Building a Cryptocurrency Payment Provider

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühlingssemester 2022

Autor(en):	Armend Lesi und Marco Endres
Betreuer:	Dr. Thomas Bocek
Experte:	Dr. Guilherme Sperb Machado
Gegenleser:	Prof. Stefan Richter

Abstract

Die Blockchain Technologie ermöglicht das Entwickeln von Kryptowährungen. Immer mehr Menschen nutzen eine Kryptowährung als Investitionsmittel oder als Zahlungsmittel. Das Ziel dieser Arbeit ist es, einen Provider für Kryptowährungen zu entwickeln. Dabei wird untersucht, wie Ethereum und Bitcoin angebunden werden können. Durch diese Arbeit sollen Händler die Möglichkeit besitzen, Zahlungen in Ethereum und Bitcoin zu akzeptieren. Bitcoin und Ethereum wurden als initiale Kryptowährungen dieses Payment Providers implementiert. Wenn es um Geldtransaktionen geht, ist neben der Funktionalität die Sicherheit und Stabilität ein wichtiges Thema. Aus diesem Grund wurde ebenfalls Zeit in die Erarbeitung einer Lösung investiert, welche gegen Missbrauch geschützt ist und bei Teilausfällen des Systems weiterhin valide Daten besitzt. Der Payment Provider mit einer Ethereum und Bitcoin Integration wurde jeweils an das Mainnet und Testnet angebunden.

Management Summary

Kryptowährungen werden auf der ganzen Welt immer häufiger benutzt, nicht nur als Investitionsmittel, sondern auch als Zahlungsmittel. Zum Beispiel können bei einem grossen Elektronikhändler die Waren oder beim Kanton Zug die Steuern mit Kryptowährungen bezahlt werden.

Händler haben dabei das Ziel möglich viel zu verkaufen. Dabei spielt das Zahlungsmittel eine wichtige Rolle. Je mehr Möglichkeiten zur Verfügung stehen, desto wahrscheinlicher ist ein Verkauf. Aus diesem Grund wurde in dieser Arbeit ein Minimal Viable Product von einem Payment Provider für Kryptowährungen entwickelt, damit Händler Zahlungen in Kryptowährungen akzeptieren können.

Für den Minimal Viable Product wurden Anforderungen von potenziellen Kunden gesammelt. Die Einfachheit und Benutzerfreundlichkeit hatten dabei eine hohe Priorität. Es wurden Ethereum und Bitcoin als erste Kryptowährungen angebunden, weil sie die grösste Marktkapitalisierung besitzen. Die Anbindung an das Produktive- sowie das Testnetz der Blockchains war ebenfalls wichtig, um den Händlern die Möglichkeit anzubieten, den Service zu testen, ohne echte Kryptowährungen zu bezahlen. Ein Fehler bei Geldtransaktionen kann schnell zu Misstrauen und grossem Umsatzverlust führen. Deshalb wurde bei der Entwicklung zusätzlich Zeit in die Erarbeitung einer Lösung investiert, die gegen Missbrauch geschützt ist und bei Teilausfällen des Systems weiterhin valide Daten besitzt.

In der nachfolgenden Abbildung 1 wird gezeigt, wie der Service das Geld von Käufern erhält und an die Händler weiterleitet.

Abbildung 1: CHainGate Übersicht

Die Entwicklung konnte erfolgreich abgeschlossen werden. Dabei wurden Erfahrungen gesammelt, wie ein möglicher Service aussieht und welche Funktionalität angeboten werden könnte. Der nächste Schritt wäre es, eine genaue Marktanalyse zu erstellen, um herauszufinden, in welche Richtung sich der Service entwickeln muss.

Aktuell wird in der einbezahlten Kryptowährung ausbezahlt. Eine mögliche Erweiterung wäre es Auszahlungen in einen Stablecoin oder Fiatgeld anzubieten. Diese Option können Kunden nutzen, welche eine stabilere Währung wollen, jedoch Bitcoin und Ethereum als Zahlungsmittel ihren Kunden anbieten möchten.

Inhaltsverzeichnis

1. Einleitung und Übersicht

1.1 Ausgangslage

In der Semesterarbeit «Kryptowährungen als Zahlungsmittel bei FlatFeeStack» wurde NOWPayments als Provider für Kryptowährungen eingesetzt. Der Service besitzt Einschränkungen im Bereich der Integration von Testnetzen. Dies hat zur Folge, dass bei einem Systemtest von FlatFeeStack eine Zahlung mit einer echten Kryptowährung getätigt werden muss. Durch diese Einschränkung wurde entschieden, in der vorliegenden Bachelorarbeit einen eigenen Provider für Kryptowährungen zu entwickeln. Dadurch ist die Idee von CHainGate entstanden.

1.2 Motivation

Beim Onlineeinkauf ist der Bezahlprozess sehr entscheidend. 72 % aller Benutzer besitzen ein präferiertes Zahlungsmittel und 11 % wollen ausschliesslich nur ein bestimmtes Zahlungsmittel verwenden. [1]

Die unterschiedlichen Zahlungsmittel haben sich in den letzten Jahren stetig weiterentwickelt. Während Bezahlapplikationen im Jahr 2017 noch eine Randerscheinung waren, sind sie im Jahre 2020 bereits eine ernstzunehmende Alternative geworden. Obwohl beispielsweise Twint erst 8 Jahre alt ist, nennen laut einem Bericht der SNB aus dem Jahre 2020 bereits 69 % aller Befragten Twint als mögliches Zahlungsmittel.[2]

Auch Kryptowährungen wurden in den letzten Jahren als Zahlungsmittel verwendet, jedoch nicht sehr häufig. Der grösste Schweizer Onlinehändler Digitec bezeichnete vor einem Jahr bereits pro Woche etwa 200 Käufe mit Kryptowährungen, Tendenz steigend. [3, 4]

Zudem haben Regionen wie Lugano oder sogar Länder wie Panama Bitcoin als Zahlungsmittel akzeptiert. Der Kanton Zug hat in einem Mail mitgeteilt, dass sie für die Steuern im letzten Jahr 103 Transaktionen im Wert von 800'000 CHF in Kryptowährung erhalten haben. [5, 6]

Neben den Gründen für ein alternatives Zahlungsmittel gab es auch den Anspruch, ein besseres Produkt zu entwickeln als NOWPayments und die Problemstellungen eines Providers für Kryptowährungen kennenzulernen. Dadurch entsteht auch ein besseres Verständnis für die Limitationen der bestehenden Payment Provider.

1.3 Zielsetzung

Das Ziel dieser Bachelorarbeit ist es, einen Provider für Kryptowährungen zu entwickeln. Es soll ein erster Minimal Viable Product (MVP) mit den wichtigsten Funktionalitäten entwickelt werden. Die Anforderungen des Services werden im Kapitel 3.1 Anforderungen weiter erläutert. In dieser Arbeit wird dieser Provider für Kryptowährungen «CHainGate» genannt.

Abgrenzung

Folgende Punkte sind **nicht** Teil dieser Arbeit:

- Ablösung von NOWPayments bei FlatFeeStack
- Deployment auf einem Server

1.4 Unterschiede zu traditionellen Bezahlsystemen

Im Folgenden werden auf die Unterschiede zwischen einem Payment Provider in der Fiat-Welt und der Krypto-Welt eingegangen. Als Fiatgeld wird die nationale Währung bezeichnet. Zum Beispiel der Schweizer Franken, der Euro oder der US-Dollar. Für den Vergleich wurden zwei der grössten Payment Provider auf dem Markt untersucht: **Stripe** und **PayPal**.

1.4.1 Preis

Der Preis für Händler ist bei PayPal und Stripe ähnlich. Für eine Transaktion wird bei PayPal 2.9 % des Preises + 30 Cent und bei Stripe 2.8 % + 30 Cent bezahlt.

CHainGate bezieht ebenfalls eine prozentuale Gebühr pro Transaktion. Diese können durch Automatisierung des Prozesses niedrig gehalten werden. Zusätzlich entstehen Transaktionskosten bei der Nutzung der Blockchain. Diese Kosten sind abhängig von der Auslastung der Blockchain oder der Transaktionsgrösse. Im Gegensatz zur Gebühr von Stripe, PayPal oder CHainGate, sind diese Transaktionskosten unabhängig vom Transaktionsbetrag. Das bedeutet, dass bei kleinen Transaktionen die Kosten prozentual mehr zu Buche schlagen. [7]

Kryptowährungen hatten in der Vergangenheit durchschnittlich tiefere Kosten für Transaktionen als andere Payment Provider für Fiatgeld. Diese Kosten haben sich jedoch in den letzten Jahren bei manchen Blockchains durch den steigenden Bedarf geändert. Die momentane durchschnittliche Transaktionsgebühr bei Ethereum liegt beispielsweise bei etwa 5 Dollar. [8, 9]

1.4.2 Vertrauen

Eine grosse Frage zu einem Service ist, warum CHainGate vertraut werden soll. Es gibt bei einem Geldgeschäft zwei Arten von Vertrauen: **Vertrauen in Personen und Organisationen** und **Vertrauen in Systeme** [10]

Zusätzlich muss darauf geachtet werden, dass die Sicherheit das Nutzererlebnis nicht zu sehr beeinträchtigt.

Vertrauen in Personen und Organisationen

CHainGate präsentiert sich so transparent wie möglich, um ein solches Vertrauen zu schaffen. Jedoch kann nie ein Vertrauen ohne Chancen in diesem Bereich aufgebaut werden. Deshalb wird auch versucht diverse Ausfälle im System zu berücksichtigen. [11]

Vertrauen in Systeme

Weil sich CHainGate auf Kryptowährungen fokussiert hat und diese auf einem Konsensalgorithmus basieren, entfällt die Bank als Vertrauensinstanz. Kryptowährungen haben über die Jahre mehr an Vertrauen gewonnen und gezeigt, dass das System dahinter funktioniert.

Durch die öffentliche Natur der Blockchain können dem Benutzer zusätzliche Informationen gezeigt werden. Beispielweise kann ein Benutzer sein Geld und die getätigten Transaktionen verfolgen, um nachzuvollziehen, wohin dieses geht.

1.4.3 Sensible Informationen

Ein weiterer Unterschied zwischen normalen Bezahlsystemen und einer Zahlung mit Kryptowährung ist der Umgang mit sensiblen Inhalten. Beispielsweise müssen bei einer Zahlung mit Stripe sensible Daten wie Kreditkartennummer, CVV und der richtige Name angegeben werden. Bei einer Zahlung mit einer Kryptowährung hingegen ist nur eine Empfängeradresse und ein zu bezahlender Betrag notwendig. Ein Kunde kann dabei über ein Wallet seines Vertrauens Geld auf die Adresse überweisen. Dies bietet folgende Vor- und Nachteile:

+ Es braucht keine Auflagen, wie sensible Informationen gespeichert werden müssen.
+ Es kann nicht mehr Geld als der Benutzer will abgebucht werden.
+ Nutzer können das Tool verwenden, welches sie wollen, um die Zahlung zu tätigen.
+ Die Adresse selbst kann beobachtet werden, um zu sehen, was mit dem Geld passiert.
− Es gibt keine einfache Möglichkeit, ein Abonnement Modell zu erstellen, welches automatisch monatlich Geld abbucht.
− Nutzer benötigen ein 3rd-Party Applikation.

2. Theorie

In dieser Arbeit werden die zwei Blockchains **Ethereum** und **Bitcoin** im Detail behandelt.

Um auf die später erwähnten Konzepte und Implementationen besser eingehen zu können, werden in diesem Kapitel einige allgemeine Theorien und die Eigenheiten der Blockchains erläutert.

In der Welt von Kryptowährungen gibt es viele neue Konzepte, welche grundlegend für die Arbeit mit einer Blockchain sind. Die folgenden fünf waren für diese Arbeit am bedeutsamsten.

1. Das Geld wird in sogenannten **Wallets** / **Adressen** gehalten und diese können dynamisch generiert werden.
2. Überweisungen werden über **Transaktionen** getätigt. Diese Transaktionen werden mit dem Private Key der Adresse signiert.
3. Jede Transaktion verursacht **Transaktionskosten** auch **Fees** genannt. Diese sind abhängig von der Blockchain und deren Auslastung.
4. Eine Blockchain ist eine Kette von Blöcken und jeder Block basiert auf dem vorherigen. In diesen Blöcken wird Geld mit Transaktionen von der Adresse A zu der Adresse B gesendet.
5. Für jede Blockchain gibt es eine gewünschte Blockzeit. Jede Blockchain versucht nach Ablauf der Blockzeit einen neuen Block mit **X** Transaktionen darin aufzunehmen**.**

2.1 Blockchain Einführung

Die Blockchain ist eine verteilte Datenbank. Dafür benötigt es ein Netzwerk aus verschiedenen Computern, auch Nodes genannt, welche die Informationen der Datenbank miteinander teilen. Eine der wichtigsten Eigenschaften der Blockchain ist, dass sie Vertrauen ohne eine Drittpartei aufbaut. Dieses Vertrauen wird durch den Konsensalgorithmus erreicht. Ebenfalls sind die Daten nur schwer manipulierbar. [12]

Die nachfolgende Abbildung 2 zeigt eine vereinfachte Version einer Blockchain. Ein Block besteht aus einem Header und einem Block mit einer oder mehreren Transaktionen. Im Header ist immer der Hash vom vorherigen Block gespeichert. Dadurch sind die Blöcke miteinander verkettet. Diese Verkettung stellt sicher, dass Blöcke in der Blockchain nicht ohne grossen Aufwand modifiziert werden können. Um eine Modifikation durchzuführen, müssen auch alle nachfolgenden Blöcke modifiziert werden, damit die Hashes wieder übereinstimmen und eine korrekte Kette gebildet wird. Das Aufnehmen einer Transaktion in einen Block wird «schürfen» oder «minen» genannt. [13]

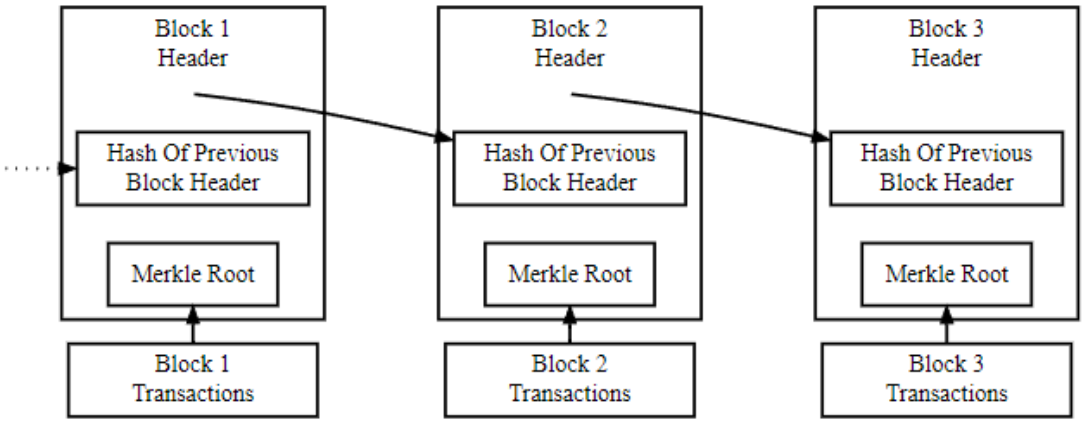

Abbildung 2: Vereinfachte Bitcoin-Blockchain

Alle Transaktionen werden in einem Block gespeichert. Zusätzlich wird der Merkle Root im Header gespeichert. Die nachfolgende Abbildung 3 zeigt, wie ein Merkle Tree aussieht und wie der Merkle Root gebildet wird. Um den Merkle Root zu erhalten, werden zwei Transaktionen zu einem Paar gebildet und der Hash von diesem Paar generiert. Dieser neue Hash wird wiederum mit einem zweiten Hash kombiniert und aus diesen zwei wird wieder ein neuer Hash gebildet. Dies geschieht so lange, bis nur noch ein Hash existiert. Dieser Hash wird Merkle Root genannt. Hat eine Transaktion keinen Partner, kombiniert er sich mit sich selbst. [13]

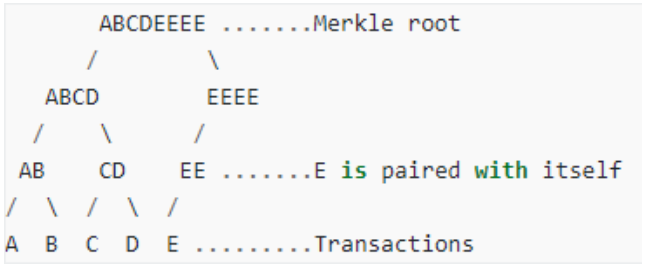

Abbildung 3: Merkle Tree

Der Merkle Tree wird für das «Simplified Payment Verification», kurz SPV, verfahren genutzt. SPV ermöglicht es zu überprüfen, ob eine Transaktion in der Blockchain vorhanden ist oder nicht, ohne die gesamte Blockchain herunterzuladen. Mit diesem Verfahren wird nur der Merkle Tree und die Transaktions ID gebraucht. Muss zum Beispiel die Transaktion D überprüft werden, werden nur die Hashes C, AB und EEEE gebraucht. Dadurch kann viel Speicherplatz gespart werden. [13]

Die Implementierung in den einzelnen Blockchains kann sich jedoch unterscheiden. Beispielsweise wird bei Ethereum ein «Merkle Patricia Tree» eingesetzt. [14]

2.2 Konsensmechanismus

Ein Konsensmechanismus definiert bei einer Blockchain, wie das verteilte System eine Übereinkunft trifft und welcher Block der nächste auf der Blockchain sein soll. Ebenfalls schützt der Mechanismus die Blockchain selbst vor Angriffen. Dies wird erschwert, indem eine bestimmte Ressource benötigt wird, um einen

Angriff überhaupt in Gang setzen zu können. Falls bei einem Konsensalgorithmus eine Partei 51 % der Ressource besitzt, ist die Integrität der Blockchain nicht mehr gewährleistet, da die Partei die Mehrheit der Stimmen kontrolliert. Dabei gibt es verschiedene Ansätze. Die zwei bekanntesten sind Proof of Work und Proof of Stake. [15, 16]

2.3 Proof of Work

Jeder Block, der in die Blockchain aufgenommen werden will, muss beweisen, dass eine gewisse Arbeit aufgebracht wurde. Dies ist nötig, damit nicht vertrauenswürdige Nodes keine alten Blöcke modifizieren können. Zudem erhöht sich der Aufwand mit jedem neu hinzugefügtem Block, da diese auch angepasst werden müssen. Bei Proof of Work, muss durch Rechenleistung ein Rätsel gelöst werden, um einen neuen Block zu schürfen. [17]

Bei Bitcoin beispielsweise setzt der Miner einen Nonce (eine Zahl, die nur einmal genutzt wird) ein und hasht den gesamten Header. Damit ein Block in der Blockchain aufgenommen wird, muss dieser Hash eine bestimmte Anzahl führenden Nullen besitzen. Um den korrekten Nonce zu finden, muss der Miner zufällige Werte einsetzen, den Header hashen und überprüfen, ob der Hash die Anzahl geforderter Nullen besitzt. Die Anzahl Nullen kann sich alle zwei Wochen ändern. Ebenfalls soll alle 10 Minuten ein neuer Block erstellt werden. Alle zwei Wochen wird überprüft, ob im Durchschnitt alle 10 Minuten ein Block erstellt worden ist oder nicht. Wenn die Blöcke schneller erstellt wurden, erhöht sich die Schwierigkeit und wenn Blöcke langsamer erstellt wurden, wird sie verkleinert. Der Schwierigkeitsgrad wird dadurch definiert, wie viele führende Nullen der Hash besitzen muss. [13, 17]

2.4 Wallets / Adressen

Ein Wallet ist ein Programm, welches Private und Public Keys verwaltet, um Geld zu empfangen oder zu versenden. Eine Adresse wird aus dem Public Key gebildet. An diese Adresse wird jeweils das Geld gesendet. [18]

Der Private Key ist eine 256-Bit Nummer ($1 – 2^{256}$). Aus diesem Private Key wird mit dem Elliptic-Curve-Verfahren der Public Key generiert. [18]

Ein wichtiges Thema bei der Blockchain ist die Anonymität. Weil alle Informationen öffentlich sind und für jeden einsehbar, kann jeder herausfinden, wie viel Geld jemand besitzt, wenn er weiss welche Adresse jemandem gehört. Um die Anonymität zu erhöhen, wird empfohlen, für jede Transaktion eine neue Adresse zu nutzen. Jedoch ist das Verwalten und Sicherstellen aller Private und Public Keys sehr aufwändig, wenn sich diese mit der Anzahl an Transaktionen erhöht. Aus diesem Grund gibt es ein Verfahren, um «Hierarchical Deterministik» Keys zu erstellen. Die Wallets, die solche Keys erstellen werden auch HD Wallets genannt. [19]

Ein HD Wallet kann aus einem Parent Private und Public Key mehrere Child Private und Public Keys erstellen. Zusätzlich wird ein zufälliger Master Seed genutzt, damit es eine zufällige Komponente bei der Erstellung der Children Keys gibt. Ein HD Wallet vereinfacht das Verwalten mehrerer Adressen. Es wird nur der Master Seed, der Parent Public und Private Key benötigt, um alle Children Keys wieder herzustellen. Durch diese Methode kann ganz einfach bei jeder neuen Transaktion eine neue Adresse genutzt werden. [20]

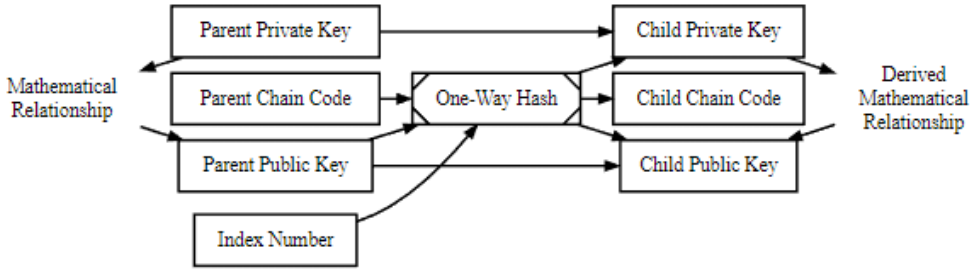

Abbildung 4: HD Wallet

Die Abbildung 4 zeigt, wie Child Private und Public Keys von einem Parent Private und Public Key erstellt werden. Die Index Number ist dabei der zufällige Seed. [20]

2.5 Transaktion

In einer Transaktion steht drin, wer von wem wie viel Geld erhält. Nachdem eine Transaktion erstellt wurde, muss diese mit dem Private Key signiert werden. Danach wird sie im Blockchain Netzwerk veröffentlicht und muss nur noch warten, bis die Transaktion validiert und in einem Block aufgenommen wird. Sobald der Block geschürft wurde, ist die Transaktion gültig. [21]

Eine Transaktion, welche nur veröffentlicht wurde, ist noch keine Garantie das der Empfänger bezahlt wurde. Es kann sein, dass diese Transaktion in keinem Block landet oder dass zwei Blocks gleichzeitig erstellt werden. Zusätzlich gibt es auch «Double-Spending»-Attacken, bei der der Angreifer sein Guthaben zwei Mal ausgibt. Deshalb sollte eine gewisse Anzahl Bestätigungen gewartet werden, um sich gegen Double-Spending-Attacken zu schützen und sicher zu sein, dass die Transaktion gültig ist. Je mehr Bestätigungen abgewartet werden, desto aufwändiger ist eine Double-Spending-Attacke. Je nach Blockchain ist die Anzahl empfohlener Bestätigungen unterschiedlich. Bei Bitcoin sind es beispielweise 6 Bestätigungen. [22]

Für jede Transaktion müssen Transaktionskosten bezahlt werden. Diese Kosten sind dynamisch und werden anhand des Angebots und der Nachfrage berechnet.

Rückerstattungen können bei Kryptowährungen nicht einfach automatisiert werden. Dies hat drei Gründe. Der erste Grund ist, dass viele Endbenutzer ein Wallet auf einer Plattform wie Coinbase besitzen und nicht die volle Kontrolle über ihr Wallet haben. Somit kann es sein, dass die Absenderadresse nicht dem Benutzer gehört, sondern der Plattform. Wird dabei eine Rückerstattung auf die Sender Adresse getätigt, erhält nicht der Benutzer das Guthaben, sondern die Plattform. [22]

Der zweite Grund ist, dass Adressen teilweise pro Transaktion nur einmal genutzt werden. Dies wird vor allem für eine erhöhte Anonymität getan. Wird hier das Guthaben an die Sender Adresse gesendet, wird das Guthaben für weitere Transaktionen nicht weiter benutzt. [23]

Der dritte Grund ist, dass Kryptowährungs-Mixer genutzt werden können. Ein Mixer ist ein Dienst, welcher Geldmengen mit anderen vermischt, um die Anonymität zu steigern. Sollte ein Benutzer einen Mixer nutzen, würde bei einer automatischen Rückerstattung die falsche Person das Geld erhalten. [24, 25]

Um Rückerstattungen zu gewährleisten, muss der Benutzer CHainGate in irgendeiner Weise seine Rückerstattungsadresse mitteilen. Je nach Blockchain existieren hier verschiedene Varianten. Die einfachste ist, dass sich der Benutzer beim Händler meldet und die Adresse per E-Mail mitteilt. Danach muss der Händler in Zusammenarbeit mit CHainGate die Rückerstattung ausführen.

2.6 UTXO Model vs. Account Model

Es gibt zwei unterschiedliche Varianten, wie das Geld auf der Blockchain verwaltet werden kann. Die erste ist über «Unspent Transaction Output (UTXO)» und die zweite ist «Account»-basiert.

2.6.1 UTXO

Bei UTXO existiert kein globaler Status mit dem gesamten Guthaben einer Adresse. Stattdessen werden alle Transaktionen geprüft und gezählt, wie viele Einnahmen nicht ausgegeben wurden. Die nicht ausgegebenen Einnahmen werden Unspent Transaction Output (UTXO) genannt. [26]

Eine Transaktion besteht immer aus mindestens einem Input und einem Output. Dabei ist ein Input ein Output, welcher noch nicht ausgegeben wurde. Ein nicht ausgegebener Output wird als UTXOs (Unspent Transaction Output) bezeichnet. Ein Input (UTXO) wird immer komplett ausgegeben. Das Wechselgeld wird über einen zweiten Output an eine neue Adresse gesendet, die einem selbst gehört. Die Differenz zwischen Input und Outputs sind die Transaktionskosten. [26]

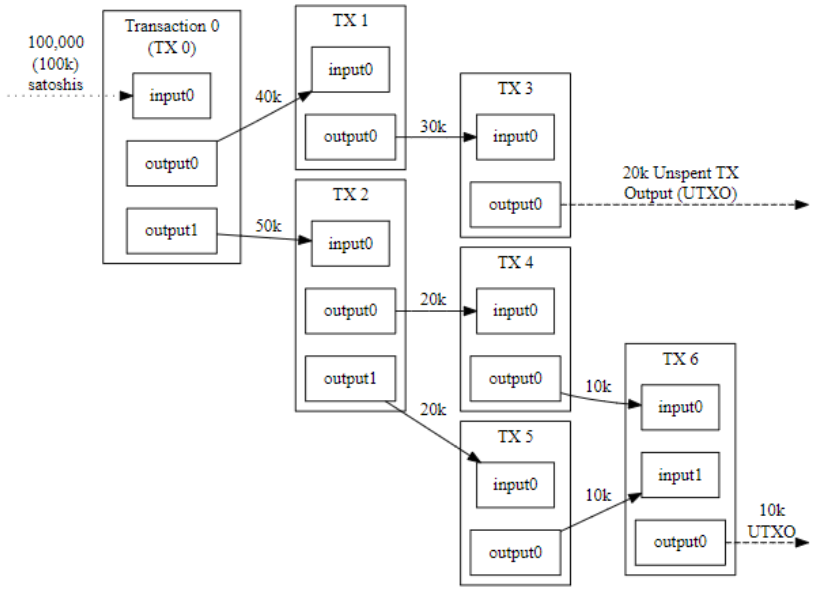

Abbildung 5: Bitcoin Transaktionen

In der Abbildung 5 sind mehrere Transaktionen visualisiert. Dabei hat jede Transaktion mindestens einen Input und einen Output. [13]

2.6.2 Account Model

Das Account-basierte Model funktioniert ähnlich wie ein Bankkonto. Wird eine neue Adresse erstellt, wird diese Adresse zum globalen Status der Blockchain hinzugefügt. Auf diesem Status ist jeweils das Guthaben der Adresse gespeichert. Eine Transaktion bei dieser Variante vergrössert oder verkleinert das Guthaben auf dem globalen Status. [26]

2.7 Integrierte Blockchains

Die meisten Blockchains besitzen mehrere Umgebungen, unter anderem ein Mainnet, Testnet und Privatnet.

Mainnet: Ist die produktive und öffentliche Umgebung.

Testnet: Ist die Testumgebung. Über ein Faucet können Entwickler und Tester Geld kostenlos erhalten.

Privatnet: Ist eine private oder lokale Umgebung, die von einem selbst verwaltet wird. Zum Beispiel kann bei Bitcoin, mittels Regtest, eine lokale Blockchain für das Testen aufgesetzt werden.

Die Tabelle 1 zeigt die expliziten Netze mit Namen basierend auf der Blockchain.

Netz	Ethereum	Bitcoin
Mainnet	Main	Main
Testnet	Rinkeby, Ropsten, Görli	Testnet3
Privatnet	Lokal	Regtest

Tabelle 1: Blockchain Umgebungen

In den nachfolgenden Unterkapiteln wird detaillierter auf die Ethereum und Bitcoin Blockchain eingegangen.

2.7.1 Ethereum

Basierend auf der Marktkapitalisierung ist Ethereum nach Bitcoin die zweitgrösste Blockchain auf der Welt. Die Währung wird Ether genannt, kann aber in eine kleinere Währung namens Wei geteilt werden. Ethereum ist gerade dabei, von einem Proof of Work Konsensalgorithmus in ein Proof of Stake überzugehen.

2.7.1.1 Einheiten

Die folgende Tabelle 2 zeigt die unterschiedlichen Geldeinheiten bei Ethereum. [27]

Ether	Einheit
1.0	Ether (ETH)
1 × 10-3	Milliether (finney)
1 × 10-6	Microether (szabo)
1 × 10-9	Gwei (shannon)
1 × 10-12	Mwei (lovelace)
1 × 10-15	Kwei (babbage)
1 × 10-18	Wei

Tabelle 2: Ethereum Einheiten

2.7.1.2 Wallets / Adressen

Um eine Adresse in Ethereum zu generieren, wird lediglich ein Private Key mit einer Nummer zwischen 1 und 2^{256} benötigt. Basierend auf dem Private Key kann die Adresse und der Public Key generiert werden. Ethereum benutzt ein Elliptic-Curve-Verfahren, um den Public Key zu generieren. Eine Adresse in Ethereum ist ein Hash über den Public Key mit der Keccak-256-Hash-Funktion. Dabei werden vom Hash nur die letzten 20 Bytes des Hashes als Adresse genommen. [28]

2.7.1.3 Transaktionskosten

Die Berechnung der Fees wurde bei Ethereum in dem London Upgrade mit dem Change EIP-1559 geändert. Damit wurde bezweckt, dass das Netzwerk sich bei etwa 50 % Netzwerklast einpendelt. Dadurch können Spitzen im Netzwerk besser geglättet und die Transaktionskosten besser eingeschätzt werden. [29]

Um die Änderung besser zu verdeutlichen, wird folgendes als Beispiel genommen: **Alice will Bob 1 ETH senden**. Damit Bob 1 ETH erhält, muss Alice die Transaktionskosten bezahlen. Die Transaktionskosten belaufen sich auf den Aufwand, welcher ein Miner tätigen muss. Bei einer Transaktion werden immer 21'000 Gas benötigt. Gas ist die Menge an Leistung, die benötigt wird, um eine Aktion auf der Blockchain auszuführen. Der Gaspreis variiert je nach Auslastung der Blockchain.

Vor dem London Upgrade

Vor dem Upgrade gab es einen Gaspreis pro Einheit. Dieser ist in diesem Beispiel 200 Gwei. Der Preis für die Transaktion wurde folgendermassen berechnet:

$$Gas\ units\ (limit) \times Gas\ price\ per\ unit$$

Mit dem Beispiel:

$$21{,}000 \times 200 \ = \ 4{,}200{,}000\ gwei\ or\ 0.0042\ ETH$$

Das würde bedeuten, dass Alice total 1.0042 ETH senden muss. 1 ETH erhält Bob und 0.0042 ETH der Miner.

Nach dem London Upgrade

Neu gibt es keinen Gaspreis pro Einheit mehr, sondern diese Zahl wird in eine Basisgebühr und ein Trinkgeld geteilt. Für dieses Beispiel ist die Basisgebühr 100 Gwei und das Trinkgeld ist 10 Gwei. Die Berechnung lautet folgendermassen:

$$Gas\ units\ (limit)\ *\ (Base\ fee\ +\ Tip)$$

Mit unserem Beispiel:

$$21{,}000\ *\ (100\ +\ 10)\ =\ 2{,}310{,}000\ gwei\ or\ 0.00231\ ETH$$

Alice muss nun 1.00231 ETH senden. Dabei erhält Bob wieder den 1 ETH. Im Gegensatz zu früher werden jetzt aber 0.0021 verbrannt und der Miner erhält nur 0.00021 ETH.[30, 31]

Die ETHs werden aus 2 Gründen neu verbrannt:

1. Um einer Inflation entgegenzuwirken
2. Damit Miner kein Nullsummenspiel machen können, indem sie Transaktionen auf das Netz veröffentlichen und diese selbst Minen.

Die Grundgebühr ist abhängig von dem vorherigen Block und deren Grundgebühr. Wenn der vorherige Block zu 100 % ausgelastet war, wird die Grundgebühr um 12.5 % erhöht. Wenn der vorherige Block leer war, wird die Grundgebühr um 12.5 % reduziert.

2.7.1.4 Dust

Für die theoretisch Berechnung der Fees wird die **Basisgebühr** und das momentane **Trinkgeld** benötigt. Zu beachten ist jedoch, dass sich der Preis, zwischen der Erstellung einer Transaktion und dessen Ausführung, verändern kann. Dadurch kann der finale Preis nie mit Sicherheit genau vorausgesagt werden. Somit muss die ganze Berechnung als Richtwert betrachtet werden.

Beim Setzen des Richtwertes wird immer ein Maximum angegeben. Falls das Maximum nicht benötigt wird, behält der Sender das restliche Geld. Falls das Maximum zu tief ist, wird die Transaktion nicht geschürft.

Im Falle von CHainGate bedeutet ein hohes Maximum des Gaspreises, dass die Transaktion schneller beim Händler ankommt. Der Nachteil hingegen ist, dass der Händler höhere Transaktionskosten bezahlt, weil die Differenz zwischen Maximum und tatsächlichen Kosten nicht an den Empfänger gesendet werden, sondern an den Sender. Das heisst, dass das Maximum nicht zu hoch gesetzt werden darf.

Wenn der Restbestand einer Adresse niedriger ist wie die Transaktionskosten, wird dieser Betrag als «Dust» bezeichnet, weil dieser nicht ausgegeben werden kann.

In der folgenden Abbildung 6 wird der Unterschied der einzelnen Gaslimiten gezeigt.

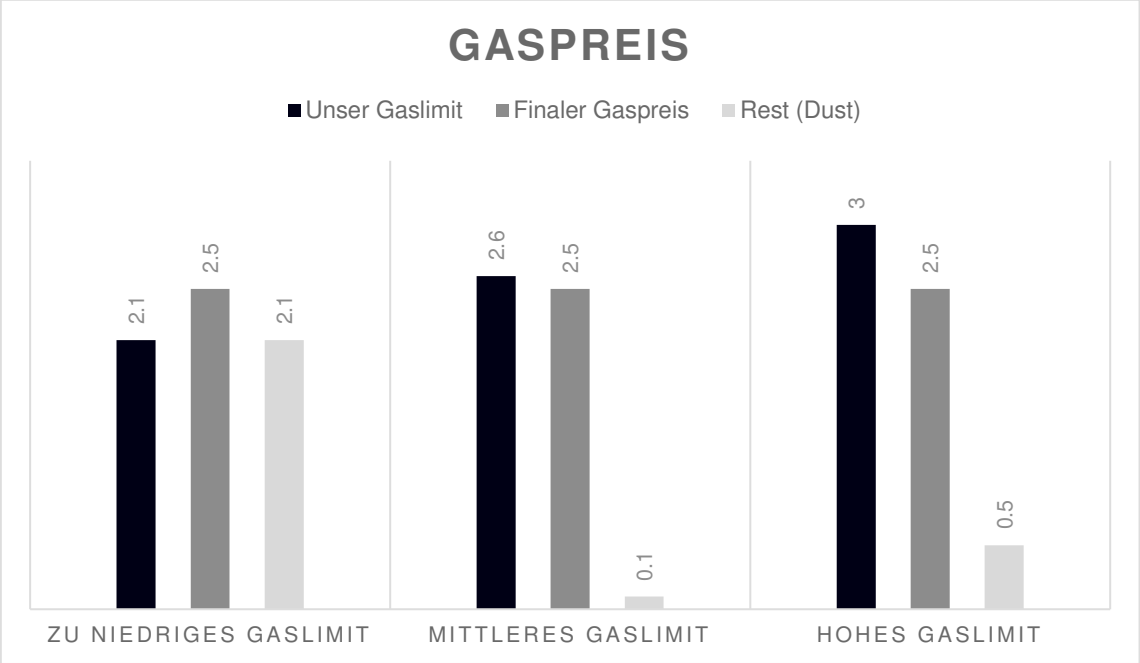

Abbildung 6: Gaspreis

Es gibt die Möglichkeit, eine Adresse theoretisch komplett zu leeren, bei dieser Variante würde jedoch nur der Miner profitieren, da dieser den Rest erhalten würde. [32]

2.7.1.5 Doppeltes Senden verhindern

Bei Ethereum gibt es einen Account-Nonce für jede Adresse. Dieser Nonce startet bei 0. Bei jeder Transaktion wird dieser um 1 erhöht. Wenn ein Nonce bereits in einen Block aufgenommen wurde, für eine Transaktion, wird die neue Transaktion abgelehnt. Ist die Transaktion im Pool und noch nicht geschürft, ersetzt sie die alte Transaktion. Ist der Nonce zu hoch, bleibt diese offen, bis alle fehlenden Nonces dazwischen geschürft worden sind. [33]

2.7.2 Bitcoin

Bitcoin ist die grösste Kryptowährung gemäss Marktkapitalisierung und die erste mit hohem Bekanntheitsgrad. [34, 35]

2.7.2.1 Einheiten

Bitcoin besitzt verschiedene Einheiten. Dabei ist Bitcoin die Grösste und Satoshi die Kleinste. Die folgende Tabelle 3 zeigt die unterschiedlichen Geldeinheiten von Bitcoin. [22]

Bitcoins	Einheit
1.0	Bitcoin (BTC)
1 × 10-2	Bitcent (cBTC)
1 × 10-3	Millibitcoin (mBTC)
1 × 10-6	Microbitcoin (uBTC, «bits»)
1 × 10-7	Finney
1 × 10-8	Satoshi

Tabelle 3: Bitcoin Einheiten

2.7.2.2 Wallets / Adressen

Bitcoin besitzt verschiedene Adresstypen. Hier werden diese kurz erklärt.

P2PK

P2PK bedeutet «Pay-to-Public-Key». Dabei ist die Empfangsadresse direkt der Public Key. Dieses Format wurde vor allem von der Bitcoin Software genutzt, um die Miner zu bezahlen. [36]

P2PKH

P2PKH bedeutet «Pay-to-Public-Key-Hash». Hier ist die Adresse nicht der Public Key, sondern der Hash davon. Dies hat den Vorteil, dass die Adresse kürzer und der Public Key nicht sofort ersichtlich ist. [36]

P2SH

P2SH bedeutet «Pay-to-Script-Hash». Bei dieser Variante wird nicht definiert, welche Adresse der Empfängt hat, sondern welches Script der Empfänger einreichen muss. Ein Script besitzt definierte Regeln vom Sender, welche erfüllt sein müssen, um das Geld auszugeben. Dabei wird ein Hash vom Script erstellt und als Empfänger angegeben. Der Empfänger muss dabei das Script einreichen, damit er Zugriff auf das Guthaben hat. Dieses Format wurde vor allem eingeführt, um Multisignatur-Transaktionen zu ermöglichen. Bei Multisignatur-Transaktionen muss die Transaktion von mehreren Parteien signiert werden, damit sie gültig ist. [36]

P2WPKH / Bech32 (SegWit Adresse)

P2WPKH oder auch Bech32 ist «Pay-to-Witness-Public-Key-Hash». Dieses Format ist durch das Segregated Witness (SegWit) Update eingeführt worden. Dieser Adresstyp dient vor allem der Reduktion der Blockgrösse. Die Vorteile sind niedrige Transaktionsgebühren und die hohe Verarbeitungsgeschwindigkeit. [37]

Die Reduktion der Grösse wird erreicht, indem die Signatur aus den Transaktionsdaten entfernt wird. [38]

2.7.2.3 Transaktionskosten

Die Transaktionskosten werden anhand der Grösse der Transaktion und der Kosten pro Byte berechnet.

Die Formel für die Transaktionsgrösse für SegWit Transaktionen ist folgende.

$$Transaktionsgrösse = (Anzahl\ Inputs \times 68.5) + (Anzahl\ Outputs \times 31) + 10$$

Die Transaktionsgrösse ist in vBytes (virtuelle Bytes). Zusätzlich müssen die Inputs und Outputs P2WPKH Adressen sein. [39]

Die Kosten pro Byte können je nach Anfrage und Auslastung variieren. Um eine gute Schätzung zu erhalten, bietet der Bitcoin Node eine Funktion an. [40]

Um die gesamten Transaktionskosten zu berechnen, kann folgende Formel gebraucht werden.

$$Gesamte\ Transaktionskosten = \frac{fee\ rate \times 100'000'000}{1000} \times 141 vBytes$$

Die «fee rate» sind die Kosten in BTC/kvB. Als Beispiel kann ein Wert von 0.00001016 BTC/kvB genutzt werden.

Somit sind die gesamten Transaktionskosten 144 Satoshis.

$$\frac{0.00001016 \times 100'000'000}{1000} \times 141 vBytes = 143.256 = 144\ Satoshis$$

2.7.2.4 Dust

Dust spielt bei Bitcoin eine grössere Rolle, da Bitcoin UTXO-basierend ist und die Transaktionskosten abhängig von der Anzahl inkludierten Adressen sind.

Dust Transaktionen

Eine Dust Transaktion ist eine sehr kleine Transaktion. Der Output ist dabei so klein, dass dieser nicht als Input verwendet werden kann, weil die Transaktionsgebühren höher sind als der Betrag. Aus diesem Grund werden solche Transaktionen nicht erlaubt, da sie keinen Sinn ergeben. [41]

Dust Formel

$$Dust = \frac{Transaktionsgrösse \times dustRelayFee}{1000}$$

Beispiel:

Transaktion Output: 34 Bytes (non-SegWit)
Transaktion Input: Mindestens 148 Byte
dustRelayFee Standardwert: 3000 Satoshi/kvB

$$\frac{(34 + 148) \times 3000}{1000} = 546\ Satoshis$$

Somit muss der Output einer Transaktion mindestens 546 Satoshis sein. [42]

Gemäss Source Code gibt es zusätzlich den «discardfee» Parameter, welche bei der Dust Definition eine Rolle spielt. [43]

Cost of Change

Zusätzlich zum Dust gibt es noch die «Cost of Change». Dabei muss das Rückgeld höher sein als der Cost of Change. [43]

Hier ist die Formel für die Cost of Change Berechnung.

$$Change\ Fee = \frac{Fee\ Rate \times 100'000'000}{1000} \times Change\ Output\ Size$$

$$Cost\ of\ Change = \frac{Discard\ Fee\ Rate \times Change\ Spend\ Size}{1000} + Change\ Fee$$

Fee Rate: Transaktionskosten in Satoshis pro Kilobyte.
Change Output Size: Die Grösse der Rückgeldadresse. Bei einer Bech32 Adresse sind es 32 Byte. [44]
Discard Fee Rate: Gemäss Konfiguration.
Change Spend Size: Die Grösse des Rückgeldes, wenn es als Input verwendet wird.

2.7.2.5 Doppeltes Senden verhindern

Bei Bitcoin gibt es keine Möglichkeit, wie das doppelte Senden verhindert werden kann. Es existiert nur die Möglichkeit, eine offene Transaktion durch eine ähnliche mit höheren Transaktionskosten zu ersetzen. Dies wird genutzt, wenn die Transaktionskosten schnell steigen und die Transaktion weiterhin schnell bestätigt werden soll. Soll das doppelte Senden verhindert werden, muss die Logik selbst implementiert werden. [45]

3. Planung und Vorbereitung

In diesem Kapitel wird die Umsetzung für einen eigenen Payment Provider analysiert, entschieden und beschrieben.

3.1 Anforderungen

CHainGate besitzt verschiedene funktionale und nicht funktionale Anforderungen. Dabei soll CHainGate eine Managementseite für die Händler besitzen. Auf dieser Seite können API-Keys erstellt, Kryptowährungen aktiviert oder deaktiviert und Statusinformationen angeschaut werden. Weiter können Händler den Service gegen eine kleine Gebühr pro Transaktion nutzen, um eine Zahlung in Ethereum oder Bitcoin zu erstellen. Dabei wird das erhaltene Geld von einem Käufer nach der Validierung und Verifikation an den Händler weitergeleitet. Zusätzlich erhält der Händler automatisch ein Update, wenn sich der Status einer Zahlung ändert. Ausserdem soll der Händler entscheiden können, ob er die Auswahl der Kryptowährung und das Handling der Updates selbst übernehmen will oder nicht. Für den Fall, dass der Händler diese nicht selbst übernehmen will, wird von CHainGate eine Payment Page (Zahlungsseite) zur Verfügung gestellt. Sie soll äquivalent zu einer Payment Page wie beispielsweise von Twint sein.

Zudem soll das System schnell mit neuen Kryptowährungen erweiterbar und die Sicherheit gewährleistet sein. Dabei darf Geld nicht verloren gehen.

Die detaillierten Anforderungen sind im Anhang unter dem Kapitel «B. Anforderungen» zu finden.

3.2 Architektur

Um eine passende Architektur zu finden, wurde eine Monolithische, Self-Contained System (SCS), Microservice und Microkernel Architektur miteinander verglichen. Es wurde geprüft, wie gut die Kriterien Erweiterbarkeit, Fehlertoleranz, Wartbarkeit, Skalierbarkeit und Komplexität erfüllt werden.

Monolithische Architektur

Ein monolithisches System zeichnet sich dadurch aus, dass es als eine Einheit bereitgestellt werden kann. Dabei wird das gesamte System gemäss der Mehrschichtenarchitektur in mehrere horizontale Schichten unterteilt. Jede Schicht hat ihre klar zugeordnete Aufgabe und die Zugriffe geschehen immer von einer höheren auf eine tiefere oder innerhalb einer Schicht. Die Anzahl und deren Bezeichnung ist dabei nicht definiert. [46]

Self-Contained System (SCS) Architektur

Self-Contained Systeme (SCS) haben das Ziel, das Gesamtsystem in verschiedene unabhängige Teilsysteme aufzuteilen. Die Teilsysteme werden gemäss ihrer Funktionalität aufgeteilt. SCS sind sehr ähnlich zu den Microservice Systemen, jedoch sind SCS grösser und bestehen aus einer Benutzeroberfläche, der Businesslogik und einer Datenbank. [47]

Microservice Architektur

Bei der Microservice Architektur wird das gesamte System in kleine unabhängige Services aufgeteilt. Jeder Service hat dabei eine Aufgabe und ist klar von den anderen abgegrenzt. Die Kommunikation untereinander geschieht über klar definierte synchrone oder asynchrone Schnittstellen. Für die Integration der Benutzeroberfläche und der Datenbank gibt es unterschiedliche Methoden. [48]

Microkernel / Plugin Architektur

Das Microkernel Pattern besteht aus zwei Komponenten. Es besitzt einen Kern und mehrere Plugin Module. Der Kern beinhaltet die minimalen Funktionalitäten, um das System lauffähig zu machen. Die Plugin Module können genutzt werden, um die Funktionalität des Systems flexibel zu erweitern. Dabei ist es wichtig, dass die Plugins unabhängig sind. Trotz der Unabhängigkeit ist es möglich, dass Plugins untereinander über den Kern kommunizieren können. [49]

Die folgende Tabelle 4 zeigt eine Übersicht über die erstellte Analyse der einzelnen Architekturen.

	Monolith	**SCS**	**Microservice**	**Microkernel**
Erweiterbarkeit	Schlecht	Mittel	Gut	Gut
Fehlertoleranz	Mittel	Gut	Gut	Gut
Wartbarkeit	Mittel	Gut	Mittel	Gut
Skalierbarkeit	Mittel	Gut	Gut	Mittel
Komplexität	Gut	Mittel	Schlecht	Schlecht
Rang	**4**	**3**	**1**	**2**

Tabelle 4: Analyse Architektur

Aufgrund der Analyse wurde eine Microservice Architektur gewählt. Der wichtigste Grund ist, dass neue Kryptowährungen problemlos angebunden werden können. Dabei kann jede neue Kryptowährung ihren eigenen Technologiestack besitzen. Dies ist nötig, weil nicht jede Blockchain dieselben Programmiersprachen unterstützt.

Die Microkernel Architektur auf dem zweiten Rang wäre auch eine sehr gute Lösung gewesen. Soll eine Microkernel Architektur skalierbar sein, sollten die Plugins als Microservices implementiert werden. Zusätzlich muss geklärt werden, wie das Vertragsmanagement aussieht, die Plugins sich registrieren können und wie sie miteinander funktionieren. Dadurch steigt die Komplexität enorm. Wenn bei der Microkernel Architektur, Microservices für die Plugins benutzt werden, ist es mit kleinerem Aufwand möglich, von der Microservice Architektur zu einer Microkernel Architektur zu wechseln und umgekehrt.

Die detaillierte Analyse zur Architektur ist im Anhang unter dem Kapitel «C. Architektur» zu finden.

3.3 Datenbanken

Bei einer Microservice Architektur, stellt sich die Frage, wie die Datenbank aussieht und wie viele es gibt. Dabei gibt es die Möglichkeit, dass eine gemeinsame Datenbank oder eine Datenbank pro Microservice genutzt wird. Nachfolgend werden jeweils die Vor- sowie Nachteile der beiden Varianten aufgezählt. [50]

Eine Datenbank pro Microservice

Die Vorteile einer Datenbank pro Microservice sind:

+ Jeder Blockchain Service kann die Datenbank nach seinen Bedürfnissen konfigurieren.
+ Datenbank Anpassungen von einem Service betreffen die anderen Services nicht.
+ Die Services haben eine tiefe Kopplung.
+ Skalierung ist auf Service-Ebene möglich.

Die Nachteile einer Datenbank pro Microservice sind:

– Mehrere Datenbanken erhöhen die Komplexität.
– SQL-Abfragen über mehrere Services sind eine Herausforderung

Eine gemeinsame Datenbank

Die Vorteile einer gemeinsamen Datenbank sind:

+ Es muss nur eine Datenbank verwaltet werden.
+ Keine SQL-Abfragen über mehrere Services notwendig.

Die Nachteile einer gemeinsamen Datenbank sind:

– Werden unterschiedliche Datenbanken oder Schemas benötigt, wird es schwierig, diese in einer Datenbank abzubilden.
– Eine Skalierung ist nicht auf Service-Ebene möglich.
– Die Datenbankanpassungen für einen Service betreffen alle weiteren Services.

Entscheidung

Es gibt viele unterschiedliche Blockchains. Dabei kann es sein, dass jede Blockchain für eine Zahlung andere Informationen benötigt. Mit einer gemeinsamen Datenbank besteht das Risiko, dass in Zukunft das Schema alle möglichen Blockchains zufriedenstellen muss und deshalb sehr komplex und unübersichtlich wird. Zusätzlich muss bei einer Integration einer neuen Blockchain Schema angepasst werden, was im Falle eines Fehlers zu einem Ausfall von mehreren Services führen kann. Aus diesem Grund wurde entschieden eine Datenbank pro Microservice einzusetzen.

3.4 Kommunikation zwischen den Microservices

Die Kommunikation zwischen den Microservices kann synchron oder asynchron geschehen. [51]

- Synchrone Kommunikation – HTTP ist ein synchrones Protokoll. Ein Client sendet über HTTP eine Anfrage an einen Server und wartet so lange, bis er die Antwort erhält.

- Asynchrone Kommunikation – AMQP ist ein asynchrones Protokoll. Hier sendet der Client auch eine Anfrage an einen Server, wartet jedoch **nicht** auf eine Antwort. Es werden jeweils asynchrone Nachrichten hin und her geschickt.

Grundsätzlich sollte die Kommunikation asynchron geschehen, um die Performance und Resilienz zu verbessern. Weil in dieser Arbeit ein erster Entwurf entwickelt wird, ist die Performance kein wichtiges Kriterium. Zusätzlich wird die Komplexität erhöht. Aus diesem Grund wurde sich für eine synchrone Kommunikation entschieden. Sollte das System an seine Performancegrenzen stossen, sollte jedoch auf eine asynchrone Kommunikation gewechselt werden. [51]

Weitere Details sind im Anhang unter dem Kapitel «D. Kommunikation zwischen den Microservices» ersichtlich.

3.5 Werkzeuge

Um die Entwicklung zu vereinfachen, wurden verschiedene Werkzeuge eingesetzt. Die drei wichtigsten sind dabei Docker, OpenAPI Spezifikation und GORM.

Docker

Docker wird genutzt, um alle Microservices in einem Container zu starten. Zusätzlich wird docker-compose genutzt, um das gesamte CHainGate System hochzufahren. Dies bietet den Vorteil, dass jeder Entwickler den gleichen Stand hat und ein Deployment simpel umzusetzen werden kann. [52]

OpenAPI Spezifikation

Alle Services kommunizieren synchron und über REST. Mit einer OpenAPI Spezifikation lassen sich die REST-Schnittstellen definieren und der Client- sowie der Servercode kann generiert werden. Somit kann besser sichergestellt werden, dass die Services nicht auseinanderlaufen. Ein weiterer Vorteil ist, dass über OpenAPI automatisch eine Dokumentation erstellt werden kann. [53]

GORM

GORM ist eine Object-Relational Mapping (ORM) Library für Golang. Durch GORM kann Code gespart werden, weil nicht alle SQL-Zugriffe programmiert werden müssen. Zusätzlich erlaubt GORM einen Code-First Ansatz. [54]

Weitere Details sowie die Vor- und Nachteile der Werkzeuge sind im Anhang unter dem Kapitel «E. Werkzeuge» genauer beschrieben.

3.6 Zahlungsprozess

In diesem Abschnitt wird der einfache Zahlungsprozess erläutert und zusätzlich werden fünf wichtige Fragen abgeklärt.

Hier ein kleines Beispiel anhand der nachfolgenden Abbildung 7:

«Der Käufer möchte bei einem Händler ein Notebook mit ETH kaufen. Der Händler erstellt eine neue Zahlung bei CHainGate über die API. Als Antwort erhält der Händler die Zahlungsinformationen wie die Ad-

resse, an welche bezahlt werden muss, sowie den Betrag. Diese Informationen werden dem Kunden weitergeleitet. Sobald der Kunde bezahlt hat, erhält der Händler automatisch Updates von CHainGate über den Status seiner Zahlung. Nach der Validierung wird das Geld an den Händler weitergeschickt.»

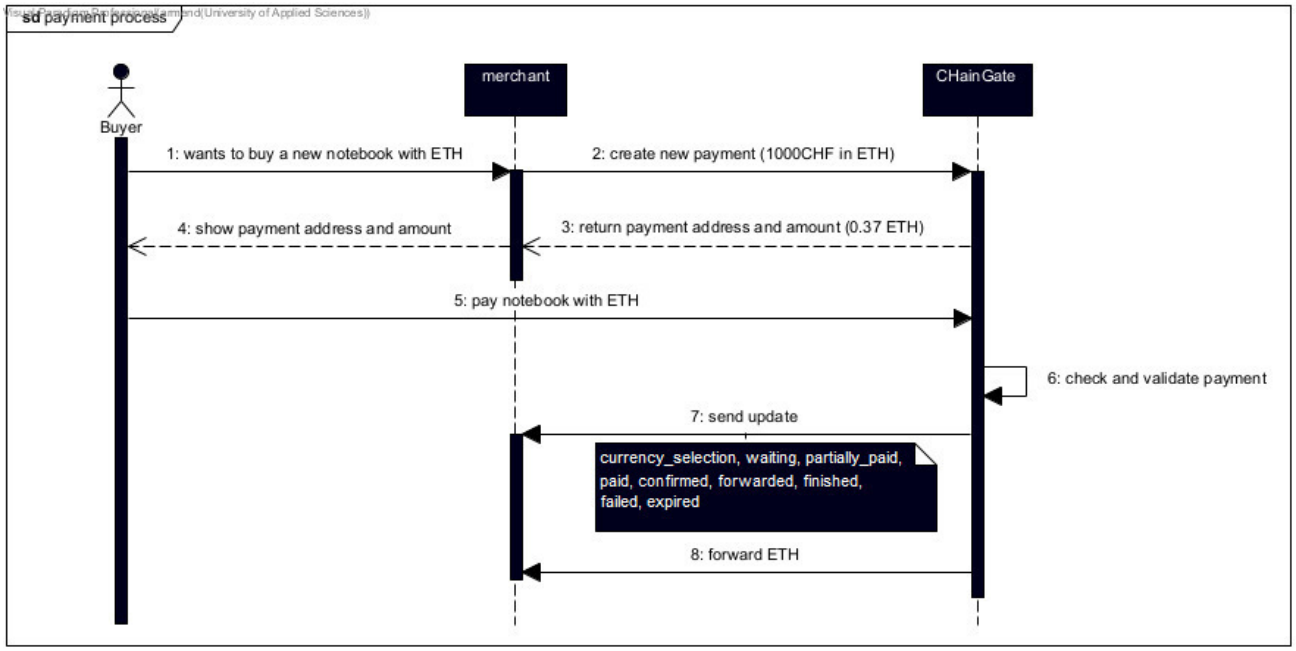

Abbildung 7: Zahlungsprozess

Bei diesem Prozess müssen folgende Fragen beantwortet werden:

- Wie wird eine Zahlung entgegengenommen?
- Was geschieht, wenn der Käufer zu viel bezahlt?
- Was geschieht, wenn der Käufer zu wenig bezahlt?
- Wer bezahlt die Transaktionskosten der Blockchain?
- Wie erhält CHainGate eine Provision?

3.6.1 Wie wird eine Zahlung entgegengenommen?

Bei der Entgegennahme einer Zahlung ist es wichtig, dass die einzelnen Zahlungen dem Käufer zugewiesen werden können. Ethereum und Bitcoin haben dabei unterschiedliche Möglichkeiten, wie dieses Ziel erreicht werden kann. Wichtig ist zudem, dass es für den Käufer so einfach wie möglich ist und die Transaktionskosten so niedrig wie nötig gehalten werden können.

In den nachfolgenden Unterkapiteln werden die unterschiedlichen Möglichkeiten von Ethereum und Blockchain genauer untersucht.

3.6.1.1 Ethereum

Bei Ethereum existieren folgende drei Varianten:

- 1 Adresse pro Transaktion
- 1 Adresse für alle Transaktionen
- Smart Contract

1 Adresse pro Transaktion

Die erste Variante generiert pro Transaktion eine neue Adresse. Dadurch entsteht ein Overhead. Dem kann mit der Wiederverwendung von Adressen entgegengewirkt werden. Ebenfalls kann eine Adresse genau einer Transaktion zugewiesen und überwacht werden.

Bei dieser Variante würden folgende Vor- und Nachteile entstehen:

+ Klare Identifikation eines Käufers ist möglich.
+ Nach der Generierung der Adresse sind es nur noch simple Blockchain Operationen.
+ Falls die Generierung der Adressen nicht automatisiert werden kann, kann anfänglich eine fixe Anzahl vorerstellter Adressen verwendet werden.
− Generierung einer Adresse.
− Realisierung des Gewinns ist erschwert.
− Wiederverwendung von Adressen erhöht die Komplexität.

1 Adresse für alle Transaktionen

Eine weitere Variante wäre es, eine einzige Adresse zu benutzen und jeder Käufer zahlt auf diese Adresse ein. Dabei entsteht die Problematik herauszufinden, welcher Käufer wie viel bezahlt hat. Um dem entgegenzuwirken, kann die Senderadresse der Transaktion analysiert werden. Dies würde jedoch voraussetzen, dass der Käufer diese CHainGate mitteilt. Bei vielen Kryptobörsen wie Coinbase oder Binance ist die Adresse, von welchem die Kryptowährung geschickt wird, vielmals nicht bekannt. Dadurch ist es für den Käufer selbst schwer, diese zu wissen. Ebenfalls würde es den Bezahlprozess für den Käufer erschweren, was unter allen Umständen vermieden werden sollte.

Bei dieser Variante würden folgende Vor- und Nachteile entstehen:

+ Einfache Realisierung des Gewinns möglich.
+ Keine Generierung der Adresse notwendig.
− Senderadresse muss bekannt sein.
− Der Kaufprozess wird für den Kunden erschwert.

Smart Contract

Die letzte untersuchte Variante ist die Möglichkeit, mit Smart Contracts die Bezahlungen entgegenzunehmen. In der Vorarbeit mit FlatFeeStack wurde ein ähnlicher Ansatz verfolgt. Wenn der Smart Contract ähnlich wie eine Adresse benutzt wird, entstehen die gleichen Probleme wie bei der Variante mit einer

Adresse. Stattdessen sollte eine Funktion vom Smart Contract aufgerufen werden, welcher die momentane «Bestell ID», die zu bezahlenden Kryptowährung und die finale Adresse entgegennimmt. Um diesen Aufruf benutzerfreundlich zu realisieren, bräuchte es eine Client-Integration, welche den Aufruf mit den nötigen Parametern entgegennimmt. Bei Ethereum wäre das zum Beispiel «Metamask». Die nötige Logik für das Weiterleiten würde dann im Smart Contract behandelt werden. Nach ersten Recherchen besteht zusätzlich die Möglichkeit, den Händler direkt aus dem Smart Contract über einen REST-Aufruf zu benachrichtigen. [55]

Bei dieser Variante würden folgende Vor- und Nachteile entstehen:

+ Haltung der ETHs wäre zentralisiert und öffentlich in einem Smart Contract.
+ CHainGates-Verdienst wäre öffentlich, was die Transparenz erhöht und das Vertrauen stärkt.
– Pro Währung bräuchte es eine eigene Smart Contract-Implementation und Client-Integration.
– Die Adresse des Händlers wäre öffentlich und dadurch der Verdienst durch Kryptowährungen.
– Käufer bräuchten ein Plugin pro Kryptowährung. Im Beispiel für Ethereum wäre es MetaMask.
– ChainGates-Verdienst wäre öffentlich, somit sind wichtige Geschäftszahlen für jeden einsehbar.
– Initiale Kosten für das Aufsetzen des Smart Contracts sind notwendig.
– Implementierungsfehler können durch den offenen Sourcecode einfacher identifiziert und ausgenutzt werden.

Entscheidung

Da die Variante mit nur einer Adresse zu technisch unlösbaren Problemen führen würde, wurde diese Variante früh eliminiert. Die Nachteile und die Komplexität der Lösung mit Smart Contracts überwiegen die Vorteile. Auch die Lösung selbst benötigt mehr initialen Aufwand als eine Lösung mit mehreren Adressen. Die erste Lösung mit mehreren Adressen hat ebenfalls einige Nachteile, welche jedoch durch mehr Implementierungsaufwand negiert werden können.

3.6.1.2 Bitcoin

Bei Bitcoin existieren folgende zwei Varianten, wie eine Zahlung entgegengenommen werden kann:

- 1 Adresse für alle Transaktionen
- 1 Adresse pro Transaktion

Im Gegensatz zu Ethereum bietet Bitcoin keine Smart Contracts an.

1 Adresse für alle Transaktionen

Die erste Variante ist es, eine Adresse für alle Transaktionen zu nutzen. Das Problem hierbei ist jedoch die Zuweisung von einer Transaktion zu einem Käufer. Ohne die Adresse vom Käufer kann CHainGate nicht wissen, welche Transaktion zu welcher Zahlung gehört. Bitcoin besitzt hier die gleichen Probleme wie Ethereum.

Bei dieser Variante würden folgende Vor- und Nachteile entstehen:

+ Weil bei Bitcoin bei einer Transaktion alle Inputs ausgegeben werden müssen, besteht hier kein Vorteil für die Realisierung des Gewinns.
− Senderadresse muss bekannt sein.
− Der Kaufprozess wird für den Kunden erschwert.

1 Adresse pro Transaktion

Bei dieser Variante wird für jede Transaktion eine neue Adresse erstellt. Mit einem HD-Wallet können neue Adressen erstellt werden. Bitcoin empfiehlt, dass eine Adresse maximal zwei Mal (einmal für das Empfangen und einmal für das Senden) genutzt wird. Das erhöht die Privatsphäre enorm, weil nicht mehr so einfach herausgefunden werden kann, wie viel Geld eine Person besitzt. [56]

Bei dieser Variante würden folgende Vor- und Nachteile entstehen:

+ Einfache Zuweisung einer Transaktion an einen Käufer.
+ Für die Realisierung des Gewinns kann eine Rückgeldadresse angegeben werden.
− Dust Guthaben kann nur ausgegeben werden, wenn die Adresse erneut Guthaben erhält.

Entscheidung

Eine Adresse pro Transaktion besitzt nur den Nachteil, dass Dust Transaktionen nicht ausgegeben werden können. Durch eine Wiederverwendung der Adressen erhalten diese zu einem späteren Zeitpunkt zusätzliches Guthaben und können wieder genutzt werden. Aus diesem Grund wurde sich für die Variante 1 Adresse pro Transaktion entschieden.

3.6.2 Was geschieht, wenn der Käufer zu viel bezahlt?

Es gibt 3 Varianten, was getan werden kann, wenn ein Käufer zu viel bezahlt hat.

Diese währen:

1. Den Restbetrag bei CHainGate verwahren
2. Den Restbetrag an den Händler weiterüberweisen
3. Den Restbetrag an den Käufer zurücküberweisen

Den Restbetrag bei CHainGate verwahren

Bei dieser Variante wird der zu bezahlende Betrag gemäss der erstellten Zahlung an den Händler weitergeleitet. Der Restbetrag, welcher zu viel bezahlt wurde, wird bei CHainGate verwahrt.

Dies bietet folgende Vorteile:

+ Der Händler muss keine eigene Logik implementieren für das Verwalten des Restbetrags.
+ Benutzerfreundlicher, weil sich CHainGate um die Rückzahlungen kümmert.
+ Es könnte eine automatische Rückerstattung implementiert werden.

Dies bietet folgende Nachteile:

- Implementationsaufwand für CHainGate für das Verwalten des Restbetrags.
- CHainGate muss die Rückzahlungen ausführen.

Den Restbetrag an den Händler weiterüberweisen

Der gesamte Betrag inklusive Restbetrag wird an den Händler überwiesen.

Es bietet CHainGate folgenden Vorteil:

+ Abwälzen des Problems an den Händler.

Jedoch besitzt der Händler folgenden Nachteil:

- Der Händler muss sich um mögliche Rückzahlungen kümmern. Das heisst, er muss sich mit Kryptowährungen auseinandersetzen.

Den Restbetrag an den Käufer zurücküberweisen

Zurücküberweisen klingt anfänglich sinnvoll, jedoch gibt es in der Kryptowährung-Welt Verschleierungstaktiken und auch gemeinsame Wallets für mehrere Kunden.

- Wenn ein Mixer genutzt wurde, bekommt der Falsche das Geld zurück. [24, 25]
- Wenn ein gemeinsames Wallet benutzt wurde, erkennt die dahinterliegende Software gegebenenfalls nicht, für wen das Geld gedacht ist.
- Gewisse Blockchains haben eine Option, dass eine Adresse nur einmal pro Transaktion genutzt wird. Der Grund ist, dass dadurch die Anonymität erhöht wird. [57]

Dadurch wurde auf diese Variante verzichtet.

Entscheidung

CHainGate will dem Kunden so viel wie möglich abnehmen. Zusätzlich kann es Probleme geben, wenn der Käufer zu wenig bezahlt hat (siehe «Kapitel 3.6.3 Was geschieht, wenn der Käufer zu wenig bezahlt?» . Aus diesen Gründen wurde sich für die Variante «Den Restbetrag bei CHainGate verwahren» entschieden.

3.6.3 Was geschieht, wenn der Käufer zu wenig bezahlt?

Eine Zahlung läuft nach 15 Minuten ab. Wenn ein Käufer innerhalb dieser Zeit nur einen Teilbetrag überweist, muss sich überlegt werden, was mit diesem Teilbetrag geschehen soll.

Dazu gibt es folgende 3 Varianten.

1. Den Betrag an den Käufer zurücksenden.
2. Den Betrag an den Händler senden.
3. Den Betrag bei CHainGate aufbewahren.

Den Betrag an den Käufer zurücksenden

Das Geld direkt an den Käufer zurückzusenden, wäre die beste Variante. Jedoch besteht das Problem von Verschleierungstaktiken und gemeinsamen Wallets wie im Kapitel «3.6.2 Was geschieht, wenn der Käufer zu viel bezahlt?» erwähnt wurde. Aus diesem Grund wurde diese Option nicht gewählt.

Den Betrag an den Händler senden

Eine weitere Möglichkeit ist es, den Betrag an den Händler zu senden. Dies macht jedoch wenig Sinn, weil der Händler keine vollständige Zahlung erhalten hat und somit der Kauf nicht abgeschlossen ist. Somit müsste sich der Händler um die Rücküberweisung an den Käufer kümmern. Das ist umständlich für den Benutzer und daher eine schlechte Option.

Den Betrag bei CHainGate aufbewahren

Die beste Variante ist es, dass CHainGate dieses Geld aufbewahrt und auf Anfrage des Käufers die Rücküberweisung tätigt. Vor der Überweisung muss jedoch eine Verifikation stattfinden, um zu überprüfen, dass dieses Geld wirklich dem Käufer gehört.

3.6.4 Wer bezahlt die Transaktionskosten der Blockchain?

Der Zahlungsprozess beinhaltet immer zwei Transaktionen. Die erste Transaktion ist vom Käufer zu CHainGate und die zweite von CHainGate zum Händler. Somit entstehen zwei Mal die Transaktionskosten, zudem ist eine Gebühr von CHainGate fällig. Diese Kosten können vom Käufer, Händler oder von Beiden bezahlt werden. Wenn der Käufer alle Kosten trägt, ist die Kryptowährung als Bezahlungsmittel und der Service von CHainGate unattraktiv. Jedoch ist es auch für den Händler unattraktiv, wenn er alle Kosten tragen muss. Deshalb werden die Kosten zwischen Käufer und Händler aufgeteilt. Die Gebühren für die Transaktion vom Käufer zu CHainGate wird vom Käufer bezahlt und die Gebühr von CHainGate zum Händler vom Händler. Zusätzlich bezahlt der Händler die CHainGate Gebühr.

Eine zusätzliche Möglichkeit wäre es, wenn jeder Händler selbst entscheiden könnte, welche Kosten er und welche der Käufer bezahlen muss. Aus Zeitgründen wurde dieser Ansatz nicht weiterverfolgt, kann jedoch in Zukunft implementiert werden.

3.6.5 Wie erhält CHainGate eine Provision?

CHainGate erhält eine Provision über eine Gebühr pro Transaktion. Diese Gebühr darf maximal 2.9 % des Kaufbetrags betragen, um konkurrenzfähig zu bleiben. Der durchschnittliche Onlineeinkauf ist etwa bei 125 CHF, somit beträgt der maximale CHainGate-Gewinn durchschnittlich 4 CHF pro Transaktion. [58]

Es ist wichtig, dass von dem Gewinn möglichst wenig Transaktionskosten abgezogen werden, um ihn zu realisieren. Die Ethereum und Bitcoin Implementationen verfolgen hier unterschiedliche Strategien.

Ethereum

Um den Gewinn zu erhalten, muss das Geld von den verschiedenen Adressen auf eine gemeinsame Adresse übertragen werden. Eine Transaktion kostet bei Ethereum zurzeit etwa 5 CHF und mit einem durchschnittlichen Gewinn von 4 CHF würde die Transaktion für die Realisierung des Gewinns zwar funktionieren, jedoch wird diese Transaktion beim jetzigen Kurs nicht geschürft. Deshalb wurde eine Wiederver-

wendung der Adressen geplant, welches die bestehenden Adressen erneut benutzt. Somit können Transaktionskosten gespart werden, weil weniger Transaktionen benötigt werden. Da bei Ethereum die Transaktionskosten schwanken, wird angestrebt, ein Vielfaches der Transaktionskosten als Grenze für die Realisierung zu definieren. Beispielweise wäre es denkbar, dass wenn der Wert der Adresse das 100-fache der Transaktionskosten übersteigt, dieses Guthaben automatisch an eine definierte CHainGate Adresse gesendet wird. Statistisch gesehen können damit niedrige Transaktionszeiten ausgenutzt werden.

Um einen solchen Mechanismus implementieren zu können, muss der aktuelle Betrag der Adresse zu jeder Zeit bekannt sein und es darf nie zu einer Mehrfachüberweisung oder einer zu hohen Überweisung kommen. Andernfalls würde der Gewinn von CHainGate verloren gehen.

Bitcoin

Weil Bitcoin UTXO basierend ist und bei einer Transaktion immer der gesamte Input ausgegeben werden muss, ist die Realisierung des Gewinns simpel. Dafür wird eine Adresse genutzt, welche den gesamten Gewinn beinhaltet. Immer wenn das Geld an den Händler weitergeschickt wird, wird diese CHainGate Adresse als Rückgeldadresse angegeben. Damit wird ohne zusätzliche Kosten das Geld an eine Gewinnadresse gesendet. Der einzige Nachteil dieser Variante ist, dass jeder den Bitcoin Gewinn von CHainGate einsehen kann.

3.7 Erstellung einer Zahlung

Gemäss den Anforderungen soll der Händler die Möglichkeit haben, eine Zahlung mit oder ohne Payment Page zu erstellen. Wird eine Zahlung mit einer Payment Page erstellt, muss der Händler den Kunden auf die Payment Page von CHainGate weiterleiten. Dort wählt der Käufer seine bevorzugte Währung aus und erhält anschliessend die Bezahlinformationen. Nachdem der Käufer bezahlt hat, wird er automatisch über den Status seiner Zahlung informiert. Zusätzlich erhält der Händler diese Informationen ebenfalls, damit er weiss, ob der Kauf abgeschlossen ist oder nicht.

Wird hingegen eine Zahlung ohne Payment Page erstellt, muss der Händler den Käufer selbst fragen, mit welcher Währung er bezahlen will und anschliessend die Zahlung bei CHainGate erstellen. Nach der Erstellung erhält der Händler die Zahlungsinformationen, welche vom Händler selbst an den Kunden weitergeleitet werden müssen. Zusätzlich muss der Händler den Käufer selbst über den Status seiner Zahlung informieren.

Eine Zahlung **mit** einer Payment Page sollte genutzt werden, wenn sich der Händler den Aufwand für die Kommunikation zum Käufer sparen will.

Eine Zahlung **ohne** Payment Page sollte genutzt werden, wenn es für den Händler wichtig ist, dass die Benutzeroberfläche wie aus einem Guss aussieht. Mit dieser Variante hat er die Möglichkeit, sein eigenes Design für die Bezahlung zu implementieren.

3.8 Zahlungsinformationen darstellen

Um die Zahlungsinformationen dem Käufer mitzuteilen, gibt es zwei verschiedene Varianten:

1. Eine Library schreiben, welche das Rendern beim Händler übernimmt.
2. CHainGate stellt eine Payment Page zur Verfügung.

Library

Die angebotene Library kann vom Händler auf seinem Shop benutzt werden.

Die Vorteile einer Library sind:

+ Das einheitliche Design des Händlers kann besser beibehalten werden.
+ Der Händler kann selbst entscheiden, welche Informationen dem Kunden angezeigt werden.

Die Nachteile einer Library sind:

− Mehr Entwicklungsaufwand für den Händler.
− Spätere Anpassungen an der Library können zu schwer kalkulierenden Folgen führen.
− Mehr Supportaufwand für CHainGate.

Payment Page

Die Vorteile einer Payment Page sind:

+ Kein zusätzlicher Entwicklungsaufwand für den Händler.
+ Änderungen an der Payment Page können getätigt werden, ohne dass der Händler Anpassungen vornehmen muss.

Die Nachteile einer Payment Page sind:

− Das einheitliche Design kann nicht übernommen werden.

Entscheidung

CHainGate hat sich für eine eigene gehostete Payment Page entschieden, da eine Library für den Händler aufwändiger ist. Es besteht jedoch eine Proof of Concept-Library im CHainGate-Repository, welche die Idee demonstriert.

3.9 Ethereum Anbindung

Die Anbindung an Ethereum wurde über eine API implementiert namens «Infura». Infura ist ein Ethereum Node Provider, welcher eine API über HTTP anbietet. Der Service übernimmt das Aufsetzen eines Ethereum-Nodes und bietet eine einfache REST-Schnittstelle, um Informationen von der Blockchain zu erhalten. Bei Infura gibt es ein Freemium-Packet von bis zu 100’000 Aufrufen pro Tag, was für einen MVP ausreichend ist.

Für einen ersten Prototypen war es wichtig, schnell ein lauffähiges System zu haben. Ebenfalls wurde in einem Team gearbeitet und durch das Benutzen eines Node-Anbieters in der Cloud, können so Arbeiten schnell auf andere Teammitglieder übertragen werden. Ebenfalls besitzt Infura eine Redundanz, falls ein Node ausfällt.

Die Alternative wäre das Hosten eines eigenen Nodes. Das Aufsetzen mit einem Tool wie Geth ist simpel. Mit Geth gibt es drei verschiedene Modis:

- Full Node
- Light Node
- Snap Sync

Full Node

Ein Full Node ist ein Node, welcher die ganze Blockchain vom Genesis-Block an synchronisiert. Dies kann auf einer starken Maschine bis zu zehn Tage dauern und benötigt bis zu 1 TB Speicherplatz. [59]

Light Node

Ein Light Node lädt im Vergleich zum Full Node nur die Header der Blöcke herunter und kommuniziert mit einem Full Node. Der Light Node agiert lokal wie ein Full Node. Falls ein Light Node Informationen von einem Block benötigt, fragt dieser einen Full Node an. Dies kann dazu führen, dass Full Nodes überfordert werden mit Anfragen von Light Nodes und Anfragen nicht beantwortet werden.

Das Synchronisieren benötigt hingegen nur 15-20 Minuten und etwa 500 MB Speicherplatz.

Snap Sync

Der Snap Sync wurde vor einem Jahr implementiert und löste den damaligen «fast sync» ab. Dieser Modus lädt alle Blöcke herunter, prüft die Proof of Work-Garantie, verarbeitet jedoch nicht die Transaktionen. Daher müssen Informationen wie die Nonce oder Balance einer Adresse in einem separaten Prozess(state trie download phase) synchronisiert werden. Im Vergleich zum Vorgänger werden die State Daten heruntergeladen und der Merkle Trie lokal aufgebaut, was den Prozess erheblich verschnellert. Ein Nachteil vom Snap Sync ist, dass Informationen von Blöcken, die älter sind als 128 Blöcke nicht vorhanden sind, da der Status vom aktuellen Snapshot plus 128 Blöcke genommen wird. Eine Synchronisation dauert mit dem Snap Sync gemäss Ethereum selbst etwa zwei Stunden. [60–63]

3.10 Bitcoin Anbindung

Es gibt verschiedene Möglichkeiten, wie der Bitcoin-Service angebunden werden kann. Hier werden fünf verschiedene Varianten erklärt und ausgewertet. Dabei gibt es verschiedene Komponenten, welche miteinander kombiniert werden können.

Bitcoin Node

Der Bitcoin Node bietet eine JSON RPC Schnittstelle an, um mit der Blockchain zu kommunizieren. Zusätzlich bietet der Node zwei Möglichkeiten an, wie Benachrichtigungen über neue Transaktionen oder Blöcke erhalten werden können. Die Erste ist über eine «Notifications»-Option in der Konfigurationsdatei. Dort kann ein Skript angegeben werden, welches bei den entsprechenden Events ausgeführt wird. Die Zweite ist über ZeroMQ, eine Messaging-Library. [64–66]

Bitcoin Wallet RPC

Bitcoin bietet eine Wallet RPC an. Die RPC vereinfacht viele Abfragen. Zum Beispiel kann abgefragt werden, wie viele UTXO eine Adresse hat. Zusätzlich kann Geld darüber verschickt oder eine neue Adresse erstellt werden. Um Geld zu versenden, braucht der Node Zugriff auf den Private Key. Anstatt der Wallet RPC für das Versenden von Geld kann auch manuell eine Transaktion erstellt werden. Dies hat den Vorteil, dass der Node keinen Zugriff auf den Private Key braucht, jedoch muss das Erstellen und Signieren manuell erstellt werden.

Ohne Wallet-RPC ist es aufwendig, das Guthaben einer Adresse herauszufinden, weil es keinen globalen Status mit dem Guthaben pro Adresse gibt. Entweder wird eine externe API genutzt, um das Guthaben einer Adresse abzufragen oder es muss durch die gesamte Blockchain iteriert und alle Transaktionen zusammengerechnet werden. [64]

Bitcoin Watch-Only Wallet

Ein Watch-Only Wallet kann nur für das Lesen von Informationen benötigt werden. Dabei wird nur der Public Key benötigt. Eine Transaktion kann nicht über die Wallet RPC erstellt werden, weil der Private Key fehlt. Um Geld zu versenden, muss eine Transaktion manuell erstellt und signiert werden. [67]

BTCWallet

BTCWallet ist eine Wallet Implementation in Go. Um diesen zu nutzen, wird jedoch der btcd Node (Bitcoin Node Implementation in Go) gebraucht. Der Vorteil dieser Variante ist, dass sie mehr Funktionalität anbietet, als der offizielle Bitcoin Node und das Wallet unabhängig vom Node ist. [68]

Externe API

Es gibt auch externe APIs, welche die Kommunikation zu Bitcoin für einen übernehmen. Ein Beispiel ist BlockCypher. Hierbei ist wichtig, dass der API-Betreiber keinen Zugriff auf die Private Keys besitzt. Zudem haben viele API-Betreiber Einschränkungen bei der kostenlosen Nutzung. [69]

Auswertung

In der nachfolgenden Tabelle 5 wurden die fünf Varianten ausgewertet.

	Bitcoin Node mit Wallet	Bitcoin Node mit Watch-Only Wallet	Bitcoin Node ohne Wallet	BTCWallet	Externe API
Private Keys unabhängig vom Node	Nein	Ja	Ja	Ja	Ja
Benachrichti-gungen mög-lich	Ja	Ja	(Ja)	Ja	(Ja)
Restriktionen	Keine	Keine	Keine	Btcd Node notwendig	Ja (Je nach Preis-Tarif)
Komplexität	Tief	Mittel	Hoch	Tief	Mittel
Erstellen einer Adresse mög-lich	Einfach	Einfach	Mittel	Einfach	Mittel
Skalierbarkeit	Schlecht	Mittel	Gut	Mittel	Gut
UTXO Abfrage einer Adresse	Einfach	Einfach	Schwer	Einfach	Einfach

Tabelle 5: Bitcoin Anbindung Analyse

Für diese Arbeit wurde sich für eine Anbindung mit einem Bitcoin Node inklusive Wallet entschieden. Trotz der schlechten Skalierung ist es aktuell die beste Entscheidung, weil die Komplexität niedrig ist, die UTXO einfach abgefragt werden kann, es keine Restriktionen und keine Abhängigkeit zu Dritten gibt.

4. Umsetzung

In diesem Kapitel wird gezeigt, wie CHainGate umgesetzt wurde.

4.1 Architektur

In diesem Abschnitt wird die umgesetzte Architektur aufgezeigt. Sie besteht aus fünf verschiedenen Microservices und mehreren Umsystemen. Dabei besitzt jede Kryptowährung ihren eigenen Service, damit sie unabhängig voneinander entwickelt werden können und technologieunabhängig sind.

In der Abbildung 8 ist die gesamte System-Architektur visualisiert.

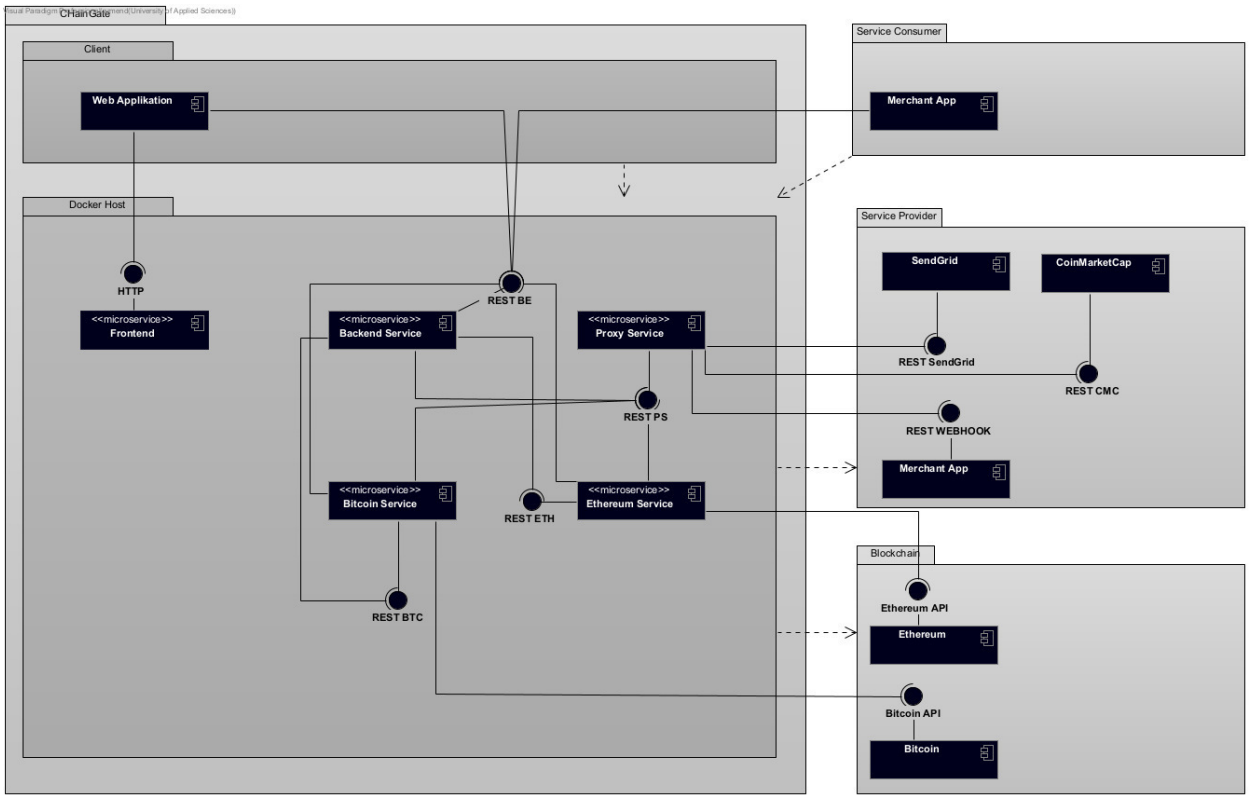

Abbildung 8: CHainGate Architektur

4.1.1 Frontend

Das Frontend besteht aus einer Next.js[1]-Applikation, welche mit REST-Aufrufen Daten aus dem Backend erhält

4.1.1.1 WebSocket

Für die Payment Page wurden WebSockets eingesetzt. Zuerst wurde die Implementation der Vorarbeit bei FlatFeeStack als Vorlage verwendet. Nach ersten Erfahrungen kam es zu der Limitation, dass nur ein

[1] Framework für React mit Features wie zum Beispiel Server-Side-Rendering.

Tab gleichzeitig aktualisiert werden konnte, vom Backend. Dadurch wurde die Implementation mit Go-Channel neu implementiert. [70, 71]

4.1.2 Backend-Service

Der Backend-Service ist die zentrale Anlaufstelle für alle HTTP-Anfragen. Er besitzt drei Funktionen:

- Die Anfragen vom Frontend entgegennehmen und die nötigen Daten bereitstellen oder bearbeiten.
- Das Entgegennehmen der Händleranfragen. Dabei kann ein Händler eine neue Zahlung erstellen und das Backend kommuniziert mit dem nötigen Blockchain Service, um die Anfrage weiterzuleiten.
- Das Aktualisieren des Zahlungsstatus. Den aktuellen Status erhält der Service jeweils von den verschiedenen Blockchain Services. Sobald eine Aktualisierung stattfindet, ruft der Blockchain Service den Backend-Service auf. Das Backend notifiziert dabei den Händler über den definierten Web-Hook.

4.1.3 Proxy-Service

Der Proxy-Service ist für alle Verbindungen ausserhalb des Systems verantwortlich. Aktuell ist er verantwortlich für das Senden der E-Mails über SendGrid, das Abfragen des Wechselkurses über CoinMarketCap und das Aufrufen der WebHooks bei den Händlern.

Aus folgenden Gründen wurde entschieden einen Proxy-Service zu erstellen, anstatt direkt auf die externen Services zuzugreifen.

- Nutzen unterschiedliche Services das gleiche Umsystem, so muss die Logik nicht mehrmals implementiert werden. Zum Beispiel kann jeder Blockchain Service den Wechselkurs abfragen, ohne den direkten Zugriff zu implementieren.
- Muss ein Umsystem wie SendGrid oder CoinMarketCap ausgewechselt werden, muss das nur im Proxy-Service getan werden. Dabei ändert sich die Schnittstelle nicht und alle Nutzer dieser Schnittstelle können den Service ohne Anpassungen weiterhin nutzen.
- Gibt es bei einer externen API eine Anpassung, kann dies bei einer zentralen Stelle aktualisiert werden.
- Der Proxy-Service kann zusätzliche Funktionalität anbieten wie zum Beispiel einen Cache, um die Performance zu erhöhen.

4.1.4 Ethereum-Service

Der Ethereum-Service ist für die Kommunikation mit der Ethereum-Blockchain verantwortlich und dessen Zahlungen.

4.1.5 Bitcoin-Service

Der Bitcoin-Service ist für die Kommunikation mit der Bitcoin Blockchain verantwortlich und dessen Zahlungen.

4.1.6 Merchant App

Die Merchant App ist die Händler Applikation. Sie nimmt jeweils die Funktion eines Consumers und eines Providers an. Erstellt der Merchant eine Anfrage für eine neue Zahlung, dann fungiert er als Consumer. Dabei muss der Händler immer einen WebHook Endpunkt anbieten, um über Änderungen der Zahlung informiert zu werden.

4.1.7 SendGrid

SendGrid wird verwendet, um E-Mails zu versenden. Aktuell werden bei der Registration E-Mails nur für die Verifikation des Kontos erstellt.

Weitere Anwendungsfälle für das Versenden von E-Mails wären zum Beispiel:

- Händler Notifizieren, wenn der WebHook nicht erreichbar ist.
- Marketing E-Mails
    - Neuste Funktionen vorstellen
    - Spezialangebote

4.1.8 CoinMarketCap

CoinMarketCap wird aktuell für das Abfragen des Wechselkurses genutzt.

4.2 Schnittstellen

Alle Services bieten eine OpenAPI Schnittstelle an. Zusätzlich bietet jeder Service ein SwaggerUI an für die Dokumentation. Die Schnittstelle zum Händler ist wichtig, weil sie für das Erstellen von Zahlungen genutzt wird.

Zurzeit gibt es einen Endpunkt für «payment» und einen für «invoice».

Payment: Der Payment Endpunkt wird genutzt, um eine Zahlung **ohne** Payment Page zu erstellen.

Invoice: Der Invoice Endpunkt wird genutzt, um eine Zahlung **mit** Payment Page zu erstellen.

Alle Endpunkte inklusive Details sind im Anhang als OpenAPI Spezifikation vorhanden.

4.3 Zahlungsprozess

In diesem Abschnitt wird der gesamte Zahlungsprozess aufgezeigt. Wie die einzelnen Blockchain Services die Zahlungen technisch umsetzen, wird im Kapitel 4.7 Ethereum-Service und 4.8 Bitcoin-Service erklärt.

In der Abbildung 9 ist der gesamte Prozess als Sequenzdiagramm visualisiert.

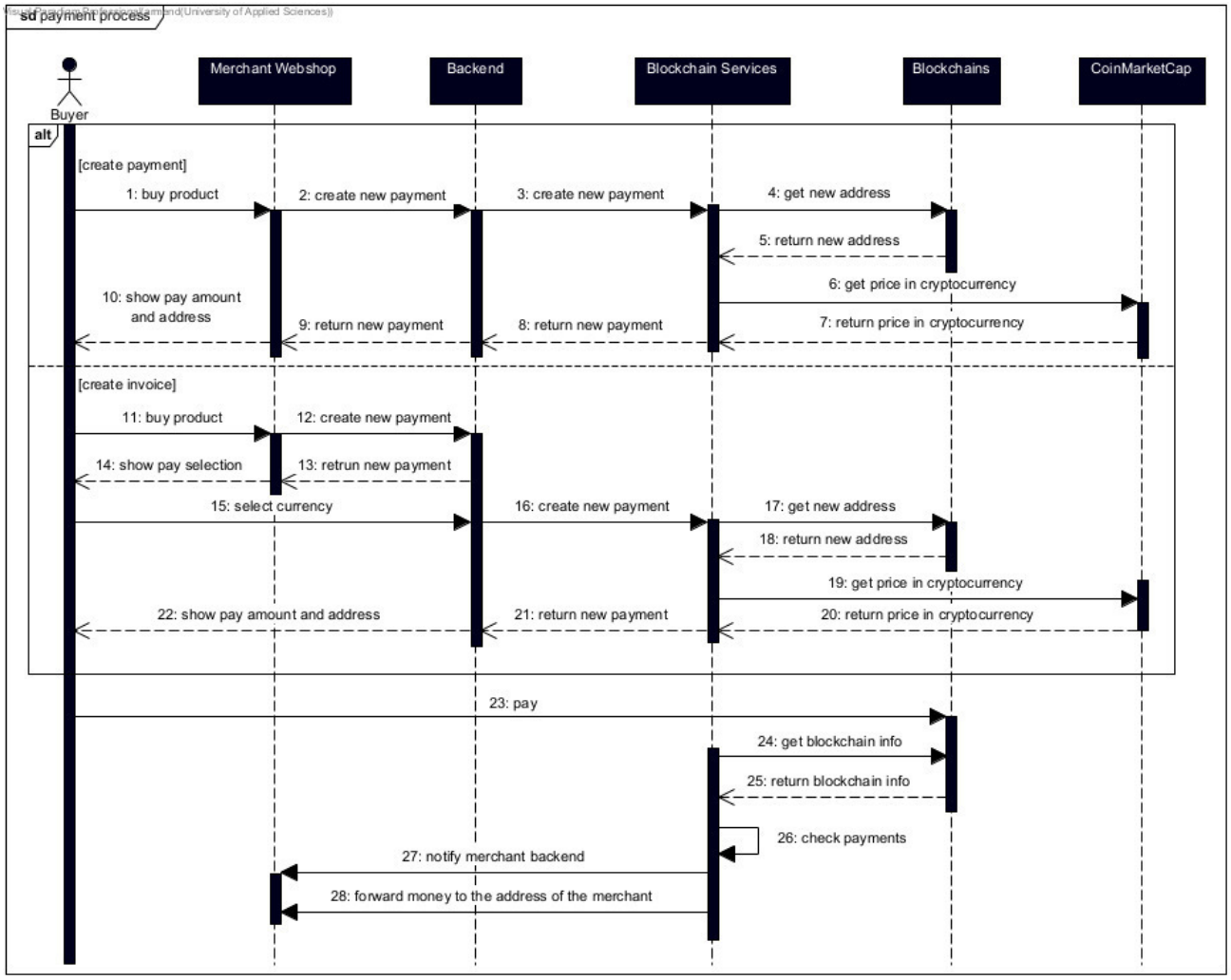

Abbildung 9: Zahlungsprozess detailliert

Zahlung erstellen (1 – 10)

In diesem Teil wird eine neue Zahlung über die «create payment» Methode erstellt. Dabei leitet das Backend die Anfrage an die richtige Blockchain weiter für das Erstellen der Zahlung. Der Blockchain Service benötigt zuerst eine Empfangsadresse. Entweder existiert eine im Pool, welche frei ist, oder er erstellt sich eine neue. Zusätzlich benötigt der Service den Wechselkurs zwischen Fiatgeld und Kryptowährung. Dafür macht er eine Abfrage an CoinMarketCap über den Proxy-Service. In der Abbildung 9 ist der Proxy-Service aus Übersichtsgründen nicht aufgezeichnet.

Rechnung erstellen (11 – 22)

Wenn eine Zahlung über die «create invoice» Methode erstellt wird, wird dieser Teil ausgeführt. In diesem Fall erstellt das Backend eine Zahlung, ohne es an einen Blockchain Service weiterzuleiten. Bei der Antwort an den Händler wird eine URL mitgegeben, welche der Kunde nutzen kann, um seine Zahlung fortzuführen. Der erste Schritt ist es, die gewünschte Kryptowährung auszuwählen. Sobald diese gewählt wurde, wird die Anfrage an den entsprechenden Blockchain Service weitergeleitet, welche eine Zahlung inklusive

Adresse und Betrag erstellt. Diese Informationen werden dem Kunden auf der Payment Page angezeigt und er kann den angezeigten Betrag an die entsprechende Adresse senden.

Zahlungsverarbeitung (23 – 28)

Sobald die Zahlung erstellt wurde, wird auf die Zahlung vom Kunden gewartet. Nachdem der Kunde bezahlt hat, wird die Zahlung überprüft und verarbeitet. Dabei erhält der Händler über WebHooks, Updates zu dem Zahlungsstatus. Nachdem die Zahlung bestätigt wurde, wird sie an den Händler weitergeschickt.

4.4 Zahlungsstatus

Eine Zahlung kann verschiedene Status besitzen. Die gesamte Verarbeitung wurde in einem Zustandsdiagramm abgebildet. Sobald gewisse Bedingungen erfüllt sind, erhält die Zahlung einen neuen Status. Wenn irgendwo ein Fehler geschieht, darf sich der Status nicht verändern. Es kann jedoch vorkommen, dass es zwischen der Blockchain und dem Zahlungsstatus eine Inkonsistenz gibt. Um dieses Problem zu lösen, ist die Blockchain die «single source of truth». Dabei werden die Informationen von der Blockchain mit der Datenbank von CHainGate bei einem Problem abgeglichen und aktualisiert.

Auf der Abbildung 10 sind alle Zustände und Übergänge visualisiert.

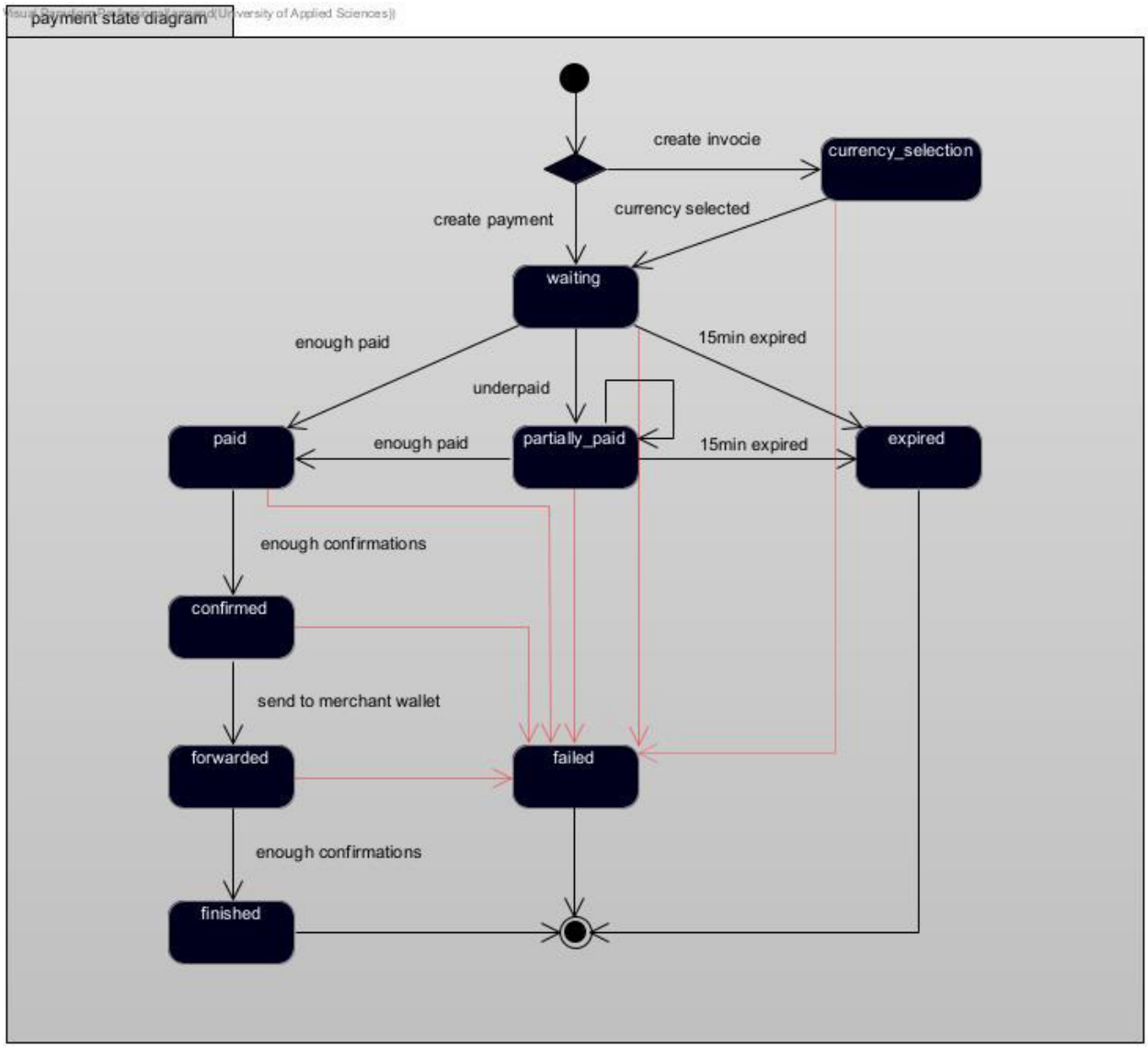

Abbildung 10: Zustandsdiagramm Zahlung

Wenn eine Zahlung über «create invoice» erstellt wird, erhält sie den Zustand «currency_selection». Sobald der Kunde eine Währung ausgewählt hat, erhält er den Zustand «waiting». Wenn eine Zahlung über «create payment» erstellt wird, erhält sie direkt den Zustand «waiting», weil der Händler die Kryptowährung bereits bei der Erstellung angibt.

Wenn der Käufer genug bezahlt hat, erhält die Zahlung den Zustand «paid». Sollte er jedoch zu wenig bezahlt haben, erhält er den Zustand «partially_paid». Wenn mehrere Zahlungen getätigt werden, jedoch die Gesamtsumme immer kleiner als der zu bezahlende Betrag ist, bleibt die Zahlung im Zustand «partially_paid». Erst wenn die Summe gleich oder grösser, wie der zu bezahlende Betrag ist, wechselt die Zahlung den Zustand zu «paid».

Eine Zahlung in dem Zustand «waiting» oder «partially_paid» läuft 15 Minuten nach der Erstellung ab und wird nicht weiterverarbeitet.

Damit eine Zahlung von «paid» nach «confirmed» kommt, muss sichergestellt werden, dass eine gewisse Anzahl Blöcke bestätigt wurden. Je nach Blockchain ist dieser Teil individuell zu handhaben. Eine Zahlung, welche «confirmed» ist, gilt als sicher und kann an den Händler weitergeleitet werden und erhält den Zustand «forwarded».

Auch die weitergeleitete Transaktion muss beobachtet werden, dass eine gewisse Anzahl an Blöcken bestätigt wurden, damit sie als sicher gilt. Sobald dies geschehen ist, ist die Zahlung beenden und im Zustand «finished».

Wenn bei einem Zustand ein Fehler geschieht, welcher nicht selbst behoben werden kann, landet die Zahlung im Zustand «failed».

4.5 Backend-Service

Der Backend-Service übernimmt neben der Koordination zwischen den einzelnen Blockchain Services noch folgende vier wichtige Funktionen.

1. Registration / Authentisierung
2. API-Key Generierung
3. API-Key Authentisierung
4. Hashen der WebHooks zur Überprüfung der Echtheit

In den Nachfolgenden Unterkapiteln wird zuerst die Datenbank und anschliessend alle vier Funktionen beschrieben.

4.5.1 Datenbank

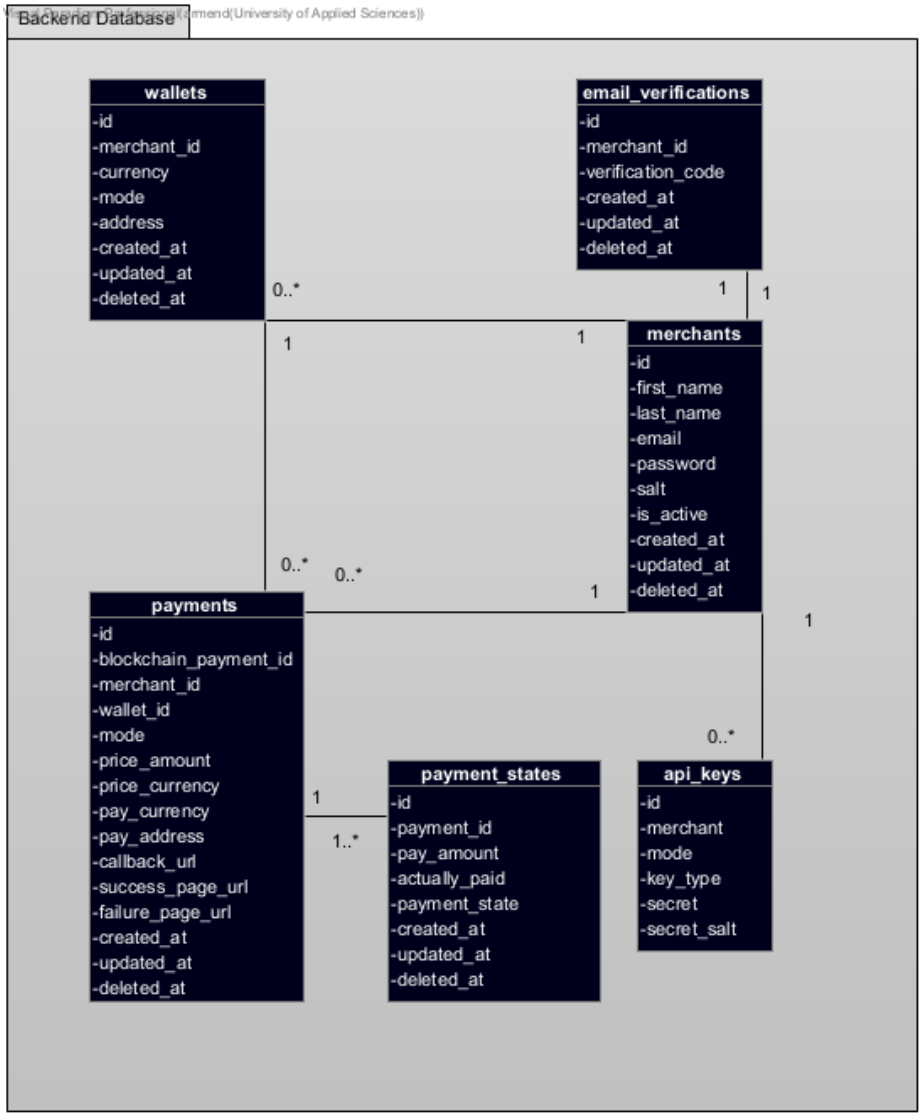

Abbildung 11: Backend Datenbank

In der Abbildung 11 ist das Datenbankmodell des Backend Services ersichtlich. Die Tabellen «email_verifications», «merchants», «api_key» und «wallets» werden für die Benutzerverwaltung gebraucht. Die Tabellen «payments» und «payment_states» werden gebraucht, um dem Händler all seine Zahlungen auf dem Dashboard anzuzeigen. Es gibt jedoch noch eine zweite Variante, wie das gleiche erreicht werden könnte.

Variante 1

Die Zahlungsinformationen werden nicht nur in den Blockchain Services abgespeichert, sondern auch auf der Datenbank vom Backend Service. Dabei muss das Schema auf der Backend Datenbank so aufgebaut sein, dass es keine Schemaänderungen braucht, wenn eine neue Kryptowährung hinzukommt.

Diese Variante hat folgende Vorteile:

+ Es müssen nicht alle Blockchain Services abgefragt werden.
+ Wenn ein Blockchain Service ausfällt, können weiterhin seine Zahlungen angezeigt werden.
+ Für die Implementation einer Payment Page, muss die Zahlung zuerst auf dem Backend gespeichert werden, weil der Käufer zu einem späteren Zeitpunkt entscheidet, in welcher Kryptowährung er bezahlen will.

Diese Variante hat folgende Nachteile:

– Die Daten werden redundant gehalten.
– Es kann eine Inkonsistenz zwischen den Daten des Backend Services und den Blockchain Services entstehen.

Variante 2

Jeder Blockchain Service hat seine Zahlungen und deren Status in seiner Datenbank abgespeichert. Damit der Backend Service alle Zahlungen eines Händlers an das Frontend schicken kann, muss dieser alle Blockchain Services abfragen und die Zahlungen des Händlers abrufen.

Diese Variante hat folgende Vorteile:

+ Es müssen keine redundanten Daten auf dem Backend Service gespeichert werden.
+ Es können keine Inkonsistenzen auftreten, weil es keine doppelten Daten gibt.

Diese Variante hat folgende Nachteile:

– Es müssen immer alle Blockchain Services abgefragt werden, weil der Backend Service nicht weiss, für welche Kryptowährungen bereits Zahlungen erstellt worden sind.
– Es entstehen viele REST Abfragen über das gesamte System.
– Fällt ein Blockchain Service aus, sind diese Informationen nicht verfügbar.

Entscheidung

Wie in der Abbildung 11 zu sehen ist, wurde sich für die erste Variante entschieden. Der Grund ist, dass für die Payment Page Zahlungsinformationen auf dem Backend Service benötigt werden und nicht jedes Mal alle Blockchain Services abgefragt werden müssen. Sollten in Zukunft 30 Kryptowährungen angebunden sein, werden bei jedem Händler immer alle 30 Blockchain Services abgefragt. Dabei kann der Händler nur eine Kryptowährung nutzen haben. Dies ist sehr ineffizient.

4.5.2 Registration / Authentisierung

Wenn sich ein Händler registriert, wird ein 6-stelliger Verifikationscode erstellt, welcher zufällig ist. Danach ist das Konto inaktiv und der Händler erhält eine E-Mail mit einem Verifikationslink. Sobald die E-Mail verifiziert wurde, ist das Konto aktiv und der Händler kann sich anmelden.

Der Registrierungsprozess ist in der Abbildung 12 in einem Sequenzdiagramm visualisiert.

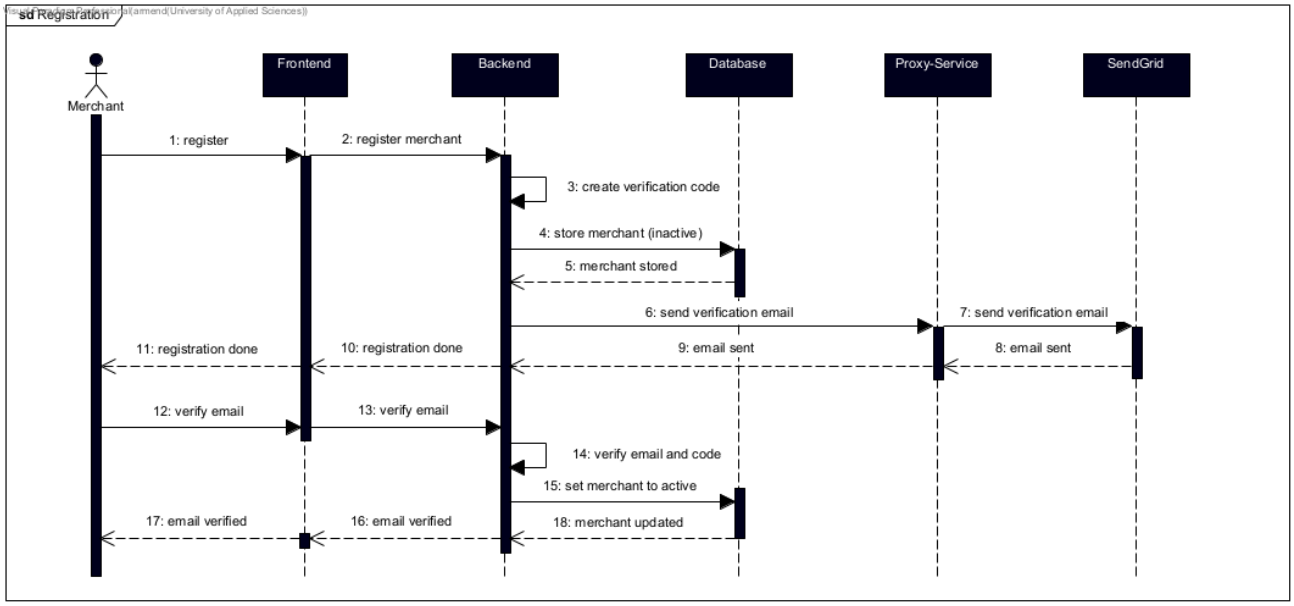

Abbildung 12: Registrationsprozess

Für die Authentisierung wird JWT genutzt. In Zukunft könnte eine Authentisierung über externe Provider wie Google oder GitHub integriert werden.

4.5.3 API-Key Generierung

Der API-Key besteht aus zwei Teilen. Der erste Teil ist die ID des API-Keys und der zweite Teil ist ein geheimer zufälliger Schlüssel. Die ID wird genutzt, um bei der Authentifizierung herauszufinden, zu welchem Händler der API-Key gehört.

Der Ablauf der Generierung ist in der Abbildung 13 visualisiert.

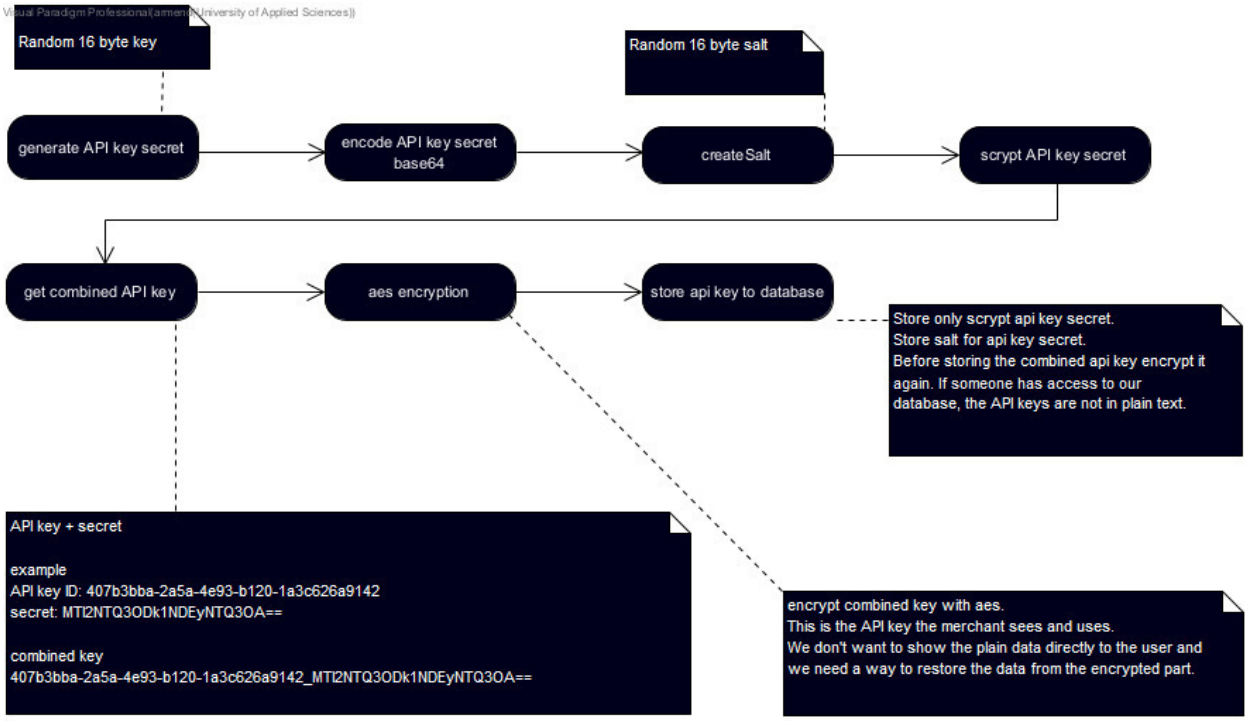

Abbildung 13: Ablauf API-Key Generierung

1. Als Erstes wird ein 16-stelliger zufälliger Schlüssel generiert. Dieser ist geheim.
2. Der Schlüssel wird zu base64 encoded.
3. Bevor der Schlüssel mit scrypt verschlüsselt werden kann, wird ein 16-stelliger zufälliger Salt generiert.
4. Der geheime Schlüssel wird mit scrypt verschlüsselt und nur in verschlüsselter Form abgespeichert.
5. Um einen API-Key zu erhalten, wird die API-Key ID (Datenbank ID) mit dem geheimen Schlüssel kombiniert.
6. Damit der Händler keine Rohdaten als API-Key erhält, wird er mit AES verschlüsselt. Diesen Schlüssel nutzt der Händler, um seine Anfragen an CHainGate zu erstellen und um WebHooks zu validieren. Durch die AES-Verschlüsslung kann bei einer Anfrage der API-Key entschlüsselt werden und anhand der ID weiss CHainGate um welchen Händler es sich handelt.
7. Bevor die Daten in die Datenbank gespeichert werden, wird der API-Key nochmals verschlüsselt.

4.5.4 Authentisierung

Bei der API-Key Authentisierung wird der API-Key zuerst entschlüsselt und anschliessend wird die ID und der geheime Schlüssel herausgelesen. Anhand der ID und des Schlüssels wird überprüft, ob der API-Key gültig ist.

In der Abbildung 14 ist der Ablauf visualisiert.

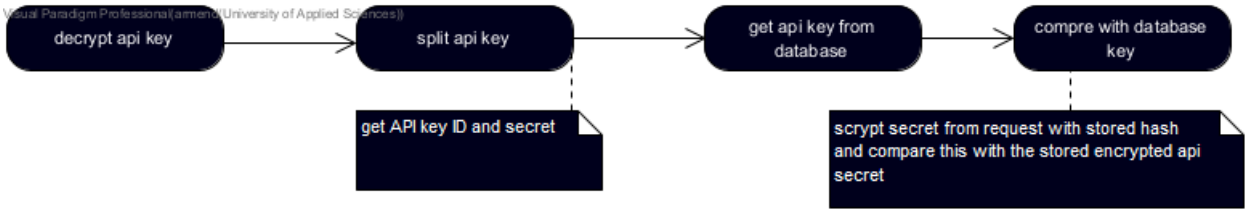

Abbildung 14: Ablauf Authentisierung

4.5.5 Hashen der Nachricht zur Überprüfung der Echtheit

Weil CHainGate Updates an die Händler über einen öffentlichen WebHook schickt, ist es wichtig, dass es eine Methode gibt, wie die Händler überprüfen können, ob es sich um ein echtes Update von CHainGate handelt oder nicht.

In der Abbildung 15 ist zu sehen, wie eine WebHook Nachricht aufgebaut ist. Dabei gibt es einen Daten- sowie Signatur-Teil. Im Daten-Teil sind alle Information zur Zahlung hinterlegt. Der Signatur-Teil wird genutzt, damit der Händler überprüfen kann, ob dieses Update von CHainGate kommt.

```
{
  "data": {
    "payment_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "pay_address": "string",
    "price_amount": 0,
    "price_currency": "usd",
    "pay_amount": "string",
    "pay_currency": "eth",
    "actually_paid": "string",
    "payment_state": "waiting",
    "created_at": "2022-05-24T15:42:57.859Z",
    "updated_at": "2022-05-24T15:42:57.859Z"
  },
  "signature": "string"
}
```

Abbildung 15: WebHook Nachricht

Die Signatur wird erstellt, indem über die **sortierten** Daten ein hmac mit dem geheimen API-Key vom Händler erstellt wird. Will der Händler die Nachricht auf ihre Echtheit überprüfen, muss er über die **sortierten** Daten einen hmac mit seinem API-Key erstellen und seinen Wert mit der Signatur überprüfen. Stimmen diese überein, ist die Nachricht echt. Damit diese Methode funktioniert, ist es wichtig, dass nur CHainGate und der Händler den API-Key kennen.

Ein Beispiel einer Implementation in Node.js kann auf GitHub gefunden werden.

4.6 Allgemein Blockchain Services

Im Folgenden wird erläutert, wie ein Blockchain Service im Grundsatz aufgebaut ist. Durch die Microservicearchitektur gibt es jedoch im Detail feine Unterschiede, welche erläutert werden, um auf die Bedürfnisse der einzelnen Blockchains besser eingehen zu können.

4.6.1 Service Architektur

Gemäss der Abbildung 16 sind die Blockchain Services nach einer Schichten-Architektur aufgebaut. In der obersten Schicht sind die REST-Eingangspunkte. Es gibt einen «CreatePayment» Endpunkt, welcher Anfragen für das Erstellen neuer Zahlungen entgegennimmt. Die «Business logic / Service» Schicht verarbeitet die Anfragen und kommuniziert mit der Blockchain. Die «Repository» Schicht ist für den Datenbankzugriff zuständig.

Eine Blockchain besitzt normalerweise mindestens drei unterschiedliche Umgebungen: Mainnet, Testnet und lokal. Der Zugriff auf diese geschieht über JSON-RPC oder einer REST-Schnittstelle. Üblicherweise kann auch über ein CLI Tool mit der Blockchain kommuniziert werden. Für den produktiv Einsatz wird das Mainnet angesprochen. Um Testzahlungen durchzuführen, wird das Testnet verwendet. Das lokale Netz wird nur für lokale Tests genutzt.

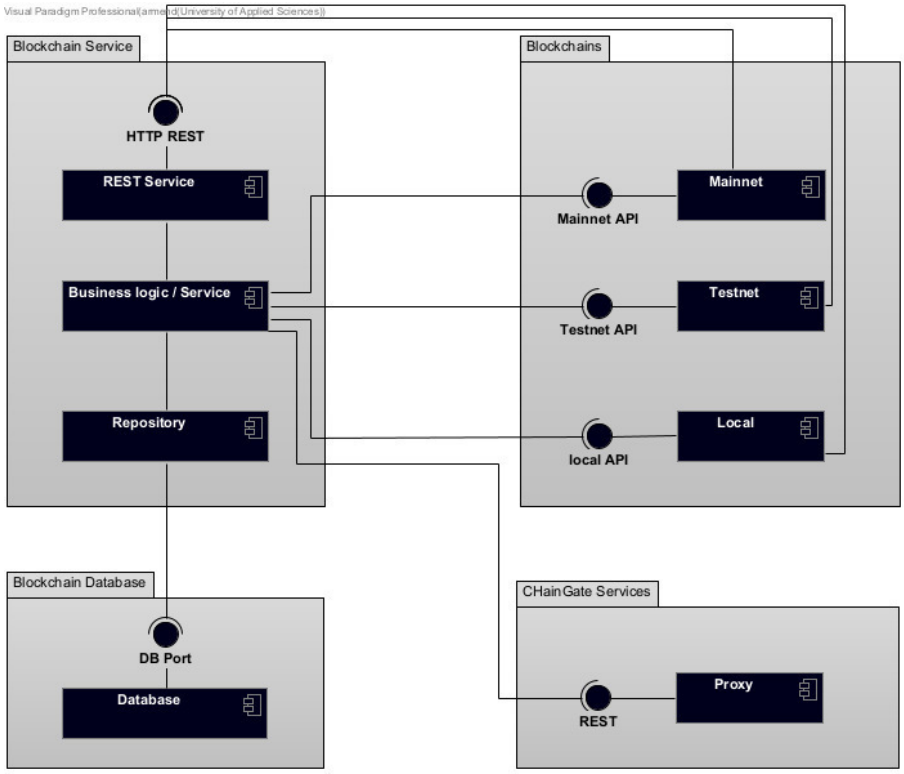

Abbildung 16: Blockchain Architektur

4.6.2 Datenbank

Jeder Blockchain Service besitzt seine eigene Datenbank. Dabei sieht das Schema zwischen Bitcoin und Ethereum ähnlich aus. In der Abbildung 17 ist ein einheitliches Model zu sehen, mit den unterschieden zwischen Bitcoin und Ethereum.

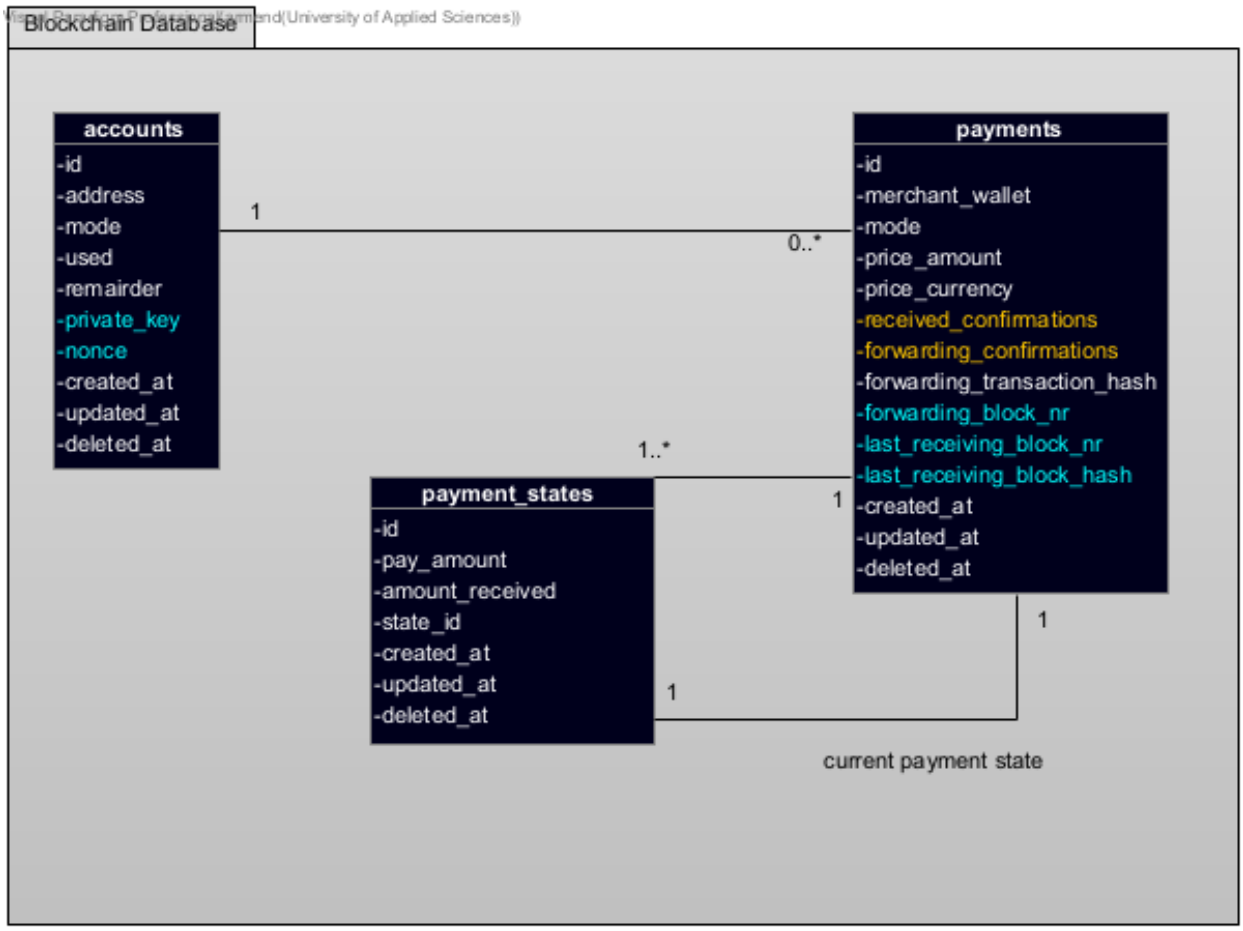

Abbildung 17: Blockchain Datenbank Modell

Allgemein Ethereum Bitcoin

Tabelle «accounts»

Die «accounts» Tabelle speichert die Empfangsadressen. Wird eine neue Zahlung erstellt, wird entweder eine freie Adresse aus der accounts Tabelle ausgewählt oder es wird eine neue Empfangsadresse auf der Blockchain erstellt und in der Datenbank abgespeichert.

address: Die Blockchain Empfangsadresse. Der Kunde bezahlt auf diese Adresse, welche CHainGate gehört.

mode: Für welches Netz das Wallet gedacht ist. Ohne mode kann es Probleme mit dem folgenden Feld «Remainder» geben, der unterschiedlich pro Netz ist.

used: Ein Flag, ob die Empfangsadresse bereits in einer offenen Zahlung benutzt wird oder nicht.

remainder: Das überschüssige Geld. Wird benötigt für Retouren und Wiederverwendung von einer Adresse.

private_key (eth): der verschlüsselte Private Key

nonce (eth): Der Account-Nonce, um doppelte Zahlungen zu verhindern.

Tabelle «payments»

Die «payments» Tabelle speichert alle Zahlungen. Sobald eine neue Zahlung erstellt wird, wird ein neuer Eintrag erstellt. Der Status der Zahlung wird in der Tabelle «payment_states» hinterlegt. Jede Zahlung hat zusätzlich eine Referenz auf den aktuellen Status.

merchant_wallet: Die Adresse des Händlers. Sobald der Kunde bezahlt hat, wird das Geld an diese Adresse weitergeleitet.

mode: Ob die Zahlung zum Mainnet oder Testnet gehört.

price_amount: Der zu bezahlende Preis in Fiatgeld.

price_currency: Die Währung in Fiatgeld.

received_confirmations (btc): Wie viele Bestätigungen die Transaktion des Käufers von ihm zu CHainGate hat.

forwarding_confirmations (btc): Wie viele Bestätigungen die Transaktion von CHainGate zu dem Händler hat.

forwarding_block_hash: Der Hash der Transaktion von CHainGate zu Händler.

Last_receiving_block_nr (eth): Die Nummer des Blocks, in der die letzte benötigte Transaktion für den Status «paid» enthält.

Last_receiving_block_hash (eth): Der Hash des Blocks, in der die letzte benötigte Transaktion für den Status «paid» enthält.

forwarding_block_nr (eth): Die Nummer des Blocks, in der die Transaktion von CHainGate zum Händler ist.

Tabelle «payment_states»

Die «payment_states» Tabelle speichert alle Status einer Zahlung.

pay_amount: Der zu bezahlende Betrag in Kryptowährung.

amount_received: Wie viel BTC oder ETH CHainGate erhalten hat.

state_id: Id des Status.

4.7 Ethereum-Service

Der Ethereum-Service beobachtet die einzelnen Transaktionen und ist verantwortlich für das Ausführen der einzelnen Schritte. Bei einem Statuswechsel informiert der Service den Backend-Service.

4.7.1 Erzeugung eines Wallets / einer Adresse

Bei der Implementierung wurde auf die go-ethereum Library zurückgegriffen. Diese bietet für die benötigen Operationen bereits Methoden an:

```
privateKey, err := crypto.GenerateKey()

publicKey := privateKey.Public()

address := crypto.PubkeyToAddress(*publicKeyECDSA).Hex()
```

4.7.2 Transaktionskosten

Zur Simplifikation des Problems wurde sich in dieser Arbeit auf die empfohlenen Werte der benutzten Go-Ethereum-Library gestützt. Diese bietet bereits zwei Methoden, welche es erlauben den momentanen Gaspreis und das momentane Trinkgeld abzufragen: `SuggestGasPrice, SuggestGasTipCap`

Dadurch fällt ein Grossteil der Berechnung und Überwachung der Blockchain weg. Um die Theorie hinter der Berechnung zu überprüfen, wurden mehrere Tests mit fixem Gaspreis implementiert. Dadurch wurde sichergestellt, dass kein Geld zu viel überwiesen wird oder Geld auf einer Adresse liegen bleibt.

4.7.3 Abfragen des Guthabens

Um das Guthaben einer Adresse abzufragen, wird auf Infura zugegriffen. Durch eine JSON-RPC-Anfrage und der genannten Library kann der aktuelle Betrag von einer Adresse abgefragt werden:

```
client.BalanceAt(context.Background(), address, nil)
```

4.7.4 Interne Transaktionen

Bei Ethereum gibt es neben den üblichen Transaktionen von Adresse zu Adresse auch «interne Transaktionen». Diese werden nicht über die Blockchain-API mitgeteilt und müssen separat gehandhabt werden. Interne Transaktionen sind beispielweise Transaktionen, welche von Smart Contracts ausgeführt werden. Bevor eine Zahlung abläuft, wird das Guthaben auf der Adresse überprüft, damit interne Transaktionen nicht verloren gehen. [72]

Die Implementierung könnte zukünftig verbessert werden, wenn eine API für interne Transaktionen angebunden wird. [73]

4.8 Bitcoin-Service

Der Bitcoin-Service ist für die Kommunikation mit dem Bitcoin-Node zuständig. Neben den üblichen REST-Endpoints einer Blockchains gibt es zusätzlich noch einen «BlockNotify» und «WalletNotify» Endpunkt. Sie werden vom Bitcoin Node aufgerufen, um den Service zu informieren, wenn ein neuer Block erstellt wurde oder wenn eine neue Zahlung eingegangen ist.

4.8.1 Erzeugung eines Wallets / einer Adresse

Ein HD-Wallet lässt sich bei Bitcoin ganz einfach erstellen. Entweder wird die Wallet Applikation genutzt oder die Wallet RPC / CLI Funktion «createwallet». Wichtig ist, dass das Wallet mit einem Passphrase verschlüsselt wird.

Folgende Zeile erstellt ein Wallet mit dem Namen «chaingate-main-wallet» und dem Passphrase «main_wallet_passphrase» welches beim Starten immer geladen wird.

```
createwallet "chaingate-main-wallet" false false "main_wallet_passphrase" false false true
```

Sobald das Wallet erstellt wurde, kann mit der «getnewaddress» RPC-Funktion eine neue Adresse erstellt werden.

4.8.2 Transaktionskosten

Die Transaktionskosten sind wichtig, damit der Mindestzahlungsbetrag berechnet werden kann. Folgende Bedingungen müssen erfüllt sein, damit eine Zahlung erstellt werden kann.

- Die CHainGate Gebühr muss höher sein als das Minimum Rückgeld.
- Der Zahlungsbetrag muss höher sein als zwei Mal die Transaktionskosten.

Damit das Rückgeld so tief wie möglich gehalten werden kann, müssen Dust Transaktionen erlaubt werden. Mit folgenden zwei Optionen können sie erlaubt werden.

```
Discardfee = 0
dustRelayFee= 0
```

Nach diesen Anpassungen muss die Gebühr von CHainGate höher sein als die «cost of change» damit sie an die Rückgeldadresse gesendet wird.

Die Kosten pro BTC/kB bei einer Transaktion sind abhängig von der Blockchain Auslastung. Um eine gute Schätzung zu erhalten, bietet Bitcoin eine «estimatesmartfee» Funktion an. Dabei können folgende zwei Parameter gesetzt werden.

Conf_target: In wie vielen Blöcken soll die Transaktion bestätigt werden.
estimate_mode: Eine Auswahl zwischen «UNSET», «ECONOMICAL» UND «CONSERVATIVE».

Durch die estimatesmartfee und der Berechnungsgrundlage gemäss Theorie kann der Mindestzahlbetrag berechnet werden.

4.8.3 Initiale Zahlung

Um benachrichtigt zu werden, wenn ein Käufer eine Zahlung tätig, wird ein Benachrichtigungsfeature vom Bitcoin Node genutzt. Dabei können in der Konfigurationsdatei zwei Parameter konfiguriert werden. Bei einer Benachrichtigung wird ein HTTP GET auf einen entsprechenden Endpunkt vom Bitcoin-Service gemacht, welcher die Nachricht verarbeitet.

```
walletnotify=curl localhost:9001/api/notification/walletnotify?tx_id=%s&mode=test
blocknotify=curl localhost:9001/api/notification/blocknotify?block_hash=%s&mode=test
```

Walletnotify: Diese Benachrichtigung wird ausgeführt, wenn eine neue Transaktion erstellt wird. Dabei wird sie für empfangene sowie für gesendete Transaktionen ausgeführt. Weiter wird sie pro Transaktion immer zwei Mal ausgeführt. Einmal mit 0 Bestätigungen und das zweite Mal mit einer Bestätigung. Diese Meldung wird genutzt, um benachrichtig zu werden, wenn eine neue Zahlung ausgeführt wurde. [65]

Blocknotify: Diese Benachrichtigung wird ausgeführt, wenn ein neuer Block erstellt wird. Auf dem Mainnet ist das etwa alle 10 Minuten. Diese Meldung wird genutzt, um nach jedem Block zu überprüfen, ob die offenen Zahlungen bereits 6 Bestätigungen haben oder nicht. [65]

4.8.4 Abfragen des Guthabens

Um herauszufinden, wie viel Geld auf einer Adresse vorhanden ist, kann «listunspent» genutzt werden. Dabei werden drei Paramenter mitgegeben.

minconf: Wie viele Bestätigungen das Guthaben mindestens haben muss.

maxconf: Wie viele Bestätigungen das Guthaben maximal haben kann.

addresses: Die Adressen, für welche die Abfrage erstellt werden soll.

4.8.5 Weiterleiten des Guthabens

Für das Senden von Bitcoin wird eine Transaktion manuell über «rawtransaction» erstellt. Dabei besteht der gesamte Ablauf aus drei Schritten.

1. Erstellen der Transaktion über «rawtransaction». Dabei muss als Input die UTXO angegeben werden, welche der Käufer gesendet hat. Und als Output muss die Adresse vom Händler inklusive Betrag angegeben werden.
2. Als Nächstes wird die Transaktion mit weiteren Informationen über «fundrawtransaction» ergänzt. Dabei wird die Rückadresse für das Rückgeld (CHainGate-Gewinn) angegeben, die Transaktionskosten und wer diese Transaktionskosten bezahlen soll. Im Falle von CHainGate übernimmt diese der Händler.
3. Im letzten Schritt wird die Transaktion signiert und an das Netzwerk zur Bestätigung gesendet. Durch die Verschlüsselung des Wallets, muss das Wallet vor dem Senden entsperrt und später wieder gesperrt werden.

4.9 Payment Page

Neben der normalen Webapp wurde noch eine Bezahlseite implementiert. Auf diese werden Käufer, nachdem sie das Produkt beim Händler bestellt haben, weitergeleitet. Die Seite wurde mit WebSockets implementiert, um den Käufer schnellstmöglich mitzuteilen, wenn sich etwas am Status der Seite verändert hat. Dabei wurde auch darauf geachtet, dass alle mit der gleichen Bestellung gleichzeitig benachrichtig werden, um Missverständnisse zu vermeiden.

Die Payment Page kann in die folgenden Schritte unterteilt werden:

1. Auswahl der Kryptowährung
2. Bezahlen
3. Erhalten der Bezahlung
4. Confirmed
5. Expired

4.9.1 Auswahl der Kryptowährung

Um dem Händler von CHainGate so viel wie möglich abzunehmen, wurde die Wahl der Kryptowährung auf der Seite von CHainGate implementiert. Zurzeit kann zwischen Ethereum und Bitcoin entschieden werden. Auf der Abbildung 18 ist die Auswahl einer Kryptowährung ersichtlich. Die Abbildung 19 zeigt eine ausgewählte Kryptowährung.

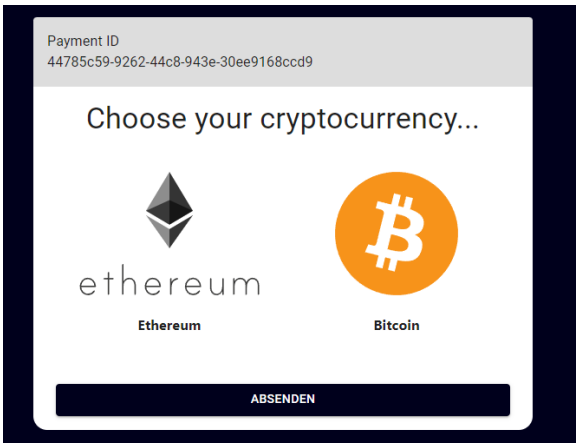

Abbildung 18: Payment Page - Currency selection

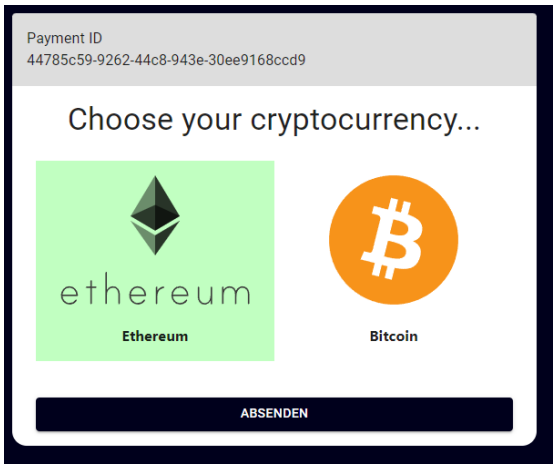

Abbildung 19: Payment Page - Currency selected

4.9.2 Bezahlen

Nachdem ein Käufer sich für eine Kryptowährung entscheiden hat, beginnt ein Countdown. In dieser Zeit verändert sich der Preis der Kryptowährung nicht. Dem Käufer wird neben der verbleibenden Zeit, auch der zu bezahlende Betrag und die Zieladresse angezeigt. In der folgenden Abbildung 20 ist ersichtlich, wie die Zahlungsinformationen dem Käufer dargestellt werden.

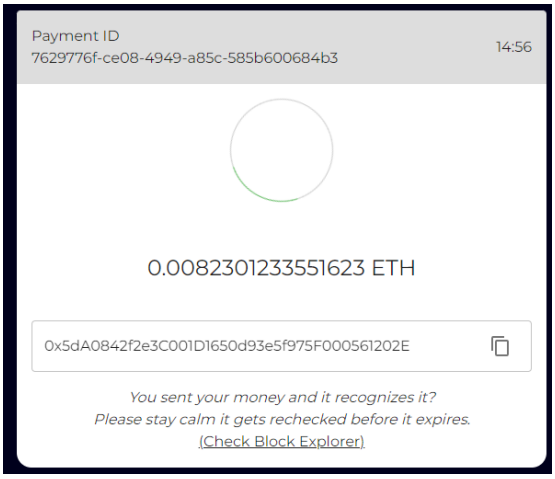

Abbildung 20: Payment Page - Waiting

4.9.3 Expired

Falls in den 15 Minuten keine vollständige Zahlung stattgefunden hat, wird die Adresse zurück in den Pool gelegt und die Zahlung als expired markiert. Der Käufer erhält für 3 Sekunden eine Anzeige, wie in der nachfolgenden Abbildung 21 ersichtlich ist und wird anschliessend zurück zum Händler geleitet.

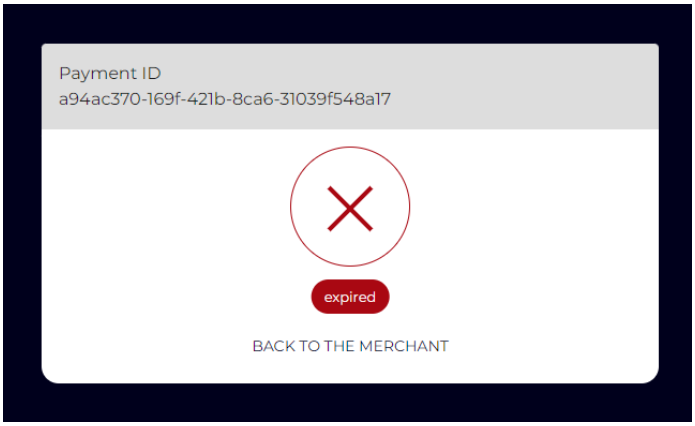

Abbildung 21: Payment Page - Expired

4.9.4 Erhalten einer Teilzahlung

Wenn in der Zeit ein Teil des Geldes überwiesen wurde, wird die Zahlung aktualisiert. Dem Käufer wird die Abbildung 22 angezeigt. Dabei erhält er zusätzlich die Information, wie viel noch bezahlt werden muss, um die Zahlung abzuschliessen.

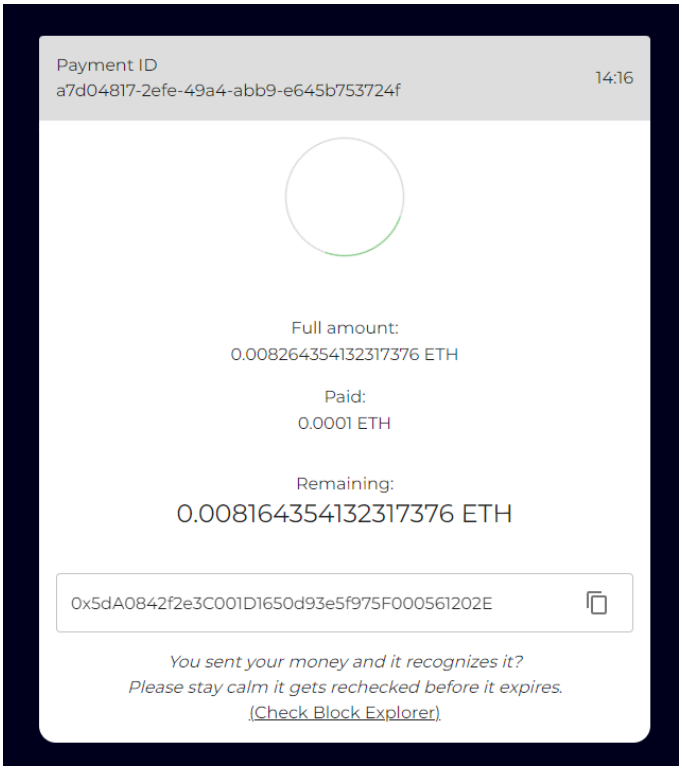

Abbildung 22: Payment Page - Partially Paid

4.9.5 Erhalten der Bezahlung

Falls in der Zeit genug Geld überwiesen wurde, wird die Zahlung bestätigt. Dem Käufer wird die Abbildung 23 für 3 Sekunden angezeigt und wird danach weitergeleitet.

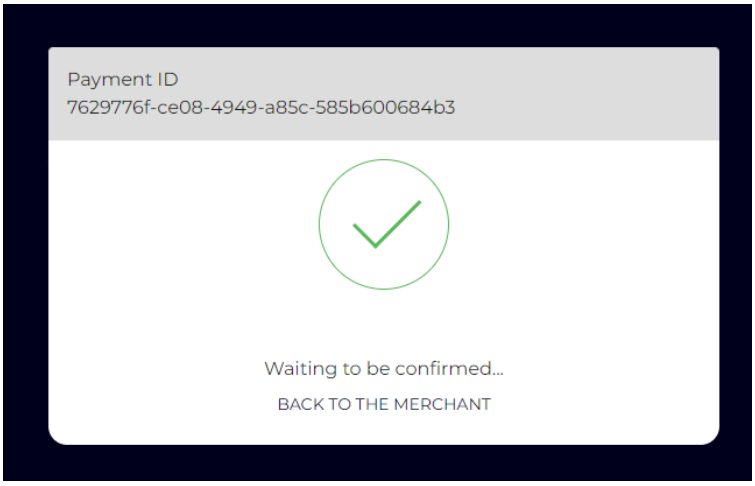

Abbildung 23: Payment Page - Paid

4.9.6 Confirmed

Wenn der Block bei Bitcoin mindestens 6 Mal und bei Ethereum 12 Mal bestätigt wurde, wird die Zahlung als Confirmed markiert und an den jeweiligen Händler weitergeleitet. Die folgende Abbildung 24 zeigt eine bestätigte Zahlung aussieht.

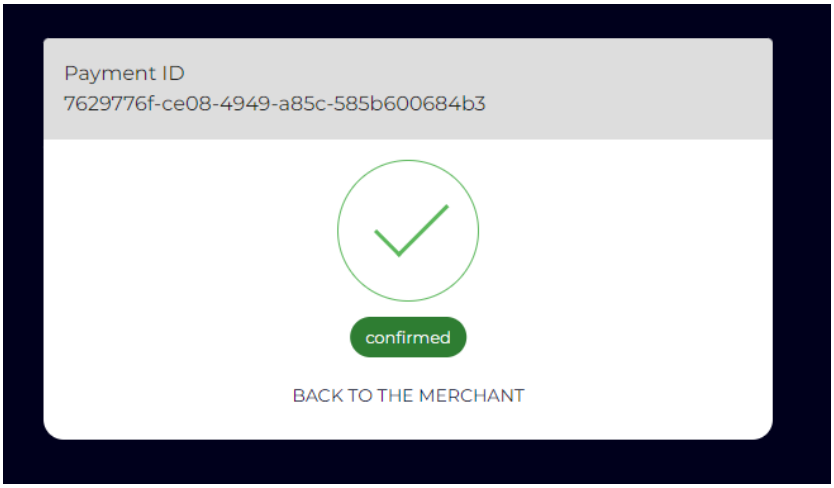

Abbildung 24: Payment Page - Confirmed

Diese Ansicht wird dem Käufer nur angezeigt, wenn er auf die Payment Page zurückkehrt, nachdem er weitergeleitet wurde.

4.10 Sicherheit

In diesem Abschnitt wird beschrieben, welchen Risikofaktoren CHainGate ausgesetzt ist. Dabei wird zuerst das Risiko erläutert und anschliessend eine Lösung beschrieben.

4.10.1 51%-Attacke

Sobald ein Miner oder eine Gruppe von Minern, die zusammenarbeiten 51% der Hashing-Power besitzen, wird von einer 51 % Attacke gesprochen. Durch diese Mehrheit hat der Angreifer die volle Kontrolle über die Blockchain und kann Transaktionen invalidieren, Blöcke nicht in die Blockchain aufnehmen (Zensur) oder eine Double-Spending-Attacke ausführen. Bei einer Double-Spending-Attacke wird das Geld zwei Mal ausgegeben. [74]

Folgendes Beispiel veranschaulicht eine solche Attacke:

«Ein Angreifer kauft ein Notebook bei einem Online-Shop, welcher den Service von CHainGate nutzt. Nachdem der Angreifer das Notebook erhalten hat, muss er bei der Blockchain eine längere Kette hinzufügen. Bei dieser neuen Kette wird das Geld nicht an CHainGate überwiesen, sondern an eine zweite Adresse, die auch dem Angreifer gehört. Weil immer die längste Kette gültig ist, wird die alte Transaktion rückgängig gemacht und wieder in den Pool zum Validieren hinzugefügt. Weil jedoch das Geld bereits ausgegeben wurde, wird die alte Transaktion nicht mehr validiert von den Minern. Somit hat der Online-Shop das Geld verloren und das Notebook ist bereits verschickt worden.»

Lösung

Eine Lösung für dieses Problem existiert leider nicht, da der Konsensalgorithmus darauf basiert, dass der, der die 51 % Rechenpower besitzt, die Wahrheit sagt. Es gibt aber eine Möglichkeit es zu erschweren, indem die Bestätigungszeit erhöht wird. Wenn ein Service beispielweise auf 100 Blockbestätigungen war-

tet, dann muss ein Angreifer 101 Blöcke selbst schürfen, um den Service zu überlisten. Dies wurde beispielweise von Coinbase unternommen, als Ethereum Classic unter einer 51 %-Attacke litt. Angreifer besitzen nur über einen bestimmten Zeitraum 51 % des Netzwerks, da es zum einen teuer ist, so viel Rechenkapazität zu betreiben und wenn eine Attacke bekannt wird, wird die Währung destabilisiert, da das Vertrauen sinkt und dass viele Besitzer dazu veranlasst, ihren Besitz zu verkaufen. [75–77]

4.10.2 Sicherstellung der Transaktion

Eine Blockchain ist ein verteiltes System, welches über den Konsensalgorithmus versucht, einen einheitlichen Stand zu erzeugen. Dabei kann es vorkommen, dass zwei Nodes gleichzeitig einen validen Beitrag in die Blockchain schreiben und dadurch eine Verzweigung entsteht. Diese Verzweigung löst sich schnell auf, da die Wahrscheinlichkeit, dass erneut zwei Blöcke auf den zwei verschiedenen Zweigen entstehen, sich mit jedem Block verkleinert. In der folgenden Abbildung 25 wird eine Verzweigung der Blockchain gezeigt, bei der ein Zweig verwaist ist.

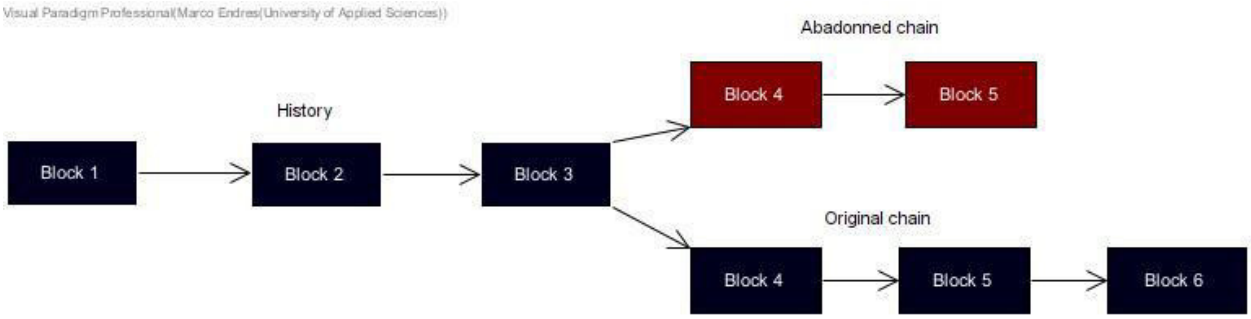

Abbildung 25: Blockchain Zweige

Eine Transaktion, welche in einem verwaisten Zweig bestätigt wurde, wird automatisch zurück in den Pool gelegt, damit sie nochmals auf dem offiziellen Zweig bestätigt wird.

Das eigentliche Problem, welches entstehen kann, ist das ein Händler basierend auf der verwaisten Transaktion bereits Güter zur Post gebracht hat und die Transaktion in der Zwischenzeit rückgängig gemacht wurde, weil es diese in der längsten Kette nicht mehr gibt. In dieser Zeit kann der Käufer das gleiche Geld nochmals ausgeben, um eine Double-Spending-Attacke auszuführen.

Lösung

Um dieses Problem zu lösen, wurde das X-Confirmation-Pattern benutzt. Mit dem Pattern wird, nachdem eine Transaktion entgegengenommen wurde, eine bestimmte Anzahl Blöcke abgewartet.

Das X-Confirmation-Pattern wird ebenfalls für die Weiterleitung benutzt. Dies wäre nicht nötig, da eine Transaktion, die in einem Block ist, welche nicht mehr gültig ist, automatisch wieder in den Pool zur Validierung aufgenommen wird. Das Schlimmste ist, dass der Händler das Geld zu einem späteren Zeitpunkt wieder erhält. Es wurde jedoch trotzdem implementiert, um dem Händler mit erhöhter Sicherheit mitteilen zu können, dass das Geld übertragen wurde und die Chance, dass die Transaktion verändert werden kann, sehr klein ist. [78]

4.10.3 Zu tiefe Beträge

Die folgende Abbildung 26 zeigt, wie es vorkommen kann, dass durch das Zahlen eines zu tiefen Betrages die Zahlung nicht weitergeleitet werden kann.

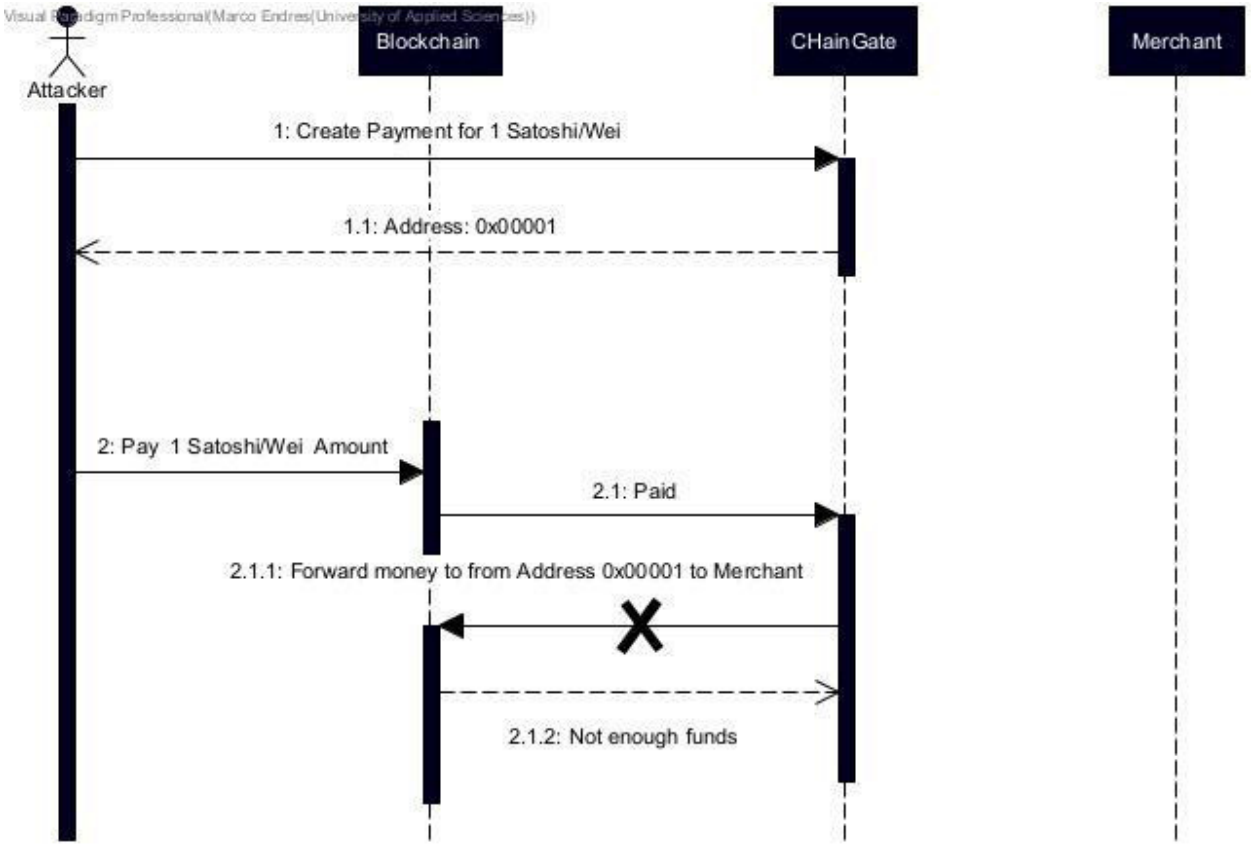

Abbildung 26: Problem - Zu tiefe Beträge

Wenn der Händler eine Zahlung erstellt, bei dem der Zahlungsbetrag geringer ist als die Transaktionskosten, gibt es ein Problem. Dadurch kann die Zahlung nicht weiterverarbeitet werden. Deshalb ist es wichtig, dass nur Zahlungen erstellt werden, welche sicher weitergeleitet werden können. Wichtig ist ebenfalls zu beachten, dass die Kosten für eine Transaktion, wie bereits erwähnt, fluktuieren und daher keine fixe Grenze gesetzt werden kann.

Es können theoretisch Transaktionen mit tieferen Transaktionskosten als den momentanen Marktwert erstellt werden. Diese werden jedoch erst in einen Block aufgenommen, wenn der Markt am gesetzten Marktwert ankommt, da aus wirtschaftlichen Gründen zuerst die lukrativeren Transaktionen ausgeführt werden.

Lösung

Der Händler soll nur eine Zahlung erstellen können, wenn der Zahlungsbetrag genug hoch ist. Bei CHainGate muss der Betrag höher sein als das doppelte der Transaktionskosten. Es macht wenig Sinn, wenn der Zahlungsbetrag genau gleich hoch wie die Transaktionskosten ist, weil der Händler somit kein Geld erhalten würde.

$$Zahlungsbetrag > Transaktionskosten * 2$$

4.10.4 Zu viele API-Aufrufe

Bei Ethereum gibt es durch den API-Provider «Infura» ein weiteres Problem, dass beachtet werden muss. Die API ist limitiert auf 100'000 Aufrufe am Tag. Es muss verhindert werden, dass eine Aktion eines Käufers mit einem Aufruf der API skaliert, da sonst ein Käufer alle Aufrufe aufbrauchen kann.

Problem 1

Bei Problem 1 erstellt ein Käufer viele Zahlungen, welche iterativ auf ihren Status über Infura geprüft werden.

Die folgende Abbildung 27 zeigt das erste von zwei Problemen, wie ein Käufer vermehrt API-Aufrufe auslösen kann.

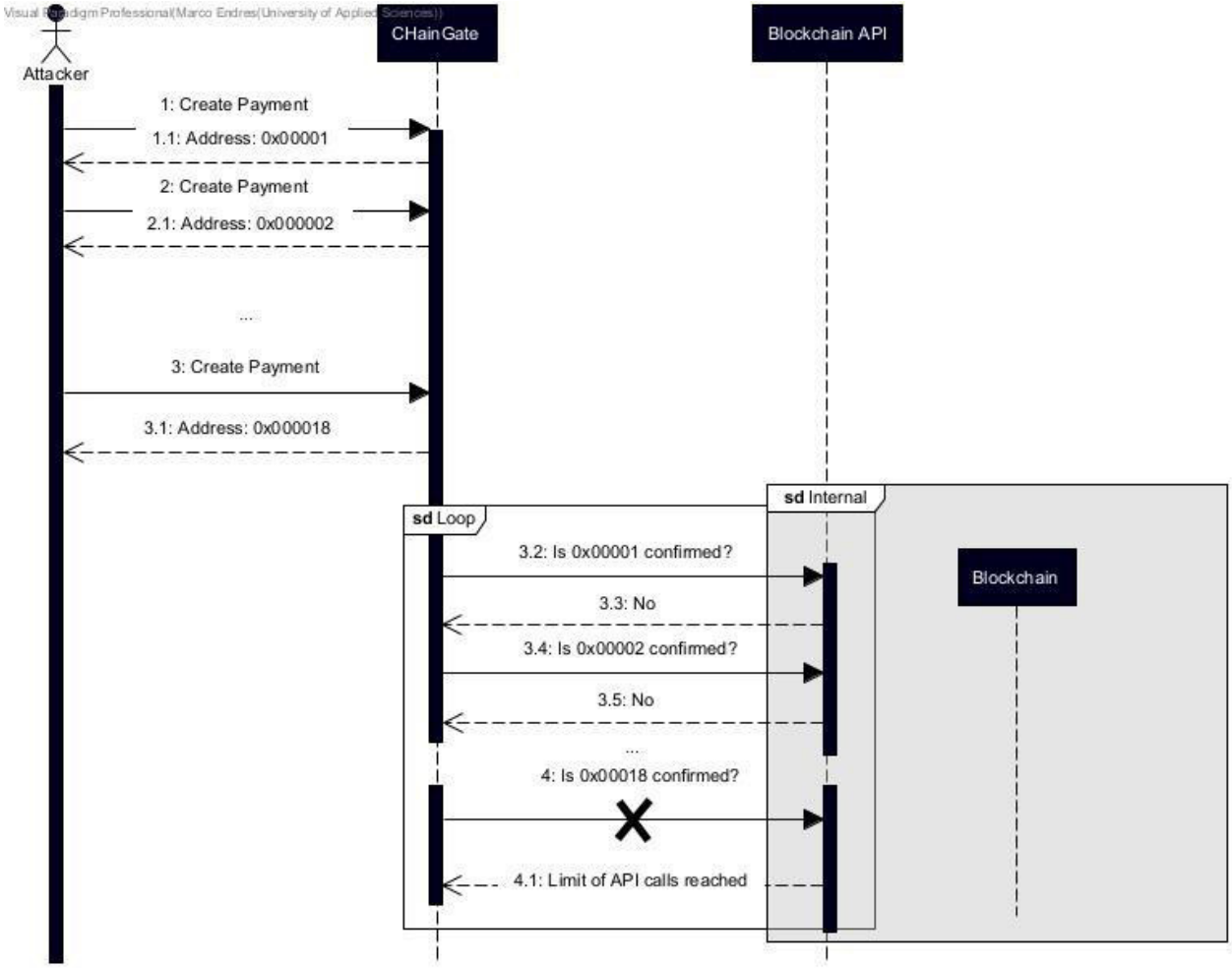

Abbildung 27: Problem - Zu viele API-Aufrufe 1

Problem 2

Bei Problem 2 erstellt ein Käufer viele kleine Transaktionen, die alle über die Infura-API überprüft werden müssen.

Die folgende Abbildung 28 zeigt das zweite von zwei Problemen, wie ein Käufer vermehrt API-Aufrufe auslösen kann.

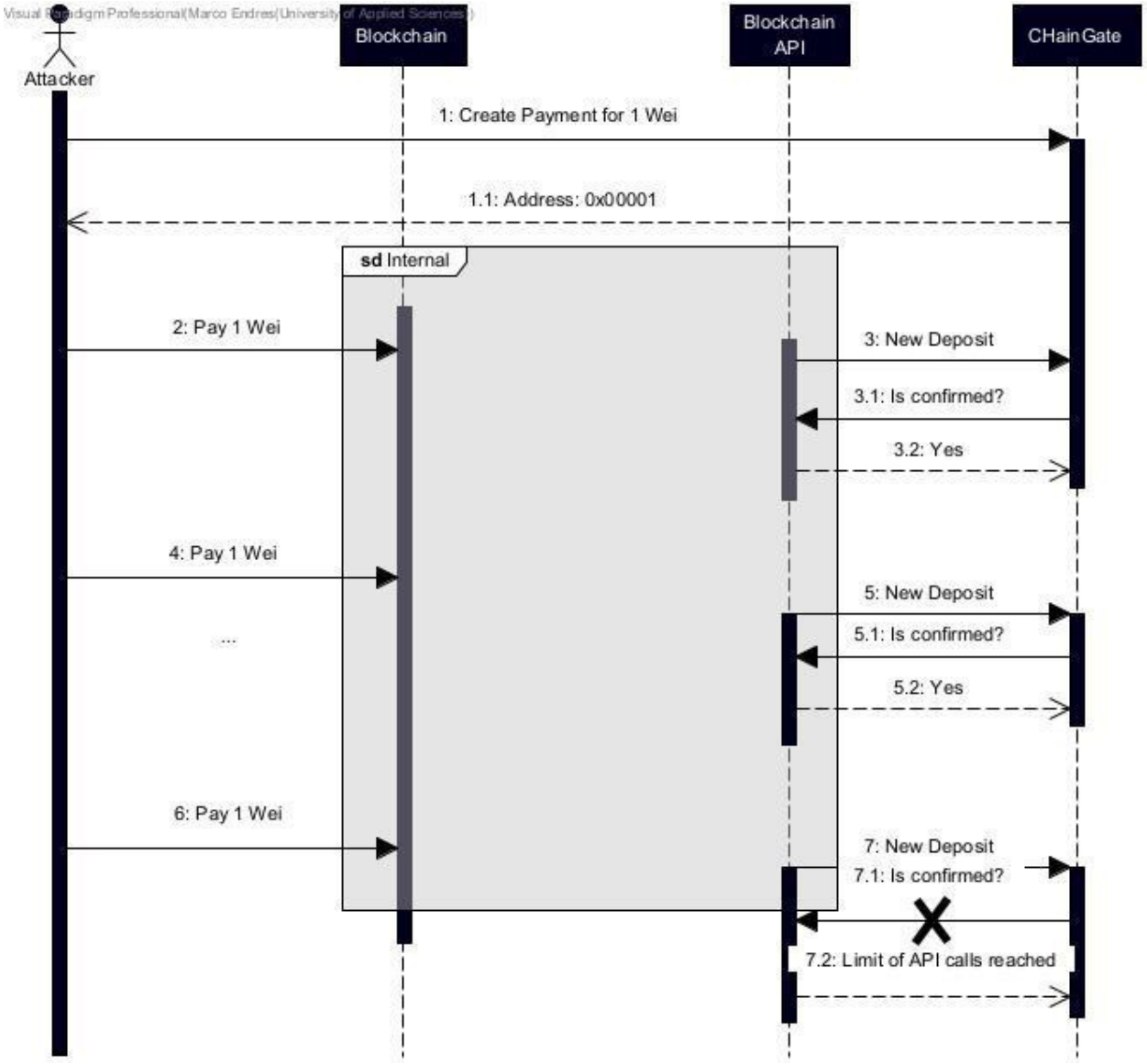

Abbildung 28: Problem - Zu viele API-Aufrufe 2

Ethereum Lösung

Um Transaktionen nach aussen zu vermeiden und so mehr an Bandbreite zu sparen, wurden die folgenden zwei Optimierungen in Ethereum vorgenommen:

1. Die Informationen der Blockchain über Events erhalten.
2. Überprüfen des letzten Blockes, welcher nötig war, um die Zahlung erfolgreich zu machen.

Events der Blockchain

Anfänglich wurde der Ethereum-Service mit einem Pull-Prinzip implementiert. Durch einen Cronjob wurde alle 15 Sekunden alle offenen Adressen abgefragt. Dies hat zur Folge, dass maximal durchschnittlich 17 Transaktionen offen sein dürfen, damit das Tageslimit an API-Anfragen für Infura von 100'000 nicht überschritten wird.

$$\frac{100'000\ Maximum\ API\ Calls}{24\ Stunden * 60\ Minuten * \frac{60}{15}} = 17.36111\ \frac{API\ Calls}{Sekunde}$$

Durch ein eventbasiertes System wird immer eine Nachricht an den Ethereum-Service gesendet, wenn ein neuer Block geschürft wurde mit den enthaltenen Transaktionen. Dadurch entfallen alle *getBalance*-Aufrufe des Systems, wie die folgende Abbildung 29 beweist.

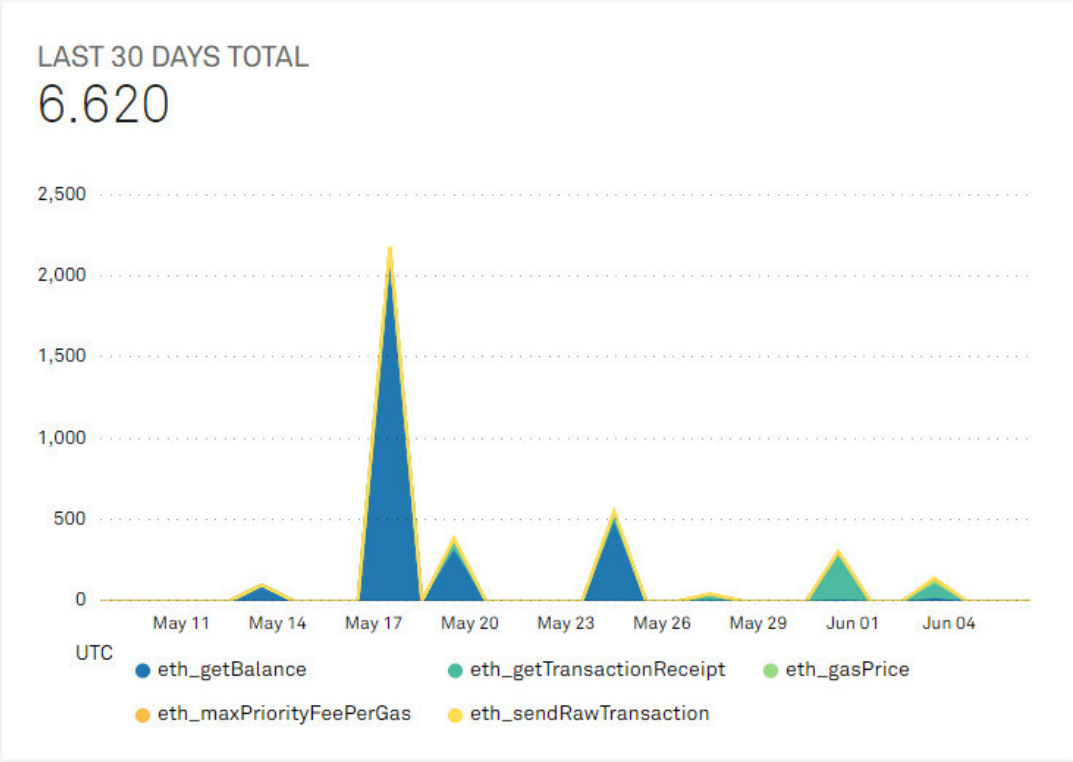

Abbildung 29: Umstellung Eventbasiert Ethereum (24. Mai)

Überprüfen des letzten Blockes

Um die Zahlung zu validieren, wurde, wie bereits erwähnt ein Confirming implementiert. Dieses Confirming könnte theoretisch auf jede Teilzahlung implementiert werden. Dies hätte zur Folge, dass ein Kunde dies zu böswilligen Zwecken missbrauchen und ganz kleine Minitransaktionen durchführen könnte. Dadurch müsste CHainGate viele kleine Transaktionen validieren. Daher wurde entschieden, nur die letzte nötige Transaktion zu validieren. Durch die Natur einer Blockchain sollte bei einer Invalidation eines älteren Blocks ebenfalls ein neuer Block invalidiert werden.

Ein solcher Angriff ist auf dem Mainnet eher unwahrscheinlich, da jede Transaktion den Angreifer auch Geld kosten würde. Ein Angriff auf dem Testnet hingegen wäre denkbar.

Bitcoin Lösung

Bei Bitcoin wurde diese Attacke nicht behandelt, da ein eigener Node aufgesetzt wurde und somit die Anfragen unlimitiert sind.

4.10.5 API-Key wird gestohlen

Wenn nur der API-Key gestohlen wurde, können dabei Zahlungen erstellt werden. Jedoch kann mit einer offenen Zahlung nichts gemacht werden. Besitzt der Angreifer hingegen noch Informationen wie die Payment ID und weiss, wie ein Händler die WebHooks von CHainGate verarbeitet, können Zahlungen gefälscht werden.

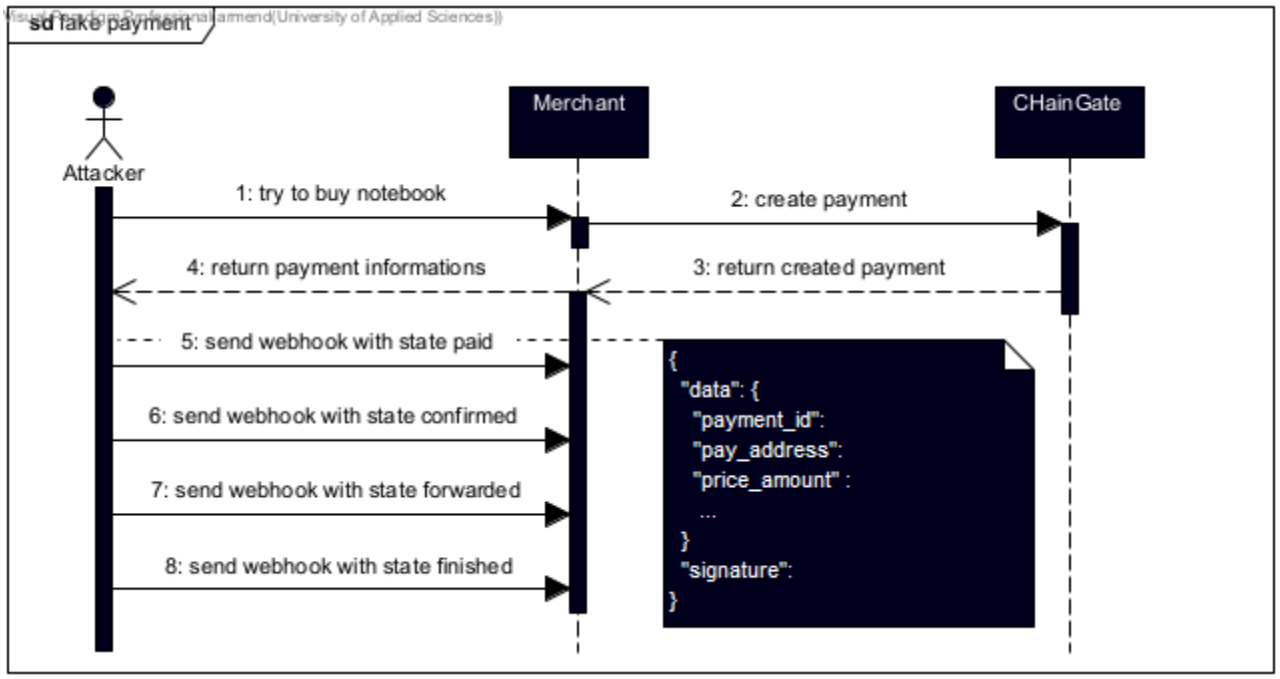

Abbildung 30: Zahlung fälschen

In der Abbildung 30 ist abgebildet, wie eine Attacke aussehen könnte.

Für eine Attacke muss der Angreifer beispielsweise zuerst versuchen ein Notebook zu kaufen. Nachdem die Zahlung erstellt wurde, wartet der Händler auf die WebHooks von CHainGate, um die Zahlung intern

abzuschliessen. Wenn der Angreifer den API-Key hat, besitzt er die Möglichkeit eine gültige Signatur zu erstellen. Eine weitere wichtige Information, die der Angreifer braucht, ist die payment_id, weil viele Händler diese brauchen werden, um die Zahlung zu identifizieren. Je nach Implementation beim Händler kann es sein, dass der Angreifer weitere Informationen wie pay_address oder price_amount benötigt, um einen gültigen WebHook zu simulieren.

Lösung

Ein Kunde kann jederzeit einen neuen API-Key generieren und somit den alten invalidieren lassen.

4.10.6 Datenbank wird geleaked

Wenn eine Datenbank geleaked wird, können folgende kritischen Informationen gestohlen werden.

- API-Keys
- Passwörter der Benutzer
- Private Keys der Wallets / Adressen

Backend Lösung

In der Backend Datenbank sind die API-Keys und Passwörter gespeichert. Die API-Keys sind mit AES verschlüsselt und die Passwörter mit Scrypt. Die API-Keys werden nur entschlüsselt, um sie dem Benutzer auf der Webseite anzuzeigen. Die Passwörter können nicht entschlüsselt werden, da bei einem Login jeweils das Passwort mit Scrypt verschlüsselt wird und die Hashes miteinander verglichen werden. Das Secret für die AES-Verschlüsslung ist in einer .env-Variable hinterlegt.

Ethereum Lösung

Bei Ethereum wurde dieses Problem mit einer Verschlüsselung des Private Keys gelöst. Der Private Key wird nur entschlüsselt, wenn eine Transaktion getätigt werden muss. Das Secret für das Entschlüsseln der Private Keys ist in einer lokalen .env-Variable abgelegt. Dadurch braucht es einen Leak des Secrets und der Datenbank, um auf die Adressen zugreifen zu können.

Bitcoin Lösung

Bei Bitcoin ist das Wallet verschlüsselt auf der Platte des Nodes gespeichert. Dadurch kann durch einen Leak der Datenbank das Bitcoin Wallet nicht kompromittiert werden.

4.10.7 Verlorene Zahlung

Die folgende Abbildung 31 zeigt, wie eine Transaktion in der Theorie verloren gehen könnte ohne entsprechende Gegenmassnahmen.

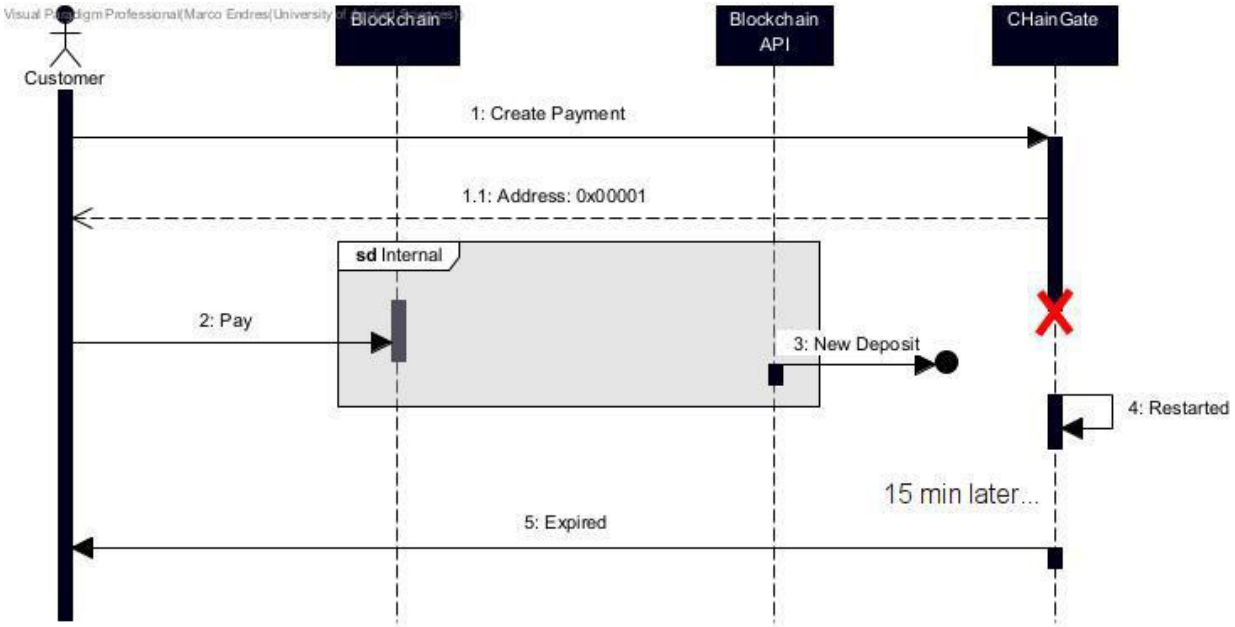

Abbildung 31: Problem - Verlorene Zahlung

Wenn ein Blockchain Service aussteigt, werden in dieser Zeit weiterhin Blöcke auf der Blockchain geschürft, welche die Services nicht mitbekommen.

Ethereum Lösung

Da in der Downtime des Services weiterhin Blöcke hinzukamen und ein Kunde in dieser Zeit bezahlt haben könnte, müssen alle offenen Zahlungen geprüft werden, bevor sie auf expired gesetzt werden können. Da es bei Ethereum aufwändig sein kann, alle Blöcke im Nachhinein noch einmal zu durchsuchen, wird im Recovery stattdessen auf das Guthaben der Adresse geachtet. Dies ist um einiges effizienter, hat jedoch den Nachteil, dass der Service den Block, in welchem die Transaktion stattgefunden hat, nicht kennt. Das hat die Folge, dass nicht gesagt werden kann, ob der Block bereits 12 Bestätigungen besitzt.

Um diese Schwachstelle auszunützen, müssen folgende Schritte in dieser Reihenfolge ausgeführt werden:

1. Ethereum Service muss online sein
2. Einzahlung erstellen
3. Ethereum-Service muss offline sein
4. Transaktion erstellen
5. Ethereum-Service starten inklusive Recovery
6. Double-Spending-Attacke ausführen. Je weniger Bestätigungen die Transaktion beim Recovery besitzt, desto einfacher ist es eine Double-Spending-Attacke auszuführen.

Ebenfalls wird jede Transaktion wegen «internen Transaktionen» noch einmal am Ende des Auslaufens der Zahlung auf die Balance überprüft.

Bitcoin Lösung

Wenn der Bitcoin-Service eine Zahlung aus einem Grund nicht mitbekommt, bleibt die Zahlung offen und im Status waiting, bis dieser auf expired gesetzt wird. Jedoch wird vor dem Setzen auf expired immer geprüft ob allenfalls genügend Guthaben auf der Adresse vorhanden ist. Sollte genügen Guthaben vorhanden sein, kann davon ausgegangen werden, dass die Zahlungsnotifikation verloren gegangen ist und die Zahlung wird nicht auf expired gesetzt, sondern auf paid und der Zahlungsprozess kann ganz normal weitergehen.

Aus diesem Grund benötigt der Bitcoin-Service kein aktives Recovery beim Starten des Services. Der Nachteil ist, dass erst nach 15 Minuten eine verlorene Zahlung erkannt wird.

4.10.8 Doppeltes Bezahlen

Die folgende Abbildung 32 zeigt, wie Geld doppelt weitergeleitet werden könnte, wenn keine Sicherheitsmassnahmen dagegen unternommen werden.

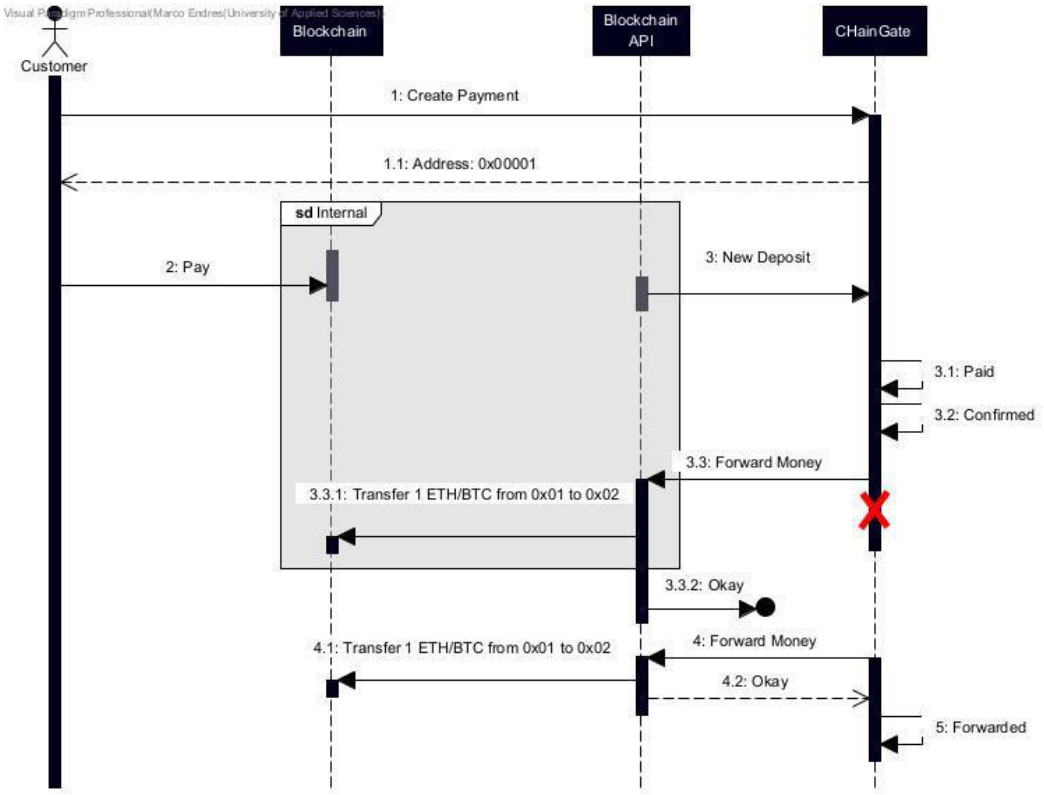

Abbildung 32: Problem - Doppeltes Bezahlen

Einer der grössten Schwachstellen im Zahlungsablauf ist, dass die Transaktion für das Weiterleiten des Guthabens auf der Blockchain stattgefunden hat, jedoch der Service vor dem Speichern auf die Datenbank abstürzt. Als Folge könnte es vorkommen, dass der CHainGate-Service beim Starten das Geld erneut sendet.

Ethereum Lösung

Um dies bei Ethereum zu verhindern, wurde in der Datenbank zusätzlich der Nonce der Adresse abgespeichert. Dieser Nonce wird bei jedem Weiterleiten um eins erhöht. Falls der Service abstürzt, nachdem die ETHs gesendet wurden, wird entweder die Transaktion mit der Neuen ersetzt, wenn die Alte noch nicht geschürft wurde oder die neue Transaktion wird abgelehnt, da bereits eine Transaktion mit dem Nonce existiert.

Bitcoin Lösung

Beim Bitcoin-Service wird, bevor eine Transaktion gesendet wird, geprüft, ob genügend Geld auf der Adresse vorhanden ist. Sollte es kein Geld auf der Adresse haben, wurde die Transaktion bereits ausgelöst, konnte jedoch nicht auf der Datenbank abgespeichert werden. In diesem Fall werden alle gesendeten Transaktionen auf der Blockchain durchsucht, bis die verlorene Transaktion gefunden wurde. Nachdem sie gefunden wurde, wird der Status der Zahlung auf forwarded aktualisiert.

4.10.9 Zugang zur Adresse verloren

Wenn ein Händler den Private Key seiner Adresse verliert oder er öffentlich gemacht wird, kann die Adresse nicht mehr zuverlässig als Ziel für die Transaktionen genutzt werden.

Lösung

Der Händler hat die Möglichkeit, die Zieladresse pro Kryptowährung festzulegen und zu ändern. Diese wird zurzeit erst für neue Transaktionen übernommen. Zukünftig wäre es denkbar, bereits gestartete Transaktionen zu aktualisieren.

4.10.10 Service Ausfall

Bei CHainGate können alle möglichen Services ausfallen. Es könnte die Datenbank, der Backend-Service, Infura oder der Bitcoin Node ausfallen. Dabei muss sichergestellt werden, dass der Zustand einer Zahlung immer korrekt ist.

Lösung

Die Blockchain Services funktionieren wie eine Zustandsmaschine. Wenn der Übergang von einem Zustand zum nächsten nicht vollständig abgeschlossen werden kann, bleibt die Zahlung beim alten Zustand. Somit hat die Zahlung bei der nächsten Iteration erneut die Möglichkeit, ihren Zustand zu wechseln. Dabei muss sichergestellt sein, dass ein Übergang mehrfach versucht werden kann, ohne dass die Daten korrupt werden.

5. Ergebnisse

In diesem Kapitel werden die Ergebnisse präsentiert. Dabei geht es um das Thema Testing sowie Ausblick. Im Anhang unter dem Kapitel «F. Umgesetzte Anforderungen» werden alle Ergebnisse beschrieben inklusive Screenshots der Applikation.

5.1 Testing

Das Testing wurde bei CHainGate auf der Stufe von Unit-, Integrations- und Systemtests durchgeführt. Durch die verteilte Natur von Blockchains ist ein gutes Testing aufwändig. In den folgenden Abschnitten werden die einzelnen Varianten erläutert.

5.1.1 Sicherstellung von Konsistenz

Neben dem Testing selbst wurde auf die Reduktion von Boilerplate-Code und Entwicklerfreundlichkeit gesetzt, da diverse Fehler durch Repetition und Copy-Paste entstehen. Folgende Tools unterstützten dieses Vorgehen:

- OpenAPI [79, 80]
- ORMapper [54]

OpenAPI

Durch die definierten OpenAPI-Schnittstellen wird die Konsistenz von Anfrage und Antwort auf dem Server und Client gewahrt. Dadurch sind zukünftige Änderungen einfacher durchzuziehen und weitere Personen können einfacher die Schnittstellen erweitern, ohne genau zu wissen, wo diese Schnittstellen überall angesprochen werden. Dies trifft auf alle Schnittstellen zu, ausser auf die gegenüber dem Kunden, da diese nicht unter der Kontrolle von CHainGate ist.

OR Mapper

Durch GORM konnten die SQL-Abfragen gegen die PostgreSQL-Datenbank abstrahiert werden. Zusätzlich werden die Codeobjekte automatisch auf Datenbank Tabellen gemappt. Ebenfalls wird verhindert, dass Fehler bei SQL abfragen oder gefährliche SQL-Injections entstehen.

5.1.2 Unit Tests

Für die Unit Tests wurden zwei Tools eingesetzt, welche das Mocken der Datenbank und des HTTP-Anfragen erlauben.

- Go-sqlmock [81]
- HTTP-Interceptor Gock [82]

Go-sqlmock

Die go-sqlmock Library wird verwendet, damit Methoden getestet werden können, welche eine Datenbankanbindung benötig. Durch die Library ist es möglich, diese Methoden ohne Datenbankanbindung zu testen. Die grösste Herausforderung war es, die abstrahierten SQL-Queries von GORM zu verstehen, um diese zu mocken.

HTTP-Interceptor Gock

Durch die Microservicearchitektur werden diverse HTTP-Anfragen an Services im System gesendet. Um Methoden mit ausgehenden Anfragen ebenfalls zu testen, wurde die Gock-Library eingesetzt, welche es erlaubt, HTTP-Anfragen abzufangen und eine beliebige Antwort zurückzusenden.

5.1.3 Integration Tests

Für die Integrationstests wurde das Tool dockertest eingesetzt, um Umsysteme wie Datenbank und Blockchain als Docker-Container hochzufahren und nach dem Test wieder herunterzufahren. Diese Methode wurde vor allem beim Bitcoin-Service eingesetzt, weil es dort keine Bitcoin In-Memory Blockchain gab.

5.1.4 System Tests

Über das ganze System wurde ein manuelles Testing vollzogen. Die Systemtest Spezifikation ist im Anhang zu finden. Dabei wurden alle Fälle getestet. Jedoch wurde für das Mainnet eine lokale Blockchain verwendet, um keine echten Kryptowährungen auszugeben. Es wurde jeweils eine Zahlung in Bitcoin und Ethereum auf dem Mainnet erstellt, um zu überprüfen, ob diese auch funktioniert.

5.1.5 Ethereum-Service

Im Ethereum-Service wurde eine Testabdeckung von 50 % erreicht. Die Tests wurden mit Unit- und Integrationstests umgesetzt. Für die Integrationstests wird eine In-Memory-Ethereum-Blockchain eingesetzt.

In-Memory-Ethereum-Blockchain

Beim Testen mit einer Blockchain ist eine Herausforderung jeweils das Abstrahieren der Blockchain im Test. Beim Ethereum-Service war es dank einer In-Memory-Ethereum-Blockchain möglich, eine Blockchain inklusive Miner aufzusetzen. Dadurch konnten die nötigen Tests lokal und in einer geschützten Umgebung ausgeführt werden. Zusätzlich konnte die Berechnung der Transaktionskosten geprüft werden, weil der Preis nicht fluktuiert.

5.1.6 Bitcoin-Service

Für den Bitcoin-Service wurde ein Integrationstest geschrieben. Dabei wurde dockertest genutzt, um eine Bitcoin Regtest Instanz und die Datenbank zu starten. Für den Test wurde jeweils der Erfolgspfad getestet. Der Test geht von der Erstellung einer Zahlung bis zur Überprüfung, ob die Zahlung an den Händler korrekt weitergeleitet wurde.

5.2 Ausblick

In diesem Abschnitt geht es um eine mögliche Weiterentwicklung von CHainGate. Die Möglichkeiten wurden in drei verschiedene Kategorien unterteilt, Vertrauen, Funktionalität und Kosten.

5.2.1 Vertrauen

Folgend werden verschiedene Möglichkeiten aufgezählt, wie das Vertrauen in CHainGate erhöht werden könnte.

5.2.1.1 Adressen aufteilen pro Kunden

Wenn die Adressen pro Kunde aufgeteilt werden, können die Private Keys mit dem Kunden geteilt werden, um mehr Vertrauen zu gewinnen. Dabei müssten die Private Keys mit der entsprechenden Händlernummer in der Datenbank verbunden werden. Bei einer neuen Zahlung müsste geprüft werden, ob ein Private Key für diesen Händler frei ist. Im Dashboard könnten die aktuellen Private Keys für einen Kunden angezeigt werden. Mit diesen kann er bei einer beliebigen Applikation die Adresse importieren und Zahlungen ausführen.

Dadurch müsste jedoch das Kostenmodell geändert werden, weil der Händler Zugriff auf die Provision hat. Eine mögliche Anpassung wäre es, eine fixe Gebühr zu verrechnen oder eine Gebühr pro Adresse.

5.2.1.2 Mehr Informationen auf der Payment Page für Ethereum Zahlungen

Bei Bitcoin ist die Blockzeit 10 Minuten. Aus diesem Grund wird eine Zahlung als bezahlt markiert, sobald sie im Pool ist, damit der Käufer nicht 10 Minuten warten muss. Bei Ethereum hingegen ist die Blockzeit nur 12-14 Sekunden und deshalb wird hier eine Zahlung erst nach der ersten Blockbestätigung als bezahlt markiert. In Zukunft könnte dem Käufer bei Ethereum angezeigt werden, dass seine Zahlung im Pool ist und die Zahlung bald bestätigt ist.

5.2.2 Funktionalität

Folgend werden verschiedene Möglichkeiten aufgezählt, wie die Funktionalität von CHainGate erhöht werden könnte.

5.2.2.1 Ethereum 2.0

Ethereum 2.0 ist der Umstieg von Proof of Work zu Proof of Stake. Diese Änderung ist bereits seit Langem geplant, jedoch hat sich der Termin schon mehrmals verschoben. Einen fixen Termin gibt es noch nicht, jedoch ist ein Termin im Juni 2022 angepeilt. Damit würde sich der Energiekonsum stark reduzieren, da es keinen Energie-getriebenen Schürfkampf mehr geben würde. [83]

5.2.2.2 Library für Rendering der Zahlung

Die entwickelte Proof of Conept-Library für das Rendern auf der Seite des Kunden könnte implementiert werden, um dem Händler mehr Flexibilität zu geben.

5.2.2.3 Weitere Kryptowährungen

Neben den bereits integrierten Kryptowährungen wäre es von Vorteil, weitere Kryptowährungen einzubinden, um mehr Kunden zu gewinnen.

5.2.2.4 Auszahlungen in Stablecoins oder Fiatgeld

Die Volatilität der Kryptowährungen kann für gewisse Kunden von Nachteil sein. Deshalb könnten in Zukunft Auszahlungen in einem Stablecoin oder in Fiatgeld angeboten werden.

5.2.2.5 QR-Code

Viele Wallets und Seiten mit Kryptowährung bieten heutzutage neben der Adresse gleich einen QR-Code an, um die Adresse zu lesen. Dies wäre für den Kunden einfacher als die Adresse zu kopieren.

5.2.2.6 Tests

Da die Applikation mit Geld arbeitet, ist es wichtig, dass sie zu jeder Zeit richtig funktioniert. Daher wäre es wichtig, eine höhere Unit-Testabdeckung zu erreichen.

Ebenfalls wäre es wichtig, über alle Systeme hinweg zu testen. Daher wären End-to-End Tests eine gute Ergänzung.

5.2.3 Kosten

Folgend werden verschiedene Möglichkeiten aufgezählt, wie die Kosten bei CHainGate reduziert werden könnten.

5.2.3.1 Limitieren der offenen Adressen

Um die Kosten zu optimieren, sollten die alten Adressen im Ethereum-Service wieder geschlossen werden, um so den Gewinn wieder zu maximieren. Wenn es weniger Adressen gibt, wird das Geld weniger verteilt, somit kann der benötigte Faktor für die Realisierung des Gewinns schneller erreicht werden.

5.2.3.2 Auszahlungen in einer Kryptowährung mit niedrigen Transaktionskosten

Nicht jede Kryptowährung besitzt so hohe Transaktionskosten wie Ethereum. Deshalb könnte dem Händler, eine Auszahlung in einer Kryptowährung mit niedrigen Transaktionskosten angeboten werden.

6. Schlussfolgerungen

Das Entwickeln eines Payment Providers für Kryptowährungen mit den unterstützen Währungen Ethereum und Bitcoin konnte erfolgreich abgeschlossen werden. Dabei wurde ein MVP erstellt, welches die nötigen Features implementiert, damit es genutzt werden kann. Für diese Arbeit wurde auf ein Deployment verzichtet, um sich vollständig auf die Entwicklung fokussieren zu können. Nach einem Deployment ist der Service durch dritte nutzbar. Somit kann ein erstes Feedback gesammelt werden.

Um die Wartbarkeit für die Zukunft zu erhöhen, sollte mehr Zeit in das automatisierte Testing investiert werden. Da es sich hierbei um eine kritische Applikation mit Geldtransfer handelt, sollte sichergestellt sein, dass es bei einer Transaktion zu keinem Fehler kommt.

Während der Arbeit sind weitere Ideen entstanden, welche das Produkt erweitern oder verbessern könnten. Alle Ideen sind im Ausblick genauer aufgelistet. Jedoch lohnt es sich vor der Weiterentwicklung ein erstes Kundenfeedback einzuholen, um die Wünsche und Anregungen der Kunden besser zu verstehen. Ebenfalls sollte der Markt noch genauer evaluiert werden.

7. Glossar

BTC ist die Währung bei Bitcoin.

Coins sind die einzelnen Geldstücke einer Kryptowährung.

Dust wird ein sehr kleines Guthaben genannt, welches nicht ausgegeben werden kann, weil die Transaktionskosten höher als das Guthaben sind.

ETH ist die Währung bei Ethereum.

Faucet ist ein Geldgeber bei Blockchains. Dieser wird verwendet, um auf einem Testnet Kryptowährungen zu erhalten.

Fees sind die Gebühren, die für eine Operation auf der Blockchain gezahlt werden müssen.

Gas ist die Menge an Leistung, die benötigt wird, um eine Transaktion auf der Blockchain auszuführen. Gas wird in Ethereum gezahlt.

Genesis-Block ist der erste Block auf einer Blockchain.

go-ethereum ist eine Go-Library, welche es erleichtert, mit der Ethereum-Blockchain zu kommunizieren.

Hash ist eine Funktion, welche aus einem Input ein Output generiert. Dabei kann aus dem Output der Input nicht wiederhergestellt werden.

HD-Wallet ist ein Wallet, welches mehrere Kinder Adressen erstellen kann.

hmac auch Hash-based Message Authentication Code, benutzt einen kryptografischen Schlüssel mit einer Hash-Funktion.

Infura ist eine Ethereum API.

Input ist bei Bitcoin das Guthaben, welches ausgegeben wird.

Mainnet ist das produktive Netz einer Blockchain.

Merkle Tree ist eine Datenstruktur, welche bei Blockchains verwendet wird.

Merkle Patriacia Tree ist ein Merkle Tree und die verwendete Datenstruktur bei Ethereum.

MetaMask ist eine Browser Extension, damit ein Benutzer auf einer Seite mit Ethereum bezahlen kann.

Miner werden Teilnehmer in der Blockchain genannt, welche versuchen das Rätsel zu lösen, um den Gewinn zu erhalten.

Mixer ist ein Dienst, welcher Krypto Geldmengen mit anderen vermischt, um die Anonymität zu steigern.

Nonce ist eine Zahl, die nur einmal genutzt wird.

OpenAPI ist ein Standard, um Schnittstellen zu dokumentieren.

Output ist die Zieladresse bei einer Bitcoin Transaktion.

Privatnet ist eine lokale Blockchain Instanz.

RPC (Remote Procedure Call) wird verwendet, um Funktionen auf der Blockchain auszuführen.

Schürfen ist das Bestätigen von Blöcken. Wird von Minern gemacht.

Testnet ist das Testnetzwerk einer Blockchain.

UTXO (Unspent Transaction Output) wird das Guthaben bei Bitcoin genannt, welches ausgegeben werden kann.

8. Abbildungsverzeichnis

9. Tabellenverzeichnis

10. Literaturverzeichnis

[1] ibi research an der Universität Regensburg GmbH, "Erfolgsfaktor Payment – Der Einfluss der Zahlungsverfahren auf Ihren Umsatz," [Online]. Available: https://www.six-payment-services.com/dam/download/newsletter/Studie-Erfolgsfaktor-Payment.pdf

[2] S. N. BNS, "Zahlungsmittelumfrage 2020," [Online]. Available: https://www.snb.ch/de/mmr/reference/paytrans_survey_report_2020/source/paytrans_survey_report_2020.de.pdf

[3] B. Manz, "Corona: Bargeld nicht mehr wichtigstes Zahlungsmittel," *moneyland.ch*, 23 Feb., 2021. https://www.moneyland.ch/de/zahlungsmittel-studie-schweiz-corona-2021 (accessed: May 2 2022).

[4] A. Hämmerli, "digitec und Galaxus akzeptieren neu Kryptowährungen," *digitec*, 19 Mar., 2019. https://www.digitec.ch/de/page/digitec-und-galaxus-akzeptieren-neu-kryptowaehrungen-11214 (accessed: May 3 2022).

[5] tagesschau, "Zahlung per Kryptos: Das verspricht sich Panama vom Bitcoin," *tagesschau.de*, 29 Apr., 2022. https://www.tagesschau.de/wirtschaft/weltwirtschaft/bitcoin-kryptowaehrungen-regulierung-el-salvador-panama-zahlungsmittel-101.html (accessed: May 2 2022).

[6] J. Steinschaden, "Lugano: Schweizer Stadt führt Bitcoin und Tether als Zahlungsmittel ein," *Trending Topics*, 03 Mar., 2022. https://www.trendingtopics.eu/lugano-schweizer-stadt-fuhrt-bitcoin-und-tether-als-zahlungsmittel-ein/ (accessed: May 2 2022).

[7] R. Carter, "Stripe vs PayPal: Welches Payment Gateway ist das richtige für Sie?," *Ecommerce Platforms*, 12 Sep., 2019. https://ecommerce-platforms.com/de/payment-processing/stripe-vs-paypal (accessed: May 2 2022).

[8] BitInfoCharts, *Ethereum Transaktionsgebühr grafiken.* [Online]. Available: https://bitinfocharts.com/de/comparison/ethereum-transactionfees.html#3y (accessed: May 18 2022).

[9] T. Hakki, "Ethereum Gas Fees Have Risen 2,300% Since June," *Decrypt*, 30 Oct., 2021. https://decrypt.co/84866/ethereum-gas-fees-have-risen-2300-since-june (accessed: May 18 2022).

[10] F. Klein, *Kulturgeschichte des Geldflusses: Die entwicklung des Zahlungsverkehrs mit fokus Schweiz,* 1st ed. Zurich: Verlag SKV, 2003. [Online]. Available: https://www.einfach-zahlen.ch/dam/download/qrrechnung/Kulturgeschichte-des-Geldflusses.pdf

[11] D. Däppen, *Sicherheit oder Vertrauen – was kommt zuerst?* [Online]. Available: https://www.infoguard.ch/de/blog/sicherheit-oder-vertrauen-was-kommt-zuerst (accessed: May 2 2022).

[12] A. Hayes, "Blockchain Explained," *Investopedia*, 13 Jun., 2014. https://www.investopedia.com/terms/b/blockchain.asp (accessed: Jun. 14 2022).

[13] *Block Chain — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/devguide/block_chain.html (accessed: May 31 2022).

[14] Ethereum Wiki, *patricia-tree | Ethereum Wiki.* [Online]. Available: https://eth.wiki/fundamentals/patricia-tree (accessed: Jun. 13 2022).
[15] Armend Lesi und Marco Endres, *Kryptowährungen als Zahlungsmittel bei FlatFeeStack*, 2021.
[16] T. Kölblinger, "Blockchain und der Konsens Algorithmus," *BLOGCHAIN*, 23 Dec., 2021. https://imb-student.donau-uni.ac.at/blogchain/blockchain-und-der-konsens-algorithmus/ (accessed: Jun. 15 2022).
[17] S. Squarepants, "Bitcoin: A Peer-to-Peer Electronic Cash System," *SSRN Journal*, 2008, doi: 10.2139/ssrn.3977007.
[18] *Wallets — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/devguide/wallets.html (accessed: Jun. 14 2022).
[19] A. Eberle, "Was ist ein (Bitcoin) HD Wallet?," *Coincierge*, 05 Feb., 2018. https://coincierge.de/2018/bitcoin-hd-wallet/ (accessed: May 31 2022).
[20] *Wallets — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/devguide/wallets.html#hierarchical-deterministic-key-creation (accessed: May 31 2022).
[21] *Transactions — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/examples/transactions.html (accessed: Jun. 6 2022).
[22] *Payment Processing — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/devguide/payment_processing.html#verifying-payment (accessed: Jun. 6 2022).
[23] *Address reuse - Bitcoin Wiki.* [Online]. Available: https://en.bitcoin.it/wiki/Address_reuse (accessed: Jun. 7 2022).
[24] M. Recksiek, "Ethereum Mixer erreicht neues Allzeithoch - ETH Transaktionen anonymisieren mit Tornado Cash," *Bitcoin2Go*, 14 Nov., 2020. https://bitcoin-2go.de/ethereum-mixer-erreicht-neues-allzeithoch-eth-transaktionen-anonymisieren-mit-tornado-cash/ (accessed: Apr. 11 2022).
[25] S. User, *Was sind Kryptowährungs-Mixer?* [Online]. Available: https://www.eurospider.com/de/know-how/compliance/210-was-sind-kryptow%C3%A4hrungs-mixer (accessed: Apr. 11 2022).
[26] Horizen Academy, *UTXO vs. Account Model.* [Online]. Available: https://academy.horizen.io/technology/expert/utxo-vs-account-model/ (accessed: May 31 2022).
[27] *Ether — Ethereum Homestead 0.1 documentation.* [Online]. Available: https://ethdocs.org/en/latest/ether.html (accessed: Jun. 15 2022).
[28] O'Reilly Online Learning, *Mastering Ethereum.* [Online]. Available: https://www.oreilly.com/library/view/mastering-ethereum/9781491971932/ch04.html (accessed: May 18 2022).
[29] D. Scheuermann, "Was ist EIP-1559 und kann die Implementierung Ethereum helfen, deflationär zu werden?," *Crypto Valley Journal*, 05 Aug., 2021. https://cvj.ch/fokus/hintergrund/was-ist-eip-1559-und-kann-die-implementierung-ethereum-helfen-deflationaer-zu-werden/ (accessed: Jun. 7 2022).
[30] GitHub, *EIPs/eip-1559.md at master · ethereum/EIPs.* [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md (accessed: Apr. 11 2022).
[31] *Gas and fees | ethereum.org.* [Online]. Available: https://ethereum.org/en/developers/docs/gas/ (accessed: Mar. 15 2022).
[32] reddit, *r/ethereum - How to completely empty a wallet?* [Online]. Available: https://www.reddit.com/r/ethereum/comments/q6og0z/how_to_completely_empty_a_wallet/ (accessed: May 18 2022).
[33] *What Is Nonce? | MyCrypto Knowledge Base.* [Online]. Available: https://support.mycrypto.com/general-knowledge/ethereum-blockchain/what-is-nonce/ (accessed: Jun. 5 2022).
[34] CoinMarketCap, *Cryptocurrency Prices, Charts And Market Capitalizations | CoinMarketCap.* [Online]. Available: https://coinmarketcap.com/ (accessed: Jun. 14 2022).

[35] N. Reiff, “Were There Cryptocurrencies Before Bitcoin?,” *Investopedia*, 03 Apr., 2018 (accessed: Jun. 14 2022).
[36] BitcoinBlog.de - das Blog für Bitcoin und andere virtuelle Währungen, *P2PK, P2PKH, P2SH, P2WPK – was diese wichtigen Abkürzungen bedeuten.* [Online]. Available: https://bitcoinblog.de/2019/07/19/p2pk-p2pkh-p2sh-p2wpk-was-diese-wichtigen-abkuerzungen-bedeuten/ (accessed: Jun. 6 2022).
[37] FixedFloat, *Bitcoin-Adressformate und Leistungsvergleich | FixedFloat.* [Online]. Available: https://fixedfloat.com/de/blog/guides/bitcoin-address-formats (accessed: Jun. 6 2022).
[38] J. Frankenfield, “Segregated Witness (SegWit),” *Investopedia*, 03 May., 2017. https://www.investopedia.com/terms/s/segwit-segregated-witness.asp (accessed: Jun. 7 2022).
[39] *Bitcoin Fee Calculator & Estimator - BTC & USD - Segwit Support.* [Online]. Available: https://btc.network/estimate (accessed: May 28 2022).
[40] *estimatesmartfee — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/reference/rpc/estimatesmartfee.html (accessed: May 28 2022).
[41] GitHub, *bitcoin/policy.cpp at v22.0 · bitcoin/bitcoin.* [Online]. Available: https://github.com/bitcoin/bitcoin/blob/v22.0/src/policy/policy.cpp (accessed: Jun. 6 2022).
[42] GitHub, *bitcoin/policy.cpp at 623745ca74cf3f54b474dac106f5802b7929503f · bitcoin/bitcoin.* [Online]. Available: https://github.com/bitcoin/bitcoin/blob/v22.0/src/policy/policy.cpp#L14 (accessed: May 28 2022).
[43] GitHub, *bitcoin/spend.cpp at v22.0 · bitcoin/bitcoin.* [Online]. Available: https://github.com/bitcoin/bitcoin/blob/v22.0/src/wallet/spend.cpp (accessed: Jun. 6 2022).
[44] *Bech32 - Bitcoin Wiki.* [Online]. Available: https://en.bitcoin.it/wiki/Bech32 (accessed: May 28 2022).
[45] River Financial, *Replace-By-Fee (RBF) | River Glossary.* [Online]. Available: https://river.com/learn/terms/r/replace-by-fee-rbf/ (accessed: May 31 2022).
[46] H. Tremp, *ARCHITEKTUREN VERTEILTER SOFTWARESYSTEME: Soa & microservices - mehrschichtenarchitekturen -… anwendungsintegration,* 1st ed. [S.l.]: MORGAN KAUFMANN, 2021.
[47] *SCS: Self-Contained Systems.* [Online]. Available: https://scs-architecture.org/ (accessed: Jun. 15 2022).
[48] microservices.io, *Microservices Pattern: Microservice Architecture pattern.* [Online]. Available: https://microservices.io/patterns/microservices.html (accessed: Jun. 15 2022).
[49] Alibaba Cloud Community, *What Is Microkernel Architecture Design?* [Online]. Available: https://www.alibabacloud.com/blog/what-is-microkernel-architecture-design_597605 (accessed: Jun. 15 2022).
[50] microservices.io, *Microservices Pattern: Database per service.* [Online]. Available: https://microservices.io/patterns/data/database-per-service.html (accessed: Jun. 15 2022).
[51] Nishanil, *Communication in a microservice architecture.* [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture (accessed: Jun. 15 2022).
[52] J. Ratliff, “Home - Docker,” *Docker*, 10 May., 2022. https://www.docker.com/ (accessed: Jun. 15 2022).
[53] *OpenAPI Specification - Version 3.0.3 | Swagger.* [Online]. Available: https://swagger.io/specification/ (accessed: Jun. 15 2022).
[54] GORM, *GORM.* [Online]. Available: https://gorm.io/index.html (accessed: May 18 2022).
[55] A. Roan, “How to Call APIs From Ethereum Smart Contracts - Better Programming,” *Better Programming*, 14 Jul., 2020. https://betterprogramming.pub/how-to-call-apis-from-ethereum-smart-contracts-e2f1500198c7 (accessed: Jun. 7 2022).

[56] *Transactions — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/devguide/transactions.html (accessed: Jun. 6 2022).
[57] *sendtoaddress — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/reference/rpc/sendtoaddress.html (accessed: Jun. 7 2022).
[58] *Average order value benchmarks for eCommerce.* [Online]. Available: https://marketing.dynamicyield.com/benchmarks/average-order-value/ (accessed: Jun. 2 2022).
[59] Etherscan.io, *Ethereum Full Node Sync (Default) Chart | Etherscan.* [Online]. Available: https://etherscan.io/chartsync/chaindefault (accessed: Jun. 13 2022).
[60] E. Foundation, *Ask about Geth: Snapshot acceleration.* [Online]. Available: https://blog.ethereum.org/2020/07/17/ask-about-geth-snapshot-acceleration/ (accessed: Jun. 13 2022).
[61] E. Foundation, *Geth v1.10.0.* [Online]. Available: https://blog.ethereum.org/2021/03/03/geth-v1-10-0/ (accessed: Jun. 13 2022).
[62] *FAQ | Go Ethereum.* [Online]. Available: https://geth.ethereum.org/docs/faq (accessed: Jun. 13 2022).
[63] GitHub, *devp2p/snap.md at master · ethereum/devp2p.* [Online]. Available: https://github.com/ethereum/devp2p/blob/master/caps/snap.md (accessed: Jun. 13 2022).
[64] *RPC API Reference — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/reference/rpc/index.html (accessed: Jun. 15 2022).
[65] *bitcoind(1) — Arch manual pages.* [Online]. Available: https://man.archlinux.org/man/bitcoind.1.en (accessed: Jun. 15 2022).
[66] GitHub, *bitcoin/zmq.md at master · bitcoin/bitcoin.* [Online]. Available: https://github.com/bitcoin/bitcoin/blob/master/doc/zmq.md (accessed: Jun. 15 2022).
[67] *createwallet — Bitcoin.* [Online]. Available: https://developer.bitcoin.org/reference/rpc/createwallet.html (accessed: Jun. 15 2022).
[68] GitHub, *btcsuite/btcwallet: A secure bitcoin wallet daemon written in Go (golang).* [Online]. Available: https://github.com/btcsuite/btcwallet (accessed: Jun. 15 2022).
[69] *BlockCypher - Blockchain Web Services.* [Online]. Available: https://www.blockcypher.com/ (accessed: Jun. 15 2022).
[70] E. Forbes, *Part 4 - Handling Multiple Clients.* [Online]. Available: https://tutorialedge.net/projects/chat-system-in-go-and-react/part-4-handling-multiple-clients/ (accessed: May 10 2022).
[71] *Go by Example: Channels.* [Online]. Available: https://gobyexample.com/channels (accessed: May 18 2022).
[72] Infura Community, *Fetch Internal transactions in a block / per transaction hash - Questions - Infura Community.* [Online]. Available: https://community.infura.io/t/fetch-internal-transactions-in-a-block-per-transaction-hash/1186 (accessed: Jun. 2 2022).
[73] *Accounts.* [Online]. Available: https://docs.etherscan.io/api-endpoints/accounts#get-a-list-of-internal-transactions-by-address (accessed: Jun. 2 2022).
[74] J. Frankenfield, "51% Attack," *Investopedia*, 07 Sep., 2016. https://www.investopedia.com/terms/1/51-attack.asp (accessed: Jun. 15 2022).
[75] A. Behrens, "How Coinbase dodged the $9 million Ethereum Classic 51% attacks," *Decrypt*, 22 Aug., 2020. https://decrypt.co/39513/how-coinbase-dodged-the-9-million-ethereum-classic-51-attacks (accessed: Jun. 2 2022).
[76] Quora, *Why is it that bitcoin transactions require multiple "confirmations"? (Could something change between the 1st and 3rd confirmation that c.* [Online]. Available: https://www.quora.com/Why-is-it-

that-bitcoin-transactions-require-multiple-confirmations-Could-something-change-between-the-1st-and-3rd-confirmation-that-could-alter-the-transaction (accessed: Jun. 5 2022).
[77] *Cost of a 51% Attack for Different Cryptocurrencies | Crypto51.* [Online]. Available: https://www.crypto51.app/ (accessed: Jun. 5 2022).
[78] Blockchain Patterns, *X-Confirmation - Blockchain Patterns.* [Online]. Available: https://research.csiro.au/blockchainpatterns/general-patterns/security-patterns/x-confirmation/ (accessed: Jun. 2 2022).
[79] *Code Generation | Redux Toolkit.* [Online]. Available: https://redux-toolkit.js.org/rtk-query/usage/code-generation#openapi (accessed: May 18 2022).
[80] *Hello from OpenAPI Generator.* [Online]. Available: https://openapi-generator.tech/ (accessed: May 18 2022).
[81] GitHub, *DATA-DOG/go-sqlmock: Sql mock driver for golang to test database interactions.* [Online]. Available: https://github.com/DATA-DOG/go-sqlmock (accessed: May 18 2022).
[82] GitHub, *h2non/gock: HTTP traffic mocking and testing made easy in Go*. [Online]. Available: https://github.com/h2non/gock (accessed: May 18 2022).
[83] M. Bauer, "ETH 2.0: Das große Ethereum-Upgrade kommt später," *COMPUTER BILD*, 20 Apr., 2022. https://www.computerbild.de/artikel/cb-News-Finanzen-ETH-2.0-Ethereum-Upgrade-kommt-spaeter-32490585.html (accessed: Jun. 5 2022).

11. Anhang

Alle Anhänge wurden separat abgegeben. Dabei wurden folgende Anhänge abgegeben:

- Projektplan
- Zeitrapport
- Systemtest-Spezifikation
- Betriebsdokumentation
- Source Code
- Protokolle

Building a Cryptocurrency Payment Provider
Anhang

Bachelorarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühlingssemester 2022

Autor(en):	Armend Lesi und Marco Endres
Betreuer:	Dr. Thomas Bocek
Experte:	Dr. Guilherme Sperb Machado
Gegenleser:	Prof. Stefan Richter

Inhaltsverzeichnis

A. Projektplan

Wir arbeiten zu zweit an dieser Bachelorarbeit. Unterstützt werden wir von unserem Betreuer Dr. Thomas Bocek und dem Experten Dr. Guilherme Serb Machado. Weil nur zwei Personen am Projekt beteiligt sind, haben wir uns entschieden eine schlanke Projektplanung zu gestalten. Dabei ist es wichtig, dass die Teammitglieder eng zusammenarbeiten und es einmal in der Woche eine Besprechung mit dem Betreuer und Experten gibt.

In diesem Projekt müssen zuerst die Anforderungen gesammelt und anschliessend ein passendes Produkt entwickelt werden. Aus diesem Grund haben wir uns entschieden nach Scrum zu arbeiten und wöchentliche Sprints geplant, um möglichst schnell auf neue Erkenntnisse reagieren zu können. Weil die Bachelorarbeit einen fixen Endtermin hat, wurden Meilensteine definiert, um das Ziel nicht aus den Augen zu verlieren.

A.1 Arbeitsaufteilung

Grundsätzlich sollten beide Teammitglieder alles verstehen. Jedoch ist es nicht möglich das beide alles bis ins kleinste Detail verstehen. Deshalb haben wir die Arbeit in zwei Bereiche unterteilt. Das Backend und die Bitcoin Anbindung wird von Armend Lesi und das Frontend und die Ethereum Anbindung von Marco Endres implementiert.

A.1 Zeitliche Planung

Das Projekt **beginnt am 21.02.2022** und **endet am 17.06.2022**. In diesem Zeitraum gibt es eine Woche Ferien in der KW 15 (11.04.2022 – 17.04.2022). Zusätzlich wurden insgesamt 16 Sprints geplant.

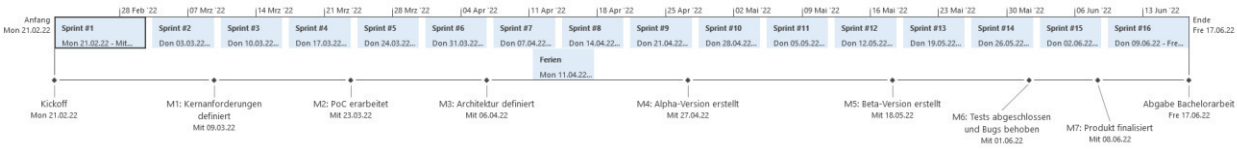

Abbildung 1: Projektplan

A.1.1 Zeitbudget

Hier sind die Eckpunkte zum Zeitbudget aufgelistet.

- 360 Stunden pro Person
- Total 16 Wochen
    - 14 Wochen an denen Teilzeit an der Bachelorarbeit gearbeitet wird.
    - An den letzten 2 Wochen wird 100% an der Bachelorarbeit gearbeitet.
- KW 15 (11.04.2022 – 17.04.2022) Frühlingsferien
- In den 14 Wochen Teilzeit werden 20 Stunden pro Woche geleistet.
- In den letzten 2 Wochen werden 40 Stunden pro Woche geleitet.

A.1.1 Besprechungen

Es ist eine Besprechung mit allen vier Teilnehmern einmal in der Woche am Mittwoch von 09:00 Uhr bis 10:00 Uhr geplant. Einmal im Monat findet die Sitzung vor Ort an der OST statt. Je nach Verfügbarkeit kann sich der Termin jedoch ändern.

A.1.2 Sprints

Durch die kurze Dauer des Projektes, haben wir uns für 1-wöchige Sprints entschieden. Ein Sprint beginnt immer am Donnerstag und endet am Mittwoch.

A.1.3 Meilensteine

Es wurden insgesamt 7 Meilensteine definiert. Sie sollen helfen das Ziel bis zum 17.06.2022 zu erreicht.

M1: Kernanforderungen definiert

- Datum: 09.03.2022
- Die wichtigsten User Stories sind erfasst und priorisiert.
- Nicht funktionale Anforderungen sind definiert und priorisiert.
- Das Lizenzmodell wurde definiert.

M2: PoC erarbeitet

- Datum: 23.03.2022
- Entwicklungsumgebung ist aufgesetzt.
- Datenbank, Backend und Frontend starten im Docker.
- GitHub Repository ist eingerichtet.
- Blockchain lokal zum Entwickeln ist aufgesetzt.
- Unit Testing ist getestet.

M3: Architektur definiert

- Datum: 06.04.2022
- Applikationsarchitektur ist definiert.
- Datenbank UML ist erstellt.
- Schnittstellen sind definiert.

M4: Alpha-Version erstellt

- Datum: 27.04.2022
- Kernanforderungen sind umgesetzt.

M5: Beta-Version erstellt

- Datum: 18.05.2022
- 80% der geplanten User Stories sind umgesetzt.

M6: Tests abgeschlossen und Bugs behoben

- Datum: 01.06.2022
- Systemtest sind abgeschlossen.
- Bugs sind beheben.
- Feinschliff wurde gemacht.

M7: Produkt finalisiert

- Datum: 08.06.2022
- Wichtigste User Stories sind umgesetzt.
- Wichtigste NFA sind umgesetzt.
- Reservezeit

A.1.4 Wichtige Termine

Zusätzlich zu den Meilensteinen gibt es noch wichtige Termine, die eingehalten werden müssen.

- 21.02.2022: Beginn der Bachelorarbeit
- 13.06.2022: Abstract im Online Tool erfassen
- 15.06.2022: Korrigiertes Abstract wird weitergeleitet
- 17.06.2022: Abgabe bis 17:00 Uhr
- Bis 31.08.2022: Mündliche BA-Prüfung

B. Anforderungen

In diesem Kapitel werden die gesammelten Anforderungen beschrieben. Dabei wurden jeweils funktionale und nicht funktionale Anforderungen definiert.

B.1 Funktionale Anforderungen

Es wurden insgesamt 22 funktionale Anforderungen gesammelt. In der Tabelle 1: Funktionale Anforderungen sind alle Anforderungen aufgelistet.

Nr.	Beschreibung	Abhängigkeit	Priorität
ANF-01	Der Händler registriert sich beim Service.		1
ANF-02	Der Händler meldet sich beim Service an.	ANF-01	1
ANF-03	Der Händler meldet sich beim Service ab.	ANF-01	1
ANF-04	Der Händler erstellt einen API Key für die Kommunikation zu CHainGate.		1
ANF-05	CHainGate bietet Ethereum als Zahlungsmethode an.		1
ANF-06	Der Händler erstellt eine neue Einzahlung.	ANF-01, ANF-04	1
ANF-07	Der Händler erhält nach der Erstellung einer Einzahlung, die Adresse, an welche das Geld geschickt werden muss und den zu bezahlenden Betrag in der gewählten Kryptowährung.	ANF-06	1
ANF-08	Der Händler erhält Updates zu den erstellten Zahlungen. Updates: waiting, finished, failed	ANF-06	1
ANF-09	Der Händler hat die Möglichkeit zwischen der Produktiven und Testumgebung zu wechseln.		1
ANF-10	Der Händler definiert pro Kryptowährung eine Wallet Adresse.		1
ANF-11	CHainGate bietet USD und CHF als Grundwährung an.		1
ANF-12	Der Händler sieht auf dem Dashboard Log-Informationen.	ANF-06, ANF-08	2
ANF-13	Der Händler erhält Updates zu den erstellten Zahlungen. Updates: partially_paid	ANF-08	2
ANF-14	CHainGate bietet Bitcoin als Zahlungsmethode an.		2
ANF-15	Der Händler aktiviert die Kryptowährungen, welche er seinen Kunden anbieten will.	ANF-05, ANF-14	2

ANF-16	Der Händler fragt die aktivierten Kryptowährungen ab.	ANF-15	2
ANF-17	Der Händler erstellt eine Einzahlung mit einer Payment-Seite, welche bei CHainGate gehostet wird.		2
ANF-18	Der Handler benutzt die CHainGate Backend Library für die Kommunikation zu CHainGate.		2
ANF-19	Der Händler fragt den Status des CHainGate Services ab.		2
ANF-20	CHainGate zieht eine kleine Gebühr pro Transaktion ab.		2
ANF-21	Der Händler definiert bis zu welcher Abweichung die Kunden-zahlung akzeptiert wird.		3
ANF-22	Der Händler fügt zusätzliche Daten der Einzahlung hinzu, welche bei jedem Update mitgeschickt werden.		3

Tabelle 1: Funktionale Anforderungen

B.2 Nicht funktionale Anforderungen

Es wurden 4 nicht funktionale Anforderungen definiert.

Nr.	Beschreibung	Abhängigkeit	Priorität
NFA-01	Neue Kryptowährungen können einfach hinzugefügt werden.		1
NFA-02	Getätigte Zahlungen gehen bei einem Systemabsturz oder Netzwerkverlust nicht verloren.		1
NFA-03	Erstellte Zahlungen gehen bei einem Systemabsturz nicht verloren.		1
NFA-04	Update Meldungen können nicht gefälscht werden.		1

Tabelle 2: Nicht funktionale Anforderungen

Da es sich um einen Prototyp handelt und keine Marktanalyse getätigt wurden, wurden wenige nicht funktionale Anforderungen gesammelt. Durch eine Marktanalyse könnten noch folgende Anforderungen hinzukommen.

- Das System muss **X** Transaktionen pro Sekunde / Minute / Stunde verarbeiten können.
- Das System benötigt eine Verfügbarkeit von **X**.
- Der höchste zu verarbeitende Betrag beträgt **X**.
- Der niedrigste zu verarbeitende Betrag beträgt **X**.
- Die Beträge müssen auf **X** Nachkommastellen genau sein.

C. Architektur

Für den Aufbau der Architektur wurde vier verschiedene Architekturen miteinander verglichen. Dabei wurden die Eigenschaften sowie die Vor- und Nachteile der verschiedenen Varianten analysiert und Ausgewertet.

C.1 Monolith

Ein monolithisches System zeichnet sich dadurch aus, dass es als eine Einheit deployt werden kann. Dabei wird das gesamte System gemäss der Mehrschichtenarchitektur in mehreren horizontalen Schichten unterteilt. Jede Schicht hat ihre klar zugeordnete Aufgabe und die Zugriffe geschehen immer von einer höheren auf eine tiefere oder innerhalb einer Schicht. Die Anzahl und deren Bezeichnung ist dabei nicht definiert. Jedoch existieren dabei oft folgende drei Schichten. [1]

- Präsentationsschicht – Das Frontend für die Benutzerschnittstelle.
- Logikschicht – Die Businesslogik der Applikation.
- Datenhaltungsschicht – Zuständig für das Speichern der Daten in die Datenbank.

In einem monolithischen System wird eine Architekturplattform oder ein Technologie Stack für die gesamte Applikation gewählt.

Folgende Vor- und Nachteile besitzt ein monolithisches System. [2, 3]

Vorteile

+ Kleinere Systeme sind einfacher zu Entwicklern, weil die Logik zentralisiert ist.
+ Das deployen des Systems ist einfacher, weil es aus einer Einheit besteht und zum Beispiel nur eine WAR-Datei deployt werden muss.
+ Ein einheitlicher Technologie Stack oder die gewählte Architekturplattform hat den Vorteil, dass die Technologie gut beherrscht wird.

Nachteile

– Die gesamte Applikation muss bei einem Update neu deployt werden.
– Grosse und komplexe Systeme sind schwer zu verstehen und es ist schwierig Änderungen schnell und korrekt durchzuführen.
– Durch die grosse und komplexe Codebasis dauert das Einarbeiten neuer Entwickler länger.
– Bei einer grossen Applikation ist die Startzeit länger, was die Entwicklungszeit negativ beeinflusst.
– Der Einfluss von Änderungen ist nicht immer vorhersehbar, deshalb muss das System als ganzen stark getestet werden, um mögliche Bugs frühzeitig zu finden.
– Continuous deplyoment ist schwierig, weil bei einer kleinen Anpassung an einem Modul die gesamte Applikation neu deployt werden muss, auch wenn ein Grossteil der Applikation nicht betroffen ist.
– Fehler in einem Teil des Codes können potenziell das gesamte System beeinflussen.

- Neue Technologien zu integrieren ist schwierig, weil eine Anpassung das gesamte System beeinflusst.
- Die Entscheidung welche Architekturplattform oder Technologie Stack genutzt werden soll, muss früh entschieden werden und ist eine langfristige Entscheidung. Treten Probleme auf, welche mit der gewählten Technologie nicht gelöst werden können, kann nicht einfach eine neue gewählt oder ergänzt werden.

C.2 Self-Contained System

Self-Contained Systeme (SCS) haben das Ziel das Gesamtsystem in verschiedene unabhängige Teilsysteme aufzuteilen. Die Teilsysteme werden gemäss ihrer Funktionalität aufgeteilt. SCS sind sehr ähnlich zu den Microservice Systemen, jedoch sind SCS in der Regel grösser und bestehen aus einer Benutzeroberfläche, der Businesslogik und einer Datenbank. [4]

Hier sind die 8 Merkmale eines SCS. [4]

1. Jedes SCS ist eine eigenständige Webanwendung. Sie beinhalten die Benutzeroberfläche, die Daten und die Logik. Die primäre Aufgabe soll unabhängig von anderen Systemen erfüllbar sein.
2. Jedes SCS gehört einem Team.
3. Die Kommunikation zwischen verschiedenen SCSs oder Umsystemen soll, wenn möglich, asynchron geschehen. Dabei sollen SCSs auch weiterhin funktionieren, wenn andere SCSs temporär ausfallen.
4. Ein SCS kann optional eine API-Schnittstelle anbieten.
5. Jedes SCS muss die Daten und Logik beinhalten.
6. Das Feature eines SCS soll von einem Benutzer direkt über dessen Benutzeroberfläche benutzbar sein. Somit sollte keine geteilte Benutzeroberfläche existieren.
7. Ein SCS sollte keine geteilte Businesslogik mit anderen SCSs besitzen.
8. Geilte Infrastruktur soll minimiert werden, um ein SCS robuster und unabhängiger zu machen. Jedoch kann es Gründe für eine geteilte Infrastruktur geben. Zum Beispiel kann, um Geld zu sparen, eine geilte Datenbank mit separaten Schemas eingesetzt werden.

Folgende Vor- und Nachteile besitzt ein SCS. [4]

Vorteile

+ Saubere Trennung zwischen den einzelnen Systemen.
+ Die einzelnen Systeme können völlig unabhängig von weiteren Systemen bedient werden.
+ Einzelne SCS können gut skaliert werden.
+ Dadurch das die SCS unabhängig voneinander sind, besteht eine hohe Fehlertoleranz.
+ Die SCS können separat gewartet werden.

Nachteile

- Ein einheitliches Benutzerinterface wird erschwert, da jedes SCS sein eigenes besitzt.
- Jedes SCS braucht eine eigene Benutzeroberfläche, damit es unabhängig von weiteren Systemen genutzt werden kann.
- System in einzelne SCS zu unterteilen ist schwierig.

C.3 Microservice

Bei der Microservice Architektur wird das gesamte System in kleine unabhängige Services aufgeteilt. Jeder Service hat dabei eine Aufgabe und ist klar von den anderen abgegrenzt. Die Kommunikation untereinander geschieht über klar definierte synchrone oder asynchrone Schnittstellen. Für die Integration der Benutzeroberfläche und der Datenbank gibt es unterschiedliche Methoden. [1]

Bei der Benutzeroberfläche gibt es drei Möglichkeiten

1. Jeder Service hat seine eigene Oberfläche und sie werden über Links miteinander verknüpft
2. Jeder Service hat einen Teil der Oberfläche und werden über eine Rahmenoberfläche miteinander verknüpft.
3. Es existiert ein eigener Service für die Oberfläche und die Services bieten nur eine API-Schnittstelle an.

Bei der Datenbank gibt es zwei Möglichkeiten

1. Es gibt eine Datenbank und alle Services greifen auf diese zu.
2. Jeder Service hat seine eigene Datenbank.

Folgende Vor- und Nachteile besitzt eine Microservice Architektur. [2, 5]

Vorteile

+ Gute Wartbarkeit – Kleine Services sind überschaubar und Anpassungen sind einfacher nachzuvollziehen.
+ Gute Testbarkeit – Kleine Services sind einfacher zu testen.
+ Gutes Deployment – Services können unabhängig voneinander deployt werden.
+ Kleine Teams können sich um einen oder mehrere Services kümmern. Diese können unabhängig voneinander skaliert, getestet und weiterentwickelt werden.
+ Ermöglicht continuous delivery und die Entwicklung von grossen und komplexen Applikationen.
+ Verbesserte Fehlerisolation – Ein Fehler in einem Service verursacht nicht einen Totalausfall der gesamten Applikation. Die anderen Services können weiterhin Anfragen verarbeiten.
+ Eliminiert eine langfristige Entscheidung für einen Technologie Stack für die gesamte Applikation. Jeder Service beinhaltet seinen eigenen Stack, um seine Aufgabe möglichst effizient und gut zu lösen.

Nachteile

- Kommunikation zwischen den Services muss implementiert werden.
- Es muss ein Mechanismus für einen Teilausfall entwickelt werden.
- Die Umsetzung von Anfragen, die mehrere Services betreffen, ist schwieriger
- Testen der Interaktion zwischen mehreren Services ist schwieriger.
- Entwickler Tools/IDEs sind primär für monolithische Systeme entwickelt worden und unterstützen die Entwicklung von verteilten Systemen weniger gut.
- Das Deployment der gesamten Applikation ist komplexer, weil alle Services einzeln deployt und verwaltet werden müssen.
- Erhöhter Speicherverbrauch. Ein monolithisches System wird in mehrere Microservices aufgeteilt. Wurde zum Beispiel beim Monolith eine JVM gebraucht, braucht es bei den Microservices mehr, je nach Anzahl Services.

C.4 Microkernel / Plugin

Das Microkernel Pattern besteht aus zwei Komponenten. Er besteht aus einem Kern und Plugin Modulen. Der Kern beinhaltet die minimalen Funktionalitäten, um das System lauffähig zu machen. Die Plugin Module können genutzt werden, um die Funktionalität des Systems flexibel zu erweitern. Dabei ist es wichtig, dass die Plugins unabhängig sind. Trotz der Unabhängigkeit ist es möglich, das Plugins untereinander über den Kern kommunizieren können. [6]

Folgende Vor- und Nachteile besitzt eine Microkernel Architektur. [6–8]

Vorteile

+ Plugins können unabhängig voneinander weiterentwickelt und gewartet werden.
+ Abhängig von der Implementation, können die Plugins dynamisch zur Laufzeit hinzugefügt werden.
+ Plugins können einzeln getestet und gemockt werden.
+ Beim System können nur die gebrauchten Plugins aktiviert werden, dies führt zu einer besseren Performance.

Nachteile

- Generell sind Microkernel Systeme nicht stark skalierbar. Jedoch kann mit der richtigen Plugin Implementation die Skalierbarkeit auf Plugin Ebene verbessert werden.
- Es braucht ein Vertragsmanagement zwischen Kern und Plugin, was die Entwicklung erschwert. Zudem gibt es komplexe Themen wie Vertrags Versionierung, Plugin Registrierung, Plugin Granularität und Plugin Verbindungsmöglichkeiten, welche beantwortet werden müssen.

C.5 Analyse

Um die optimale Architektur zu wählen, wurden fünf verschiedene Kriterien gewählt und gewichtet. Dabei wurden alle Kriterien mit einer Bewertung von 0 bis 9 bewertet. 0 ist das Schlechteste und 9 das Beste.

C.5.1 Kriterien

Erweiterbarkeit

Bei CoinMarketCap sind über 1`000 verschiedene Kryptowährungen aufgelistet und es kommen immer wieder neue hinzu. Da CHainGate ein Provider für Kryptowährungen ist, ist es wichtig das flexibel auf den Markt reagiert werden kann und die wichtigsten Währungen schnell integriert werden können. Aus diesem Grund ist die Erweiterbarkeit am wichtigsten. Zusätzlich kann es sein, dass die verschiedenen Kryptowährungen unterschiedliche Programmiersprachen und Technologien unterstützen. Daher ist es wichtig, dass bei einer Integration einer neuen Währung die bevorzugten Technologien eingesetzt werden können. [9]

Fehlertoleranz

Fehler dürfen nicht dazu führen, das Geld verloren geht. Des Weiteren ist es wichtig, dass ein Problem bei einer Währung nicht die anderen beeinflusst.

Wartbarkeit

Wenn CHainGate in Zukunft mehrere Währungen unterstützt, ist es wichtig das diese unabhängig voneinander gewartet werden können. Muss zum Beispiel Ethereum auf 2.0 aktualisiert werden, soll dies keinen Einfluss auf die anderen haben.

Skalierbarkeit

Das System soll in Zukunft einfach skalierbar sein, ohne die komplette Architektur zu ändern.

Komplexität

Die Punkte der Erweiterbarkeit, Wartbarkeit, Fehlertoleranz und Skalierbarkeit haben eine höhere Priorität wie die Komplexität. Jedoch soll die Komplexität so tief wie möglich gehalten werden.

C.5.2 Monolith

Erweiterbarkeit

Bei einem Monolith wir sich zu Beginn für einen Technologie Stack entschieden. Da CHainGate in Zukunft viele Kryptowährungen anbinden will und jede Währung verschiedene Technologien unterstützt, ist die Erweiterbarkeit sehr beschränkt. Wird sich zum Beispiel für .Net entschieden und eine zukünftige Währung nur eine Go Library besitzt, kann diese nicht integriert werden.

Fehlertoleranz

Weil ein Speicherproblem bei einer Komponente die Gesamte Applikation zum Stoppen bringen kann, ist die Fehlertoleranz niedrig. Zusätzlich kann ein Problem bei einer Währung die anderen Währungen beeinflussen.

Wartbarkeit

Solange die Applikation noch klein ist, ist die Wartbarkeit sehr gut. Wird das Projekt jedoch grösser wird es immer schwerer mit der Wartbarkeit, weil kleine Änderungen immer das gesamte System betreffen.

Skalierbarkeit

Ein Monolith kann skaliert werden, indem mehrere Instanzen deployt werden. Wenn nur eine kleine Komponente Kapazitätsprobleme hat, muss trotzdem die gesamte Applikation als Instanz aufgesetzt werden. Somit werden die Ressourcen nicht optimal genutzt. Es wäre besser, wenn nur diese eine Komponente skaliert wird.

Komplexität

Kleine bis mittelgrosse Projekte sind im Vergleich zu einem SCS und Microservice weniger Komplex. Wird das Projekt hingegen grösser, steigt auch die Komplexität. Das grösste Problem hierbei ist, das die Applikation mit steigender Komplexität nicht mehr gut wartbar ist.

C.5.3 SCS

Erweiterbarkeit

Die Erweiterbarkeit bei einem SCS System ist sehr gut. Die Applikation wird in einzelne, voneinander unabhängige Services unterteilt. Die Schwierigkeit liegt hier bei der Unterteilung der Services. Sie müssen funktional voneinander unabhängig sein und jeweils eine eigene Benutzeroberfläche und Businesslogik implementieren sowie ihre eigenen Daten beinhalten. Ob dies bei CHainGate umsetzbar ist, dass jede Blockchain ihren eigenen Service hat ist unklar. Des Weiteren braucht nicht jeder Blockchain Service eine eigene Benutzeroberfläche.

Fehlertoleranz

Da jeder Service unabhängig ist, hat ein Fehler bei einem Service keinen Einfluss auf die anderen.

Wartbarkeit

Durch die Unterteilung der Applikation in funktionale Komponenten, sind die Komponenten unabhängig voneinander wartbar und tangieren die anderen nicht.

Skalierbarkeit

Die einzelnen Services können unabhängig voneinander skaliert werden.

Komplexität

Durch die einzelnen Komponenten ist die Komplexität höher als bei einem Monolith. Dabei muss die Applikation in unabhängige funktionale Einheiten aufgeteilt werden. Das Erstellen einer einheitlichen Benutzeroberfläche ist eine Herausforderung, da jeder Service seine eigene Oberfläche implementiert. Müssen die einzelnen Komponenten untereinander kommunizieren, müssen zusätzliche API-Schnittstellen definiert und entwickelt werden.

C.5.4 Microservice

Erweiterbarkeit

Durch das Einsetzen von vielen kleinen Services, ist die Applikation sehr gut erweiterbar. Kommt eine neue Kryptowährung hinzu, kann ein neuer Service erstellt werden, mit dem optimalen Technologie Stack.

Fehlertoleranz

Bei einem Microservice Ansatz ist die Fehlertoleranz sehr hoch. Jedoch ist zu bedenken, dass die einzelnen Services so implementiert werden müssen, dass sie bei einem Ausfall von externen Komponenten weiterhin funktionieren. Hier ist das Problem beim Aufwand und der Umsetzung, damit die Applikation Fehlertolerant wird.

Wartbarkeit

Die einzelnen Services sind durch ihre Grösse sehr gut wartbar. Jedoch darf die Wartbarkeit des gesamten Systems nicht ausser Acht gelassen werden. Existieren viele Microservices, die viel miteinander kommunizieren, muss eine Strategie für die Verwaltung des Gesamtsystems existieren.

Skalierbarkeit

Eine Microservice Architektur kann sehr gut vertikal skalieren. Dabei können je nach Bedarf mehrere Kopien eines Services hoch gefahren werden.

Komplexität

Je mehr Services existieren desto komplexer wird das gesamte System. Jedoch sind die einzelnen Services nicht sehr komplex.

C.5.5 Microkernel / Plugin

Erweiterbarkeit

Ein Microkernel System ist durch die Plugins sehr gut erweiterbar. Dabei kann jede Kryptowährung als Plugin implementiert werden. Die Schwierigkeit liegt bei der Komplexität.

Fehlertoleranz

Jedes Plugin sollte ohne Abhängigkeit zu den weiteren Plugins funktionsfähig sein. Somit tangieren Fehler in einem Plugin die anderen Plugins nicht.

Wartbarkeit

Durch ein Vertragsmanagement und einer Plugin Registrierung ist das System sehr gut wartbar. Weiter können alle Plugins unabhängig voneinander gewartet und weiterentwickelt werden.

Skalierbarkeit

Ein Microkernel ist in der Regel nicht sehr gut skalierbar. Jedoch können die Plugins als einzelne und unabhängige Services implementiert werden, was die Skalierung erhöht. Der Nachteil dabei ist, die erhöhte Komplexität.

Komplexität

Dadurch das ein Vertragsmanagement zwischen Kern und Plugin benötigt wird und Themen wie Vertrag Versionierung, Plugin Registrierung, Plugin Granularität und Plugin Verbindungsmöglichkeiten beantwortet werden müssen, ist diese Architektur sehr komplex.

C.5.6 Resultat

Kriterien	Gewicht	Monolith		SCS		Microservice		Microkernel	
		n	n*g	n	n*g	n	n*g	n	n*g
Erweiterbarkeit	30	4	120	6	180	9	270	9	270
Fehlertoleranz	25	5	125	8	200	8	200	9	225
Wartbarkeit	20	7	140	8	160	7	140	8	160
Skalierbarkeit	15	5	75	7	105	8	120	6	90
Komplexität	10	7	70	5	50	4	40	2	20
Summe	100		530		695		770		765
Rang			**4**		**3**		**1**		**2**

Tabelle 3: Analyse Architektur

Gemäss der Tabelle 3: Analyse Architektur ist zu sehen, dass der Gewinner eine Microservice Architektur ist. Eine Microkernel Architektur könnte für CHainGate auch in Frage kommen. Jedoch besteht bei diesem Ansatz das Problem der Skalierung. Es kann gelöst werden, indem die einzelnen Plugins als Microservices implementiert werden, jedoch ist die Komplexität grösser, weil Themen wie das Vertragsmanagement, die Plugin Registrierung und die Verbindungsmöglichkeiten hinzukommen.

Durch die Anforderungen weitere Kryptowährungen einfach integrieren zu können, kommt ein Monolith nicht in Frage. Die Blockchains unterstützen verschiedene Programmiersprachen und daher wäre es schwierig mit einem Monolith und der Einschränkung von einer Programmiersprache CHainGate zu entwickeln.

Ein SCS hat den Vorteil, dass es durch die grösseren Komponenten weniger komplex ist. Jedoch ist unklar ob bei einem SCS Ansatz CHainGate in einzelne funktionale Komponenten aufteilbar ist, wo jede Blockchain ihren eigenen Service besitzt. Dies ist nötig, damit bei jeder Blockchain die unterstütze Technologie eingesetzt werden kann. Des Weiteren ist es nötig, dass jeder Service eine eigene Benutzeroberfläche besitzt, was bei CHainGate nicht notwendig ist.

D. Kommunikation zwischen den Microservices

Es wurde sich für eine Microservice Architektur entschieden. Aus diesem Grund muss definiert werden, wie die einzelnen Services miteinander kommunizieren sollen. Dabei wird zwischen einer synchronen und asynchronen Kommunikation unterschieden. [10]

- Synchrone Kommunikation – HTTP ist ein synchrones Protokoll. Ein Client sendet über HTTP eine Anfrage an einen Server und wartet so lange bis er die Antwort erhält.
- Asynchrone Kommunikation – AMQP ist ein asynchrones Protokoll. Hier sendet der Client auch eine Anfrage an einen Server, wartet jedoch **nicht** auf eine Antwort. Es werden jeweils asynchrone Nachrichten hin und her geschickt.

Wenn möglich, sollten Microservices asynchron miteinander kommunizieren. Dadurch ist die Performance besser, da nicht immer auf eine Antwort gewartet werden muss und die Resilienz, wenn ein Service ausfällt, wird erhöht. [10]

Damit entschieden werden kann, welche Kommunikationsmethode eingesetzt wird, wurde eine Analyse mit folgenden vier Kriterien getätigt.

- Wissen
- Komplexität
- Skalierbarkeit
- Performance

D.1 Synchrone Kommunikation

Wissen

Das Wissen zur synchronen Kommunikation wie HTTP und REST ist im Team hoch. Es wurde bereits in mehreren Projekten eine REST API entwickelt und genutzt.

Komplexität

Die Architektur mit einer synchronen Kommunikation ist weniger Komplex, da keine zusätzlichen Komponenten wie ein Message Broker benötigt werden.

Skalierbarkeit

Die Skalierbarkeit ist mit einer synchronen Kommunikation nicht sehr hoch. Je mehr Services für eine Anfrage benötigt werden, desto mehr Services sind blockiert, bis die komplette Anfrage abgeschlossen ist.

Performance

Dadurch das die Services immer aufeinander warten müssen, wird die Performance schlechter, je mehr Abhängigkeiten existieren.

D.2 Asynchrone Kommunikation

Wissen

Das Wissen zur asynchronen Kommunikation wie AMQP ist im Team niedrig. Die theoretischen Grundkenntnisse sind vorhanden.

Komplexität

Die asynchrone Kommunikation ist komplexer, da zusätzlich ein Message Broker benötigt wird. Zusätzlich müssen die Services so implementiert werden, dass eine asynchrone Kommunikation möglich ist.

Skalierbarkeit

Asynchrone Kommunikation skaliert besser, weil die einzelnen Services nicht direkt voneinander abhängig sind.

Performance

Weil nicht auf die Antworten von den Abhängigen Services gewartet werden muss, ist die Performance besser.

D.3 Resultat

Ziel	Gewicht	Async		Sync	
		n	n*g	n	n*g
Wissen	30	3	90	8	240
Komplexität	30	5	150	8	240
Skalierbarkeit	20	9	180	5	100
Performance	20	8	160	5	100
Summe	100		580		680
Rang			**2**		**1**

Tabelle 4: Analyse Kommunikation

Wie in der Tabelle 4: Analyse Kommunikation zu sehen ist, hat trotz der Nachteile von der schlechteren Performance und Skalierbarkeit die synchrone Kommunikation gewonnen. Der Grund hierfür ist, dass die Applikation erst ein Proof of Concept ist und sich zu Beginn die Auslastung in Grenzen halten wird. Sollte CHainGate in Zukunft viele Anwender erhalten, sollte auf eine asynchrone Kommunikation gewechselt werden, wenn es Probleme mit der Skalierung oder Performance geben sollte. Ein weiterer Grund ist, dass die Benutzerinteraktion synchron sein sollte, damit der Benutzer direkt ein Feedback bekommt, ob etwas funktioniert hat oder nicht. Deshalb ist zu Beginn eine asynchrone Kommunikation nicht wichtig.

E. Werkzeuge

In dieser Arbeit wurden verschiedene Werkzeuge für die Entwicklung eingesetzt. Dieser Abschnitt stellt die drei wichtigsten vor.

E.1 Docker

Docker ist eine Software, welche es erlaubt Software in einem Container laufen zu lassen. Ein Container ist eine Softwareeinheit mit all den nötigen Abhängigkeiten. Somit kann diese Software überall, wo Docker installiert ist gestartet werden und alle Instanzen sind exakt gleich. Um mehrere Container gleichzeitig zu starten, wird Docker-Compose genutzt. [11, 12]

Folgende Vor- und Nachteile besitzt Docker.

Vorteile

+ Jeder Service läuft auf einem eigenen Container mit all den nötigen Abhängigkeiten. Somit kann jeder Entwickler den Service lokal starten ohne unzählige Abhängigkeiten zu installieren.
+ Jede Instanz eines Services ist immer gleich, egal wo sie läuft. Der Entwickler kann sich sicher sein, dass seine Service Instanz die gleiche ist wie die Instanz seines Kollegen oder der Produktion.
+ Alle Services sind automatisch voneinander getrennt, was die Sicherheit erhöht.
+ Einfaches starten und stoppen der gesamten Applikation über docker-compose.

Nachteile

– Es kann nicht die ganze Performance von der Hardware genutzt werden.
– Die Lernkurve von Docker ist hoch.
– Wenn viele Container verwaltet, werden müssen, reicht docker-compose nicht aus und es muss ein weiteres Tool wie zum Beispiel Kubernetes genutzt werden.
– Das Verwalten mehrerer Container ist Komplex.

Das Team besitzt bereits erste Erfahrungen mit Docker und docker-compose, somit ist ein gewisses Knowhow bereits vorhanden. Auch die Probleme der Verwaltung von vielen Containern kann zu Beginn des Projektes ignoriert werden, weil es nicht viele Container geben wird. Die nicht optimale Performance ist zu Beginn nicht relevant.

E.2 OpenAPI Spezifikation

Die OpenAPI Spezifikation ist ein Standard zur Beschreibung von RESTful APIs. Menschen sowie Computer, können durch die Spezifikation verstehen welche Möglichkeiten ein Service anbietet und wie diese genutzt werden können. Dabei ist es nicht nötig, den Source Code zu verstehen oder kennen. Die Spezifikation kann mit zusätzlicher Software genutzt werden, um den Client sowie Server Code zu generieren. Zusätzlich kann sie als Dokumentation dienen. [13]

Folgende Vor- und Nachteile besitzt OpenAPI.

Vorteile

+ Alle Services erhalten automatisch eine Schnittstellen Dokumentation. Wichtig ist, dass die Schnittstelle, welche von den Händlern genutzt wird, sauber und korrekt dokumentiert ist.
+ Dokumentation ist immer auf dem aktuellen Stand. Bevor ein neuer Endpunkt hinzukommt oder ein bestehender sich ändern, muss immer zuerst die Spezifikation angepasst werden.
+ Einfachere Kommunikation zwischen den Entwickler der einzelnen Services. Muss ein Entwickler auf einen Service zugreifen, weiss er welche Möglichkeiten dieser anbietet und wie er genutzt werden muss.
+ Bevor Code geschrieben werden kann, muss die Spezifikation erstellt werden. Dadurch wird erzwungen, sich Gedanken zum Service zu machen und nicht einfach zu entwickeln ohne konkreten Plan.
+ Durch den Einsatz von einem Generator muss weniger Code geschrieben werden. Zusätzlich ist der Code garantiert gemäss der Spezifikation.
+ Wenn sich die Spezifikation ändern, werden die Änderungen automatisch durch die Generatoren korrekt übernommen.

Nachteile

– Eine zusätzliche Software wie der Generator bedeutet, dass sie gewartet und aktualisiert werden muss.
– Nicht alle Programmiersprachen werden mit allen Features unterstützt.
– Ein zusätzliches Werkzeug muss beherrscht werden.
– Die CI/CD Pipeline ist komplexer. Der generierte Code muss jedes Mal neu generiert werden.

Die Vorteile überwiegen die Nachteile. Deshalb wurde sich entschieden OpenAPI in Kombination mit einem Generator für alle Services einzusetzen.

E.3 GORM

GORM ist eine Object-Relational Mapping (ORM) Library für Golang. Ein ORM erlaubt es, dass Objekte in einer Programmiersprache in eine relationale Datenbank abgebildet werden können. [14]

Folgende Vor- und Nachteile besitzt GORM.

Vorteile

+ Automatisches Mapping der Golang Objekte in eine Datenbank Tabelle.
+ Einfacher Zugriff auf die Datenbank, ohne viel eigenen Code zu schreiben.
+ Weniger SQL-Knowhow notwendig.
+ Automatische Migrationen sind möglich.

Nachteile

- Eine zusätzliche Library muss erlernt werden.
- Langsamer als direktes SQL.
- Komplexe SQL-Abfragen, können weniger gut umgesetzt werden. Jedoch besteht weiterhin die Möglichkeit normale SQL abfragen zu tätigen.

GORM wurde gewählt, weil viel Boilerplate-Code gespart werden kann. Die Nachteile der Performance und komplexen SQL-Abfragen, spielen aktuell keine wichtige Rolle. Sollte jedoch der Fall eintreffen, wo die Performance ein Problem wird oder komplexe SQL-Abfragen notwendig sind, besteht die Möglichkeit parallel neben GORM, SQL direkt zu nutzen.

F. Umgesetzte Anforderungen

In diesem Kapitel werden die Umgesetzten Anforderungen beschrieben. Zusätzlich werden Screenshots von der Applikation gezeigt.

A.1.1 Funktionale Anforderungen

In der Tabelle 5 sind die umgesetzten funktionalen Anforderungen grün markiert und die nicht umgesetzten rot.

Nr.	Beschreibung	Abhängigkeit	Priorität
ANF-01	Der Händler registriert sich beim Service.		1
ANF-02	Der Händler meldet sich beim Service an.	ANF-01	1
ANF-03	Der Händler meldet sich beim Service ab.	ANF-01	1
ANF-04	Der Händler erstellt einen API Key für die Kommunikation zu CHainGate.		1
ANF-05	CHainGate bietet Ethereum als Zahlungsmethode an.		1
ANF-06	Der Händler erstellt eine neue Einzahlung.	ANF-01, ANF-04	1
ANF-07	Der Händler erhält nach der Erstellung einer Einzahlung, die Adresse, an welche das Geld geschickt werden muss und den zu bezahlenden Betrag in der gewählten Kryptowährung.	ANF-06	1
ANF-08	Der Händler erhält Updates zu den erstellten Zahlungen. Updates: waiting, finished, failed	ANF-06	1
ANF-09	Der Händler hat die Möglichkeit zwischen der Produktiven und Testumgebung zu wechseln.		1
ANF-10	Der Händler definiert pro Kryptowährung eine Wallet Adresse.		1
ANF-11	CHainGate bietet USD und CHF als Grundwährung an.		1
ANF-12	Der Händler sieht auf dem Dashboard Log-Informationen.	ANF-06, ANF-08	2
ANF-13	Der Händler erhält Updates zu den erstellten Zahlungen. Updates: partially_paid	ANF-08	2
ANF-14	CHainGate bietet Bitcoin als Zahlungsmethode an.		2
ANF-15	Der Händler aktiviert die Kryptowährungen, welche er seinen Kunden anbieten will.	ANF-05, ANF-14	2

ANF-16	Der Händler fragt die aktivierten Kryptowährungen ab.	ANF-15	2
ANF-17	Der Händler erstellt eine Einzahlung mit einer Payment-Seite, welche bei CHainGate gehostet wird.		2
ANF-18	Der Handler benutzt die CHainGate Backend Library für die Kommunikation zu CHainGate.		2
ANF-19	Der Händler fragt den Status des CHainGate Services ab.		2
ANF-20	CHainGate zieht eine kleine Gebühr pro Transaktion ab.		2
ANF-21	Der Händler definiert bis zu welcher Abweichung die Kunden-zahlung akzeptiert wird.		3
ANF-22	Der Händler fügt zusätzliche Daten der Einzahlung hinzu, wel-che bei jedem Update mitgeschickt werden.		3

Tabelle 5: Umgesetzte funktionale Anforderungen

Anmerkungen

ANF-8: Es wurden im gesamten 9 verschiedene Status umgesetzt**:**

1. CurrencySelection
2. Waiting
3. PartiallyPaid
4. Paid
5. Confirmed
6. Forwarded
7. Finished
8. Expired
9. Failed

ANF-15/ANF-16: Zurzeit sind immer alle Kryptowährungen aktiviert. Sobald der Händler eine Adresse konfiguriert hat, sind Zahlungen möglich.

ANF-18: Durch die Implementierung und dem Hauptfokus auf die Einzahlungsseite rückte die Backend-Library in der Wichtigkeit nach hinten.

ANF-19: Die Kommunikation läuft bis jetzt immer von CHainGate zum Händler. Falls der Händler nicht erreichbar ist, wird es erneut versucht in einem periodischen Abstand, bis das Statusupdate erfolgreich war.

ANF-20: Die Gebühr ist prozentual und kann pro Blockchain-Service konfiguriert werden.

ANF-21: Zurzeit muss der definierte Betrag vollumfänglich bezahlt werden.

ANF-22: Zurzeit ist es nicht möglich weitere Informationen dem Service mitzugeben.

A.1.2 Nicht-Funktionale Anforderungen

Alle nicht funktionalen Anforderungen konnten umgesetzt werden.

Nr.	Beschreibung	Abhängigkeit	Priorität
NFA-01	Neue Kryptowährungen können einfach hinzugefügt werden.		1
NFA-02	Getätigte Zahlungen gehen bei einem Systemabsturz oder Netzwerkverlust nicht verloren.		1
NFA-03	Erstellte Zahlungen gehen bei einem Systemabsturz nicht verloren.		1
NFA-04	Update Meldungen können nicht gefälscht werden.		1

Tabelle 6: Umgesetzte nicht funktionale Anforderungen

A.1.3 Frontend Screenshots

In diesem Abschnitt werden verschiedene Screenshots vom Frontend abgebildet.

Webapp für die Händler

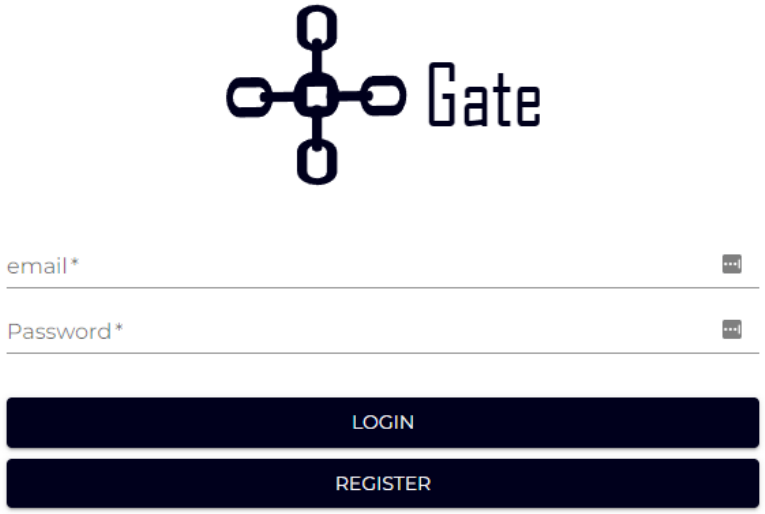

Abbildung 2: Screenshot - Login

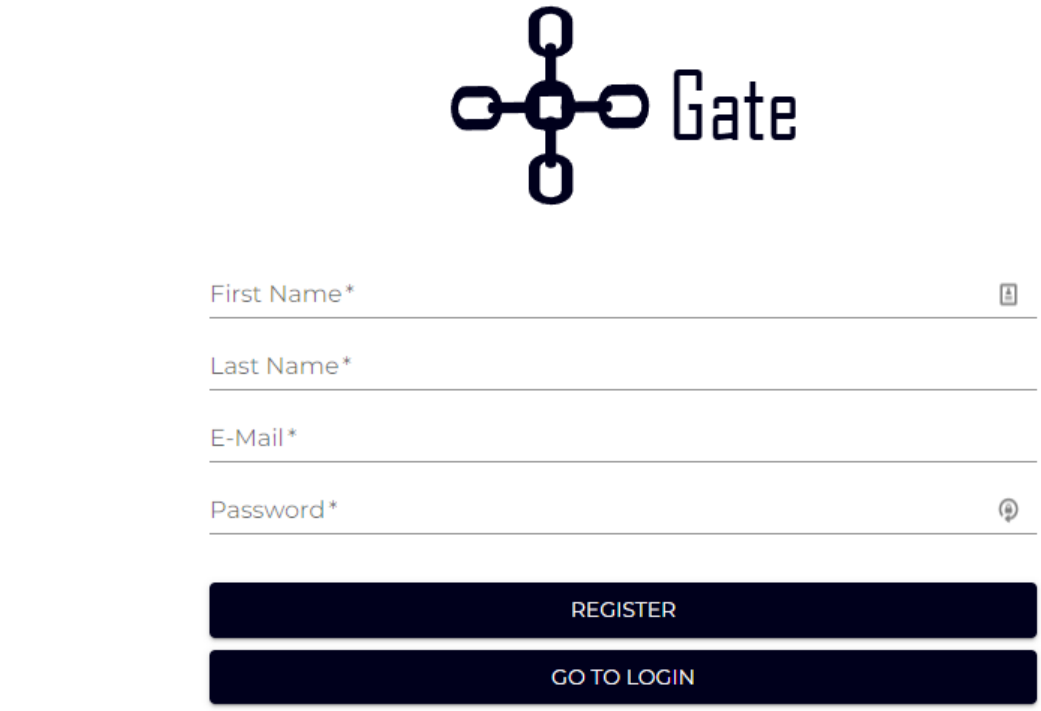

Abbildung 3: Screenshot - Registration

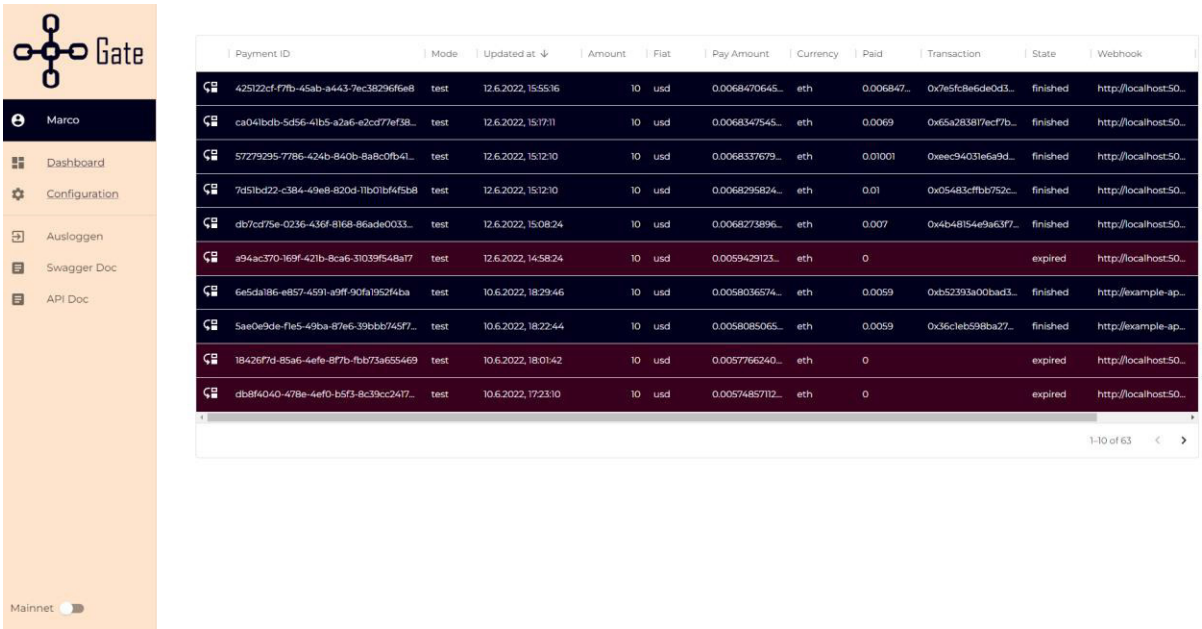

Abbildung 4: Screenshot - Dashboard

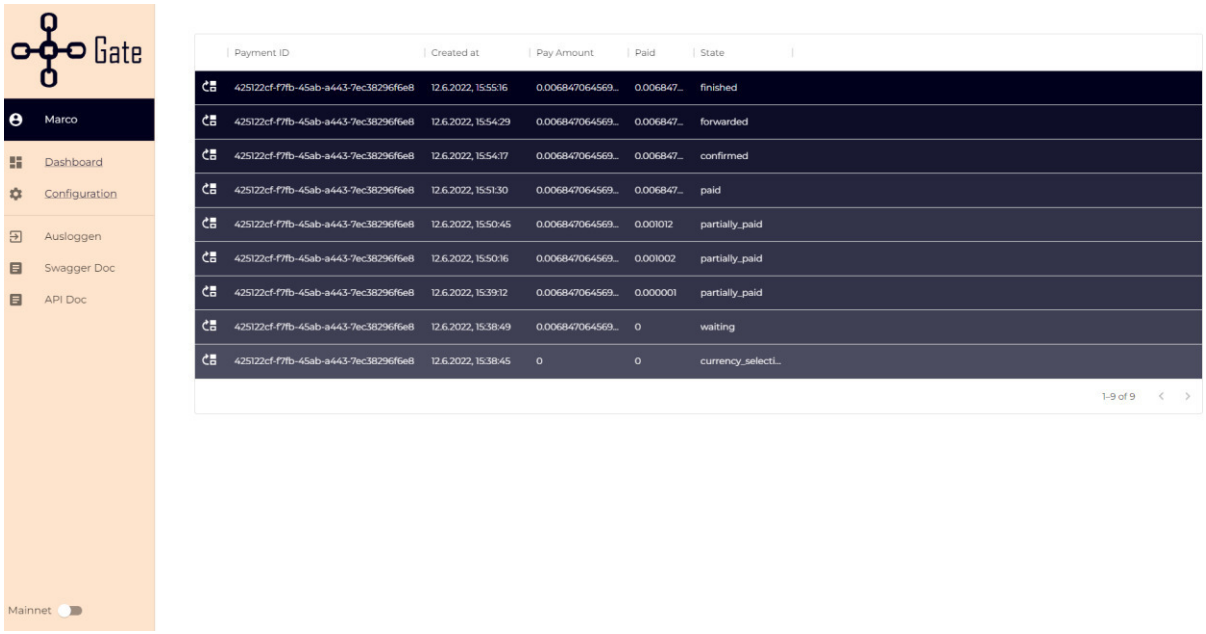

Abbildung 5: Screenshot - Detailansicht Zahlung

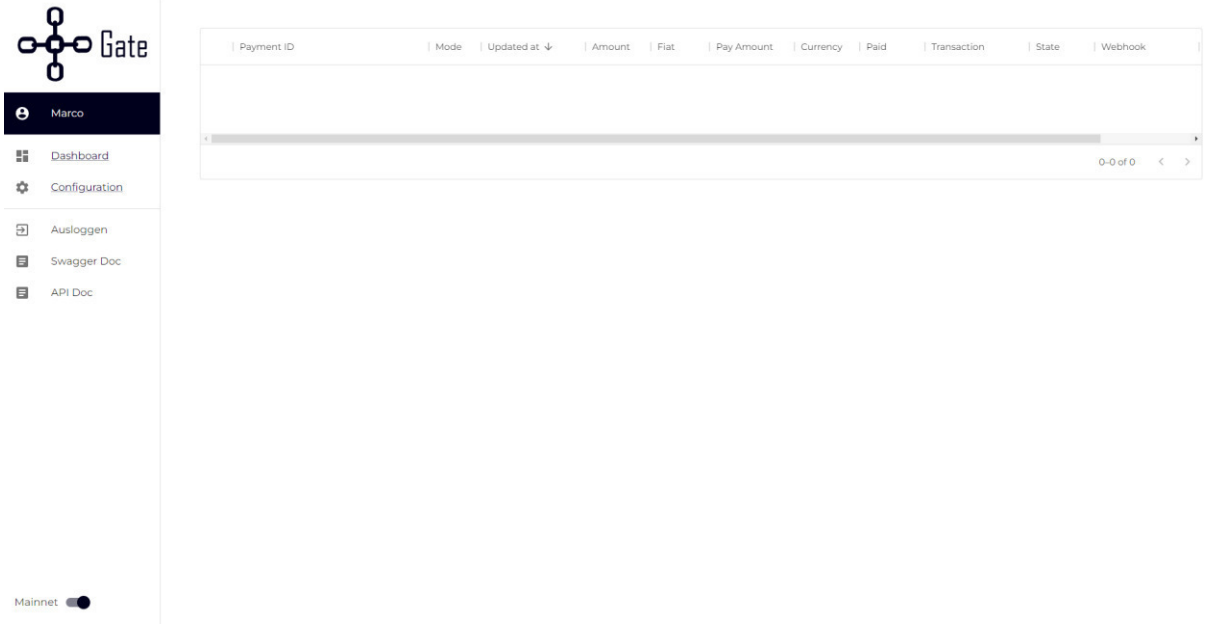

Abbildung 6: Screenshot - Umschaltung auf Mainnet

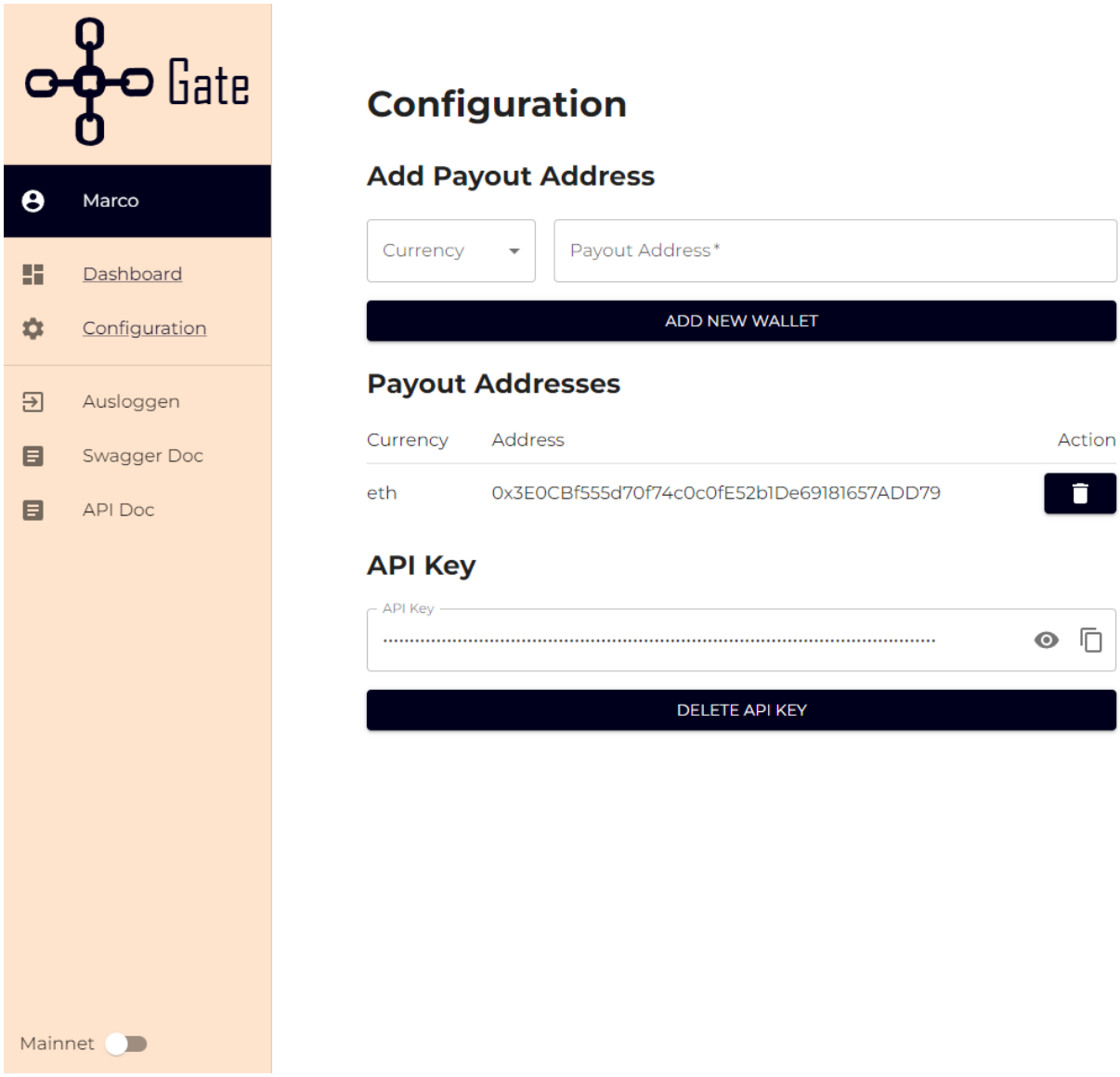

Abbildung 7: Screenshot - Konfiguration

Beispielshop

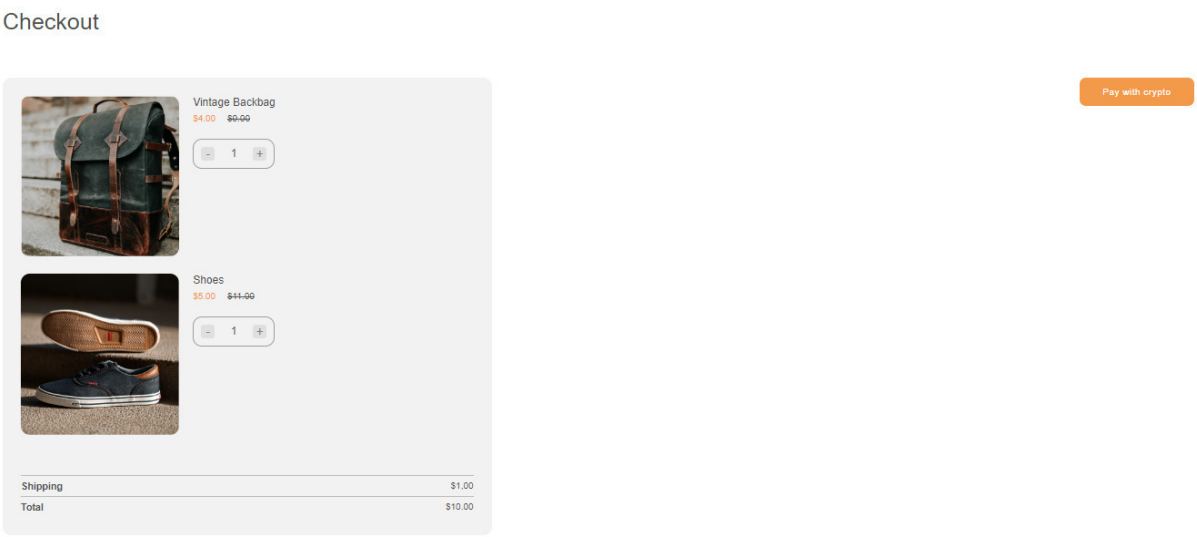

Abbildung 8: Screenshot - Shopseite vor der Zahlung

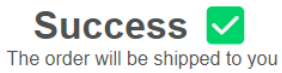

Abbildung 9: Screenshot - Seite des Händlers bei erfolgreicher Zahlung

Abbildung 10: Screenshot - Seite des Händlers bei fehlgeschlagener Zahlung

Bezahlseite

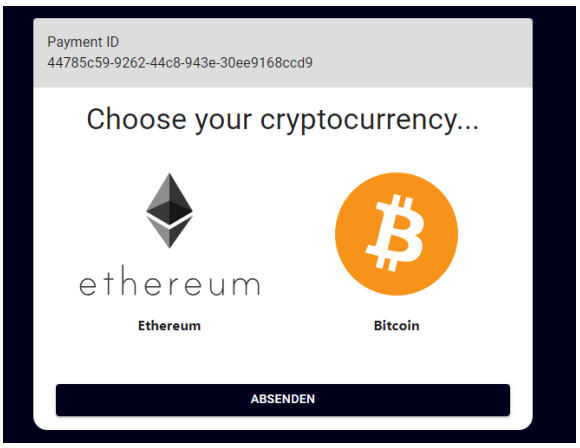

Abbildung 11: Screenshot - Auswahl Kryptowährung

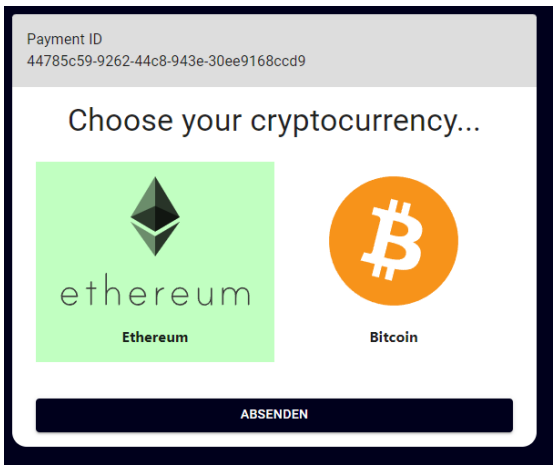

Abbildung 12: Screenshot - Kryptowährung ausgewählt

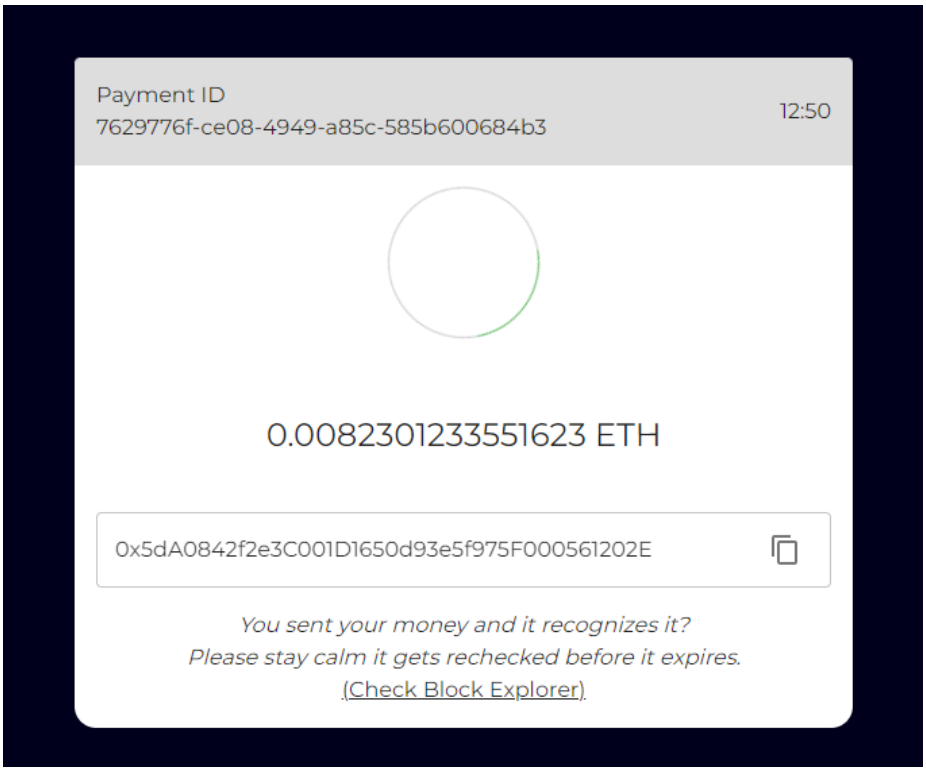

Abbildung 13: Screenshot - Warten auf die Zahlung

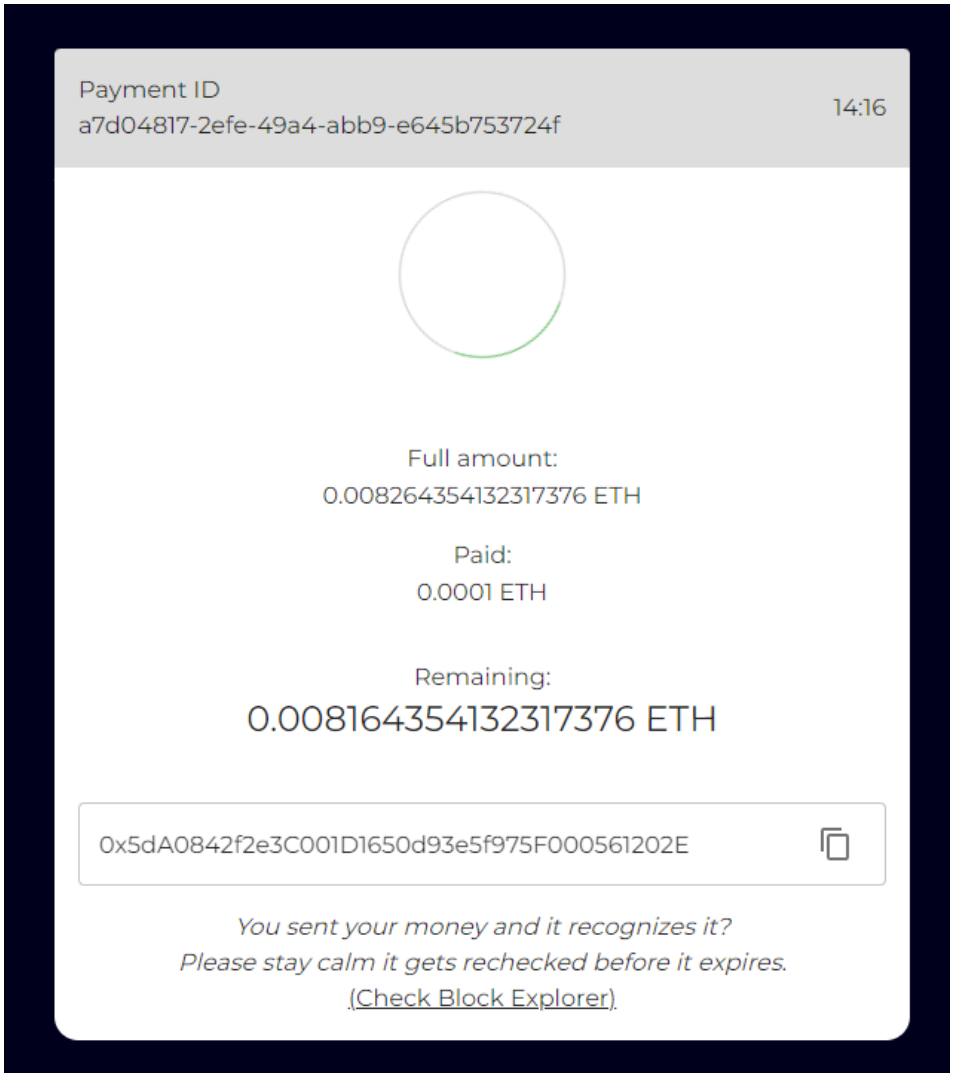

Abbildung 14: Screenshot - Warten auf den Restbetrag der Zahlung

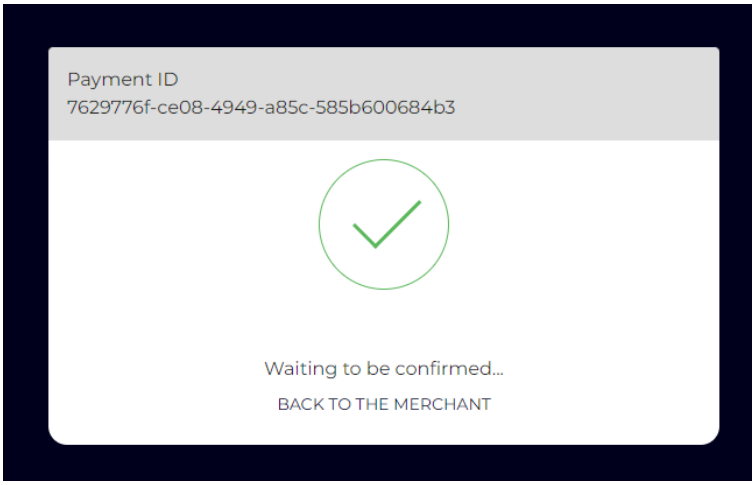

Abbildung 15: Screenshot - Zahlung erhalten

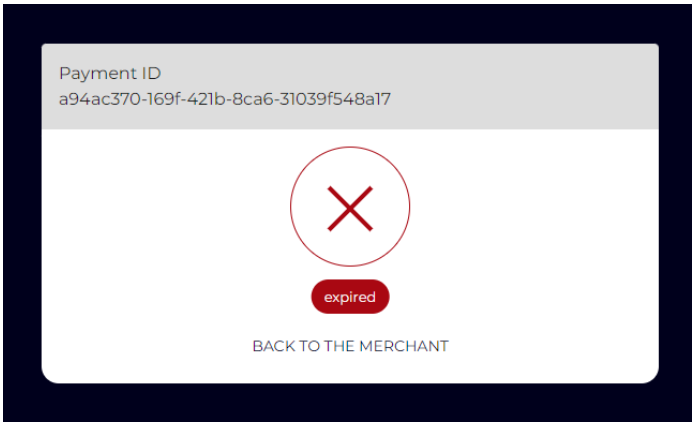

Abbildung 16: Screenshot - Zahlung abgelaufen

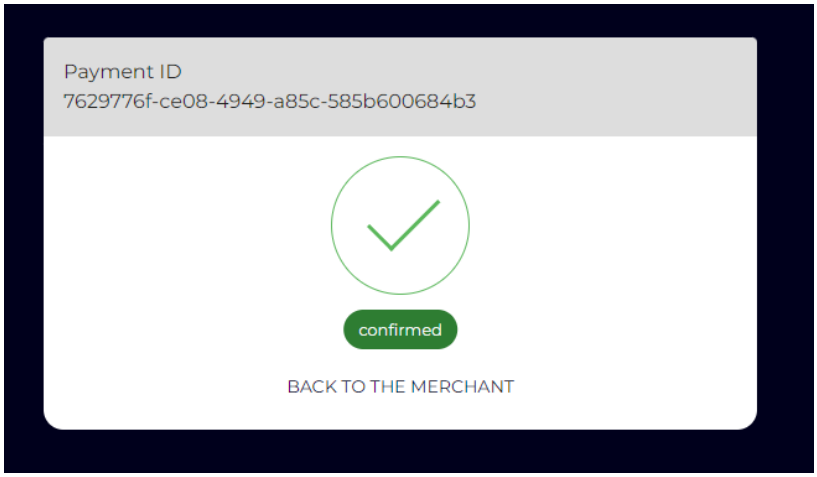

Abbildung 17: Screenshot - Zahlung bestätigt

F.1.1 Mainnet Test

Es wurde jeweils für Ethereum und Bitcoin eine Zahlung auf dem Mainnet ausgeführt, damit diese mindestens einmal getestet ist.

F.1.1.1 Ethereum

Über Bitpanda wurden 0.02 ETH überwiesen.

Blockexplorer URL von der CHainGate Adresse:
https://etherscan.io/address/0xb5c6c00d535ac4065f6dc157bdc707797375c4f9

Dort ist die Einzahlung, sowie die Auszahlung zu sehen.

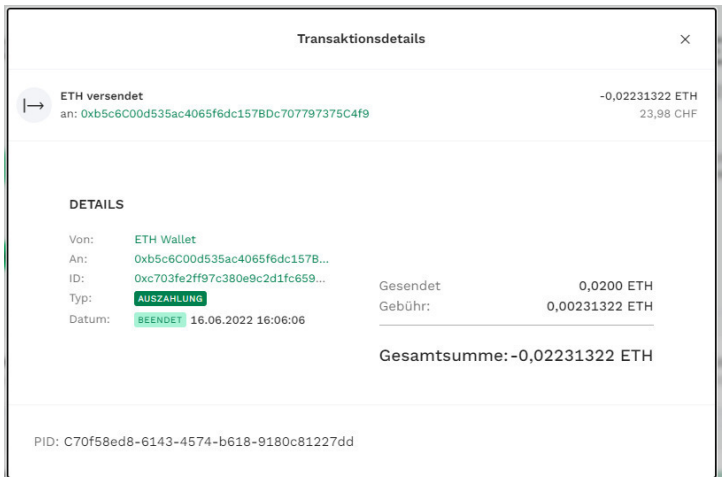

Abbildung 18: Bitpanda ETH Überweisung

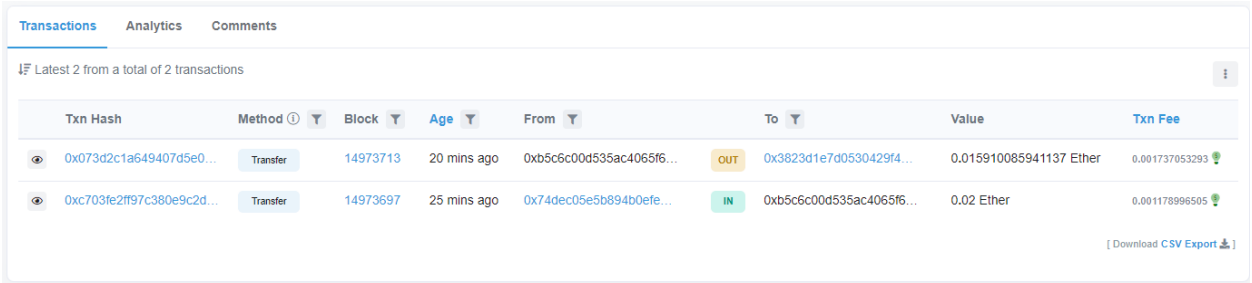

Abbildung 19: Etherscan Information

F.1.1.2 Bitcoin

Über Bitpanda wurden 0.00094116 BTC überwiesen.

Blockexplorer URL von der CHainGate Adresse:
https://www.blockchain.com/btc/address/bc1qvnrmgsr585ffl22ghx4wfmg09mpmn2l5nnnw4q

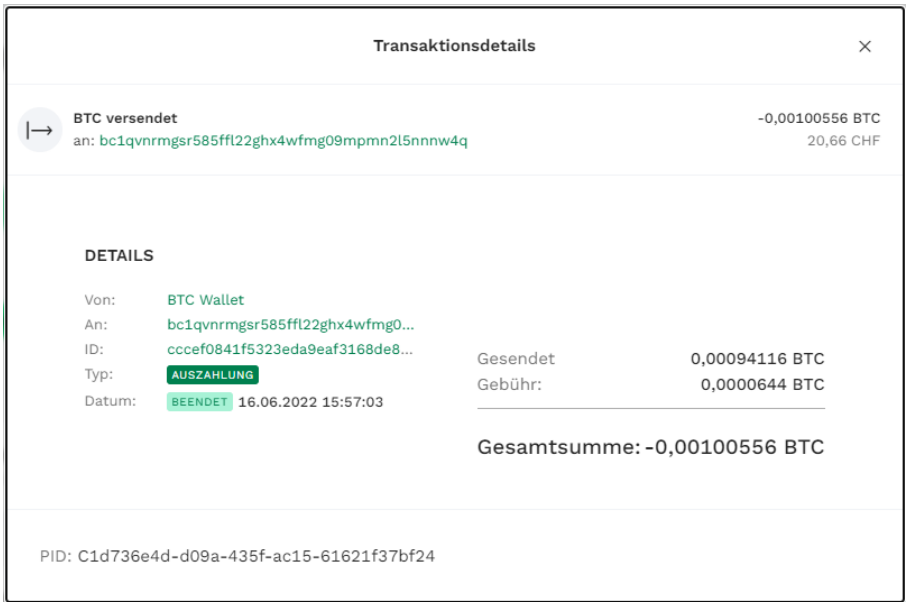

Abbildung 20: Bitpanda BTC Überweisung

Abbildung 21: Blockchain Explorer Information

G. Systemtest-Spezifikation

Vor dem Systemtest sollten die Daten auf der Datenbank gelöscht werden. Anschliessend kann CHainGate gestartet werden. Eine detaillierte Anleitung befindet sich bei der Betriebsdokumentation unter dem Kapitel «2. CHainGate starten». Wichtig ist, dass bei Bitcoin anstelle des Mainnets die Regtest Umgebung genutzt wird (siehe Punkt 2.2 Bitcoin Regtest verwenden).

Zusätzlich muss jeweils ein Wallet für den Händler und Käufer eingerichtet werden mit vorhandenem Guthaben.

Alle Testfälle sind in der Excel Datei «Systemtest-Spezifikation» aufgelistet.

H. Zeitrapport

In der Abbildung 22 ist der Aufwand pro Person zu sehen. Insgesamt wurden 763 Stunden und 20 Minuten aufgewendet.

Abbildung 22: Aufwand pro Person

In der Abbildung 23 Ist der Aufwand pro Person und Typ ersichtlich.

Abbildung 23: Aufwand pro Person und Typ

Die Rohdaten sind im Zeitrapport.xlsx

I. Abbildungsverzeichnis

J. Tabellenverzeichnis

K. Literaturverzeichnis

[1] H. Tremp, *ARCHITEKTUREN VERTEILTER SOFTWARESYSTEME: Soa & microservices - mehrschichten-architekturen -… anwendungsintegration,* 1st ed. [S.l.]: MORGAN KAUFMANN, 2021.

[2] A. Kharenko, "Monolithic vs. Microservices Architecture - Microservices Practitioner Articles," *Microservices Practitioner Articles*, 09 Oct., 2015. https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59 (accessed: Jun. 13 2022).

[3] microservices.io, *Microservices Pattern: Monolithic Architecture pattern.* [Online]. Available: https://microservices.io/patterns/monolithic.html (accessed: Jun. 13 2022).

[4] *SCS: Self-Contained Systems.* [Online]. Available: https://scs-architecture.org/ (accessed: Jun. 13 2022).

[5] microservices.io, *Microservices Pattern: Microservice Architecture pattern.* [Online]. Available: https://microservices.io/patterns/microservices.html (accessed: Jun. 13 2022).

[6] O'Reilly Online Learning, *Software Architecture Patterns.* [Online]. Available: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch03.html (accessed: Jun. 13 2022).

[7] Mammoth-AI, *5 software architecture patterns: How to make the right choice - Mammoth-AI.* [Online]. Available: https://www.mammoth-ai.com/5-software-architecture-patterns-how-to-make-the-right-choice/ (accessed: Jun. 13 2022).

[8] Alibaba Cloud Community, *What Is Microkernel Architecture Design?* [Online]. Available: https://www.alibabacloud.com/blog/what-is-microkernel-architecture-design_597605 (accessed: Jun. 13 2022).

[9] CoinMarketCap, *Cryptocurrency Prices, Charts And Market Capitalizations | CoinMarketCap.* [Online]. Available: https://coinmarketcap.com/ (accessed: Jun. 13 2022).

[10] Nishanil, *Communication in a microservice architecture.* [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture (accessed: Jun. 13 2022).

[11] J. Ratliff, "Home - Docker," *Docker*, 10 May., 2022. https://www.docker.com/ (accessed: Jun. 13 2022).

[12] Docker Documentation, *Overview of Docker Compose.* [Online]. Available: https://docs.docker.com/compose/ (accessed: Jun. 13 2022).

[13] *OpenAPI Specification - Version 3.0.3 | Swagger.* [Online]. Available: https://swagger.io/specification/ (accessed: Jun. 13 2022).

[14] GORM, *GORM.* [Online]. Available: https://gorm.io/index.html (accessed: Jun. 13 2022).