Kryptowährungen als Zahlungsmittel bei FlatFeeStack

Studienarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2021

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

Abstract

Die heutige IT-Industrie ist sich ohne Open-Source Software nicht mehr vorzustellen. In vielen Services steckt ein Teil Open-Source drin. Darum ist es umso wichtiger, dass diese Software regelmässig gewartet wird, damit keine Sicherheitslücken auftreten. Ein Beispiel dafür ist die Schwachstelle in Log4j, die in der ganzen Welt für Unruhe gesorgt hat. FlatFeeStack ist eine Plattform, die es vereinfacht Open-Source-Projekte zu unterstützen damit in Zukunft solche Probleme vermieden werden können.

Das Ziel dieser Arbeit ist es die bestehende Applikation, um eine Einzahlung in Kryptowährungen zu erweitern. FlatFeeStack wünscht sich, dass Einzahlungen in Ethereum, NEO und Tezos in Zukunft möglich sind.

Als erstes wurde die aktuelle Applikation analysiert und die bestehenden Prozesse diskutiert, um ein Verständnis für die Abläufe zu entwickeln. Ebenfalls wurde die einzusetzende Technologie unter die Lupe genommen, um die Möglichkeiten und Limiten der einzelnen Kryptowährungen kennenzulernen. Nach dem Ausarbeiten mehrerer Möglichkeiten für die Implementation, wurde die beste Option gewählt. Darauffolgend wurde das Halten und Auszahlen schrittweise um die Möglichkeit von Kryptowährungen erweitert. Zeitgleich wurde eine Währung nach der anderen angebunden.

Die Integration der Kryptowährungen als neues Zahlungsmittel, inklusive Deployment auf die Testumgebung, konnte erfolgreich abgeschlossen werden. Um die Implementation zu vereinfachen, wurde NOWPayments als Gateway für Kryptowährungen eingesetzt.

NOWPayments bietet noch gewisse Einschränkungen im Bereich der Integration an die Test Blockchains sowie der NEO3 Unterstützung. In Zukunft könnte NOWPayments durch eine alternative Lösung ersetzt werden, die Testnetze sowie NEO3 unterstützt.

Inhaltsverzeichnis

1. Einleitung und Übersicht

1.1 FlatFeeStack

Auf GitHub gibt es über 73 Millionen Entwickler und 200 Millionen Repositories [1]. Viele der Entwickler arbeiten unentgeltlich an Open-Source Projekten. Grosse sowie kleine Unternehmen setzen immer mehr Open-Source Tools ein, um ihr Geschäft voranzutreiben und Geld zu verdienen. Deshalb sollte es einen fairen und einfachen Weg geben, wie die Entwickler für ihren wertvollen Beitrag entlohnt werden können. Diese Aufgabe hat sich FlatFeeStack zu Herzen genommen und will mit ihrer Plattform das Unterstützen dieser Entwickler vereinfachen.

Benutzer können sich bei der Plattform anmelden und GitHub Projekte auf der FlatFeeStack Plattform favorisieren. In Zukunft soll es möglich sein, dass auch Git-Projekte unabhängig von GitHub favorisiert werden können. Zurzeit kann der Benutzer für eine Pauschalgebühr von 125.47 US-Dollar (USD) im Jahr oder 31.61 USD alle 3 Monate seine Lieblingsprojekte unterstützen. Den Autoren des Projektes wird basierend auf Ihrem Beitrag zum Code Geld zu Verfügung gestellt. Die Autoren, die nicht auf FlatFeeStack registriert sind, werden per E-Mail benachrichtig, dass sie ab dem Zeitpunkt der Registration bei der Verteilung des Guthabens mitberücksichtig werden.

1.2 Ausgangslage

FlatFeeStack bietet zum aktuellen Zeitpunkt die Möglichkeit von Einzahlungen in USD und Auszahlungen in der Kryptowährung Ethereum (ETH) an. Die bestehende Implementation besteht aus drei Grundphasen, die **Einzahlung**, die **Berechnung** und die **Auszahlung**.

Einzahlung

Eine Einzahlung kann jeweils jährlich zu 125.47 USD oder quartalsweise zu 31.61 USD getätigt werden. Die Beträge wurden so gewählt, dass abzüglich der Gebühren ein Tag 0.33 USD kostet. Die gesamte Zahlungsabwicklung wird von Stripe, einem Online-Bezahldienst, übernommen. FlatFeeStack wird über einen Web-Hook benachrichtig, ob die Einzahlung erfolgreich war oder nicht.

Berechnungen

Die Berechnungen welcher Entwickler wie viel Geld von welchem Sponsor erhält, wird täglich ausgeführt. Weil mit sehr kleinen Zahlen gerechnet wird und Berechnungen im Programmcode mit Fliesskommazahlen zu Problemen führen können, wurden die USD in Micro USD umgewandelt (1 USD = 1'000'000 Micro USD) [2].

Auszahlung

Weil die Auszahlung in ETH geschieht, muss zuerst das Guthaben von USD in ETH umgewandelt werden. Dies passiert über einen manuellen Prozess. Das Geld wird vom Bankkonto auf eine Handelsplattform für Kryptowährungen überwiesen und umgewandelt. Für die Auszahlung über ETH wird ein Smart Contract verwendet. Dieser wird täglich aufgerufen und mit den Informationen abgefüllt, welcher Entwickler wie viel ETH erhält. Der tägliche Job überweisst das Geld noch nicht, sondern der Benutzer kann zu einem

späteren Zeitpunkt selbst entscheiden, wann er das Geld auf sein Wallet überweisen will. Dieser Prozess wird über die Browser Extension MetaMask gestartet [3].

In der Grafik «Abbildung 1: Ausgangslage» wird der gesamte Ablauf über alle drei Phasen inklusive Zwischenschritten der Ausgangslage dargestellt. Weitere Details zur Architektur und Design sehen sie im Kapitel 3. Planung und Vorbereitung.

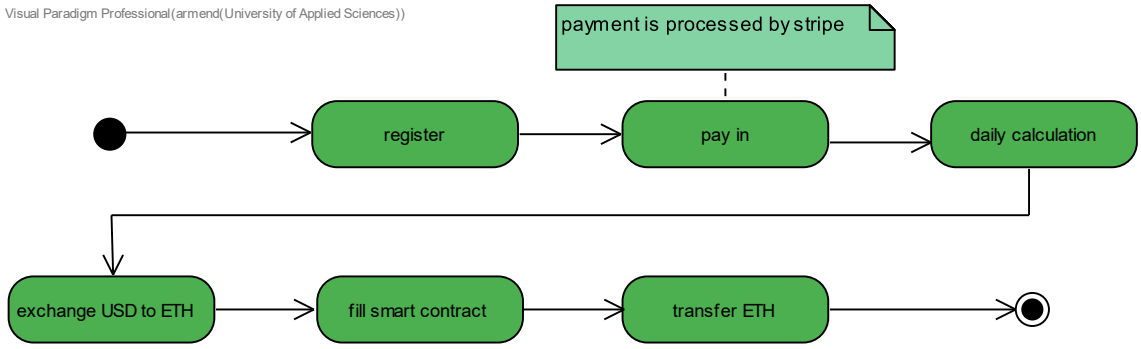


Abbildung 1: Ausgangslage

Zusätzlich existieren drei verschiedene Rollen, welche ein Benutzer annehmen kann. Dabei kann ein Benutzer die Rolle **Sponsor** und **Contributor** gleichzeitig annehmen.

- Rolle **Sponsor:** Ein Sponsor tätigt Einzahlungen und favorisiert seine Lieblingsprojekte.
- Rolle **Contributor:** Ein Contributor hat seine GitHub Adresse registriert und erhält durch sein Mitwirken an Open-Source Projekten Geld.
- Rolle **Organisation:** Eine Organisation ist ein Unternehmen oder ein Team mit mehreren Mitarbeitern. Eine Organisation kann für ihre Mitarbeiter die Sponsor Kosten übernehmen. Dabei kann sie Seats (Sponsor Abonnemente für ihre Mitarbeiter) kaufen und diese den Mitarbeitern zur Verfügung stellen. Eine Organisation kann keine Projekte favorisieren.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist es die bestehende Applikation, um eine Einzahlung in Kryptowährungen zu erweitern. FlatFeeStack wünscht sich eine Integration der Kryptowährungen Ethereum, NEO und Tezos. Dabei ist es wichtig, dass die Lösung die Transaktionskosten geringhält und simple umgesetzt wird. Ebenfalls soll die Implementierung für zukünftige Kryptowährungen erweiterbar sein.

2. Theorie

Die relevante theoretische Grundlage für diese Arbeit wird in diesem Kapitel aufbereitet. Das Konzept einer Blockchain und die Verwendung von Smart Contracts wird erklärt. Ausserdem wird auf die unterschiedlichen Technologien eingegangen.

2.1 Einführung Blockchain

Eine Blockchain ist eine verteilte Datenbank, welche es ermöglicht diversen unabhängigen Nutzern Daten fälschungssicher in das System zu schreiben. Eine wichtige Eigenschaft der Blockchain ist die Unveränderbarkeit der vergangenen Einträge, auch Blöcke genannt. Dies wird erreicht, indem jeder neue Block auf einen vergangenen referenziert und dadurch eine Kette bildet.

Um den Geldtransfer möglichst autonom und einfach zu halten, setzt FlatFeeStack auf diverse Blockchains. Diese haben den Vorteil, dass man unabhängig einer Drittperson Geld senden kann. Ebenfalls ermöglichen Blockchains eine hohe Transparenz durch Smart Contracts. FlatFeeStack nutzt einen Smart Contract für das Halten des Guthabens und die Auszahlungen. Dieser Smart Contract ist für alle sichtbar und daher Transparent für die Benutzer. Dadurch kann ein Contributor bereits im Vorhinein wissen wie viel Geld ihm zusteht, unabhängig von der ganzen FlatFeeStack-Applikation.

Neben der Blockchain selbst bieten auch die Accounts auf der Blockchain weitere Vorteile im Vergleich zu den herkömmlichen Bankkonten. Nicht jeder besitzt ein Bankkonto auf der Welt. In manchen Ländern gibt es Beschränkungen von wo und wohin welches Geld gesendet werden darf. Ebenfalls kann ein Konto in vielen Ländern nur mit ID oder einem Pass eröffnet werden. Auf der Blockchain sind solche Restriktionen fremd. Ebenfalls bietet dies den Contributoren zusätzlich eine gewisse Anonymität. Falls ein User ein eher kontroverses Projekt verwaltet und nicht sein richtiges Bankkonto angeben will, um seine wahre Identität nicht preisgeben zu müssen, sind Blockchains ideal.

Bei FlatFeeStack ist Umwelt ein wichtiges Thema, darum werden für die Umsetzung nur Proof-of-Stake-Blockchains verwendet. Als Alternative gibt es die bekanntere Proof-of-Work Variante.

Neben dem Konsensalgorithmus wurde auch Wert daraufgelegt, dass alle eingesetzten Blockchains Smart Contracts besitzen.

Im Folgenden werden die erwähnten Blockchains und Technologien kurz vorgestellt.

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. Dabei ist zu beachten, dass diese Ressource verteilt sein muss, um die Unabhängigkeit der Blockchain zu gewährleisten. 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.

2.3 Proof-of-Work (PoW)

Proof-of-Work ist der bekannteste Ansatz und wurde vor allem durch Bitcoin bekannt. Dieser setzt darauf, dass alle Teilnehmer versuchen ein Puzzle zu lösen. In dem Fall von Bitcoin ist es das Finden eines Hash mit einer Anzahl führenden «0». Die Anzahl «0» variiert hierbei, basierend auf der Kapazität des Netzes. Falls es jemandem gelingt das Rätsel zu lösen, wird er dafür mit der Währung selbst belohnt. Durch den starken Preisanstieg von Bitcoin hat sich ein Konkurrenzkampf entwickelt das Rätsel zu lösen und somit die Prämie zu erhalten. Der derzeitige Gewinn für das Knacken des 10-minütigen Rätsels liegt heute umgerechnet bei etwa 300'000 USD [4]. Weil das Lösen des Rätsels viel Rechenleistung benötigt, wird der Stromverbrauch Ende 2021 auf 200 Terawattstunden geschätzt. Das ist etwa so viel wie der Stromverbrauch für Thailand [5].

Da das Lösen im Grundsatz nur ein Mechanismus ist, um ein Konsens zu erreichen und die Generierung der vielen Hashes nutzlos für die Gesellschaft selbst ist, kam der Proof-of-Work-Ansatz in den letzten Jahren in die Kritik.

2.4 Proof-of-Stake (PoS)

Der Proof-of-Stake-Ansatz wurde im Jahre 2012 von Scott Nadal und Sunny King entwickelt, um die Probleme von Proof-of-Work zu eliminieren [6]. Der Ansatz verfolgt den Plan anstatt alle um ein Rätsel konkurrenzieren zu lassen, hat man eine Gruppe Validatoren, welche den neuen Block validieren. Um ein Validator zu werden, müssen diese einen Teil Ihrer Coins ans Netzwerk binden. Wenn die Validatoren richtig validieren, werden sie mit der Kryptowährung belohnt. Bei böswilligem Verhalten wird hingegen ein Teil oder alle ihre Coins, welche sie gebunden haben abgezogen.

Eine 51%-Attacke kann in diesem Fall nicht direkt verhindert werden. Jedoch wird damit gerechnet, dass die Währung selbst an Wert verliert, wenn eine solche Attacke bekannt wird. Das hat zur Folge, dass die zusätzlich erbeuteten Coins weniger wert sind. Im Endeffekt verliert der Angreifer mehr Geld als er dadurch gewinnt. Durch das Wegfallen des Wettbewerbs sinkt der Stromverbrauch gewaltig, jedoch ist das Konzept noch relativ neu und noch nicht so stark erprobt wie Proof-of-Work [7].

2.5 Smart Contract

Ein Smart Contract ist ein programmierbarer Vertrag auf einer Blockchain. Er bietet eine ähnliche Funktionalität wie ein handelsüblicher Vertrag. Der grosse Unterschied ist, dass ein Smart Contract keine Drittinstanz benötigt, um das Vertrauen zu etablieren. Das Vertrauen wird geschaffen durch die Eigenschaften der Blockchain selbst und der programmierten Logik des Vertrages. Jeder hat die Möglichkeit den Vertrag selbst anzuschauen und die Logik zu untersuchen.

Ein Vertrag ist abhängig von der Blockchain auf der er implementiert ist. Alle Contracts besitzen jedoch Methoden und Werte. Durch die Methoden kann mit den Werten auf dem Smart Contract interagiert

werden. Die Methoden des Smart Contracts können von jeder Person aufgerufen werden, jedoch können die Methoden durch Abfrage des Aufrufers abgesichert werden. Dadurch wird sichergestellt das wichtige Funktionen, wie das Übertragen des Besitzers an eine neue Adresse, nur vom jeweiligen Besitzer gemacht werden.

Eine weitere wichtige Eigenschaft eines Vertrages ist das Empfangen, Halten und Senden von Kryptowährung. Wichtig ist, dass das Schreiben auf die Blockchain und das Aufrufen von Funktionen immer einen kleinen Betrag kosten, um das Netzwerk nicht zu überlasten mit unsinnigen Anfragen.

2.6 Ethereum

Ethereum ist, nach Bitcoin, die zweit grösste Blockchain auf der Welt basierend auf der Marktkapitalisierung [8]. Die einzelnen Coins werden Ether (ETH) genannt. Die Blockchain besitzt neben der Ether-Währung auch Wei's. 1 Wei entspricht 10^{-18} ETH. Sie werden benötigt, um die Kosten für Smart Contracts zu berechnen. Pro Sekunde werden etwa 13 Transaktionen ausgeführt [9].

Im Vergleich zu den übrigen Kryptowährungen bei FlatFeeStack basiert Ethereum 1.0 noch auf Proof-of-Work. Mit dem Upgrade auf Ethereum 2.0 soll die Umstellung auf Proof-of-Stake gelingen [10].

2.6.1 Smart Contract

Ethereum war einer der ersten Blockchains, welche es ermöglichte Smart Contracts auf eine Blockchain zu deployen [11]. Ein Smart Contract in Ethereum wird normalerweise in Solidity programmiert. Solidity wurde speziell für die Smart Contract Entwicklung auf Ethereum entworfen und erinnert an JavaScript von der Syntax her.

2.7 NEO

NEO ist unter den Top 100 der grössten Blockchains vertreten. NEO hat sich als Ziel, die Digitalisierung von Assets gesetzt. Dadurch soll zum Beispiel der Besitz sowie der Transfer von Eigentum verwaltet werden. Neben der Währung NEO gibt es auch noch GAS. GAS wird für das Validieren von Transaktionen ausgeschüttet und bei Arbeiten, wie das Verarbeiten eines Smart Contracts, erhoben. NEO selbst unterstützt bis zu 10'000 Transaktionen pro Sekunde. Anders als bei Ethereum ist ein NEO nicht teilbar und es wurden bereits alle NEOs gemined. Die eine Hälfte wurde zu Beginn an die Investoren verteilt und von der anderen Hälfte werden jedes Jahr 15'000'000 NEO's durch einen Smart Contract an die Entwickler von NEO gesendet [12] [13].

Zurzeit ist die Ablösung von NEO Legacy durch NEO3 im Gange. Die Legacy-Variante wird etwa noch 1 Jahr unterstützt.

2.7.1 Smart Contracts

NEO bietet im Vergleich zu Ethereum die Möglichkeit seinen Smart Contract in diversen Sprachen zu implementieren. Die besten unterstützen Sprachen sind: C#, Go, Python, Java oder TypeScript [14]. Seit

NEO3 kann beim Erstellen eines Smart Contracts genauer spezifiziert werden was erlaubt ist. Beispielweise kann definiert werden, welche anderen Contracts vom eigenen Contract aufgerufen werden dürfen [15].

2.8 Tezos

Tezos ist eine junge Blockchain und ist erst seit 2018 auf dem Markt. Die einzelnen Coins werden XTZ genannt. Die Einführung von Tezos war für damalige Verhältnisse das zweitgrösste ICO (Initial Coin Offering) der Geschichte in der Krypto Welt mit 232 Millionen US-Dollar. Tezos versucht eine sichere und stets aktuelle Blockchain zu sein. Das versuchen sie zum Beispiel indem sie auf Hard-Forks verzichten. Ein Hard-Fork ist, wenn eine neue Version der Blockchain nicht mehr rückwärtskompatibel ist und so in zwei Teile aufgeteilt wird, oder die alte Version der Blockchain stirbt. Durch den Verzicht eines Hard-Forks soll einem Tezos-nutzer langfristig eine Blockchain zur Verfügung gestellt werden. Tezos besitzt ähnlich wie andere Kryptowährungen Miner, jedoch werden diese bei Tezos «Baker» genannt und die Tokens werden «gebacken». Ein grosser Unterschied von den Bakern bei Tezos ist, dass ein Einstieg als Baker einen Mindestbetrag von 10'000 XTZ benötigt [16].

2.8.1 Smart Contracts

Die Smart Contracts in Tezos gelten als besonders sicher, da sie durch einen mathematischen Prozess geprüft werden. In Tezos kann wie bei NEO ein Smart Contract in mehreren High-Level-Sprachen entwickelt werden. Diese werden jedoch nach dem Implementieren in die gemeinsame Sprache Michelson kompiliert. Michelson ist eine Low-Level-Sprache und operiert auf einem Stack [17, 18]. Ein weiterer Unterschied zu Ethereum und NEO ist, dass bei Tezos ein Smart Contract nicht direkt über eine Überweisung aufgeladen werden kann, sondern eine Funktion genutzt werden muss.

3. Planung und Vorbereitung

In diesem Kapitel geht es primär um das Design und die Architektur des Ist-Zustandes. Da FlatFeeStack erweitert wird und nicht komplett neu geschrieben werden muss, ist es zwingend erforderlich die bestehende Architektur und das Design zu verstehen. Sobald die Ausgangslage geklärt ist, können die neuen Anforderungen aufgenommen werden und basierend auf diesen kann die Architektur und das Design erweitert werden.

3.1 Ist-Zustand

3.1.1 Architektur

In diesem Abschnitt wird die Architektur der Applikation und die Ausgangslage vorgestellt. Die einzelnen Komponenten werden erklärt und ihre einzelnen Aufgaben werden beschrieben.

FlatFeeStack ist eine Web-Applikation, die auf Microservices basiert, welche in einem Docker Container laufen. Zusätzlich werden externe Services wie Stripe für den Zahlungsverkehr oder Ethereum für die Auszahlung benutzt.

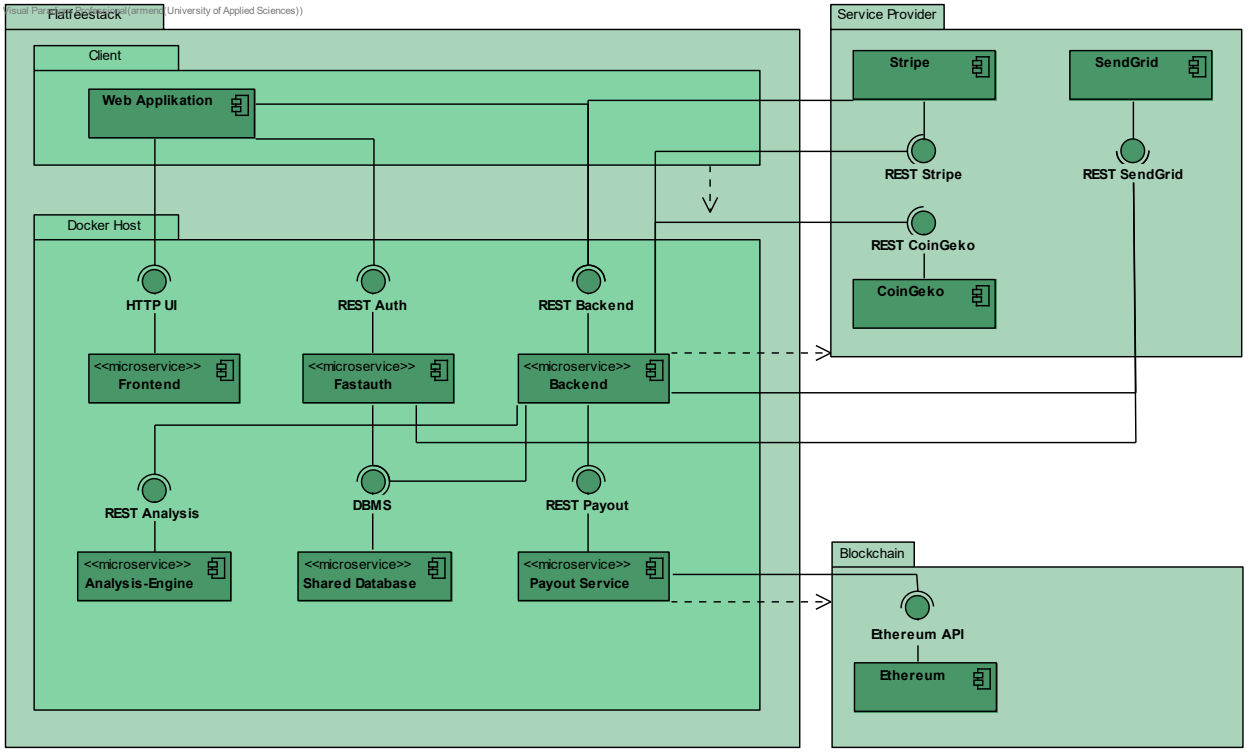


Abbildung 2: Architektur Ist-Zustand

Web-Applikation

Die Web-Applikation läuft auf allen Clients, welche einen Web-Browser installiert haben und JavaScript ausführen können.

Frontend

Das Frontend ist eine Single Page Applikation, welche auf dem Webframework Svelte [19] basiert. Es werden API-Calls verwendet, um Informationen aus dem Backend oder Fastauth Service zu erhalten. Zusätzlich werden WebSockets verwendet, um Informationen vom Server direkt an den Client zu senden, ohne eine Anfrage vom Frontend an das Backend.

Das Frontend ist auf der Produktions-, Test- und Entwicklungsumgebung gleich. Die unterschiedlichen Variablen und Konfigurationen werden über einen Backend Endpunkt verwaltet, welcher die Konfiguration an das Frontend sendet.

Fastauth

Die Authentifizierung der Benutzer geschieht über den Fastauth Service. Für die Autorisierung kommt ein JSON Web Token (JWT) zum Einsatz. Dieser Token wird verwendet, um Informationen sicher vom Frontend zum Backend zu senden. Die Informationen sind signiert und können deshalb überprüft werden. Sobald sich ein Benutzer einloggt, wird ein Token für ihn vom Backend erstellt, mit den nötigen Angaben (Ablaufdatum, Rolle etc.) und mit einem sicheren Schlüssel signiert. Danach sendet der Browser bei jeder Anfrage diesen Token mit und das Backend kann die Signatur überprüfen, um sicher zu stellen das die Informationen nicht manipuliert wurden [20].

Backend

Im Backend ist der grösste Teil der Logik hinterlegt. Er dient als Zentrale, welche mit den anderen Microservices wie Payout, Datenbank und Analysis-Engine interagiert. Auch die Anbindung und Orchestrierung der externen Services wie Stripe und CoinGecko werden von hier gesteuert.

Analysis-Engine

Als Grundlage für die Auszahlungen an die Contributoren, werden Metriken wie die Anzahl der Commits und Lines of Code analysiert. Die ganze Berechnung und Analyse, welcher Contributor wie viel an einem Projekt beigetragen hat, wird von diesem Service übernommen.

Payout

Der Payout Service wird benutzt, um den Contributoren ihren Anteil zu überweisen. Hierzu wird das Ethereum Netzwerk genutzt, um ETH zu senden. Es wurde sich aktiv gegen USD als Auszahlungswährung entschieden, da der Aufwand über Stripe allen möglichen Contributoren Geld zu senden viel höher ist als über Ethereum.

Stripe

Stripe wird für die Zahlungsabwicklung der Einzahlungen in USD verwendet. Es kann mit einer VISA Kreditkarte einbezahlt werden und FlatFeeStack kann jährlich das Geld automatisiert vom Sponsor abbuchen, damit das Abo automatisch verlängert wird. Die automatische Verlängerung kann von jedem Sponsor jederzeit aufgehoben werden. [21]

SendGrid

Der Service von SendGrid wird verwendet, um E-Mails an die Benutzer zu senden. Die E-Mails werden bei der Registration für die Bestätigung genutzt oder um Benutzer zu Informieren. [22]

CoinGecko

CoinGecko wird verwendet, um den aktuellen Wechselkurs zwischen USD und ETH abzufragen. Bei einer Auszahlung muss der Wechselkurs, welcher bei der Umwandlung von USD zu ETH genutzt wurde, eingegeben werden. Der Wert von CoinGecko wird genutzt, um einen Standardwert abzufüllen.

Ethereum

Die Ethereum Blockchain wird für die Auszahlungen von ETH an die Contributoren genutzt.

Shared Database

Es gibt zurzeit eine PostgreSQL Datenbank, welche aktuell vom Backend und Fastauth Service genutzt wird, um die Daten zu persistieren.

3.1.2 Datenbank

Das Datenbank Modell ist für jede Applikation essenziell. In der Grafik «Abbildung 3: Datenbank UML Ist-Zustand» wird das UML Diagramm der aktuellen Datenbank vorgestellt. Nach der Abbildung «Abbildung 3: Datenbank UML Ist-Zustand» werden die wichtigsten Tabellen und Relationen beschrieben.

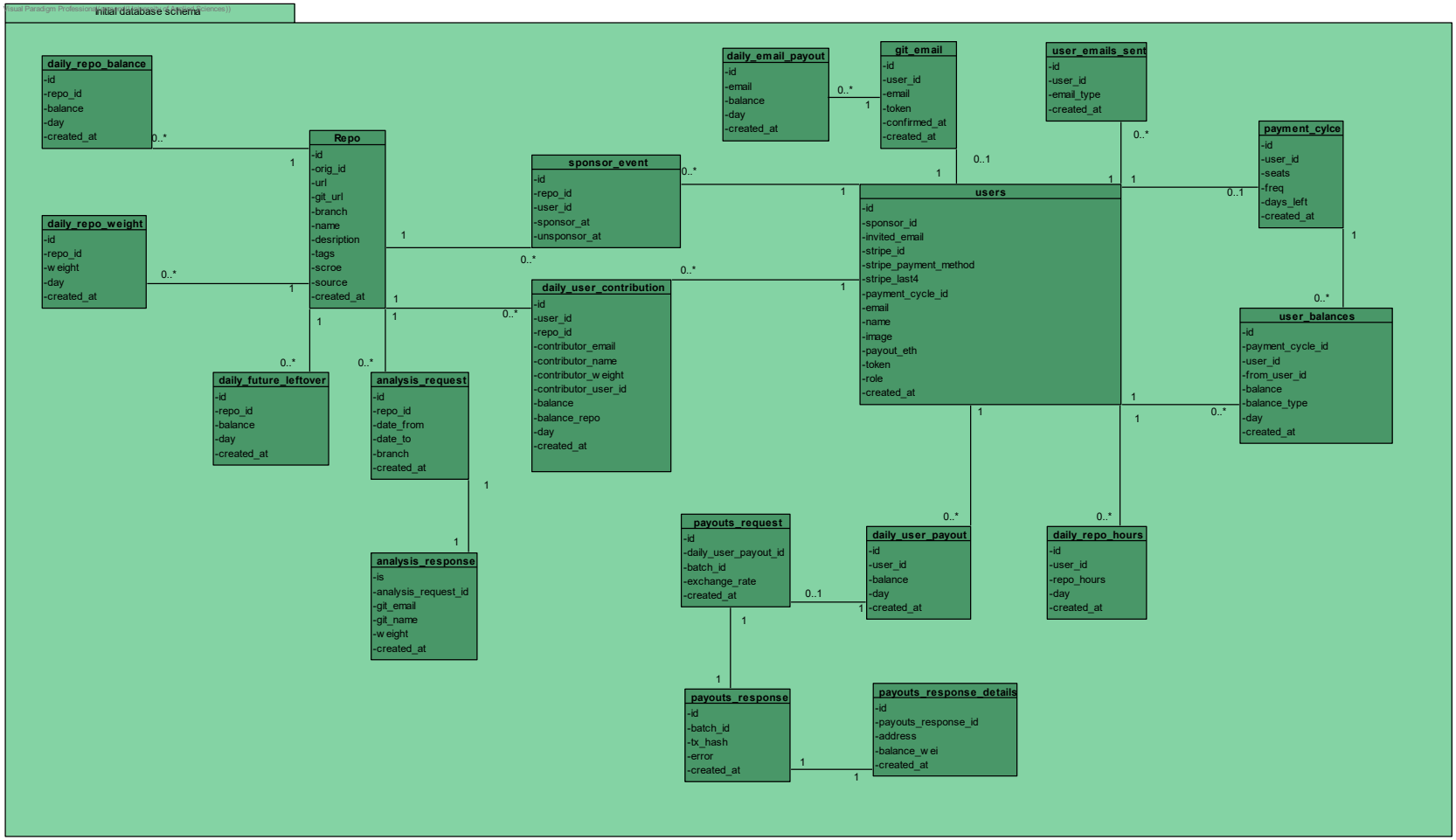


Abbildung 3: Datenbank UML Ist-Zustand

Tabelle users

In der Tabelle users, werden alle Informationen zu einem Benutzer abgelegt wie Name, E-Mail, Rolle und viele mehr. Der Benutzer spielt eine zentrale Rolle, deshalb besitzt die Tabelle users viele Relationen. Eine wichtige Information ist der aktuelle Payment Cycle.

Tabelle repo

Die Tabelle repo bildet die jeweiligen GitHub Repositories ab, welche auf FlatFeeStack favorisiert werden können.

Tabelle payment_cylce

Der Payment Cycle spiegelt das Abonnement eines Sponsors wider. Dabei wird bei jeder neuen Einzahlung ein neuer Payment Cycle erstellt. Hier werden wichtige Informationen gespeichert wie, für wie viele Seats bezahlt wurde, welches Abonnement gelöst wurde und die Restdauer. Diese Tabelle wird auch als History genutzt, um zu sehen, wann welcher Sponsor bezahlt hat. Zu einem Payment Cycle gehört immer eine User Balance. Weitere Details zum User Balance finden Sie im nächsten Abschnitt.

Tabelle user_balances

In der Tabelle user_balances wird das gesamte Guthaben von einem Sponsor dokumentiert. Zahlt er ein, gibt es einen `PAYMENT` Eintrag. Sobald er ein Projekt sponsort, wird im täglich ein Betrag abgezogen. Für den täglichen Abzug gibt es dann jeweils einen `PAY` Eintrag. Alle Balances gehören immer zu einen Payment Cycle. Wird ein neuer Payment Cycle erstellt, wird das alte Guthaben auf den neuen Payment Cycle übertragen.

Tabelle sponsor_event

Die Tabelle sponsor_event wird genutzt, um zu wissen, welcher Sponsor welches Projekt favorisiert hat. Diese Information wird benötigt, um zu wissen welche Projekte am Tag der Berechnung berücksichtig werden müssen. Sobald ein Projekt mehr wie eine Stunde favorisiert wurde, wird es berücksichtig und erhält seinen Anteil vom Guthaben.

Tabelle analysis_request und analysis_respone

Die Informationen welcher Contributor wie viel zu einem Projekt beigetragen hat, wird von der Analysis-Engine bereitgestellt. Die Daten werden vom Backend über eine REST API abgefragt und in die Datenbank abgespeichert. Dieser Job wird täglich von einem Cronjob ausgeführt, jedoch werden neue Daten erst abgefragt, wenn die Informationen auf der Datenbank älter als fünf Tage sind.

Alle daily Tabellen

Alle daily Tabellen werden für die täglichen Berechnungen benötigt, um herauszufinden, welcher Contributor wie viel Geld erhält. Genauere Erläuterungen dazu, gibt es im Abschnitt 3.1.4 Tägliche Berechnungen.

3.1.3 Einzahlung

Der Einzahlungsprozess ist der erste Schritt dieser Arbeit. Daher ist es wichtig diesen gut zu verstehen. In der Abbildung «Abbildung 4: Einzahlung Sequenzdiagramm Ist-Zustand», sieht man alle Aktoren, die Komponenten und wie der Prozess abläuft.

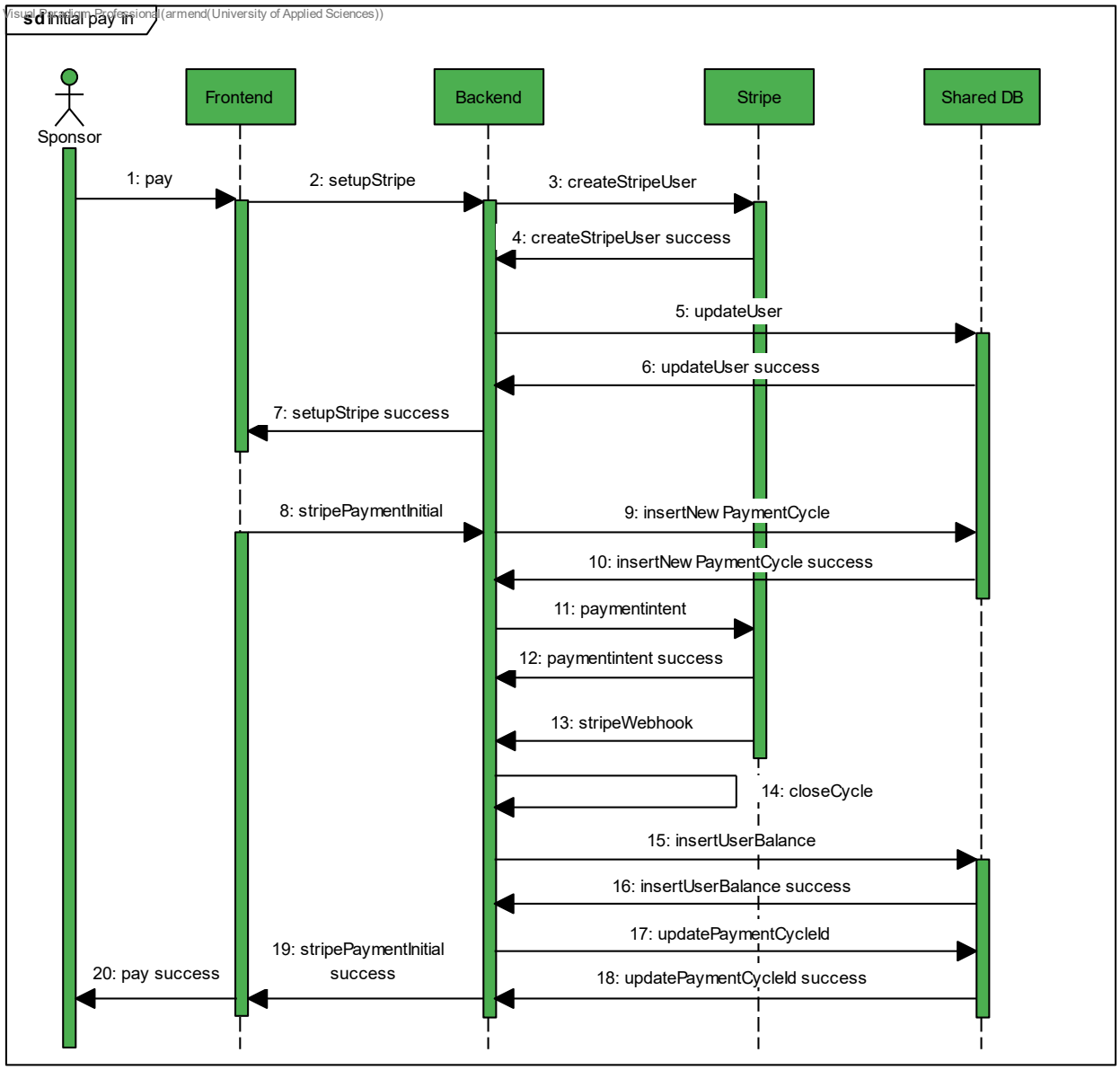


Abbildung 4: Einzahlung Sequenzdiagramm Ist-Zustand

1: pay

Ein Sponsor ist auf der Website von FlatFeeStack und möchte gerne eine Einzahlung tätigen. Hierbei kann er wählen, ob er jährlich bezahlen will oder quartalsweise. Hat er sich für eine Option entschieden, gibt er seine Kreditkarteninformationen ein und klickt auf den «Support» Button.

2-7: setupStripe

Die Schritte 2 bis 7 werden nur ausgeführt, wenn der Sponsor das erste Mal eine Einzahlung tätigt und noch kein Stripe Benutzer erstellt wurde. Dabei wird über die Stripe API ein neuer Benutzer bei Stripe erstellt und die erhaltenen Informationen werden in der Datenbank abgespeichert.

8-12: stripePaymentInitial

Zuerst wird ein neuer Payment Cycle erstellt und danach wird über Stripe eine Einzahlung ausgelöst.

13-20: stripeWebhook

Sobald die Zahlung von Stripe verarbeitet wurde, wird FlatFeeStack über einen WebHook benachrichtigt. War die Einzahlung erfolgreich, wird der alte Payment Cycle geschlossen, das aktuelle Guthaben auf den neuen Payment Cycle übertragen und die neue Einzahlung hinzugefügt. Zum Schluss wird der aktive Payment Cycle beim Benutzer aktualisiert.

3.1.4 Tägliche Berechnungen

Es werden täglich 10 Berechnungen durchgeführt, um herauszufinden wie viel Geld welcher Contributor erhält. Diese Jobs bauen teilweise aufeinander auf, daher ist die Reihenfolge wichtig. Nach der Grafik «Abbildung 5: Tägliche Berechnungen Ist-Zustand» werden alle Berechnungen kurz erläutert.

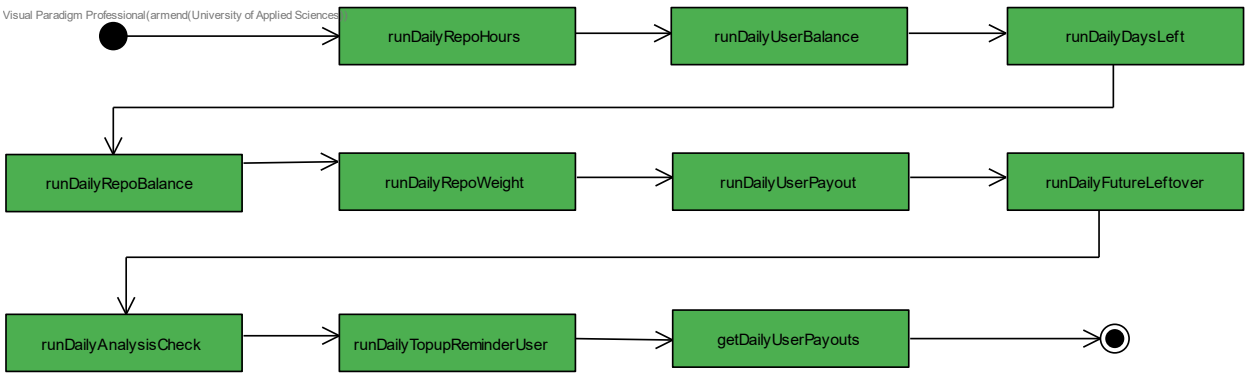


Abbildung 5: Tägliche Berechnungen Ist-Zustand

runDailyRepoHours

Hier wird festgehalten welcher Sponsor wie viele Stunden pro Tag supportet hat. Supportet er mehr als eine Stunde pro Tag, wird der volle Tagessatz abgezogen. Die Summe der Stunden, werden gebraucht, um später herauszufinden welches Repository welchen Anteil vom Tagessatz erhält. Somit hat der Sponsor die Möglichkeit stündlich seine favorisierten Projekte zu ändern und sie erhalten auf Basis der Anzahl Stunden, ihren prozentuellen Anteil.

runDailyUserBalance

Der Tagessatz wird vom Sponsor täglich abgezogen, wenn er ein Projekt mehr als eine Stunde an diesem Tag favorisiert hat. Über einen «unique index» in der Datenbank wird sichergestellt, dass der Tagessatz maximal einmal pro Tag abgezogen werden kann.

runDailyDaysLeft

Der Payment Cycle enthält die Information, wie viele Tage der Benutzer noch ein Sponsor ist. Dieser Wert wird täglich neu berechnet und aktualisiert. Hierbei wird das gesamte Guthaben durch den Tagessatz geteilt.

runDailyRepoBalance

Bevor berechnet werden kann, welcher Contributor wie viel Geld erhält, muss zuerst das Geld auf die einzelnen Projekte verteilt werden. Der Betrag pro Sponsor wird auf seine Projekte auf Basis der Anzahl favorisierten Stunden verteilt. Wurde zum Beispiel «Projekt A» 24 Stunden favorisiert und «Projekt B» 12 Stunden, erhält Projekt A $^2/_3$ und Projekt B $^1/_3$ vom Tagessatz.

runDailyRepoWeight

Die Daten, welcher Contributor wie viel an einem Projekt auf GitHub geleistet hat, werden von der Analysis-Engine aufbereitet. Bei den Berechnungen werden jedoch nur Contributor berücksichtig, welche auf FlatFeeStack registriert sind. Deshalb ist es nötig zu wissen wie viele der GitHub Contributoren pro Projekt auf FlatFeeStack registriert sind. Dieser Wert wird hier als Prozentsatz ausgerechnet.

runDailyUserPayout

In diesem Schritt wird das Guthaben von den Projekten auf die einzelnen Contributoren verteilt. Hierzu werden die Daten aus der Analysis-Engine zur Hilfe genommen, um das Guthaben fair zu verteilen.

runDailyFutureLeftover

Ist für ein Projekt kein Contributor auf FlatFeeStack registriert, landet das Geld in einem Restetopf. Sobald sich einer der Contributoren auf FlatFeeStack registriert, wird der Restetopf bei der runDailyUserPayout Berechnung berücksichtig.

runDailyAnalysisCheck

Sind die Daten der Analysis-Engine älter als fünf Tage, wird eine Anfrage an die Analysis-Engine geschickt, dabei werden die Daten neu berechnet und aktualisiert.

runDailyTopupReminderUser

Hier werden die Sponsoren benachrichtig, welche nur noch ein Guthaben für einen Tag besitzen. Sie werden freundlich erinnert, dass wenn sie keine weitere Einzahlung tätigen, ihr Sponsoring abläuft.

getDailyUserPayouts

Contributor selbst können auch andere Projekte als Sponsor unterstützen. Deshalb wird das erhaltene Geld als Contributor zum Sponsoring Guthaben gutgeschrieben. So können Contributoren, ohne selbst eine Einzahlung zu tätigen auch ihre Lieblingsprojekte unterstützen.

3.1.5 Auszahlungen

Die Auszahlungen geschehen in Ethereum. Weil alle Einzahlungen in USD sind, müssen die USD in Ethereum umgewandelt werden. Dieser Schritt wird manuell über eine Handelsplattform, wie Kraken oder Bitstamp durchgeführt. Da dieser Schritt nicht automatisiert ist, muss auch die Auszahlung manuell über ein Admin Interface getätigt werden. Wichtig ist, dass der richtige USD zu ETH Wechselkurs angegeben wird, damit die Umrechnung korrekt abläuft.

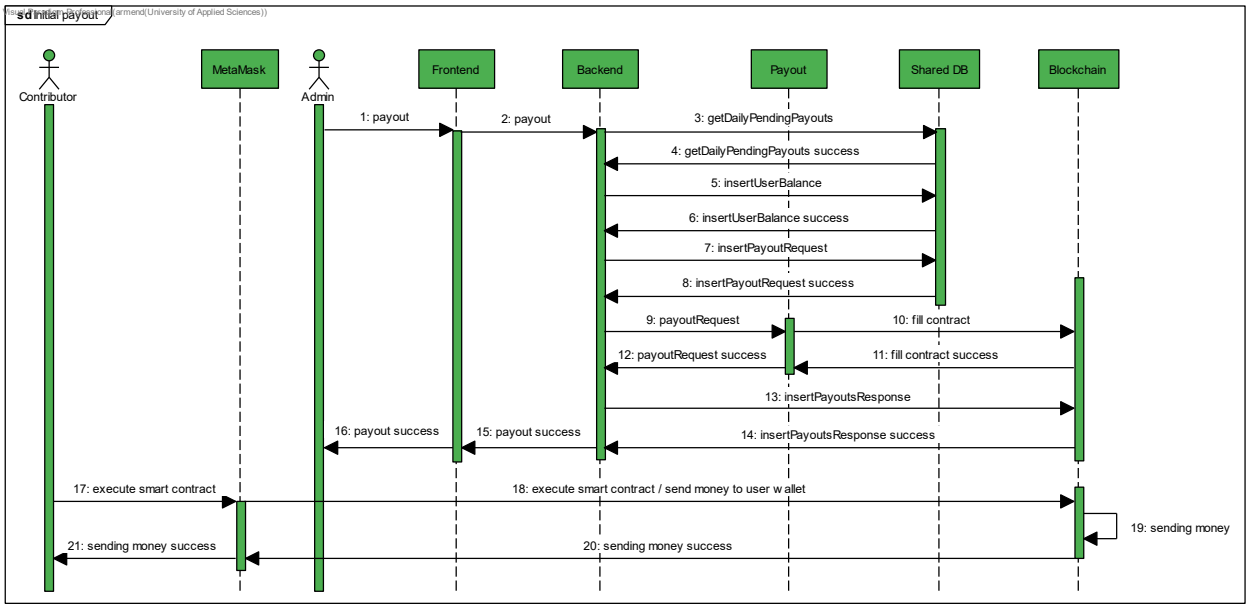


Abbildung 6: Auszahlung IST Zustand

1-2: payout

Der Admin muss die Auszahlung manuell über die Admin Seite starten. Dabei ist es wichtig, dass er den USD zu ETH Wechselkurs angibt. Es ist der Kurs, welcher genutzt wurde, um die USD in ETH umzuwandeln.

3-8: backend process

Sobald der manuelle Payout Request vom Admin gestartet wurde, muss er im Backend vorbereitet werden. Dazu werden zuerst alle Auszahlungen, die ausstehend sind, gesucht und den Contributoren als `INCOME_REQUEST` übertragen. Danach wird der Payout Request für den Payout Service vorbereitet und die Daten in die Datenbank geschrieben.

9-16: payoutRequest

Das Backend macht einen Payout Request an den Payout Service mit den Informationen, welcher Benutzer (Wallet) wie viel WEI erhält. Der Payout Service ruft den Smart Contract auf und füllt diesen mit den Informationen welche Adresse wie viel WEI bekommt. Das abgespeicherte Guthaben auf dem Smart Contract kann vom Benutzer durch die Schritte 17-21 abgeholt werden.

17-21: execute smart contract

Ist das Geld auf dem Smart Contract, kann sich der Contributor das Guthaben jederzeit auszahlen lassen, über eine Smart Contract Funktion. Um dies zu tun, braucht er das Browser Plugin MetaMask, welches benutzt wird, um die Funktion auf der Blockchain auszuführen. Nach dem erfolgreichen Ausführen erhält er seine ETH auf sein persönliches Wallet.

3.2 Anforderungen

3.2.1 Funktionale Anforderungen

In diesem Projekt gibt es zwei fix definierte Anforderungen, welche gleichzeitig den Auftrag widerspiegeln. Da die technischen Möglichkeiten unklar sind, wurden alle weiteren Anforderungen mit dem Auftraggeber angeschaut und diskutiert. Deshalb wurden die Anforderungen und neuen Erkenntnisse wöchentlich abgesprochen.

Die zwei Haupt Anforderungen sind

1. Ein Sponsor hat die Möglichkeit eine Einzahlung in USD, Ethereum, NEO oder Tezos zu tätigen.
2. Jeder Contributor erhält seinen korrekten Anteil.

Anforderungen die später hinzugekommen sind oder im Verlauf des Projektes angepasst wurden:

1. Jeder Contributor kann selbst entscheiden, wann er sich das Geld ausbezahlen lässt.
    a. Durch Änderungen am Ethereum und NEO Smart Contract von Michael Bucher während seiner Masterarbeit, welche Parallel mit dieser Semesterarbeit verlief, hat sich diese Anforderung geändert. Neue Anforderung siehe Punkt b.
    b. Die Auszahlungen werden einmal im Monat über einen Batch Job gemacht
2. Deployment der Erweiterung auf die Test- und Produktionsumgebung
    a. Migration der Datenbank
    b. Erstellung der Wallets
    c. Deployment der Smart Contracts

3.2.2 Use Case Diagramm

Um die Use Cases von FlatFeeStack besser zu verstehen, wurde ein Use Case Diagramm erstellt. Die blauen Use Cases sind die, welche bei dieser Arbeit umgesetzt wurden. Die Multiplikationsfunktionalität für Organisationen, stand zur Auswahl, jedoch wurde sich für die Einzahlung in Kryptowährungen entschieden und ist deshalb kein bestand dieser Arbeit.

Abbildung 7: Use Case Diagramm

3.3 Analyse

Um eine geeignete Lösung zu finden, wurden verschiedene Möglichkeiten in Betracht gezogen und analysiert. In diesem Abschnitt werden die analysierten Möglichkeiten aufgezeigt.

3.3.1 Zahlungsverkehr

Um die Einzahlung von weiteren Kryptowährungen zu akzeptieren, gibt es verschiedene Lösungsstrategien. Es werden alle Varianten vorgestellt mit ihren Vor- und Nachteilen sowie einer Entscheidung der Analyse.

Dabei wurden drei verschiedene Bezahldienste für Kryptowährungen analysiert, welche den Zahlungsverkehr übernehmen, wie Stripe für USD. Zusätzlich gab es die Alternative keinen externen Service zu nutzen, sondern eine Eigenimplementierung.

Die Analyse basiert auf den folgenden 5 Kriterien:

- Unterstützung von ETH, NEO und XTZ
- Komplexität
- Zeitlicher Aufwand
- Vertrauenswürdigkeit
- Gebühren

3.3.1.1 CoinGate

Unterstützung von ETH, NEO und XTZ

CoinGate unterstützt ETH, NEO und XTZ für die Einzahlung. Um das Geld vom CoinGate Konto abzuheben, gibt es Einschränkungen. Von den drei Währungen wird nur ETH als Auszahlungsmethode unterstützt [23].

Komplexität

CoinGate besitzt ein Dashboard mit Informationen über Einzahlungseingänge und Auszahlungen. Es existieren vier Möglichkeiten wie CoinGate angebunden werden kann. Neben der klassischen Anbindung über eine API kann CoinGate auch über einen Point of Sales (PoS), Payment Button oder einer Modul Integration angebunden werden [24]. Alle Möglichkeiten können ohne grosse Komplexität implementiert werden.

Zeitlicher Aufwand

Bevor CoinGate genutzt werden kann, muss das Konto überprüft werden. Diese Überprüfung kann bis zu 14 Tage dauern [25]. Der Implementationsaufwand über die API wird als gering eingeschätzt.

Gebühren

Pro Einzahlungsprozess muss 1% an Gebühren bezahlt werden [26].

Vertrauenswürdigkeit

Auf Trustpilot hat CoinGate viele fünf Sterne Bewertungen, aber auch viele ein Stern Bewertungen [27].

3.3.1.2 NOWPayments

Unterstützung von ETH, NEO und XTZ

NOWPayments unterstützt alle Währungen, die benötigt werden. Im Gegensatz zu CoinGate sind Auszahlungen in ETH, NEO und XTZ möglich [28]. Sobald eine Einzahlung über NOWPayments getätigt wird, wird das Geld an ein hinterlegtes Wallet überwiesen. Gibt es kein passendes Wallet für die einbezahlte Währung, wird der Betrag in die Währung des ersten Wallets umgewandelt und überwiesen.

Komplexität

NOWPayments hat ein Dashboard und bietet eine simple, dokumentierte API an. Durch die simple API ist die Integration nicht komplex [29]. Der grösste Nachteil ist, dass es keine Anbindung an die Testnetze der

Blockchains anbieten. Dies hat zur Folge, dass echtes Geld genutzt werden muss, um den Service zu testen.

Zeitlicher Aufwand

Der Zeitliche Aufwand für die Integration von NOWPayments wird durch die Simple API als gering eingeschätzt.

Gebühren

Die Gebühren sind abhängig vom Transaktionsvolumen. Je mehr Transaktionen über NOWPayments getätigt werden, desto weniger Gebühren müssen bezahlt werden. Die maximalen Kosten belaufen sich auf 0.5% pro Transaktion [30].

Vertrauenswürdigkeit

Die Bewertungen auf Trustpilot von NOWPayments sehen ähnlich aus wie bei CoinGate, ausser dass sie weniger 1 Stern Bewertungen haben [31].

3.3.1.3 Datatrans / Coinify

Unterstützung von ETH, NEO und XTZ

Datatrans ist ein Konkurrent zu Stripe. Ihr Fokus liegt primär auf Fiat Währungen, jedoch bieten sie über ihren Partner Coinify auch Kryptowährungen an. Sie machen einen professionellen und guten Eindruck, jedoch wird von den drei benötigten Währungen nur ETH unterstützt. [32, 33]

Weil NEO und XTZ nicht unterstützt werden, wurden keine weiteren Recherchen betrieben.

3.3.1.4 Eigenimplementierung

Unterstützung von ETH, NEO und XTZ

Durch eine eigene Implementation können alle benötigten Währungen bei der Einzahlung sowie bei der Auszahlung unterstützt werden.

Komplexität

Bei einer Eigenimplementierung müssen folgende Punkte beachtet werden

- Die Lösung muss für alle 3 Währungen implementiert werden
- Es braucht einen Mechanismus zur Überprüfung, ob ein Sponsor bereits eingezahlt hat oder nicht
- Was geschieht, wenn zu viel oder zu wenig einbezahlt wird?
- Die Lösung soll so geringe Transaktionskosten haben wie möglich
- Anbindung neuer Kryptowährungen soll möglichst einfach funktionieren

Durch all die offenen Fragen ist eine Implementierung komplex.

Zeitlicher Aufwand

Weil ein zusätzlicher Service mit hoher Komplexität implementiert werden muss, wird der zeitliche Aufwand hochgeschätzt.

Gebühren

Die Gebühren können nicht genau berechnet werden, weil sie von der Implementation abhängt. Kann eine Lösung mit wenigen Transaktionen implementiert werden, sind die Kosten tief. Verursacht die Lösung hingegen viele Transaktionen, steigen die Kosten.

Vertrauenswürdigkeit

Weil die komplette Kontrolle bei FlatFeeStack liegt, ist die Vertrauenswürdigkeit maximal.

3.3.1.5 Auswertung

Bei der Analyse wurde den einzelnen Kriterien eine Gewichtung zugeteilt. Dann wurden alle Möglichkeiten bewertet mit einer möglichen Bewertung von 0 bis 9. Dabei ist 0 das Schlechteste und 9 das Beste. Es wird sich für die Lösung mit der besten Gesamtwertung entschieden.

Ziel	**Gewicht**	**Coingate**		**NOW-payments**		**Datatrans / Coinify**		**Selbs implementieren**	
		n	n*g	n	n*g	n	n*g	n	n*g
Unterstützt von ETH, NEO und XTZ	30	5	150	9	270	1	30	9	270
Komplexität	10	8	80	8	80	0	0	1	10
Zeitlicher Aufwand	10	7	70	7	70	0	0	3	30
Gebühren	30	5	150	6	180	0	0	5	150
Vertrauenswürdigkeit	20	4	80	6	140	8	160	9	180
Summe	100	530		**740**		190		640	
Rang		3		**1**		4		2	

Tabelle 1: Zahlungsverkehr Analyse

Basierend auf der Analyse, siehe «Tabelle 1: Zahlungsverkehr Analyse», wurde NOWPayments als Lösung gewählt. Trotz der Einschränkung, dass es keine Anbindung an die Testnetz der Blockchains anbietet, ist es die Lösung mit den meisten Vorteilen.

Will man in Zukunft eine Anbindung an die Testnetze haben, muss die Lösung selbst implementiert werden. Dabei kann ein Konkurrenzprodukt zu NOWPayments entwickelt werden, welches unabhängig von FlatFeeStack läuft.

3.3.2 Analyse der Transaktionen

Es gibt verschiedene Möglichkeiten, wie der einbezahlte Betrag an die Contributoren ausbezahlt werden kann. Aus diesem Grund wurde eine Analyse der Transaktionen erstellt.

Dabei gibt es drei Hauptunterscheidungen:

1. Alle Auszahlungen geschehen in einem Stable Coin, eine digitale Währung, die an einen stabilen Reservewert wie den US-Dollar oder Gold gekoppelt ist, zum Beispiel USDT [34].
2. Alle Auszahlungen werden in der einbezahlten Währung wieder ausbezahlt, ohne dass die Einbezahlte Währung umgetauscht wird.
3. Alle Auszahlungen werden in der einbezahlten Währung wieder ausbezahlt, jedoch werden alle Währungen in eine gemeinsame Währung umgewandelt für die Berechnungen. Vor der Auszahlung müssen sie wieder in ihre Ursprungsform umgewandelt werden.

3.3.2.1 Variante 1

In der Variante 1 geht es darum die einbezahlte Währung in einen Stable Coin umzuwandeln. Dieser Stable Coin wird genutzt, um das Geld an die Contributoren zu senden. Bei dieser Lösung gibt es drei Subvarianten.

Subvarianten

1. Jeder Sponsor besitzt ein eigenes Wallet. Nach der Einzahlung wird das Guthaben sofort in einen Stable Coin umgewandelt und dann ausbezahlt.
2. Jeder Sponsor besitzt ein eigenes Wallet. Nach der Einzahlung landet das Guthaben in einen grossen Topf pro Währung. Basierend auf diesen Töpfen werden die Berechnungen durchgeführt. Danach werden die verschiedenen Kryptowährungen in einen Stable Coin getauscht und ausbezahlt.
3. Die Einzahlung geschieht über einen externen Service, welcher das Geld an das jeweilige Wallet pro Währung sendet. Nach den Berechnungen wird alles in einen Stable Coin umgewandelt und versendet.

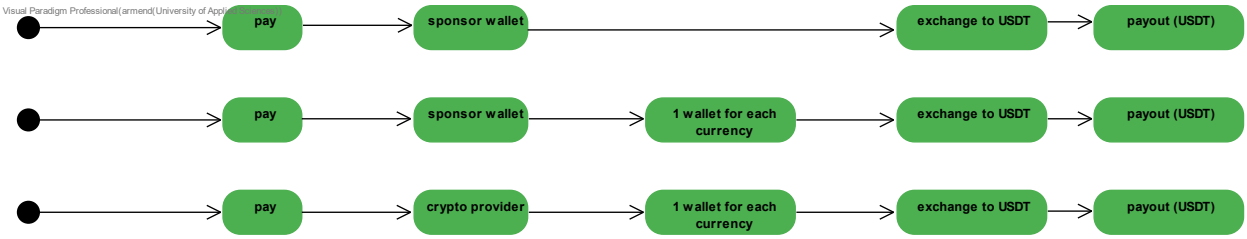


Abbildung 8: Transaktionsanalyse Variante 1

3.3.2.2 Variante 2

In der Variante 2 geht es darum, dass die einbezahlte Währung auch ausgezahlt wird. Wird in vier verschiedenen Währungen einbezahlt, werden bis zu vier verschiedenen Währungen an einen Contributor ausbezahlt. Auch hier gibt es drei Subvarianten

Subvarianten

1. Jeder Sponsor besitzt ein eigenes Wallet. Nach der Einzahlung wird das Guthaben direkt ausbezahlt.
2. Jeder Sponsor besitzt ein eigenes Wallet. Nach der Einzahlung landet das Guthaben in einen grossen Topf pro Währung. Basierend auf diesen Töpfen werden die Berechnungen durchgeführt. Danach werden sie ausbezahlt.
3. Die Einzahlung geschieht über einen externen Service, welcher das Geld an das jeweilige Wallet pro Währung sendet. Nach der Berechnung wird das Geld ausbezahlt.

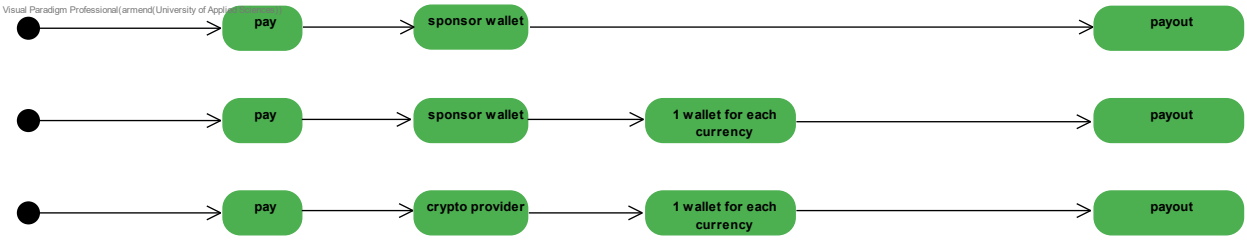


Abbildung 9: Transaktionsanalyse Variante 2

3.3.2.3 Variante 3

Die Variante 3 ist ähnlich zur Variante 2. Der einzige Unterschied ist, dass alle verschiedenen Währungen in eine einheitliche umgewandelt werden für die Berechnungen. Bevor eine Auszahlung stattfindet, muss das Geld wieder zur ursprünglichen Währung umgewandelt werden.

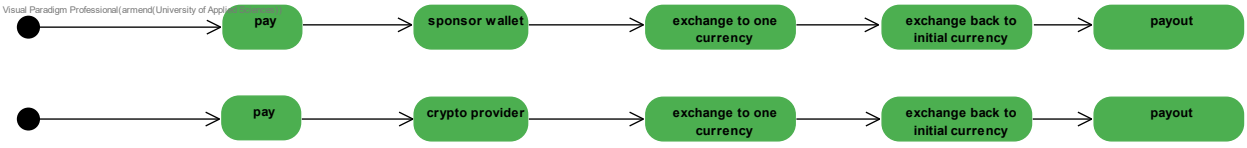


Abbildung 10: Transaktionsanalyse Variante 3

3.3.2.4 USD Variante (Vision)

Aktuell findet die Einzahlung von USD via Stripe statt. Danach muss über einen manuellen Prozess das Geld auf eine Handelsplattform für Kryptowährungen überwiesen und zu ETH umgewandelt werden. NOWPayments bietet die Möglichkeit, dass mit USD einbezahlt werden kann und sie die Umwandlung von einer Fiat Währung zu einer Kryptowährung übernehmen. Somit könnte der manuelle Prozess automatisiert werden. Diese Lösung wird in dieser Arbeit jedoch nicht weiterverfolgt.

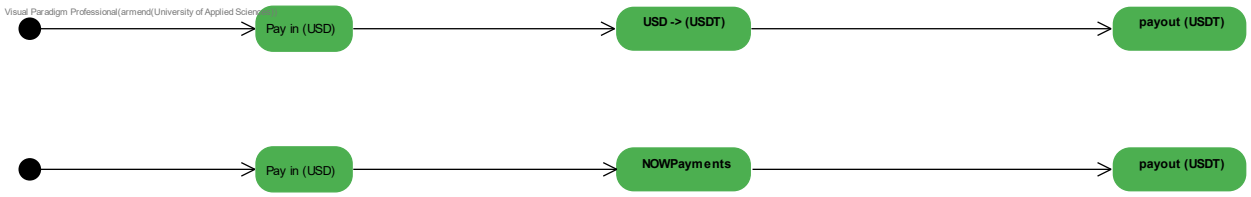


Abbildung 11: Transaktionsanalyse USD Variante (Vision)

3.3.2.5 Auswertung

Der Use Case «Multiplikator für Organisationen» welcher erst in Zukunft umgesetzt wird, spielt bei der Entscheidung eine wichtige Rolle. Bei diesem Use Case geht es darum, dass Organisationen wie NEO gewisse Projekte mit einem Multiplikator unterstützen können. Eine wichtige Argumentation, um Organisationen zu gewinnen ist, dass sie mit ihrer eigenen Währung die Projekte unterstützen können. So hat NEO zum Beispiel die Möglichkeit mit NEO zu bezahlen. Werden die Auszahlungen in die Einbezahlte Währung getätigt, ist der Multiplikator für Organisationen attraktiver, weil es für mehr Aufmerksamkeit für die Kryptowährung sorgt.

Aus diesem Grund scheidet die Variante 1 bereits aus. Die Variante 3 scheidet ebenfalls aus, da die Währung oft umgewandelt wird und so viele Transaktionskosten entstehen. Zusätzlich spielen die starken Kursschwankungen eine wichtige Rolle, weil das Umwandeln zum falschen Zeitpunkt dazu führen kann, dass Geld verloren geht. Indem auf eine Umwandlung verzichtet wird, kann dieses Risiko eliminiert werden.

Somit bleibt nur noch Variante 2 bestehen. Die Entscheidung der Subvariante ist basierend auf der Analyse des Zahlungsverkehrs gefallen. Dort wurde sich für eine Lösung mit einem Online Bezahldienst entschieden. Deshalb ist die Entscheidung auf die Variante 2 und Subvariante 3 gefallen.

3.3.3 Wallet vs. Smart Contract

Bei der Konzeption der Auszahlung der Spenden wurden zwei mögliche Wege gegenübergestellt, um die Vorteile der jeweiligen Variante zu analysieren.

Variante Wallet:

Es wird für jeden Contributor ein Wallet erstellt, welches die erhaltenen Spenden zugeschickt bekommt. Das Wallet wird nur dem jeweiligen Benutzer gezeigt und er kann dadurch entscheiden auf welches eigene Wallet seine Spenden gesendet werden sollen.

Variante Smart Contract:

Es wird ein Smart Contract implementiert, welcher die Spenden jeder einzelnen Adresse aufnimmt. Der Smart Contract kann dann durch diverse Methoden ergänzt werden, um Beispielweise eine Auszahlung seitens eines Users oder eine Auszahlung für mehrere User von einem Admin zu ermöglichen.

Wallet	Smart Contract
1. Ein User muss keine Adresse hinterlegt haben 2. Einfacher zu benutzen, da sich User an Wallets von Trading Sites gewöhnt sind 3. Keine Initiale Kosten beim Aufsetzen vom Wallet	1. Billiger [35] 2. Erster Smart Contract existiert bereits 3. Weniger Code welcher entwickelt werden muss 4. Transparenz gegenüber den Usern durch die Einsehbarkeit eines Smart Contracts 5. Autonome Auszahlung seitens User möglich, ohne Interaktion mit dem Backend 6. Adresse kann bereits eingetragen werden und nur Benutzer mit Adressen werden berücksichtigt
1. Verwalten von vielen Wallets kann aufwändig werden 2. Spenden können vergessen gehen	1. Fehlimplementierung bedeutet den Verlust aller Spenden (Single Point of Failure) 2. Erfordert mehr Aufwand, um mit einem Smart Contract zu interagieren. (Wie viele Spenden hat ein User? Wie kommt ein User an seine Spenden?)

Tabelle 2: Wallets vs. Smart Contracts

Basierend auf den Vorteilen der Smart Contracts wurde der bestehende Ansatz weitergeführt. Ebenfalls können die Nachteile bei korrekter Implementation und genügend Implementationszeit vernachlässigt werden. Für die Sicherstellung der Kostenreduktion der Smart Contracts wird zeitgleich eine Masterarbeit geschrieben.

4. Umsetzung

In diesem Kapitel wird die Umsetzung der Integration von Einzahlungen mit Kryptowährungen in FlatFeeStack dokumentiert. Dabei gibt es eine wichtige Änderung der Smart Contract-Funktionalität, welche die Auszahlung und das Verwalten der Wallet Adressen beeinflusst hat.

4.1 Smart Contract Änderung

Parallel zu dieser Arbeit, wurde eine Masterarbeit geschrieben, welche die Aufgabe hatte, die Smart Contracts von Ethereum und NEO zu optimieren, um die Transaktionskosten zu senken. Die Änderungen wurden in der Woche 7 präsentiert. Aufgrund der Anpassungen musste das Payout und die Verwaltung der Wallet Adressen geändert werden.

Vor der Masterarbeit wurde dem Smart Contract täglich der Betrag mitgegeben, welcher ausbezahlt werden soll. Dabei wurden diese Informationen auf dem Contract gespeichert und aufsummiert. Sobald eine Auszahlung stattgefunden hat, wurde die Summe ausbezahlt und auf 0 gesetzt.

Neu wird der Smart Contract nicht mehr täglich aufgerufen, um Informationen zu speichern. Sondern der Contract wird nur noch für Auszahlungen aufgerufen mit dem Total Earn Amount (TEA). Der TEA ist die Summe aller Einnahmen eines Contributors. Wichtig ist das dieser Wert sich nie verkleinert, sondern immer grösser wird. Bei jeder Auszahlung merkt sich der Smart Contract für welche Adresse er wie viel ausbezahlt hat. Kommt eine weitere Auszahlung hinzu wird der neue TEA minus dem bereits Ausbezahlten Betrag gerechnet und somit wird nur die Differenz gesendet. Wird eine Transaktion mit demselben TEA doppelt ausgeführt, wird beim zweiten Aufruf 0 überwiesen.

Diese Änderung führte bei der Auszahlung zu einem Problem. Der TEA wird pro Benutzer berechnet. Da der Smart Contract keine Benutzer kennt, sondern nur Adressen kann dieses Verhalten ausgenutzt werden. Dabei muss der Benutzer nur warten, bis die erste Auszahlung auf sein Wallet geschehen ist. Ändert er danach die Wallet Adresse, wird bei der zweiten Auszahlung nochmal der gesamte TEA ausbezahlt und nicht die Differenz. Weil der Smart Contract nicht weiss das dieser Benutzer die Adresse geändert hat.

Aus diesem Grund werden beim Berechnen des TEAs die Wallet Adressen berücksichtig. Somit existiert pro Benutzer, Währung und Wallet ein TEA.

Zusätzlich entsteht ein Problem, wenn mehrere Benutzer dasselbe Wallet nutzen wollen. Da der Smart Contract nur Adressen kennt, gibt es für ihn nur einen Benutzer und nicht zwei. In der Applikation sind es jedoch zwei separate Auszahlungen.

Beispiel: Contributor A und Contributor B teilen sich ein Wallet. Zuerst wird dem Contributor A 15 XTZ ausbezahlt, danach soll dem Contributor B 20 XTZ ausbezahlt werden. Schlussendlich ist auf dem Wallet 20 XTZ und nicht 35 XTZ, weil der Smart Contract insgesamt nie mehr als den TEA auf eine Adresse überweist.

Aus diesem Grund wurde entschieden, dass ein Wallet aktuell nicht für mehrere Benutzer genutzt werden kann. Diese Einschränkung kann jedoch in Zukunft behoben werden, indem der TEA pro Wallet berechnet und unabhängig vom Benutzer betrachtet wird.

4.2 Architektur

Damit die neuen Anforderungen umgesetzt werden können, musste die Architektur erweitert werden. Die Komponenten in Lila Blau wurden neu hinzugefügt. Zusätzlich mussten bestehende Services wie Frontend, Backend, Shared Database und Payout Service angepasst werden. Welche Anpassungen genau vorgenommen wurden, wird in den weiteren Kapiteln beschrieben. Hier werden nur die neu eigesetzten Komponenten beschrieben.

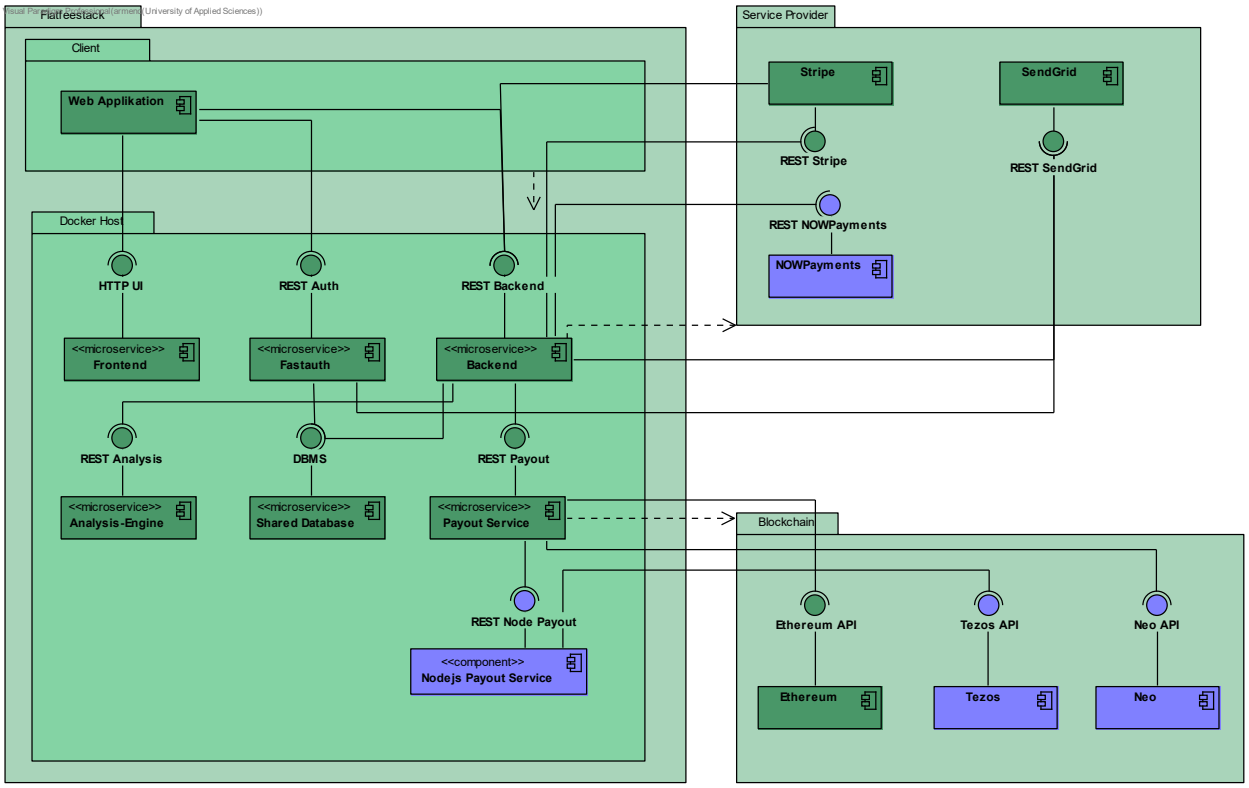


Abbildung 12: Neue Architektur

4.2.1 Backend

Das Backend ist das Herzstück der Applikation. Sie verbindet und koordiniert alle weiteren Services. Durch die neuen Einzahlungen mit Kryptowährungen, wird zusätzlich NOWPayments angesprochen und die Einzahlungen verarbeitet. Weiter wurden die täglichen Berechnungen an die neuen Währungen angepasst. Dabei war es wichtig, die bestehende Funktionalität beizubehalten und die neue Lösung zu integrieren.

4.2.2 Payout

Der simple Payout Service wurde erweitert, damit eine Auszahlung in den neuen Währungen Tezos und NEO ermöglicht wird.

4.2.3 Node.js Payout Service

Die bestehende Code Basis ist in Go. Jedoch gab es für die Blockchain Tezos keine Go Library, welche alle Bedürfnisse gedeckt hat. Da es für Node.js eine gute Library gibt, wurde entschieden einen zusätzlichen Node.js Payout Service zu implementieren. Dieser Service wird aktuell nur für Tezos verwenden. Sollte in Zukunft eine Go Library für Tezos erscheinen, welche alle Bedürfnisse deckt, kann sie einfach abgelöst werden.

4.2.4 NOWPayments

NOWPayments wurde als Service für die Verarbeitung der Einzahlungen in einer Kryptowährung gewählt. Ihr API-Service bietet die Möglichkeit Invoices (Rechnungen) zu erstellen. Bei der Erstellung einer Invoice, wird von NOWPayments eine Einzahlungsmaske erstellt mit dem zu bezahlenden Betrag, die Adresse, an die das Geld gesendet werden muss und einem QR Code für eine vereinfachte Einzahlung. Über einen WebHook wird der FlatFeeStack Service über Updates zur Einzahlung informiert.

4.2.5 Frontend

Das Frontend wurde um eine neue Komponente ergänzt, damit neben der bestehenden USD Einzahlung auch Kryptowährungen ausgewählt werden können. Zusätzlich mussten alle Komponenten, welche in Verbindung mit einer Währung standen, nachgezogen werden.

4.2.6 Tezos

Die Tezos Blockchain wird verwendet, um die XTZ Überweisungen vorzunehmen.

4.2.7 NEO

Die NEO Blockchain wird verwendet, um die NEO Überweisungen vorzunehmen.

4.3 DB

Das Datenbank Modell wurde erweitert für die Integration der neuen Lösung. In der Grafik «Abbildung 13: Neues Datenbank Modell» ist das neue Datenbank Modell zu sehen. Die Tabellen in Lila Blau und spalten in Rot wurden neu hinzugefügt.

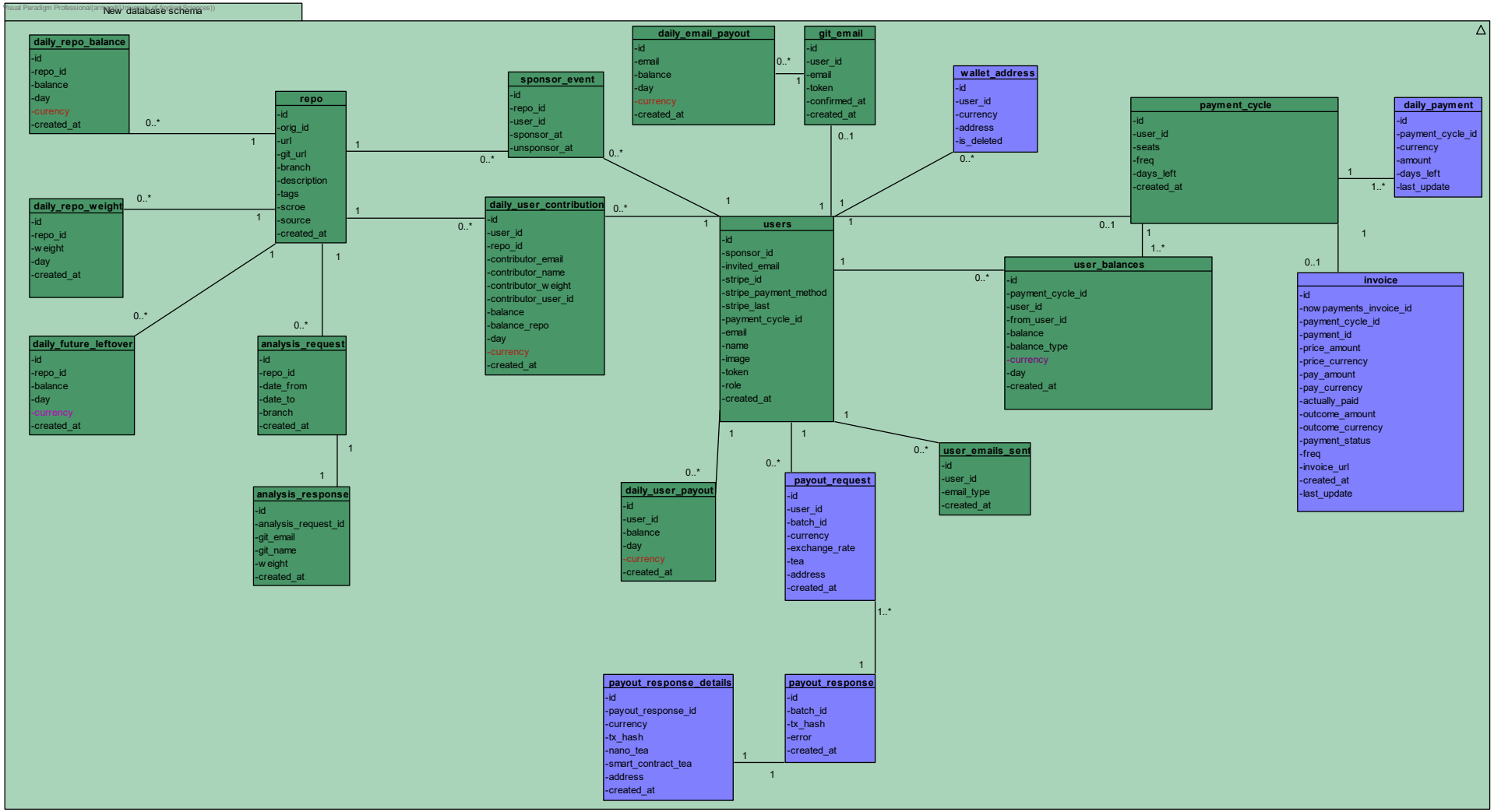


Abbildung 13: Neues Datenbank Modell

Tabelle invoice

Stripe besitzt die Möglichkeit eigene Metainformationen zu einer Einzahlung hinzuzufügen. Dabei können Informationen wie die Payment Cycle ID und die User ID hinterlegt werden. Bei einem WebHook werden diese Metainformationen mitgesendet. Durch diese Informationen ist es möglich herauszufinden, zu welchem Benutzer und Einzahlung dieser WebHook gehört. NOWPayments bietet diese Möglichkeit nicht an. Um zu wissen welcher WebHook zu welchem Benutzer und Payment Cycle gehört, wurde eine neue Tabelle invoice erstellt. Sobald eine neue Invoice über die NOWPayments API erstellt wird, werden diese Informationen in der Datenbank abgespeichert. Mit der invoice_id, welche beim WebHook mitgesendet wird, ist es möglich herauszufinden, zu welchem Benutzer und Payment Cycle der WebHook gehört.

Tabelle daily_payment

Bis anhin wurde allen Sponsoren ein fixer Betrag von 0.33 USD für das Sponsoring abgezogen. Da neu mehrere Währungen angeboten werden und Kryptowährungen einen stark schwankenden Wechselkurs haben, kann kein Fixbetrag mehr für alle Währungen abgezogen werden. Aus diesem Grund wurde die Tabelle daily_payment hinzugefügt. Damit hat jeder Sponsor für jede Währung einen eigenen Wert, welcher täglich abgezogen wird. Für die Berechnung dieses Wertes wird der einbezahlte Betrag durch die Anzahl ausgewählter Tage (365 oder 91) geteilt. Hat der Sponsor noch Restguthaben auf seinem Konto, wird dieses bei der Berechnung mitberücksichtig.

Tabelle wallet_address

Die Ethereum Wallet Adresse wurde vor der Änderung direkt beim Benutzer in der Spalte payout_eth abgespeichert. Mit der Anbindung mehrerer Kryptowährungen, kann ein Contributor aktuell bis zu drei Wallets besitzen. Weil in Zukunft zusätzliche Kryptowährungen hinzukommen könnten, muss das Speichern der Wallet Adressen skalieren, ohne dass die Datenbank angepasst werden muss, wenn eine neue Kryptowährung hinzukommt. Aus diesem Grund wurde entschieden eine eigene Tabelle wallet_address für die Wallets zu erstellen. Kommt in Zukunft eine neue Währung hinzu, kann einfach ein neuer Eintrag für den Benutzer, mit der neuen Währung, erstellt werden.

Tabelle payout_request

Neu werden monatliche Batch Jobs für das Ausbezahlen ausgeführt. Durch diese Änderung wurde entschieden den payout_request neu mit der Tabelle users zu verbinden und nicht mehr mit daily_user_payouts.

Tabelle payout_response / payout_response_details

In den Tabellen payout_response und payout_response_details werden die Antworten vom Payout Service bei einer Auszahlung abgespeichert. Ein Payout Request wird an den Payout Service gestellt. Dieser kann fehlschlagen, deshalb werden die Antworten in der Datenbank gespeichert. Zusätzlich wird die Information benötigt, um dem Contributor anzuzeigen, wie viel Geld er bei der nächsten Auszahlung erhält. Dabei wird der TEA von allen Auszahlungen minus dem bereits ausbezahlten TEA gerechnet.

Spalte Currency

Weil zwischen den einzelnen Währungen unterschieden werden muss, wurde in allen Tabellen, in denen Geld vorkommt eine neue Spalte Currency hinzugefügt.

4.4 Einzahlung

Bei der Stripe Einzahlung hat sich am grundlegenden Prozess nichts geändert. Es wurden kleinere Anpassungen getätigt, damit sie weiterhin mit der neuen Lösung kompatibel bleibt.

Bei der Einzahlung in USD werden die USD in Micro USD umgewandelt. NOWPayments liefert ihre Beträge mit acht Nachkommastellen aus. Damit auch hier nicht mit Fliesskommazahlen gerechnet werden muss, wurde entschieden alle Kryptowährungen in Nanogrösse umzuwandeln (zum Beispiel: 1 ETH = 1'000'000'000 Nano ETH).

In dieser Grafik «Abbildung 14: Einzahlung mit einer Kryptowährung» sieht man den Einzahlungsprozess mit einer Kryptowährung über NOWPayments.

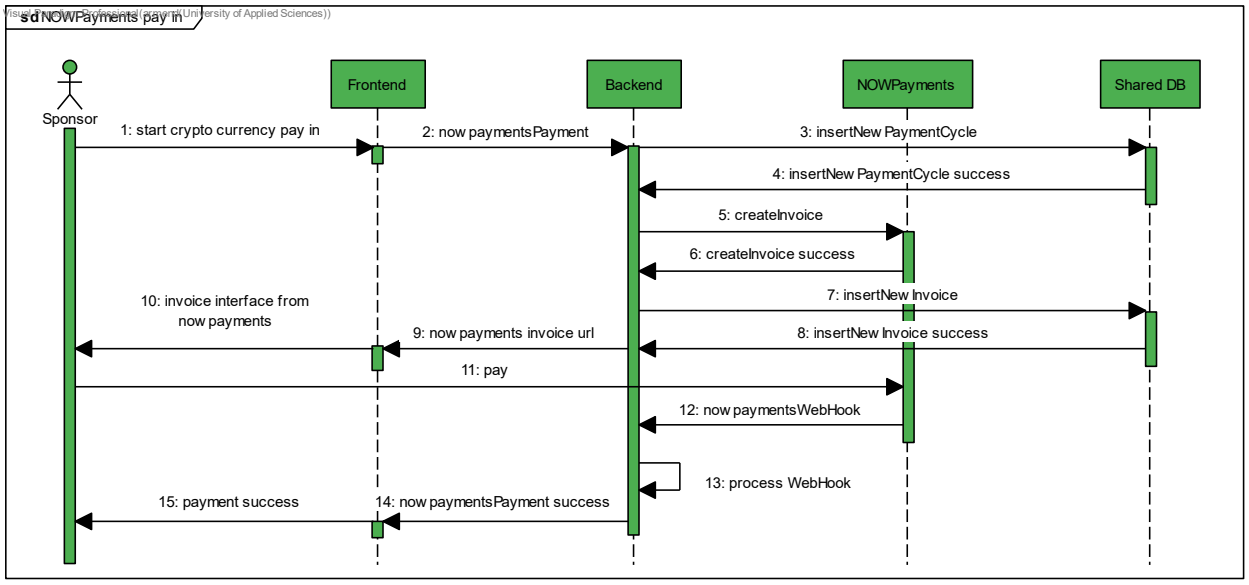


Abbildung 14: Einzahlung mit einer Kryptowährung

1: start crypto currency pay in

Der Sponsor entscheidet sich für eine Einzahlung in einer der zur Verfügung stehenden Kryptowährungen.

2-10: nowpaymentsPayment

Bei der Einzahlung wird als erstes ein neuer Payment Cycle erstellt. Danach wird über die NOWPayments API eine Invoice erstellt und diese Daten werden in der Datenbank abgespeichert. Das Abspeichern ist nötig, weil beim WebHook herausgefunden werden muss, zu welchem Sponsor dieser gehört. Zusätzlich liefert NOWPayments eine Invoice URL. Dieser Link stellt alle nötigen Informationen zur Verfügung, um zu bezahlen.

Dabei werden folgende Informationen angezeigt:

- Der zu Bezahlende Betrag in der ausgewählten Währung. Dabei wird der Betrag alle 5 Minuten aktualisiert.
- Die Wallet Adresse, an die der Betrag gesendet werden muss.
- Ein QR-Code, welcher eingescannt werden kann, für ein einfacheres Bezahlen.

11-12: pay

Der Sponsor überweist das Geld an die angezeigte Adresse und FlatFeeStack wird über die Updates der Überweisung informiert. Dabei können bei der Überweisung zwei Probleme auftreten.

Problem 1: Die Volatilität

Beispiel: Der zu bezahlende Betrag wird von NOWPayments alle 5 Minuten aktualisiert. Der Sponsor muss 28.46 XTZ bezahlen. Er überweist den Betrag. Bevor die Überweisung abgeschlossen wird, aktualisiert NOWPayments den zu bezahlenden Betrag und somit muss er neu 28.59 XTZ bezahlen. NOWPayments erhält die 28.46 XTZ, jedoch ist die Einzahlung nicht abgeschlossen, weil sich der Betrag um 0.13 XTZ erhöht hat.

Problem 2: Transaktionsgebühren

Es gibt verschiedene Möglichkeiten wie ein Benutzer das Geld an NOWPayments senden kann. Wie die Transaktionsgebühren verrechnet werden, ist dabei immer schwer vorhersehbar. Es gibt zwei Möglichkeiten wie die Gebühren verrechnet werden können. Hier ein Beispiel mit einer Überweisung von 100 XTZ und einer Transaktionsgebühr von 5 XTZ.

1. Die Software, über welche die Überweisung getätigt wird, zeigt den Preis inklusive Gebühren an. Der Sponsor bezahlt 105 XTZ und FlatFeeStack erhält 100 XTZ.
2. Die Software, über welche die Überweisung getätigt wird, zeigt den Preis exklusive Gebühren an. Der Sponsor bezahlt 100 XTZ und FlatFeeStack erhält 95 XTZ.

Um die beiden Probleme zu minimieren, bietet NOWPayments die Option «Payment covering» an. Wird sie auf 5% gesetzt und der Sponsor muss 100 XTZ bezahlen, werden Zahlungen ab 95 XTZ akzeptiert. Die Option wird am Anfang auf 5% gesetzt, kann später jedoch geändert werden, wenn man merkt das diese zu hoch oder zu tief ist.

13-15: process WebHook

Wird ein WebHook empfangen, wird er zuerst validiert und geprüft, ob es wirklich ein gültiger WebHook von NOWPayments ist. Bei der Validierung wird der Request Body sortiert und zu einem String umgewandelt. Dieser wird mit einem geheimen Schlüssel, welcher über das NOWPayments Dashboard erstellt wird, signiert. Das Resultat wird mit der mitgelieferten Signatur verglichen. Bei einer Übereinstimmung wurde die Anfrage nicht manipuliert.

Bei einem WebHook mit einer erfolgreichen Einzahlung, werden alle nötigen Daten in der Datenbank aktualisiert. Ist bei der Überweisung etwas schiefgelaufen, wird der Benutzer per E-Mail benachrichtig was das Problem war.

4.5 Tägliche Berechnungen

Dadurch das mehrere Währungen als Bezahlmethode angeboten werden, wurden alle Berechnungen angepasst. In diesem Abschnitt werden alle Anpassungen an die Berechnungen erläutert.

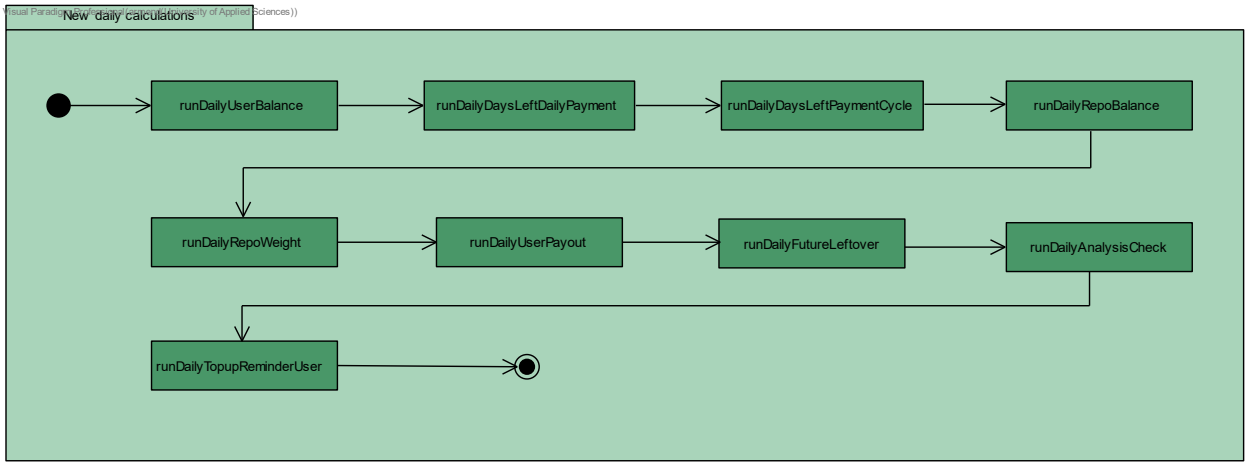


Abbildung 15: Tägliche Berechnungen neu

runDailyUserBalance

Ein Sponsor kann Guthaben in mehreren Währungen haben. Beim Abziehen des täglichen Betrages, wird immer das Geld von der Währung abgezogen, welche am wenigsten Guthaben hat.

runDailyDaysLeft

Die noch zur Verfügung stehenden Tage können nicht mehr in einem einzelnen Task berechnet werden. Die Komplexität ist hoch, da jeder Sponsor mehrere Währungen haben kann und pro Währung hat jeder Sponsor unterschiedliche tägliche Beiträge zu bezahlen. Deshalb wurde diese Aufgabe in zwei Teilaufgaben aufgeteilt. Einmal in runDailyDaysLeftDailyPayment und einmal in runDailyDaysLeftPaymentCycle.

runDailyDaysLeftDailyPayment

In diesem Schritt wird berechnet wie viele Tage ein Sponsor pro Währung noch zur Verfügung hat. Dazu wird sein Guthaben durch seine persönlichen täglichen Beiträge pro Währung geteilt.

runDailyDaysLeftPaymentCycle

Dadurch das ein Sponsor Guthaben in mehreren Währungen haben kann, müssen alle zur Verfügung stehenden Tage pro Währung aufsummiert werden, um die zur Verfügung stehenden Tage pro Payment Cycle herauszufinden.

runDailyRepoBalance, runDailyUserPayout, runDailyFutureLeftover

Diese Berechnung, wurde angepasst, damit die verschiedenen Währungen berücksichtigt werden.

runDailyRepoHours

Durch die steigende Komplexität mit mehreren Währungen wurde entschieden das nur noch Geld an Projekte zugewiesen wird, welche für mindestens 24h gesponsort sind. Aus diesem Grund kann die DailyRepoHours Berechnung gelöscht werden.

getDailyUserPayouts

Das Übertragen der ausstehenden Auszahlungen in die User Balance wird nicht mehr getätigt. Will ein Contributor auch Sponsor sein, so muss immer eine Einzahlung gemacht werden. Somit kann das erhaltene Geld als Contributor nicht mehr genutzt werden, um weitere Projekte zu unterstützen.

Diese Änderung wurde gemacht, weil es durch die extremen Kursschwankungen bei den Kryptowährungen zu einem Edge-Case kommt, welche zu unerwünschten Nebeneffekten führen kann.

Beispiel: Sponsor X bezahlt 365 XTZ für 1 Jahr, da der Kurs niedrig war. Täglich wird ihm 1 XTZ abgezogen und an seine Favorisierten Projekte gutgeschrieben. Nach 9 Monaten steigt der Kurs und 1 Jahr kostet nur noch 1 XTZ. Täglich müsste man neu nur noch 0.0027 XTZ bezahlen. Wenn der Sponsor jetzt gleichzeitig ein Contributor ist, erhält er pro Tag einen 365 Mal kleineren Betrag an XTZ, da die Kurse stark gestiegen sind. Werden die Contributor Einnahmen auch genutzt, um weitere Projekte zu favorisieren, werden sie mit dem Kurs von vor 9 Monaten behandelt und verlieren somit an Wert.

Aus diesem Grund wurde entschieden das Guthaben von den Sponsoren und Contributoren zu trennen. In Zukunft kann ein Feature angeboten werden, welches das Contributor-Guthaben zum Sponsor-Guthaben umwandelt. Bei der Umwandlung muss der aktuelle Kurs berücksichtigt werden, damit die Kryptowährung nicht an Wert verliert oder gewinnt.

4.6 Auszahlung

Die Auszahlung ist neu ein monatlicher manueller Job. Der Unterschied zur Ausgangslage ist, dass das Geld direkt auf das Wallet vom Contributor überwiesen und die Informationen nicht mehr nur im Smart Contract abspeichert wird. Bei der neuen Lösung ist für den Benutzer alles automatisiert. Sobald er eine Wallet Adresse angegeben hat, wird sein Geld monatlich direkt auf seine Adresse gesendet. Zu einem späteren Zeitpunkt soll es noch möglich sein, dass der Benutzer selbst entscheiden kann, wann er sich das Geld auf sein Wallet überweisen will.

Die Auszahlung von USD und ETH musste zusammengeführt werden. Das Problem ist, das USD in ETH ausbezahlt und für alle Auszahlungen der TEA genutzt wird und nicht mehr der zu überweisende Betrag. Der Smart Contract merkt sich jeweils den letzten TEA und bezahlt nur die Differenz aus.

Beispiel: Es müssen 100 USD (0.022 ETH) und 0.05 ETH ausbezahlt werden. Zuerst werden die 100 USD (0.022 ETH) überwiesen. Der Smart Contract merkt sich diesen Betrag. Bei der zweiten Auszahlung sollen

0.05 ETH ausbezahlt werden, da aber bereits 0.022 ETH überwiesen wurden, überweist der Smart Contract nur die Differenz, also 0.028 ETH. Somit hat der Contributor am Ende 0.05 ETH erhalten, statt 0.072 ETH.

Aus diesem Grund werden die Auszahlungen in USD in ETH umgewandelt und mit den anderen ETH zusammengerechnet. Zusätzlich wird eine Transaktion weniger ausgeführt und spart somit an Transaktionskosten.

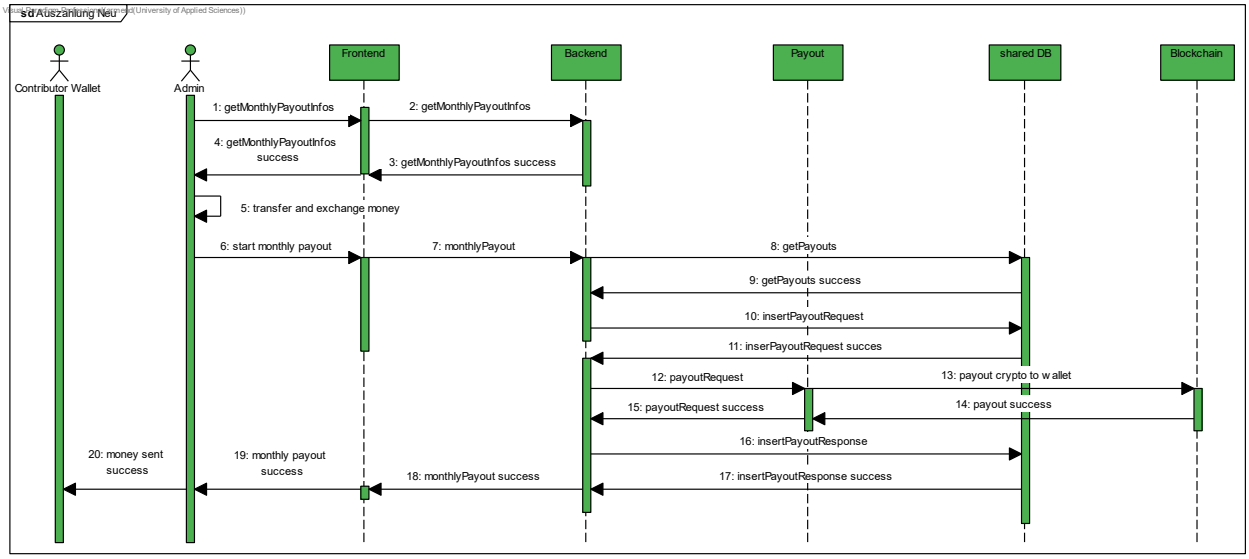


Abbildung 16: Auszahlung neu

1-4: getMonthlyPayoutInfos

Diese Schritte dienen als Vorbereitung. Bevor eine Überweisung stattfinden kann, muss auf den Smart Contracts genügend Guthaben vorhanden sein. Dieser Schritt ist aktuell nicht automatisierbar, da es bei Tezos nicht möglich ist, das Geld direkt auf die Smart Contract Adresse zu senden. Es wurde jedoch bereits eine Methode im Smart Contract vorbereitet, um den Smart Contract aufzuladen. Da bei diesem Aufruf weitere Transaktionskosten entstehen, ist durch diese Methode der Prozess noch nicht vollkommen automatisierbar. Ausserdem müssen die USD Einzahlungen manuell zu ETH umgetauscht und auf den Contract überwiesen werden.

5: transfer and exchange money

In diesem Schritt werden die USD manuell vom Bankkonto auf eine Handelsplattform für Kryptowährungen überwiesen und in ETH umgewandelt. Zusätzlich muss sichergestellt werden, dass bei allen Smart Contracts genügend Guthaben vorhanden ist. Wenn Guthaben fehlt, müssen die Contracts manuell aufgeladen werden.

6-11: start monthly payout

Der Admin muss das monatliche Payout manuell starten. Wichtig ist, dass der korrekte USD zu ETH Wechselrate angegeben wird. Dieser Schritt kann erst automatisiert werden, wenn es eine automatisierte Lösung gibt USD in ETH umzuwandeln.

12-20: payoutRequest

Hier werden die verschiedenen Blockchains über den Payout Service / Node.js Payout Service angesprochen. Dabei werden die Smart Contract Funktionen aufgerufen, um das Geld zu überweisen.

4.6.1 Payout Service

Im Folgenden wird der Ablauf eines Auszahlungsprozess seitens des Payout-Service kurz beschrieben:

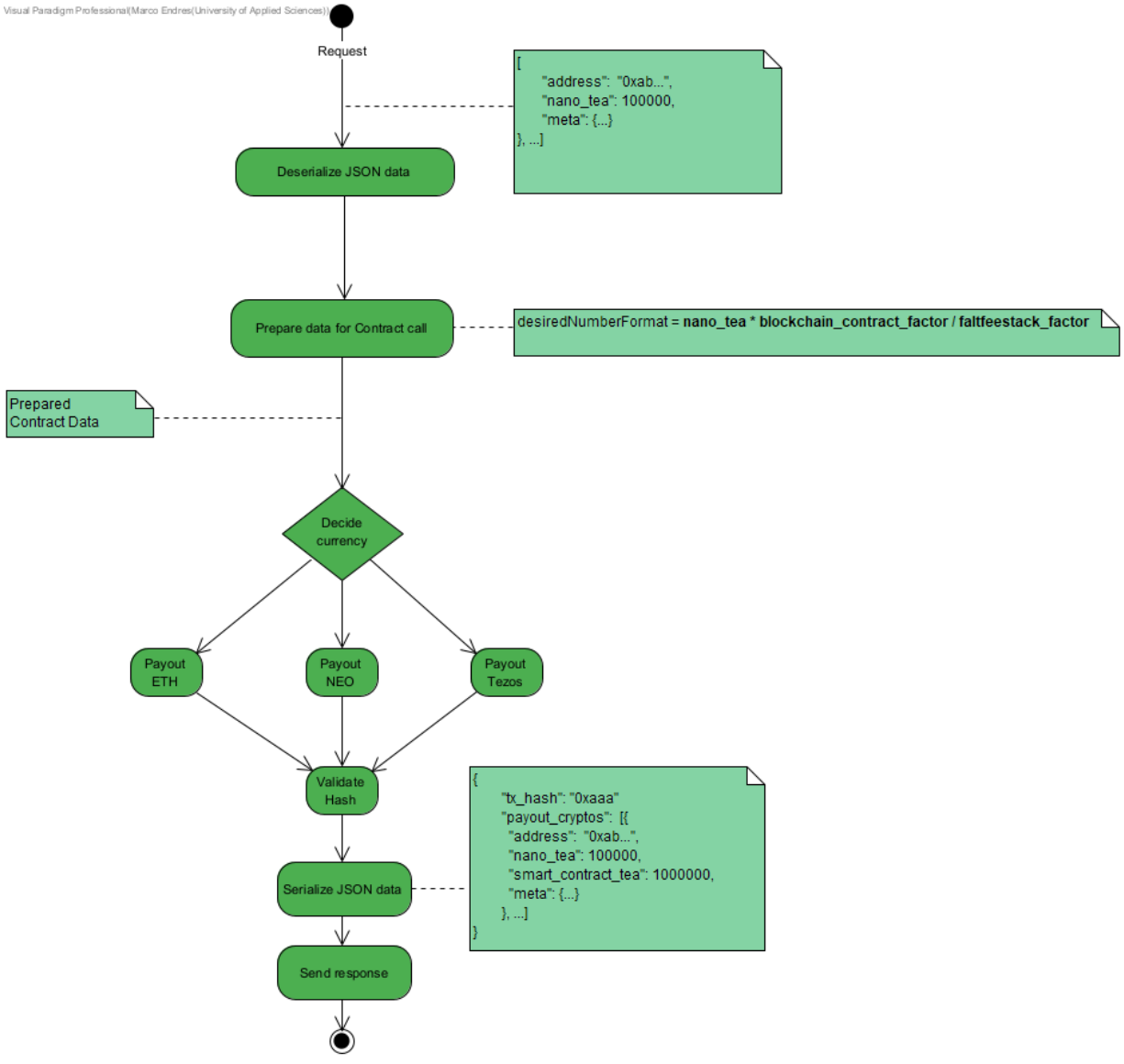


Abbildung 17: Ablauf Payout

Deserialize JSON data

Als erstes werden die Daten, welche von REST-Service empfangen werden in richtige Objekte transformiert und geprüft. Die Daten bestehen dabei aus der Adresse des Benutzers, des maximalen TEAs und gewissen Metadaten, welche vom Backend benötigt werden, um die einzelnen Zahlungen im Nachhinein wieder identifizieren zu können.

Prepare data for Contract call

In diesem Part werden die Daten für den jeweiligen Smart Contract vorbereitet. Dies beinhaltet zurzeit die Umrechnung der Währung von Nano in das richtige Format. Bei Ethereum ist das zum Beispiel Wei (1 Eth = 10^{18} Wei). Dies wird gemacht, indem der Betrag mit dem Faktor der Währung multipliziert und durch den Nanofaktor geteilt wird.

Beispiel: Berechnung von 10^4 Nano ETH in WEI

$$\frac{10^4\ NanoEth * 10^{18}}{10^9} = 10^{13}\ Wei$$

Pro Währung wird also ein Faktor in einer HashMap gehalten. Zusätzlich wird der Standardfaktor beim Aufstarten der Applikation gesetzt. Die Faktoren werden anfänglich als Float initialisiert, um eine spätere Konversion zu verhindern. Da bei den Berechnungen immer abgerundet wird, ist die Verwendung von Float ohne weitere Probleme möglich.

```
eth := Blockchain{Factor: big.NewFloat(1000000000000000000)}
neo := Blockchain{Factor: big.NewFloat(100000000)}
xtz := Blockchain{Factor: big.NewFloat(1000000)}
defaultCryptoFactor.SetString("1000000000")
```

Die Zahlen werden folgendermassen in Go verrechnet:

```
balance = balance.Mul(balance, opts.Blockchains[cur].Factor)
balance = balance.Quo(balance, defaultCryptoFactor)
```

Decide Currency

Beim Entscheiden der Währung wird auf den mitgegebenen URL-Parameter geachtet. Dabei betrifft ein Aufruf immer eine Währung.

Für jede Währung wurde eine ähnliche Methode zum Auszahlen implementiert. Dadurch wurde ein einfacher switch-case über die verschiedenen Währungen implementiert, um so die richtige Auszahlung für einen Aufruf auszuführen.

```
switch cur {
case "eth":
    txHash, err = payoutEth(ethClient, addresses, amount)
    if err != nil {...}
    break
case "neo":
    txHash, err = payoutNEO(addresses, amount)
    if err != nil {...}
    break
case "xtz":
    txHash, err = payoutNodejsRequest(payoutCrypto, currency: "xtz")
    if err != nil {...}
    break
default:...
}
if txHash == "" {...}
p = &PayoutCryptoResponse{TxHash: txHash, PayoutCryptos: payoutCrypto}
```

Abbildung 18: Switch-case über die verschiedenen Kryptowährungen

Validate Hash

Nach der Auszahlung wird geprüft, ob wirklich ein Hash von der Transaktion erzeugt wurde. Falls dies der Fall ist, wird das Antwortobjekt erzeugt.

Serialize JSON data

Vor dem Senden werden die gezahlten Anteile und der dazugehörige Hash zurück in ein JSON konvertiert.

Send Response

Ganz am Ende wird eine positive Response an das Backend zurückgegeben.

4.7 Kryptowährungen

Die höchste Priorität hatte Ethereum, da ein grosser Teil der bestehenden Codebasis übernommen werden konnte. Die niedrigste Priorität wurde NEO zugeordnet, da eine volle Integration von NEO durch die Entscheidung 4.8.1 nicht möglich war. Während der Implementierung wurde mit Michael Bucher zusammengearbeitet, welcher bei seiner Masterarbeit die Smart Contracts von Ethereum und NEO optimiert hat. Er unterstütze den Prozess der Anbindung durch die Bereitstellung der Smart Contracts und allfälligen Beispielcode seinerseits.

Jede Währung implementiert für ein einfacheres Testing, Deployment und zukünftige Implementierungen eine `deploy-` und eine `payout-`Methode. Das Deployment wird über eine `.env`-Variable gesteuert.

Ebenfalls wurden alle bestehenden und neuen `.env`-Variablen so gescoped, dass man zukünftig einfach neue Kryptowährungen hinzufügen kann. Folgende Variablen werden pro Währung gebraucht:

Variablenname	Beschreibung
[CUR]*_URL	URL zur Kommunikation mit der Blockchain
[CUR]*_DEPLOY	Ob der Smart Contract deployed werden soll
[CUR]*_PRIVATE_KEY	Der Private Key mit welchem der Smart Contract deployed und aufgerufen wird
[CUR]*_CONTRACT	Die Adresse des Smart Contracts

[CUR]* ersetzen durch die Währung

Tabelle 3: .env Variablen Definition

4.7.1 Ethereum

Bei der Implementierung in Ethereum wurde sich mehrheitlich an das bestehende Setup gehalten. Es wurde ein privates Netzwerk, um den Smart Contract zu deployen, benutzt. Ebenfalls wurde ein Tool eingesetzt, um die Interaktion mit dem Smart Contract zu vereinfachen. Das Tool nennt sich «abigen» und generiert ein Binding für Go. Die genaue Implementierung des Binding wird später noch genauer erläutert. Erste Versuche wurden mit Remix, einer Online IDE von Ethereum, gemacht [36].

Privates Netzwerk

Für das private Netz wurde, im Vergleich zum bisherigen Setup, Ganache [37] eingesetzt. Das wurde gemacht, um ein möglichst einfaches und gleichzeitig einsteigerfreundliches Setup für die Smart Contract Anbindung zu haben, da zur Zeit der Implementierung keine Erfahrungen im Bereich der Blockchain-Entwicklung existierten.

Es folgt zur Veranschaulichung die Abbildung «Abbildung 19: Screenshots von Ganache GUI» vom GUI von Ganache. Es zeigt welche Adressen existieren und welche Transaktionen getätigt wurden auf der privaten Blockchain.

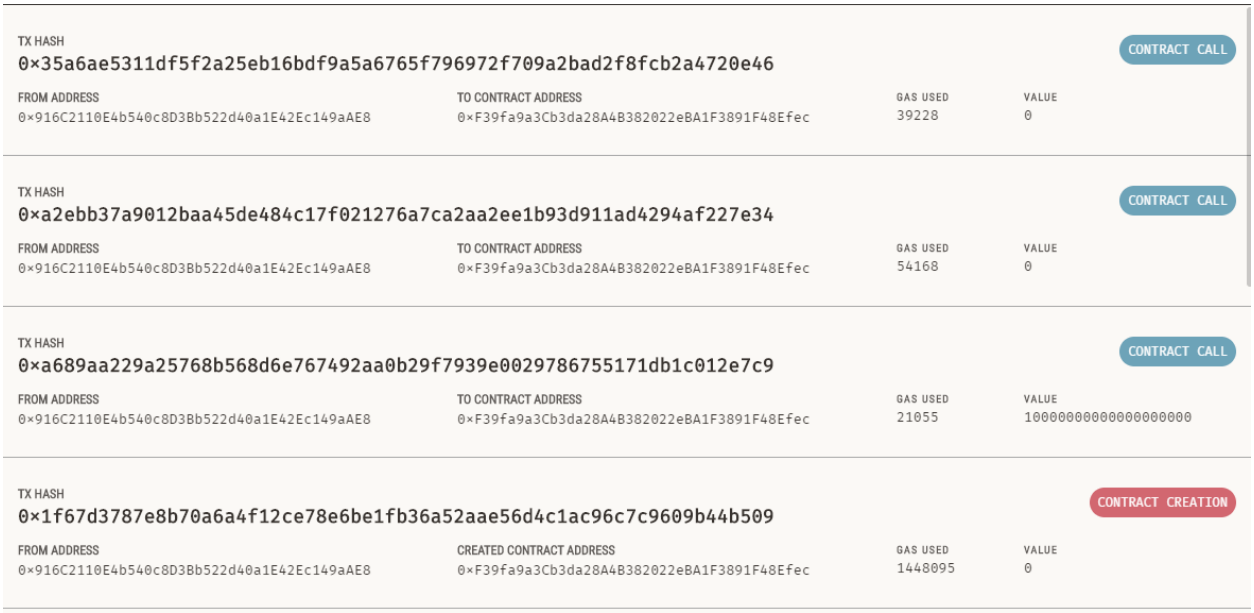


Abbildung 19: Screenshot von Ganache GUI

Bibliothek / Go-Binding

Das Go-Binding ermöglicht es im Programmcode von Go die Smart Contract Funktionen typisiert aufzurufen. Die Typensicherheit erhöht die Sicherstellung der korrekten Parameterübergabe beim Funktionsaufruf. Ebenfalls kann das Deployment über das Binding abgewickelt werden. Ohne das Binding muss auf die Methoden über einen String zugegriffen und die Parameter können nicht validiert werden.

Der Generator für das Binding kommt von der go-ethereum Library und nutzt darunter den Solidity Compiler «Solc» [38]. Der folgende Befehl wird für die Generierung benutzt:

```
abigen --pkg main --sol FlatFeeStack.sol --out ./contract.go
```

Als Input braucht es einen Package-Namen, eine Solidity-Datei und eine Ziel-Go-Datei [39].

Durch dieses Binding kann danach auf eine Smart Contract Methode wie auf eine native Methode typensicher zugegriffen werden mit den entsprechenden Parametern:

```
01: transactor, err := bind.NewKeyedTransactorWithChainID(ethClient.privateKey,
ethClient.chainId)
02: tx, err := ethClient.contract.BatchPayout(transactor, addresses, teas)
```

Auch für das Deployment ist bereits eine Methode durch das Binding vorbereitet:

```
01: transactor, err := bind.NewKeyedTransactorWithChainID(ethClient.privateKey,
ethClient.chainId)
02: address, tx, contract, err := DeployPayoutEth(transactor, ethClient.c)
03: _, err = bind.WaitDeployed(context.Background(), ethClient.c, tx)
```

4.7.2 NEO

Bei NEO wurde ebenfalls eine Anbindung mit Go implementiert. Im Vergleich zu Ethereum, gab es bis anhin aber keine bestehende NEO Anbindung. Darum war der initiale Aufwand grösser, um mit einem Smart Contract auf der NEO3-Blockchain zu interagieren. Die Anbindung wurde mit der NEO-GO-Library implementiert [40].

Privates Netzwerk

Für die lokale Entwicklung gibt es kein simples Docker-Image. Jedoch wurde für die lokale Entwicklung «neo3-privatenet-docker» verwendet, welches es ermöglicht mittels drei Befehlen eine private NEO3-Chain zu starten [41].

neo-go

Die Implementierung mit neo-go war zeitintensiver, da es keine Beispiele mit Smart Contracts gab. Dazu begrenzt sich die Dokumentation der Library auf das Minimum.

Ein Smart Contract Aufruf benötigt schon mehrere Zeilen mithilfe der Library. Der Code ist inspiriert aus der CLI Implementation von neo-go, da diese ebenfalls auf der neo-go Library basiert.

Payout Implementierung

Es folgt der Code wie der Aufruf des Auszahlens mit neo-go implementiert wurde.

```
01: w := io.NewBufBinWriter()
02: emit.AppCall(w.BinWriter, payoutNeoHash, "batchPayout", callflag.All, devP, teaP)
03: script := w.Bytes()
04: sender := acc.PrivateKey().GetScriptHash()
05: signer := transaction.Signer{
06:   Account: sender,
07:   Scopes:  transaction.CalledByEntry,
08: }
09: tx, err := c.CreateTxFromScript(script, acc, -1, 0, []client.SignerAccount{{
10:   Signer: signer,
11: }})
12: err = acc.SignTx(c.GetNetwork(), tx)
13: hash, err := c.SendRawTransaction(tx)
```

Zeile 01-03

Zuerst wird ein Bytecode aus den Parametern und den aufzurufenden Smart Contract erstellt.

Zeile 04-08

Als nächstes wird ein Signer, basierend auf einem Private Key erstellt. Mit Scopes kann bei NEO3 die Gültigkeit der Signatur eingeschränkt werden. Zum Beispiel kann eingeschränkt werden, dass die Signatur nur innerhalb des Smart Contracts gültig ist [42] [43].

Zeile 09-11

Danach wird die Transaktion erstellt mit dem Byteskript, dem Account, den Kosten welche bei kleiner -1 berechnet werden via RPC-Call in der Methode, den zusätzlichen Netzwerkkosten und dem Account, welcher das Skript aufrufen darf.

Zeile 12-13

Am Ende wird die Transaktion mit dem Ersteller Account unterzeichnet und an das NEO-Netzwerk gesendet.

Deployment Implementierung

Das Deployment eines Smart Contracts auf der NEO3-Chain ähnelt dem des Auszahlungs-Aufrufes:

```
01: nativeManagementContractHash, err :=
        c.GetNativeContractHash(nativenames.Management)
02: ne, nefB, err := readNEFFile("./PayoutNeo.nef")
03: _, mfB, err := readManifest("./PayoutNeo.manifest.json")
04: sender := acc.PrivateKey().GetScriptHash()
05: pk := acc.PrivateKey().PublicKey().Bytes()
06: appCallParams := []smartcontract.Parameter {
07:   { Type:  smartcontract.ByteArrayType, Value: nefB,},
08:   {Type:  smartcontract.ByteArrayType, Value: mfB, },
09:   {Type:  smartcontract.PublicKeyType, Value: pk, },
10: }
11: contractHash := state.CreateContractHash(sender, ne.Checksum, "PayoutNeo")
12: signer := transaction.Signer{
13:   Account: sender,
14:   Scopes:  transaction.Global,
15: }
16: resp, _ := c.InvokeFunction(nativeManagementContractHash, "deploy",
        appCallParams, []transaction.Signer{signer})
17: tx, err := c.CreateTxFromScript(resp.Script, acc, -1, 0, []client.SignerAccount{{Signer:
        signer}})
18: txHash, err := c.SignAndPushTx(tx, acc, nil)
```

Zeile 01

Zuerst wird der Hash des «ManagementContract» geholt. Dieser ist dazu gedacht um neue Smart Contracts auf der NEO3-Chain zu deployen [44].

Zeile 02-03

Als nächstes wird das «.nef» und das Manifest geholt, um die nötigen Infos über den Smart Contract zu erhalten. Beide Dateien werden beim Kompilieren des Smart Contracts erzeugt [45].

Zeile 04-10

Danach wird der Public Key und der Private Key vorbereitet und der Public Key wird zusammen mit den gelesenen Dateien als Smart Contract-Parameter verpackt.

Zeile 11

Mit den Dateien und den Parametern kann der endgültige Hash des Smart Contracts berechnen werden.

Zeile 12-15

Als nächstes wird der Signer für die Transaktion erstellt. Der Scope sollte so eingeschränkt werden, dass der Smart Contract keinen weiteren Smart Contract mit der Signatur aufrufen darf. Jedoch gab es zur Zeit der Implementierung noch ein Problem in der neo-go Library mit `AllowedContracts` und `AllowedGroups`. Dadurch musste auf `transaction.Global` ausgewichen werden [46].

Zeile 16

Um den Contract zu deployen, werden die Parameter an den ManagementContract gesendet und die Methode `deploy` wird aufgerufen.

Zeile 17-18

Am Ende wird der gleiche Ablauf wie beim Auszahlungs-Aufruf ausgeführt, mit dem Unterschied, dass die Transaktion in einem Schritt unterzeichnet und versendet wird.

4.7.3 Tezos

Die Tezos-Anbindung wurde mit einem Node.js-Backend implementiert, basierend auf den Gründen welche in Punkt 4.8.3 erwähnt werden. Diese Library war zu Beginn als Library zur Kommunikation mit dem Temple-Wallet gedacht (siehe Entscheidung von Punkt 4.8.2). Das ganze Node.js-Backend wurde bereits so vorbereitet, dass auch andere zukünftige Kryptowährungen einfach implementiert werden können, wie dies bereits der Fall ist mit dem Go-Backend.

Die Auszahlungsanfragen laufen weiterhin zuerst über den Payout-Service in Go. Dies hat mehrere Vorteile. Zum einen wird die Duplikation der Logik von der Umwandlung von der Nano-Währung in die Smart Contract Währung verhindert. Um einen einheitlichen Ablauf der Auszahlungen zu gewährleisten, wird der Go Payout-Service als Ankerpunkt genutzt. Der Go-Service leitet die Anfragen, falls nötig, an die entsprechenden Services weiter.

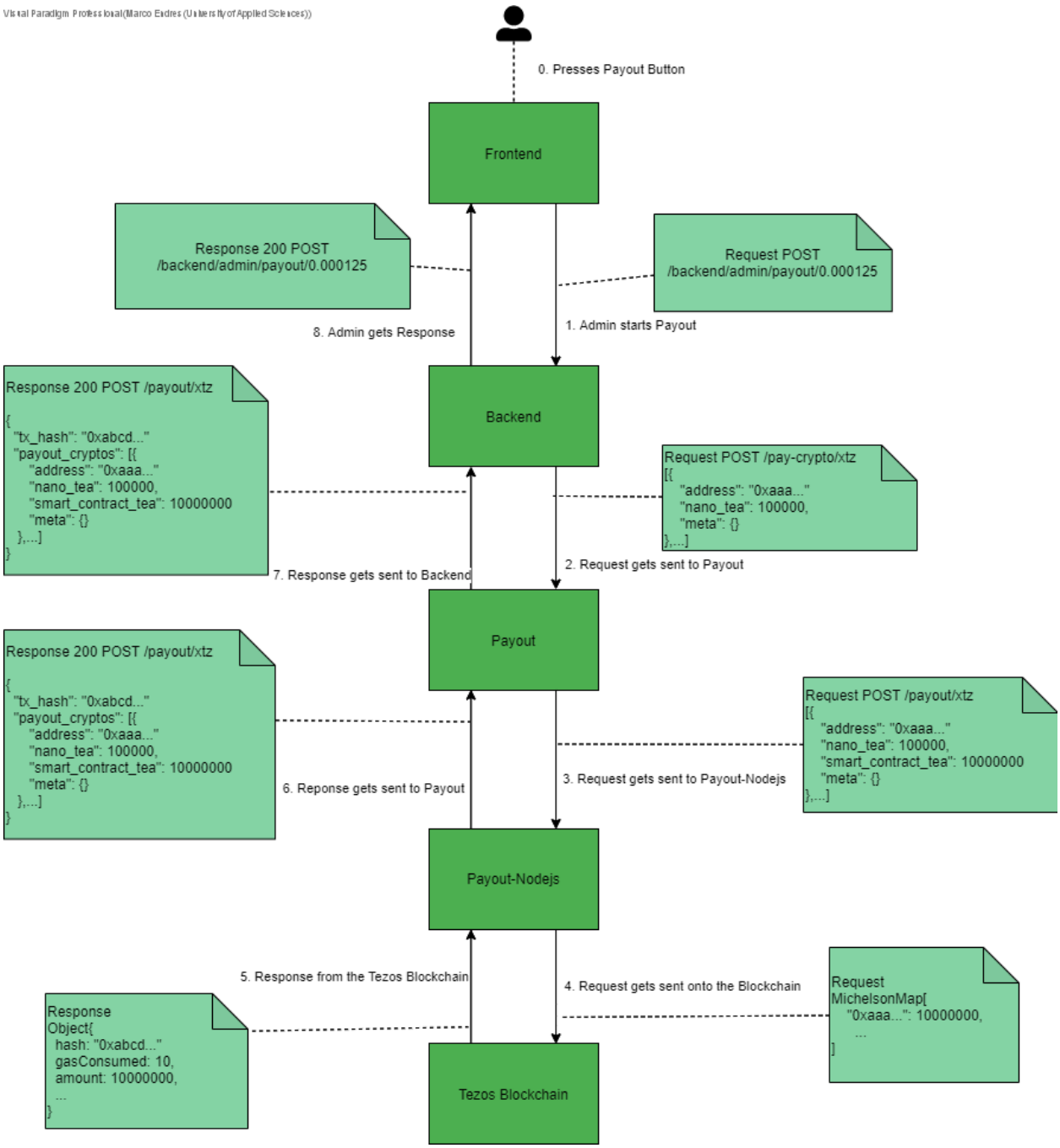


Abbildung 20: Payout Tezos Ablauf

Node.js

Die Kommunikation geschieht über einen REST-Service, welcher auf Express basiert [47]. Die Anbindung an den Smart Contract wurde mit der Open-Source-Library Taquito realisiert [48]. Da Tezos auf der Low-Level-Sprache Michelson basiert, wurde für die Smart Contract-Implementierung eine High-Level-Sprache ausgesucht. Zum Einsatz kam der TypeScript-Compiler von SmartPy [49]. Die Entscheidung fiel auf TypeScript, da die Erfahrung in den Web-Technologien höher ist als in Python. Ebenfalls überzeugte die ganze

Web-IDE von SmartPy. Sie bietet diverse initiale Codebeispiele an, ermöglicht es direkt zu kompilieren, zu testen und bietet eine direkte Möglichkeit an, den Smart Contract auf das Test- oder Produktivnetz von Tezos zu deployen.

Privates Netzwerk

Für das private Tezos-Netzwerk, auch Sandbox genannt, wurde Flextesa verwendet [50]. Flextesa kommt als Docker-Image daher und wurde ebenfalls im Docker-setup eingebaut, um so den Einstieg für weitere Entwickler in die Tezos Blockchain zu vereinfachen. Das Image besitzt zwei Wallets mit 2'000'000 Tezos und einer CLI mit welcher man mit der Blockchain interagieren kann

```
 $ docker exec my-sandbox $script info
Usable accounts:

- alice
  * edpkvGfYw3LyB1UcCahKQk4rF2tvbMUk8GFiTuMjL75uGXrpvKXhjn
  * tz1VSUr8wwNhLAzempoch5d6hLRiTh8Cjcjb
  * unencrypted:edsk3QoqBuvdamxouPhin7swCvkQNgq4jP5KZPbwWNnwdZpSpJiEbq
- bob
  * edpkurPsQ8eUApnLUJ9ZPDvu98E8VNj4KtJa1aZr16Cr5ow5VHKnz4
  * tz1aSkwEot3L2kmUvcoxzjMomb9mvBNuzFK6
  * unencrypted:edsk3RFfvaFaxbHx8BMtEW1rKQcPtDML3LXjNqMNLCzC3wLC1bWbAt

Root path (logs, chain data, etc.): /tmp/mini-box (inside container).
```

Abbildung 21: Flextesa

Die Einbindung in die FlatFeeStack-Applikation war anhand weniger Zeilen schnell implementiert.

```
01:  flextesa:
02:    container_name: FlatFeeStack-flextesa
03:    image: tqtezos/flextesa:20210602
04:    environment:
05:      - block_time=15
06:    ports:
07:      - 20000:20000
08:    command: flobox start
```

Die Parameter geben folgende Dinge an:

container_name	Wie heisst der Container
image	Auf welchem Docker-Image basiert der Service
environment	Welche Environment-Variablen gesetzt werden sollen. In diesem Fall soll alle 15 Sekunden 1 Block «gebaked» werden.
ports	Welche Ports gemappt werden sollen.
command	Welcher Command ausgeführt werden soll, wenn die Docker Instanz läuft.

Tabelle 4: Docker Konfiguration Beschreibung

Probleme mit Flextesa

Neben den Vorteilen von Flextesa gibt es zurzeit auch noch zwei offene Probleme, welche aufgetreten sind:

- Flextesa startet nur auf Prozessoren, welche den ADX-Instruktionsset unterstützen. Dies ist auf ein Problem in der Tezos Blockchain zurückzuführen [51].
- Flextesa braucht IPv6 Unterstützung. Ebenfalls ein Problem, welches in der Tezos-Chain festgestellt wurde [52, 53].

Taquito

Zur Kommunikation wurde, wie erwähnt Taquito, verwendet. Die Dokumentation, Beispiele und Community sind dabei positiv aufgefallen. Ebenfalls gibt es für Ethereum-Entwickler eine Seite mit den Unterschieden zur bekannten Ethereum-Library Web3js [54], um einen Einstieg in die Tezos Library zu vereinfachen [55].

Die Library ist dabei simpel und modern mit async /await-Syntax und einfachen Methodenaufrufen aufgebaut. Es folgen ein paar Beispiele:

Initialisierung:

```
01: const Tezos = new TezosToolkit(process.env.XTZ_URL);
02: Tezos.setProvider({
03:     signer: new InMemorySigner(process.env.XTZ_PRIVATE_KEY),
04: });
```

Initialisierung des Smart Contracts:

```
01: await Tezos.contract
02:   .originate({send tz
03:     code: contractCode,
04:     storage: {
05:       owner: await Tezos.signer.publicKeyHash(),
06:        teaMap : new MichelsonMap()
07:     },
08:   })
```

Contract holen:

```
01: let contract = await Tezos.contract.at(tezosSmartContract)
```

BatchPayout Aufruf auf dem Smart Contract:

```
01: let op = await contract.methods.batchPayout(storageMap).send();
```

Probleme mit Taquito

Neben den vielen Vorteilen von Taquito kam es beim lokalen Deployment mit Taquito auch zu Problemen. Durch das Zusammenspiel von Flextea und Taquito kann es zum Fall kommen, dass versucht wird auf eine leere Blockchain einen Smart Contract zu erstellen. Dabei wurde der Case nicht in der Taquito Bibliothek berücksichtig [56]. Basierend auf diesem Problem wurde ein Timeout zum Deployment eingebaut welches über eine Variable gesteuert werden kann. Bei einem lokalen Deployment wird dadurch empfohlen circa eine Minute zu warten, bis ein paar Blöcke gebackt wurden.

```
01: setTimeout(async () => {
02:   CONTRACT_ADDRESS = await deploySmartContract(Tezos)
03:   console.log('---------------------------------');
04:   console.log('Using contract address: ' + CONTRACT_ADDRESS);
05:   console.log('---------------------------------');
06: }, WAIT_BEFORE_DEPLOY)
```

Aufladen des Smart Contracts

Da das Aufladen des Contracts bei Tezos nicht direkt möglich ist, weder über den Temple Client noch über NOWPayments, musste zusätzlich ein REST-Endpoint implementiert werden. Dieser ermöglicht das einfache Transferieren von Tezos vom Besitzer-Wallet auf den Smart Contract. Ebenfalls wurde diskutiert, ob der Prozess automatisiert werden soll, dies wurde jedoch für den Moment abgelehnt und wird aktuell manuell gemacht. Da der Node.js-Endpunkt nur von lokal erreichbar ist und nicht vom reverse Proxy weitergeleitet wird, ist er vor öffentlichem Zugriff geschützt.

4.8 Entscheidungen

In diesem Kapitel werden wichtige Entscheidungen festgehalten, die mit dem Betreuer und allen Beteiligten besprochen wurden. Dabei geht es hauptsächlich um Abweichungen vom eigentlichen Plan, zusammen mit der Begründung.

4.8.1 NEO3 anstatt NEO 2.x (NEO Legacy)

In der Woche 5 wurde bei der Implementierung von NEO festgestellt, dass durch die Umstellung von NEO Legacy auf NEO3 ein besonderes Augenmerk daraufgelegt werden muss, was das jeweilige Produkt unterstützt. Am Anfang wurde geplant, dass FlatFeeStack mit NEO Legacy implementiert werden soll, da NOWPayments nur die alte Blockchain unterstützt. Neben den offenen Fragen, über den Ablauf einer möglichen Migration, kamen noch technische Probleme hinzu:

Technische Probleme

1. Synchronisierung der Blockchain stoppt bei einem zufälligen Block
2. Der Faucet für NEO 2.x funktioniert nicht
3. Das ausgewählte Browser-Wallet (NEOLine) unterstützt Private Netzwerke nur für NEO3

Zukünftige Probleme

1. Wann soll NEO3 implementiert werden?

Offene Fragen für die Zukunft

1. Wie sieht eine Migration seitens FlatFeeStack von den bestehenden NEOs aus?

Ebenfalls wurde mit Leuten aus der NEO-Community auf Discord gesprochen und viele waren irritiert, warum FlatFeeStack es mit der alten Blockchain noch betreiben will [57].

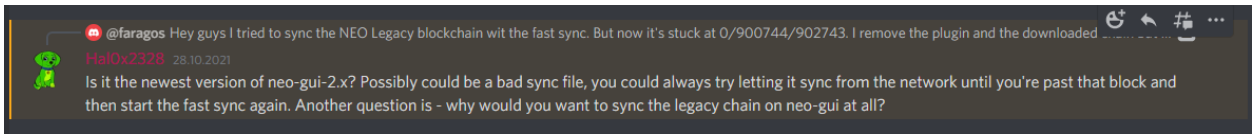


Abbildung 22: Warum Neo 2.x auf Discord 1/2

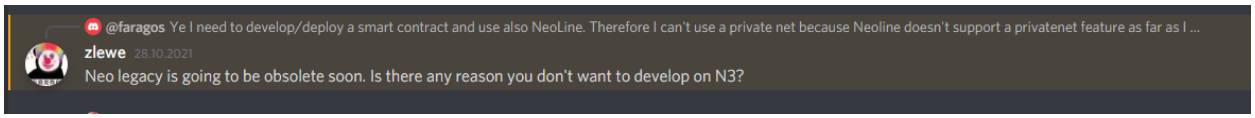


Abbildung 23: Warum Neo 2.x auf Discord 2/2

Basierend auf dem Feedback der Community, den offenen Fragen und Problemen wurde sich in der Woche 6 dafür entschieden NEO3 so gut wie möglich anzubinden, um für die Zukunft gewappnet zu sein. Eine komplette Anbindung wird dadurch verunmöglicht, da die Einzahlung mit NOWPayments nur mit NEO Legacy möglich ist.

4.8.2 Nur Batch Payout wird implementiert

Neben der Studienarbeit arbeitete Michael Bucher an der Optimierung der Smart Contracts für seine Master-Thesis. In Woche 7 wurde das erste Mal die zukünftige API der Smart Contract vorgestellt. Basierend darauf wurde erkannt, dass sich der Smart Contract erheblich ändert und neu für die selbstbestimmte Auszahlung eines einzelnen Users eine Backend-API benötigt wird. Diese API bestand zu diesem Zeitpunkt nicht. Damit die Blockchain Anbindung nicht von der Arbeit von Michael abhängte, wurde entschieden nur den Batch-Payout des neuen Smart Contracts zu implementieren, da dieser der Implementierung der alten Variante gleicht und ohne grösseren Aufwand implementiert werden konnte.

Durch diesen Entscheid wurde die Integration der Browser Wallets hinfällig. Die vorbereitete Implementierung wird nicht verwendet, blieb jedoch bestehen für eine spätere geplante Anbindung.

4.8.3 Batch Payout in Tezos wird mit einem Node.js backend realisiert

In Woche 8 wurde nach einer Tezos Bibliothek in Go recherchiert. Dabei stellte sich heraus, dass es keine gepflegte Library gibt in Go gibt, welche auf der Tezos Blockchain mit Smart Contracts kommunizieren kann. Es wurde dabei auf dem Reddit-Subreddit [58] und danach noch genauer bei den einzelnen Libraries nachgefragt.

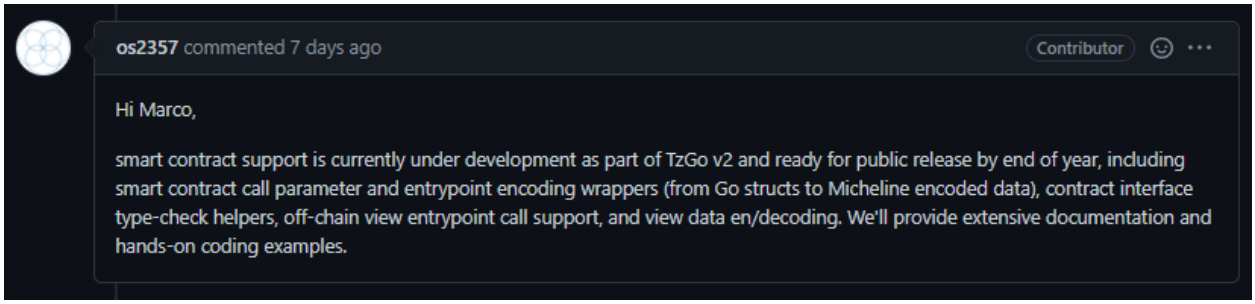


Abbildung 24: Go Library für die Kommunikation mit dem Smart Contract

Es wurde daraufhin in Woche 8 entschieden ein Node.js-backend zu realisieren und mit der bereits bekannten Library Taquito zu realisieren, welche bei der Wallet-Analyse, für die die Wallet-Anbindung angedacht war.

4.8.4 Übernahme der Gebühren von FlatFeeStack

In der Woche 7 wurde besprochen wie FlatFeeStack mit den Gebühren umgeht. Die Transaktionskosten bei der Einzahlung durch das Blockchain Netzwerk und NOWPayments, werden vom Sponsorenbetrag abgezogen. Bezahlt ein Sponsor zum Beispiel 25 XTZ ein und es wird insgesamt 1 XTZ an Gebühren abgezogen, wird ihm 24 XTZ gutgeschrieben. Das Guthaben reicht für jeweils ein Jahr oder drei Monate, je nachdem welcher Plan ausgewählt wurde.

Zusätzlich entstehen Gebühren bei der Ausführung eines Smart Contracts und von manuellen Überweisungen. Die Kosten für die Ausführung eines Smart Contracts und einer manuellen Überweisung, können

bei den täglichen Berechnungen nicht ohne grossen Mehraufwand berücksichtig werden. Deshalb wurde in einem ersten Schritt entschieden, dass FlatFeeStack diese übernimmt.

4.9 Migration

Ein Teil der Arbeit ist es die neuen Änderungen auf das Test- und Produktivsystem zu deployen. Bevor die Änderungen ausgeliefert werden können, muss die Datenbank und die Daten migriert werden. Für die Datenbank Migration wurde ein kleines SQL Skript geschrieben, welches die nötigen Änderungen vornimmt. Das Skript wurde so entwickelt, dass es für die Produktiv- und Testumgebung verwendet werden kann.

Welche Änderungen nimmt das Skript vor:

1. Die Spalte currency wird in allen Tabellen hinzugefügt, wo es nötig ist. Dabei wird für die bestehenden Daten der Wert «USD» eingefügt.
2. Die unique indexe werden angepasst
3. Die Tabellen daily_repo_hours, payout_request, payout_response und payout_response_details werden gelöscht. Die Payout Tabellen haben keine Daten und die daily_repo_hours Daten können einfach gelöscht werden.
4. Die Spalte payout_eth wird gelöscht
5. Die neuen Tabellen werden erstellt
6. Migration der daily_payment Daten für USD

4.10 Deployment

Das Deployment wurde lokal vorbereitet und erfolgreich auf der Testumgebung durchgeführt. Durch die Eigenschaften der Blockchain und den fehlenden Tests, wurde bewusst auf die Liveschaltung der Änderungen auf das Produktivsystem verzichtet, da ein irreversibler finanzieller Schaden entstehen kann.

4.10.1 Kryptowährungen

Für das Deployment der Kryptowährungen wurden die implementierten Deploy-Methoden genutzt. Dafür wurden zuerst die Wallets aufgeladen für die einzelnen Kryptowährungen, danach die `.env`-Variablen (Private Key und Deploy) gesetzt. Nach einem erfolgreichen Deployment des Contracts wurden die Adressen des Contracts ebenfalls eingetragen und das Deployment abgeschaltet. Eine Schritt-für-Schritt-Anleitung existiert im Anhang.

4.11 Weitere Implementierungen

4.11.1 FlatFeeStack Bundling als Vorbild für Tezos Library

Am Anfang war das Ziel, dass sich die User den Betrag selbst auszahlen können. Darum wurde für jede Blockchain ein entsprechendes Wallet im Browser mit API untersucht.

Bei Tezos wurde sich für das Temple-Wallet entschieden, da es das weit verbeitestet Wallet für Tezos ist. Für die Anbindung des Wallets wurde sich für Taquito entschieden, da die Doku ausführlich und beliebt in der Wallet Anbindung für Temple ist [59].

Bei der Implementierung kam es jedoch zu einem Problem mit dem Rollup-Bundler (Details können aus dem GitHub-Issue entnommen werden) [60]. Als das Problem gelöst wurde, ist der Lösungsweg dem Ersteller des Tickets mitgeteilt worden. Es stellte sicher heraus, dass er ein Entwickler von Taquito ist. Dieser war so begeistert von der Lösung, dass er sie austestete und diese als Standardsetup für Taquito, Svelte und Rollup [61] verwenden will.

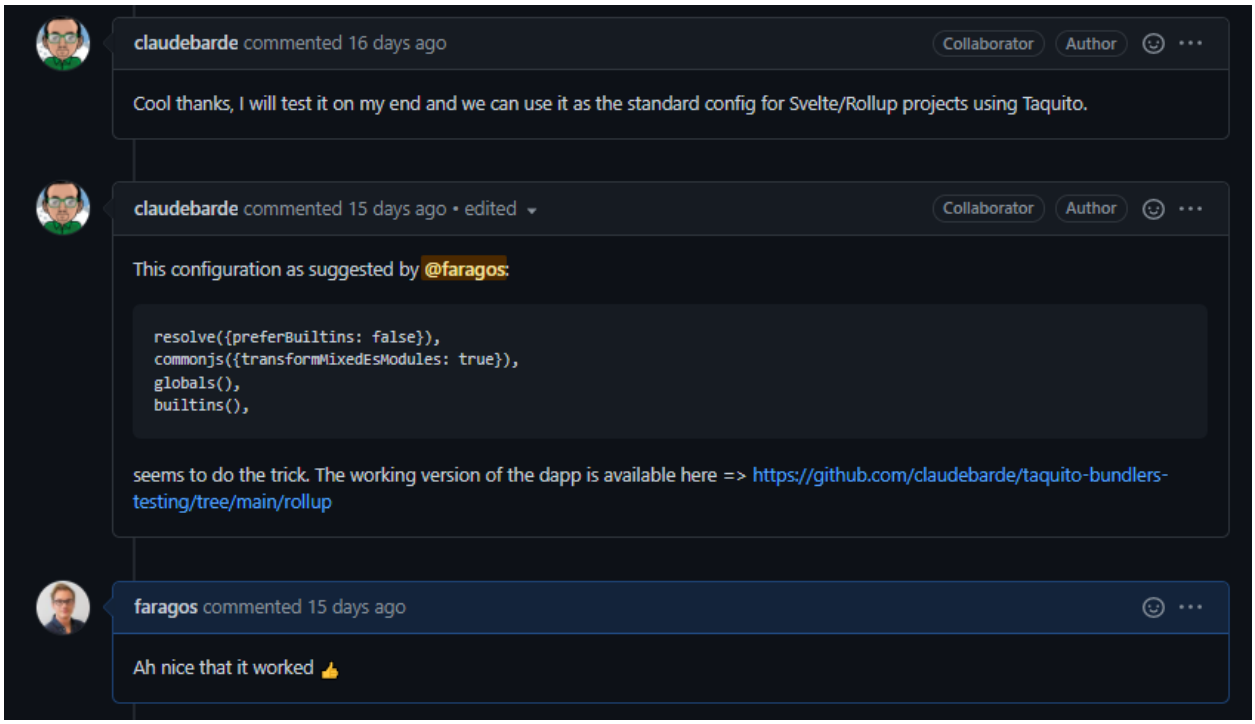


Abbildung 25: Bundler-issues gelöst und als Standardkonfiguration in Taquito benutzen

5. Ergebnisse

5.1 Testing

5.1.1 Einzahlung

Da NOWPayments keine Anbindung an die Testnetze der Blockchains anbietet, sind automatisierte Tests schwer umzusetzen. Um dennoch die Einzahlung zu testen, wurden die WebHooks simuliert. So ist es möglich über das Web Interface eine Einzahlung zu starten und den WebHook zum Beispiel über Postman zu simulieren. Dabei kann man im Body die nötigen Daten eingeben und überprüfen, ob das Guthaben beim Sponsor richtig angerechnet wird.

5.1.2 Tägliche Berechnungen

Um die Täglichen Berechnungen zu testen, wurde ein Testszenario erstellt. Es existieren 5 Sponsoren, 4 Projekte und 4 Contributoren. Welcher Sponsor welches Projekt unterstützt und welcher Contributor wie viel Prozent am Projekt beigetragen hat, ist in der untenstehenden Grafik «Abbildung 25: Tägliche Berechnungen Test Szenario» ersichtlich. Zusätzlich hat jeder Sponsor an einem unterschiedlichen Tag begonnen mit dem Favorisieren der Projekte.

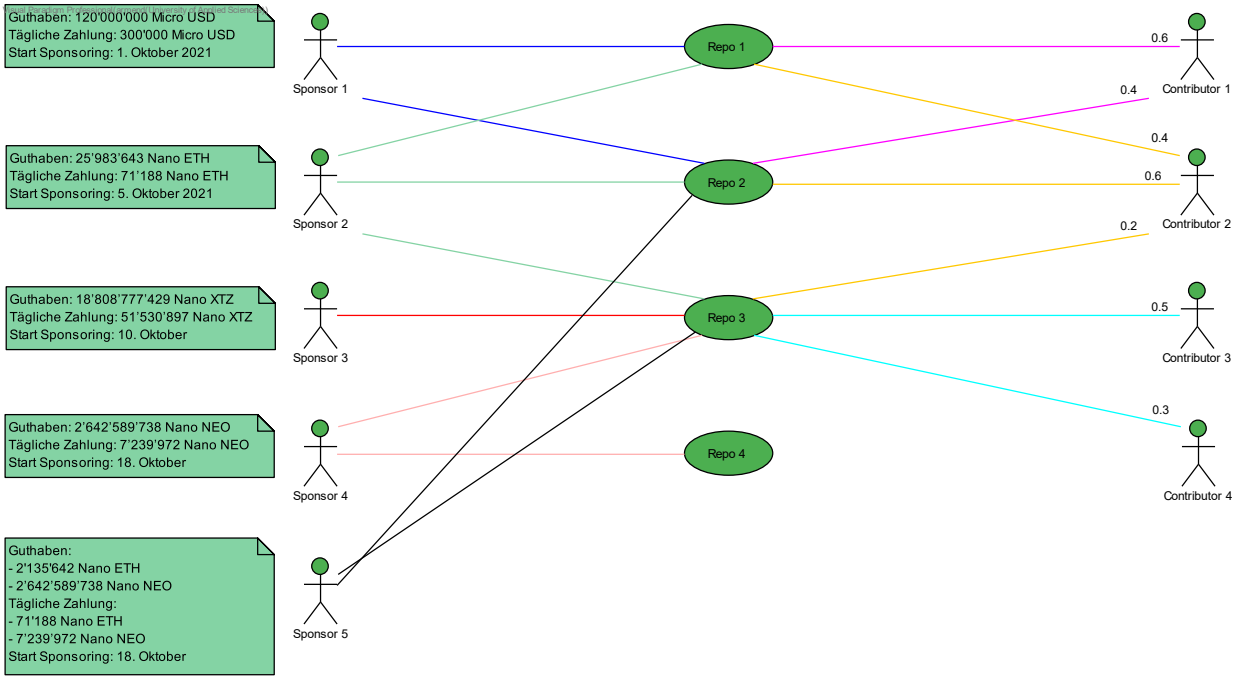
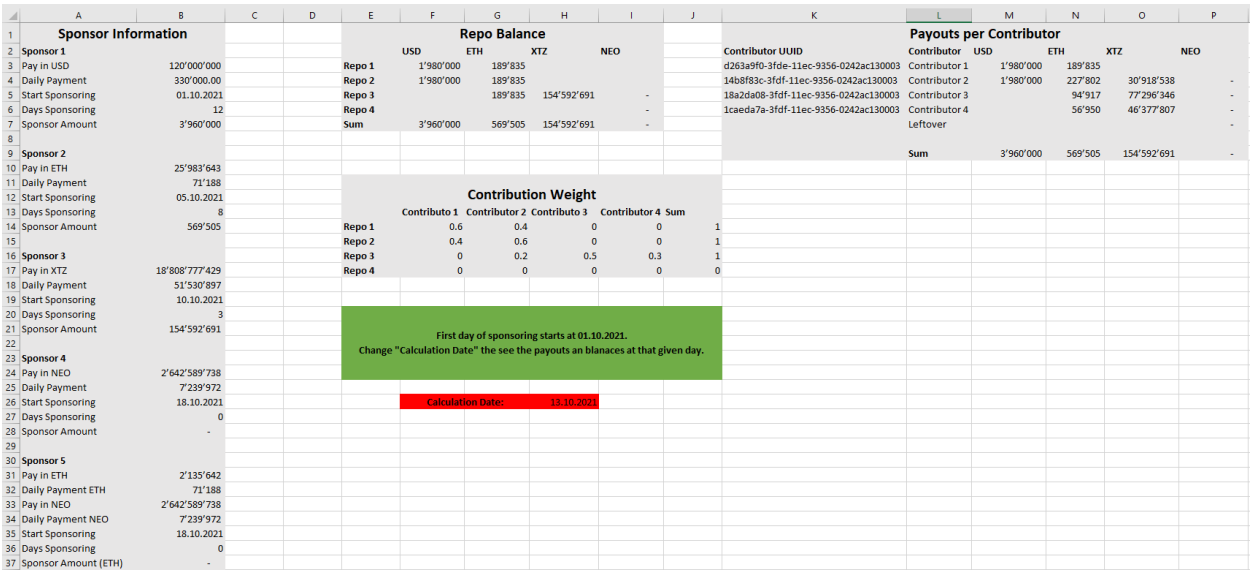


Abbildung 26: Tägliche Berechnungen Test Szenario

Für das Szenario wurden Testdaten für die Datenbank erstellt, welche importiert werden können. Zusätzlich gibt es ein Excel, mit dem es möglich ist die Berechnungen zu überprüfen. Es wurde so weit vorbereitet, dass nur noch der Stichtag angepasst werden muss. Es wird einem angezeigt welches Projekt und welcher Contributor wie viel Geld an diesem Tag haben sollte. Um die Berechnung auf dem System zu

starten, wurde für das lokale Testen ein neuer Endpunkt erstellt, welcher ein Datum entgegennimmt und für dieses Datum die Berechnungen durchführt. So können die Berechnungen vom 1. Oktober bis zum 10. Oktober ausgeführt werden und im Excel kann der Stichtag 10. Oktober eingetragen werden. Danach kann der Stand der Datenbank mit dem Excel überprüft werden.



Sponsor Information

Sponsor 1	
Pay in USD	120'000'000
Daily Payment	330'000.00
Start Sponsoring	01.10.2021
Days Sponsoring	12
Sponsor Amount	3'960'000
Sponsor 2	
Pay in ETH	25'983'643
Daily Payment	71'188
Start Sponsoring	05.10.2021
Days Sponsoring	8
Sponsor Amount	569'505
Sponsor 3	
Pay in XTZ	18'808'777'429
Daily Payment	51'530'897
Start Sponsoring	10.10.2021
Days Sponsoring	3
Sponsor Amount	154'592'691
Sponsor 4	
Pay in NEO	2'642'589'738
Daily Payment	7'239'972
Start Sponsoring	18.10.2021
Days Sponsoring	0
Sponsor Amount	-
Sponsor 5	
Pay in ETH	2'135'642
Daily Payment ETH	71'188
Pay in NEO	2'642'589'738
Daily Payment NEO	7'239'972
Start Sponsoring	18.10.2021
Days Sponsoring	0
Sponsor Amount (ETH)	-

Repo Balance

	USD	ETH	XTZ	NEO
Repo 1	1'980'000	189'835		
Repo 2	1'980'000	189'835		
Repo 3		189'835	154'592'691	-
Repo 4				-
Sum	3'960'000	569'505	154'592'691	-

Contribution Weight

	Contributo 1	Contributor 2	Contributo 3	Contributor 4	Sum
Repo 1	0.6	0.4	0	0	1
Repo 2	0.4	0.6	0	0	1
Repo 3	0	0.2	0.5	0.3	1
Repo 4	0	0	0	0	0

First day of sponsoring starts at 01.10.2021.
Change "Calculation Date" the see the payouts an blanaces at that given day.

Calculation Date: 13.10.2021

Payouts per Contributor

Contributor UUID	Contributor	USD	ETH	XTZ	NEO
d263a9f0-3fde-11ec-9356-0242ac130003	Contributor 1	1'980'000	189'835		
14b8f83c-3fdf-11ec-9356-0242ac130003	Contributor 2	1'980'000	227'802	30'918'538	-
18a2da08-3fdf-11ec-9356-0242ac130003	Contributor 3		94'917	77'296'346	-
1caeda7a-3fdf-11ec-9356-0242ac130003	Contributor 4		56'950	46'377'807	-
	Leftover				-
	Sum	3'960'000	569'505	154'592'691	-

Abbildung 27: Tägliche Berechnungen Test Excel

5.1.3 Auszahlung

Für das Testen der Auszahlung, wurden die Smart Contracts auf die Testnetze ausgeliefert und Test-Accounts erstellt. Diese Accounts wurden den Contributor 1 bis Contributor 4 zugewiesen. Danach wurden das Initiale Setup von den täglichen Berechnungen aufgesetzt und es wurde die Berechnung für ein paar Tage ausgeführt. Wurde der Auszahlungsprozess gestartet, konnte man anhand vom Test Excel überprüfen ob die einzelnen Accounts den richtigen Betrag erhalten haben.

5.1.4 Smart Contracts

Der Ethereum- und NEO-Smart Contracts wurden ausserhalb dieser Arbeit entwickelt und optimiert. Dadurch wurde das Testing ebenfalls in diese Projekte ausgelagert, um Duplikation und Redundanz zu vermeiden.

Tezos Smart Contract

Der Tezos Smart Contract wurde mittels eingebautem Testing Framework von SmartPy.io getestet, welche Assign-Methoden anbietet und diese gegen den Smart Contract prüft. Diese Tests werden bei der Kompilation des Smart Contracts geprüft. Dabei können Methoden einfach ohne viel zusätzlichen Code aufgerufen und validiert werden.

```
Dev.test( string: { name: 'Flatfeestack' }, () => {
    const c1 = Scenario.originate(new Flatfeestack( storage: {owner
    Scenario.transfer(c1.deposit( params: { amount: 1000 }), {amou
    Scenario.transfer(c1.batchPayout( params: {devTeaMap: [['tz1aS
    Scenario.transfer(c1.changeOwner( params: {newOwnerAddress: 't
    //Scenario.isFailing(Scenario.transfer(c1.batchPayout({devTe
    Scenario.transfer(c1.batchPayout( params: {devTeaMap: [['tz1aS
});
```

Abbildung 28: Tezos Tests

Es wurde keine Möglichkeit gefunden einen Negativtest durchzuführen, welcher fehlschlägt, wenn er fehlschlagen muss. Durch das Fehlschlagen wird auch die Kompilation verhindert. Ein solcher Test wurde implementiert, aber auskommentiert und eine Kompilation weiterhin zu ermöglichen.

Abbildung 29: Fehlermeldung Szenario

5.1.5 Testumgebung

Um nach dem Deployment auf der Testumgebung zu überprüfen, ob sich die Applikation richtig verhält, wurde ein ähnliches Szenario wie im Abschnitt «5.1.2 Tägliche Berechnungen» erstellt. Dabei wurden die gleichen Sponsoren und Contributoren erstellt, mit dem Unterschied das die Einzahlungen der Sponsoren über das Web Interface erstellt wurden, inklusive Simulation des NOWPayments WebHooks. Auf GitHub wurden vier Projekte inklusive den Commits mit den E-Mail-Adressen der Contributoren erstellt.

Um für die täglichen Berechnungen nicht tagelang warten zu müssen, wurde eine existierte Zeitsprung-Funktionalität benutzt. Sie erlaubt das Ändern der Systemzeit. So besteht die Möglichkeit 24 Stunden in die Zukunft zu springen, damit die Berechnungen gestartet werden können.

Die Auszahlungen wurden über das Web Interface auf den jeweiligen Testnetzen getestet.

5.2 Schlussfolgerungen

Die Integration der Kryptowährungen ETH, NEO und XTZ als neue Einzahlungsmethode konnte erfolgreich abgeschlossen werden. Zusätzlich konnten die neuen Änderungen auf die Testumgebung ausgeliefert werden. Durch die Absenz der automatisierten Tests wurde bewusst auf ein produktives Deployment verzichtet, da durch die Eigenschaften der Blockchains ein irreversibler finanzieller Schaden entstehen kann. Es wird empfohlen zuerst die benötigte Test-Infrastruktur aufzubauen, um den fehlerfreien Betrieb zu garantieren.

Bei der Recherche von möglichen Einzahlungsvarianten wurde klar, dass es keine Lösung gibt, die alle Bedürfnisse vollumfänglich zufriedenstellt. Durch die Analyse hat sich herauskristallisiert, dass NOWPayments, trotz der Schwächen im Bereich Testing und NEO3 Integration, für den aktuellen Use Case die Anforderungen am besten abdeckt. Diese Limitationen wurden aus Zeitgründen für diese Arbeit in Kauf genommen.

Die eingesetzten Blockchains unterscheiden sich stark bei ihrem Tooling und Dokumentation. Bekannte Blockchain wie Ethereum sind gut dokumentiert und die Integration gestaltet sich einfach. Weniger bekannte Blockchain wie NEO oder Tezos haben eine steilere Lernkurve, da die Unterschiede meist im Detail liegen und weniger Ressourcen im Internet zur Verfügung stehen. Der Grund für diese Unterschiede und dem fehlenden Standard, kann durch das junge Alter der Technologie erklärt werden.

5.3 Ausblick

5.3.1 NOWPayments Ablösung

NOWPayments bietet eine simple Lösung, die einfach zu Nutzen ist. Es werden fast alle Bedürfnisse abgedeckt. Ein grosser Nachteil ist jedoch, dass sie keine Anbindung an die Blockchain Testnetze anbieten. Aus diesem Grund müssen System- oder Einzahlungstests immer mit echtem Geld getätigt werden oder der NOWPayments Service muss simuliert werden. Ein weiteres Problem ist, dass NEO3 noch nicht unterstützt wird. Beide Lösungen sind nicht zufriedenstellend, deshalb könnte in Zukunft der NOWPayments Service durch eine bessere Alternative abgelöst werden. Ob es in Zukunft einen besseren Anbieter geben oder der Service selbst implementiert wird, bliebt offen.

5.3.2 Automatisierte Tests

FlatFeeStack besitzt aktuell wenige automatisierte Tests. Das Erweitern der Applikation ohne Test ist gefährlich, da es kein Feedback gibt, falls ein Applikationsteil brechen würde. Damit das Erweitern in Zukunft einfacher wird, sollten automatisierte Tests hinzugefügt werden.

5.3.3 USD Einzahlung automatisieren

Aktuell werden Einzahlungen in USD über Stripe getätigt und werden später über einen manuellen Prozess auf eine Handelsplattform für Kryptowährungen überwiesen und zu ETH umgewandelt. Dieser Schritt könnte automatisiert werden. Zum Beispiel bietet NOWPayments eine solche Lösung an. Auch wenn für die Einzahlung mit einer Kryptowährung NOWPayments nicht optimal ist, da sie keine Testnetz anbieten,

könnten sie für diesen einzelnen Use Case genutzt werden. Weitere Lösungsmöglichkeiten wurden nicht weiter recherchiert.

5.3.4 Node.js TypeScript Implementation

Durch das iterative Vorgehen von Woche zu Woche wurde der heutige Node.js in reinem JavaScript geschrieben. Eine mögliche Verbesserung wäre es den Service mit TypeScript zu schreiben. Dies hätte aber auch zur Folge, dass es einen Build Prozess braucht, welcher das TypeScript in JavaScript umwandelt. Eine weitere Lösung wäre es Node.js ganz durch Deno, eine moderne Runtime für JavaScript und TypeScript, zu ersetzen und sich den Build Prozess zu sparen [62].

5.3.5 Ablösung von der Währung NEO durch GAS

Während der Arbeit wurde, jeweils von NEO als neues Bezahlungsmittel gesprochen. Dabei wurde nie explizit erwähnt, dass nicht NEO selbst sondern GAS ausbezahlt werden soll, da NEO nicht teilbar ist. Dadurch entstand das Missverständnis, dass dazu führte, das NEO, die Währung selbst, als Auszahlungswährung implementiert wurde. Dies ist nun inkompatibel mit dem optimierten Smart Contract. Ebenfalls kann nur NEO selbst einbezahlt werden und kein GAS. Darum muss zurzeit die Umwandlung von NEO zu GAS manuell stattfinden. Eine Verbesserung für die Zukunft wäre die Implementierung umzustellen, dass GAS anstatt NEO als Mittel zur Zahlung ausgewählt werden kann. Da NOWPayments GAS bereits unterstützt, ist die Umstellung problemlos möglich. Ebenfalls muss die Auszahlung von NEO auf GAS umgestellt werden.

6. Glossar

.env-Datei wird benötigt, um Environment-spezifische Variablen zu setzen.

51% Attacke ist eine Attacke, die durch 51% der Ressourcen vom Netzwerk erzielt werden kann. Dies ist abhängig vom Konsensmechanismus. Es destabilisiert jedoch die Währung und eine Fairness ist nicht mehr garantiert, da eine Partei die Oberhand hat.

Application Binary Interface (ABI) wird benötigt, damit zwei Programme kommunizieren können.

Baker ist ein Miner bei Tezos.

Browser Extension ist eine Applikation, die auf einen Browser installiert werden kann und unabhängig von der Webseite läuft, jedoch mit dieser interagieren kann.

Coins sind die einzelnen Geldstücke einer Kryptowährung.

Command Line Interface (CLI) ist ein Programm, welches ohne GUI auskommt und mit welchem nur über die Kommandozeile interagiert wird.

Contributor ist ein bei FlatFeeStack registrierter Benutzer, welcher an Open-Source Projekten mitgearbeitet hat.

Cronjob ist ein Task, der in regelmässigen Zeitabständen automatisiert ausgeführt wird.

Docker Container ist eine Standard-Softwareeinheit, die den Code und alle seine Abhängigkeiten zusammenfasst, damit die Applikation überall schnell hochgefahren werden kann [63].

ETH ist die Währung bei Ethereum.

Faucet ist ein Geldgeber bei Blockchains. Ein solcher wird meistens benötigt, um auf den Testnetzen Kryptowährung zu erhalten, ohne etwas bezahlen zu müssen.

Fiat Währung ist ein Zahlungsmittel, welches von einer Regierung oder einem Staat erschaffen wird, wie zum Beispiel CHF, EUR und USD.

(Transaktions-)Hash wird benutzt, um Transaktionen auf der Blockchain benennen zu können. Jede Transaktion generiert einen Transaktionshash, welcher benutzt werden kann, um die Transaktion auf diversen Seiten wiederzufinden und eindeutig zu Identifizieren**.**

Initial coin offering (ICO) ist ein Initiales Anbieten einer Kryptowährung für die Finanzierung.

Invoice ist eine Rechnung. Diese Rechnung wird für die Kryptowährungen über den Service für NOWPayments generiert.

JSON Web Token wird benutzt, um Informationen sicher zwischen verschiedenen Parteien als JSON Objekt zu senden [20].

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

Microservice ist ein Service, welcher eine spezielle Aufgabe gut lösen kann. Zusätzlich kann er eigenständig erweitert und skaliert werden, ohne das andere Services betroffen sind [64].

Micro USD wird als Einheit innerhalb der Applikation genutzt. 1 Micro USD = 1/1'000'000 USD

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

Mining wird der Prozess genannt, bei welchem versucht wird das Kryptographische Puzzle zu lösen.

NEO ist die Währung selbst, sowie die Blockchain NEO selbst.

NEO2/NEO Legacy ist die alte NEO-Blockchain

NEO3/N3 ist die neue NEO-Blockchain

NOWPayments ist der Online Bezahldienst, welcher für die Einzahlung mit Kryptowährungen genutzt wird.

Open-Source Software ist eine Applikation, bei welcher der Source-Code der Öffentlichkeit zugänglich gemacht wurde.

Organisation ist die Rolle, die eine Firma oder Abteilung erhält beim Registrieren. Sie kann die Kosten für das Sponsoring der Mitarbeiter übernehmen.

Payment covering ist eine Option bei NOWPayments. Sie erlaubt es einen prozentualen Toleranzwert für eine Einzahlung zu definieren. So können Einzahlungen akzeptiert werden, bei denen weniger einbezahlt wurde.

Payment Cycle ist das Abonnement eines Sponsors. Damit sind Informationen zu finden wie, vorhandenes Guthaben und verfügbare Tage.

Privates Netzwerk ist ein Netzwerk, bei welchem die volle Kontrolle beim Betreiber liegt, da es nur lokal läuft. Durch ein solches Netz wird die Entwicklung auf der Blockchain vereinfacht.

Proof of Work (PoW) ist ein Konsensalgorithmus, bei dem es darum geht, möglichst schnell ein Kryptographisches Puzzle zu lösen. Die Fairness wird dadurch gewährleistet, dass keine Partei 51% der Rechenleistung besitzt.

Proof of Stake (PoS) ist ein Konsensalgorithmus, bei dem ausgewählte Blockchain Teilnehmer die Transaktionen überprüfen. Diese Teilnehmer entledigten sich zuvor einem Teil ihrer Coins. Diese werden im Falle einer Fehlvalidierung dem Teilnehmer entnommen. Die Chance zu validieren, steigt proportional zu den entledigten Coins. Die Fairness wird dadurch gewährleistet, dass eine Person 51% der Währung haben müsste, was die Währung destabilisieren würde und einen Verlust für den Angreifer nach sich ziehen würde.

Reverse Proxy stellt eine oder mehrere interne Ressourcen für externe Clients zur Verfügung.

Seat ist die Definition eines Mitarbeiters. Eine Organisation kann sich Seats kaufen und den Mitarbeitern zur Verfügung stellen. So können Mitarbeiter auf die Kosten der Firma ihre Lieblingsprojekte unterstützten.

Smart Contract ist ein Programm, welches auf der Blockchain ausgeführt wird.

Solidity ist die Programmiersprache für Smart Contracts bei Ethereum.

Sponsor ist ein Benutzer, welcher einbezahlt hat und Projekte favorisiert.

Stable Coin ist eine Digitale Währung, die an eine Fiat Währung gekoppelt ist.

Stripe ist ein Online Bezahldienst für Fiat Währungen.

Temple-Wallet ist eine Browser Extension, damit ein User auf einer Seite mit Tezos bezahlen kann.

Testnetz ist ein Blockchain-Netz, welches nicht für den produktiven Betreib gedacht ist, sich jedoch gleich verhält. Es wird benutzt, um Smart Contracts vor der produktiven Nutzung zu testen. Auf dem Testnetz kann ebenfalls kostenfrei die Währung der jeweiligen Blockchain erworben werden.

Tezos ist eine Kryptowährung.

Total Earned Amount (TEA) ist die Summe aller einnahmen von einem Contributor.

unique index ist in der Datenbank ein eindeutiger Index. Dieser kann nicht doppelt vorhanden sein.

User Balance ist das Guthaben eines Benutzers.

Wallet ist die Geldbörse bei Kryptowährungen.

WebHook ist eine Methode wie zwei Server miteinander kommunizieren können. Dabei werden URL Endpunkte aufgerufen, sobald ein Ereignis stattgefunden hat.

WebSocket ist ein Protokoll, welches eine beidseitige Kommunikation ermöglicht.

WEI ist die kleinste Währung bei Ethereum. 1 ETH sind 10^{18} Wei.

XTZ ist die Währung bei Tezos.

7. Abbildungsverzeichnis

8. Tabellenverzeichnis

9. Literaturverzeichnis

[1] GitHub, *Build software better, together.* [Online]. Available: https://github.com/about (accessed: Dec. 22 2021).
[2] M. Chandel, "Why You Should Never Use Float and Double for Monetary Calculations," *DZone*, 21 Aug., 2018. https://dzone.com/articles/never-use-float-and-double-for-monetary-calculatio (accessed: Dec. 22 2021).
[3] *MetaMask - A crypto wallet & gateway to blockchain apps.* [Online]. Available: https://metamask.io/ (accessed: Dec. 12 2021).
[4] *Bitcoin Block Reward Halving Countdown.* [Online]. Available: https://www.bitcoinblockhalf.com/ (accessed: Dec. 5 2021).
[5] Digiconomist, *Bitcoin Energy Consumption Index - Digiconomist.* [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption/ (accessed: Dec. 15 2021).
[6] *Peercoin whitepaper - whitepaper.io.* [Online]. Available: https://whitepaper.io/document/139/peercoin-whitepaper (accessed: Dec. 22 2021).
[7] Bitcoin Suisse, *Was ist Proof-of-Stake? | Bitcoin Suisse.* [Online]. Available: https://www.bitcoinsuisse.com/de/fundamentals/was-ist-proof-of-stake (accessed: Dec. 5 2021).
[8] CoinMarketCap, *Cryptocurrency Prices, Charts And Market Capitalizations | CoinMarketCap.* [Online]. Available: https://coinmarketcap.com/ (accessed: Oct. 22 2021).
[9] M. Sigalos, "Ethereum had a rough September. Here's why and how it's being fixed," *CNBC*, 10 Feb., 2021 (accessed: Oct. 22 2021).
[10] *Proof-of-stake (PoS) | ethereum.org.* [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (accessed: Dec. 4 2021).
[11] S. N. Khan, F. Loukil, C. Ghedira-Guegan, E. Benkhelifa, and A. Bani-Hani, "Blockchain smart contracts: Applications, challenges, and future trends," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 5, pp. 1–25, 2021, doi: 10.1007/s12083-021-01127-0.
[12] B. Toti, "Was ist NEO? Alles, was Sie wissen müssen| Coin-Hero," 15 Aug., 2017. https://coin-hero.de/neo/ (accessed: Sep. 27 2021).
[13] P. Schmitz, "Was ist NEO?," *Blockchain-Insider*, 02 Sep., 2021. https://www.blockchain-insider.de/was-ist-neo-a-998702/ (accessed: Sep. 27 2021).
[14] Neo-project, *Neo Smart Economy.* [Online]. Available: https://neo.org/ (accessed: Dec. 5 2021).
[15] Bitcoin Suisse, *Neo Blockchain: Was kommt als Nächstes? | Bitcoin Suisse.* [Online]. Available: https://www.bitcoinsuisse.com/de/outlook/neo-blockchain-was-kommt-als-naechstes (accessed: Dec. 5 2021).
[16] "Tezos: Wer steckt hinter der noch jungen Kryptowährung und was macht sie so beliebt?," *finanzen.net*, 2021-11-16T6:14:0+02:00, 2021-11-16T6:14:0+02:00. https://www.finanzen.net/nachricht/devisen/besonders-sicher-tezos-wer-steckt-hinter-der-noch-jungen-kryptowaehrung-und-was-macht-sie-so-beliebt-8921539 (accessed: Dec. 5 2021).
[17] *Michelson.* [Online]. Available: https://wiki.tezosagora.org/learn/smartcontracts/michelson (accessed: Nov. 17 2021).
[18] *Michelson: the language of Smart Contracts in Tezos — Tezos (master branch, 2021/11/17 13:50) documentation.* [Online]. Available: https://tezos.gitlab.io/active/michelson.html (accessed: Nov. 17 2021).

[19] *Svelte • Cybernetically enhanced web apps.* [Online]. Available: https://svelte.dev/ (accessed: Nov. 17 2021).
[20] Auth0.com, *JWT.IO - JSON Web Tokens Introduction.* [Online]. Available: https://jwt.io/introduction (accessed: Dec. 12 2021).
[21] *Online-Zahlungsabwicklung für Internetunternehmen – Stripe.* [Online]. Available: https://stripe.com/de-ch (accessed: Dec. 23 2021).
[22] *Email Delivery Service | SendGrid.* [Online]. Available: https://sendgrid.com/ (accessed: Dec. 22 2021).
[23] *Supported Cryptocurrencies For Each Service - CoinGate.* [Online]. Available: https://coingate.com/supported-currencies (accessed: Nov. 20 2021).
[24] CoinGate, "Accept Bitcoin payments in your business: why and how," *CoinGate*, 29 Nov., 2018. https://blog.coingate.com/2018/11/accept-bitcoin-payments-in-business-why-and-how/ (accessed: Dec. 23 2021).
[25] CoinGate, *How long does the merchant verification process take?* [Online]. Available: https://support.coingate.com/hc/en-us/articles/4402506213522-How-long-does-the-merchant-verification-process-take- (accessed: Dec. 23 2021).
[26] *What fees do I have to pay when I use CoinGate services? - CoinGate Support.* [Online]. Available: https://support.coingate.com/en/4/what-fees-do-i-have-to-pay-when-i-use-coingate-services (accessed: Nov. 20 2021).
[27] *CoinGate is rated "Great" with 4 / 5 on Trustpilot.* [Online]. Available: https://www.trustpilot.com/review/coingate.com (accessed: Dec. 16 2021).
[28] NOWPayments, *What cryptocurrencies are supported for payments? | NOWPayments.* [Online]. Available: https://nowpayments.io/supported-coins (accessed: Nov. 20 2021).
[29] NOWPayments API, *NOWPayments API.* [Online]. Available: https://documenter.getpostman.com/view/7907941/S1a32n38?version=latest (accessed: Dec. 23 2021).
[30] NOWPayments, *Pricing on crypto payments | Transaction & Exchange fees | NOWPayments.* [Online]. Available: https://nowpayments.io/pricing (accessed: Dec. 12 2021).
[31] *NOWPayments is rated "Average" with 3.7 / 5 on Trustpilot.* [Online]. Available: https://www.trustpilot.com/review/nowpayments.io (accessed: Dec. 12 2021).
[32] Coinify, *List of supported cryptocurrencies for merchants* (accessed: Dec. 12 2021).
[33] Datatrans Documentation, *Cryptocurrencies (Coinify).* [Online]. Available: https://docs.datatrans.ch/v1.0.1/docs/crypto-coinify (accessed: Dec. 12 2021).
[34] "Was ist ein Stablecoin?," *Coinbase.* https://www.coinbase.com/de/learn/crypto-basics/what-is-a-stablecoin (accessed: Dec. 12 2021).
[35] M. Vatiero, "Smart Contracts and Transaction Costs," *SSRN Journal*, no. 1, 2018, doi: 10.2139/ssrn.3259958.
[36] *Remix - Ethereum IDE.* [Online]. Available: https://remix.ethereum.org/#optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.7+commit.e28d00a7.js (accessed: Nov. 17 2021).
[37] Truffle Suite, *Ganache | Truffle Suite.* [Online]. Available: https://www.trufflesuite.com/ganache (accessed: Nov. 17 2021).
[38] GitHub, *go-ethereum/cmd/abigen at master · ethereum/go-ethereum.* [Online]. Available: https://github.com/ethereum/go-ethereum/tree/master/cmd/abigen (accessed: Nov. 17 2021).

[39] Marius Van Der Wijden, *Intro to Web3.go Part 2: abigen.* [Online]. Available: https://medium.com/@m.vanderwijden1/web3-go-part-2-aebdcb8d926e (accessed: Nov. 17 2021).
[40] GitHub, *nspcc-dev/neo-go: Go Node and SDK for the NEO blockchain.* [Online]. Available: https://github.com/nspcc-dev/neo-go (accessed: Dec. 5 2021).
[41] GitHub, *AxLabs/neo3-privatenet-docker: Docker images for running a Neo N3 private network, in record time!* (accessed: Dec. 5 2021).
[42] Neo-project, *Neo Documentation.* [Online]. Available: https://docs.neo.org/docs/en-us/basic/concept/transaction.html (accessed: Dec. 19 2021).
[43] I. Machado, "Scoped Witnesses: How to Securely Transfer Assets on Neo 3," *Neo Smart Economy*, 16 Sep., 2019. https://medium.com/neo-smart-economy/scoped-witnesses-how-to-securely-transfer-assets-on-neo-3-6ac012221188 (accessed: Dec. 8 2021).
[44] GitHub, *Contract to manage contracts · Issue #2000 · neo-project/neo.* [Online]. Available: https://github.com/neo-project/neo/issues/2000 (accessed: Dec. 8 2021).
[45] Neo-project, *Neo Documentation.* [Online]. Available: https://docs.neo.org/docs/en-us/develop/write/manifest.html (accessed: Dec. 8 2021).
[46] GitHub, *Fix custom sender scopes in CreateTxFromScript by roman-khimov · Pull Request #2272 · nspcc-dev/neo-go.* [Online]. Available: https://github.com/nspcc-dev/neo-go/pull/2272 (accessed: Dec. 8 2021).
[47] *Express - Node.js web application framework.* [Online]. Available: https://expressjs.com/ (accessed: Nov. 17 2021).
[48] *Taquito.* [Online]. Available: https://tezostaquito.io/ (accessed: Nov. 17 2021).
[49] *Home - SmartPy.* [Online]. Available: https://smartpy.io/ (accessed: Nov. 17 2021).
[50] *Flextesa: Home.* [Online]. Available: https://tezos.gitlab.io/flextesa/ (accessed: Dec. 7 2021).
[51] *docker v11 illegal instruction core dumped (#1788) · Issues · Tezos / tezos · GitLab.* [Online]. Available: https://gitlab.com/tezos/tezos/-/issues/1788 (accessed: Dec. 7 2021).
[52] *Address family not supported by protocol (#28) · Issues · Tezos / flextesa · GitLab.* [Online]. Available: https://gitlab.com/tezos/flextesa/-/issues/28 (accessed: Dec. 7 2021).
[53] *mainnet.sh script fails to run node on ubuntu 20.04 (#884) · Issues · Tezos / tezos · GitLab.* [Online]. Available: https://gitlab.com/tezos/tezos/-/issues/884 (accessed: Dec. 7 2021).
[54] *web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation.* [Online]. Available: https://web3js.readthedocs.io/en/v1.5.2/ (accessed: Dec. 8 2021).
[55] *Web3js/Taquito differences | Taquito.* [Online]. Available: https://tezostaquito.io/docs/web3js_taquito (accessed: Dec. 8 2021).
[56] GitHub, *Better handle block headers with no operations () when originating contracts · Issue #365 · ecadlabs/taquito* (accessed: Dec. 9 2021).
[57] Discord, *Discord - A New Way to Chat with Friends & Communities.* [Online]. Available: https://discord.com/channels/382937847893590016/382977764594155551/903293983176921128 (accessed: Dec. 6 2021).
[58] reddit, *Go library to interact with smart contracts?* [Online]. Available: https://www.reddit.com/r/tezos/comments/qqccg7/go_library_to_interact_with_smart_contracts/ (accessed: Nov. 17 2021).
[59] *Temple Wallet - Cryptocurrency wallet for the Tezos blockchain | Temple - Tezos Wallet.* [Online]. Available: https://templewallet.com/ (accessed: Nov. 17 2021).
[60] GitHub, *ecadlabs/taquito: A library for building dApps on the Tezos Blockchain - JavaScript / TypeScript* (accessed: Nov. 17 2021).

[61] *rollup.js.* [Online]. Available: https://rollupjs.org/guide/en/ (accessed: Nov. 17 2021).
[62] DenoLand, *Deno - A modern runtime for JavaScript and TypeScript.* [Online]. Available: https://deno.land/ (accessed: Dec. 9 2021).
[63] Docker, *What is a Container? | Docker.* [Online]. Available: https://www.docker.com/resources/what-container (accessed: Dec. 19 2021).
[64] Amazon Web Services, Inc., *Was sind Microservices? | AWS.* [Online]. Available: https://aws.amazon.com/de/microservices/ (accessed: Dec. 19 2021).

10. Anhang

Alle Anhänge wurden separate abgegeben. Dabei wurden folgende Anhänge abgegeben.

- Projektplan
- Risikoanalyse
- Persönliches Fazit
- Zeitrapport
- Deployment Dokumentation
- Datenbank Migrationsskript
- Test-Excel
- Source Code
- Protokolle