University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

A Blockchain Explorer for Bazo

Luc Boillat
Zurich, Switzerland
Student ID: 14-715-577

Supervisor: Dr. Thomas Bocek, Bruno Rodrigues, Hamza Bedrija
Date of Submission: February 1, 2018

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Das Belohnungssystem eines Finanzdienstleisters besteht aus Bonuspunkten, welche beim gebrauch von Kredit- und Debitkarten gewonnen werden. Der Karteninhaber kann diese Punkte im Online-Shop des Dienstleisters gegen Waren und Gutscheine umtauschen. Dies bringt einen hohen administrativen Aufwand mit sich, da für jeden neuen Händler, welcher im Online-Shop seine Waren gegen Bonuspunkte verkaufen möchte, ein speziell abgestimmter Vertrag erstellt werden muss. Zusätzlich ist die Popularität des Services nicht erwartungsgemäss, da die Punkte nur in diesem einen Online-Shop benutzbar sind. Zusammen mit der Universität Zürich wurde darum die Bazo Kryptowährung entwickelt, welche eine dezentralisierte Verwaltung der Punkte und Konten ermöglicht. Dies hat den Vorteil, das Händler an ihrem eigenen PoS ihre Waren gegen Bazo Coins verkaufen können. Der einzige Kontakt, welcher die Händler mit dem Finanzdienstleister haben werden ist das Umtauschen von Bazo Coins in Fiat Währung. Die Bazo Software besteht aus zwei Kommandozeilen-Tools welche die verarbeiteten Daten der Blockchain zwar speichern, jedoch nur bedingt dem Benutzer lesbar präsentieren. Diese Arbeit dokumentiert das Design, die Entwicklung und die Evaluation eines Blockchain Explorers für die Bazo Blockchain. Der Explorer ermöglicht dem Benutzer über einen Webbrowser die Blockchain-Daten zu durchsuchen und grafisch darzustellen. Ebenfalls verfügt der Explorer über eine Benutzeroberfläche für Administratoren, damit Systemparameter für die Blockchain gesetzt werden können.

The reward system of a financial service provider, consists of bonuspoints, which can be amassed by using credit- and debit-cards. These points can be exchanged for goods and coupons in the online reward shop of the service provider. This causes significant administrative overhead for the provider, since for every merchant that wants to sell its products in the reward shop, a tailored contract has to be made. Additionally popularity of the shop is not as expected, due to the bonus points being only useable in this specific shop. Jointly with the University of Zurich, the Bazo cryptocurrency was developed to counter these disadvantages of the bonus point system. This enables a decentralized management of points and accounts, and permits merchants to sell their products at their own PoS for Bazo Coins. The only contact merchants now have with the financial service provider, is when they exchange their amassed Bazo Coins for fiat money. The Bazo software consists of two command-line interfaces which, by design, save the processed data of the blockchain. However only limited access to this data is possible. This thesis covers the design, development and evaluation of a blockchain explorer for the bazo cryptocurrency. The blockchain explorer enables users to display and browse through the blockchain data via a web-browser. Additionally, the explorer contains an admin-panel, where administrators of the system can set certain system parameters of the blockchain.

Acknowledgments

Optional

Contents

Chapter 1

Introduction

A financial service provider based in Zurich operates a bonus point system and its associated reward shop. When participants of the program make transactions using their credit cards, issued by the service provider, bonus points are awarded to the clients. The number of points a client receives depends on the amount of money they have spent. Collected points can be exchanged in the reward shop for products and coupons. This means that every merchant who wishes to sell its products on the reward shop has to contact the service provider and form a contract with him. The two main drawbacks of this approach are on one hand, the administrative effort on the provider's side which is needed to (1) maintain relations with merchants and (2) manage the reward shop, while on the other hand, the lack of awareness and interest of the point system by the clients due to its restricted nature. To counter these drawbacks, the Bazo cryptocurrency has been developed, as a possible replacement for the traditional system. A blockchain-based, decentralized payment system that alleviates the provider's administrative costs by eliminating contracts with merchants. Using the currency Bazo Coin, the clients can buy products from merchants directly at their own Point-of-Sale, since the financial service provider is no longer the centralized record keeper of transactions and accounts. It is also possible for clients to transfer funds between each other. The only interaction between the service provider and merchants is the exchange of Bazo Coins for fiat currency. Similarly to the tradtitional system, Bazo will be invite-only, meaning only the clients of the service provider can interact with the blockchain, making Bazo a so-called private blockchain [2].

1.1 Motivation

To interact with the Bazo blockchain, two command-line applications are necessary: The Bazo Miner, which, together with all other Miners, runs the network, and the Bazo Client, which is mainly used to send transactions to the network. Every Bazo Miner stores all the blockchain and state data in its built-in storage component, however there is no way for a user to browse through and make use of that data using a GUI. Information about the health and productivity of the system are not available either. This is why a blockchain explorer is needed, a separate service that runs independently from the blockchain and

lets users examine the blockchain data, without directly taking part in the network using miner or client applications.

1.2 Description of Work

This thesis documents the design, implementation and evaluation of a blockchain explorer for the private blockchain Bazo and its corresponding cryptocurrency Bazo Coin. The explorer allows users and administrators of Bazo to inspect and analyze the data making up the blockchain. Blocks, transactions and accounts are being displayed in an informative and well-structured manner, with the explorer acting as a visualizer for the blockchain. Statistical information about the blockchain will also be made available to the user. Furthermore the explorer features administator-only functionality, serving as a GUI for setting various system parameters from the web to the Bazo network.

1.3 Thesis Outline

Chapter 2 further explains the bazo blockchain in detail and analyzes existing blockchain explorers and statistics analysis platforms. Chapter 3 focuses on the design of the Bazo Blockchain Explorer, consisting of multiple components. Chapter 4 documents the implementation of the web application, followed by an evaluation in chapter 5. A summary and conclusions are presented in chapter 6.

Chapter 2

Background and Related Work

This chapter gives a detailed overview of the Bazo cryptocurrency and its underlying blockchain technology, as well as an introduction to existing applications and ones being developed at the same time as the blockchain explorer . Additionally it presents an analysis of existing blockchain explorers for two different cryptocurrencies, highlighting both similarities and differences in the implementation and functionality of the applications. The analysis plays a major role in the specification of the Bazo Blockchain Explorer, as it helps making design decisions for requirements.

2.1 Bazo Blockchain and Cryptocurrency

Developed in 2017 at the University of Zurich, the Bazo cryptocurrency is a private blockchain that aims to reduce administrative overhead, as well as extend the functionality of a financial service provider's bonus point reward system. Traditionally, for each merchant who wants to sell its products in the rewards shop of the service provider, specific contracts between the two parties need to be made. This makes expanding the bonus point system a time and resource consuming process. Bazo eliminates this restriction by introducing a cryptocurrency which allows to directly make transactions between merchants and users or even between users itself using Bazo Coins. The merchants itself do not need to form contracts with the service provider anymore, they can offer their products in exchange for Bazo Coins even at their own Point-of-Sale. The only interaction between the service provider and merchants consist of the exchange of Bazo Coins for fiat currency. A trial is planned, where the Bazo systems runs simultaneously to the existing bonus point system. Clients can request to exchange their current bonus points for Bazo Coins and vice-versa.

2.1.1 Characteristics of Bazo

Similarly to Ethereum [3], Bazo uses an account-based model, which means that every user of the blockchain has a unique keypair (public and private key) that does not change after

making a transaction. The public key acts as the address, when a user wants to receive funds from another user. The private key should only be held by its respective user and never leaves a user's device. It is used to sign transactions. A transaction only gets verified by the system if the correct private key has been used. In order to save bandwith, the blocks mined by the network do not contain all transaction data of transactions included in a block, only the hashes of transactions. The storage component of the Miner application saves all transaction data. There are 4 different types of transactions possible in the Bazo system.

- **Funds Transactions** Funds Transactions are the most commonly used transactions, as they are the ones used by users of the blockchain to send Bazo Coins from one to another. Among other information, every transaction includes identifiers for the sender and receiver, and the amount of Bazo Coins being sent.
- **Account Creation Transactions** Only available to administrators of the system, Account Creation Transactions are used to generate new accounts. The public key of the new account is included int the transaction data, however the full keypair is stored on the device that generated the account.
- **System Configuration Transactions** Due to Bazo being a blockchain built from scratch, no guidelines for parameters such as the block interval or the minimum transaction exist. This is why these parameters can be changed on-the-fly by administrators using System Configuration Transactions.
- **Stake Transactions**

Every transaction requires a fee to be processed. These fees are collected by the miners who successfully mine a block. This incentivises people to offer their processing power and in turn run the network. The administrator of Bazo will be the Financial Service Provider, meaning he alone has the power to add accounts and change system parameters.

2.1.2 Bazo Applications

In order for the Bazo system to be run, two command-line programs are needed. Both were developed as part of the original *Bazo – A Cryptocurrency from Scratch* [2] thesis.

- **Bazo Miner** This application, together with all other running Miners, makes up the network. It verifies transactions and mines blocks using a Proof-of-Work, or a Proof-of-Stake algorithm. On startup, the application copies the verified blockchain data, meaning the entire blockchain, from other miners into its storage component in order to be up-to-date and start verifying transactions. It also handles data concerning Bazo accounts and their balances, also known as the state of the blockchain.
- **Bazo Client** The Client is used for sending transactions to the network and making requests about the state. All types of transactions have to be made from the Client, including sending Configuration and Account Creation Transactions which are only

reserved for administrators of the blockchain. A drawback of the Client application is the need to download the entire blockchain, similarly to the Miner in order for it to be useable.

Simultaneously to the development of the Bazo Blockchain Explorer, additional applications were developed, which enhance the scope and functionality of Bazo.

- **Bazo Light Client** [4] The Light Client fork of the Bazo Client application makes sending transactions possible, without having to download the entire blockchain. A Bloom Filter is responsible for only having to download blocks relevant to the user. Bandwidth and storage on the device can be saved with this Client implementation.

- **Bazo Payment System** [5] A web-based wallet and payment app have been developed, which enables users to manage their accounts and make transactions from their mobile devices or desktop computers, without having installed a native application. It also features Point-of Sale functionality, making the Bazo system useable in a real-life client-merchant situation.

- **Bazo Interface** [4] This interface is needed for both the Payment System and the blockchain explorer to send transactions to the network. Due to them not having implemented a Bazo Client, both applications are not able to build transactions on their own. This service receives the transaction information without the private key of the sender via a REST interface and responds with a transaction hash. This hash then gets signed with the private key stored on the device and sent back to the Interface for it to be broadcasted to the network.

- **Proof-of-Stake Algorithm** [6]

2.2 Blockchain Explorers and Analytics Platforms

Since many cryptocurrencies are open-source and public [3] [7], often multiple blockchain explorers exist for a single currency. The most common application type for such explorers are web applications, publicly available and free to use on the internet. A blockchain explorer allows users to inspect blockchain data such as blocks and transactions, which may or may not be related to the user. A common use-case for block explorers are, after a user has submitted a transaction to the network, checking whether the user's transaction has been accepted or verified by the respective network. Some Cryptocurrency wallets, such as the Ledger Bitcoin Wallet [8], feature automatically generated links to blockchain explorers, so that a user can effortlessly check, whether his transaction has gone through or not. Other information blockchain users may find interesting, is statistical data about the network, such as 24 hour transaction volumes or market capitalization compared to other cryptocurrencies.

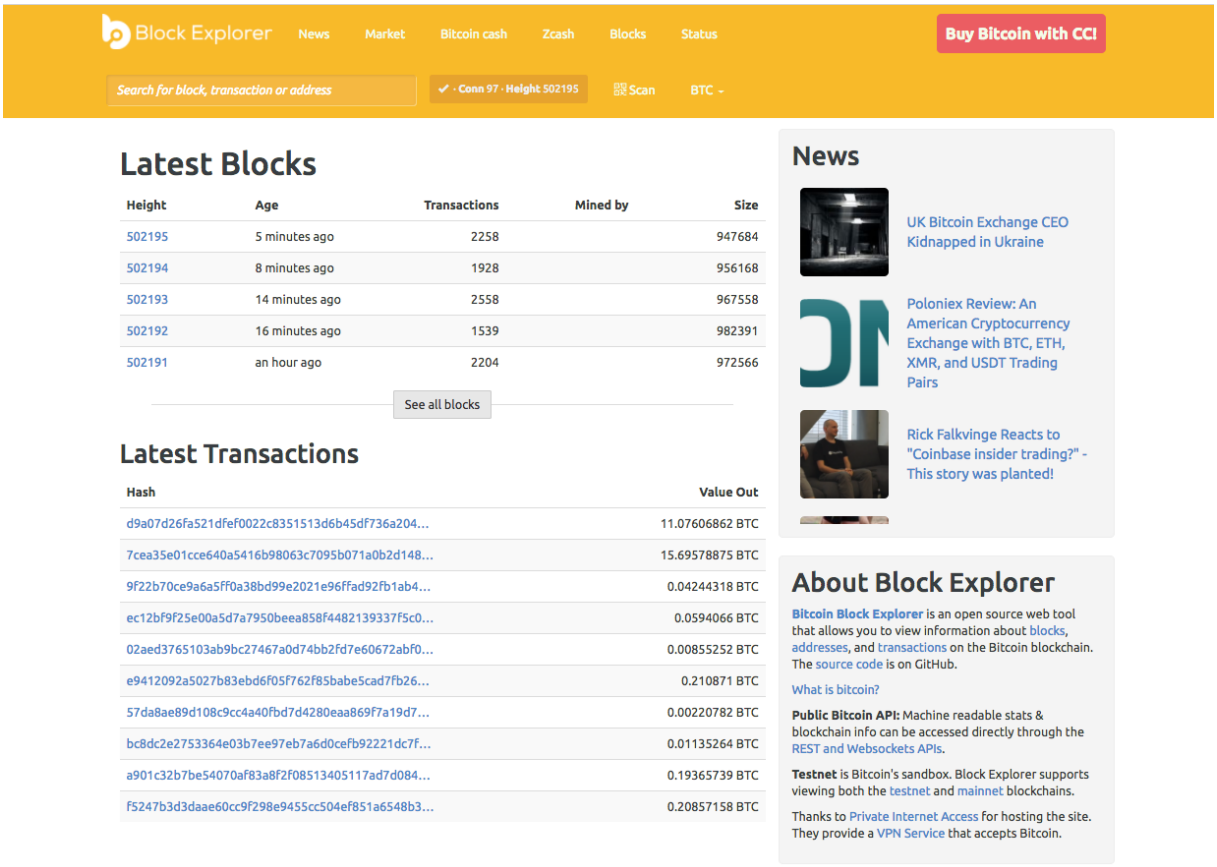

Figure 2.1: Landing Page of blockexplorer.com [9]

2.2.1 Blockexplorer.com [9]

This blockchain explorer was built for both the Bitcoin [7] and Bitcoin Cash [10] blockchains. The frontend of the web application is called Insight UI [11] and is built using AngularJS [12], a JavaScript [13] framework. It interacts with the Insight API [14], the corresponding backend. Insight API consists of a REST and websocket API for Bitcore Node [15], a query and indexing service for the Bitcoin blockchain [7]. The source code for both frontend and backend are available on GitHub [11] [14].

2.2.2 Etherscan.io [16]

Etherscan is a blockchain explorer and statistics analysis platform for the Ethereum blockchain [3]. It uses Go Ethereum [17], an implementation of the Ethereum protocol in the Go language [18], in combination with Parity [19], a client for interacting with the Ethereum blockchain [3]. Etherscan is a closed source project.

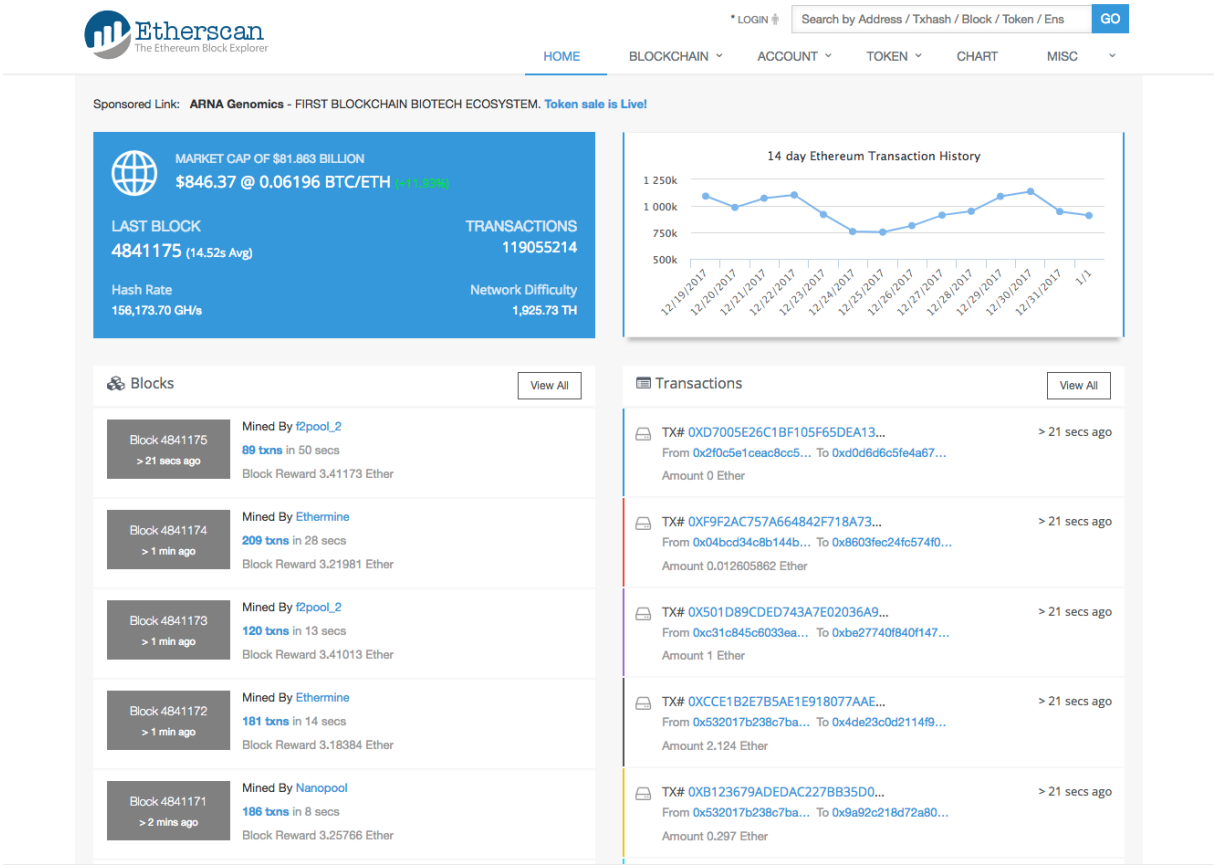

Figure 2.2: Landing Page of etherscan.io [16]

2.3 Analysis

Both explorers offer similar functionality as their core-feature: Structured views of blocks and transactions. The landing pages display the most recently mined blocks and transactions, with Blockexplorer offering real-time updates. Etherscan also displays statistical data about the chain, such as the market capitalization, mining difficulty and hash rate. A search feature is present on both sites, offering the user to search for transactions, blocks and accounts via their respective hashes. To browse the chain, links are used extensively (e.g. every block on the landing page links to its respective detailed block page). When presenting multiple objects on the same page, such as a list of blocks, the data is structured using tables. When multiple items are displayed using lists, less information about the items is given, compared to when a single item is viewed.

2.3.1 Mutual Functions and Components

DIFFERENE ETHEREUM BITCOIN

- **Landing Page** Both explorers offer various information about their respective blockchain on the landing pages of the websites. The most recent blocks and transactions are displayed, with Etherscan using formatted list elements to differentiate between the individual blocks or transactions. Blockexplorer uses tables, that structures columns vertically. Among other, information regarding each block's age, height and number of transcations are featured. Etherscan displays more information regarding transactions, such as sender, receiver, compared to Blockexplorer, which only displays the transaction's hash and amount. Links to the full lists of blocks are present on both applications, however only Etherscan links to a full list of transactions.

- **Blocks** Lists, where all blocks ordered by age, are formatted using tables. More information about each block are displayed here, than on the landing page. Etherscan uses pagination to avoid having *all* blocks displayed at once. Blockexplorer features a date-picker, that enables the display of only blocks that were mined on the chosen date. Both applications feature views for specific blocks, which display all information about a block. Links to other items like accounts and transactions are used heavily to make navigation of the page easier.

- **Transactions** As mentioned above, only Etherscan has a list displaying all transactions at once. It is structured using a table, similar to how blocks are structured on the page. Specific transactions can be viewed on both applications, however. Since Bitcoin uses a UTXO model [7], often not only one sender and one receiver are featured in a transaction, but multiple. This irregularity makes displaying senders and receivers in tables difficult and only possible on each transaction's detailed page on Blockexplorer.

- **Accounts/Addresses** Only Etherscan features a list of the most affluent accounts on the blockchain. On the detailed page for an account, both applications display

all transactions where that account was either a sender or a receiver. The balance of each account or address is desplayed as well.

- **Navbar and Search** Both applications use a navbar, which is present on all pages. The navbar features links and dropdown menus to guide a user to the specific functions of the application. Etherscan has a significantly more extensive navbar than Blockexplorer, which makes an assessment ofthe overall funcitonality of Etherscan much easier. A search bar is also included in both navbars, where a user can search through blocks, transactions and addresses using hashes. The system does not require the user to choose emphwhat he is actually looking for, but finds that out automatically, given the hash a user enters is valid.

- **Status**

2.3.2 Functions and Components Specific to Blockexplorer.com

- **Blockchain-Related News Feed** A section displaying news, related to the blockchain community.

2.3.3 Functions and Components Specific to Etherscan.io

- **Statistical Information** Etherscan offers a vide variety of statistical information about the Ethereum network, displayed using graphs over time. A detailed overview of total Ether supply is also available.

- **Calculation Tools** A mining calculator is offered, that lets users evaluate the profitability of their mining operations.

Chapter 3

Design

This chapter covers the design decisions taken for the Bazo Blockchain Explorer web app and the additional components necessary to run the application.

3.1 Requirements for the Bazo Blockchain Explorer

Based on the analysis performed in 2.3 and meetings held with members from the financial service provider and the University of Zurich, the use cases listed in figure 3.1 and the following functional requirements were elicited:

- **Blocks** A user should be able to view all validated blocks of the blockchain and the information they contain. In a list-view, the most recent blocks are being displayed, identified by their respective hashes and timestamps. If a user wants to get more comprehensive information about a block, he can display one block in a detailed-view, where additional information, such as all the transaction hashes contained in this block or the address hash of the block-reward beneficiary are presented.
- **Transactions** Similarly to the requirement above, a user should be able to view all validated transactions that have been broadcasted by him or other users of the blockchain. Due to the Bazo system having 4 different types of transactions (Funds Transactions, Account Creation Transactions and System Configuration Transactions from the original Bazo paper [2] and Stake Transactions from the Proof-of-Stake implementation [6]), different implementations for each of them have to be made. A list-view displays the most recent transactions of each type, offering information such as the sender, receiver and the amount of the transactions. Detailed views for all transaction types are needed as well, presenting more information, for example each transaction's signature or block it is contained in.
- **Accounts** With Bazo using an account-based model, every user that actively interacts with the blockchain owns an account. The application should maintain a state of all accounts, that gets updated with every newly mined block. A list with the most affluent accounts should be made available, as well as a detailed view of for

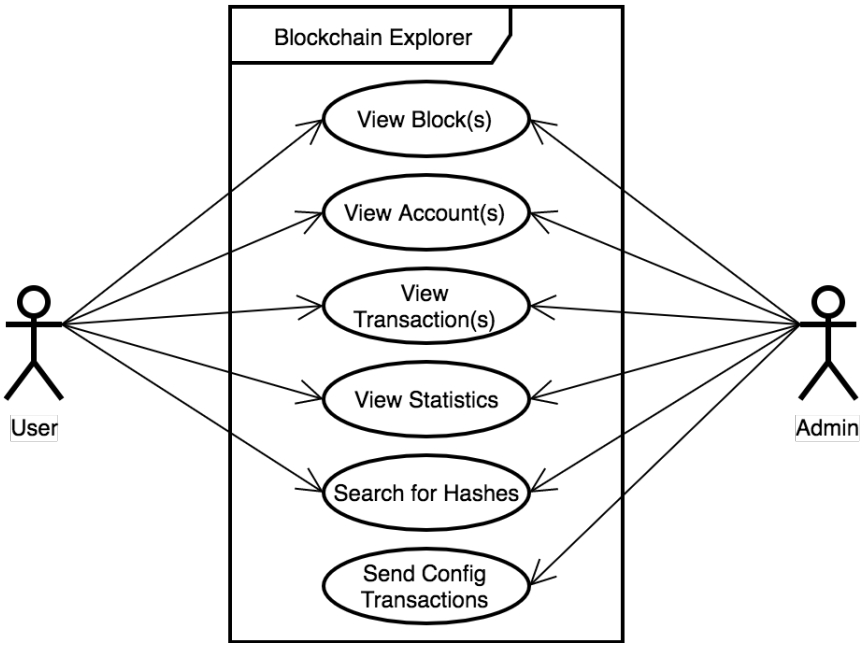

Figure 3.1: Use Cases of the Block Explorer

single accounts, which displays all transactions this account was either the sender or receiver of.

- **Search** A user should be able to search for blockchain data using hashes. These hashes can be identifiers of blocks, transactions or accounts, with accounts having both addresses and address-hashes. A user should not need to choose what he actually searches for, meaning the application searches through all data it has collected and, in case of a hit presents the data associated with the hash to the user, and in case of a miss, notifies the user of not finding any relevant data.

- **Navbar** Featured on every page of the application, a navbar, that lets a user access all functionality of the website, is required. This functionality includes, among others, links to lists of blocks, transactions and accounts. The search-functionality is also located here, being available at all times to the user.

- **Statistical Information** The system should calculate statistical information about the network and make it available to users. This includes information such as a graphical history of transactions, or the total amount of Bazo Coins currently in the system.

- **Admin Panel** Only available to admins of the system, a panel that lets them change system parameters using System Configuration Transactions has to be implemented. These transactions get sent via an interface [4] to the network, meaning no Bazo Client is running on the server of the website.

- **Fetch and Store Blockchain Data** The application needs to automatically gather the latest Bazo Blockchain data and save it independently from the blockchain. This data also needs to be correctly formatted, in order to make it accessible to the users.

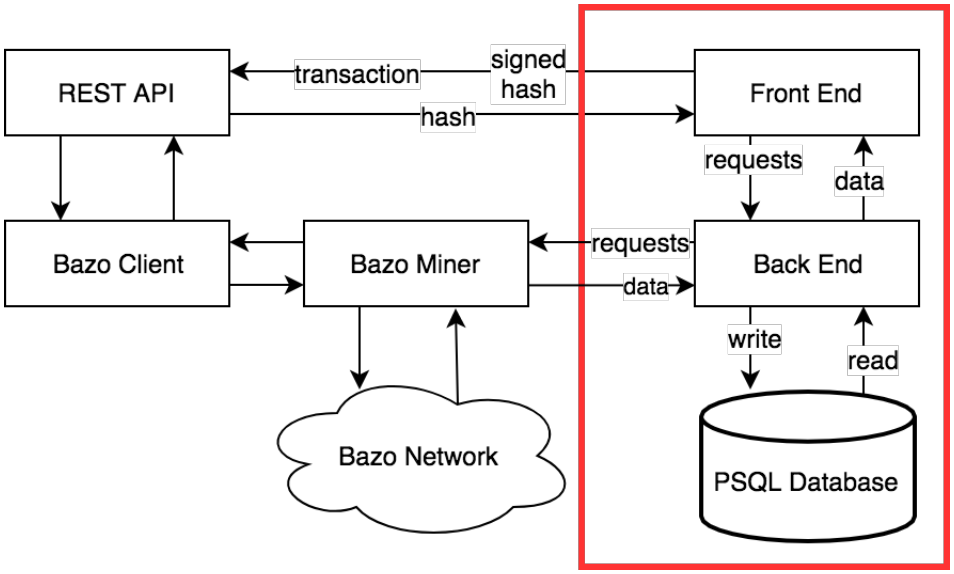

Figure 3.2: Structure of the Blockexplorer and Bazo Components

3.2 Structure of the Service

The main component of the Bazo Blockchain Explorer that users and admins use to view Bazo Blockchain data is a web app, which is made up of a front- and a backend. However, to run the blockchain explorer on its own, additional components beside the front- and backend components are required. Highlighted in red in figure 3.2 are the components that were implemented as part of this thesis. The app fetches blockchain data from a database that runs independently from the blockchain miner's built-in storage. A separate database was chosen, because additional data like statistical information needs to be calculated and stored as well, which would bloat all miner's built-in databases with information, if implemented in the miner. This also makes running the website possible without having a miner running in the backend of the web app. However, this requires a component that copies data from a running Bazo mining node's storage and stores it in the new database. As mentioned above, the web app's backend accesses this database by making queries for relevant data and sending the results to the frontend to be displayed to the user. In order to fulfil the requirement of being able to send System Configuration Transactions from the application, an interface that acts as an extension for a Bazo Client is required. With the use of this interface, signing transactions without a running Bazo Client will be made possible.

3.2.1 Trust

The operator of the blockchain hosts the system, which in this case is connected to a miner that also belongs to the operator. Due to the open-source nature of the block explorer however, anyone can run and host a Bazo Blockchain Explorer on his or her servers and connect to a miner of choice. A discussion about whether the miners, to which the explorer connects, need to be trusted or not, is not required, since (1) the operator has no interest

in presenting its users falsified data, (2) in case doubt arises about the data's authenticity, anyone can host his own explorer, and (3) from a user's perspective, the explorer only *consumes* data from the blockchain. The running Bazo blockchain is therefore not affected by any block explorers.

3.3 Web Application

A web application was chosen, because of its accessibility and its convenience. Users do not need to download software or store data on their devices to inspect the blockchain. The user interface of the application always displays the most recent blockchain data. In order to structurally separate the functionality and to fulfil all requirements, specific pages featuring the following content need to be included in the application:

- List of Most Recent Blocks
- Detailed Block
- List of Most Recent Funds Transactions
- Detailed Funds Transaction
- List of Most Recent Account Creation Transactions
- Detailed Account Creation Transaction
- List of Most Recent Configuration Transaction
- Detailed Configuration Transaction
- List of Most Affluent Accounts
- Detailed Account
- Statistical Information
- Administrator Panel

From a list-view of a data-type, each individual item can be accessed. The goal is to first, provide the user with a list view of a data-type, such as blocks. Once he finds an object of interest, he clicks on the hash of the block, which is its unique identifier, and gets presented with detailed information about that object. Depending on the data-type, the user is viewing, links to related elements enable further browsing of the data. For example, on a detailed block page, a link to the detailed account page of the account that received the block reward of said block, is listed. Since all different types of blockchain data have their own data structures, custom tables for all listed pages, that contain data need to be built. In figure 3.3, a mockup of the table containing the most recent blocks is displayed. The block's hash, the timestamp when it was mined and the counters for each transaction type are listed. Each single block hash links to the detailed page of that

Latest Blocks

Hash	Timestamp	NrFundsTX	NrAccTX	NrConfigTX
blocknumber1_l9oweuhy0q1i24ow7iz	2017-11-07T13:33:21	6	0	0
blocknumber2_l86k7bsv0kvc9e1i59w	2017-11-07T13:39:07	7	0	0
blocknumber3_0m79xdpm23vkeixrw95	2017-11-07T13:44:27	7	0	0

Figure 3.3: Mockup of the List of Most Recent Blocks

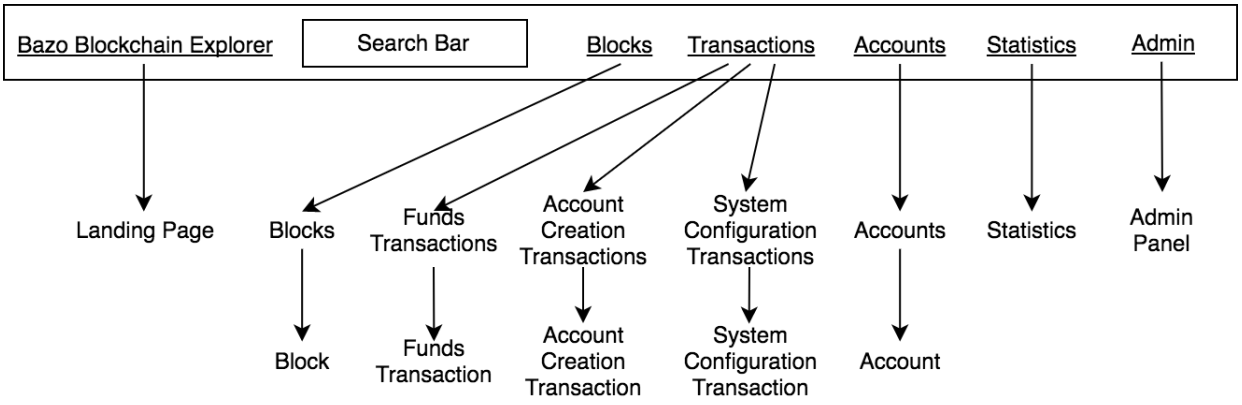

Figure 3.4: Functionality and Links on the Navbar

block. On the detailed block page, all transaction hashes that are included in a block, are ordered by type and displayed in a list. Similarly on a detailed account's page, all transactions related to that account are displayed as well.

3.3.1 Navbar and Search

The navbar is featured on every page of the application. Figure 3.4 displays all possible pages that are accessible from the navbar and the their respective detailed pages. Since there are multiple types of transactions, a dropdown menu on the *Transactions* button, displays links to list-views of all types of transactions. The administrator panel is also accessed from here. Included in the navbar, is the search function. A user can enter a hash and the backend of the application searches for that hash in the database. A successful search will automatically present the data to the user in the right formatting, meaning the system displays the page according to the data type of the search result. It is possible to search for blocks, all transaction types, account hashes and account addresses, since Bazo uses both hashes and addresses to identify an account.

3.3.2 URL-Scheme

Keeping URLs as simple as possible was a priority. Since a specific object may only be identified by its hash or its address, either is the only variable that is needed in a URL

for specific objects.

3.3.3 Administrator Panel

Since there currently are five different types of System Configuration Transactions defined by IDs 1 through 5 [2], a *Submit* form for every type needs to be available on the administrator panel. Depending on which administrator is currently logged in and using the panel, account information, such as the current transaction count and public key of the administrator, do not need to be entered in the transaction forms. These values get automatically included in the transactions are are derived from the public key, the administrator entered to log in. Detailed information about the verification process can be found in 4.2.3. The administrator enters the payload for the transaction he wishes to send and the transaction fee. The application validates the user input before sending transaction details. The panel also displays all current system parameters and updates them after each validated System Configuration Transaction.

3.3.4 Statistical Information

On this section of the application, users can view data about the overall health and productivity of the Bazo blockchain. A chart featuring a 14-Day transaction history visualizes how frequently the Bazo system is used, as well as shows, whether usage increases over time of operation. Information regarding total Bazo Coin supply and the total number of Bazo accounts is also provided, since viewing accounts and their balances by all users, is possible anyway. The statistical information is updated in a regular interval, to match the recent blockchain data.

3.4 Database

A relational database was chosen for the explorer, because the data that gets saved always has the same format. Efficiently sorting data by an attribute, is an ability that is needed in order to provide a responsive user experience. Ideally the database is hosted on the same server, as the block explorer is, to ensure optimal read and write times. Functions to read and write single items to the database need to be available, as well as functions that return multiple rows of data. In the case of presenting statistical information to the user, this information needs to be calculated with the data from the database.

Chapter 4

Implementation

This chapter documents the implementation of the components listed in chapter 3 and how they interact with each other. An additional section concerning hosting of the application on the internet is also included. GO INTRODUCTION AND LIBRARIES

4.1 Frontend

The frontend of the application is the interface, the user interacts with.

4.1.1 HTML Templates

To interact with the Golang-Backend of the application, Gohtml templates [21] have been used to define the markup of the pages. One one hand, this makes way for a modular view component by letting programmers define reusable HTML modules such as headers, footers or table-templates. Using templates can minimize code duplication, which in turn makes maintenance on the code an overall less risky task. On the other hand, Golang templates allow for some limited logic in the Gohtml files, which is needed for handling variables that get passed from the backend component. For example, if the passed variable is an array of integers and the goal is to display all variables in a list, golang can create a new <li> tag for every value in the array. The HTML code that gets passed to the end user after making a request contains no Golang code, since the backend renders the Gohtml template files and passed variables to a useable HTML file that can be displayed by a web browser [22]. Except for some Bootstrap [23] functionality mentioned in 4.1.3 and one case discussed in section 4.1.4, all rendered pages of the application are static HTML files, because for every action a user may make on the website, a request has to be made to the backend. The website aims to always display the most recent data, so storing information that may not be up to date on the client's machine in order to save bandwidth, is outweighed by the possibility of having more recent data.

Reusable Modules

- **Navbar**
- **HTML Head**
- **Public Key Modal**
- **Private Key Modal**
- **Script Imports**

4.1.2 Handling Passed Variables

If the backend of the application passes any variable to a template, Gohtml templates can render these variables by including "{{ . }}" in the template. Golang code is delimited by curly braces inside templates. The period is the placeholder, the passed variable replaces, before being rendered to HTML. When passing a struct to a template, all attributes of the struct can be accessed, by appending the attribute name to the period. In order to pass different structs to the template, such as in the landing page's case, where slices of blocks *and* slices of transactions are passed, structs containing structs need to be defined, since only one object can be passed to the template. PSEUDOCODE SHOWCASING RANGE, END FUNCITONALITY

4.1.3 UI Framework

In the beginning of development, a custom CSS built from scratch was created to style the application, however, due to concerns about usability and overall presentation of the website, the Bootstrap v4 UI Framework [23] was chosen as a replacement and enhancement of the original styling. "Originally created by a designer and a developer at Twitter, Bootstrap has become one of the most popular front-end frameworks and open source projects in the world." [30] It offers a wide variety of content and components to quickly build a responsive user interface. Most components are defined in the Bootstrap CSS, however for some client-side funcitonality, JavaScript can be enabled. Block explorer pages are heavily dependent on Bootstrap, since all UI components use Bootstrap classes. In cases where further styling was needed, a separate CSS overwrites Bootstrap stylings, for example the width of the Search Bar had to be manually widened. In some tables, inline styling in their HTML was used, when certain columns needed to have fixed widths.

4.1.4 Client-Side Logic

Vue.js [24] and Libraries

Transaction Signing

A requirement of the Bazo Blockchain Explorer is to be able to send System Configuration Transactions from the browser. However, since no Bazo Client should run in the backend of the application, the ability to build and sign transactions is not provided. A Bazo Client takes transaction details and the private key of the sender in order to build a transaction. Extracting only the transaction signing functionality of the Bazo Client and implementing it into the backend of the explorer would violate security constraints, since the private key would have to be sent over the network to the server. The private key should always stay with the user and never leave his device. This is where the Bazo Interface [4] comes in place, a REST API that builds an unfinished transaction hash using the transaction details without the issuer's private key. In a second step the user enters his private key in order to sign the transaction client-side and then send it back to the Interface, which then publishes it to the network.

The process is detailed in figure 4.1. A user enters the values that make up the transaction into HTML-input fields on the administrator panel, which are linked to a Vue.js application [24] . In this case, that would be the type of Configuration Transaction (id), the payload (new value) and the fransaction fee.When the user hits *Send*, JavaScript code extracts the public key stored in a cookie [26] (user verification is described in **??**) and requests the account data concerning that user. The REST interface responds with the account information in JSON format [28] , of which the *isRoot* attribute is important to the application. If the public key, stored in the cookie belongs to a root account, the JavaScript code builds a URL that contains all transaction data, previously entered by the user. Another request to the interface, using that URL is made, which then builds a transaction hash using the transaction data. If that hash was returned successfully, a modal opens up on the block explorer, prompting the user to enter a root private key. Using a JavaScript implementation of elliptic signing [27], the transaction hash gets signed with the entered private key, which results in a new, signed transaction hash and sent to the interface, along with the unsigned hash for identification purposes.

4.2 Backend

The backend of the web app is written in Golang [18], which is well suited for writing web applications, due to its included http [22]and HTML [21] packages. Since the Bazo Blockchain was also written in Golang [2], compability between the two programs was ensured. The program is split up into distinct files, to separate functionality. GO??

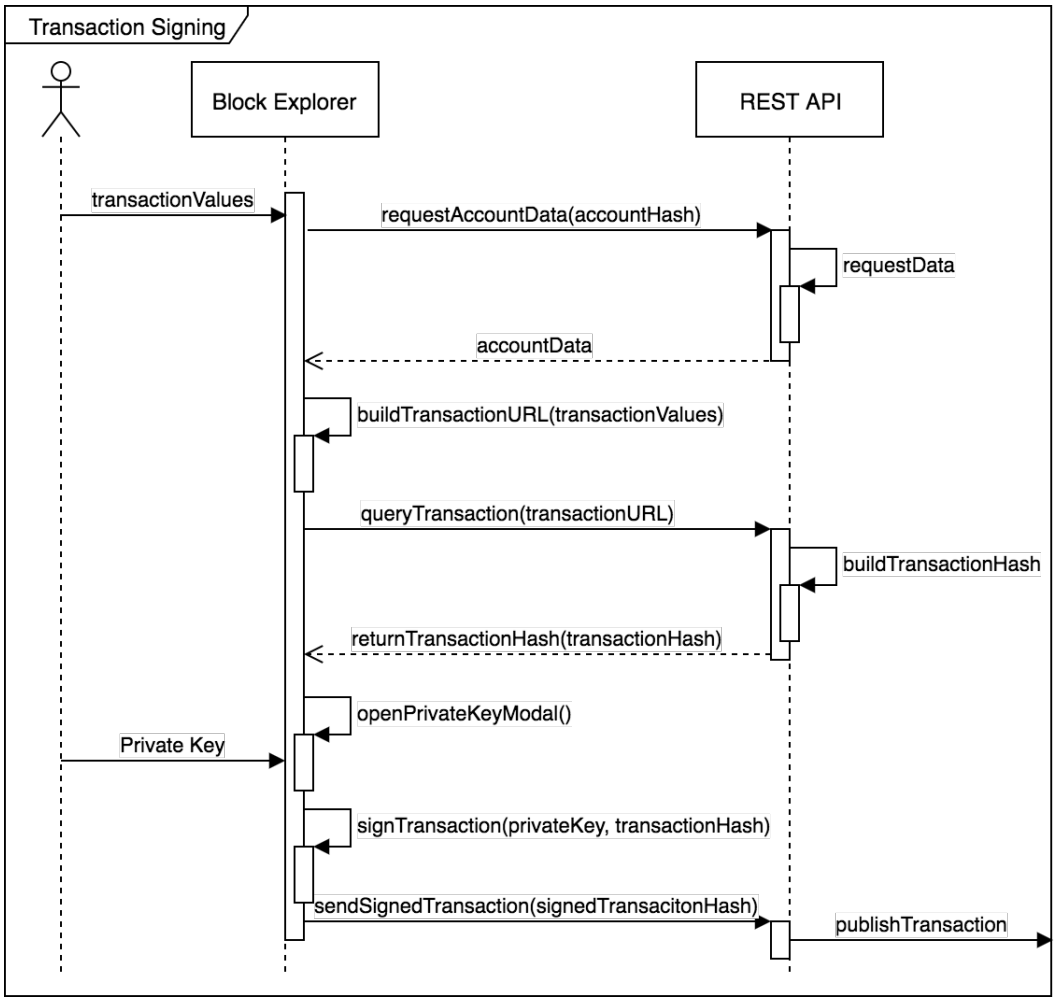

Figure 4.1: Sequence Diagram of How the Application Builds and Signs Transactions

4.2.1 Packages of the Block Explorer

The Golang code of the block explorer is split up into... GRAPHIC

4.2.2 Router

The router handles all requests made to the application. On startup, the router gets initialized and registers all possible routes. Furthermore, all Gohtml templates and JavaScript files are passed to the router, so they can be rendered and used upon request [22]. Additionally to the built-in http package in Golang, HttpRouter by Julien Schmidt [20] has been used as well.

"In contrast to the default mux of Go's net/http package, this router supports variables in the routing pattern and matches against the request method. It also scales better." [20]

HttpRouter makes building a routing table convenient. All routes and the functions they invoke, once called, can be defined in a well-structured way. Except for the */login* and */search* routes, which are called using POST methods, all routes are called via GET request.

4.2.3 Cookies

By design, only administrators of the system can successfully publish a System Configuration Transaction to the network, since only they possess a root private key that lets them sign the transactions. However, using a Bazo Client, a non-root user may still spam the network with admin-only transactions, without them ever getting accepted by the system. Therefore implementing a thorough login-system on the explorer that requests a user's private key to verify whether he has administrator rights, in order to prevent non-administrator users from trying to publish transactions to the network seemed unnecessary. This would have also meant that a private key would get send over the network to check its access rights. The chosen verification process is as follows: Upon requesting to log in, a user enters a public key. This public key then gets sent to the Bazo Interface [4], where more information regarding that key is requested. One attribute of the response the interface returns, is *isRoot*. If *isRoot* returns true, a cookie containing the entered public key gets saved to the user's machine and the user is verified. When the user then requests the administrator panel, the public key in the cookie gets checked again and sent to the interface. If the returned *isRoot* is still true, he gets access to the administrator panel. The public key contained in the cookie then gets extracted by client-side code described in 4.1.4.

4.2.4 Concurrency

The application makes use of Golang's built-in concurrency functionality. By calling *runDB()* in the main function as a Goroutine by appending the keyword *go*, the mechanism

that automatically updates the database with new blockchain data, runs separately from the router's request handling.

4.2.5 Structs

Hashes and other generated values in the Bazo blockchain are stored using fixed-sized byte-slices. This has several drawbacks in terms of usability, since they first have to be converted into strings by representing each byte in its hexadecimal value and concatenating all hex values, when displayed to the user. Because of difficulty preserving the byte-slices while storing them in a PSQL database, and the fact that users search for string representations of these values, all blockchain data first gets converted to human-readable form before saving it in the database. Additionally for easier data-management, all entries in all transaction tables now have added *BlockHash* and *Timestamp* attributes, since using raw blockchain data, transactions could not be traced back to a block or a mining date.

Conversions

4.2.6 Data Transfer

Procedure for Saving a Block

4.3 Hosting

Chapter 5

Evaluation

Chapter 6

Summary and Conclusions

Bibliography

[1] Autoren: Titel, Verlag, http://..., Datum.

[2] Livio Sgier: Bazo - A Cryptocurrency from Scratch, University of Zurich, August 23, 2017

[3] Ethereum - A Next-Generation Smart Contract and Decentralized Application Platform, https://github.com/ethereum/wiki/wiki/White-Paper, accessed on January 10, 2018

[4] Marc-Alain Chetelat: Bazo Light Client and Multisig

[5] Jan von der Assen: Wallet

[6] Simon Bachman: Proof of stake

[7] Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf, accessed on January 10, 2018

[8] Ledger Bitcoin Wallet, https://www.ledgerwallet.com/apps/bitcoin, accessed on January 10, 2018

[9] Blockexplorer.com, https://blockexplorer.com/, accessed on January 10, 2018

[10] Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System https://www.bitcoincash.org/bitcoin.pdf, accessed on January 10, 2018

[11] Bitpay: Insight UI, https://github.com/bitpay/insight, accessed on January 10, 2018

[12] AngularJS, https://angularjs.org/, accessed on January 10, 2018

[13] JavaScript, https://developer.mozilla.org/bm/docs/Web/JavaScript, accessed on January 10, 2018

[14] Bitpay: Insight API, https://github.com/bitpay/insight-api, accessed on January 10, 2018

[15] Bitpay: Bitcore Node, https://github.com/bitpay/bitcore-node, accessed on January 10, 2018

[16] Etherscan.io, https://etherscan.io/, accessed on January 10, 2018

[17] Ethereum: Go Ethereum, https://github.com/ethereum/go-ethereum, accessed on January 10, 2018

[18] The Go Programming Language, https://golang.org/, accessed on January 10, 2018

[19] Parity, https://www.parity.io/, accessed on January 10, 2018

[20] Julien Schmidt: HttpRouter, https://github.com/julienschmidt/httprouter, accessed on January 10 2018

[21] Go Package Template, https://golang.org/pkg/html/template/, accessed on January 10 2018

[22] Go Package Http, https://golang.org/pkg/net/http/, accessed on January 10 2018

[23] Bootstrap v4, https://getbootstrap.com/, accessed on January 10 2018

[24] Vue.js, https://vuejs.org/, accessed on January 10 2018

[25] Axios, https://github.com/axios/axios, accessed on January 10 2018

[26] vue-cookies, https://github.com/cmp-cc/vue-cookies, accessed on January 10 2018

[27] Elliptic, https://github.com/indutny/elliptic, accessed on January 10 2018

[28] JSON, https://www.json.org/, accessed on January 10 2018

[29] Go Package sha3, https://godoc.org/golang.org/x/crypto/sha3, accessed on January 10 2018

[30] Bootstrap History, https://getbootstrap.com/docs/4.0/about/overview/, accessed on January 10 2018

Abbreviations

AAA	Authentication, Authorization, and Accounting

Glossary

Authentication

Authorization Authorization is the decision whether an entity is allowed to perform a particular action or not, e.g. whether a user is allowed to attach to a network or not.

Accounting

List of Figures

List of Tables

Appendix A

Installation Guidelines

Appendix B

Contents of the CD