoins), sondern dienen als Zugriffsberechtigung auf die in der Blockchain referenzierten Vermögenswerte.

Die Verwahrung von Kryptowährungen in einer Wallet lässt sich hinsichtlich der Nutzerkontrolle und der Sicherheit in zwei Kategorien einteilen: Zentrale Verwahrung durch einen Zahlungsdienstleister (*Custodian*) und Selbstverwahrung (*Self Custody*).

Bei der Verwahrung durch einen Zahlungsdienstleister übernimmt dieser die Verwaltung der privaten Schlüssel im Auftrag des Nutzers [Bon+15]. Der Nutzer erhält über die Benutzeroberfläche oder die API des Anbieters Zugriff auf seine Vermögenswerte (sogenannte *Custodial Wallet*) [Bit22a]. Diese Lösung bietet in der Regel eine höhere Benutzerfreundlichkeit und überträgt die Verantwortung für die Sicherheit der privaten Schlüssel an den Anbieter [DSG19]. Der Zugriff auf die in der Wallet gespeicherten Vermögenswerte ist somit nur über den jeweiligen Anbieter möglich.

Bei der Selbstverwahrung behält der Nutzer die volle Kontrolle über seine privaten Schlüssel und ist allein für deren sichere Aufbewahrung und Verwaltung verantwortlich. Ein Verlust der privaten Schlüssel führt in diesem Fall unwiderruflich zum Verlust des Zugriffs auf die gespeicherten Vermögenswerte.

2.2.2 Bitcoin

Bitcoin ist die erste und gemessen an der Marktkapitalisierung bisher erfolgreichste Anwendung der Blockchain-Technologie [Coi24a]. Bitcoin ist eine dezentrale Kryptowährung, die den direkten Austausch von Vermögenswerten zwischen zwei Parteien ermöglicht, ohne dass eine zentrale Instanz als Vermittler benötigt wird. Die Kryptowährung wurde im Jahr 2008 unter dem Pseudonym *Satoshi Nakamoto* im Whitepaper „Bitcoin: A Peer-to-Peer Electronic Cash System“ [Nak08a] vorgestellt. Bitcoin basiert auf dem PoW-Konsensmechanismus [Nak08a].

Die maximal mögliche Anzahl an Bitcoins ist auf 21 Millionen Einheiten begrenzt. Diese festgelegte Obergrenze verleiht der Kryptowährung einen deflationären Charakter. Diese bewusste Verknappung ist ein integraler Bestandteil der langfristigen Wertstabilität von Bitcoin [Bon+15].

2.2.3 Regulatorisches Umfeld

Der Betrieb eines Bitcoin-Handelssystems unterliegt spezifischen gesetzlichen Anforderungen, die darauf abzielen, die Integrität des Finanzsystems zu wahren, Geldwäsche und Terrorismusfinanzierung zu verhindern und den Verbraucherschutz zu gewährleisten.

Anti-Geldwäsche-Richtlinien

Die Anti-Geldwäsche-Richtlinien (*AML-Richtlinien*) sind gesetzliche Regelungen zur Bekämpfung von Geldwäsche und Terrorismusfinanzierung. Unternehmen – insbesondere Finanzdienstleister im Bereich Kryptowährungen – sind daher verpflichtet, die AML-Richtlinien einzuhalten. Ein wichtiger Bestandteil ist das *KYC*-Verfahren (*Know Your Customer*), bei dem die Identität der Kunden gemäß den gesetzlichen Vorgaben überprüft wird [BaF18]. In Deutschland gilt die 5. EU-Geldwäscherichtlinie [Eur18].

MiCA-Verordnung

Die *Markets in Crypto-Assets Regulation* (MiCA) ist eine EU-Verordnung, die einheitliche Regeln für den Umgang mit Kryptowährungen innerhalb der EU schaffen soll [Eur23]. Das Ziel ist es, die Marktintegrität, die Finanzstabilität und den Schutz der Anleger zu gewährleisten. Finanzdienstleister im Bereich Kryptowährungen müssen die Bestimmungen der MiCA-Verordnung einhalten, um ihre Dienstleistungen in der EU anbieten zu können.

BaFin

Die *Bundesanstalt für Finanzdienstleistungsaufsicht* (BaFin) ist die zuständige Aufsichtsbehörde für Finanzdienstleister in Deutschland. Mit der zunehmenden Nutzung der Blockchain-Technologie hat die BaFin ihre Aufsichtstätigkeit auf den Bereich der Kryptowährungen ausgeweitet. Die BaFin ist für die Lizenzierung von Kryptoverwahrern zuständig und erteilt hierfür eine *Kryptoverwahrlizenz*.

> Wer in Deutschland gewerbsmäßig oder in einem kaufmännischen Umfang das Kryptoverwahrgeschäft erbringen will, benötigt noch vor der Aufnahme der Geschäftstätigkeit nach § 32 Abs. 1 Satz 1 KWG eine schriftliche Erlaubnis der BaFin. [BaF22]

Die BaFin prüft neue Geschäftsmodelle im Bereich Kryptowährungen und ist für die Marktaufsicht, den Verbraucherschutz und die Einhaltung der AML-Richtlinien zuständig.

3 Anforderungsanalyse

Die grundlegende Architektur des Softwaresystems wird in einem strukturierten Entwicklungsprozess nach dem Wasserfallmodell entwickelt. Innerhalb der Anforderungsanalyse werden die funktionalen und nicht-funktionalen Anforderungen an das Softwaresystem definiert. Die funktionalen Anforderungen werden in Anwendungsfällen (Use Cases) beschrieben. Die logische Umsetzung der funktionalen Anforderungen wird in einem Klassendiagramm im Analysemodell dargestellt. Die nicht-funktionalen Anforderungen werden anhand von Qualitätsmerkmalen zur Bewertung von Softwaresystemen definiert und an die projektspezifischen Anforderungen angepasst. Die Anforderungsanalyse bildet die Grundlage für einen systematisch strukturierten Aufbau des zugrundeliegenden Softwaresystems.

3.1 Systembeschreibung

Die Plattform – „Cash2Coin“ – besteht aus einer Kundenanwendung, einem Kassensystem und einem Zahlungsdienstleister. Innerhalb der Kundenanwendung können aktuelle Marktinformationen abgerufen, Kauf- und Verkaufsaufträge erstellt sowie Bitcoins gesendet und empfangen werden. Partnerfilialen sind mit einem Kassensystem ausgestattet, das den Kassierer durch den Kauf- und Verkaufsprozess leitet. Der Zahlungsdienstleister realisiert den Bitcoin-Handel und verwahrt die Bitcoins für den Nutzer.

Der Nutzer installiert die Kundenanwendung als mobile App auf seinem Smartphone, registriert sich und verifiziert seine Identität über ein KYC-Verfahren. In einer Kartenansicht werden Partnerfilialen angezeigt, in denen der Nutzer Bitcoins gegen Euro kaufen oder verkaufen kann. Der Nutzer erstellt einen Kaufauftrag und tätigt die Zahlung innerhalb einer Partnerfiliale. Der Kassierer bestätigt die Zahlung über das Kassensystem (Kassenanwendung) und löst den Kaufauftrag aus. Der Zahlungsdienstleister führt den Bitcoin-Kauf durch und stellt dem Nutzer die Bitcoins in einer Custodial Wallet zur Verfügung. Der Verkauf erfolgt analog zum Kaufprozess, unterscheidet sich jedoch in der Auszahlung. Diese erfolgt durch den Zahlungsdienstleister direkt auf das Bankkonto des Nutzers oder nach dem Gutscheinprinzip. In ausgewählten Partnerfilialen erhält der Nutzer gegen Vorlage eines Gutscheincodes den Verkaufserlös in bar. Ein Monitoringsystem überwacht alle Transaktionen und informiert den Support bei Auffälligkeiten. Der Support und Administratoren haben erweiterten Zugriff auf das System und können Nutzerdaten einsehen und bearbeiten.

Folgende Akteure interagieren mit dem System:

- Nutzer (Kunde)
- Partner
- Kassierer
- Smartphone (Kundenanwendung)
- Smartphone (Kassenanwendung)
- KYC-Dienstleister
- Zahlungsdienstleister

- Support
- Administrator

3.2 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben das erwartete Verhalten und die Kernfunktionen der Software. Die Anforderungen werden in Anwendungsfällen (Use Cases) beschrieben und in Use-Case-Diagrammen visualisiert.

3.2.1 Kundenanwendung

Die Kundenanwendung ermöglicht es dem Nutzer, Partnerfilialen in seiner Nähe zu finden, Kauf- und Verkaufsaufträge zu erstellen, Bitcoins zu senden und zu empfangen sowie Kauf- und Verkaufsaktivitäten zu überwachen.

Benutzerregistrierung

Der Nutzer installiert die Kundenanwendung auf seinem Smartphone, öffnet die App und wählt die Option zur Registrierung. Der Nutzer gibt seine E-Mail-Adresse und ein sicheres Passwort ein, akzeptiert die Nutzungsbedingungen und bestätigt die Registrierung. Mit der Bestätigung der Registrierung wird ein neues Benutzerkonto in der Datenbank angelegt und eine Bestätigungs-E-Mail mit einem Verifizierungslink an den Nutzer gesendet. Anschließend wird der Nutzer aufgefordert, seine persönlichen Daten einzugeben und seine Identität zu verifizieren. Dazu lädt der Nutzer ein Identifikationsdokument hoch und bestätigt die Verifizierung durch den KYC-Dienstleister. Die App informiert den Nutzer über das Ergebnis der Verifizierung und die Freischaltung des Benutzerkontos. Das Use-Case-Diagramm in Abbildung 3.1 zeigt die Interaktion mit dem Softwaresystem bei der Benutzerregistrierung.

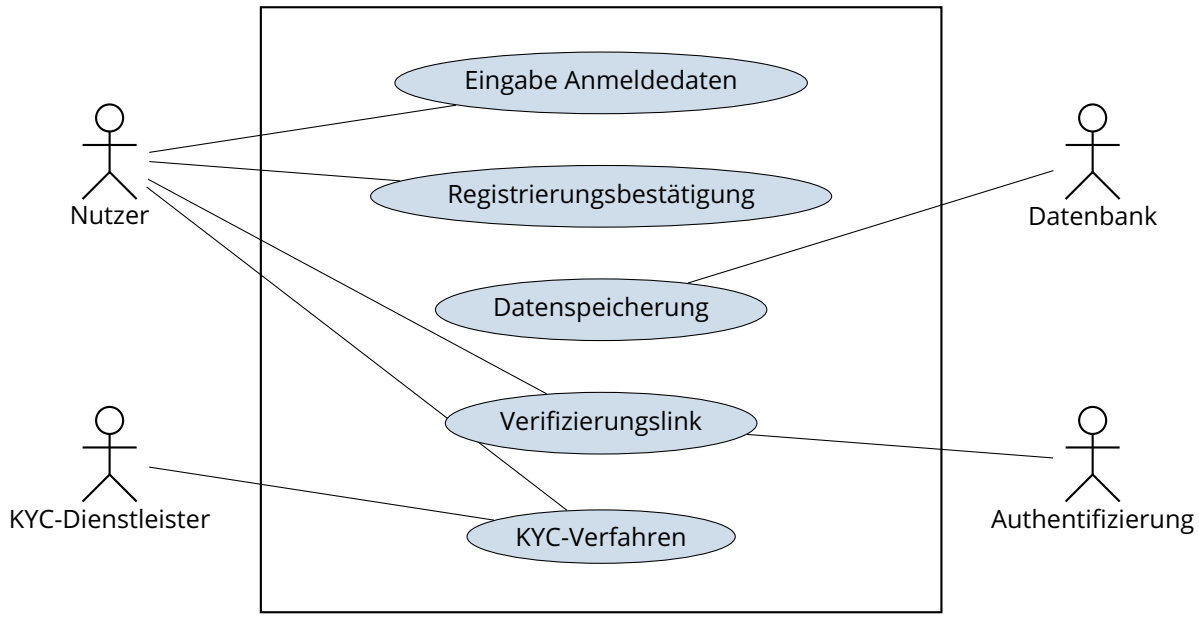

Abbildung 3.1: Use Case: Benutzerregistrierung

Benutzerverwaltung

Die Benutzerverwaltung ermöglicht es dem Nutzer, seine E-Mail-Adresse, sein Passwort und seine persönlichen Daten zu ändern sowie sein Benutzerkonto zu löschen. Die App zeigt dem Nutzer seine persönlichen Daten an und ermöglicht die Änderung der E-Mail-Adresse und des Passworts. Die Änderung der persönlichen Daten und die Löschung des Benutzerkontos werden in der App beantragt und vom Support bearbeitet. Eine Wiederherstellung des Passwortes ist ebenfalls über die App möglich.

Kartenansicht

Die Startseite der App zeigt eine Übersicht der Partnerfilialen, die den Kauf oder Verkauf von Bitcoins unterstützen. Ein Klick auf die Vorschau öffnet die Kartenansicht und zeigt alle Partnerfilialen in der Nähe des Nutzers. Ein Klick auf eine Filiale öffnet die Detailansicht und zeigt weitere Informationen zu dieser Filiale an. Bei Bedarf kann der Nutzer die Navigation über seine bevorzugte Kartenanwendung starten.

Finanzübersicht

Auf der Startseite werden ein Bitcoin-Preis-Chart und der aktuelle Bitcoin-Wechselkurs angezeigt. Der Nutzer kann sein Bitcoin-Guthaben, den aktuellen Marktwert, den durchschnittlichen Kaufpreis und den daraus resultierenden Gewinn oder Verlust einsehen.

Auftragsübersicht

Die Startseite enthält außerdem eine Übersicht der letzten Kauf- und Verkaufsaktivitäten des Nutzers. Über eine Schaltfläche gelangt der Nutzer zu einer vollständigen Liste aller Aufträge. In einer Detailansicht werden alle relevanten Informationen – einschließlich Auftragsnummer, Zeitstempel, Gebühren und Auftragsstatus – zu einem bestimmten Auftrag angezeigt. Der Nutzer hat die Möglichkeit, alle oder zeitlich gefilterte Transaktionen für Steuerzwecke im CSV-Format zu exportieren.

Kaufauftragserstellung

Der Nutzer wählt die Option zum Kauf von Bitcoins und gibt den Kaufbetrag ein. Auf der Kaufauftragsseite werden der aktuelle Bitcoin-Wechselkurs und die für den Kauf anfallenden Gebühren angezeigt. Sobald der Nutzer den Kauf bestätigt, wird der Kaufauftrag mit dem Status „unbezahlt“ in der Datenbank gespeichert. Die Datenbank generiert eine eindeutige Auftragsnummer und ordnet diese dem Kaufauftrag zu. Die App generiert einen Barcode, der die Auftragsnummer referenziert und mit dem Kasensystem der Partnerfilialen kompatibel ist. Das Use-Case-Diagramm in Abbildung 3.2 zeigt die Interaktion mit dem Softwaresystem bei der Kaufauftragserstellung.

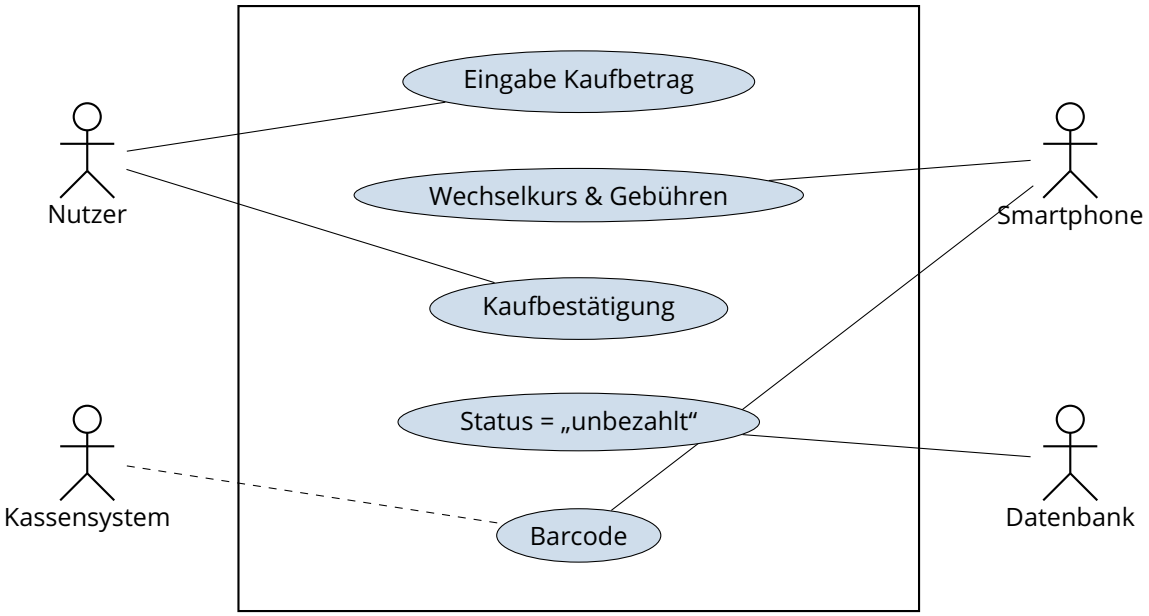

Abbildung 3.2: Use Case: Kaufauftragserstellung

Verkaufsauftragserstellung

Der Nutzer wählt die Option zum Verkauf von Bitcoins, wählt eine Auszahlungsmethode (*Bankkonto* oder *Gutschein*) und gibt den Verkaufsbetrag ein. Bei einer Auszahlung auf das Bankkonto des Nutzers ist die Eingabe der Kontodaten erforderlich. Die App zeigt den aktuellen Bitcoin-Wechselkurs und die für den Verkauf anfallenden Gebühren an. Sobald der Nutzer den Verkauf bestätigt, wird der Verkaufsauftrag mit dem Status „verkauft“ in der Datenbank gespeichert. Die Auszahlung auf das Bankkonto des Nutzers erfolgt durch eine Überweisung des Zahlungsdienstleisters. Bei einer Auszahlung nach dem Gutscheinprinzip wird ein Gutscheincode generiert, der in ausgewählten Partnerfilialen eingelöst werden kann. Das Use-Case-Diagramm in Abbildung 3.3 zeigt die Interaktion mit dem Softwaresystem bei der Verkaufsauftragserstellung.

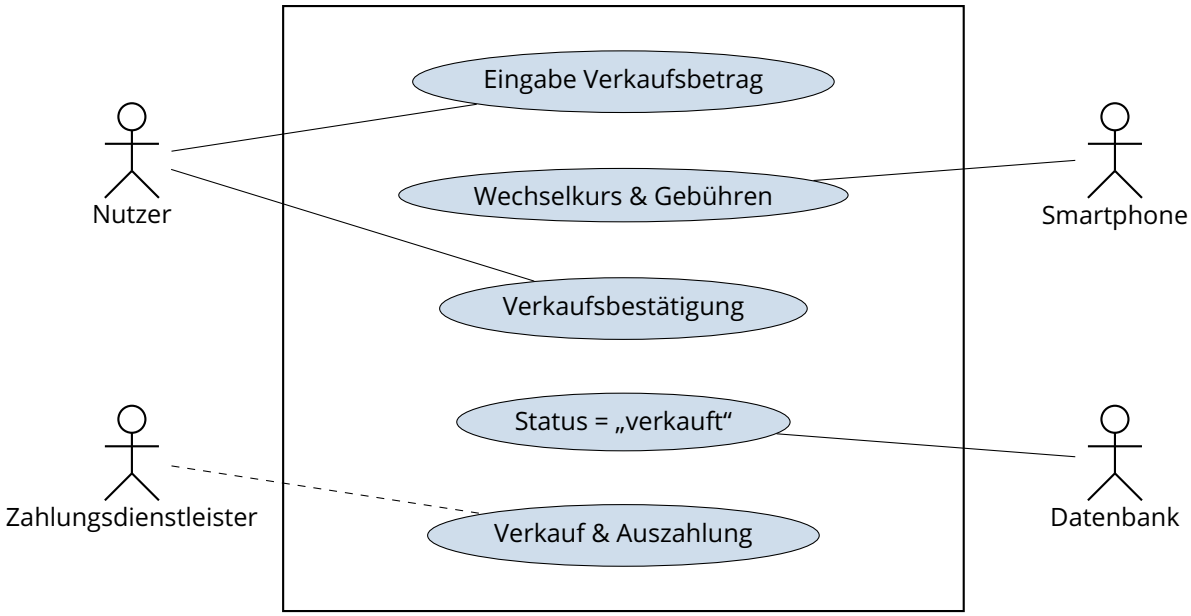

Abbildung 3.3: Use Case: Verkaufsauftragserstellung

Senden von Bitcoins

Dem Nutzer steht eine Custodial Wallet zur Verfügung, über die er Bitcoins an andere Wallets senden und von diesen empfangen kann. Die Wallet-Ansicht zeigt den aktuellen Saldo und die Transaktionshistorie des Nutzers an. Der Nutzer wählt die Option zum Versenden von Bitcoins und gibt den Betrag sowie die Adresse der Empfänger-Wallet ein. Die App überprüft die Gültigkeit der eingegebenen Daten, zeigt die erforderlichen Gebühren des Zahlungsdienstleisters an und fordert den Nutzer zur Bestätigung auf. Sobald der Nutzer die Transaktion bestätigt, wird der Sendeauftrag vom Zahlungsdienstleister ausgeführt. Die App zeigt den Status der Transaktion an und informiert den Nutzer, sobald die Transaktion abgeschlossen ist.

Empfangen von Bitcoins

Der Empfang von Bitcoins erfolgt ebenfalls über die Wallet-Ansicht in der App. Der Nutzer öffnet diese und wählt die Option zum Empfang von Bitcoins. Der Zahlungsdienstleister generiert eine Empfangsadresse, die dem Nutzer in der App angezeigt wird. Der Nutzer kann die Empfangsadresse oder einen QR-Code, der die Empfangsadresse referenziert, mit dem Sender teilen.

3.2.2 Kassensystem

Das Kassensystem (Kassenanwendung) ermöglicht es dem Kassierer, den Kauf- und Verkaufsprozess innerhalb einer Partnerfiliale zu steuern und bietet eine Umsatzübersicht und Verwaltungsoptionen für den Partner.

Partnerregistrierung

Die Registrierung des Partners erfolgt über die Kassenanwendung, die auf einem Tablet oder Smartphone installiert wird. Der Partner öffnet die App, wählt die Registrierungsoption und bestätigt die Konditionen und Nutzungsbedingungen für Partnerfilialen. Der Partner gibt seine Unternehmensdaten und die zugehörige E-Mail-Adresse ein, wählt ein sicheres Passwort und bestätigt die Registrierung. Das Partnerkonto wird in der Datenbank angelegt, eine eindeutige Partner-ID generiert und eine Bestätigungs-E-Mail an die hinterlegte E-Mail-Adresse versendet. Sobald der Partner die E-Mail bestätigt hat, wird er aufgefordert, seine Unternehmensdaten durch den KYC-Dienstleister und den Support verifizieren zu lassen. Bei erfolgreicher Prüfung wird das Partnerkonto und damit die Partnerfiliale freigeschaltet. Das Use-Case-Diagramm in Abbildung 3.4 zeigt die Interaktion mit dem Softwaresystem bei der Partnerregistrierung.

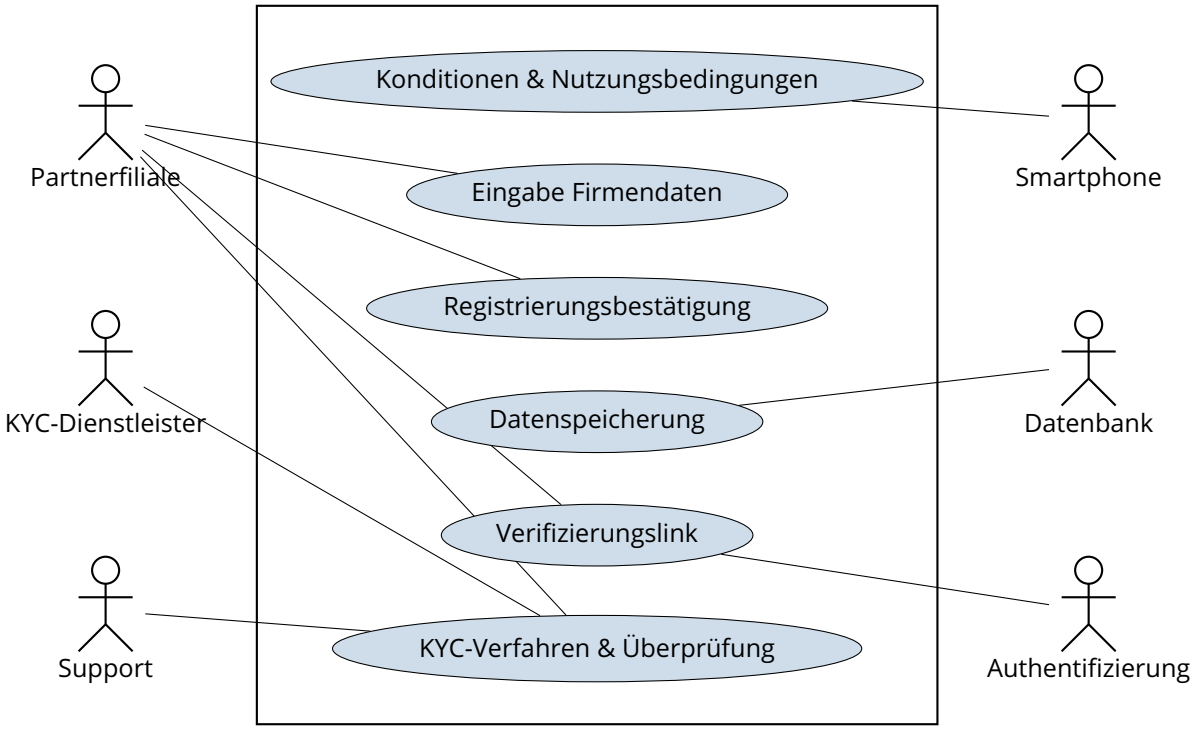

Abbildung 3.4: Use Case: Partnerregistrierung

Auftragserfassung

Die Auftragserfassung erfolgt durch den Kassierer mit Hilfe der Kassenanwendung. Der Kassierer startet die Auftragserfassung innerhalb der App und scannt den Barcode des Nutzers. Das Kassensystem interpretiert den Barcode und ruft die entsprechenden Auftragsdaten aus der Datenbank ab.

Erfassung eines Kaufauftrags: Das Kassensystem prüft die Gültigkeit des Kaufauftrags und zeigt den zu zahlenden Betrag an. Der Kassierer nimmt die Zahlung in bar oder per EC-Karte entgegen und bestätigt diese in der App. Die Datenbank speichert die Zahlungsinformationen

(Zeitstempel, Betrag, Signatur der Partnerfiliale) und aktualisiert den Status des Kaufauftrags auf „bezahlt“. Das Use-Case-Diagramm in Abbildung 3.5 zeigt die Interaktion mit dem Softwaresystem bei der Erfassung von Kaufaufträgen.

Erfassung eines Verkaufsauftrags: In ausgewählten Partnerfilialen können Gutscheine für den Verkauf von Bitcoins eingelöst werden. Das Kassensystem prüft die Gültigkeit des Gutscheins und zeigt den auszuzahlenden Betrag an. Der Kassierer zahlt den Betrag an den Kunden aus und bestätigt die Auszahlung in der App. Die Datenbank speichert die Zahlungsinformationen (Zeitstempel, Betrag, Signatur der Partnerfiliale) und aktualisiert den Status des Verkaufsauftrags auf „ausgezahlt“.

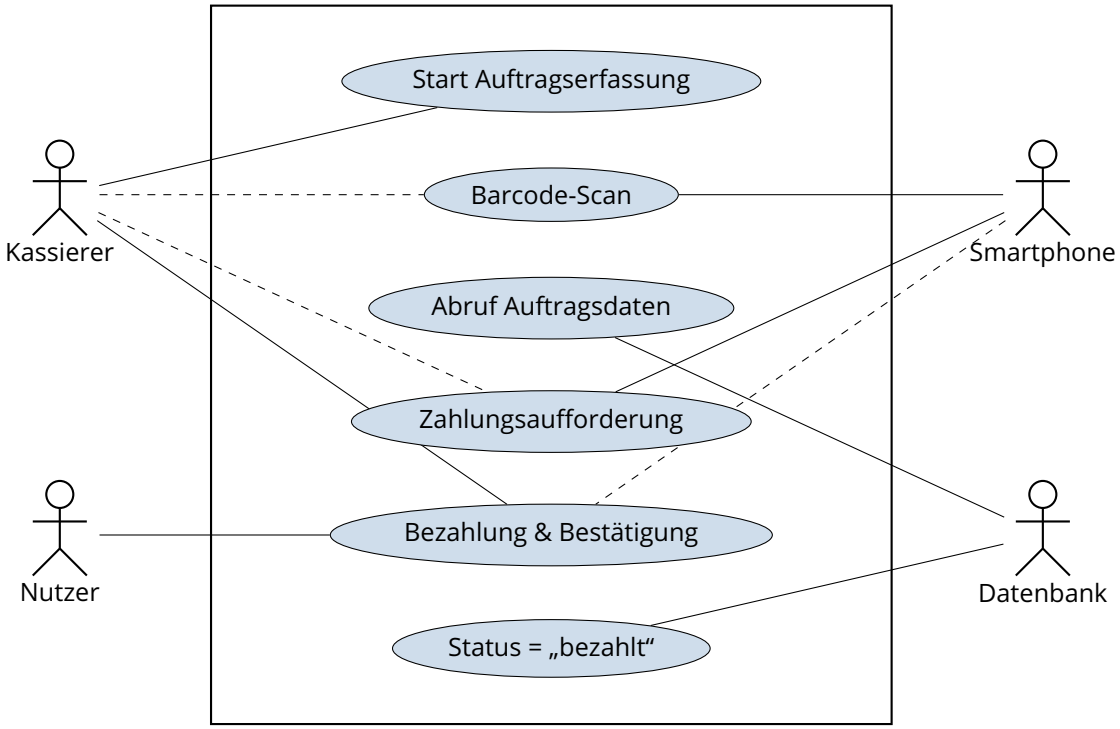

Abbildung 3.5: Use Case: Erfassung von Kaufaufträgen

Umsatzübersicht

Die Umsatzübersicht zeigt dem Partner eine Liste aller getätigten Kauf- und Verkaufsaktivitäten in seiner Filiale. Der Partner hat die Möglichkeit, alle oder zeitlich gefilterte Transaktionen für Steuerzwecke zu exportieren. Die steuerrelevanten Informationen können im CSV-Format exportiert werden.

Kassenterminal-Integration

Das Kassensystem kann optional in ein handelsübliches Kassenterminal integriert werden. Die Anbindung erfolgt über die Schnittstelle des Kassenterminals und ermöglicht die Auftragserfassung und Zahlungsabwicklung in Euro in einem gemeinsamen System. Der Kassierer authentifiziert sich gegenüber dem System und erhält Zugang zur Auftragserfassung. Die

Auftragserfassung erfolgt analog zur Kassenanwendung (vgl. Auftragserfassung) und wird über das Display und den Barcodescanner des Kassenterminals gesteuert. Die Integration in ein handelsübliches Kassenterminal ist kein Anfangsbestandteil der Software und steht unter dem Vorbehalt der technischen Machbarkeit.

3.2.3 Zahlungsdienstleister

Der Zahlungsdienstleister realisiert den Kauf und Verkauf von Bitcoins und verwahrt diese für den Nutzer in einer Custodial Wallet.

Kaufprozess

Das System verarbeitet die bezahlten Kaufaufträge automatisiert und beauftragt den Zahlungsdienstleister mit dem Kauf der Bitcoins. Der Zahlungsdienstleister führt den Bitcoin-Kauf durch, stellt die gekauften Bitcoins in einer Custodial Wallet zur Verfügung und meldet den erfolgreichen Abschluss der Transaktion an das System zurück. Das System speichert die Kaufinformationen (Zeitstempel, Betrag, Signatur des Zahlungsdienstleisters) und aktualisiert den Status des Auftrags auf „gekauft“. Die Kundenanwendung informiert den Nutzer über den erfolgreichen Abschluss des Kaufauftrags. Das Use-Case-Diagramm in Abbildung 3.6 zeigt die Interaktion mit dem Softwaresystem während des Kaufprozesses.

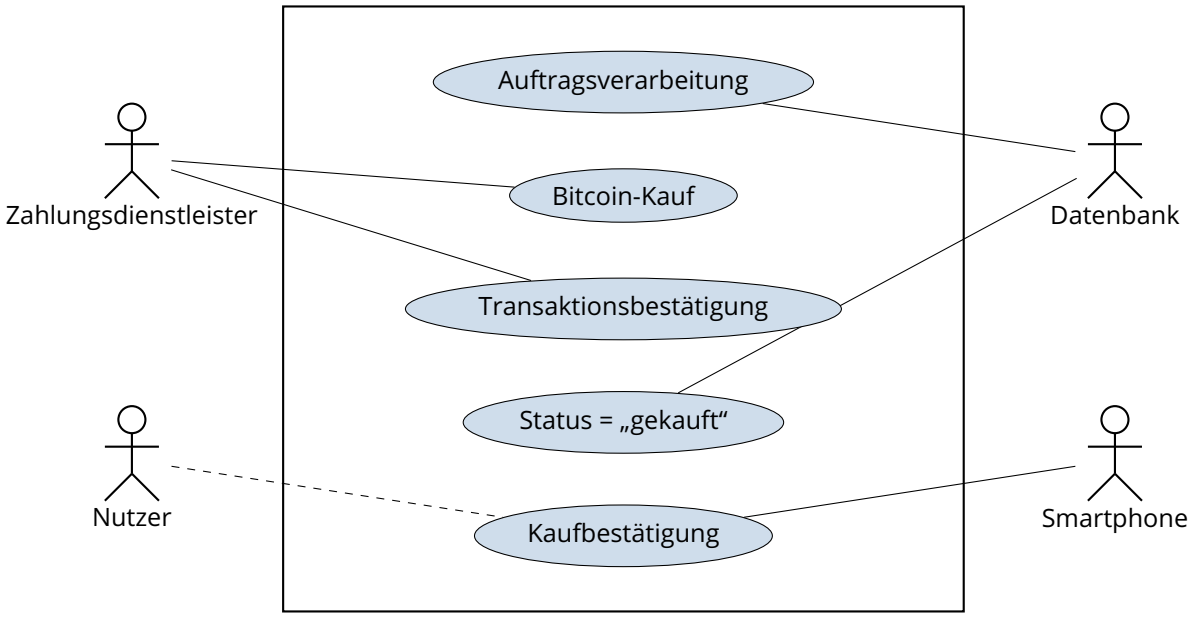

Abbildung 3.6: Use Case: Kaufprozess

Verkaufsprozess

Der Verkaufsprozess verläuft analog zum Kaufprozess, unterscheidet sich aber in der Auszahlung. Vor dem Bitcoin-Verkauf prüft der Zahlungsdienstleister die Einhaltung der AML-Richtlinien. Liegt ein AML-Verdacht vor, erhält der Nutzer eine Benachrichtigung und wird an den Support verwiesen. Liegt kein AML-Verdacht vor, erfolgt der Verkauf der Bitcoins und die Auszahlung gemäß der gewählten Auszahlungsmethode.

Auszahlung auf das Bankkonto: Die Auszahlung erfolgt durch Überweisung auf das Bankkonto des Nutzers. Das System speichert die Verkaufsinformationen (Zeitstempel, Betrag, Signatur des Zahlungsdienstleisters) und aktualisiert den Status des Auftrags auf „ausgezahlt“. Die Kundenanwendung informiert den Nutzer über die erfolgreiche Auszahlung des Verkaufsbetrags. Das Use-Case-Diagramm in Abbildung 3.7 zeigt die Interaktion mit dem Softwaresystem während des Verkaufsprozesses bei einer Auszahlung auf das Bankkonto.

Auszahlung nach dem Gutscheinprinzip: Das System generiert einen Gutscheincode und aktualisiert den Auftragsstatus auf „Gutscheincode“. Die Kundenanwendung informiert den Nutzer über den erfolgreichen Abschluss des Verkaufsauftrags und zeigt den Gutscheincode als Barcode an. Der Nutzer kann den Gutscheincode in ausgewählten Partnerfilialen gegen Bargeld einlösen.

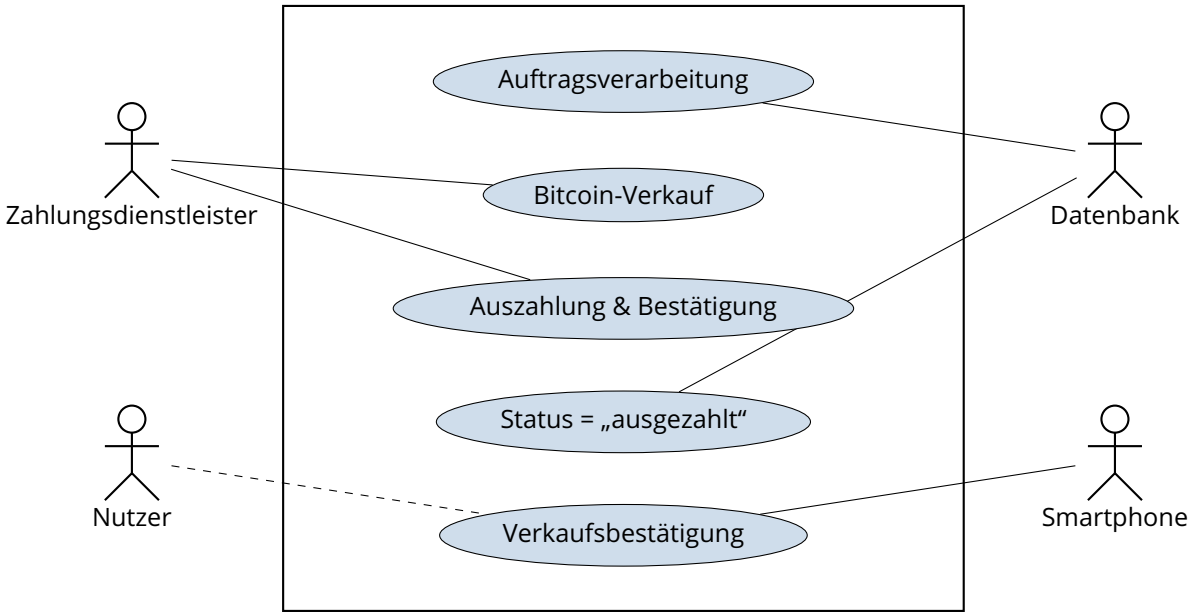

Abbildung 3.7: Use Case: Verkaufsprozess

3.2.4 Monitoring & Reporting

Das System überwacht alle Transaktionen und speichert wichtige System- und Nutzerdaten in einer Datenbank. Die Daten werden zu Analyse- und Berichtszwecken sowie zur Einhaltung gesetzlicher Vorschriften verwendet. Die Datenbank stellt verschiedene Filter- und Exportfunktionen zur Verfügung, um die Daten gezielt abrufen und analysieren zu können. Das System versendet regelmäßig Statusberichte und informiert den Support bei Auffälligkeiten. Steuerrelevante Informationen können im CSV-Format exportiert werden.

3.2.5 Verwaltung

Support und Administratoren haben einen erweiterten Zugriff auf die Datenbank und das System. Das System bietet eine Verwaltungsoberfläche für die Verwaltung von Benutzerkonten, Partnerfilialen und Aufträgen. Administratoren haben vollen Zugriff auf die Datenbank und können Daten einsehen, bearbeiten und löschen.

3.3 Analysemodell

Das Analysemodell beschreibt die logische Umsetzung der funktionalen Anforderungen. Es enthält keine Details zur Implementierung [Gmb21]. Die Hauptkomponenten des Systems werden identifiziert und in einem Klassendiagramm dargestellt. Basierend auf den funktionalen Anforderungen kann das System in die folgenden Hauptkomponenten unterteilt werden:

- Geschäftslogik
- Kundenanwendung
- Kassensystem
- KYC-Dienstleister-Schnittstelle
- Zahlungsdienstleister-Schnittstelle
- Authentifizierung
- Datenbank
- Monitoringsystem
- Verwaltung

Die Geschäftslogik umfasst die Kernfunktionen der Software und ist über Schnittstellen mit den internen und externen Systemkomponenten verbunden. Die Kundenanwendung und das Kassensystem werden projektintern entwickelt und als mobile Anwendungen für iOS und Android bereitgestellt. Authentifizierung, Datenbank, Monitoringsystem und Verwaltung werden ebenfalls projektintern entwickelt und betrieben. Die Verifizierung der Nutzeridentität mittels KYC-Verfahren wird an einen externen KYC-Dienstleister ausgelagert. Die Abwicklung von Kauf- und Verkaufsaufträgen sowie die Verwahrung der Bitcoins erfolgt über einen externen Zahlungsdienstleister. Das Klassendiagramm in Abbildung 3.8 zeigt die logische Struktur des Systems.

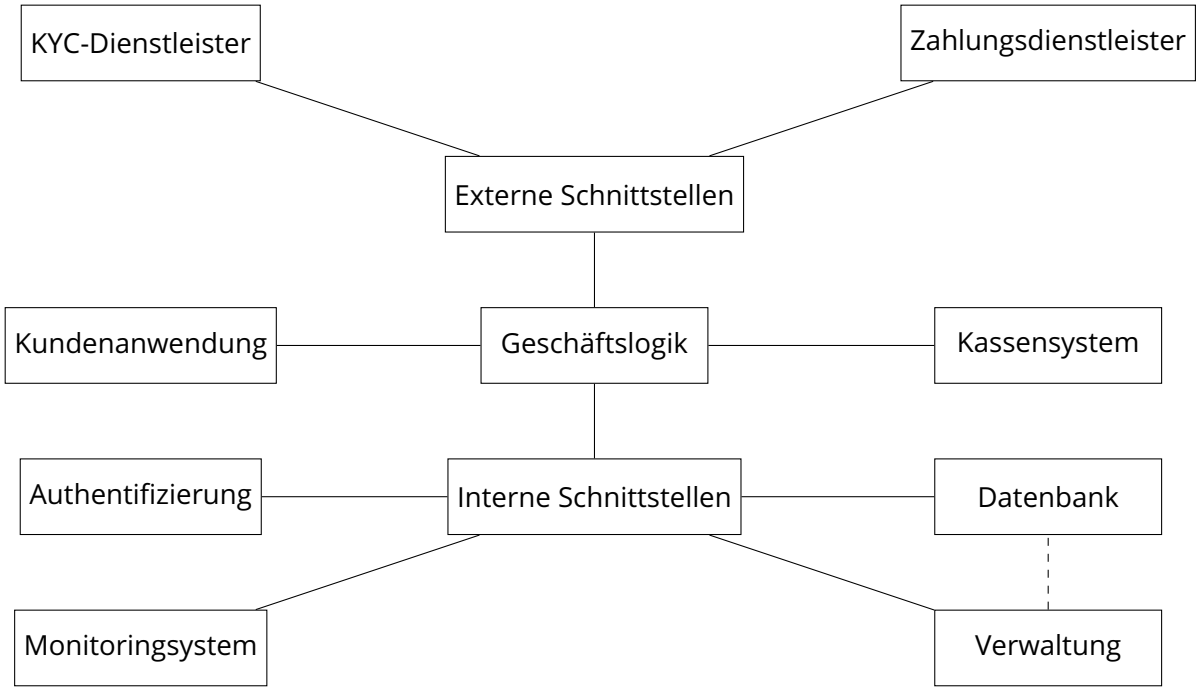

Abbildung 3.8: Klassendiagramm: Logische Struktur des Systems

3.4 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen beschreiben die Qualitätsanforderungen an das System [Bal10]. Die *ISO/IEC 25010* definiert acht Qualitätsmerkmale, die auf einem internationalen Konsens zur Bewertung von Softwaresystemen basieren: *Sicherheit*, *Zuverlässigkeit*, *Usability*, *Performanz*, *Kompatibilität*, *Wartbarkeit*, *Funktionalität* und *Portabilität* [Hao+17]. Diese sind entscheidend für den sicheren und zuverlässigen Betrieb von Softwaresystemen und die Zufriedenheit der Nutzer. Die Qualitätsmerkmale werden an die Anforderungen des Projekts angepasst und entsprechend ihrer Bedeutung für das Projekt priorisiert. Darüber hinaus ergeben sich weitere anwendungsspezifische Anforderungen in den Bereichen Regulatorik & Datenschutz und Monetarisierung. Dies betrifft die Einhaltung gesetzlicher Vorschriften beim Bitcoin-Handel und den Schutz personenbezogener Daten sowie die Sicherstellung der Wirtschaftlichkeit des Projekts. Für das Projekt ergeben sich daraus folgende nicht-funktionale Anforderungen:

3.4.1 Regulatorik & Datenschutz

> Die Geldwäscheaufsicht der BaFin [...] hat das Ziel, bei systemischen Mängeln in der Prävention von Geldwäsche und Terrorismusfinanzierung bei den Verpflichteten einzugreifen und diesen entgegenzuwirken. Die BaFin kann [...] Bußgelder verhängen oder Geschäftsleiterinnen und -leiter verwarnen oder gar abberufen. Als ultima ratio kann sie auch einen Antrag bei der EZB stellen, die Erlaubnis eines Instituts aufzuheben. [Mül18]

Die „Cash2Coin“-Plattform muss die AML-Richtlinien bei der Auftragsabwicklung einhalten und die Nutzerdaten gemäß den gesetzlichen Bestimmungen speichern und verarbeiten. Der externe Zahlungsdienstleister muss die Einhaltung der AML-Richtlinien beim Bitcoin-Handel sicherstellen und zusätzlich über eine Kryptoverwahrlizenz verfügen. Zur Einhaltung der AML-Richtlinien sind die Transaktionen auf Auffälligkeiten in Bezug auf Geldwäsche und Terrorismusfinanzierung zu prüfen [Mül18] und die Nutzerdaten mittels eines KYC-Verfahrens zu verifizieren [BaF18]. Die Datenschutz-Grundverordnung (DSGVO) ist für alle Mitgliedsstaaten der Europäischen Union verbindlich und regelt den Schutz personenbezogener Daten. In Deutschland wird die DSGVO durch das Bundesdatenschutzgesetz (BDSG) konkretisiert [Bun17]. Die Einhaltung der DSGVO erfordert eine DSGVO-konforme Speicherung und Verarbeitung von Nutzerdaten. Dies umfasst die Einhaltung datenschutzrechtlicher Grundsätze wie die *Rechtmäßigkeit der Verarbeitung*, *Transparenz*, *Zweckbindung*, *Datensparsamkeit*, *Richtigkeit*, *Speicherbegrenzung*, *Datenintegrität*, *Datenvertraulichkeit* und eine *Rechenschaftspflicht* [Bun17; Ans22]. Darüber hinaus gelten für den Handel mit Kryptowährungen steuerliche Regelungen und Meldepflichten [Kös24], die von den Nutzern und der Plattform eingehalten werden müssen. Dazu gehören die Erfassung und Speicherung von Transaktionsdaten, die Erstellung von Transaktionsberichten und eine Exportfunktion für die Finanzbehörden.

3.4.2 Sicherheit

Der Handel mit Kryptowährungen macht die Plattform zu einem attraktiven Ziel für Angreifer und Betrüger. Innerhalb der letzten 12 Monate wurden weltweit rund 1,58 Milliarden US-Dollar durch Angriffe auf Kryptowährungen erbeutet. Ein Großteil der Angriffe richtet sich gegen zentralisierte Börsen [Cha24].

> Funds stolen in crypto heists increased Year-over-Year (YoY), nearly doubling from $857M to $1.58B through the end of July. Crypto thieves also appear [...] targeting centralized exchanges with greater frequency rather than prioritizing DeFi protocols, which are less popular vehicles for trading BTC. [Cha24]

Die Sicherheit der personenbezogenen Daten, der Transaktionen und der Serverinfrastruktur ist für das Vertrauen in die Plattform von zentraler Bedeutung. Die Plattform muss über ein sicheres Authentifizierungs- und Autorisierungssystem verfügen, um den unbefugten Zugriff auf sensible Daten zu verhindern. Die Datenübertragung zwischen den Systemkomponenten und den Nutzern muss stets verschlüsselt erfolgen. Sicherheitslücken durch veraltete oder falsch konfigurierte Softwarekomponenten stellen ein großes Risiko dar. Das System muss regelmäßig auf Sicherheitslücken überprüft und aktualisiert werden, um Angriffe auf bekannte Schwachstellen zu verhindern. Wichtige Systemkomponenten müssen überwacht und auf Auffälligkeiten überprüft werden. Regelmäßige Datensicherungen sollen eine Wiederherstellung im Fehlerfall ermöglichen.

3.4.3 Zuverlässigkeit & Verfügbarkeit

Das System soll eine hohe Ausfallsicherheit bieten und gegen Angriffe auf die Serverinfrastruktur geschützt sein.

> The DoS attack is a major concern in cybersecurity, as it results in disruption of digital services availability related to the targeted large businesses, government institutions, and other industries. Centralized crypto exchange platforms [...] are vulnerable to distributed denial of service attacks. [Cha+22]

Ein zentrales Risiko stellt die Möglichkeit von Denial-of-Service-Angriffen (DoS) dar [Cha+22]. Dabei versuchen Angreifer durch gezielte Angriffe auf die Netzwerk- und Serverinfrastruktur die Verfügbarkeit der Plattform zu beeinträchtigen. Insbesondere zentralisierte Systeme sind anfällig für DoS-Angriffe. Einzelne Dienste und Server können durch gezielte Anfragen überlastet und die Systemressourcen erschöpft werden. Dadurch können keine weiteren Anfragen mehr bearbeitet werden, was zum Ausfall einzelner Dienste oder des gesamten Systems führt [Cha+22]. Zur Abwehr von DoS-Angriffen müssen sowohl netzwerkbasierte als auch systeminterne Schutzmaßnahmen implementiert werden. Darüber hinaus soll die Zuverlässigkeit des Systems durch regelmäßige Funktionstests und die Überwachung der Serverinfrastruktur sichergestellt werden. Eine redundante Datenhaltung soll Datenverluste verhindern und im Fehlerfall den Weiterbetrieb des Systems ermöglichen. Bei einem Ausfall von Systemkomponenten oder der gesamten Plattform muss eine schnelle Wiederherstellung des Systems möglich sein.

3.4.4 Performanz & Skalierbarkeit

Für einen reibungslosen Ablauf im Kauf- und Verkaufsprozess ist es wichtig, dass die Kundenanwendung und das Kassensystem schnell und zuverlässig auf Anfragen reagieren. Transaktionen und Serviceanfragen sollen in Echtzeit verarbeitet werden, um lange Wartezeiten in den Partnerfilialen zu vermeiden und die Zufriedenheit der Nutzer zu erhöhen. Das System soll auch bei hohen Nutzerzahlen und Transaktionsaufkommen schnelle Reaktionszeiten und kurze Ladezeiten gewährleisten. Eine skalierbare Serverinfrastruktur soll sicherstellen, dass steigende Nutzerzahlen bewältigt und weitere Partnerfilialen und Dienstleister integriert werden können. Eine effiziente Serverauslastung und die Optimierung von Datenbankabfragen sollen die Wirtschaftlichkeit des Systems soll auch bei steigenden Nutzerzahlen gewährleisten.

3.4.5 Kompatibilität & Erweiterbarkeit

Die Software muss flexibel und erweiterbar sein, um den sich ändernden Anforderungen der Nutzer und des Marktes gerecht zu werden. Die Integration neuer Funktionen und Dienstleister sowie die Anpassung interner Prozesse soll ohne großen Aufwand möglich sein. Das System soll einen einfachen Wechsel zwischen verschiedenen Kassensystemen, Zahlungsdienstleistern und KYC-Dienstleistern ermöglichen. Die Interoperabilität mit externen Systemen und Dienstleistern soll durch die Verwendung offener Standards und einheitlicher Schnittstellen gewährleistet werden. Eine besondere Herausforderung stellt die Integration in bestehende Kassensysteme dar. Daher soll das Kassensystem als eigenständige Anwendung entwickelt und erst nach erfolgreicher Erprobung in bestehende Kassensysteme integriert werden. Die Schnittstelle zur Kundenanwendung und zum Kassensystem soll plattform- und geräteunabhängig sein. Im Kauf- und Verkaufsprozess sollen standardisierte Barcode- und QR-Code-Formate verwendet werden, um die Kompatibilität mit unterschiedlichen Endgeräten zu gewährleisten. Eine mögliche Expansion in internationale Märkte und die Akzeptanz alternativer Währungen und Zahlungsmethoden sollen bereits bei der Entwicklung berücksichtigt werden.

3.4.6 Wartbarkeit

Die Software soll einfach zu warten und zu aktualisieren sein, um die Lebensdauer des Systems zu erhöhen und die Entwicklungskosten zu senken. Die Aktualisierung einzelner Komponenten und Schnittstellen soll ohne Beeinträchtigung des Gesamtsystems möglich sein. Der Quellcode soll übersichtlich und gut dokumentiert sein, um die Wartung und Weiterentwicklung zu erleichtern. Das System soll über eine zentrale Benutzeroberfläche zur Verwaltung und Überwachung aller Systemkomponenten verfügen. Wichtige Systemereignisse sollen automatisch protokolliert und überwacht werden. Anhand von Fehlerberichten und Protokollen sollen Probleme schnell erkannt und behoben werden können.

3.4.7 Usability & Support

Die Interaktion mit der Kundenanwendung und dem Kassensystem soll intuitiv und benutzerfreundlich sein, um die Akzeptanz und Zufriedenheit der Nutzer zu erhöhen. Die Bedienung der Anwendung soll selbsterklärend und auch für unerfahrene Nutzer leicht verständlich sein. Auch Status- und Fehlermeldungen sollen für den Nutzer verständlich und nachvollziehbar sein. Die Kundenanwendung und das Kassensystem sollen personalisierbar sein, um den individuellen Bedürfnissen und Fähigkeiten der Nutzer gerecht zu werden. Die Nutzer sollen die Möglichkeit haben, ihre bevorzugte Sprache, ein individuelles Design und ihre Währungspräferenzen einzustellen. Der Support soll Nutzer bei Fragen und Problemen unterstützen und eine schnelle Problemlösung gewährleisten. Zur effizienten Bearbeitung von Supportanfragen soll eine interne Benutzeroberfläche für Administratoren und Supportmitarbeiter zur Verfügung stehen.

3.4.8 Monetarisierung

Die Wirtschaftlichkeit des Systems soll durch Minimierung der Betriebskosten und durch Einnahmen aus Gebühren und Kooperationen mit Geschäftspartnern sichergestellt werden. Die Betriebskosten des Systems sollen durch den Einsatz von Open Source Software und kostengünstigen Drittanbieterlösungen minimiert werden. Die Einnahmen sollen durch Gebühren für den Kauf und Verkauf von Bitcoins sowie durch die Kooperationen mit Partnerfilialen und externen Dienstleistern generiert werden. Die Gebühren sollen für den Nutzer transparent und nachvollziehbar sein und in einem angemessenen Verhältnis zur erbrachten Leistung stehen. Die Geschäftspartner sollen regelmäßig über die erzielten Umsätze und wichtige Finanzkennzahlen informiert werden.

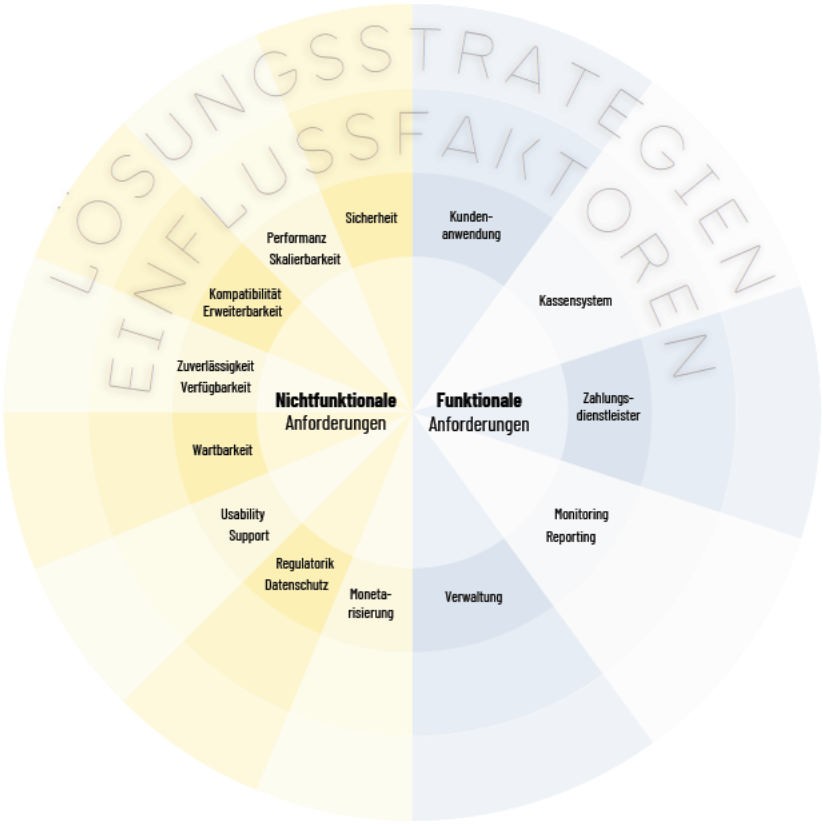

Abbildung 3.9: Strukturierter Entwicklungsprozess – Anforderungen

4 Architekturentwurf

Basierend auf den Ergebnissen der Anforderungsanalyse (vgl. Strukturdiagramm Abb. 3.9) erfolgt der Entwurf der Softwarearchitektur. Die Softwarearchitektur soll die funktionalen und nicht-funktionalen Anforderungen an das Softwaresystem erfüllen und eine solide Grundlage für die Implementierung bilden. Im Rahmen des Architekturentwurfs werden die Einflussfaktoren auf die Architektur identifiziert und geeignete Lösungsstrategien entwickelt. Anschließend werden die konkreten Architekturentscheidungen zusammengefasst und übersichtlich dargestellt.

4.1 Einflussfaktoren

Im Folgenden werden die Einflussfaktoren identifiziert, die bei der Entwicklung der Softwarearchitektur eine wesentliche Rolle spielen. Die Einflussfaktoren ergeben sich aus den Anforderungen an das Softwaresystem und den technischen Rahmenbedingungen des Softwareprojekts. Die Einflussfaktoren werden in die Kategorien Produktfaktoren, Qualitätsfaktoren und technische Faktoren unterteilt.

4.1.1 Produktfaktoren

Die Produktfaktoren umfassen die Einflussfaktoren, die das Produkt und seine Funktionalität betreffen.

Einfluss P1 → Benutzeroberfläche

Das System soll eine intuitive und benutzerfreundliche Oberfläche bieten, die einer breiten Zielgruppe – unabhängig von Alter und technischen Kenntnissen – den Zugang zum Bitcoin-Handel erleichtert. Durch eine klare Trennung von Benutzeroberfläche und Geschäftslogik sollen Designanpassungen ohne Auswirkungen auf das Gesamtsystem möglich sein. Die Kundenanwendung und die Kassenanwendung werden als mobile Anwendungen für die Betriebssysteme iOS und Android entwickelt. Es wird erwartet, dass beide Anwendungen unabhängig von der Plattform ein einheitliches Design aufweisen und eine konsistente Benutzererfahrung bieten.

→ Lösungsstrategie P1

Einfluss P2 → Benutzerinteraktion

Status- und Fehlermeldungen des Systems müssen für den Nutzer verständlich und nachvollziehbar sein. Die Nutzer sollen per E-Mail und Push-Benachrichtigung über wichtige Ereignisse und den Status ihrer Transaktionen informiert werden. Der Versand von E-Mails zur Bestätigung der Registrierung soll innerhalb der Komponente zur Benutzerrregistrierung erfolgen.

Der Versand von E-Mails zur Kundeninteraktion und zu Marketingzwecken soll durch einen externen E-Mail-Dienstleister realisiert werden. Für den Versand von Push-Benachrichtigungen an die Kundenanwendung ist eine Schnittstelle zu einem Push-Benachrichtigungsdienst erforderlich.

→ Lösungsstrategie P2

Einfluss P3 → Personalisierung

Die Architektur ist so zu konzipieren, dass die Nutzer die Kundenanwendung und die Kassenanwendung an ihre individuellen Bedürfnisse anpassen können. Die Nutzer sollen ihre bevorzugte Sprache, ihr bevorzugtes Design und ihre Währungspräferenzen einstellen können. Anpassungen der individuellen Benutzereinstellungen sollen ohne Auswirkungen auf das Gesamtsystem erfolgen. Änderungen an Benutzerprofilen und -einstellungen müssen innerhalb des Systems synchronisiert und zentral gespeichert werden.

→ Lösungsstrategie P3

Einfluss P4 → Bitcoin-Handel

Der Kauf und Verkauf sowie die Verwahrung von Bitcoins wird an einen externen Dienstleister ausgelagert. Die Architektur muss eine robuste Schnittstelle zur Anbindung des Zahlungsdienstleisters bereitstellen. Ausfälle oder Verzögerungen seitens des Zahlungsdienstleisters müssen vom System erkannt und behandelt werden. Betrugsversuche und verdächtige Transaktionen müssen durch ein Betrugserkennungssystem erkannt und blockiert werden.

→ Lösungsstrategie P4

Einfluss P5 → Abrechnung

Die Abrechnung muss für Partnerfilialen und externe Dienstleister transparent und nachvollziehbar sein. Das System muss sicherstellen, dass alle Finanzdaten korrekt erfasst und verbucht werden. Die Geschäftspartner sollen regelmäßig über die erzielten Umsätze und wichtige Finanzkennzahlen informiert werden. Die notwendigen Finanzdaten sollen automatisiert erfasst und in einem zentralen Dashboard dargestellt werden. Finanzberichte und Abrechnungen sollen automatisiert erstellt und in einem standardisierten Format für den Export bereitgestellt werden. Für eine effiziente Verwaltung und Auswertung der Finanzdaten kann zukünftig eine direkte Anbindung an eine externe Buchhaltungssoftware erforderlich sein.

→ Lösungsstrategie P5

Einfluss P6 → Gebührenverwaltung

Die Gebühren müssen für den Nutzer transparent und nachvollziehbar sein. Die Erfassung und Abrechnung der Gebühren soll automatisiert erfolgen und in einem zentralen Dashboard verwaltet werden können. Innerhalb der Kundenanwendung soll der Nutzer über die Gebühren für den Kauf und Verkauf sowie für das Senden und Empfangen von Bitcoins informiert werden. Dazu soll der aktuelle Bitcoin-Wechselkurs abgefragt und die Gebühren dynamisch berechnet werden.

→ Lösungsstrategie P6

Einfluss P7 → Support

Der Support hat die Aufgabe, Benutzeranfragen schnell und effizient zu bearbeiten und die Nutzer bei Fragen und Problemen zu unterstützen. Für die effiziente Bearbeitung von Supportanfragen ist ein zentrales Support-System erforderlich. Das System soll eine Plattform für die Erfassung, Priorisierung und Bearbeitung von Supportanfragen bieten. Eine hohe Anzahl von Supportanfragen kann zu einer Überlastung des Support-Teams und zu langen Bearbeitungszeiten führen. Der Prozess der Bearbeitung von Supportanfragen sollte daher weitestgehend automatisiert werden.

→ Lösungsstrategie P7

4.1.2 Qualitätsfaktoren

Die Qualitätsfaktoren umfassen die Einflussfaktoren, die die Qualitätsanforderungen an das Produkt betreffen.

Einfluss Q1 → Transaktionssicherheit

Die Nutzer müssen vor finanziellen Verlusten geschützt werden. Die Integrität der Transaktionen, die Sicherheit der Zahlungen und die Vertraulichkeit der Daten müssen gewährleistet sein. Die Überwachung und Kontrolle der Transaktionen erfordert die Implementierung eines Betrugserkennungssystems. Verdächtige Transaktionen und ungewöhnliche Aktivitäten müssen identifiziert und blockiert werden. Potenzielle Sicherheitslücken müssen durch regelmäßige Überprüfungen identifiziert und behoben werden.

→ Lösungsstrategie Q1

Einfluss Q2 → Netzwerksicherheit

Teile des Systems sind über das Internet öffentlich zugänglich und damit einer Vielzahl von Angriffen ausgesetzt. Die Serverinfrastruktur und die Netzwerkkomponenten müssen vor Angriffen und unberechtigten Zugriffen geschützt werden. Dies erfordert eine sichere Konfiguration der Netzwerkkomponenten, regelmäßige Sicherheitsupdates und eine kontinuierliche Überwachung und Wartung des Systems. Alle Schnittstellen und Kommunikationsprotokolle müssen den aktuellen Sicherheitsstandards entsprechen und korrekt konfiguriert sein.

→ Lösungsstrategie Q2

Einfluss Q3 → Zugriffsschutz

Unbefugter Zugriff auf Benutzerkonten oder sensible Daten kann zu finanziellen Verlusten und zu einem Vertrauensverlust bei den Nutzern führen. Betrugsversuche und Datenlecks müssen verhindert und die Vertraulichkeit und Integrität der Daten sichergestellt werden. Das System soll eine sichere Benutzerauthentifizierung und eine rollenbasierte Zugriffskontrolle unterstützen. Nur Nutzer mit gültigen Anmeldeinformationen und entsprechenden Berechtigungen dürfen Zugang zum System und zu sensiblen Daten erhalten.

→ Lösungsstrategie Q3

Einfluss Q4 → Datenspeicherung

Die Nutzerdaten müssen sicher und gesetzeskonform (DSGVO) gespeichert und verarbeitet werden. Die Speicherung der persistenten Daten soll auf einem zentralen Datenbankserver erfolgen. Die Daten müssen verschlüsselt und vor unberechtigtem Zugriff geschützt gespeichert werden. Der Zugriff auf die Datenbank darf nur durch autorisierte Nutzer und Systemkomponenten erfolgen. Um Datenverluste zu vermeiden sind geeignete Sicherheitsmaßnahmen, regelmäßige Backups und ein zuverlässiges Wiederherstellungsverfahren erforderlich. Dies beeinflusst die Wahl der Datenbanktechnologie, die Konfiguration der Datenbankserver und die Zugriffsmethoden auf die Datenbank.

→ Lösungsstrategie Q4

Einfluss Q5 → Datenübertragung

Die Integrität und Vertraulichkeit der Daten muss bei der Übertragung im Netzwerk gewährleistet sein. Die Datenübertragung zwischen den Systemkomponenten, den Nutzern und den externen Dienstleistern muss verschlüsselt erfolgen. Dies erfordert die Auswahl sicherer Verschlüsselungsverfahren und Kommunikationsprotokolle sowie die Konfiguration sicherer Netzwerkverbindungen.

→ Lösungsstrategie Q5

Einfluss Q6 → Systemverfügbarkeit

Das System muss jederzeit betriebsbereit sein und eine hohe Verfügbarkeit der Dienste gewährleisten. Hohe Datenlasten dürfen nicht zu Performanceproblemen und langsamen Antwortzeiten führen. Der Ausfall einzelner Komponenten oder Server darf nicht zum Ausfall des Gesamtsystems führen. Die Architektur muss Mechanismen zur automatischen Wiederherstellung der Dienste im Fehlerfall vorsehen. Die Serverinfrastruktur, die Datenbank und andere kritische Systemkomponenten müssen redundant ausgelegt sein.

→ Lösungsstrategie Q6

Einfluss Q7 → Skalierbarkeit

Das System muss auch bei hoher Last durch steigende Nutzerzahlen und Transaktionsvolumina eine hohe Performanz und Verfügbarkeit gewährleisten. Bei gleichzeitiger Nutzung des Systems durch eine hohe Anzahl von Nutzern müssen die Kundenanwendung und das Kassensystem in der Lage sein, Anfragen zeitnah und zuverlässig zu bearbeiten. Um lange Wartezeiten in den Partnerfilialen zu vermeiden, soll die Transaktionsverarbeitung in Echtzeit erfolgen. Durch die Implementierung automatisierter Skalierungsmechanismen soll das System bei Bedarf zusätzliche Ressourcen bereitstellen und die Last entsprechend verteilen.

→ Lösungsstrategie Q7

Einfluss Q8 → Monitoring

Die Verfügbarkeit des Systems ist durch die kontinuierliche Überwachung und die Analyse der Systemressourcen sicherzustellen. Wichtige Systemereignisse und Leistungsdaten sind durch ein Monitoringsystem zu erfassen, zu protokollieren und auszuwerten. Leistungsengpässe oder Ausfälle müssen frühzeitig erkannt und behoben werden können. Fehlerberichte und Protokolle sollen dem Support und den Administratoren über ein zentrales Dashboard zur Verfügung gestellt werden.

→ Lösungsstrategie Q8

4.1.3 Technische Faktoren

Die technischen Faktoren umfassen die Einflussfaktoren, die die technische Umsetzung und Integration des Produkts betreffen.

Einfluss T1 → Kompatibilität

Das System muss mit den Schnittstellen externer Dienstleister kompatibel sein. Ein Wechsel des Zahlungsdienstleisters darf keine Auswirkungen auf die Systemfunktionalität haben. Eine mögliche Integration in bestehende Kassensysteme muss bereits in der Architektur berücksichtigt werden. Dies erfordert die Verwendung standardisierter Schnittstellen und Protokolle zur Kommunikation mit externen Systemen. Im Kauf- und Verkaufsprozess müssen

standardisierte Barcode- und QR-Code-Formate verwendet werden, um die Kompatibilität mit den Eingabegeräten der Partnerfilialen zu gewährleisten. Der Abgleich von Kauf- und Verkaufsaufträgen zwischen der Kundenanwendung und dem Kassensystem erfolgt anhand der Auftragsnummern. Das System muss sicherstellen, dass die Auftragsnummern mit dem verwendeten Barcodeformat kompatibel sind.

→ Lösungsstrategie T1

Einfluss T2 → Systemintegration

Externe Systeme müssen nahtlos in das Softwaresystem integriert werden können. Die Sicherheit und Funktionalität des Gesamtsystems darf dadurch nicht beeinträchtigt werden. Die internen Systemkomponenten sind unabhängig voneinander zu entwickeln und von den externen Systemkomponenten zu entkoppeln. Der Zugriff auf systeminterne Funktionen und Daten darf nur über definierte Schnittstellen und Protokolle erfolgen.

→ Lösungsstrategie T2

Einfluss T3 → Änderbarkeit

Das System muss flexibel und anpassungsfähig sein, um den sich ändernden Anforderungen der Nutzer und des Marktes gerecht zu werden. Änderungen müssen schnell und effizient umgesetzt werden können, ohne den Betrieb des Systems zu beeinträchtigen. Dies erfordert einen modularen Aufbau der Software, bei dem einzelne Komponenten unabhängig voneinander angepasst werden können. Die Anpassung an aktuelle Gesetze und Vorschriften soll an zentraler Stelle erfolgen. Kleinere Regeländerungen sollen ohne Änderung des Quellcodes durch den Support oder die Administratoren vorgenommen werden können.

→ Lösungsstrategie T3

Einfluss T4 → Erweiterbarkeit

Neue Funktionen und Dienste sollen einfach integriert und bestehende Komponenten in ihrem Funktionsumfang erweitert werden können. Erweiterungen des Softwaresystems sollen ohne Beeinträchtigung der Systemfunktionalität und im laufenden Betrieb durchgeführt werden können.

→ Lösungsstrategie T4

Einfluss T5 → Internationalisierung

Eine mögliche Expansion in internationale Märkte erfordert die Unterstützung alternativer Währungen und Zahlungsmethoden sowie die Anpassung an lokale Gesetze und Vorschriften. Die Architektur soll die Integration von Partnerfilialen und Dienstleistern in verschiedenen Ländern ermöglichen. Eine Anpassung des Systems an die lokalen Gegebenheiten soll ohne großen Aufwand und ohne Auswirkungen auf das Gesamtsystem möglich sein.

→ Lösungsstrategie T5

Einfluss T6 → Systemwartung

Die Architektur muss die Wartung und Aktualisierung einzelner Systemkomponenten ohne Beeinträchtigung des Gesamtsystems ermöglichen. Wartungsarbeiten sollen in der Regel im laufenden Betrieb durchgeführt werden, um die Ausfallzeiten zu minimieren. Dies erfordert die Implementierung unabhängiger Module und Dienste, die leicht aktualisiert und ausgetauscht werden können.

→ Lösungsstrategie T6

Einfluss T7 → Rechtssicherheit

Die Plattform muss die gesetzlichen Anforderungen an den Bitcoin-Handel, den Datenschutz sowie die App-Store-Richtlinien von Apple und Google erfüllen. Insbesondere sind die Anforderungen der MiCA-Verordnung, der DSGVO und die steuerlichen Vorgaben der Finanzbehörden zu beachten. Bei Nichteinhaltung drohen drastische Konsequenzen bis hin zur Entfernung aus dem App-Store oder der Schließung des Geschäftsbetriebs [Mül18].

→ Lösungsstrategie T7

Einfluss T8 → MiCA-Verordnung

Das System muss die Anforderungen der MiCA-Verordnung erfüllen und die AML-Richtlinien umsetzen. Zur Überprüfung der Identität des Nutzers und zur Einhaltung der AML-Richtlinien muss ein KYC-Verfahren implementiert werden. Das KYC-Verfahren soll über einen externen Dienstleister abgewickelt und über eine API in das System integriert werden. Die Einhaltung der AML-Richtlinien ist im Rahmen der Zusammenarbeit mit dem Zahlungsdienstleister sicherzustellen. Verdächtige Aktivitäten müssen erkannt und zwischen den Systemen synchronisiert werden. Relevante Daten müssen bei AML-Verdachtsfällen protokolliert und an die zuständigen Behörden gemeldet werden.

→ Lösungsstrategie T8

Einfluss T9 → Datenschutz

Das System muss die datenschutzrechtlichen Anforderungen der DSGVO erfüllen (vgl. Regulatorik & Datenschutz). Die Daten der Nutzer müssen DSGVO-konform gespeichert und verarbeitet werden und die Rechte der Nutzer auf Auskunft, Berichtigung und Löschung ihrer Daten müssen gewährleistet sein [Bun17]. Die Datenübertragung zwischen der Kundenanwendung, dem Kassensystem und den externen Dienstleistern muss verschlüsselt erfolgen und die Daten müssen vor unberechtigtem Zugriff geschützt sein.

→ Lösungsstrategie T9

Einfluss T10 → Steuerrecht

Die steuerlichen Regelungen und Meldepflichten für den Handel mit Bitcoins müssen von den Nutzern und der Plattform eingehalten werden. Dies erfordert die Erfassung und Speicherung von Transaktionsdaten sowie die Erstellung von Transaktionsberichten und eine Exportfunktion für die Finanzbehörden. Das System muss sicherstellen, dass alle steuerlich relevanten Daten korrekt erfasst, zu Nachweiszwecken gespeichert und innerhalb der gesetzlichen Fristen an die Finanzbehörden übermittelt werden. Die Berichte müssen den gesetzlichen Anforderungen der Finanzbehörden entsprechen und in einem standardisierten Format zur Verfügung gestellt werden. Eine direkte Anbindung an einen Steuerberater oder eine externe Steuersoftware kann in Zukunft relevant werden und sollte daher bereits in der Architektur vorgesehen werden.

→ Lösungsstrategie T10

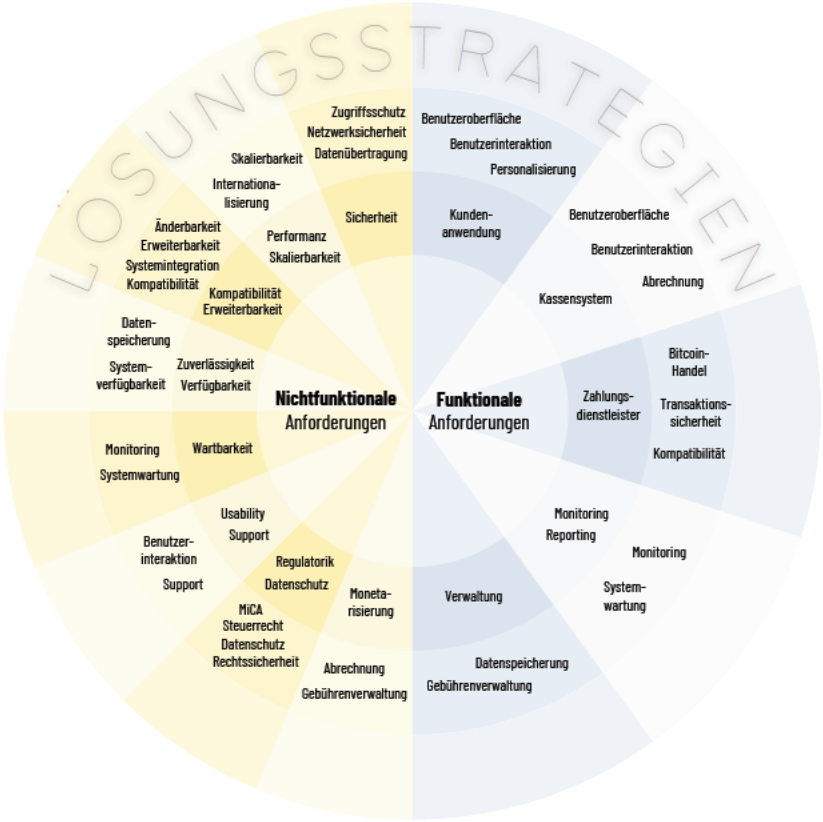

Abbildung 4.1: Strukturierter Entwicklungsprozess – Einflussfaktoren

4.2 Lösungsstrategien

Im Folgenden werden für die identifizierten Einflussfaktoren (vgl. Strukturdiagramm Abb. 4.1) konkrete Architekturlösungen entwickelt und in Lösungsstrategien zusammengefasst. Die Lösungsstrategien werden entsprechend der Einflussfaktoren in die Kategorien Produktfaktoren, Qualitätsfaktoren und technische Faktoren unterteilt.

4.2.1 Produktfaktoren

Strategie für Einfluss P1 → Benutzeroberfläche

Die Architektur wird nach dem SoC-Prinzip durch die Einführung einer Schichtenarchitektur entworfen. Es wird eine Drei-Schichten-Architektur entwickelt, die das Softwaresystem in eine Präsentationsschicht, eine Anwendungsschicht und eine Persistenzschicht unterteilt. Die Präsentationsschicht kapselt die Benutzeroberfläche der Kundenanwendung, der Kassenanwendung und der Verwaltungsoberfläche des Systems. Änderungen an der Benutzeroberfläche können unabhängig von der Geschäftslogik vorgenommen werden und haben keine Auswirkungen auf die anderen Schichten. Diese Trennung verbessert die Wartbarkeit und Erweiterbarkeit des Systems.

Die Benutzeroberfläche der Kundenanwendung und der Kassenanwendung wird mit dem *React Native* Framework entwickelt. React Native ermöglicht die Erstellung plattformübergreifender Anwendungen für die Betriebssysteme iOS und Android [Met22]. Die Entwicklung erfolgt für beide Plattformen in einer gemeinsamen Codebasis in der Programmiersprache *TypeScript*. TypeScript erweitert JavaScript um ein Modulsystem, Klassen, Interfaces und ein statisches Typsystem [BAT14]. Typbedingte Fehler können bereits zur Entwicklungszeit erkannt, die Codequalität erhöht und die Wartbarkeit verbessert werden. Die Entwicklung in React Native ermöglicht ein einheitliches Design und eine konsistente Benutzererfahrung auf beiden Plattformen.

Die Verwaltungsoberfläche des Systems wird als Webanwendung entwickelt und mit dem *React* Framework umgesetzt. React ermöglicht die effiziente Entwicklung interaktiver Webanwendungen auf Basis von Komponenten, die von unabhängigen Entwicklern erstellt und nahtlos integriert werden können [Met24]. Die Entwicklung erfolgt ebenfalls in der Programmiersprache TypeScript.

Diese Arbeit konzentriert sich auf die Entwicklung der serverseitigen Anwendung, die aus der Anwendungsschicht und der Persistenzschicht besteht.

→ Einflussfaktor P1

Strategie für Einfluss P2 → Benutzerinteraktion

Die Entwicklung der serverseitigen Anwendung erfolgt in der Programmiersprache TypeScript für die Laufzeitumgebung *Node.js*. Node.js ermöglicht eine effiziente Entwicklung von serverseitigen Anwendungen und bietet über den Paketmanager npm eine Vielzahl von Mo-

dulen und Bibliotheken [Iuk23]. Die Entwicklung des Softwaresystems erfolgt somit in einer einheitlichen Programmiersprache. Dies erleichtert die Wartung und Weiterentwicklung des Systems und ermöglicht eine effiziente Zusammenarbeit der Entwickler. Die Kommunikation zwischen der client- und der serverseitigen Anwendung erfolgt über GraphQL-Schnittstellen. GraphQL ermöglicht es, Schnittstellen für eine präzise und effiziente Datenabfrage zu definieren und diese flexibel zu erweitern (vgl. Webservices). Im Vergleich zu REST bietet GraphQL eine präzisere Datenabfrage und eine bessere Syntax, was die Implementierung von API-Abfragen erleichtert [BV20].

> [...] GraphQL requires less effort to implement API queries, when compared with REST. Also, interestingly, experts in REST APIs can also write GraphQL queries with less effort [BV20].

Die Anwendungsschicht – der Geschäftslogik-Teil der serverseitigen Anwendung – ist für die Verarbeitung von Benutzeranfragen und die Koordination der Systemkomponenten zuständig. Die Anwendungsschicht wird nach dem Geheimnisprinzip durch die Umsetzung einer modularen Architektur entworfen. Dabei wird die Geschäftslogik in unabhängige Module aufgeteilt, von denen jedes eine bestimmte Aufgabe erfüllt. Dies ermöglicht eine einfache Wartung und Erweiterung des Systems, da jede Komponente isoliert entwickelt und getestet werden kann.

Wichtige Systemereignisse wie Status- und Fehlermeldungen werden von einem *Logging-Modul* verarbeitet und dem Nutzer in leicht verständlicher Form angezeigt. Gemäß den Best Practices für das Logging in Node.js-Anwendungen werden die Ereignisse in einem strukturierten Format mit einer aussagekräftigen Beschreibung und einem eindeutigen Fehlercode protokolliert [Isa23]. Ein *Benachrichtigungs-Modul* informiert die Nutzer per E-Mail oder Push-Benachrichtigung über wichtige Ereignisse und den Status ihrer Transaktionen. Der E-Mail-Versand erfolgt über die API-Schnittstelle eines externen E-Mail-Dienstes wie *SendGrid* oder *Mailchimp*. Diese Dienste bieten eine hohe Zustellrate und eine einfache Integration in Node.js-Anwendungen [Sen24; Mai24]. Für den Versand von E-Mails zur Registrierungsbestätigung wird das Benachrichtigungs-Modul an das Authentifizierungs-Modul (vgl. Lösungsstrategie Q3) angebunden. Das Benachrichtigungs-Modul wird mit einem externen Push-Benachrichtigungsdienst verbunden, um Push-Benachrichtigungen an die Kundenanwendung zu senden. *Firebase Cloud Messaging* und *OneSignal* bieten jeweils eine API-Schnittstelle zur Integration von Push-Benachrichtigungen in Node.js-Anwendungen [Goo24c; One24].

→ Einflussfaktor P2

Strategie für Einfluss P3 → Personalisierung

Die Kundenanwendung und die Kassenanwendung können an die individuellen Bedürfnisse der Nutzer angepasst werden. Sprache, Design und Währungspräferenzen können in den Benutzereinstellungen konfiguriert werden.

Die Spracheinstellungen werden über das Framework *i18next* realisiert. Das Framework unterstützt die Integration von Übersetzungen in Node.js-, React- und React-Native-Anwendungen [i1824a]. Es erkennt die Spracheinstellungen des Nutzers und lädt die entsprechenden Übersetzungen zur Laufzeit [i1824b]. Die Übersetzungen können in mehreren Dateien organisiert und im JSON-Format gespeichert werden. Die Benutzeroberfläche wird zunächst zweisprachig in Deutsch und Englisch entwickelt. Durch einfaches Hinzufügen von Übersetzungsdateien können weitere Sprachen hinzugefügt werden.

Die Persistenzschicht ist für die dauerhafte Speicherung der Nutzerdaten zuständig. Die Speicherung erfolgt in einer zentralen Datenbank (vgl. Lösungsstrategie Q4). Die Benutzereinstellungen werden einem Benutzerprofil zugeordnet und mit dem lokalen Gerätespeicher des Nutzers synchronisiert. Der Datenaustausch zwischen den clientseitigen Anwendungen und der Datenbank erfolgt über eine API-Schnittstelle, die von der Anwendungsschicht bereitgestellt wird.

→ Einflussfaktor P3

Strategie für Einfluss P4 → Bitcoin-Handel

Es wird ein externer Zahlungsdienstleister eingebunden, der den Kauf und Verkauf von Bitcoins abwickelt und die Bitcoins für den Nutzer sicher verwahrt. Der Anbieter muss die gesetzlichen Anforderungen für den Handel mit Bitcoins in Deutschland erfüllen (vgl. Regulatorik & Datenschutz) und eine kompatible API zur Anbindung an das System bereitstellen.

Bitpanda verfügt über eine *Kryptoverwahrlizenz* und bietet eine Komplettlösung für den Handel und die Verwahrung von Bitcoins [Sch22; Bit22a]. *Bitpanda Custody* bietet mit *TrustVault* eine Kryptoverwahrungslösung für institutionelle Kunden und Unternehmen an [Bit22a]. Die *TrustVault API* ist eine GraphQL-basierte API zur Interaktion mit der *TrustVault*-Plattform. Die API ermöglicht das Erstellen von Bitcoin-Adressen, das Senden und Empfangen von Bitcoins und das Abrufen der Transaktionshistorie. Die Einhaltung der AML-Richtlinien wird durch Bitpanda Custody sichergestellt (vgl. Lösungsstrategie T8) [Bit22b].

Ein *Transaktionsverarbeitungs-Modul* koordiniert die Auftragsabwicklung zwischen der Kundenanwendung, dem Kassensystem und dem Zahlungsdienstleister und verwaltet den Status der Transaktionen in der Datenbank. Das Transaktionsverarbeitungs-Modul realisiert die Verarbeitung von Kauf- und Verkaufsaufträgen über die TrustVault API. Die TrustVault API ist als npm-Bibliothek für Node.js verfügbar [Bit24a]. Zur Fehlerbehandlung wird ein Fallback-Mechanismus implementiert, der alternative API-Endpunkte des Zahlungsdienstleisters oder – falls verfügbar – alternative Zahlungsdienstleister einsetzt. Dazu werden die API-Endpunkte des Zahlungsdienstleisters in einer Konfigurationsdatei hinterlegt und im Fehlerfall automatisch auf die alternativen Endpunkte umgeleitet. Die Kundenanwendung, das Kassensystem und die Datenbank werden jeweils über GraphQL-Schnittstellen mit dem Transaktionsverarbeitungs-Modul verbunden. Das *Transaktionsvalidierungs-Modul* überprüft die Transaktionen und erkennt verdächtige Aktivitäten und Betrugsversuche (vgl. Lösungsstrategie Q1). Das *Monitoring-Modul* überwacht die Verfügbarkeit und Performanz der API-Endpunkte des Zahlungsdienstleisters und erkennt Ausfälle oder Verzögerungen (vgl. Lösungsstrategie Q8).

→ Einflussfaktor P4

Strategie für Einfluss P5 → Abrechnung

Abrechnungsrelevante Daten wie Zeitstempel, Betrag, Partnerfiliale und Zahlungsdienstleister zu den Transaktionen werden in einer zentralen Datenbank gespeichert (vgl. Lösungsstrategie Q4). Die Erfassung der Finanzdaten erfolgt durch ein *Abrechnungs-Modul*, das über definierte Schnittstellen in der Persistenzschicht auf die Datenbank zugreift. Das Abrechnungs-Modul aggregiert die Transaktionsdaten, ordnet sie den Partnerfilialen und externen Dienstleistern zu und erstellt die Abrechnungen für die erbrachten Leistungen. Ein Dashboard zeigt die wichtigsten Finanzkennzahlen an und ermöglicht den Export der Daten im CSV-Format. Eine direkte Anbindung an eine externe Buchhaltungssoftware wird zukünftig über eine API-Schnittstelle im Abrechnungs-Modul realisiert Das Benachrichtigungs-Modul informiert die Partnerfilialen und Dienstleister per E-Mail über den Status ihrer Abrechnungen und wichtige Finanzereignisse (vgl. Lösungsstrategie A2).

→ Einflussfaktor P5

Strategie für Einfluss P6 → Gebührenverwaltung

Die Gebührenverwaltung umfasst die Definition und Verwaltung individueller Gebührensätze für den Kauf und Verkauf sowie das Senden und Empfangen von Bitcoins. Die Gebührenverwaltung wird über das *Konfigurations-Modul* realisiert und in die zentrale Verwaltungsoberfläche integriert, so dass die Konfiguration der Gebührensätze an zentraler Stelle erfolgt (vgl. Lösungsstrategie T3).

Die Berechnung der Gebühren für den Bitcoin-Handel erfolgt auf Basis der konfigurierten Gebührensätze, des Transaktionsbetrages und des aktuellen Bitcoin-Wechselkurses. Der aktuelle Wechselkurs wird über eine externe API abgefragt und in Echtzeit aktualisiert. Die Gebührensätze werden dem Nutzer in der Kundenanwendung vor der Bestätigung der Transaktion angezeigt. Sobald die Transaktion bestätigt wurde, werden die Gebühren berechnet und die Transaktion samt Gebühren in der Datenbank gespeichert.

→ Einflussfaktor P6

Strategie für Einfluss P7 → Support

Ein zentrales *Support-Modul* ermöglicht die Bearbeitung von Supportanfragen und die Kommunikation zwischen den Nutzern und dem Support-Team. Das Support-Modul wird auf Basis eines Ticketsystems entwickelt, das eine strukturierte Erfassung, Priorisierung und Bearbeitung von Anfragen ermöglicht und die Effizienz des Support-Teams steigert. Für das Support-Modul wird ein Dashboard entwickelt, das die Bearbeitung von Supportanfragen vereinfacht und wichtige Systemereignisse in Echtzeit anzeigt. Zur Problemverfolgung können Status- und Fehlermeldungen über das Logging-Modul (vgl. Lösungsstrategie P2) abgerufen und im Dashboard angezeigt werden. Dazu wird das Logging-Modul um eine Funktion zur Abfrage von Fehlerberichten und Protokollen für Support-Zwecke erweitert.

Eine E-Mail-Benachrichtigung für den Support ist über das Benachrichtigungs-Modul realisierbar (vgl. Lösungsstrategie A2). Um die Kommunikation zwischen den Nutzern und dem Support-Team zu optimieren, könnte ein integriertes Kommunikationssystem mit Live-Chat eingeführt werden. Eine Live-Chat-Funktion könnte mit Hilfe von *Firebase Realtime Database* realisiert werden. Firebase Realtime Database ist eine nicht-relationale Datenbank von Google, die eine Datensynchronisation mit Web- und Mobilanwendungen in Echtzeit ermöglicht [Goo24d].

Um die Nutzer bei wiederkehrenden Problemen zu unterstützen, wird eine Wissensdatenbank mit häufig gestellten Fragen und Lösungen entwickelt. Chatbots und KI-gestützte Systeme können die Nutzer bei der Problemlösung unterstützen und die Antwortzeiten des Support-Teams verkürzen. Diese Maßnahmen reduzieren die Anzahl der Supportanfragen und ermöglichen es den Nutzern, Probleme schnell und einfach zu lösen.

→ Einflussfaktor P7

4.2.2 Qualitätsfaktoren

Strategie für Einfluss Q1 → Transaktionssicherheit

Für die Transaktionsverarbeitung und die Transaktionsvalidierung werden zwei unabhängige Module entwickelt. Das Transaktionsverarbeitungs-Modul koordiniert den Kauf- und Verkaufsprozess zwischen der Kundenanwendung, dem Kassensystem und dem Zahlungsdienstleister (vgl. Lösungsstrategie P4). Ein *Transaktionsvalidierungs-Modul* identifiziert verdächtige Transaktionen und ungewöhnliche Aktivitäten. Mit Hilfe von *Apache Kafka* können Transaktionsdaten in Echtzeit verarbeitet und analysiert werden (vgl. Lösungsstrategie Q7). Die Daten werden auf Integrität geprüft und eine Betrugserkennung anhand bekannter Betrugsmuster durchgeführt.

Eine *Rules Engine* bietet die Möglichkeit, vordefinierte Regeln zur Mustererkennung zu definieren. Mit der *json-rules-engine* steht eine leistungsfähige und leichtgewichtige Rules Engine für Node.js-Anwendungen zur Verfügung [cac23]. Die Regeln werden über Bedingungen und Ereignisse im JSON-Format definiert und können leicht angepasst und erweitert werden. Die Bedingungen definieren Anforderungen an Transaktionen, die erfüllt sein müssen, um ein bestimmtes Ereignis auszulösen. Eine effiziente Regelausführung wird durch eine priorisierte Ausführung der Regeln und den Einsatz von Caching-Techniken erreicht [cac23].

→ Einflussfaktor Q1

Strategie für Einfluss Q2 → Netzwerksicherheit

Das System wird mit der Orchestrierungslösung K8s betrieben, um die Verfügbarkeit und Skalierbarkeit der Anwendung zu gewährleisten (vgl. Lösungsstrategie Q6). K8s basiert auf einer Cloud-nativen Architektur und setzt die Sicherheitsempfehlungen des CNCF um [Kub24e]. Die Konfiguration von K8s sollte sich an den Best Practices für die Sicherheit in Cloud-nativen

Umgebungen orientieren und regelmäßig überprüft und aktualisiert werden. Angreifer könnten über Schwachstellen in der K8s-Konfiguration in das System eindringen und die Kontrolle über einzelne Container erlangen. Sobald ein Container kompromittiert ist, könnte der Angreifer versuchen, seine Privilegien auszuweiten und die Kontrolle über weitere Container oder den gesamten Cluster zu übernehmen.

Aufgrund der Komplexität und der verteilten Architektur von K8s sind verschiedene Angriffsvektoren zu berücksichtigen. Zur Bewertung der Sicherheit von K8s-Clustern wird das *STRIDE*-Modell verwendet. Das STRIDE-Modell unterscheidet sechs Kategorien von Angriffsvektoren: *Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service* und *Elevation of Privilege* [Pou20].

Spoofing bezeichnet das Vortäuschen einer falschen Identität, um sich unberechtigten Zugang zum System zu verschaffen. Zum Schutz vor Spoofing-Angriffen wird in K8s eine gegenseitige *TLS*-Authentifizierung (*mTLS*) zwischen den Komponenten eingesetzt [Pou20]. Zertifikate und Schlüssel müssen dabei sicher verwaltet und regelmäßig erneuert werden. Für die Erstellung von *TLS-Zertifikaten* bietet K8s ein integriertes Zertifikatsmanagement (vgl. Lösungsstrategie Q5).
Tampering bezeichnet die böswillige Veränderung von Daten oder Systemkomponenten wie Konfigurationsdateien oder Container Images. K8s adressiert dieses Risiko durch die Verwendung von TLS für die Kommunikation zwischen den Komponenten und durch die Verwendung von schreibgeschützten Dateisystemen für die Container [Pou20].
Repudiation bezeichnet das Leugnen einer Aktion oder eines Ereignisses durch den Angreifer. Zur Überwachung der K8s-Umgebung wird *K8s Auditing* eingesetzt. Durch die Analyse der *Audit-Protokolle* können ungewöhnliche Aktivitäten nachverfolgt und Repudiation-Angriffen vorgebeugt werden (vgl. Lösungsstrategie Q8). Voraussetzung ist, dass die Audit-Protokolle sicher gespeichert und vor unbefugtem Zugriff geschützt werden.
Information Disclosure bezeichnet die unbeabsichtigte Offenlegung sensibler Daten wie Konfigurationsdateien oder Daten aus dem Cluster Store. Sensible Daten wie Passwörter und Zugriffstoken sollten nicht im Anwendungscode gespeichert werden. Die *Secret*-API von K8s ermöglicht die sichere Speicherung sensibler Daten (vgl. Lösungsstrategie Q3) [Kub24f].
DoS-Angriffe zielen darauf ab, die Verfügbarkeit der Plattform zu beeinträchtigen (vgl. Zuverlässigkeit) und etwa den API-Server oder den Cluster Store zu überlasten. Der K8s-Cluster sollte in einer Hochverfügbarkeitskonfiguration (*HA*) betrieben werden und über ein Load-Balancing-System verfügen (vgl. Lösungsstrategie Q6). Ressourcenbeschränkungen für Speicher und CPU können die Auswirkungen von DoS-Angriffen minimieren [Pou20].
Elevation of Privilege bezeichnet die unberechtigte Erhöhung von Zugriffsrechten durch einen Angreifer. Eine rollenbasierte Zugriffskontrolle ermöglicht die Definition von Zugriffsrichtlinien und die Einschränkung von Zugriffsrechten (vgl. Lösungsstrategie Q3). Die Isolierung der Pods und Container im K8s-Cluster erfolgt gemäß den Best Practices nach den Empfehlungen der CNCF (*Pod Security Standards*) [Kub24g]. Es stehen drei Profile mit unterschiedlichen Einschränkungen zur Verfügung, um sicherzustellen, dass die Pods gegen bekannte Privilegieneskalationen geschützt sind [Kub24g]. Das erste Profil – *Privileged* – bietet uneingeschränkte Berechtigungen für die Verwaltung kritischer Systemkomponenten. Das zweite Profil – *Baseline* – bietet eine Standardkonfiguration zur Vermeidung von Privile-

gieneskalationen. Das dritte Profil – *Restricted* – entspricht den aktuellen Best Practices für eine strikte Isolierung von Pods und Containern. Nutzer und Dienste sollten nur die minimal erforderlichen Rechte erhalten (*Least Privilege Principle*) [Pou20].

Der Netzwerkverkehr zwischen den Pods und dem öffentlichen Netzwerk wird über die *Network Policies* gesteuert. Die Network Policies ermöglichen die Definition von Regeln für den eingehenden und ausgehenden Netzwerkverkehr auf IP-Adress- und Port-Ebene (OSI-Schichten 3 und 4) [Kub24h].

Für die sichere Konfiguration der K8s-Umgebung wird die aktuelle *Security Checklist* von K8s (Stand: 15.02.2024) verwendet [Kub24i]. Die Checkliste gibt Hinweise zur sicheren Authentifizierung und Autorisierung, zur Netzwerk- und Pod-Sicherheit, zur Überwachung und Protokollierung sowie zur allgemeinen Sicherheit des K8s-Clusters.

→ Einflussfaktor Q2

Strategie für Einfluss Q3 → Zugriffsschutz

Die Benutzerauthentifizierung für die Kundenanwendung, das Kassensystem und die Verwaltungsoberfläche wird über ein zentrales *Authentifizierungs-Modul* realisiert, das den *OAuth 2.0*-Standard integriert. OAuth 2.0 ist der Industriestandard für die Authentifizierung in Web- und Mobilanwendungen [Par24]. Es bietet einen standardisierten Authentifizierungsmechanismus für die sichere Anmeldung von Nutzern und Diensten.

Die Autorisierung der Nutzer erfolgt über ein *Autorisierungs-Modul*, das eine rollenbasierte Zugriffskontrolle (*RBAC*) implementiert. Das Autorisierungs-Modul verwaltet die Zugriffsrechte der Nutzer und definiert Zugriffsrichtlinien für einzelne Benutzergruppen. Das Autorisierungs-Modul wird auf Basis des *Open Policy Agent* (*OPA*) realisiert. OPA ist ein Open-Source-Projekt, das eine deklarative Sprache zur Definition von Zugriffsrichtlinien in Cloud-nativen Umgebungen bietet [Ope24a]. OPA verwendet *Admission Controllers*, um Zugriffsrichtlinien in K8s-Clustern durchzusetzen [Ope24b]. Admission Controllers prüfen Anfragen an die K8s-API und erlauben oder verweigern diese auf Basis der definierten Richtlinien. Für eine einfache Integration in K8s-Anwendungen wird standardmäßig *OPA Gatekeeper* verwendet [Ope24b]. OPA Gatekeeper ist ein Cloud-natives Zugriffskontrollsystem, das OPA in K8s integriert und definierte Zugriffsrichtlinien durchsetzt [Ope24b].

Sensible Daten wie Passwörter und Zugriffstoken werden in den *Secrets* von K8s abgelegt und über die *Secret*-API verwaltet [Kub24f]. Eine zusätzliche Verschlüsselung der Secrets und die sichere Ablage der verwendeten Schlüssel – vorzugsweise in einem cloudbasierten Schlüsselverwaltungsdienst (*KMS*) – erhöhen die Sicherheit der sensiblen Daten [Pou20].

→ Einflussfaktor Q3

Strategie für Einfluss Q4 → Datenspeicherung

Die Persistenzschicht – die Datenzugriffsschicht der serverseitigen Anwendung – ist für die dauerhafte Speicherung der Anwendungsdaten (Nutzerdaten, Transaktionsdaten und Konfigurationsdaten) verantwortlich und stellt Methoden zum Lesen, Schreiben und Löschen von Daten zur Verfügung. Für die Datenspeicherung wird die relationale Datenbank MySQL verwendet. MySQL unterstützt bei korrekter Konfiguration die ACID-Eigenschaften und bietet damit eine hohe Datenkonsistenz (vgl. Relationale Datenbanken). MySQL ist optimal für den Betrieb in einer K8s-Umgebung geeignet, da es eine leistungsstarke Plattform bietet, die sich nahtlos in containerisierte Umgebungen integrieren lässt [Arn23].

Es wird eine verteilte Datenbanklösung mit MySQL unter Verwendung von NDB Cluster implementiert. Mit NDB Cluster steht eine hochverfügbare und ausfallsichere Datenbanklösung mit synchroner Datenreplikation zur Verfügung. NDB Cluster zeichnet sich durch eine geringe Latenz und eine hohe Skalierbarkeit aus und ermöglicht die Bereitstellung eines hochverfügbaren und hochperformanten MySQL-Clusters in einer containerisierten Umgebung (vgl. NDB Cluster).

Die Datenbankzugriffe erfolgen über ein *Datenbank-Modul*, das die Interaktion mit der Datenbank abstrahiert und die Datenbankoperationen in objektorientierten Methoden kapselt. Das Datenbank-Modul wird auf Basis eines ORM-Frameworks entwickelt, das die Datenbankinteraktion in objektorientierten Anwendungen vereinfacht und die Datenkonsistenz sicherstellt (vgl. Object-Relational Mapping). Dies erleichtert die Entwicklung und Wartung der Datenbankzugriffe und schützt die Anwendung vor *SQL-Injection*-Angriffen. Zwei beliebte ORM-Frameworks für Node.js-Anwendungen sind *Sequelize* und *TypeORM* [npm24].

→ Einflussfaktor Q4

Strategie für Einfluss Q5 → Datenübertragung

K8s verwendet *TLS* zur sicheren Kommunikation zwischen den Komponenten und bietet ein integriertes Zertifikatsmanagement [Kub24f]. Die TLS-Verschlüsselung stellt sicher, dass die Daten während der Übertragung nicht von Dritten abgefangen oder manipuliert werden können. Das Zertifikatsmanagement in K8s ermöglicht die Erstellung, Verwaltung und Erneuerung von Zertifikaten für die sichere Kommunikation zwischen den Komponenten. Für die Erstellung von TLS-Zertifikaten stellt K8s eine *Certificates*-API (*certificates.k8s.io*) zur Verfügung [Kub23b].

Die Datenübertragung zwischen der Kundenanwendung, dem Kassensystem und der serverseitigen Anwendung erfolgt ebenfalls verschlüsselt über *HTTPS* (*HTTP over TLS*).

Die Überwachung der Netzwerkverbindungen und die Analyse des Datenverkehrs erfolgt über das *Monitoring-Modul* (vgl. Lösungsstrategie Q8).

→ Einflussfaktor Q5

Strategie für Einfluss Q6 → Systemverfügbarkeit

Eine SOA ermöglicht die Bewältigung hoher Datenlasten, indem einzelne Services bei Bedarf horizontal skaliert werden können. Die Steuerung der Services erfolgt jedoch über ein zentrales System und kann zu hoher Komplexität und einer monolithischen Struktur führen (vgl. Serviceorientierte Architektur). Die Microservice-Architektur ist eine Weiterentwicklung der SOA und folgt dem Prinzip der Dezentralität und Autonomie der Services (vgl. Microservices). Laut einer weltweiten Umfrage im Jahr 2023 wird die Microservice-Architektur von der Mehrheit der Entwickler bei der Softwareentwicklung bevorzugt [Sta24].

> In 2023, microservices were the primary approach used in system design, as reported by 82 percent of respondents worldwide, beating all other approaches by a wide margin [Sta24].

Die Microservices werden in Docker-Containern bereitgestellt und mit K8s orchestriert. Im Vergleich zur Hypervisor-basierten Virtualisierung ermöglicht die Containerisierung mit Docker eine effizientere Ressourcennutzung und eine schnellere Bereitstellung von Anwendungen [Pot+20].

> Performance evaluation is carried out on virtual machine and Docker container-based hosts in terms of CPU Performance, Memory throughput, Disk I/O, Load test, and operation speed measurement. It is observed that Docker containers perform better over VM in every test, as the presence of QEMU layer in the virtual machine makes it less efficient than Docker containers. [Pot+20]

Im Vergleich zu Docker Swarm bietet K8s eine umfassendere Lösung für die Orchestrierung von Containern und eignet sich besser für komplexe Softwareprojekte [Pan+19].

> Docker Swarm [...] functionality is limited by what is available in the Docker API, and hence it only provides limited fault tolerance capabilities. Kubernetes comes with several years of expert experience in production deployment environments and hence provides comprehensive features and support. For larger deployments, Kubernetes' auto-scaling features make maximum use of resources, and the overheads of running Kubernetes are reduced. [Pan+19]

Docker und K8s haben sich als Standardlösungen für die Bereitstellung von Cloud-nativen Anwendungen etabliert [Pot+20; Vai23]. Laut einer weltweiten Umfrage im Jahr 2022 haben bereits mehr als die Hälfte der Unternehmen K8s in ihre Systeme integriert [Vai23]. Die Entwicklung einer Microservice-Architektur auf Basis von Docker und K8s gewährleistet eine hohe Systemverfügbarkeit, Skalierbarkeit und Ausfallsicherheit (vgl. Containerisierung, Kubernetes).

Ein Load Balancer sorgt für eine effiziente Verteilung der Anfragen auf die verfügbaren Ressourcen. Dadurch werden Performanceprobleme vermieden und die Systemverfügbarkeit erhöht. Für die Lastverteilung in K8s-Clustern wird der NGINX Ingress Controller eingesetzt.

Der NGINX Ingress Controller wird als Pod innerhalb des Clusters bereitgestellt und fungiert als zentraler Einstiegspunkt für den eingehenden Datenverkehr (vgl. Lastverteilung). Diese Architektur ermöglicht eine dynamische Skalierung der Systemressourcen und bietet einen wirksamen Schutz gegen Angriffe und Ausfälle. Im Falle eines DoS-Angriffs kann K8s automatisch die betroffenen Services isolieren, zusätzliche Instanzen bereitstellen und die Last auf mehrere Server verteilen.

Für eine hohe Datenverfügbarkeit wird eine verteilte Datenbanklösung mit MySQL eingesetzt (vgl. Lösungsstrategie Q4).

→ Einflussfaktor Q6

Strategie für Einfluss Q7 → Skalierbarkeit

Eine hohe Skalierbarkeit des Systems wird durch die Implementierung einer ereignisgesteuerten Microservice-Architektur erreicht. Eine ereignisgesteuerte Architektur (EDA) ermöglicht die Entkopplung der einzelnen Services und die asynchrone Kommunikation zwischen den Komponenten. Dies ermöglicht eine dynamische Skalierung der Systemressourcen und die Verarbeitung von Transaktionen in Echtzeit (vgl. Entwurfsmuster).

Die EDA wird durch den Einsatz von *Apache Kafka* realisiert. Apache Kafka bietet eine leistungsstarke und skalierbare Lösung für die Echtzeitverarbeitung von Transaktionsdaten in Cloud-nativen Anwendungen [Red22]. Apache Kafka ist eine Open-Source-Plattform für die Verarbeitung und Analyse großer Datenmengen in Echtzeit und ermöglicht die asynchrone Kommunikation zwischen den Microservices in einer ereignisgesteuerten Architektur [Red22].

Der K8s-Cluster wird in einer Cloud-Umgebung wie GCP, AWS oder Microsoft Azure betrieben. Diese Cloud-Anbieter bieten eine skalierbare Infrastruktur und eine hohe Verfügbarkeit für die Bereitstellung von Cloud-nativen Anwendungen. Sie stellen hochverfügbare virtuelle Maschinen, Speicherlösungen und Netzwerkdienste bereit und ermöglichen eine einfache Integration mit K8s (vgl. Kubernetes).

→ Einflussfaktor Q7

Strategie für Einfluss Q8 → Monitoring

Ein *Monitoring-Modul* ist für die kontinuierliche Überwachung und Analyse von Systemressourcen, Systemereignissen und Netzwerkaktivitäten verantwortlich. Das Monitoring-Modul wird als eigenständiger Microservice implementiert. Die Überwachung innerhalb des K8s-Clusters erfolgt durch *K8s Auditing* [Pou20]. K8s Auditing liefert eine detaillierte chronologische Aufzeichnung aller Aktivitäten wie Benutzeraktivitäten, API-Aufrufe und Änderungen des Clusterzustands [Kub23c]. Anhand der Audit-Protokolle (*audit logs*) können ungewöhnliche Aktivitäten erkannt und Konfigurationsänderungen nachverfolgt werden [Cho24].

Das Monitoring-Modul überwacht den Ressourcenverbrauch, die Anwendungsleistung und Sicherheitsereignisse im K8s-Cluster anhand von Metriken. Die Erfassung und Analyse der Metriken liefert Informationen über die Leistung und den Zustand des Clusters. Für die Erfassung der Metriken in K8s wird Prometheus eingesetzt. Prometheus erkennt automatisch neue Dienste und Endpunkte und ermöglicht detaillierte PromQL-Abfragen zur Analyse der Metriken (vgl. Monitoring). Prometheus *Alertmanager* unterstützt bei der Verwaltung der von Prometheus generierten Alarmierungsdaten und ermöglicht die Definition von Benachrichtigungsregeln [Pro24]. Basierend auf den gesammelten Metriken und definierten Regeln (Schwellwerte, Ereignisse, etc.) werden Alarme ausgelöst und Benachrichtigungen versendet. Der Alertmanager ermöglicht die Definition komplexer Benachrichtigungsregeln, um sicherzustellen, dass die richtigen Personen zur richtigen Zeit informiert werden. Für die Visualisierung der Metriken über Dashboards wird Grafana verwendet. Grafana ermöglicht die Erstellung interaktiver Dashboards zur Visualisierung der gesammelten Metriken und Trends (vgl. Monitoring).

Das Monitoring-Modul wird in die zentrale Verwaltungsoberfläche (vgl. Lösungsstrategie P1) integriert. So erhalten die Administratoren und das Support-Team einen umfassenden Überblick über den Zustand des Systems und können bei Bedarf schnell auf Probleme reagieren.

→ Einflussfaktor Q8

4.2.3 Technische Faktoren

Strategie für Einfluss T1 → Kompatibilität

Das Transaktionsverarbeitungs-Modul koordiniert den Kauf- und Verkaufsprozess zwischen der Kundenanwendung, dem Kassensystem und dem Zahlungsdienstleister (vgl. Lösungsstrategie P4). Dabei muss die Kompatibilität zur API des Zahlungsdienstleisters und zum Kassensystem gewährleistet sein. Die API-Endpunkte des Zahlungsdienstleisters werden in einer Konfigurationsdatei hinterlegt und können bei Bedarf angepasst werden (vgl. Lösungsstrategie P4). Das Transaktionsverarbeitungs-Modul muss die Kompatibilität zum Kassensystem sicherstellen und die Kommunikation über eine standardisierte Schnittstelle ermöglichen. Die Kassenanwendung wird über eine GraphQL-Schnittstelle mit dem Transaktionsverarbeitungs-Modul verbunden. Die Schnittstelle ist so zu gestalten, dass die Kassenanwendung die erforderlichen Daten abrufen und den Auftragsstatus aktualisieren kann. Die Implementierung der Schnittstelle erfolgt auf Basis des *Adaptermusters* (*Adapter Pattern*). Das Adaptermuster ermöglicht die Anpassung inkompatibler Schnittstellen, ohne deren Funktionalität zu beeinträchtigen. Dadurch kann die Kompatibilität bei der Integration in bestehende Kassensysteme sichergestellt werden.

Ein *Barcode-Modul* dient der Erstellung und Verarbeitung von Barcodes, die mit den Eingabegeräten der Partnerfilialen kompatibel sind. Das Barcode-Modul kodiert die in der Datenbank generierten Auftragsnummern, die als Referenz für Kauf- und Verkaufsaufträge dienen, in ein kompatibles Barcode-Format. Ein Barcodegenerator, der die gängigen Barcode-Formate

unterstützt, steht als npm-Bibliothek für React Native zur Verfügung [kic21]. Der Barcodegenerator erhält die kodierte Auftragsnummer als Eingabeparameter und generiert den entsprechenden Barcode. Die Kassenanwendung nutzt die Kamerafunktion des mobilen Geräts, um den Barcode zu scannen. Das Barcode-Modul dekodiert den gescannten Barcode und extrahiert die Auftragsnummer entsprechend dem gewählten Barcodeformat. Das Barcode-Modul stellt die Kompatibilität bei der Integration in bestehende Kassensysteme sicher.

→ Einflussfaktor T1

Strategie für Einfluss T2 → Systemintegration

Eine Microservice-Architektur ermöglicht die nahtlose Integration externer Systeme in eigenständige Services (vgl. Microservice-Architektur). Externe APIs und Bibliotheken werden in eigenständigen Microservices entwickelt und über standardisierte Schnittstellen in das Gesamtsystem integriert. Die Kommunikation zwischen den Services erfolgt synchron über GraphQL und asynchron über Apache Kafka (vgl. Lösungsstrategie Q7). Das Monitoring-Modul überwacht die Kommunikation zwischen den Services und erkennt ungewöhnliche Aktivitäten oder Ausfälle (vgl. Lösungsstrategie Q8).

→ Einflussfaktor T2

Strategie für Einfluss T3 → Änderbarkeit

Das Softwaresystem wird nach dem Geheimnisprinzip durch die Implementierung einer modularen Architektur entworfen. Die einzelnen Module weisen eine hohe Kohäsion und lose Kopplung auf und können unabhängig voneinander entwickelt und ausgetauscht werden (vgl. Entwurfsmuster). Die Module werden als Microservices implementiert und in einer containerisierten K8s-Umgebung betrieben (vgl. Lösungsstrategie Q6). Dadurch wird die Flexibilität des Systems erhöht und Änderungen können schnell und ohne Auswirkungen auf andere Komponenten durchgeführt werden.

Ein *Konfigurations-Modul* realisiert die zentrale Verwaltung von Systemkonfigurationen und ermöglicht die einfache Anpassung an Gesetze und Vorschriften. Das Konfigurations-Modul speichert die Konfigurationsdaten in speziellen Konfigurationsdateien im JSON-Format und stellt eine Schnittstelle zur Abfrage der Daten zur Verfügung. Änderungen an der Konfiguration werden zentral über das Konfigurations-Modul vorgenommen und sind sofort für alle Komponenten verfügbar. Dazu wird das Konfigurations-Modul in die zentrale Verwaltungsoberfläche des Support-Systems integriert (vgl. Lösungsstrategie P7).

→ Einflussfaktor T3

Strategie für Einfluss T4 → Erweiterbarkeit

Die modulare Architektur des Systems und die Verwendung von Microservices ermöglichen eine einfache Erweiterung des Systems um neue Funktionalitäten und Dienste. Neue Funktionalitäten werden durch das Hinzufügen von Microservices realisiert. Die neuen Services

werden in Docker-Containern bereitgestellt und über K8s orchestriert. K8s ermöglicht die automatisierte Bereitstellung und Skalierung der neuen Services und stellt sicher, dass die Erweiterung des Systems ohne Ausfallzeiten erfolgt.

→ Einflussfaktor T4

Strategie für Einfluss T5 → Internationalisierung

Internationale Spracheinstellungen, Währungs- und Datumsformate werden über das Framework i18next realisiert (vgl. Lösungsstrategie P3). Die Formatierung von Währungs- und Datumsangaben wird über die *Formatting*-Komponente von i18next realisiert [i1822]. Durch einfaches Hinzufügen von Übersetzungsdateien können weitere Sprachen, Währungen und Datumsformate unterstützt werden.

Die Währungsumrechnung erfolgt über die API des Zahlungsdienstleisters. Alternativ stehen Node.js-Bibliotheken wie *money.js* für eine Währungsumrechnung auf Basis der Wechselkurse von *Open Exchange Rates* zur Verfügung [Ope22].

Die Anpassung an lokale Gesetze und Vorschriften erfolgt über das zentrale Konfigurations-Modul (vgl. Lösungsstrategie T7).

→ Einflussfaktor T5

Strategie für Einfluss T6 → Systemwartung

Um einen kontinuierlichen Betrieb während der Wartungsarbeiten zu gewährleisten, wird eine *Blue/Green-Deployment*-Strategie umgesetzt. Blue/Green Deployments ermöglichen die Bereitstellung von Updates ohne Ausfallzeiten und bieten die Möglichkeit eines schnellen Rollbacks bei Problemen [Ama21]. Eine *blaue Umgebung* repräsentiert die aktuelle Version der Anwendung, die in Produktion ist. Parallel dazu wird eine *grüne Umgebung* mit einer neuen Version der Anwendung bereitgestellt. Sobald die grüne Umgebung produktionsbereit ist, wird der Datenverkehr von der blauen Umgebung auf die grüne Umgebung umgeleitet. Sollten Probleme auftreten, wird der Datenverkehr zurück in die blaue Umgebung umgeleitet.

K8s unterstützt die Blue/Green-Deployment-Strategie durch die Bereitstellung von Containern und die automatische Umschaltung des Datenverkehrs sowie das Löschen alter Container. K8s speichert den Verlauf aller Änderungen und ermöglicht ein einfaches Rollback zu einer früheren Version. Der NGINX Ingress Controller unterstützt Traffic Splitting und ermöglicht so die Aufteilung des Datenverkehrs zwischen den verschiedenen Versionen der Anwendung (vgl. Lastverteilung). Dadurch wird die Systemverfügbarkeit während der Wartungsarbeiten gewährleistet und die Auswirkungen auf die Nutzer minimiert.

→ Einflussfaktor T6

Strategie für Einfluss T7 → Rechtssicherheit

Alle Transaktionen müssen gemäß der MiCA-Verordnung protokolliert und überwacht werden (vgl. Lösungsstrategie T8). Die Speicherung und Verarbeitung personenbezogener Daten muss nach den Vorgaben der DSGVO erfolgen (vgl. Lösungsstrategie T9). Alle steuerlich relevanten Daten müssen korrekt erfasst, zu Nachweiszwecken gespeichert und für den Export an die Finanzbehörden bereitgestellt werden (vgl. Lösungsstrategie T10). Darüber hinaus sind die App-Store-Richtlinien von Apple (*App Review Guidelines*) und Google (*Developer Programme Policy*) zu beachten. Die App-Store-Richtlinien enthalten Vorgaben zum Schutz der Privatsphäre, zur Sicherheit der Nutzerdaten und zur Einhaltung von Inhaltsrichtlinien [App24; Goo24e].

Die Umsetzung der regulatorischen Anforderungen erfolgt durch die Implementierung fest definierter Regeln, die mit Hilfe der Rules Engine durchgesetzt werden (vgl. Lösungsstrategie Q1). Die Regeln werden in separaten Modulen implementiert, die die Einhaltung der gesetzlichen Vorgaben überwachen und bei Verstößen entsprechende Maßnahmen einleiten. Die Anpassung an aktuelle Gesetze und Vorschriften erfolgt über die zentrale Verwaltungsoberfläche durch das Konfigurations-Modul (vgl. Lösungsstrategie T3).

Alle Daten und Protokolle, die einer gesetzlichen Aufbewahrungspflicht unterliegen oder zu Nachweiszwecken benötigt werden, werden in einer verteilten Datenbanklösung mit MySQL sicher gespeichert (vgl. Lösungsstrategie Q4). Im Falle einer behördlichen Prüfung oder eines Rechtsstreits stehen die Protokolle als Beweismittel zur Verfügung. Über die zentrale Verwaltungsoberfläche und das Datenbank-Modul können die Daten selektiv abgerufen und in einem standardisierten Format für die Behörden oder externe Prüfer exportiert werden.

→ Einflussfaktor T7

Strategie für Einfluss T8 → MiCA-Verordnung

Die Regeln zur Einhaltung der MiCA-Verordnung werden durch die Implementierung einer Rules Engine in zwei separaten Modulen – dem Transaktionsvalidierungs-Modul und einem *KYC-Modul* – umgesetzt (vgl. Lösungsstrategie Q1). Das *Transaktionsvalidierungs-Modul* überwacht alle Kauf- und Verkaufsaktivitäten und identifiziert verdächtige Transaktionen auf Grundlage der AML-Richtlinien. Die AML-Richtlinien werden in der Rules Engine im JSON-Format definiert und über die API des Transaktionsvalidierungs-Moduls abgerufen (vgl. Lösungsstrategie Q1). Verdächtige Transaktionen werden gestoppt, protokolliert und mit dem Zahlungsdienstleister synchronisiert. Die weitere Bearbeitung erfolgt durch den Zahlungsdienstleister und das Support-Team. Die Protokolle werden in der Datenbank gespeichert und bei Bedarf den zuständigen Behörden gemeldet (vgl. Lösungsstrategie T7).

Das KYC-Modul verwendet die API eines externen KYC-Dienstleisters, um die Identität von Neukunden (Kundenanwendung) und Partnerfilialen (Kassenanwendung) gemäß den AML-Richtlinien zu überprüfen. Anbieter wie *Faceki* und *Identitypass* stellen API-Schnittstellen als npm-Bibliotheken für React, React Native und Node.js zur Verfügung [fac24; pre24]. Das

Ergebnis der Identitätsprüfung wird systemintern in der Datenbank gespeichert und mit dem Zahlungsdienstleister synchronisiert. Das KYC-Modul wird mit dem Authentifizierungs-Modul (vgl. Lösungsstrategie Q3) verknüpft und in den Anmeldeprozess integriert.

Der Zahlungsdienstleister Bitpanda Custody ist ein zertifizierter Anbieter für die Kryptoverwahrung und bietet eine integrierte KYC-Lösung zur Einhaltung der AML-Richtlinien [Bit22b].

> Bitpanda Custody is an FCA registered custodian wallet provider, per 5th AML directive definition. Our built-in KYC and KYT controls ensure we can monitor all users and transactions for AML compliance [Bit22b].

→ Einflussfaktor T8

Strategie für Einfluss T9 → Datenschutz

Das Softwaresystem wird als eine Cloud-native Anwendung entwickelt und in einer K8s-Umgebung in der Cloud betrieben (vgl. Lösungsstrategie Q7). Das System muss so konfiguriert werden, dass die datenschutzrechtlichen Anforderungen der DSGVO und des BDSG erfüllt werden (vgl. Regulatorik & Datenschutz).

Die Speicherung und Verarbeitung der Daten erfolgt ausschließlich auf Servern innerhalb der Europäischen Union. Der Cloud-Standort kann in der Konfiguration des Cloud-Anbieters festgelegt werden. In GCP stehen für den Standort Europa 12 Rechenzentren zur Verfügung, davon 2 in Deutschland (Stand 04.09.2024) [Goo24f].

Der Zugriff auf die Daten wird durch die verteilte Datenbanklösung mit MySQL und die synchrone Datenreplikation mit NDB Cluster sichergestellt (vgl. Lösungsstrategie Q4). Die Rechte der Nutzer auf Auskunft, Berichtigung und Löschung ihrer Daten sind jederzeit gewährleistet.

Die Datenübertragung zwischen der Kundenanwendung, dem Kassensystem und der serverseitigen Anwendung erfolgt verschlüsselt über HTTPS (vgl. Lösungsstrategie Q5). Die Kommunikation zwischen den Microservices erfolgt verschlüsselt über mTLS (vgl. Lösungsstrategie Q2). Sensible Nutzerdaten werden verschlüsselt gespeichert und die Schlüssel sicher in der K8s-Umgebung verwaltet (vgl. Lösungsstrategie Q3).

Es gilt stets der Grundsatz der Datensparsamkeit. Es werden nur die Daten erhoben, die für die Funktionalität des Systems unbedingt notwendig sind. Jeder Microservice erhält nur Zugriff auf die Daten, die er zur Erfüllung seiner Aufgaben benötigt. Personenbezogene Daten in den Protokollen werden anonymisiert und nur für einen begrenzten Zeitraum gespeichert.

→ Einflussfaktor T9

Strategie für Einfluss T10 → Steuerrecht

Zur Einhaltung der steuerrechtlichen Vorschriften wird ein *Reporting-Modul* implementiert. Das Reporting-Modul ermöglicht die Erstellung von Finanzberichten für steuerliche Zwecke auf Basis der gespeicherten Transaktionsdaten. Das Reporting-Modul erfasst die steuerrelevanten Daten, erstellt eine Umsatzübersicht und berechnet die anfallenden Steuern. Der Export der Daten erfolgt im CSV-Format, um die Kompatibilität mit den Finanzbehörden zu gewährleisten. Auf das Modul kann sowohl über die Benutzeroberfläche der Kundenanwendung als auch über die Kassenanwendung zugegriffen werden. Der Zugriff auf die Daten wird durch die verteilte Datenbanklösung in Kombination mit einer synchronen Datenreplikation sichergestellt (vgl. Lösungsstrategie Q4).

→ Einflussfaktor T10

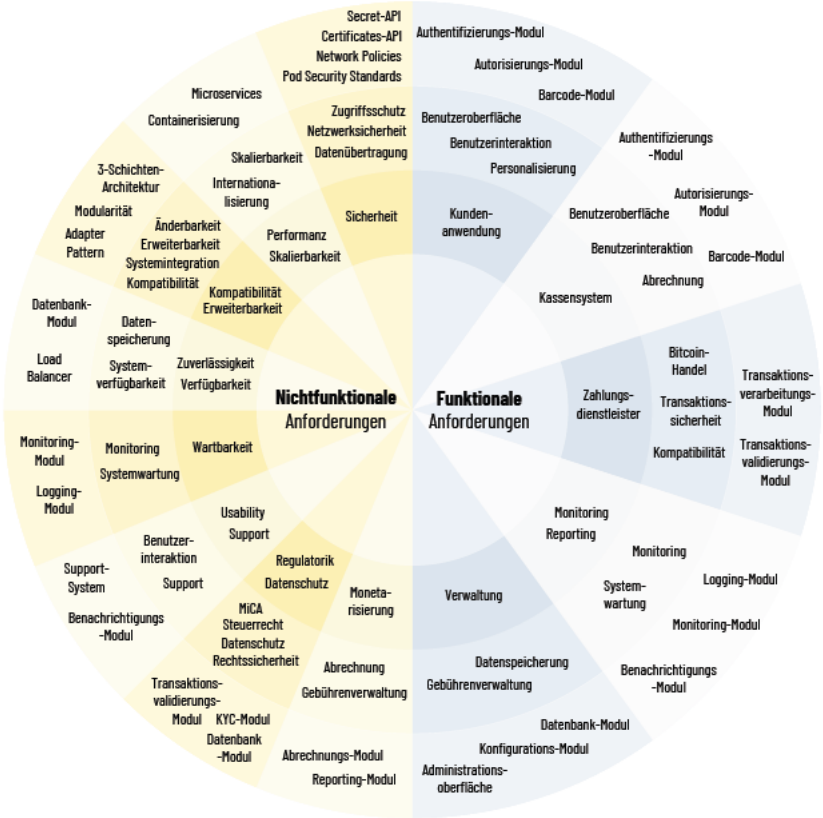

Abbildung 4.2: Strukturierter Entwicklungsprozess – Lösungsstrategien

4.3 Architekturentscheidungen

In diesem Kapitel werden die Architekturentscheidungen für das Softwaresystem zusammengefasst. Diese Entscheidungen basieren auf den zuvor identifizierten Einflussfaktoren und Lösungsstrategien (vgl. Strukturdiagramm Abb. 4.2) und bilden die Grundlage für die Imple-

mentierung des Systems. Ziel der Architektur ist es, eine modulare, skalierbare und sichere Plattform zu schaffen, die den regulatorischen Anforderungen des Bitcoin-Handels gerecht wird.

Die Softwarearchitektur wird als eine Drei-Schichten-Architektur konzipiert, die Benutzeroberfläche, Geschäftslogik und Datenzugriff voneinander trennt. Die clientseitigen Anwendungen – Verwaltungsoberfläche, Kundenanwendung und Kassenanwendung – werden in **React** und **React Native** entwickelt. Die serverseitige Anwendung wird in eigenständigen Microservices realisiert, die in **Node.js** entwickelt werden. Die Programmierung erfolgt einheitlich in der Programmiersprache **TypeScript**. Die Microservices werden in **Docker**-Containern bereitgestellt und in einem **K8s-Cluster** in der Cloud betrieben. Die Kommunikation zwischen den Services erfolgt synchron über **GraphQL** und asynchron über **Apache Kafka** (EDA). Für die Anbindung externer Systeme und Dienste werden standardisierte Schnittstellen – **GraphQL-APIs** – implementiert. Der Datenzugriff erfolgt über eine verteilte **MySQL**-Datenbank mit einer synchronen Datenreplikation über **NDB Cluster**. Der Zugriff auf den K8s-Cluster erfolgt über einen **NGINX Ingress Controller** mit entsprechender Konfiguration für die Lastverteilung und **Traffic Splitting**. Diese Architektur bietet einen wirksamen Schutz gegen Angriffe und Ausfälle und ermöglicht eine dynamische Skalierung der Systemressourcen sowie die Durchführung von **Blue/Green Deployments**.

Folgende Microservices werden entwickelt:

- Authentifizierungs-Modul: Verwaltet die Benutzeranmeldung und -authentifizierung unter Verwendung des **OAuth 2.0**-Standards.
- Autorisierungs-Modul: Verwaltet die Zugriffsrechte der Nutzer über eine RBAC mittels **OPA Gatekeeper**.
- KYC-Modul: Integriert die **KYC-Dienstleister-API** zur Identitätsprüfung gemäß den AML-Richtlinien.
- Transaktionsverarbeitungs-Modul: Integriert die **Zahlungsdienstleister-API** und koordiniert die Auftragsabwicklung im Kauf- und Verkaufsprozess.
- Transaktionsvalidierungs-Modul: Identifiziert verdächtige Transaktionen zur Einhaltung der AML-Richtlinien anhand bekannter Betrugsmuster über die **json-rules-engine**.
- Barcode-Modul: Dient der Erstellung und Verarbeitung von Barcodes zur Referenzierung von Kauf- und Verkaufsaufträgen.
- Konfigurations-Modul: Ermöglicht die zentrale Verwaltung von Systemkonfigurationen und die einfache Anpassung an gesetzliche Anforderungen über Konfigurationsdateien im **JSON**-Format.
- Monitoring-Modul: Überwacht den Ressourcenverbrauch, die Anwendungsleistung und Sicherheitsereignisse im K8s-Cluster anhand von Metriken und Audit-Protokollen über **Prometheus** und **Grafana**.
- Logging-Modul: Protokolliert wichtige Systemereignisse gemäß den Best Practices für das Logging in Node.js-Anwendungen.
- Benachrichtigungs-Modul: Informiert die Nutzer per E-Mail – über **SendGrid** oder **Mailchimp** – oder per Push-Benachrichtigung – über **Firebase Cloud Messaging** oder **OneSignal** – über wichtige Ereignisse.
- Abrechnungs-Modul: Erfasst Finanzdaten und stellt Abrechnungen für Partnerfilialen und externe Dienstleister zum Export im **CSV**-Format bereit.

- Reporting-Modul: Generiert Finanzberichte zu Steuerzwecken und ermöglicht den Export steuerrelevanter Daten im **CSV**-Format.
- Support-Modul: Erfasst und priorisiert Supportanfragen auf der Grundlage eines **Ticketsystems**, um die Kommunikation zwischen Nutzern und dem Support-Team zu optimieren.
- Datenbank-Modul: Abstrahiert die Interaktion mit der Datenbank über ein ORM-Framework wie **TypeORM** oder **Sequelize**.

5 Implementierung

In diesem Kapitel wird die praktische Umsetzung der serverseitigen Anwendung beschrieben und die Bitcoin-Kauffunktion in einer realitätsnahen Umgebung getestet. Aufbauend auf dem im Architekturentwurf definierten Konzept werden die Schritte zur Einrichtung der Entwicklungs- und Produktionsumgebung, die Implementierung ausgewählter Kernkomponenten der Software sowie die Integration des Zahlungsdienstleisters erläutert. Das Ziel ist es, einen umfassenden Einblick in die technischen Aspekte der Implementierung zu geben und die getroffenen Architekturentscheidungen fundiert zu begründen.

5.1 Einrichtung der Produktionsumgebung

Die Produktionsumgebung bildet die Grundlage für den zuverlässigen und sicheren Betrieb des Softwaresystems. Bei der Konzeption und Implementierung wurden insbesondere die hohen Anforderungen an die Verfügbarkeit, Skalierbarkeit und Sicherheit einer Cloud-nativen Anwendung berücksichtigt.

Auswahl des Cloud-Anbieters

Die **Google Cloud Platform** wurde aufgrund ihrer umfassenden Funktionen in den Bereichen Virtualisierung, Container-Orchestrierung und Netzwerkmanagement als Cloud-Anbieter ausgewählt. **Google Compute Engine** ermöglicht die Bereitstellung und Verwaltung virtueller Maschinen und bietet robuste Sicherheitsmechanismen, flexible Skalierungsmöglichkeiten und eine benutzerfreundliche Oberfläche zur Ressourcenverwaltung [Goo21]. **Google Artifact Registry** ermöglicht die sichere Speicherung und Verwaltung von Docker Images in einem zentralen Repository [Goo24g]. GCP bietet außerdem flexible Standortoptionen, um die Latenzzeiten zu minimieren, die Verfügbarkeit zu maximieren und die Einhaltung von Datenschutzbestimmungen zu gewährleisten (vgl. Lösungsstrategie T9).

Einrichtung der virtuellen Maschinen

Zur Bereitstellung der notwendigen Ressourcen wurden in GCP mehrere virtuelle Maschinen konfiguriert. Die Auswahl und Spezifikation der VMs erfolgte entsprechend den Anforderungen der jeweiligen Systemkomponenten:

Der Load Balancer wird auf einer VM des Typs *e2-micro* (2 vCPUs, 1 GB RAM) [Goo24h] betrieben. Die Lastverteilung des eingehenden Datenverkehrs auf die verfügbaren Ressourcen erfordert im Vergleich zu den anderen Komponenten des K8s-Clusters geringe Ressourcen. Die Control Plane des K8s-Clusters wird auf einer VM des Typs *e2-medium* (2 vCPUs, 4 GB RAM) [Goo24h] betrieben. Diese Maschine übernimmt die Steuerung und Verwaltung des Clusters und benötigt daher erweiterte Ressourcen, insbesondere hinsichtlich des Arbeitsspeichers. Zwei VMs vom Typ *e2-small* (2 vCPUs, 2 GB RAM) [Goo24h] dienen als Worker Nodes, auf denen die containerisierten Anwendungen ausgeführt werden.

Alle VMs wurden mit einem aktuellen LTS-Release des **Ubuntu-Betriebssystems** – *Ubuntu 22.04.4 LTS* [Ubu24] – ausgestattet, um eine konsistente und stabile Basis für die Installation der weiteren Softwarekomponenten zu gewährleisten.

Konfiguration der virtuellen Maschinen

Auf den virtuellen Maschinen wurden zunächst **Node.js** und **npm** installiert, um die Ausführung der serverseitigen Anwendung zu ermöglichen.

```
# Installation von Node.js und npm
sudo apt install nodejs
sudo apt install npm
```

Zur Containerisierung der Anwendung wurde die aktuelle Version von **Docker Engine** installiert [Doc24d]. Für die Integration mit K8s wurde das Container Runtime Interface **CRI-Dockerd** installiert, das als Schnittstelle zwischen K8s und der Docker Engine fungiert [Kub24c]. Die Docker-Installationen konnten anschließend mithilfe eines *hello-world*-Containers und der *Systemd*-Dienste verifiziert werden.

```
# Installation von Docker Engine
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker
    -compose-plugin
# Installation von CRI-Dockerd
wget https://github.com/.../<CRI-DOCKERD-VERSION>.deb
sudo dpkg -i <CRI-DOCKERD-VERSION>.deb
# Verifizierung der Docker-Installationen
docker run hello-world
> Hello from Docker!
sudo systemctl status cri-docker.service
sudo systemctl status cri-docker.socket
> Active: active (running)
```

Schließlich wurden die K8s-Komponenten **kubeadm**, **kubelet** und **kubectl** installiert [Kub24j]. Kubeadm dient zur Initialisierung und Verwaltung des K8s-Clusters. Kubelet läuft auf jedem Knoten und steuert die Ausführung der Pods (vgl. Kubernetes). Kubectl ist das Kommandozeilenwerkzeug zur Interaktion mit dem K8s-Cluster (vgl. Kubernetes). Um Integrationsprobleme zu vermeiden und einen reibungslosen Betrieb des Clusters zu gewährleisten, wurden anschließend die Versionen der K8s-Komponenten fixiert.

```
# Installation der K8s-Komponenten
sudo apt-get install -y kubelet kubeadm kubectl
# Festlegung der Versionen im Paketmanager
sudo apt-mark hold kubelet kubeadm kubectl
```

Aufbau des Kubernetes-Clusters

Der K8s-Cluster wurde für den Einsatz in der Produktionsumgebung konfiguriert [Kub24k]. Zunächst erfolgte die Initialisierung der Control Plane auf der entsprechenden VM mittels kubeadm. Anschließend wurde das Netzwerk-Plugin **Calico** konfiguriert, welches die Kommunikation zwischen den Pods über entsprechende Network Policies steuert (vgl. Lösungsstrategie Q2). Im Vergleich zu traditionellen Paketfilterlösungen wie *iptables* bietet Calico eine

signifikante Verbesserung der Latenz und des Durchsatzes. Calico kann die Regeln für den Netzwerkverkehr auch bei einer steigenden Anzahl von Pods und Policies effizient durchsetzen und bietet einen geringen Overhead bei der Kommunikation zwischen den Pods [Bud+21]. Als Grundlage dient eine Konfigurationsdatei im YAML-Format [Tig24].

```
# Initialisierung der Control Plane mit Angabe der Container Runtime und des IP-
    Adressbereichs
sudo kubeadm init --cri-socket /run/cri-dockerd.sock --pod-network-cidr=192.168.0.0/16
> Your Kubernetes control-plane has initialized successfully!
# Calico Konfigurationsdatei herunterladen und anwenden
curl -O https://calico-v3-25.netlify.app/archive/v3.25/manifests/calico.yaml
kubectl apply -f calico.yaml
```

Im Anschluss wurden die Worker Nodes mithilfe des kubeadm-join-Befehls dem Cluster hinzugefügt. Dadurch wurden die Knoten in die Clusterverwaltung integriert und können nun Containeranwendungen (Pods) ausführen.

```
# Generierung des Join-Befehls auf der Control Plane
sudo kubeadm token create --print-join-command
> kubeadm join <MASTER_NODE_INTERNAL_IP> <TOKEN> <HASH>
# Integration der Worker Nodes in den Cluster
sudo kubeadm join <MASTER_NODE_INTERNAL_IP> <TOKEN> <HASH>
> This node has joined the cluster
```

Zur logischen Trennung der Umgebungen wurden innerhalb des Clusters unterschiedliche Namespaces – darunter *development*, *test* und *production* – erstellt. Diese Strukturierung erleichtert die Verwaltung und Isolierung der Ressourcen und die Durchführung von **Blue/-Green Deployments**.

```
# Erstellen der Namespaces
kubectl create namespace development
kubectl create namespace test
kubectl create namespace production
```

Implementierung des Load Balancers

Für die Verteilung des eingehenden Datenverkehrs auf die verfügbaren Ressourcen im K8s-Cluster wurde ein Load Balancer auf Basis von **NGINX** implementiert. NGINX wurde auf der dafür vorgesehenen VM installiert und entsprechend konfiguriert. Die NGINX-Konfiguration wurde so angepasst, dass der Server als Reverse Proxy fungiert und den Datenverkehr an die entsprechenden K8s-Services weiterleitet. Es wurden Upstreams für die verschiedenen Dienste definiert und spezifische Proxy-Regeln implementiert, die eine Weiterleitung des Datenverkehrs basierend auf den Endpunkten ermöglichen. Diese Konfiguration gewährleistet eine effiziente Lastverteilung und erhöht die Verfügbarkeit der Services (vgl. Lastverteilung).

```
# Upstream: Authentifizierungs-Service
upstream auth_service {
    server <WORKER-1-INTERNAL-IP>:<NODEPORT-AUTH-SERVICE>;
    server <WORKER-2-INTERNAL-IP>:<NODEPORT-AUTH-SERVICE>;
}
# Upstream: Transaktions-Service
upstream transaction_service {
```

```
        server <WORKER-1-INTERNAL-IP>:<NODEPORT-TRANSACTION-SERVICE>;
        server <WORKER-2-INTERNAL-IP>:<NODEPORT-TRANSACTION-SERVICE>;
    }
    # Auth-Service auf Endpunkt /login
    location /login {
        proxy_pass http://auth_service;
    }
    # Auth-Service auf Endpunkt /protected
    location /protected {
        proxy_pass http://auth_service;
    }
    # Transaction-Service auf Endpunkt /transactions
    location /transactions {
        proxy_pass http://transaction_service;
    }
```

Um die Sicherheit der Kommunikation zu gewährleisten, wurden Sicherheitszertifikate von der Control Plane zum Load Balancer übertragen und in die NGINX-Konfiguration integriert. Dadurch wird sichergestellt, dass der Datenverkehr zwischen den Clients und dem Load Balancer sowie zwischen dem Load Balancer und den K8s-Services verschlüsselt übertragen wird. Anschließend wurde die NGINX-Konfiguration getestet und der Dienst neu gestartet.

```
    # Testen der NGINX-Konfiguration
    sudo nginx -t
    # Neustarten des NGINX-Dienstes
    sudo systemctl restart nginx
```

Um die Erreichbarkeit des Load Balancers zu gewährleisten, wurde der VM eine statische IP-Adresse zugewiesen und eine entsprechende Firewall-Regel konfiguriert.

```
    # Reservierung einer statischen IP-Adresse
    gcloud compute addresses create <ADDRESS_NAME>
    gcloud compute addresses describe <ADDRESS_NAME>
    > address: <STATIC-IP-ADDRESS>
    # Zuweisung der IP-Adresse an den Load Balancer
    gcloud compute instances add-access-config <VM-NAME> --access-config-name="external-
        nat" --address=<STATIC-IP-ADDRESS>
    # Konfiguration der Firewall-Regeln
    gcloud compute firewall-rules create allow-k8s-api-server --allow tcp:<PORT>
```

5.2 Einrichtung der Entwicklungsumgebung

Zur effizienten Entwicklung und Wartung des Softwaresystems wurde die lokale Entwicklungsumgebung mit **Visual Studio Code** und der Erweiterung **Remote - SSH** eingerichtet.

Lokale Entwicklungsumgebung

Die lokale Entwicklungsumgebung basiert auf *macOS Sonoma*. Als Code-Editor wurde Visual Studio Code mit der Erweiterung Remote - SSH verwendet. Visual Studio Code ist ein plattformübergreifender Code-Editor, der eine Vielzahl von Programmiersprachen und Erweiterungen unterstützt [Mic24a]. Durch die Einrichtung sicherer SSH-Verbindungen und die

Anpassung der Firewall-Regeln konnten Verbindungsprobleme behoben und eine sichere Remote-Entwicklung gewährleistet werden. Es wurden SSH-Schlüssel generiert und in der Konfigurationsdatei *~/.ssh/config* hinterlegt, um eine passwortlose Authentifizierung zu den VMs zu ermöglichen. Außerdem wurde die SSH-Verbindung in den Firewall-Einstellungen der VMs konfiguriert, um den SSH-Zugriff von der lokalen Entwicklungsumgebung zu ermöglichen. Die Erweiterung Remote - SSH ermöglichte eine nahtlose Integration zwischen der lokalen Entwicklungsumgebung und der Produktionsumgebung in der Cloud [Mic24b]. Es konnte eine direkte Verbindung mit Visual Studio Code zu den VMs hergestellt werden, wodurch das Bearbeiten von Dateien und das Ausführen von Anwendungen direkt auf den Servern möglich war.

Außerdem wurden Node.js und npm in denselben Versionen wie in der Produktionsumgebung installiert, um eine konsistente Entwicklungsumgebung zu gewährleisten und Versionskonflikte zu vermeiden. **TypeScript** wurde als einheitliche Programmiersprache für die Entwicklung der Anwendungen verwendet, um von den Vorteilen der statischen Typisierung und der verbesserten Codequalität zu profitieren.

Versionsverwaltung

Für die Versionsverwaltung des Quellcodes wurde **Git** [Sof24] verwendet. Auf einer der VMs wurde ein zentrales **Bare Repository** eingerichtet, das als *Remote Repository* fungierte [Gee19]. Die lokale Entwicklungsumgebung wurde mit diesem Remote Repository verbunden. Mit dieser Konfiguration konnten Änderungen lokal entwickelt, in das zentrale Repository übertragen und auf den VMs synchronisiert werden.

5.3 Transaktions-Service

Die Bitcoin-Kauffunktion wurde als Microservice (**Transaktions-Service**) realisiert und in den K8s-Cluster integriert.

Entwicklung des Microservices

Der Transaktions-Service wurde in Node.js unter Verwendung des Frameworks **Express.js** entwickelt. Express.js ist ein serverseitiges Framework zur einfachen und effizienten Entwicklung von Webanwendungen und APIs in Node.js [Ope24c]. Es wurde eine klare Projektstruktur definiert, die eine Trennung von Endpunkten, Authentifizierung, Datenbankzugriff und Konfiguration ermöglicht. Es wurden drei API-Endpunkte implementiert, die die Erstellung von Kaufaufträgen, die Ausgabe der Transaktionshistorie und die Aktualisierung des Auftragsstatus ermöglichen.

```
# Express.js Framework
import express from 'express';
const app = express();
app.use(express.json());

# API-Endpunkte
app.post('/transactions', authMiddleware, createTransaction);
```

```
app.put('/transactions/:id/status', authMiddleware, updateTransactionStatus);
app.get('/transactions', authMiddleware, getTransactions);
```

Um die Robustheit des Microservices zu gewährleisten, wurden Mechanismen zur Fehlerbehandlung und zum Logging implementiert.

Containerisierung & Deployment

Um eine konsistente und zuverlässige Bereitstellung zu gewährleisten, wurde der Transaktions-Service mit Docker containerisiert, in die Google Artifact Registry hochgeladen und im K8s-Cluster bereitgestellt. In einem **Dockerfile** wurden die notwendigen Schritte zur Erstellung des **Docker Images** definiert. Dazu gehören die Installation der notwendigen Abhängigkeiten, die Kompilierung des TypeScript-Codes, die Konfiguration der Umgebungsvariablen und der Start der Container-Anwendung.

```
FROM node:current # Basis-Image von Node.js
WORKDIR /app # Arbeitsverzeichnis im Container
COPY package*.json ./ # Paketdefinitionen
RUN npm install # Installation der Pakete

COPY src ./src # Quellcode der Anwendung
COPY tsconfig.json ./ # TypeScript-Konfiguration
COPY .env ./ # Umgebungsvariablen
RUN npx tsc # Kompilierung des TypeScript-Codes

EXPOSE <PORT> # Port der Containeranwendung

CMD ["node", "dist/index.js"] # Startbefehl
```

Das resultierende Docker Image wurde in die Artifact Registry hochgeladen, um es für das Deployment im K8s-Cluster verfügbar zu machen.

```
# Erstellung des Docker Images
docker build -t transaction-service:<VERSION> .
# Hochladen des Images in die Artifact Registry
docker push <LOCATION>/<PROJECT>/docker-repo/transaction-service:<VERSION>
```

Anschließend wurde ein K8s **Deployment** erstellt, das die Verteilung des Microservices auf die Worker Nodes des Clusters steuert. Zusätzlich wurde ein K8s **Service** konfiguriert, um den Zugriff auf den Microservice innerhalb des Clusters und von externen Clients zu ermöglichen. Schließlich wurden die notwendigen **Secrets** definiert, um eine sichere Kommunikation mit der MySQL-Datenbank und der Authentifizierung zu gewährleisten. Diese Vorgehensweise gewährleistet eine sichere, skalierbare und hochverfügbare Bereitstellung des Microservices.

```
# Kubernetes Deployment
apiVersion: apps/v1 # Version der K8s API
kind: Deployment # Ressourcentyp zur Bereitstellung von Pods
metadata:
    name: transaction-service # Name des Deployments
    namespace: transaction-prod # Namespace des Deployments
spec:
    replicas: 3 # Anzahl der Pod-Instanzen (Replikate)
```

```
        template:
            imagePullSecrets:
                name: gcr-json-key # Zugriff auf die Artifact Registry
            containers:
                name: transaction-service # Name des Containers
                image: <LOCATION>/<PROJECT>/docker-repo/transaction-service:<VERSION> #
                    Docker Image
                ports:
                    containerPort: <PORT> # Port der Containeranwendung
                env: # Umgebungsvariablen
```

```
    # Kubernetes Service
    apiVersion: v1
    kind: Service # Ressourcentyp zur Bereitstellung von Netzwerkdiensten
    metadata:
        name: transaction-service # Name des Services
        namespace: transaction-prod # Namespace des Services
    spec:
        type: NodePort # Erreichbarkeit des Services
        ports:
            port: <PORT> # Cluster-interne Portnummer
            targetPort: <PORT> # Port der Containeranwendung
            nodePort: <NODE-PORT> # Externe Portnummer
```

```
    # Kubernetes Deployment und Service erstellen
    kubectl apply -f <KUBERNETES-DEPLOYMENT>.yaml
    kubectl apply -f <KUBERNETES-SERVICE>.yaml
    # Kubernetes Secrets erstellen
    kubectl apply -f <MYSQL-SECRET>.yaml
    kubectl apply -f <AUTH-SECRET>.yaml
```

Integration der Bitpanda API

Die Integration der **Bitpanda API** wurde mit Hilfe der von Bitpanda zur Verfügung gestellten *OpenAPI*-Spezifikation (*openapi.json*) [Bit24b] realisiert. Diese Spezifikation diente als Grundlage für die Implementierung der Schnittstellen und die Kommunikation mit dem Zahlungsdienstleister. Durch die Verwendung der OpenAPI-Spezifikation konnte eine konsistente und fehlerfreie Implementierung der API-Aufrufe sichergestellt werden.

Für Entwicklungs- und Testzwecke wurde ein *Mock-Server* mit **prism-cli** [sto24] eingerichtet, der die Bitpanda API-Endpunkte lokal zur Verfügung stellte. Der Mock-Server ermöglichte die Simulation realer API-Interaktionen sowie die Entwicklung und das Testen ohne direkten Zugriff auf die Bitpanda API. Dies ermöglichte eine frühzeitige Integration der Bitpanda API in die Anwendung und beschleunigte den Entwicklungsprozess.

Die Authentifizierung gegenüber der Bitpanda API erfolgte über **JWT-Token** (*JSON Web Tokens*), die in den Anfragen an die API im *HTTP-Header* übermittelt wurden. JSON Web Tokens sind base64-kodierte Token, die ein JSON-Objekt mit Informationen über den Nutzer und die Berechtigungen enthalten [JBS15]. Die Verarbeitung der API-Anfragen erfolgte gemäß der OpenAPI-Spezifikation, so dass die Transaktionsdaten in der Anwendung korrekt verarbeitet

und der Transaktionsstatus aktualisiert werden konnte. Diese Integration ermöglicht eine nahtlose Kommunikation mit dem Zahlungsdienstleister und stellt die korrekte Verarbeitung von Bitcoin-Kaufaufaufträgen sicher.

```
# Download der OpenAPI-Spezifikation
curl https://techsolutions.bitpanda.com/download/openapi.json
# Einrichtung des Mock-Servers
npx @stoplight/prism-cli mock openapi.json --host <IP>
# Erstellung von JWT-Tokens
import jwt from 'jsonwebtoken';
<JWT-TOKEN> = jwt.sign(<PAYLOAD>, <JWT_SECRET>, <EXPIRATION>);
# API-Anfrage mit JWT-Token (Bitcoint-Kauf)
curl -X POST http://<MOCK-SERVER-IP>:<PORT>/v1/offers \
    -H "Content-Type: application/json" \
    -H "Authorization: Bearer <JWT-TOKEN>" \
    -H "bp-user-id: <BITPANDA-USER-ID>" \
    -d '{
        "type": "buy",
        "fiat_id": 1,
        "asset_id": 1,
        "amount": "100.00",
        "amount_defined_for": "fiat"
    }'
> {"data":{"id":"<PAYMENT-ID>","type":"buy","fiat_id":1,"asset_id":1,"fiat_amount":"
    100.00","asset_amount":"0.0018","asset_price":"56021.68","offer_validity":<
    VALIDITY>,"expires_at":<EXPIRATION>,"time":<TIMESTAMP>}}
```

5.4 Authentifizierungs-Service

Für die sichere Authentifizierung und Autorisierung der Nutzer wurde ein **Authentifizierungs-Service** mit Open Policy Agent implementiert und in den K8s-Cluster integriert.

Entwicklung des Microservices

Der Authentifizierungs-Service wurde analog zum Transaktions-Service in Node.js mit dem Express.js Framework entwickelt. Es wurden API-Endpunkte für die Anmeldung und zur Überprüfung des Authentifizierungsstatus implementiert. Nach erfolgreicher Authentifizierung wird ein JWT-Token generiert, das den Nutzer bei zukünftigen Anfragen authentifiziert. Der Authentifizierungs-Service überprüft die Gültigkeit des JWT-Tokens und steuert den Zugriff auf die geschützten Ressourcen. Dadurch wird sichergestellt, dass nur authentifizierte Nutzer Zugriff auf geschützte Ressourcen erhalten.

Für Autorisierungsentscheidungen wurde **OPA** (Open Policy Agent) integriert. Der Authentifizierungs-Service kommuniziert mit dem Open Policy Agent, um anhand definierter Richtlinien zu entscheiden, ob ein Nutzer Zugriff auf eine bestimmte Ressource erhält. Diese Trennung von Authentifizierung und Autorisierung erhöht die Flexibilität und Sicherheit des Systems.

```
# API-Endpunkte
app.post('/login', checkCredentials, generateToken, returnToken);
app.get('/protected', verifyToken, checkAuthorization, grantAccess);
```

Einsatz von OPA

Der Open Policy Agent wurde als zentrale Policy Engine für die Autorisierung in den K8s-Cluster integriert. Die Zugriffsrichtlinien wurden in *Rego* – der Policy-Sprache von OPA – [Ope24d] definiert und in einer **ConfigMap** im K8s-Cluster hinterlegt.

```
# OPA-Policy-Configmap
apiVersion: v1
kind: ConfigMap # Ressourcentyp zur Speicherung von Konfigurationsdaten
metadata:
    name: opa-policy # Name der ConfigMap
    namespace: auth-prod # Namespace der ConfigMap
data:
    policy.rego: | # Richtlinie in Rego
        package authz # Paketdefinition
        default allow = false # Standardrichtlinie (Zugriff verweigern)
        allow {input.user.role == "admin"} # Admin-Zugriff erlauben
```

OPA wurde als eigenständiger Service im K8s-Cluster bereitgestellt. Die Microservices kommunizieren über HTTP mit OPA, um Autorisierungsentscheidungen für eingehende Anfragen zu erhalten. Diese Architektur ermöglicht eine flexible und zentrale Verwaltung der Zugriffsrichtlinien und erleichtert die Anpassung an sich ändernde Sicherheitsanforderungen.

5.5 Datenbank-Service

Die Datenhaltung wurde mit Hilfe einer **MySQL**-Datenbank als Microservice – dem **Datenbank-Service** – innerhalb des K8s-Clusters realisiert.

Bereitstellung der Datenbank

Für die persistente Speicherung von Daten wurde eine MySQL-Datenbank innerhalb des K8s-Clusters bereitgestellt. Um dauerhaften Speicher bereitzustellen, wurden ein **Persistent Volume** (*PV*) und ein **Persistent Volume Claim** (*PVC*) konfiguriert. Diese Ressourcen gewährleisten, dass die Daten auch bei einem Neustart oder Ausfall der Pods erhalten bleiben. Die MySQL-Datenbank wurde als Deployment im Cluster bereitgestellt und über einen internen Service für andere Pods innerhalb des Clusters erreichbar gemacht.

```
# Persistent Volume
apiVersion: v1
kind: PersistentVolume # Ressourcentyp zur Bereitstellung von persistentem Speicher
metadata:
    name: mysql-pv # Name des Persistent Volumes
spec:
    capacity:
        storage: 1Gi # bereitgestellter Speicherplatz
    accessModes:
        ReadWriteOnce # bereitgestellter Zugriffsmodus
    persistentVolumeReclaimPolicy: Retain # Wiederverwendung des Volumes
    hostPath:
        path: "/mnt/data/mysql" # Speicherort auf dem Host
```

```
# Persistent Volume Claim
apiVersion: v1
kind: PersistentVolumeClaim # Ressourcentyp zur Anforderung von persistentem Speicher
metadata:
    name: mysql-pv-claim # Name des Persistent Volume Claims
    namespace: database-prod # Namespace des Claims
spec:
    accessModes:
        - ReadWriteOnce # angeforderter Zugriffsmodus
    resources:
        requests:
            storage: 1Gi # Speicheranforderung
```

```
# Kubernetes Volume erstellen
kubectl apply -f <KUBERNETES-PV>.yaml
kubectl apply -f <KUBERNETES-PVC>.yaml
# Kubernetes Deployment und Service erstellen
kubectl apply -f <KUBERNETES-DEPLOYMENT>.yaml
kubectl apply -f <KUBERNETES-SERVICE>.yaml
```

Sensible Konfigurationsdaten wie Datenbankpasswörter wurden mithilfe von Secrets verwaltet, um die Sicherheit der Daten zu gewährleisten und unautorisierten Zugriff zu verhindern.

Datenbankverwaltung

Die Datenbankverwaltung erfolgte über einen **MySQL-Client**, der innerhalb des MySQL-Pods ausgeführt wurde.

```
# Zugriff auf den MySQL-Pod
kubectl exec -it <MYSQL-POD> -n <NAMESPACE> -- bash
# Verbindung zur MySQL-Datenbank
mysql -u <USERNAME> -p <PASSWORD>
# Erstellung der Transaktions-Tabelle
CREATE TABLE transactions (
    id INT AUTO_INCREMENT PRIMARY KEY, # Primary Key mit automatischer Inkrementierung
    user_id INT, # Foreign Key auf Nutzer-ID (nicht implementiert)
    amount DECIMAL(10, 2), # Formatierung des Kaufbetrags
    type ENUM('buy', 'sell'), # Transaktionstyp als Enum
    status ENUM('unbezahlt', 'bezahlt', 'gekauft', 'verkauft', 'ausgezahlt', '
        Gutscheincode'), # Auftragsstatus als Enum
    payment_id VARCHAR(36), # Zahlungs-ID von Bitpanda API
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP # Erstellungszeitpunkt der
        Transaktion
);
```

Datebankzugriff

Für die Anbindung an die Datenbank wurde das Node.js-Modul **mysql2** verwendet. Es wurde ein Verbindungspool eingerichtet, um die Effizienz der Datenbankzugriffe zu optimieren und die Ressourcen optimal zu nutzen [Sid24].

```
# MySQL-Verbindungspool
import mysql from 'mysql2/promise';
export const pool = mysql.createPool(<DB-CONFIG>);

# Datenbankabfrage
import { pool } from '../db';
const [result] = await pool.query(<SQL-QUERY>);
```

5.6 Qualitätssicherung

Nach der Einrichtung der Produktionsumgebung und der Implementierung der einzelnen Komponenten wurde die serverseitige Anwendung umfassend getestet, um die Funktionalität und Zuverlässigkeit sicherzustellen.

Systemprüfung

Zunächst wurden die Komponenten der Control Plane überprüft, um sicherzustellen, dass die Steuerung und Verwaltung des K8s-Clusters ordnungsgemäß funktionieren.

```
# Cluster-Diagnose
kubectl cluster-info
> Kubernetes control plane is running at https://<CLUSTER-IP>:<PORT>
# Control-Plane-Komponenten
kubectl logs <KUBE-APISERVER -n kube-system
kubectl logs <KUBE-CONTROLLER-MANAGER> -n kube-system
kubectl logs <KUBE-ETCD> -n kube-system
kubectl logs <KUBE-SCHEDULER> -n kube-system
```

Anschließend wurden die Worker Nodes sowie die Pods und Services im K8s-Cluster getestet, um sicherzustellen, dass die Containeranwendungen korrekt bereitgestellt werden und verfügbar sind.

```
# Knotenstatus
kubectl get nodes
# Status der Pods und Services
kubectl get pods -n <NAMESPACE>
kubectl get svc -n <NAMESPACE>
```

Funktionsprüfung

Im Rahmen des Funktionstests wurden die API-Endpunkte der Microservices auf ihre korrekte Funktionalität überprüft. Typische Anwendungsfälle wie die Benutzeranmeldung, die Erstellung von Kaufaufträgen und die Aktualisierung des Auftragsstatus wurden simuliert und die Ergebnisse validiert. Durch diese Tests wurde sichergestellt, dass die Komponenten reibungslos zusammenarbeiten und die Geschäftsprozesse korrekt abgebildet werden.

```
# Benutzeranmeldung (Authentifizierungs-Service + OPA)
curl -X POST http://<LOAD-BALANCER-IP>/login -H "Content-Type: application/json" -d '
    {"username":<USERNAME>,"password":<PASSWORD>}'
> {"token":"<JWT-TOKEN>"}
```

```
# Kaufauftrag erstellen (Transaktions-Service + Datenbank-Service)
curl -X POST http://<LOAD-BALANCER-IP>/transactions -H "Authorization: Bearer <JWT-
    TOKEN>   -H "Content-Type: application/json" -d '{"amount":<BETRAG>,"type":"buy"}'
> {"transactionId":<TRANSACTION-ID>,"status":"unbezahlt"}
# Auftragsstatus aktualisieren (Transaktions-Service + Datenbank-Service + Bitpanda
    API)
curl -X PUT http://<LOAD-BALANCER-IP>/transactions/<TRANSACTION-ID>/status -H "
    Authorization: Bearer <JWT-TOKEN> -H "Content-Type: application/json" -d '{"status
    ":"bezahlt"}'
> {"message":"Transaction updated and payment initiated successfully","paymentId":"<
    PAYMENT-ID>"}
# Transaktionshistorie abrufen (Transaktions-Service + Datenbank-Service)
curl -X GET http://<LOAD-BALANCER-IP>/transactions -H "Authorization: Bearer <JWT-
    TOKEN>"
> [{"id":1,"user_id":1,"amount":<BETRAG>,"type":"buy","status":"bezahlt","payment_id"
    :<PAYMENT-ID>,"created_at":<TIMESTAMP>}]
```

Skalierbarkeit

Durch den Einsatz von K8s und die Verwendung von Microservices kann das System flexibel skaliert und an unterschiedliche Lastanforderungen angepasst werden. Die Anzahl der Pod-Replikate und die Ressourcenanforderungen der Services können jederzeit dynamisch angepasst werden, um eine optimale Ressourcenauslastung zu gewährleisten.

```
# OPA-Service skalieren
kubectl scale deployment <OPA-SERVICE> --replicas=<ANZAHL-REPLIKATE> -n <NAMESPACE>
# Authentifizierungs-Service skalieren
kubectl scale deployment <AUTH-SERVICE> --replicas=<ANZAHL-REPLIKATE> -n <NAMESPACE>
# Database-Service skalieren
kubectl scale deployment <MYSQL-SERVICE> --replicas=<ANZAHL-REPLIKATE> -n <NAMESPACE>
# Transaktions-Service skalieren
kubectl scale deployment <TRANSACTION-SERVICE> --replicas=<ANZAHL-REPLIKATE> -n <
    NAMESPACE>
```

6 Fazit

Aufbauend auf den Grundlagen der Softwaretechnik und der Blockchain-Technologie wurden die spezifischen Anforderungen an die „Cash2Coin“-Plattform systematisch ermittelt und in ein technisches Konzept überführt. Es wurde eine modulare Softwarearchitektur entwickelt, die auf bewährten Prinzipien der Softwaretechnik basiert und eine sichere, benutzerfreundliche und skalierbare Lösung für den Bitcoin-Handel in Partnerfilialen bietet. Die Bitcoin-Kauffunktion wurde prototypisch implementiert und die realisierten Architekturentscheidungen in einer realitätsnahen Umgebung validiert. Damit kann diese Arbeit einen wichtigen Beitrag zur Gestaltung innovativer technischer Lösungen im Bereich des Kryptowährungshandels leisten und den Zugang zu Bitcoin erleichtern.

6.1 Ergebnis

Das Ergebnis dieser Arbeit ist eine modulare Softwarearchitektur für die serverseitige Anwendung der Plattform sowie die prototypische Implementierung ausgewählter Kernkomponenten.

Ergebnisse des Architekturentwurfs

Im Rahmen des Architekturentwurfs wurden die funktionalen und nicht-funktionalen Anforderungen an das Softwaresystem analysiert und relevante Einflussfaktoren identifiziert. Diese wurden in die Kategorien Produktfaktoren, Qualitätsfaktoren und technische Faktoren unterteilt. Für jeden Einflussfaktor wurden spezifische Lösungsstrategien entwickelt, die als Grundlage für die Architekturentscheidungen dienten.

Diese Arbeit konzentrierte sich auf die Entwicklung der serverseitigen Anwendung, die als zentrale Komponente der Plattform die Geschäftslogik und die Datenverarbeitung bereitstellt. Die Softwarearchitektur wurde als Drei-Schichten-Architektur konzipiert, die eine klare Trennung zwischen der Präsentationsschicht, der Anwendungsschicht und der Persistenzschicht vorsieht. Die Anwendungsschicht wurde als Sammlung unabhängiger Microservices konzipiert, die in Node.js entwickelt, in Docker-Containern bereitgestellt und mit K8s orchestriert werden. Die Kommunikation zwischen den Services erfolgt entweder synchron über GraphQL oder asynchron über Apache Kafka. Diese Architektur bietet ein hohes Maß an Flexibilität, Skalierbarkeit und Ausfallsicherheit und ermöglicht die Verarbeitung von Transaktionen in Echtzeit. Die Persistenzschicht wurde als verteilte MySQL-Datenbank mit synchroner Datenreplikation über NDB Cluster konzipiert, um eine hohe Datenverfügbarkeit und Ausfallsicherheit zu gewährleisten.

Die Software wurde in unabhängige Module unterteilt, von denen jedes eine klar definierte Aufgabe erfüllt und als Microservice implementiert wird. Ein Authentifizierungs-Modul realisiert die Benutzeranmeldung und -authentifizierung über den OAuth 2.0-Standard, während ein Autorisierungs-Modul die rollenbasierte Zugriffskontrolle mittels OPA ermöglicht.

Sensible Daten wie Passwörter und Zugriffstoken werden sicher in den K8s Secrets gespeichert und verwaltet. Ein Transaktionsverarbeitungs-Modul realisiert die Verarbeitung von Kauf- und Verkaufsaufträgen über die API des Zahlungsdienstleisters. Bitpanda Custody bietet eine Komplettlösung für den sicheren Handel und die sichere Verwahrung von Bitcoins und wurde als Zahlungsdienstleister ausgewählt. Ein Transaktionsvalidierungs-Modul und ein KYC-Modul implementieren die Regeln zur Einhaltung der MiCA-Verordnung und der AML-Richtlinien innerhalb einer Rules Engine. API-Schnittstellen zur Identitätsprüfung gemäß den AML-Richtlinien werden von Anbietern wie Faceki und Identitypass als npm-Bibliotheken zur Verfügung gestellt. Ein Datenbank-Modul kapselt den Datenbankzugriff und bietet über das ORM-Framework eine objektorientierte Schnittstelle zur Datenbank. Weitere Module wie ein Konfigurations-Modul zur zentralen Verwaltung der Systemkonfiguration, ein Monitoring-Modul auf Basis von Prometheus und Grafana zur Überwachung der Systemressourcen sowie ein Logging-Modul zur Protokollierung wichtiger Systemereignisse wurden bei den Architekturentscheidungen berücksichtigt.

Die Softwarearchitektur kann für die weitere Entwicklung einer hochverfügbaren, skalierbaren und sicheren Plattform genutzt werden, die den Anforderungen des Bitcoin-Handels gerecht wird.

Ergebnisse der Implementierung

Im Rahmen der Implementierung wurden ausgewählte Kernkomponenten der serverseitigen Anwendung prototypisch realisiert, um die Architekturentscheidungen zu validieren und die praktische Umsetzbarkeit der vorgeschlagenen Lösungen zu demonstrieren. Dabei lag der Fokus auf der Implementierung der Bitcoin-Kauffunktion in einer realitätsnahen Umgebung, um die Interaktion mit dem Zahlungsdienstleister und die technische Umsetzung der Transaktionsverarbeitung zu simulieren.

Die Produktionsumgebung wurde in einer Google Cloud eingerichtet und umfasst mehrere virtuelle Maschinen für die Bereitstellung des K8s-Clusters, des Load Balancers und eines Mock-Servers für die Bitpanda API. Die Control Plane und die Worker Nodes wurden mit kubeadm initialisiert und die Netzwerkkommunikation mit dem Calico-Netzwerk-Plugin konfiguriert und abgesichert. Der Load Balancer wurde auf Basis von NGINX konfiguriert, um den eingehenden Datenverkehr auf die verfügbaren Ressourcen im Cluster zu verteilen. Diese Konfiguration ermöglicht eine effiziente und skalierbare Bereitstellung der Anwendungen in der Cloud-Umgebung.

Die serverseitige Anwendung wurde in eigenständigen Microservices implementiert, in Docker-Containern über die Google Artifact Registry bereitgestellt und über das Deployment im K8s-Cluster in Produktion geschaltet. Die Bitcoin-Kauffunktion wurde als Microservice mit Node.js und Express.js entwickelt und stellt API-Endpunkte für die Erstellung von Kaufaufträgen, die Ausgabe der Transaktionshistorie und die Aktualisierung des Auftragsstatus zur Verfügung. Die Integration der Bitpanda API erfolgte zu Entwicklungs- und Testzwecken über eine von Bitpanda bereitgestellte OpenAPI-Spezifikation. Dazu wurde ein Mock-Server mit prism-cli

aufgesetzt und die Interaktionen mit der Bitpanda API simuliert. Diese Integration ermöglicht die Sicherstellung der Kommunikation mit dem Zahlungsdienstleister und stellt die korrekte Verarbeitung von Bitcoin-Kaufaufaufträgen sicher.

Die Benutzerauthentifizierung wurde analog zur Bitcoin-Kauffunktion als Microservice mit Node.js und Express.js entwickelt und stellt API-Endpunkte für die Benutzeranmeldung zur Verfügung. Die Autorisierung wurde über OPA als zentrale Policy Engine in den K8s-Cluster integriert, um die rollenbasierte Zugriffskontrolle zu realisieren. Die Authentifizierung erfolgte über JWT-Tokens, die einen sicheren Zugriff auf geschützte Ressourcen ermöglichen.

Für die persistente Speicherung der Benutzerdaten wurde eine MySQL-Datenbank innerhalb des K8s-Clusters eingerichtet. Dazu wurden ein Persistent Volume und ein Persistent Volume Claim konfiguriert und die Datenbank über das Deployment im K8s-Cluster bereitgestellt. Der Zugriff auf die Datenbank erfolgte über das Node.js-Modul mysql2, wobei zur Ressourcenoptimierung ein Verbindungspool eingerichtet wurde.

Um die Funktionalität und Zuverlässigkeit der implementierten Komponenten sicherzustellen, wurde eine Reihe von Tests durchgeführt. Die korrekte Funktion der Control Plane und der Worker Nodes sowie die korrekte Ausführung der Pods und Services wurden erfolgreich überprüft. Funktionstests validierten die korrekte Funktionsweise der API-Endpunkte und die Interaktion zwischen den Microservices. Die Benutzeranmeldung, die Erstellung von Kaufaufträgen und die Aktualisierung des Auftragsstatus wurden erfolgreich simuliert. Diese Tests stellen sicher, dass die Komponenten reibungslos zusammenarbeiten und die Geschäftsprozesse korrekt in den Prototypen implementiert sind.

Die Skalierbarkeit und Ausfallsicherheit der Plattform konnte im Prototyp durch die dynamische Skalierung der Pods, einer Lastverteilung über den Load Balancer und eine effiziente Ressourcennutzung optimiert werden. Der Prototyp demonstriert die Eignung der gewählten Architekturlösungen für den produktiven Einsatz und dient als Grundlage für die Weiterentwicklung der Plattform. Eine detaillierte Dokumentation der Implementierung befindet sich im Anhang dieser Arbeit und bietet Entwicklern einen umfassenden Einblick in die technischen Details der serverseitigen Anwendung. Zusätzlich bietet eine Demonstration des Prototyps auf YouTube einen praktischen Einblick in die Funktionsweise des Systems und zeigt den Einsatz der serverseitigen Anwendung beim Kauf von Bitcoins.

6.2 Ausblick

Die im Rahmen dieser Arbeit entwickelte Softwarearchitektur und die prototypische Implementierung der Kernkomponenten bilden eine solide Basis für die Weiterentwicklung der „Cash2Coin“-Plattform zu einer marktreifen Anwendung. Die nächsten Schritte umfassen die vollständige Implementierung der serverseitigen Anwendung, die Entwicklung und die Veröffentlichung der mobilen Anwendungen in den App Stores sowie die Akquisition von Partnerfilialen und Zahlungsdienstleistern. Für die Weiterentwicklung der Plattform im Team wird auf das agile Vorgehensmodell Scrum zurückgegriffen und die Softwareentwicklung in Sprints organisiert.

Eine zentrale Herausforderung stellt die Integration der Software in bestehende Kassensysteme dar. Die Implementierung der Schnittstelle zur Integration in ein handelsübliches Kassenterminal erfordert eine enge Zusammenarbeit mit dem Terminalanbieter. Eine weitere Herausforderung ist die vollständige Abwicklung von Euro-Zahlungen zwischen Partnerfilialen und Zahlungsdienstleistern. Dies erfordert sowohl technische Anpassungen als auch die Berücksichtigung regulatorischer Aspekte im Bereich der Geldwäscheprävention und steuerlicher Regelungen.

In einem dynamischen Umfeld wie dem Bitcoin-Handel ist die kontinuierliche Anpassung an aktuelle regulatorische Anforderungen, technologische Entwicklungen und Sicherheitsstandards von entscheidender Bedeutung. Bei der Wahl des Zahlungsdienstleisters und des KYC-Dienstleisters steht die Einhaltung der MiCA-Verordnung und der AML-Richtlinien im Vordergrund. Bitpanda bietet eine Komplettlösung für den sicheren Handel und die sichere Verwahrung von Bitcoins und wurde als Zahlungsdienstleister ausgewählt. Die Wahl des KYC-Dienstleisters erfolgt auf Basis der Anforderungen an die Identitätsprüfung gemäß den AML-Richtlinien.

Bei der Entwicklung einer marktreifen Anwendung mit K8s sind insbesondere die Empfehlungen der CNCF zur Sicherheit von Cloud-nativen Anwendungen zu berücksichtigen [Kub24e]. Das „Cloud Native Security Whitepaper“ der CNCF [CNC24] bietet einen umfassenden Leitfaden zur Absicherung von K8s-Clustern und zur Minimierung von Sicherheitsrisiken. Aktuelle Sicherheitsvorfälle, wie die Veröffentlichung von K8s Secrets in öffentlichen Repositories [Lak24a] und neue Angriffsmethoden auf den K8s-Cluster [Lak24b] unterstreichen die Notwendigkeit einer fortlaufenden Überprüfung und Optimierung der Sicherheitsstrategien. Diese Entwicklungen werden zukünftig noch intensiver analysiert und beobachtet, um die Sicherheit und Zuverlässigkeit der Plattform weiter zu erhöhen und den Anforderungen des Datenschutzes gerecht zu werden.

6.3 Schlusswort

Dieses Projekt bietet die Möglichkeit, innovative Ideen im Bereich des Kryptowährungshandels umzusetzen und damit die Zukunft des digitalen Zahlungsverkehrs in Deutschland aktiv mitzugestalten. Entscheidend für den Erfolg des Projekts ist die kontinuierliche Weiterentwicklung der Plattform und die Erarbeitung geeigneter Lösungen für die technischen und regulatorischen Herausforderungen. „Cash2Coin“ hat das Potenzial, die Verbreitung und Akzeptanz von Bitcoin durch vereinfachte Zugangswege und eine benutzerfreundliche Plattform zu fördern und damit einen Beitrag zur digitalen Transformation der Finanzwelt zu leisten.

Diese Arbeit bildet eine solide Grundlage für die Weiterentwicklung der Plattform und soll als Referenz für zukünftige Softwareprojekte im Bereich des Kryptowährungshandels dienen. Die strukturierte Vorgehensweise bei der Entwicklung der Softwarearchitektur und die prototypische Implementierung der Kernkomponenten bieten einen praktischen Einblick in die Herausforderungen und Lösungsansätze bei der Umsetzung einer Bitcoin-Handelsplattform.

Anhang: Artefakte der Implementierung

Die Artefakte der Implementierung und eine ausführliche Software-Dokumentation befinden sich auf der CD, die dieser Arbeit beiliegt.

Literaturverzeichnis

[1] F. Roth, „The effect of the financial crisis on systemic trust“, *Intereconomics*, Jg. 44, Nr. 4, S. 203–208, 2009, Letzter Zugriff 23.07.2024. Adresse: https://link.springer.com/chapter/10.1007/978-3-030-86024-0_11.

[2] A.-K. Bauert, *Die Finanzkrise seit 2008 im Kontext ausgewählter Marktteilnehmer: Konsequenzen und Auswirkungen auf Banken und Privatanleger*. Diplomica Verlag, 2014, S. 33, Letzter Zugriff 22.07.2024. Adresse: https://books.google.co.uk/books?hl=de&lr=&id=Jk-oAgAAQBAJ.

[3] S. Nakamoto, „Bitcoin: A peer-to-peer electronic cash system“, Techn. Ber., 2008, Letzter Zugriff 19.09.2024. Adresse: https://assets.pubpub.org/d8wct41f/31611263538139.pdf.

[4] S. Nakamoto, *Bitcoin P2P e-cash paper*, Letzter Zugriff 22.07.2024, 2008. Adresse: https://www.metzdowd.com/pipermail/cryptography/2008-October/014810.html.

[5] J. Authers, C. Giles, K. Guha und N. Hume, „Market crash shakes world“, 2008, Letzter Zugriff 22.07.2024. Adresse: https://www.ft.com/content/a0eac1e2-96f0-11dd-8cc4-000077b07658.

[6] CoinMarketCap, *Bitcoin statistics*, Letzter Zugriff 20.09.2024, 2024. Adresse: https://coinmarketcap.com/currencies/bitcoin.

[7] M. Brandt, *Deutsche Kryptobegeisterung hält sich in Grenzen*, Letzter Zugriff 22.07.2024, 2023. Adresse: https://de.statista.com/infografik/22561/anteil-der-krypto-nutzer-in-ausgewaehlten-laendern.

[8] Block Inc., „Bitcoin: Knowledge and Perceptions“, Techn. Ber., 2022, Letzter Zugriff 23.07.2024. Adresse: https://block.xyz/2022/btc-report.pdf.

[9] Binance, *How to Complete Identity Verification for a Personal Account?*, Letzter Zugriff 23.07.2024, 2019. Adresse: https://www.binance.com/en/support/faq/how-to-complete-identity-verification-for-a-personal-account-360027287111.

[10] T. Tsuchiya, A. Cuevas und N. Christin, „Identifying Risky Vendors in Cryptocurrency P2P Marketplaces“, in *Proceedings of the ACM on Web Conference 2024*, Letzter Zugriff 23.07.2024, 2024, S. 99. Adresse: https://dl.acm.org/doi/pdf/10.1145/3589334.3645475.

[11] Coin ATM Radar, *Bitcoin ATM Map*, Letzter Zugriff 23.07.2024, 2024. Adresse: https://coinatmradar.com/bitcoin-atm-map.

[12] E. Denert, *Software-Engineering: Methodische Projektabwicklung*. Springer-Verlag, 2013, S. 11, 38–44, Letzter Zugriff 23.07.2024. Adresse: https://books.google.co.uk/books?hl=de&lr=&id=7jamBgAAQBAJ.

[13] H. Balzert, *Lehrbuch der softwaretechnik: Basiskonzepte und requirements engineering*. Springer-Verlag, 2010, S. 17–22, 25, 50–51, 455–456, Letzter Zugriff 08.08.2024. Adresse: https://books.google.co.uk/books?hl=de&lr=&id=8SkgBAAAQBAJ.

[14] M. Haoues, A. Sellami, H. Ben-Abdallah und L. Cheikhi, „A guideline for software architecture selection based on ISO 25010 quality related characteristics“, *International Journal of System Assurance Engineering and Management*, Jg. 8, S. 887–888, 2017, Letzter Zugriff 22.07.2024. Adresse: https://link.springer.com/article/10.1007/s13198-016-0546-8.

[15] J. Dunkel und A. Holitschke, *Softwarearchitektur für die Praxis*. Springer-Verlag, 2013, S. 3–9, 11–15, 17–22, 73–114, Letzter Zugriff 24.08.2024. Adresse: https://books.google.co.uk/books/about/Softwarearchitektur_f%C3%BCr_die_Praxis.html?id=BT0FBgAAQBAJ&redir_esc=y.

[16] H. Balzert, *Lehrbuch der Software-Technik*. Spektrum, Akad. Verlag, 1996, S. 36, Letzter Zugriff 03.08.2024. Adresse: https://books.google.co.uk/books?id=dXcERQAACAAJ.

[17] I. Sommerville, *Software Engineering* (International Computer Science Series). Pearson, 2011, S. 5, Letzter Zugriff 02.08.2024, ISBN: 9780137053469. Adresse: https://books.google.co.uk/books?id=l0egcQAACAAJ.

[18] C. Cowell, N. Lotz und C. Timberlake, *Automating DevOps with GitLab CI/CD Pipelines: Build efficient CI/CD pipelines to verify, secure, and deploy your code using real-life examples*. Packt Publishing Ltd, 2023, S. 21–23, 57–80, 85–108, 120–121, Letzter Zugriff 27.08.2024. Adresse: https://www.packtpub.com/en-us/product/automating-devops-with-gitlab-cicd-pipelines-9781803233000.

[19] W. W. Royce, „Managing the development of large software systems (1970)“, 1970, Letzter Zugriff 08.08.2024. Adresse: https://direct.mit.edu/books/edited-volume/5003/chapter-abstract/2657056/Managing-the-Development-of-Large-Software-Systems.

[20] A. A. Adenowo und B. A. Adenowo, „Software engineering methodologies: a review of the waterfall model and object-oriented approach“, *International Journal of Scientific & Engineering Research*, Jg. 4, Nr. 7, S. 429–430, 433, 2013, Letzter Zugriff 08.08.2024. Adresse: https://www.researchgate.net/publication/344194737_Software_Engineering_Methodologies_A_Review_of_the_Waterfall_Model_and_Object-_Oriented_Approach.

[21] P. Rechenberg und G. Pomberger, *Informatik-Handbuch*. Munich, Germany: Hanser, 1997, S. 642–644, Letzter Zugriff 03.08.2024. Adresse: https://www.buecher.de/artikel/buch/informatik-handbuch/23919670.

[22] W. Zayat und O. Senvar, „Framework study for agile software development via scrum and Kanban“, *International journal of innovation and technology management*, Jg. 17, Nr. 04, S. 2–3, 2020, Letzter Zugriff 08.08.2024. Adresse: https://www.worldscientific.com/doi/abs/10.1142/S0219877020300025.

[23] V. Hema, S. Thota, S. N. Kumar, C. Padmaja, C. B. R. Krishna und K. Mahender, „Scrum: An effective software development agile tool“, in *IOP Conference Series: Materials Science and Engineering*, Letzter Zugriff 08.08.2024, IOP Publishing, Bd. 981, 2020, S. 6, 8–9. Adresse: https://iopscience.iop.org/article/10.1088/1757-899X/981/2/022060/meta.

[24] D. Garlan, „Software architecture“, S. 1–2, 2008, Letzter Zugriff 03.08.2024. Adresse: https://kilthub.cmu.edu/articles/journal_contribution/Software_Architecture/6609593?file=12101711.

[25] N. Medvidovic, D. S. Rosenblum, D. F. Redmiles und J. E. Robbins, „Modeling software architectures in the unified modeling language“, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Jg. 11, Nr. 1, S. 7–8, 2002, Letzter Zugriff 08.08.2024. Adresse: https://dl.acm.org/doi/abs/10.1145/504087.504088.

[26] H. Mili, A. Elkharraz und H. Mcheick, „Understanding separation of concerns“, *Early aspects: aspect-oriented requirements engineering and architecture design*, S. 24–27, 76–77, 2004, Letzter Zugriff 05.08.2024. Adresse: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=4b53c4af6254e7530fa4652d6fb0013680835ab1#page=76.

[27] R. Lackes und M. Siepermann, *Information Hiding*, Letzter Zugriff 08.08.2024, 2018. Adresse: https://wirtschaftslexikon.gabler.de/definition/information-hiding-37305/version-260743.

[28] C. Y. Baldwin und K. B. Clark, *Design rules, Volume 1: The power of modularity*. MIT press, 2000, S. 63–64, Letzter Zugriff 04.08.2024. Adresse: https://direct.mit.edu/books/monograph/1856/Design-Rules-Volume-1The-Power-of-Modularity.

[29] R. Laigner u. a., „From a monolithic big data system to a microservices event-driven architecture“, in *2020 46th Euromicro conference on software engineering and advanced applications (SEAA)*, Letzter Zugriff 31.08.2024, IEEE, 2020, S. 1–8. Adresse: https://ieeexplore.ieee.org/abstract/document/9226286.

[30] Amazon Web Services Inc., *Event-Driven Architecture*, Letzter Zugriff 31.08.2024, 2024. Adresse: https://aws.amazon.com/event-driven-architecture.

[31] M. Meng, S. Steinhardt und A. Schubert, „Application programming interface documentation: What do software developers want?“, *Journal of Technical Writing and Communication*, Jg. 48, Nr. 3, S. 296, 324–326, 2018, Letzter Zugriff 07.08.2024. Adresse: https://journals.sagepub.com/doi/abs/10.1177/0047281617721853.

[32] R. T. Monroe, A. Kompanek, R. Melton und D. Garlan, „Architectural styles, design patterns, and objects“, *IEEE software*, Jg. 14, Nr. 1, S. 4–5, 1997, Letzter Zugriff 08.08.2024. Adresse: https://ieeexplore.ieee.org/abstract/document/566427.

[33] D. Booth, „Web services architecture“, *W3C note*, S. 7–8, 60–63, 2004, Letzter Zugriff 24.08.2024. Adresse: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211.

[34] Statista Research Department, *Approaches used in system design worldwide in 2023*, Letzter Zugriff 05.09.2024, 2024. Adresse: https://www.statista.com/statistics/1450635/system-design-approaches.

[35] D. Taibi, V. Lenarduzzi und C. Pahl, „Architectural Patterns for Microservices: A Systematic Mapping Study.“, *Closer*, S. 221–224, 2018, Letzter Zugriff 28.08.2024. Adresse: https://www.scitepress.org/PublishedPapers/2018/67983/67983.pdf.

[36] D. Gannon, R. Barga und N. Sundaresan, „Cloud-native applications“, *IEEE Cloud Computing*, Jg. 4, Nr. 5, S. 17–20, 2017, Letzter Zugriff 25.08.2024. Adresse: https://ieeexplore.ieee.org/abstract/document/8125550.

[37] T. Cerny, M. J. Donahoo und M. Trnka, „Contextual understanding of microservice architecture: current and future directions“, *ACM SIGAPP Applied Computing Review*, Jg. 17, Nr. 4, S. 30–32, 2018, Letzter Zugriff 24.08.2024. Adresse: https://dl.acm.org/doi/abs/10.1145/3183628.3183631.

[38] A. Balalaie, A. Heydarnoori und P. Jamshidi, „Migrating to cloud-native architectures using microservices: an experience report“, in *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers 4*, Letzter Zugriff 25.08.2024, Springer, 2016, S. 2–4, 8–14. Adresse: https://link.springer.com/chapter/10.1007/978-3-319-33313-7_15.

[39] G. Brito und M. T. Valente, „REST vs GraphQL: A controlled experiment“, in *2020 IEEE international conference on software architecture (ICSA)*, Letzter Zugriff 05.09.2024, IEEE, 2020, S. 2–3, 10. Adresse: https://ieeexplore.ieee.org/abstract/document/9101226.

[40] Postman Inc., *2023 State of the API Report*, Letzter Zugriff 08.08.2024, 2023. Adresse: https://www.postman.com/state-of-api/api-technologies/#architectural-style-soap-slips.

[41] L. Kamiński, M. Kozłowski, D. Sporysz, K. Wolska, P. Zaniewski und R. Roszczyk, „Comparative review of selected Internet communication protocols“, *Foundations of Computing and Decision Sciences*, Jg. 48, Nr. 1, S. 1, 2023, Letzter Zugriff 08.08.2024. Adresse: https://sciendo.com/article/10.2478/fcds-2023-0003.

[42] The GraphQL Foundation, *GraphQL*, Letzter Zugriff 08.08.2024, 2024. Adresse: https://graphql.org.

[43] A. M. Potdar, D. Narayan, S. Kengond und M. M. Mulla, „Performance evaluation of docker container and virtual machine“, *Procedia Computer Science*, Jg. 171, S. 1419–1428, 2020, Letzter Zugriff 05.09.2024. Adresse: https://www.sciencedirect.com/science/article/pii/S1877050920311315.

[44] Docker Inc., *Docker Engine overview*, Letzter Zugriff 25.08.2024, 2024. Adresse: https://docs.docker.com/engine.

[45] Y. Pan, I. Chen, F. Brasileiro, G. Jayaputera und R. Sinnott, „A performance comparison of cloud-based container orchestration tools“, in *2019 IEEE International Conference on Big Knowledge (ICBK)*, Letzter Zugriff 26.08.2024, IEEE, 2019, S. 1–8. Adresse: https://ieeexplore.ieee.org/abstract/document/8944745.

[46] N. Poulton, *The kubernetes book*. NIGEL POULTON LTD, 2020, S. 6–8, 11–26, 34, 138–154, Letzter Zugriff 03.09.2024. Adresse: https://leanpub.com/thekubernetesbook.

[47] Docker Inc., *Swarm mode overview*, Letzter Zugriff 26.08.2024, 2024. Adresse: https://docs.docker.com/engine/swarm.

[48] Docker Inc., *Swarm mode key concepts*, Letzter Zugriff 26.08.2024, 2024. Adresse: https://docs.docker.com/engine/swarm.

[49] Kubernetes, *Overview*, Letzter Zugriff 27.08.2024, 2023. Adresse: https://kubernetes.io/docs/concepts/overview.

[50] Oracle, *Licensing Information*, Letzter Zugriff 27.08.2024, 2024. Adresse: https://docs.oracle.com/en/database/other-databases/timesten/22.1/licensing/kubernetes.html.

[51] Kubernetes, *Kubernetes Components*, Letzter Zugriff 27.08.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/overview/components.

[52] Kubernetes, *kubelet*, Letzter Zugriff 28.08.2024, 2024. Adresse: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet.

[53] Kubernetes, *Container Runtimes*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://kubernetes.io/docs/setup/production-environment/container-runtimes.

[54] Kubernetes, *Ingress*, Letzter Zugriff 29.08.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/services-networking/ingress.

[55] L. S. Vailshery, *Organizations' adoption level of Kubernetes worldwide in 2022*, Letzter Zugriff 05.09.2024, 2023. Adresse: https://www.statista.com/statistics/1233945/kubernetes-adoption-level-organization.

[56] A. Sy Kim, M. Au, W. Fender und M. McCune, *Completing the largest migration in Kubernetes history*, Letzter Zugriff 28.08.2024, 2024. Adresse: https://kubernetes.io/blog/2024/05/20/completing-cloud-provider-migration.

[57] Google, *Übersicht über GKE*, Letzter Zugriff 29.08.2024, 2024. Adresse: https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview.

[58] F5 Inc., *About | NGINX Ingress Controller*, Letzter Zugriff 30.08.2024, 2024. Adresse: https://docs.nginx.com/nginx-ingress-controller/overview/about.

[59] R. Amir, *Managing Kubernetes Traffic with F5 NGINX*. 2022, S. 29–71, Letzter Zugriff 30.08.2024. Adresse: https://tech-prospect.com/software/managing-kubernetes-traffic-with-f5-nginx.

[60] Google, *GKE Ingress für Application Load Balancer*, Letzter Zugriff 30.08.2024, 2024. Adresse: https://cloud.google.com/kubernetes-engine/docs/concepts/ingress.

[61] A. MUSTYALA, „Comprehensive Monitoring and Logging in Kubernetes: Best Practices and Tools", *EPH-International Journal of Science And Engineering*, Jg. 8, Nr. 4, S. 05–14, 2022, Letzter Zugriff 31.08.2024. Adresse: https://ephijse.com/index.php/SE/article/view/235.

[62] A. Arnqvist, *Evaluating Failover and Recovery of Replicated SQL Databases*, Letzter Zugriff 03.09.2024, 2023. Adresse: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1736889&dswid=4225.

[63] Oracle, *Chapter 19 Replication*, Letzter Zugriff 02.09.2024, 2024. Adresse: https://dev.mysql.com/doc/refman/8.4/en/replication.html.

[64] A. Davies und H. Fisk, *MySQL clustering*. Sams Publishing, 2006, S. 1–3, Letzter Zugriff 02.09.2024. Adresse: https://books.google.de/books/about/MySQL_Clustering.html?id=_3nERYD9xqcC.

[65] Oracle, *Differences Between the NDB and InnoDB Storage Engines*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://dev.mysql.com/doc/refman/8.4/en/mysql-cluster-ndb-innodb-engines.html.

[66] Oracle, *Group Replication Background*, Letzter Zugriff 02.09.2024, 2024. Adresse: https://dev.mysql.com/doc/refman/8.4/en/group-replication-background.html.

[67] Oracle, *InnoDB Cluster*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://dev.mysql.com/doc/refman/8.4/en/mysql-innodb-cluster-introduction.html.

[68] Oracle, *NDB Cluster Overview*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://dev.mysql.com/doc/refman/8.4/en/mysql-cluster-overview.html.

[69] F. Tschorsch und B. Scheuermann, „Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies“, *IEEE Communications Surveys & Tutorials*, Jg. 18, Nr. 3, S. 2084–2123, 2016, Letzter Zugriff am 19.09.2024. Adresse: https://ieeexplore.ieee.org/abstract/document/7423672.

[70] R. C. Merkle, „A Digital Signature Based on a Conventional Encryption Function“, in *Advances in Cryptology – CRYPTO '87*, Ser. Lecture Notes in Computer Science, Letzter Zugriff am 19.09.2024, Bd. 293, Springer, 1988, S. 369–378. Adresse: https://link.springer.com/chapter/10.1007/3-540-48184-2_32.

[71] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll und E. W. Felten, „SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies“, in *2015 IEEE Symposium on Security and Privacy*, Letzter Zugriff am 19.09.2024, IEEE, 2015, S. 115. Adresse: https://ieeexplore.ieee.org/document/7163021.

[72] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7. Aufl. Pearson, 2017, Letzter Zugriff am 19.09.2024, ISBN: 978-0134444284. Adresse: https://focusonedesign.ca/product/cryptography-network-security.

[73] C. Dwork und M. Naor, „Pricing via Processing or Combatting Junk Mail“, in *Advances in Cryptology – CRYPTO'92*, Ser. Lecture Notes in Computer Science, Letzter Zugriff am 20.09.2024, Bd. 740, Springer, 1992, S. 139–147. Adresse: https://link.springer.com/chapter/10.1007/3-540-48071-4_10.

[74] Bitpanda Custody, *Manage your Clients Cryptoassets Efficiently with TrustVault*, Letzter Zugriff 19.09.2024, 2022. Adresse: https://custody.bitpanda.com/insights-events/trustvault.-the-safest-crypto-account-for-institutional-investors-0.

[75] D. Dasgupta, J. M. Shrein und K. D. Gupta, „A survey of blockchain from security perspective“, *Journal of Banking and Financial Technology*, Jg. 3, S. 8, 2019, Letzter Zugriff 19.09.2024. Adresse: https://link.springer.com/article/10.1007/s42786-018-00002-6.

[76] BaFin, *Know Your Customer (KYC)*, Letzter Zugriff 20.09.2024, 2018. Adresse: https://www.bafin.de/ref/19629780.

[77] Europäische Union, *RICHTLINIE (EU) 2018/843 DES EUROPÄISCHEN PARLAMENTS UND DES RATES, zur Änderung der Richtlinie (EU) 2015/849 zur Verhinderung der Nutzung des Finanzsystems zum Zwecke der Geldwäsche und der Terrorismusfinanzierung und zur Änderung der Richtlinien 2009/138/EG und 2013/36/EU*, Letzter Zugriff 20.09.2024, 2018. Adresse: https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32018L0843.

[78] Europäische Union, *VERORDNUNG (EU) 2023/1114 DES EUROPÄISCHEN PARLAMENTS UND DES RATES, über Märkte für Kryptowerte und zur Änderung der Verordnungen (EU) Nr. 1093/2010 und (EU) Nr. 1095/2010 sowie der Richtlinien 2013/36/EU und (EU) 2019/1937*, Letzter Zugriff 20.09.2024, 2023. Adresse: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX:32023R1114.

[79] BaFin, *Kryptoverwahrgeschäft*, Letzter Zugriff 01.04.2024, 2022. Adresse: https://www.bafin.de/DE/Aufsicht/FinTech/Geschaeftsmodelle/DLT_Blockchain_Krypto/Kryptoverwahrgeschaeft/Kryptoverwahrgeschaeft_node.html.

[80] I. D. GmbH, *Analysemodell*, Letzter Zugriff 22.07.2024, 2021. Adresse: https://www.ibm.com/docs/de/rsm/7.5.0?topic=model-analysis.

[81] V. Müller, *Die Rolle der BaFin im Kampf gegen Geldwäsche und Terrorismusfinanzierung*, Letzter Zugriff 31.08.2024, 2018. Adresse: https://www.bafin.de/ref/19649752.

[82] Bundesministeriums der Justiz, *Bundesdatenschutzgesetz (BDSG)*, Letzter Zugriff 31.08.2024, 2017. Adresse: https://www.gesetze-im-internet.de/bdsg_2018/BDSG.pdf.

[83] A. Ansari, *Was bedeutet eigentlich DSGVO-konform?*, Letzter Zugriff 30.08.2024, 2022. Adresse: https://www.zess.uni-goettingen.de/mediendidaktik/2022/05/25/was-bedeutet-eigentlich-dsgvo.

[84] B. Köstler, *Bitcoin und Steuern: Steuerspielregeln für den Verkauf von Kryptowährungen*, Letzter Zugriff 30.08.2024, 2024. Adresse: https://www.steuern.de/bitcoin-steuer.

[85] Chainalysis Team, *2024 Crypto Crime Mid-year Update Part 1: Cybercrime Climbs as Exchange Thieves and Ransomware Attackers Grow Bolder*, Letzter Zugriff 30.08.2024, 2024. Adresse: https://www.chainalysis.com/blog/2024-crypto-crime-mid-year-update-part-1.

[86] R. Chaganti u. a., „A comprehensive review of denial of service attacks in blockchain ecosystem and open challenges“, *IEEE Access*, Jg. 10, S. 96 542–96 543, 96545, 96548–96 551, 2022, Letzter Zugriff 31.08.2024. Adresse: https://ieeexplore.ieee.org/abstract/document/9881505.

[87] Meta Platforms, *Cross Platform Implementation*, Letzter Zugriff 01.06.2024, 2022. Adresse: https://reactnative.dev/architecture/xplat-implementation.

[88] G. Bierman, M. Abadi und M. Torgersen, „Understanding typescript“, in *ECOOP 2014–Object-Oriented Programming: 28th European Conference, Uppsala, Sweden, July 28–August 1, 2014. Proceedings 28*, Letzter Zugriff 01.06.2024, Springer, 2014, S. 257–281. Adresse: https://users.soe.ucsc.edu/~abadi/Papers/FTS-submitted.pdf.

[89] Meta Open Source, *React*, Letzter Zugriff 02.09.2024, 2024. Adresse: https://react.dev.

[90] lukekarrys, *About npm*, Letzter Zugriff 01.07.2024, 2023. Adresse: https://docs.npmjs.com/about-npm.

[91] A. Isaiah, *11 Best Practices for Logging in Node.js*, Letzter Zugriff 05.06.2024, 2023. Adresse: https://betterstack.com/community/guides/logging/nodejs-logging-best-practices.

[92] SendGrid, *Email API*, Letzter Zugriff 10.06.2024, 2024. Adresse: https://sendgrid.com/en-us/solutions/email-api.

[93] Mailchimp, *Transactional API Quick Start*, Letzter Zugriff 10.06.2024, 2024. Adresse: https://mailchimp.com/developer/transactional/guides/quick-start.

[94] Google, *feedbackSet up a JavaScript Firebase Cloud Messaging client app*, Letzter Zugriff 08.07.2024, 2024. Adresse: https://firebase.google.com/docs/cloud-messaging/js/client.

[95] OneSignal, *Node Client SDK*, Letzter Zugriff 15.06.2024, 2024. Adresse: https://documentation.onesignal.com/docs/node-client-sdk.

[96] i18next, *Supported Frameworks*, Letzter Zugriff 01.07.2024, 2024. Adresse: https://www.i18next.com/overview/supported-frameworks.

[97] i18next, *i18next documentation*, Letzter Zugriff 01.07.2024, 2024. Adresse: https://www.i18next.com/#complete-solution.

[98] D. Schwarz, *Inmitten des Krypto-Crashs: Bitpanda erhält Bafin-Lizenz*, Letzter Zugriff 16.07.2024, 2022. Adresse: https://www.handelsblatt.com/finanzen/banken-versicherungen/banken/wiener-krypto-start-up-inmitten-des-krypto-crashs-bitpanda-erhaelt-bafin-lizenz/28816660.html.

[99] Bitpanda Custody, *Features*, Letzter Zugriff 02.09.2024, 2022. Adresse: https://custody.bitpanda.com/en/features.

[100] Bitpanda Custody, *trustvault-nodejs-sdk*, Letzter Zugriff 02.09.2024, 2024. Adresse: https://github.com/Trustology/trustvault-nodejs-sdk.

[101] Google, *Firebase Realtime Database*, Letzter Zugriff 05.09.2024, 2024. Adresse: https://firebase.google.com/docs/database.

[102] cachecontrol, *json-rules-engine*, Letzter Zugriff 12.07.2024, 2023. Adresse: https://www.npmjs.com/package/json-rules-engine.

[103] Kubernetes, *Cloud Native Security and Kubernetes*, Letzter Zugriff 20.09.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/security/cloud-native-security.

[104] Kubernetes, *Security*, Letzter Zugriff 12.07.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/security.

[105] Kubernetes, *Pod Security Standards*, Letzter Zugriff 13.07.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/security/pod-security-standards.

[106] Kubernetes, *Network Policies*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/services-networking/network-policies.

[107] Kubernetes, *Security Checklist*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://kubernetes.io/docs/concepts/security/security-checklist.

[108] A. Parecki, *OAuth 2.0*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://oauth.net/2.

[109] Open Policy Agent, *Open Policy Agent | Documentation*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://www.openpolicyagent.org/docs/latest.

[110] Open Policy Agent, *Open Policy Agent | Overview & Architecture*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://www.openpolicyagent.org/docs/latest/kubernetes-introduction.

[111] npm-compare, *Which is Better Node.js ORM Libraries?*, Letzter Zugriff 03.09.2024, 2024. Adresse: https://npm-compare.com/bookshelf,objection,sequelize,typeorm.

[112] Kubernetes, *Manage TLS Certificates in a Cluster*, Letzter Zugriff 12.07.2024, 2023. Adresse: https://kubernetes.io/docs/tasks/tls/managing-tls-in-a-cluster.

[113] Red Hat Inc., *Why run Apache Kafka on Kubernetes?*, Letzter Zugriff 03.09.2024, 2022. Adresse: https://www.redhat.com/en/topics/integration/why-run-apache-kafka-on-kubernetes.

[114] Kubernetes, *Auditing*, Letzter Zugriff 13.07.2024, 2023. Adresse: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit.

[115] A. Chokalingam, *How To Monitor Kubernetes Audit Logs*, Letzter Zugriff 13.07.2024, 2024. Adresse: https://logrhythm.com/blog/how-to-monitor-kubernetes-audit-logs.

[116] Prometheus, *Alertmanager | Prometheus*, Letzter Zugriff 10.07.2024, 2024. Adresse: https://prometheus.io/docs/alerting/latest/alertmanager.

[117] kichiyaki, *@kichiyaki/react-native-barcode-generator*, Letzter Zugriff 04.09.2024, 2021. Adresse: https://www.npmjs.com/package/@kichiyaki/react-native-barcode-generator.

[118] i18next, *Formatting | i18next documentation*, Letzter Zugriff 07.07.2024, 2022. Adresse: https://www.i18next.com/translation-function/formatting#built-in-formats.

[119] Open Exchange Rates Ltd., *API Libraries & Extensions*, Letzter Zugriff 04.09.2024, 2022. Adresse: https://docs.openexchangerates.org/reference/api-libraries-extensions.

[120] Amazon Web Services Inc., *Blue/Green Deployments on AWS*, Letzter Zugriff 11.07.2024, 2021. Adresse: https://docs.aws.amazon.com/pdfs/whitepapers/latest/blue-green-deployments/blue-green-deployments.pdf.

[121] Apple Inc., *App Review Guidelines*, Letzter Zugriff 15.07.2024, 2024. Adresse: https://developer.apple.com/app-store/review/guidelines.

[122] Google, *Developer Programme Policy*, Letzter Zugriff 15.07.2024, 2024. Adresse: https://support.google.com/googleplay/android-developer/answer/14906471.

[123] faceki, *Faceki Identity Verification & Biometric Authentication*, Letzter Zugriff 15.07.2024, 2024. Adresse: https://github.com/faceki.

[124] prembly, *Identitypass*, Letzter Zugriff 15.07.2024, 2024. Adresse: https://github.com/prembly.

[125] Google, *Cloud-Standorte*, Letzter Zugriff 04.09.2024, 2024. Adresse: https://cloud.google.com/about/locations.

[126] Google, *What is Compute Engine? Use cases, security, pricing and more*, Letzter Zugriff 18.09.2024, 2021. Adresse: https://cloud.google.com/blog/topics/developers-practitioners/what-compute-engine-use-cases-security-pricing-and-more.

[127] Google, *Work with container images*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://cloud.google.com/artifact-registry/docs/docker.

[128] Google, *VM instance pricing*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://cloud.google.com/compute/vm-instance-pricing.

[129] Ubuntu Wiki, *Releases*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://wiki.ubuntu.com/Releases.

[130] Docker Inc., *Install Docker Engine on Ubuntu*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://docs.docker.com/engine/install/ubuntu.

[131] Kubernetes, *Installing kubeadm, kubelet and kubectl*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl.

[132] Kubernetes, *Production environment*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://kubernetes.io/docs/setup/production-environment.

[133] G. Budigiri, C. Baumann, J. T. Mühlberg, E. Truyen und W. Joosen, „Network Policies in Kubernetes: Performance Evaluation and Security Analysis“, in *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, Letzter Zugriff 18.09.2024, 2021, S. 407–412. Adresse: https://ieeexplore.ieee.org/abstract/document/9482526.

[134] Tigera Inc., *calico-kube-controllers.yaml*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://calico-v3-25.netlify.app/archive/v3.25/manifests/calico.yaml.

[135] Microsoft, *Visual Studio Code*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://code.visualstudio.com.

[136] Microsoft, *Remote Development using SSH*, Letzter Zugriff 18.09.2024, 2024. Adresse: https://code.visualstudio.com/docs/remote/ssh.

[137] Software Freedom Conservancy, *git*, Letzter Zugriff 19.09.2024, 2024. Adresse: https://git-scm.com.

[138] GeeksforGeeks, *Bare Repositories in Git*, Letzter Zugriff 19.09.2024, 2019. Adresse: https://www.geeksforgeeks.org/bare-repositories-in-git.

[139] OpenJS Foundation, *Express*, Letzter Zugriff 19.09.2024, 2024. Adresse: https://expressjs.com/de.

[140] Bitpanda GmbH, *openapi.json*, Letzter Zugriff 19.09.2024, 2024. Adresse: https://techsolutions.bitpanda.com/download/openapi.json.

[141] stoplight, *prism-cli*, Letzter Zugriff 19.09.2024, 2024. Adresse: https://www.npmjs.com/package/@stoplight/prism-cli.

[142] M. B. Jones, J. Bradley und N. Sakimura, *JSON Web Token (JWT)*, RFC 7519, Letzter Zugriff 19.09.2024, 2015. DOI: 10.17487/RFC7519. Adresse: https://www.rfc-editor.org/info/rfc7519.

[143] Open Policy Agent, *Policy Language*, Letzter Zugriff 19.09.2024, 2024. Adresse: https://www.openpolicyagent.org/docs/latest/policy-language.

[144] A. Sidorov, *MySQL2*, Letzter Zugriff 19.09.2024, 2024. Adresse: https://sidorares.github.io/node-mysql2/docs.

[145] CNCF, „Cloud Native Security Whitepaper“, Techn. Ber., 2024, Letzter Zugriff 20.09.2024. Adresse: https://github.com/cncf/tag-security/blob/main/community/resources/security-whitepaper/v2/CNCF_cloud-native-security-whitepaper-May2022-v2.pdf.

[146] R. Lakshmanan, *Kubernetes Secrets of Fortune 500 Companies Exposed in Public Repositories*, Letzter Zugriff 29.08.2024, 2024. Adresse: https://thehackernews.com/2023/11/kubernetes-secrets-of-fortune-500.html.

[147] R. Lakshmanan, *Researchers Uncover TLS Bootstrap Attack on Azure Kubernetes Clusters*, Letzter Zugriff 29.08.2024, 2024. Adresse: https://thehackernews.com/2024/08/researchers-uncover-tls-bootstrap.html.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei meiner Masterarbeit unterstützt haben. Mein besonderer Dank gilt Prof. Dr.-Ing. Andreas Ittner und meinem Betreuer M.Sc. Tim Käbisch für die fachliche Unterstützung und die wertvollen Ratschläge. Ohne ihre Unterstützung wäre diese Arbeit in dieser Form nicht möglich gewesen.

Eidesstattliche Erklärung

Hiermit versichere ich – André Schild – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Mittweida, 22.09.2024 [redacted]

Ort, Datum André Schild