



University of
Zurich^{UZH}

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

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 bonus points, 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

I would like to thank my supervisor, Dr. Thomas Bocek for his continuous support and inspiring enthusiasm regarding peer-to-peer and blockchain technologies. While working on the thesis, his insight into these matters provided valuable information and a pleasant environment to work in.

I would also like to thank Prof. Dr. Burkhard Stiller for letting me write this bachelor's thesis at the Communication Systems Research Group of the University of Zurich.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	2
1.3 Thesis Outline	2
2 Background and Related Work	3
2.1 Bazo Blockchain and Cryptocurrency	3
2.1.1 Characteristics of Bazo	4
2.1.2 Bazo Applications	5
2.2 Blockchain Explorers and Analytics Platforms	6
2.2.1 Blockexplorer.com [8]	6
2.2.2 Etherscan.io [15]	6
2.3 Analysis	6
2.3.1 Mutual Functions and Components	9
2.3.2 Functions and Components Specific to Blockexplorer.com	10
2.3.3 Functions and Components Specific to Etherscan.io	10

3	Design	11
3.1	Requirements for the Bazo Blockchain Explorer	11
3.2	Structure of the Service	13
3.2.1	Trust	14
3.3	Web Application	14
3.3.1	Navbar and Search	15
3.3.2	URL-Scheme	16
3.3.3	Administrator Panel	16
3.3.4	Statistical Information	16
3.4	Database	17
3.4.1	Tables	17
4	Implementation	19
4.1	Packages of the Block Explorer	19
4.2	HTML Templates	20
4.2.1	Reusable Modules	21
4.2.2	Handling Passed Variables	21
4.3	UI Framework	22
4.4	Client-Side Logic	23
4.4.1	Vue.js and Libraries	23
4.4.2	Transaction Signing	23
4.4.3	AnyChart	25
4.5	Router	25
4.6	Cookies	26
4.7	Concurrency	27
4.8	Structs	27
4.9	Data Transfer	28
4.9.1	Retrieving Blocks	28

<i>CONTENTS</i>	vii
4.9.2 Procedure for Saving a Block	28
4.10 SQL-Queries	29
4.10.1 Search Function	30
4.11 Hosting	30
5 Evaluation and Future Work	31
5.1 Evaluating Performance	31
5.2 Future Work	32
6 Summary and Conclusions	33
Bibliography	35
Abbreviations	39
List of Figures	39
List of Tables	41
List of Tables	43
A Installation Guidelines	47
A.1 Prerequisites	47
A.2 Setting Up the Database	47
A.3 Setting Up the Explorer	48
B Contents of the CD	49

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 maintain relations with merchants and 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 traditional system, Bazo is invite-only, meaning only the operator can add new users to the blockchain and thus only clients of the service provider can interact with it. [1].

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 Bazo blockchain 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 administrator-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 drawn from this thesis 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 an invite-only 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 make direct transactions between merchants and users or even between users itself using Bazo Coins. The merchants 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, since the service provider sets a fixed price in fiat for Bazo Coins. This ensures that merchants sell their products for an adequate amount of Bazo Coins. 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 during the trial.

2.1.1 Characteristics of Bazo

Similarly to Ethereum [2], 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, since 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 bandwidth, 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 in 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 fee exist. This is why these parameters can be changed on-the-fly by administrators using System Configuration Transactions.

- **Stake Transactions**

This transaction type was introduced with the Proof-of-Stake algorithm for Bazo [5]. A user can stake his current Bazo Coins in attempt to mine the block, instead of using computing power. This staking activity for each user is documented in Stake Transactions.

Every transaction requires a fee to be processed. These fees are collected by the miners who successfully mine a block. This incentivizes 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 to the blockchain 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* [1] 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 [5]. 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 and components were developed, which enhance the scope and functionality of Bazo.

- **Bazo Light Client** [3] 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** [4] 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 needing to install a native application. It also features Point-of Sale functionality, making the Bazo system useable in a real-life client-merchant situation.
- **Bazo REST Interface** [3] 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. More information on this process can be found in Subsection 4.4.2
- **Proof-of-Stake Algorithm** [5] The original Bazo implementation [1] depends on a Proof-of-Work consensus algorithm, that relies on processing power for mining blocks. In order to decrease energy consumption and security-related risks such as 51% attacks, a Proof-of-Stake algorithm has been implemented that lets users stake their coins, for a chance to successfully mine a block.

2.2 Blockchain Explorers and Analytics Platforms

Since many cryptocurrencies are open-source and public [2] [6], 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 [7], 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.

2.2.1 Blockexplorer.com [8]


This blockchain explorer was built for both the Bitcoin [6] and Bitcoin Cash [9] blockchains. A screenshot of the landing page can be seen in Figure 2.1. The frontend of the web application is called Insight UI [10] and is built using AngularJS [11], a JavaScript [12] framework. It interacts with the Insight API [13], the corresponding backend. Insight API consists of a REST and websocket API for Bitcore Node [14], a query and indexing service for the Bitcoin blockchain [6]. The source code for both frontend and backend are available on GitHub [10] [13].

2.2.2 Etherscan.io [15]

Etherscan is a blockchain explorer and statistics analysis platform for the Ethereum blockchain [2]. It uses Go Ethereum [16], an implementation of the Ethereum protocol in the Go language [17], in combination with Parity [19], a client for interacting with the Ethereum blockchain [2]. Etherscan is a closed source project. A screenshot of Etherscan's landing page can be seen in Figure 2.2.

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


Block Explorer

[News](#)
[Market](#)
[Bitcoin cash](#)
[Zcash](#)
[Blocks](#)
[Status](#)

[Buy Bitcoin with CCI](#)

✓ Conn 97 · Height 502195
🔍 Scan
BTC ~

Latest Blocks


Height	Age	Transactions	Mined by	Size
502195	5 minutes ago	2258		947684
502194	8 minutes ago	1928		956168
502193	14 minutes ago	2558		967558
502192	16 minutes ago	1539		982391
502191	an hour ago	2204		972566

See all blocks


Latest Transactions

Hash	Value Out
d9a07d26fa521dfef0022c8351513d6b45df736a204...	11.07606862 BTC
7cea35e01cce640a5416b98063c7095b071a0b2d148...	15.69578875 BTC
9f22b70ce9a6a5ff0a38bd99e2021e96ffad92fb1ab4...	0.04244318 BTC
ec12bf9f25e00a5d7a7950beea858f4482139337f5c0...	0.0594066 BTC
02aed3765103ab9bc27467a0d74bb2fd7e60672abf0...	0.00855252 BTC
e9412092a5027b83ebd6f05f762f85babe5cad7fb26...	0.210871 BTC
57da8ae89d108c9cc4a40fbd7d4280eaa869f7a19d7...	0.00220782 BTC
bc8dc2e2753364e03b7ee97eb7a6d0cefb92221dc7f...	0.01135264 BTC
a901c32b7be54070af83a8f2f08513405117ad7d084...	0.19365739 BTC
f5247b3d3daae60cc9f298e9455cc504ef851a6548b3...	0.20857158 BTC

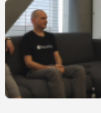
News



[UK Bitcoin Exchange CEO Kidnapped in Ukraine](#)



[Poloniex Review: An American Cryptocurrency Exchange with BTC, ETH, XMR, and USDT Trading Pairs](#)



[Rick Falkvinge Reacts to "Coinbase insider trading?" - This story was planted!](#)

About Block Explorer

Bitcoin Block Explorer is an open source web tool that allows you to view information about [blocks](#), [addresses](#), and [transactions](#) on the Bitcoin blockchain. The [source code](#) is on GitHub.

[What is bitcoin?](#)

Public Bitcoin API: Machine readable stats & blockchain info can be accessed directly through the [REST](#) and [Websockets](#) APIs.

Testnet is Bitcoin's sandbox. Block Explorer supports viewing both the [testnet](#) and [mainnet](#) blockchains.

Thanks to [Private Internet Access](#) for hosting the site. They provide a [VPN Service](#) that accepts Bitcoin.

Figure 2.1: Landing Page of blockexplorer.com [8]

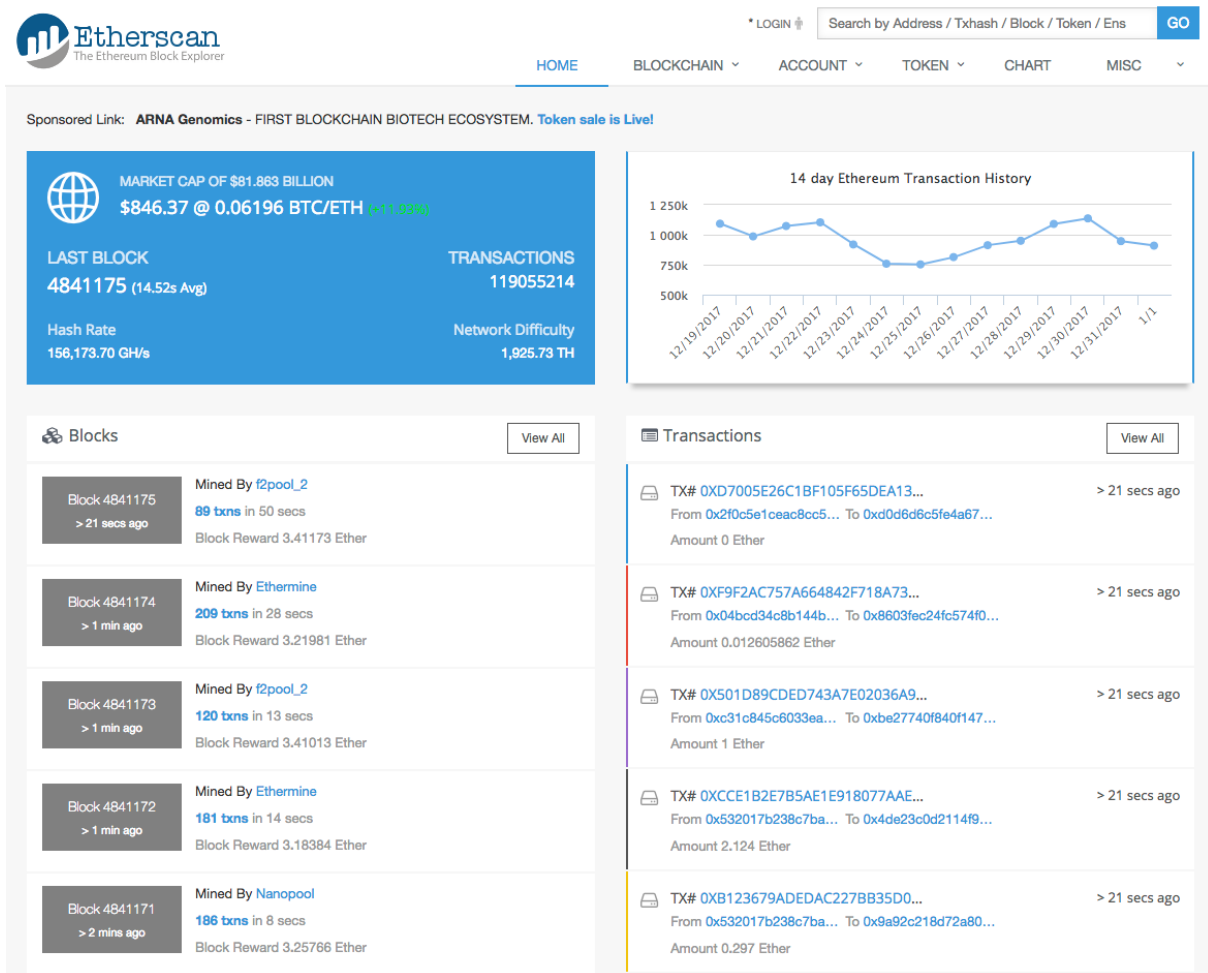


Figure 2.2: Landing Page of etherscan.io [15]

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

It has to be noted that, due to fundamental implementation differences between Bitcoin and Ethereum, even if components on both explorers aim to display data of the same informational value to the user, the way it is structured and presented may be different. For example Ethereum's accounts and Bitcoin's addresses are both representing users in the system, but are implemented and handled in different ways [2] [6].

- **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 transactions 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 [6], 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 displayed 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 of the overall functionality 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 *what* he is actually looking for, but finds that out automatically, given the hash a user enters is valid.

2.3.2 Functions and Components Specific to Blockexplorer.com

- **Status**

The status page contains information regarding the block explorer itself. Synchronization status between the explorer and the node it connects to are compared.

- **Blockchain-Related News Feed**

A section displaying news, related to the blockchain communities and markets. It is unclear whether this section is included on the explorer for generating income by featuring sponsored articles.

2.3.3 Functions and Components Specific to Etherscan.io

- **Statistical Information**

Etherscan offers a wide 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 Section 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 [1] and Stake Transactions from the Proof-of-Stake implementation [5]), 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.

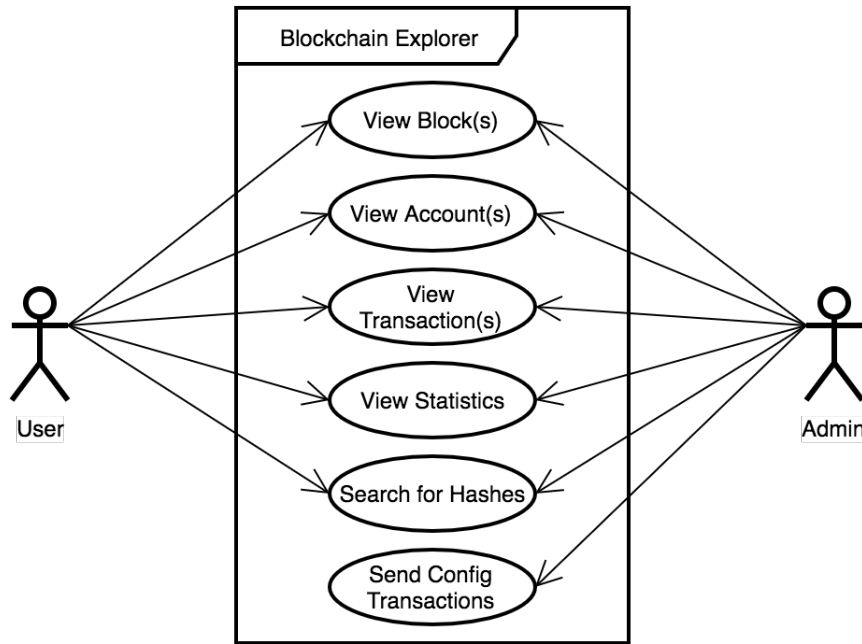


Figure 3.1: Use Cases of the Block Explorer

- **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 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.

- **Administrator 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 [3] 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.

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 backend 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.

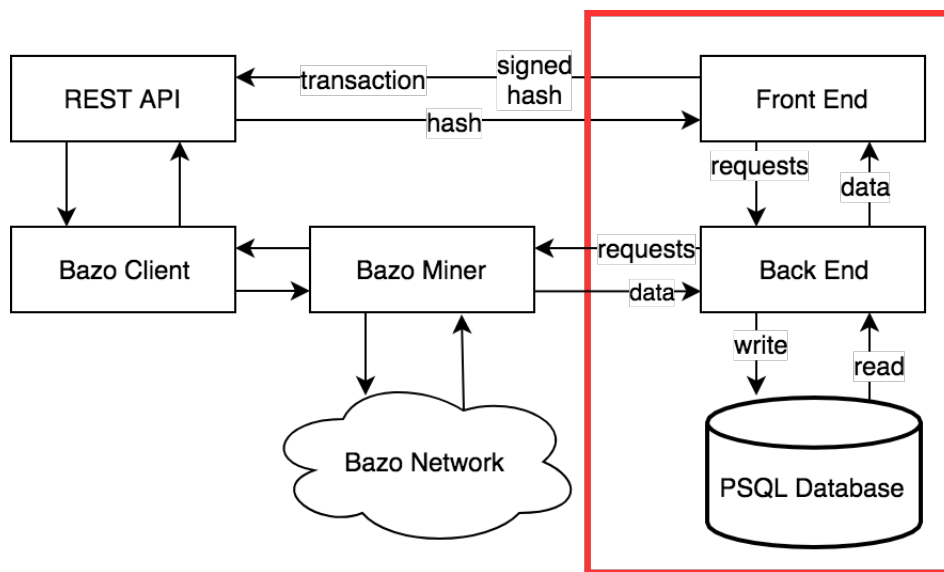


Figure 3.2: Structure of the Blockexplorer and Bazo Components

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,

Latest Blocks

Hash	Timestamp	NrFundsTX	NrAccTX	NrConfigTX
blocknumber1_l9oweuhy0qli24ow7iz	2017-11-07T13:33:21	6	0	0
blocknumber2_l86k7bsv0kvc9eli59w	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

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 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.

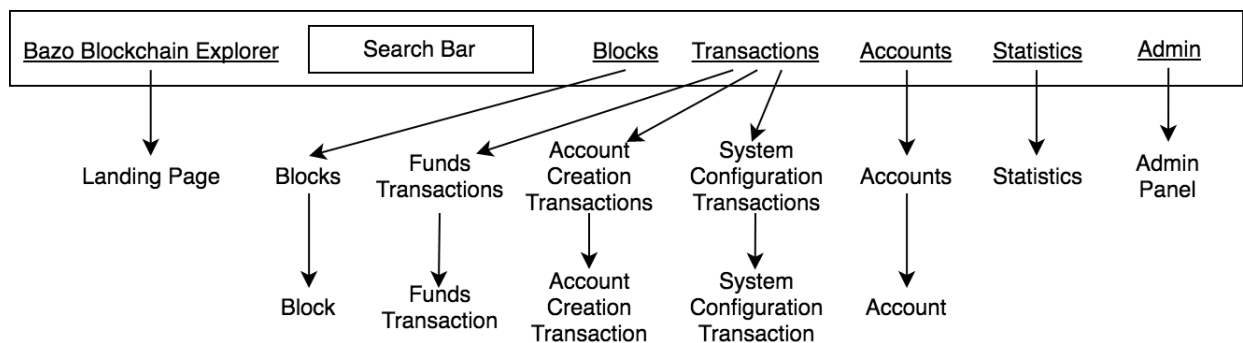


Figure 3.4: Functionality and Links on the Navbar

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. In the following frame, sample URLs and the pages they display are presented. The segment *host* is a placeholder for the domain, the block explorer is hosted on.

```
host/blocks — List of most recent blocks  
host/block/⟨⟨block_hash⟩⟩ — Details for block ⟨⟨blockhash⟩⟩  
host/adminpanel — Administrator panel  
host/tx/funds/⟨⟨hash⟩⟩ — Details for Funds Transaction ⟨⟨hash⟩⟩  
host/tx/acc — List of most recent Account Creation Transactions
```

3.3.3 Administrator Panel

Since there currently are five different types of System Configuration Transactions defined by IDs 1 through 5 [1], 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 and are derived from the public key, the administrator entered to log in. Detailed information about the verification process can be found in Section 4.6. 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 12-Hour 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 Bazo Block Explorer, because the data the explorer uses, is always structured in the same format. Blocks and transactions have attributes defined in the Bazo protocol, which were carried over for use in the database. Using tables, the data can be sorted efficiently by attributes like timestamps or transaction amounts, due to the flexibility of SQL queries. The Bazo protocol uses BoltDB [40] in its Miner program, which is a key/value based store, where the hash, a unique identifier, represents the key and encoded block or transaction information represents the value. When users perform a search on the web app, they exclusively search for hashes, unique to each object. This enables defining the hashes as primary keys in the tables. However, the database performs additional queries using that hash. For example, when a user searches for an account, the system also queries the Funds Transaction table with the same hash to look for all transactions that account was a part of. This also speaks for using SQL queries.

As mentioned in Section 3.2, additional information regarding the blockchain data needs to be calculated and stored, so additional tables for statistical data need to be defined and maintained. For optimal performance the database is hosted on the same server as the block explorer, to ensure efficient read or write times. The statistical data also needs to be calculated either in a fixed interval or after a certain event has happened.

3.4.1 Tables

The following tables and attributes need to be defined to accommodate all Bazo objects and data. Where needed, indexes have been added to increase performance. Primary keys are underlined, while indexes are in italics.

- **Blocks** Header, Hash, PreviousHash, Nonce, *Timestamp*, Timestring, MerkleRoot, Beneficiary, NrFundsTx, NrAccTx, NrConfigTx, FundsTxData, AccTxData, ConfigTxData
- **Accounts** Hash, *Address*, Balance, TxCount
- **Funds Transactions** Header, Hash, BlockHash, Amount, Fee, Txcount, *Sender*, Recipient, Timestamp, Signature
- **Account Creation Transactions** Header, Hash, BlockHash, Issuer, Fee, PublicKey, Timestamp, Signature
- **System Configuration Transactions** Header, Hash, BlockHash, Id, Payload, Fee, TxCount, Timestamp, Signature
- **Parameters** Timestamp, BlockSize, DifficultyInterval, MinimumFee, BlockInterval, BlockReward
- **Stats** TotalSupply, NrAccounts, Timestamp

Chapter 4

Implementation

This chapter documents the structuring and 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. The majority of the application was programmed in Go (Golang to avoid confusion). Go is a language that started development in 2007, as an answer to the changing computer landscape, as no new major systems language has emerged over a long time [18]. The aim of Go is to support new computing concepts such as multicore computing and make development overall a less time consuming task.

4.1 Packages of the Block Explorer

The backend of the web app is written in Golang [17], which is well suited for writing web applications, due to its included `net/http` [22] and `html/template` [21] packages. Since the Bazo Blockchain was also written in Golang [1], compability between the two programs was ensured and certain packages of Bazo were used in the Bazo Block Explorer. The program is split up into distinct modules, to separate functionality. Dependencies are presented in Figure 4.1

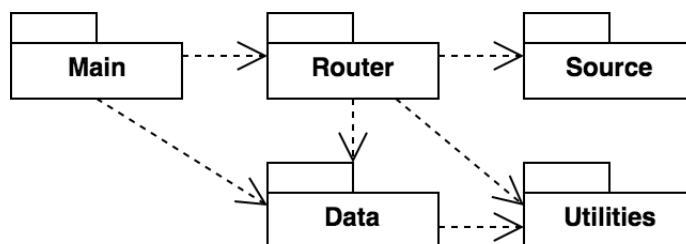


Figure 4.1: Package-Diagram of the Blockchain Explorer

- **Router**

The Router functionality is described in Section 4.5. It handles all requests made to the application, from the browser. The appropriate data is then loaded and sent in response.

- **Data**

This module handles all data processing of the application, specifically the retrieval of data from the blockchain and the saving of that data in the block explorer's own database. It is described in Section 4.9 and Section 4.10.

- **Utilities**

The Utilities module contains functionality that is used by both Router and Data modules such as the definition of structs, handling of cookies and smaller helper functions.

- **Source**

All Gohtml templates, JavaScript files and CSS files are contained in this module. Gohtml templates get passed to the router, which then get converted to HTML, whereas the other files get served to the website to be used client-side.

4.2 HTML Templates

To interact with the Golang-Backend of the application, Gohtml templates [21] have been used to define the markup of the pages. On 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, gohtml can create a new `` 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 the rendering of the chart on the statistical information page mentioned in Subsection 4.4.3 and one special case discussed in Subsection 4.4.1, 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. To appropriately handle relative paths on the templates, a *UrlLevel* string depending on the current URL, for example `../`, is always passed to the template, in addition to the regular blockchain data. This string is prepended dynamically to all links in the navbar, so that the same templates can be used on different URLs.

4.2.1 Reusable Modules

- **Navbar**

Since the Navbar is featured on every page of the explorer, it was defined as a reusable template.

- **HTML Head**

Every page features the same links to stylesheets and meta information. Thus the content of the HTML Head is reusable.

- **Public Key Modal**

The modal that expects a public key to authenticate a user, is accessible from the navbar and therefore can be defined as a template.

- **Private Key Modal**

For sending transactions from the admin panel, a modal that expects an administrator's private key is derived from the login modal and kept as a template.

- **Script Imports**

For JavaScript functionality, a group of *script* tags have been pooled, similarly to the HTML head's imports.

4.2.2 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.

In Listing 4.1, a simplified excerpt from the landing page Gohtml file is presented. Two tables, one for block-hashes and one for Funds Transaction hashes make use of templating. A struct containing a slice of blocks and a slice of Funds Transactions, gets passed to the template. On line 8, the *Blocks* slice (attribute) of the passed struct gets iterated through using the *range* and *end* keywords. For every block, a new table row gets created, containing a link to the specific block page. The period now represents the struct of a single block and its attributes can be accessed by appending the desired attribute. Similarly on line 23, the transaction slice in the struct is iterated through. On line 25, the *Hash* attribute does not access the block's hash anymore, but the Funds Transaction's, as the period now represents a transaction.

```

1 <table class="table">
2   <thead>
3     <tr id="header-row">
4       <th>Hash</th>
5     </tr>
6   </thead>
7   <tbody>
8     {{range .Blocks}}
9     <tr>
10      <td><a href="block/{{.Hash}}">{{.Hash}}</a></td>
11    </tr>
12    {{end}}
13  </tbody>
14 </table>
15
16 <table class="table">
17   <thead>
18     <tr id="header-row">
19       <th>Hash</th>
20     </tr>
21   </thead>
22   <tbody>
23     {{range .Txs}}
24     <tr>
25      <td><a href="tx/funds/{{.Hash}}">{{.Hash}}</a></td>
26    </tr>
27    {{end}}
28  </tbody>
29 </table>

```

Listing 4.1: Block and Funds Transaction Tables Accessing a *blocksandtx* Struct

4.3 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 functionality, 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. Figure 4.2 presents the landing page of the block explorer, featuring Bootstrap components.

Bazo Blockchain Explorer

BlocksTransactionsAccountsStatusAdminLogin

Search through blocks, transactions and accounts...

Search

Latest Blocks

Hash	Timestamp	NrFundsTX	NrAccTX	NrConfigTX
000e00d378c4308e329d80976cb5db4178c1ea0fc7d6...	30 Jan 2018 10:37	0	0	0
000e01ce0801eea89e1156a48e5e7382934993968e38...	30 Jan 2018 10:37	0	0	0
00ca01dc465a51d3bed70e4d5ada75051ca3ae4afccbd...	30 Jan 2018 10:35	0	0	0
004401297264e60245e5fe7490dc9abac336bc1129ec0...	30 Jan 2018 10:35	0	0	0

Latest Funds Transactions

Hash	From	To	Amount	Fee
b8531134f315a273b55573...	bbb54163468f90b773bc0c...	fdc2549fbfc6b9e4f065e59...	1	1
2cd0a0d49abdd19727623...	bbb54163468f90b773bc0c...	fdd360674b17831e004eb7...	1	1
4ef862d5ff078ad1c841d76...	bbb54163468f90b773bc0c...	a8724943dd6c95c7643cc...	1	1
317034e893ca3a00fbdd23...	bbb54163468f90b773bc0c...	09afa0abd1c853211349e7...	1	1

Figure 4.2: Landing Page of the Explorer

4.4 Client-Side Logic

The explorer uses a JavaScript [12] framework and different libraries to implement client-side logic, which was used to implement the sending of transactions and the rendering of a chart.

4.4.1 Vue.js and Libraries

For the process described in subsection 4.4.2, the Vue.js JavaScript framework [24] is used. Vue.js allows for easy client-side scripting with minimal setup. It links a vue app which is contained in a JavaScript file to an *id*-tag on a HTML file and has access to all objects such as input fields and buttons under the *div*-tag that contains the linked *id*-tag. To run a vue app, vue and all the additional libraries have to be imported on the HTML file. Additionally to the standard vue import, Axios , a promise based HTTP client [25] has been used to make GET requests from the frontend of the application. To extract cookies for verification purposes, vue-cookies [26] was used.

4.4.2 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.

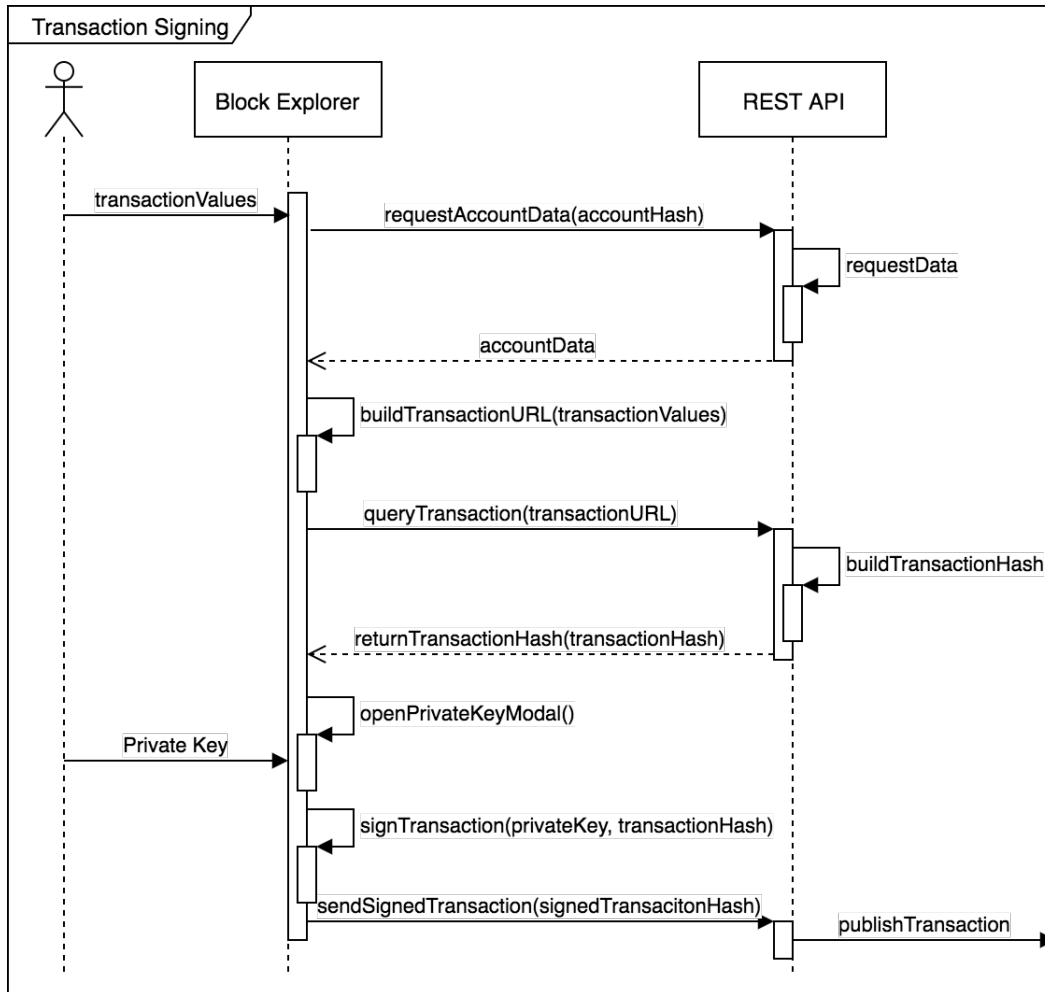


Figure 4.3: Sequence Diagram of How the Application Builds and Signs Transactions

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 [3] 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 4.3. 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 transaction fee. When the user hits *Send*, JavaScript code in the Vue.js app extracts the public key stored in a cookie [26] (user verification is described in Section 4.6) 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.

4.4.3 AnyChart

To render the chart that displays the 12-hour transaction history of the network, the AnyChart JavaScript charting library [36] was used. It allows the creation of over 70 different charts and the only thing it requires the backend to do, is format the data in the correct way. On the HTML-template, script imports can be used to make use of the library, which makes the use of a packet manager unnecessary. In addition to imports, it requires a JavaScript function, that defines the type, layout and the location of the diagram on the final page. The location is defined by an HTML *id* tag. In the Bazo Block Explorer's case, a spline chart was chosen to display the data in a line. The backend requests all blocks that were mined in the last 12 hours, calculates the total number of transactions for each hour and saves that information in a struct. That struct gets converted into JSON format and then converted to a string. This string then gets passed to the template and used by the AnyChart function, which converts the string back to JSON format and uses it to build the chart.

4.5 Router

The router handles all requests made to the application. The process of initializing the router is detailed in Listing 4.2. When *InitializeRouter()* is called, a new router is declared and registers all possible routes to it. 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. "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. Variables in the routes such as hashes can be defined by prepending a colon before a segment in the URL. For example in listing 4.2 on line 6, when */blocks/:hash* is called using a GET request, where *:hash* can be any string variable, the router automatically parses the hash to be used by the *getOneBlock()* function that gets called after the request. Except for the */login* and */search* routes, which are called using POST methods, all routes are called via GET request.

```

1 func InitializeRouter() *httprouter.Router {
2     tpl = template.Must(template.ParseGlob("source/html/*"))
3     router := httprouter.New()
4
5     router.GET("/", getIndex)
6     router.GET("/blocks", getAllBlocks)
7     router.GET("/block/:hash", getOneBlock)
8     router.GET("/tx/funds", getAllFundsTx)
9     router.GET("/tx/funds/:hash", getOneFundsTx)
10    router.GET("/tx/acc", getAllAccTx)
11    router.GET("/tx/acc/:hash", getOneAccTx)
12    router.GET("/tx/config", getAllConfigTx)
13    router.GET("/tx/config/:hash", getOneConfigTx)
14    router.GET("/account/:hash", getAccount)
15    router.GET("/accounts", getTopAccounts)
16    router.GET("/stats", getStats)
17    router.POST("/search", searchForHash)
18    router.POST("/login", loginFunc)
19    router.GET("/logout", logoutFunc)
20    router.GET("/adminpanel", adminfunc)
21
22    router.ServeFiles("/source/*filepath", http.Dir("source"))
23
24    return router
25 }

```

Listing 4.2: Initialization of the Router

4.6 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 sent over the network to check its access rights. The chosen verification process is as follows: Upon requesting to log in, a modal appears, where a user is requested to enter his *public* key. The modal is presented in Figure 4.4. This public key then gets sent to the Bazo Interface [3], 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 only page requiring verification, 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 Section 4.4.2 when the process of sending a Configuration Transaction is initiated.

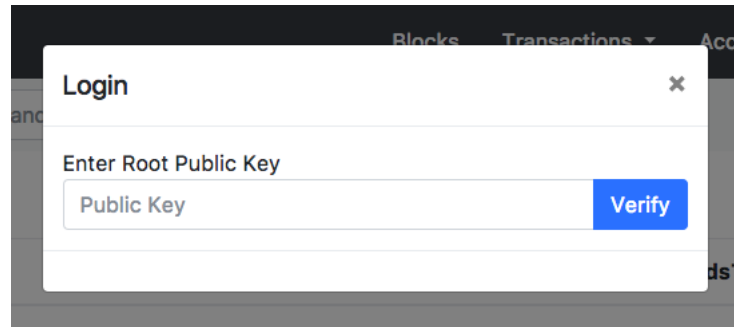


Figure 4.4: Login Modal Requesting the User's Public Key

4.7 Concurrency

The application makes use of Golang's built-in concurrency functionality. By calling `runDB()` in the main function as a Goroutine by prepending the keyword `go`, the mechanism that automatically updates the database with new blockchain data, runs separately from the router's request handling. This means that the web app can still be used, while the database is being updated with new records.

4.8 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, a mining date, or a transaction time. The struct definition of an Account Creation Transaction in the explorer can be seen in Listing 4.3

```

1 type Acctx struct {
2     Hash string
3     BlockHash string
4     Issuer string
5     Fee uint64
6     PubKey string
7     Timestamp int64
8     Signature string
9     UrlLevel string
10 }
```

Listing 4.3: Struct Definition of an Account Creation Transaction

4.9 Data Transfer

This section explains how blockchain data is retrieved from a node and stored. After requesting the block-data, the block extracts transaction information to make additional requests for transaction data. All retrieved information is then stored in the database.

4.9.1 Retrieving Blocks

When *runDB()* is called as a Goroutine, as mentioned in Section 4.7, all existing blocks are requested from the blockchain and stored in the database. To request a block and its data, the blocks's hash needs to be known. Integrated into the miner is the functionality that, if a block is requested but *nil* is passed to the function, the most recent block in the miner's storage component (known as *newestBlock*) is returned and subsequently saved to the explorer's database. Starting with this *newestBlock*'s *previousHash* attribute, the remaining blocks get requested and saved recursively until the genesis block is reached. Once all existing blocks have been saved, the system regularly checks whether there have been any new blocks mined and retrieves them in a similar fashion. The way a block is requested is similar to how Bazo Miners communicate with each other [1]. The application imports the miner's p2p package and utilizes its messaging system to connect to the bootstrap server and make requests to the network.

4.9.2 Procedure for Saving a Block

```

1 func SaveBlockAndTransactions(oneBlock *protocol.Block) {
2
3     for _, accTxHash := range oneBlock.AccTxData{ . . . }
4
5     for _, fundsTxHash := range oneBlock.FundsTxData{
6         fundsTx := reqTx(p2p.FUNDSTX_REQ, fundsTxHash)
7         convertedTx := ConvertFundsTransaction(fundsTx.(*protocol.FundsTx),
8                                             oneBlock.Hash,
9                                             fundsTxHash)
10
11         UpdateAccountData(convertedTx)
12         WriteFundsTx(convertedTx)
13     }
14
15     for _, configTxHash := range oneBlock.ConfigTxData{ . . . }
16
17     convertedBlock := ConvertBlock(oneBlock)
18     WriteBlock(convertedBlock)
19 }

```

Listing 4.4: Saving a Block and Its Transactions to the Database

Every block retrieved from the miner goes through the algorithm presented in Listing 4.4. Since every block contains a slice of transaction hashes for every transaction type, loops that iterate through the slices exist for every transaction type (lines 3, 5 and 15). Account Creation Transactions and Configuration Transactions have been minimized in the example, to highlight how Funds Transactions are saved in the database. All processes are similar and only differ in what information is saved and updated in their respective tables.

Starting on line 5, a request using the Funds Transaction's hash, to retrieve the full transaction data is made to the network. The returned transaction of type *protocol.FundsTx* is then converted to a struct that is usable to the database. For the conversion, the block's hash and the transaction hash, extracted from the slice, are needed as well. *UpdateAccountData(convertedTx)* then updates *Balance* and *TxCount* attributes for all accounts featured in that transaction in the database's *Accounts* table. The last step concerning the transaction is *WriteFundsTx(convertedTx)*, which saves the transaction in the *FundsTx* table in the database. After all slices have finished processing, the block itself is converted to a database-friendly format and saved in the *Blocks* table.

4.10 SQL-Queries

On startup, the first database queries made, are called from the *DBSetup()* function. First, all existing tables and data get dropped from the database. Then, all tables get created again to start the application with a clean database. Once the explorer begins requesting and saving blocks from the network, *INSERT* queries begin writing all data in their respective tables, block by block. *SELECT* statements are used when requesting one or more rows of data from a table. Using Golang's *database/sql* package, placeholder values, such as \$1, \$2, etc can be used to automate construction of the SQL-Statement string. The actual variables can be passed as additional arguments to the function that is used to query the request, as seen in Listing 4.5: *UrlHash* represents the hash, the user has requested. Together with the statement string, it gets passed to *db.QueryRow* and saved to *row*. The next step is mapping all values of *row* to a struct that matches all columns of the query result using the *Scan()* function of the *row* object. Additionally to write and read functions for blocks, transactions and accounts, functions that update and return the statistical information also exist.

```

1 sqlStatement := 'SELECT hash , blockhash , issuer , fee , pubkey , signature
2                   FROM acctx
3                   WHERE hash = $1; '
4 row := db.QueryRow(sqlStatement , UrlHash)
5 var returnedrow acctx
6 row.Scan(&returnedrow.Hash , &returnedrow.BlockHash , &returnedrow.Issuer ,
7         &returnedrow.Fee , &returnedrow.PubKey , &returnedrow.Signature)

```

Listing 4.5: Requesting a Specific Account Creation Transaction

4.10.1 Search Function

When a user enters a hash he wants more information on, it gets passed to the backend and processed by the *searchForHash* function, which makes a request with the entered hash on each table, until a table returns a result. If no data regarding that hash could be found, the user is notified.

4.11 Hosting

In a test-run, the Bazo Block Explorer was hosted as a Heroku app on their cloud hosting platform [31]. The PSQL database was also ported to run on Heroku. Since during development, both the explorer and the database were hosted locally, changes to how the application retrieves the target port and database connection string needed to be made. On Heroku, the application and the database were connected by adding the database as an add-on to the Heroku App. This means that the correct port and database location can be extracted using Golang's *os* package [32]. These values can also be edited manually using Heroku's dashboard.

Required for the Golang application to compile and run on Heroku, is the *heroku/go* buildpack [33], which is used to install dependencies and configure the environment. Additionally the application needs a procfile, which specifies how Heroku will name and treat the application [34]. In the block explorer's case it was defined as a *web* app.

web: BlockExplorerHeroku

Godep needs to be used to manage dependencies of the app [35]. Godep saves all dependencies in JSON [28] format, which are then interpreted by the *heroku/go* buildpack [33]. When the application is pushed to heroku, it gets built using all dependencies and is useable immediately.

Chapter 5

Evaluation and Future Work

5.1 Evaluating Performance

In order to test the Block Explorer’s performance, time measurements were taken for certain actions. First, upon starting the explorer up and letting it fill its database, it was measured how long it took to process a blockchain consisting of more than ten thousand blocks. It is important to note, that the majority of the blocks were empty, meaning no transactions were included in them. The miner, the explorer connects to was hosted on an Ubuntu [38] Virtual Machine, located at the University of Zurich. In Table 5.1, the average load times on different machines are presented.

Other measurements that were taken, include the time it took to request and display single items, a list of items and the search function. The results are displayed in Table 5.2. In this test, the explorer itself was hosted on an Ubuntu Virtual Machine at the University of Zurich and contained the blockchain with over ten thousand blocks. HTML Load Time measured the time it took from the request to the server, until it responded with the HTML file containing the requested blockchain data. Since other files from other sources, such as Bootstrap files, have shown to have irregular load times, a Low Load Time and a High Load Time have been added for measuring load times of the entire page. Testing was conducted using a Computer, connected the University’s WiFi and timed using Google Chrome’s [39] developer tools.

Table 5.1: Load Times to Copy Ten Thousand Blocks From a Miner

Computer	CPU	RAM	Load Time in Minutes
MacBook Pro	2.9 GHz Intel Core i5	8 GB	2:11
Ubuntu VM			2:28

Table 5.2: Load Times of the Web Application (Ld Tm Stands for Load Time)

Query	HTML Ld Tm	Ld Tm Low	Ld Tm High
Block	30 ms	135 ms	220 ms
Funds Transaction	33 ms		
Account with Transactions			
100 Most Recent Blocks			
Status Page			
Search of a Block			

5.2 Future Work

At time of writing, the applications's backend functionality of retrieving blocks from a miner, saving them to the database and returning them upon request is separated from the frontend that displays the returned data. This enables the possibility of switching out the frontend that displays HTML content for a REST API that returns the blockchain data in JSON format. Uses of such an interface could be an integration into the Payment System PWA, which displays data only on request. For example the PWA could display all confirmed transactions for an account in order to make the Payment app more informative, as seen in Figure 5.1.

Another feature, the current block explorer lacks, is frontend optimization for mobile devices. Dynamically styled tables and UI components are needed to accommodate for the many resolutions and aspect ratios of mobile devices. To increase performance, especially on mobile devices, CSS micro-frameworks, such as Furtive [41] or Pure.CSS [42] could be used to keep style sheets at a minimum size. This would also prevent the need for resource-heavy Bootstrap imports and links.

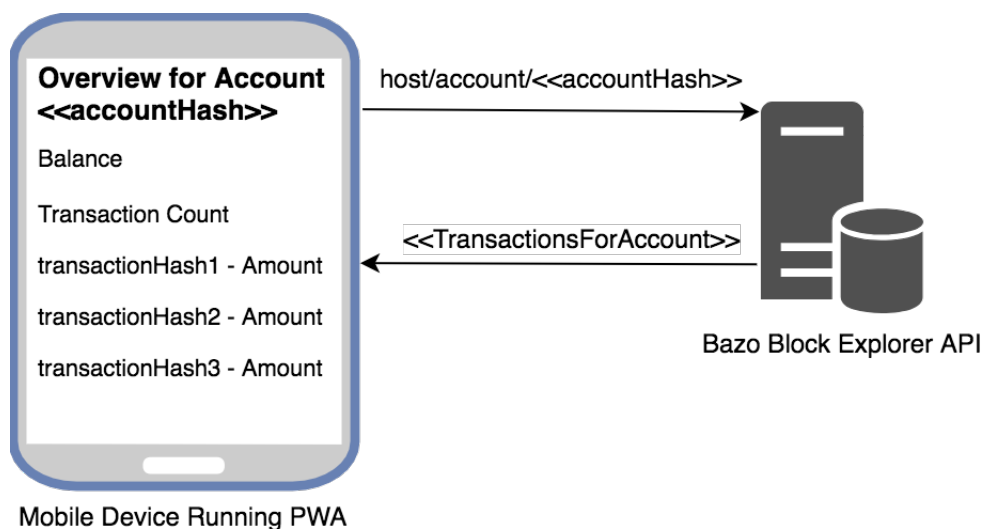


Figure 5.1: Device Running PWA Payment System Requesting a User's Transactions

Chapter 6

Summary and Conclusions

This thesis documented the development of a blockchain explorer for the Bazo blockchain, heavily based on requirements elicited from an analysis of existing blockchain explorers and input from the financial service provider. The Bazo Blockchain Explorer allows users of the blockchain to inspect blockchain data through a graphical user interface, requiring no installation to use, since the application is available on the internet as a web app. Additionally, the operator of the system can set system parameters from the application, using an administrator panel.

Explorers for the top two cryptocurrencies (at time of writing) were used for the analysis [37]. Design decisions were then made, which included the structure of the system in relation to the existing Bazo infrastructure and concrete functionality of the application. Implementation-related topics such as high level software architecture and used technologies were documented in the chapter following design-decisions. Also included are code-snippets, that highlight certain algorithms crucial to the successful execution of the program. An evaluation of the developed application follows, discussing the performance and responsiveness of the web app, while working with a large amounts of data. Possible functionality that extends the current scope of the blockchain explorer is presented in the Future Works section.

The Bazo Blockchain Explorer makes the blockchain and its data transparent and accessible to all users. In conjunction with the Bazo PWA Payment System, users can now make transactions and check if they have been accepted by the system, without having to use command-line tools. This allows a test-run of the Bazo blockchain using real customers, since no programming knowledge is required. Due to the open-source nature of the blockchain explorer, anyone can host their own version of it, as long as the IP address of the bootstrap server is known.

Bibliography

- [1] Livio Sgier: Bazo - A Cryptocurrency from Scratch, Bachelor Thesis, University of Zurich, August 23, 2017
- [2] Ethereum - A Next-Generation Smart Contract and Decentralized Application Platform, <https://github.com/ethereum/wiki/wiki/White-Paper>, accessed on January 10, 2018
- [3] Marc-Alain Chetelat: A Mobile Light Client for Bazo, Master Thesis, University of Zurich, March 11, 2018
- [4] Jan von der Assen: A Progressive Web App (PWA)-based Mobile Wallet for Bazo, Bachelor Thesis, University of Zurich, February 1, 2018
- [5] Simon Bachman: Proof of Stake for Bazo, Bachelor Thesis, University of Zurich, February 28, 2018
- [6] Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System, <https://bitcoin.org/bitcoin.pdf>, accessed on January 10, 2018
- [7] Ledger Bitcoin Wallet, <https://www.ledgerwallet.com/apps/bitcoin>, accessed on January 10, 2018
- [8] Blockexplorer.com, <https://blockexplorer.com/>, accessed on January 10, 2018
- [9] Satoshi Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System <https://www.bitcoincash.org/bitcoin.pdf>, accessed on January 10, 2018
- [10] Bitpay: Insight UI, <https://github.com/bitpay/insight>, accessed on January 10, 2018
- [11] AngularJS, <https://angularjs.org/>, accessed on January 10, 2018
- [12] JavaScript, <https://developer.mozilla.org/bm/docs/Web/JavaScript>, accessed on January 10, 2018
- [13] Bitpay: Insight API, <https://github.com/bitpay/insight-api>, accessed on January 10, 2018
- [14] Bitpay: Bitcore Node, <https://github.com/bitpay/bitcore-node>, accessed on January 10, 2018

- [15] Etherscan.io, <https://etherscan.io/>, accessed on January 10, 2018
- [16] Ethereum: Go Ethereum, <https://github.com/ethereum/go-ethereum>, accessed on January 10, 2018
- [17] The Go Programming Language, <https://golang.org/>, accessed on January 10, 2018
- [18] History of Go, <https://golang.org/doc/faq#Origins>, accessed on January 16, 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
- [31] Heroku, <https://www.heroku.com/>, accessed on January 11, 2018
- [32] Go Package Os, <https://golang.org/pkg/os/>, accessed on January 11, 2018
- [33] Heroku Go Buildpack, <https://github.com/heroku/heroku-buildpack-go>, accessed on January 11, 2018
- [34] Heroku Procfiles, <https://devcenter.heroku.com/articles/procfile>, accessed on January 11, 2018
- [35] Godep, <https://github.com/tools/godep>, accessed on January 11, 2018
- [36] AnyChart, <https://www.anychart.com/>, accessed on January 16, 2018

- [37] Cryptocurrency Market Capitalizations , <https://coinmarketcap.com/>, accessed on January 23, 2018
- [38] Ubuntu, <https://www.ubuntu.com/>, accessed on January 23, 2018
- [39] Google Chrome Browser, <https://www.google.com/chrome/browser/desktop/index.html>, accessed on January 23, 2018
- [40] Bolt, <https://github.com/boltdb/bolt>, accessed on January 24, 2018
- [41] Furtive, <http://furtive.co/>, accessed on January 29, 2018
- [42] Pure.CSS, <https://purecss.io/>, accessed on January 29, 2018
- [43] Bazo Block Explorer Source Code, <https://github.com/lucBoillat/BazoBlockExplorer>, accessed on January 30, 2018

Abbreviations

PoS	Point of Sale
Navbar	Navigation Bar
URL	Uniform Resource Locator
HTML	Hypertext Markup Language
UI	User Interface
JS	JavaScript
SQL	Structured Query Language
PSQL	PostgreSQL
GUI	Graphical User Interface
REST	Representational State Transfer
API	Application Programming Interface
UTXO	Unspent Transaction Output
HTTP	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
DB	Database
P2P	Peer to Peer
OS	Operating System
PWA	Progressive Web App
Tx	Transaction
FundsTX	Funds Transaction
AccTx	Account Creation Transaction
ConfigTx	(System) Configuration Transaction
IP	Internet Protocol

List of Figures

2.1	Landing Page of blockexplorer.com [8]	7
2.2	Landing Page of etherscan.io [15]	8
3.1	Use Cases of the Block Explorer	12
3.2	Structure of the Blockexplorer and Bazo Components	13
3.3	Mockup of the List of Most Recent Blocks	15
3.4	Functionality and Links on the Navbar	15
4.1	Package-Diagram of the Blockchain Explorer	19
4.2	Landing Page of the Explorer	23
4.3	Sequence Diagram of How the Application Builds and Signs Transactions	24
4.4	Login Modal Requesting the User's Public Key	27
5.1	Device Running PWA Payment System Requesting a User's Transactions	32

List of Tables

5.1	Load Times to Copy Ten Thousand Blocks From a Miner	31
5.2	Load Times of the Web Application (Ld Tm Stands for Load Time)	32

Listings

4.1	Block and Funds Transaction Tables Accessing a <i>blocksandtx</i> Struct	22
4.2	Initialization of the Router	26
4.3	Struct Definition of an Account Cration Transaction	27
4.4	Saving a Block and Its Transactions to the Database	28
4.5	Requesting a Specific Account Creation Transaction	29

Appendix A

Installation Guidelines

This chapter details the procedure to install and run the explorer on a computer or virtual machine with a fresh installation of Ubuntu. The source code is available on Github [43].

A.1 Prerequisites

The explorer requires a working installation of Golang (version 1.9.2 was used to develop the application). The most recent version can be found on their website, along with installation instructions. A *go* folder has to be created in the user's home directory after installation.

```
https://golang.org/dl/
```

Since the explorer requires a PSQL database, Postgres (version 9.6.6) is also required.

```
sudo apt install postgresql postgresql-contrib
```

A.2 Setting Up the Database

After installing Postgres, a database super user called 'postgres' is automatically created. With that user, a new user has to be created, which will be used to manage the block explorer's database. First, the psql shell needs to be opened using the postgres account.

```
sudo -i -u postgres  
  
psql
```

The new database-user and his password has to be created. Additionally the database 'blockexplorerdb' has to be created. Username and password of the new user can be chosen freely, however the database needs to be called 'blockexplorerdb'. The new user needs all privileges on the newly created database as well.

```
CREATE ROLE explorer WITH LOGIN PASSWORD 'bazo';  
  
CREATE DATABASE blockexplorerdb;  
  
GRANT ALL PRIVILEGES ON DATABASE blockexplorerdb TO explorer;  
  
\quit
```

If the terminal still displays the user 'postgres' as the current user, it is now time to switch back to the computer's regular user, as no further psql related actions are needed. If switching users requires a password for the 'postgres' account, a restart of the terminal will also log in the regular user.

A.3 Setting Up the Explorer

Using *go get*, a compiled binary of the explorer and all its dependencies get installed.

```
go get github.com/lucBoillat/BazoBlockExplorer
```

Since the static HTML files do not get included in the compiled binary, the *source* folder has to be copied from Golang's *src* directory to the *bin* directory, where the compiled block explorer binary is located. The following frame's folder structure assumes a default Golang installation.

```
cd GODIRECTORY/src/github.com/lucBoillat/BazoBlockExplorer  
  
sudo cp -r source/ ../../../../bin  
  
cd GODIRECTORY/bin
```

From the explorer's directory, it can now be started using the following arguments. The username and password are the values defined in the database setup step.

```
./BazoBlockExplorer :PORT USERNAME PASSWORD
```

A running Bazo Miner application is required on the bootstrap server.

Appendix B

Contents of the CD

- Bazo Blockchain Explorer Source Code
- Latex Source Code
- Thesis (PDF)
- Intermediate Presentation
- Final Presentation