



University of  
Zurich<sup>UZH</sup>

# A Progressive Web App (PWA)-based Mobile Wallet for Bazo

*Jan von der Assen  
Zurich, Switzerland  
Student ID: 14-719-132*

Supervisor: Thomas Bocek, Bruno Rodrigues, Hamza Bedrija  
Date of Submission: January 31, 2018



# Abstract

Ein Finanzdienstleister aus Zürich unterhält ein Bonusprogramm um die Verwendung von Kredit- und Debitkarten zu belohnen. Der traditionelle Prozess wie Bonuspunkte gegen Produkte getauscht werden können, bedeutete, dass Verträge zu Partnerfirmen unterhalten werden müssen. Dieser Prozess führte dazu, dass das Programm bei Kunden kaum bekannt wurde. Aus diesem Grund wurde an der Universität Zürich die Bazo Kryptowährung entwickelt, welche nur eingeladenen Teilnehmern offen steht. Um den Finanzdienstleister die Technologie evaluieren zu lassen, wurden weitere Entwicklungsaufwände betrieben um andere übliche Applikationen für Kryptowährungen bereitzustellen. Der Fokus dieser Arbeit liegt auf der Planung, Entwicklung und Evaluation einer Zahlungs-Applikation für die Bazo Währung. Neben dieser Entwicklung befasst sich diese Arbeit auch mit der Erfassung des aktuellen Stands von Web APIs, die normalerweise nur auf nativen Applikationen zur Verfügung stehen. Weiter wurde untersucht wie native APIs einer Web Applikation zur Verfügung gestellt werden können. Konkret wurde dazu ein Proof-of-Concept erstellt, womit native Funktionen einer Web Applikation bereit gestellt werden. Weiter wurden Schnittstellen zu existierenden Systemen, wie dem Bonusprogramm und der Kryptowährung, geplant und entwickelt. Ausserdem wurde im Rahmen der Arbeit evaluiert wie die existierende Client Applikation auf das mobile Betriebssystem Android portiert werden kann.

A financial service provider in Zurich maintains a bonus program to incentivize credit and debit card usage. The process of exchanging bonus points against goods implied efforts to maintain contracts with partners and led to low usage of the program by customers. This led to the development of the Bazo cryptocurrency at the University of Zurich, which is an invite-only, blockchain-based currency. In order to let the financial service provider evaluate the technology, further development efforts were taken to enrich the ecosystem of the currency. The scope of this thesis is the design, development and evaluation of a web-based payment application for the Bazo cryptocurrency. Besides the development of the application, this thesis explores related topics, such as a comparison of Web APIs with native APIs by developing a Proof of Concept for the Android operating system. Further, the development efforts for the required interfaces to the currency and to the companys existing infrastructure are described. Finally, a mobile port of the existing client application for the Bazo currency targeted to the Android operating system was created and evaluated.



# Acknowledgments

I would like to express my gratitude to my supervisor, Dr. Thomas Bocek, for sharing his passion and experience within the project. He has always provided an interesting view on various topics and thus made this thesis an exciting opportunity.

Further, I would like to thank the other members of the project for their competent work and collaboration.

Lastly, I would like to thank Prof Dr. Burkhard Stiller, head of the Communication Systems Group, for making it possible to write my thesis within the scope of this project.



# Contents

<b>Abstract</b>	<b>i</b>
<b>Acknowledgments</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Description of Work . . . . .	2
1.3 Outline . . . . .	2
<b>2 Background and Related Work</b>	<b>3</b>
2.1 Background . . . . .	3
2.1.1 Web based Wallets . . . . .	3
2.1.2 Progressive Web Applications . . . . .	4
2.1.3 Traditional payment process . . . . .	4
2.1.4 Envisioned payment process . . . . .	4
2.2 Related Work . . . . .	5
2.2.1 Bazo . . . . .	5
2.2.2 Bazo Client Implementations . . . . .	5
2.2.3 Coinblesk . . . . .	6
2.2.4 Bazo Block Explorer . . . . .	6
2.3 Requirements to the envisioned application . . . . .	6
2.3.1 Functional Requirements . . . . .	7
2.3.2 Quality Requirements . . . . .	7
2.3.3 Analyzing Requirements . . . . .	7

<b>3</b>	<b>Design</b>	<b>9</b>
3.1	Design Guidelines . . . . .	9
3.1.1	Progressive Web Applications . . . . .	9
3.1.2	Network Communication and Interfaces . . . . .	10
3.1.3	Device Sharing . . . . .	10
3.2	Transaction Information . . . . .	11
3.2.1	Schema . . . . .	11
3.2.2	Device Sharing . . . . .	12
3.2.3	Merchant options . . . . .	12
3.2.4	Onboarding Process and Account Generation . . . . .	13
<b>4</b>	<b>Implementation</b>	<b>17</b>
4.1	Architecture . . . . .	17
4.1.1	User Interface . . . . .	18
4.1.2	Storage and State management . . . . .	21
4.1.3	Network Communication . . . . .	31
4.2	Transaction Sharing . . . . .	31
4.2.1	Web APIs and Browser Support . . . . .	31
4.2.2	Web NFC . . . . .	32
4.2.3	NFC Bridge . . . . .	33
4.2.4	Bluetooth Low Energy . . . . .	34
4.2.5	Quick Response Codes . . . . .	34
4.2.6	Fallback Solutions . . . . .	35
4.3	Blockchain Interaction . . . . .	36
4.3.1	Bazo Client Web interface . . . . .	36
4.3.2	Transaction Signing . . . . .	36
4.3.3	Client Integration . . . . .	40



<i>CONTENTS</i>	vii
<b>5 Evaluation</b>	<b>41</b>
5.1 Quantitative Analysis and Optimization . . . . .	41
5.2 Qualitative Evaluation . . . . .	44
5.3 Limitations . . . . .	45
<b>6 Future Work</b>	<b>47</b>
6.1 Web-based Headers-only Client . . . . .	47
6.2 Native Mobile Client . . . . .	47
6.3 Integrating Third-Party services . . . . .	48
<b>7 Summary and Conclusions</b>	<b>49</b>
<b>Bibliography</b>	<b>51</b>
<b>Abbreviations</b>	<b>55</b>
<b>Glossary</b>	<b>57</b>
<b>List of Figures</b>	<b>57</b>
<b>List of Tables</b>	<b>60</b>
<b>A Installation Guidelines</b>	<b>63</b>
A.1 Wallet Application . . . . .	63
A.2 Installing and Running the NFC Bridge . . . . .	64
A.3 Installing and Running a Bazo node on Android . . . . .	65
<b>B Contents of the CD</b>	<b>69</b>



# Chapter 1

## Introduction

A financial service provider based in Zurich developed a bonus program that rewards customers when using their credit and debit cards. With every completed purchase, customers collect virtual points. Several industry partners are participants of this bonus program. The collected virtual points can be exchanged for gift cards from partners who are listed on a website or an app. The way that these services can be used in the traditional system puts the company into a central role, thus requiring administrative efforts to enable both customers and partners to exchange virtual points. In order to decentralize the financial service provider's position, a blockchain-based currency, Bazo (Esperanto: base; foundation), was developed at the University of Zurich. The intention of the currency is to map virtual points to coins in the crypto currency [1]. The focus of this thesis lies on the implementation and evaluation of a Mobile Wallet for said currency. Requirements and implementation details are gathered during the project in an explorative way in partnership with the company.

### 1.1 Motivation

Traditional payment systems require a centralized institution in order to maintain a currency and support operations such as issuing, transferring and determining the state of currency units. Requiring the issuer for each operation that can be performed on the virtual currency further prohibits the direct trading of virtual points between users. In the case of the partner company this resulted in a limited awareness and usage of the incentivization program by customers as well as partners. With the new cryptocurrency developed at the University of Zurich, which is tailored to the requirements of a financial service provider, the company can maintain control over the currency since it remains in an issuer position. In order to let end users such as customers and partners interact with the currency as well as making the program more transparent, further development efforts are taken. With the completion of said efforts, a first complete prototype of a currency ecosystem should be available for evaluation. In doing so, the financial service provider can evaluate the new solution with real costumers and gain insight on how blockchain

technologies can be used to improve payment processes. These development efforts include the development of a client application to let users trade Bazo coins as well as to improve the payment process, since mobile system's capabilities for sharing transaction data can be used.

## 1.2 Description of Work

This thesis covers the design, implementation, and evaluation of a Mobile Wallet for the Bazo cryptocurrency. In order to run the client as trustless as possible, a light client implementation needs to be integrated to the Bazo Wallet. This involves the integration and development of interfaces with other applications in the cryptocurrency to enable operations on the blockchain. The development should leverage existing resources from similar projects where a Mobile Wallet was implemented. It is required to find a way to transmit payment information with native elements such as NFC and Bluetooth. However, a fallback solution needs to be implemented to allow a broad range of users to participate in the program. In order to support the complete use case for customers and partners, it is required to integrate existing web services from the service provider. This is intended to support the complete payment process for proximity-based transfers. The application needs to be tested in a pilot project where the Bazo cryptocurrency is used for payments in a sandboxed environment.

## 1.3 Outline

The main section of the report is structured as follows: In Chapter 2 an introduction to leveraged and related technologies, as well as the traditional and envisioned transaction process is given. The Chapter further contains a documentation of elicited and derived requirements and an analysis thereof. Further, related work, as well as other activities in the project are introduced. Chapter 3 focuses on the design of a Mobile Wallet that satisfies the aforementioned requirements. The concrete implementation of the design is explained in Chapter 4. In Chapter 5 the prototype and the findings from developing the Wallet are evaluated and compared. Chapter 6 summarizes and concludes the report.

# Chapter 2

## Background and Related Work

This chapter aims to provide the reader with the necessary knowledge about related technologies, development activities and the initial situation of the context of this thesis.

### 2.1 Background

This chapter familiarizes the reader with core concepts and technologies, which were leveraged for the development of the Bazo Wallet. An insight into the traditional and envisioned system of the financial service provider is also given, so that the reader is introduced to the company's domain.

#### 2.1.1 Web based Wallets

A web based Wallet is a web application that allows for certain operations on the Blockchain of a currency. There are two common ways of implementing operations in the context of the web. The first one is known as a *Web Client* and allows users to operate on the Blockchain using a web application. The key difference of this approach is that assets, such as private keys, are stored on the server of the application. Since the user has to share these assets with a third party, he will lose control over his funds and is dependent on the provider. This means that funds can be stolen from users, which has happened multiple times in Web clients for the Bitcoin network [26]. The second approach is known as a *Signing-Only Client*. These applications do not maintain the blockchain or parts of it, as it is common with Full- or Headers-only clients. This allows to implement operations with minimal resources such as storage or network usage. Cryptographic operations are performed with the tools available in web applications and implemented in JavaScript. A drawback of such signing-only applications is that there still needs to be a certain amount of trust with the server, that actually maintains parts of the blockchain and exposes an interface for the web applications [23]. Both client variants are compared in Table 2.1.

### 2.1.2 Progressive Web Applications

A Progressive Web Application (PWA), is a web application that has various characteristics that are usually found within native applications. They leverage the accessibility from the web but have various enhancements to give them a user experience that is closer to native mobile applications.

A PWA can be characterized as [24]:

**Progressive.** A PWA should be progressively enhanced, based on what the user agent supports.

**Connectivity Independent.** A Progressive Web Application should always present something to the user. By employing an app-shell architecture, the application can be separated from its content. The application shell as well as cached content should always be shown.

**Installable.** A Progressive Web Application should be installable to the users home screen and accessible from there.

**Secure.** Since web application have access to powerful APIs in the browsers context, PWAs are supposed to be served from secure contexts. This can be achieved by using protocols such as *https* or *wss* [25].

### 2.1.3 Traditional payment process

In the traditional process, customers were incentivized to use debit and credit cards issued by the company by rewarding the customer with points of a virtual currency based on the transaction volumes. Through contracts that are established between the company and partner companies, the company can trade the customers virtual points against gift cards from partner companies. Based on the terms established in the contract, the partner companies are disbursed for the value of the gift cards, which customers can redeem at the partner company. Similar to the process of exchanging virtual points for goods, the company also has to act as a middleman when users want to exchange their points amongst each other. Both these cases pose a significant administrative effort for the company. Further disadvantages are the limited transparency of the program to users and that the virtual points have a relative short life cycle [1].

### 2.1.4 Envisioned payment process

The envisioned system is supposed to overcome the stated disadvantages, by mapping the virtual points to a currency that can be used directly to make transactions between users or between user and the partner company. The company rewarding with the virtual points should still remain in a central position for the currency and control aspects such as issuing new units and accounts. The Wallet that is developed for making transactions with the currency is required to support user-to-user transactions as well as transactions between a user and the merchant. This implies that there is a way that both merchants and

users can request money by exchanging payment information in a device-to-device manner. Since merchants may have more advanced payment systems, a possibility to integrate third party interfaces into the payment process is required. The detailed requirements needed to be explored with the company and are further described in Chapter 2.3.1.

## 2.2 Related Work

The development of a Wallet for the cryptocurrency, which is the focus of this thesis, was started when various parts of the ecosystem of the Bazo currency had already been implemented. This section introduces other applications and their role or influence on the Bazo ecosystem.

### 2.2.1 Bazo

Bazo is a cryptocurrency developed at the University of Zurich. The currency was tailored to the use case of the financial service provider which acts as a central institution that is able to create new coins and accounts. This makes the currency private, since an invitation needs to be used to participate. This differs from various popular cryptocurrencies such as Bitcoin and Ethereum which are both open to the public. Similar to the approach taken in Ethereum, Bazo implements an account-based data model. Consensus in the Bazo network is achieved through a Proof of Work algorithm, although there are drawbacks from employing this strategy [1]. More information about the current state of research concerning the consensus protocol is given in the *Proof of Stake* thesis [7]. The outcome of the first development efforts for Bazo consisted of two applications. The first one, *Bazo Miner*, can be classified as a Mining client. The second application is a Full client [23]. Table 2.1 compares the two applications.

### 2.2.2 Bazo Client Implementations

As part of the initial development efforts for Bazo, a full client application was created. With this application it is possible to issue transactions. However, in order to participate in the Bazo network, peers have to obtain a complete copy of the Blockchain. In order to increase applicability for various use cases and to comply with resource constraints a light client implementation is being developed at the University of Zurich. With this fork implementation of the full client, peers are not required to obtain all information from the blockchain. Validation of the requested information is still ensured by employing new techniques. This opens new use cases; for example, a Bazo client can be run on a device with limited resources such as a mobile device. This type of client can be classified as a *Headers-only Client* [23] and is compared in Table 2.1. The implementation of this client is relevant for this thesis, since the Bazo Wallet is required to communicate with a Bazo client. Through this, the Bazo Wallet can perform actions such as querying account balances, transaction states as well as creating new transactions.

	Blockchain	Block headers	Transaction Signing	Mining
Full clients	yes	yes	yes	no
Headers-only clients	no	yes	yes	no
Signing-only clients	no	no	yes	no
Mining clients	yes	no	no	yes

Table 2.1: Comparison of different client solutions existing in the Bitcoin network [23].

### 2.2.3 Coinblesk

Coinblesk is a project carried out at the Communication Systems Group (CSG) at the University of Zurich. With Coinblesk, payments can be made in the Bitcoin network without having to wait for a transaction to be fully validated in a block. This is achieved by employing a client-server architecture, where transactions are signed by multiple instances in order to ensure validity as well as transaction speed. With Coinblesk, multiple mobile payment solution implementations exist. One implementation is a native Android application and the other one is implemented as a Progressive Web Application. Although the architecture for Coinblesk is substantially different from the approach with Bazo, parts of the user interface can be reused for the Bazo Wallet. This is further described in Chapter 4.1.1.

### 2.2.4 Bazo Block Explorer

As described in Chapter 2.1.4, the financial service provider still remains in a central position in the Bazo network. Thus, administrative tasks such as inviting new users into the network as well as issuing new coins and setting parameters for the miners can be performed. A solution to bring more transparency into the network by letting users inspect and visualize the state of transactions, blocks and other details of a currency lies in providing a Block Explorer. A Block Explorer is an application, that lets users query information about the currency without necessarily requiring them to participate in the network. In order to visualize various aspects of the Bazo currency, a Block Explorer is being developed simultaneously to the implementation of the Bazo Wallet. This application is also used to let the company perform the administrative tasks such as setting the reward for mining a block in the Bazo network and other parameters [41].

## 2.3 Requirements to the envisioned application

This section captures requirements which have been known upfront or which haven been explored or derived from other requirements.



### 2.3.1 Functional Requirements

The envisioned Wallet application is supposed to enable the following operations:

1. Requesting funds from other users. This is supposed to be achieved by sending transaction data between users over multiple ways, such as NFC, BTLE, QR Codes and Links. There should be a fallback solution which is applicable even if all these technologies are not supported by the underlying platform.
2. Sending funds to users.
3. Inspecting account state such as e.g. balance.
4. Linking account details to the Bazo Block Explorer, thus directing the user to it for further details.
5. Requesting new Bazo coins from the traditional bonus points.
6. Querying transaction value of a cash register in an existing POS system.

### 2.3.2 Quality Requirements

The envisioned Wallet application is supposed to comply with the following quality requirements:

- Operations with the currency should be possible in a trustless way using the application.
- All operations requiring the users private key should be safe and run completely in the browser. It should not be necessary to send the key over a network or expose it in any other way.
- Even users on devices and platforms that do not support most ways of transferring transaction data should have a fallback solution available.

### 2.3.3 Analyzing Requirements

The functional requirements outlined in 2.3.1 were compared to the functionality of *myetherwallet.com*, a web-based Wallet for the Ethereum cryptocurrency. All of the requirements are provided by *myetherwallet.com*, except for the ways of transferring transaction data as described in the first requirement and for the integration of a point of sale system. For sharing transaction data among users, *myetherwallet.com* provides the ability to generate a QR-Code from the public address [18].

The envisioned system could be described as a Nonbank-centric person-to-person payment method. This characterization builds on person-to-person exchange of transaction data,

which is clearly met in the described requirements. The clearing of such payments can vary to fit the classification. Several advantages, such as speed and security, of such payment systems are known, although this depends on the way that payments are cleared. Since this type of payment system requires at least the payee to have an account created, drawbacks, such as the payment system not being accepted by users or users fearing that information is revealed, exist [19]. In a study that evaluated demands to payment systems, it is highlighted that users lived a mobile lifestyle and wished for their payment systems to be usable mobile. Security was also found to be an important factor for payment systems, although respondents associated the currently available systems, such as PayPal, to be risky [20]. Since the application will not only be targeted to a universal base of users but also to merchants, which have a special role, the payment solution can be characterized as a *Payment System for Merchants or Retailers*. Another indication of this classification is the usage of proximity technologies at a Point of Sale, which this application requires [21]. There are sources that characterize payment systems specifically for cryptocurrencies. Geva differentiates between Mobile and Cloud Wallets. Functionalities such as proximity-based transfers qualify as a Mobile Wallet and are said to be key enabling technologies for such mobile payments [22]. Another source evaluates different applications that can interact with the Blockchain on a more technical level. The evaluation was performed for the Bitcoin currency, however it is highly comparable since the various forms of clients can also be found in the Bazo ecosystem. Due to the design guideline of crafting the application as a web application, the application can be classified as a *Web-based Signing-Only Client*. This characterization fits, since the browser application can not validate and store parts of the Bazo blockchain. It is necessarily dependent on a web service for communication, however it is important to note that transaction signing is implemented in the browser and not by the web service [23]. It would be technically feasible to retrieve complete blocks from a blockchain and validate transactions in a web browser. With Bazo this is not directly possible, since the blocks are distributed over a TCP connection and validated on a low-level representation [1]. The first of these issues could be solved by developing a web service that would allow retrieval of block headers over a protocol that is suitable for the web.

# Chapter 3

## Design

This Chapter describes the design decisions taken for the implementation of the Bazo Wallet. Section 3.1 introduces design guidelines that were given from the start of the project. Section 3.2 and the following subsections explain how the required functionality of the Wallet was designed.

### 3.1 Design Guidelines

For the development of the Bazo Wallet, multiple requirements were explored with the financial service provider. These are described in 2.3.1. This section introduces several design guidelines which were known from the start of the project and consequences on other design decisions.

#### 3.1.1 Progressive Web Applications

The Bazo Wallet is supposed to be designed as a Progressive Web Application. Due to Progressive Web Applications having native elements, PWAs can be a solution for providing a unified experience for multiple operating systems, targeting both mobile and desktop devices. This implies that a web application is to be crafted, that supports all the functional requirements described in 2.3.1. The Web application therefore needs to contain the following pages to support the core requirements:

- Page to share transaction data
- Page to send new transactions
- Page to inspect account state
- Page to request new funds from bonus points

### 3.1.2 Network Communication and Interfaces

Since the application is designed as a PWA and a quality requirement is the fully client side approach for all the operations, all operations need to be made in the browser. Since no backend application should be leveraged, the Web app needs to be able to sign transactions in the browser. All further communication with the Bazo network needs to be done over web interfaces. This led to the design of a RESTful web interface for the Bazo light client, which was then implemented. The following operations should be supported by the API:

- Querying account state

This endpoint should return all necessary information about the account's state such as balance, the transaction counter and information if the account has root access.

- Preparing transactions

By supplying fee, transaction value, target and source address to this endpoint, the API will prepare the transaction hash and return it to the client to calculate the signature.

- Distributing transactions in the peer-to-peer network

This endpoint can be used to post a transaction hash and signature. The API will then distribute the transaction in the peer-to-peer network.

In order to depict the complete payment process for users in the role of a merchant, the application needs to retrieve information such as transaction value through a web interface. Further, an interface to the existing bonus program was designed in partnership with the company. The following operations were designed to be implemented into the backend of the financial service provider:

- Filing a new request to deduct bonus points and credit a Bazo account
- Querying the state of filed requests
- Querying the transaction value of a POS instance given a supplied ID denoting the POS system in proximity.

### 3.1.3 Device Sharing

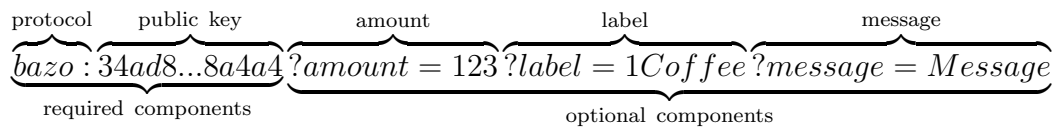
In order to evaluate how suitable a PWA approach for the implementation of a Wallet or payment application is, the application is required to share transaction data on a device-to-device basis. Technologies such as NFC and Bluetooth should be assessed for their browser support and implemented. The data model and mechanism designed to use the described technologies are explained in the consecutive sections.

## 3.2 Transaction Information

The requirement that Bazo coins can be exchanged between users without direct involvement or administration from the service provider was incorporated into the core design of the Bazo currency. To fulfill said requirement it is vital that transaction information can be exchanged between users of the Bazo Wallet. This section outlines the design of the data model and mechanisms that were used to share said information between web applications.

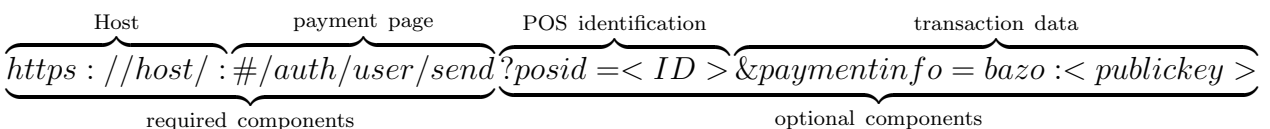
### 3.2.1 Schema

In order to communicate Bazo Transactions to other devices and applications, an URI Scheme was designed to hold transaction information, so that a requesting party can communicate it. The grammar and structure for the schema closely follows the Bitcoin Improvement Proposal 21 (BIP 21) for an URI schema since it contained all necessary elements to fulfill the requirement [6]. The URI schema consists of a required part which holds the protocol name and the recipient's fully qualified Bazo address. The structure and schema of an example transaction request are visible in 3.2.1.



Example of a transaction request with the Bazo Wallet.

The chosen data model implies that the Bazo Wallet and other applications interacting with it need to be able to encode and decode transaction information as supplied in the described grammar. This enables to encode complete transaction information as well as a an URI to the application page into a medium such as a QR-Code. With this, a convenient way of transferring transaction data is given. Since merchants frequently have more advanced payment systems and don't want to encode payment information into media such as QR-Codes for each new transaction, the payment page of the Wallet also needs to be able to accept additional parameters, by which the Wallet can then query the transaction value from a service. Figure 3.2.1 shows an URI consisting of the endpoint of the payment page, together with additional parameters that can be supplied to have the payment page filled with transaction information upon opening it. This procedure is further described in Figure 3.1.2.



Example of a complete URI pointing to the payment page and containing a valid transaction request.

### 3.2.2 Device Sharing

In order to explore further possibilities on how to extend the browser support for native APIs a Proof of Concept was designed. The PoC is targeted to the Android platform, since NFC support is still fairly limited on iOS devices at the time of the design. This means that with Core NFC, a technology by Apple, only communication with passive NFC Tags is supported [4]. The PoC, further referenced as *NFC Bridge*, should run as a background service and enable two processes.

1. Reading NFC devices: The application should be registered as a handler for NFC messages. If an NFC message is read from another NFC device such as another active device or an NFC Tag, the user is prompted with applications that can handle the data format. If the user selects the NFC Bridge application, the data received from the NFC Adapter should be parsed and the user prompted with the Payment page of the Bazo Wallet. The payment page is then prefilled with the payment information extracted from the NDEF message.
2. Pushing to NFC devices: The NFC Bridge application should allow the web-based Bazo Wallet to handover encoded transaction information which can then be pushed to NFC devices. The first design sketch included that communication between the Bridge application and the Bazo Wallet is done through an HTTP or Websocket. This implied that the NFC Bridge would need to run as a background service, listening for incoming push requests. Since the Android platform allows starting an application when a certain type of data is received through the configuration of application intents, this design was preferred. That way, it is also possible to start the Android application from the browser and pass the transaction data to it. The application does not need to run when doing so, and the effort to maintain a server in the application is omitted.

### 3.2.3 Merchant options

In order to completely support the payment process, which does require interaction of the application with backend services from the financial service provider, interfaces need to be integrated into the application. This is also necessary since most platforms do not support WebNFC in the browser, which means that the payment process can not be carried out in a complete device-to-device manner. On iOS devices, support for NFC is restricted to native APIs and the functionality is limited to reading passive NFC Tags. In order to enable the complete payment process, the mechanism that is being implemented can be described as follows. The user of an iOS device can use any existing NFC Tag Reader application from the Apple App Store. Before the payment process can be started, the merchant needs to supply payment information in form of a correct Bazo address as well as some token that represents the Point-of-sale system. The merchant should have the possibility to write these informations to the NFC Tag at the POS, using the Mobile Wallet application on Android. Once this set up is in place, the user will approach the POS System and may start the payment process. The cashier will scan the items, and

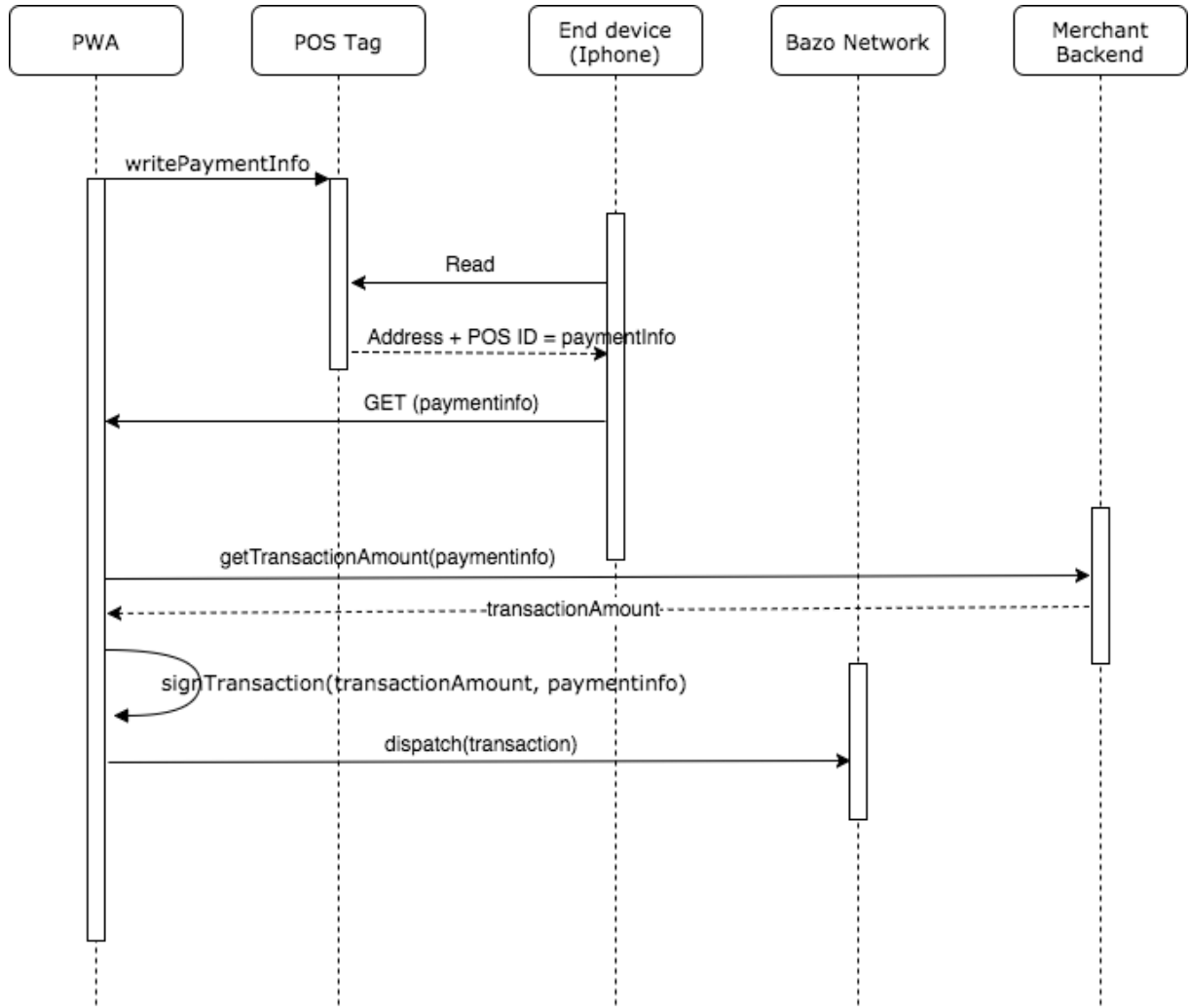


Figure 3.1: Payment process using an NFC tag with static transaction data.

the POS System will automatically associate the POS ID with the transaction value. The user can now use any existing iOS or Android application to read the NFC tag. Since the transaction information is encoded in a URI, the user is taken to the payment page of the Mobile Wallet. Here, the POS ID is used to query the transaction value associated with the POS from a web-service. Since the Bazo Address is supplied in the URI as well, the transaction request is complete and can be confirmed by the user. Once confirmed, the Mobile Wallet will sign the transaction and send it to the Bazo network. Figure 3.1 shows the process of transferring transaction information and transaction issuing under the constraint that a third party service needs to be queried.

### 3.2.4 Onboarding Process and Account Generation

This chapter describes the design of the process of creating a new account and how an existing account can be topped up with Bazo coins. Initially, a user would have to use the Bazo Wallet to generate a new ECDSA key pair consisting of private and public key. This key pair is not considered as valid account, where an inbound transaction

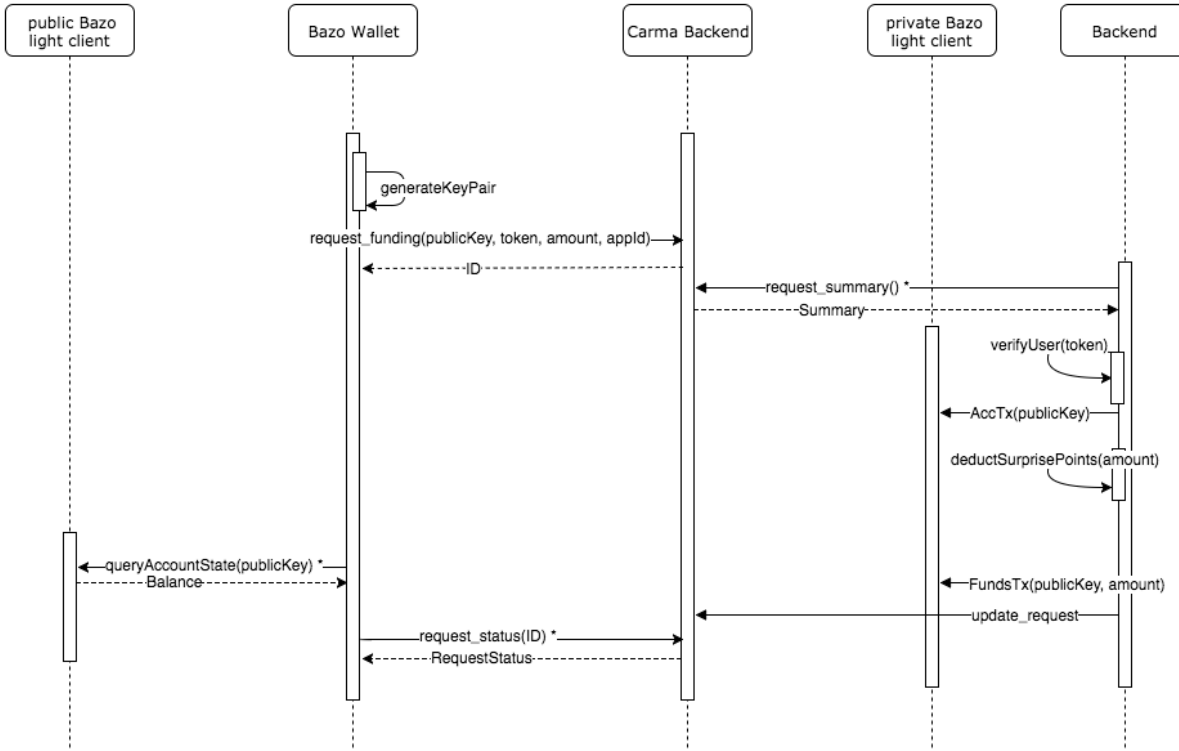


Figure 3.2: Process of generating and registering a new account.

could be executed on the account. This approach differs from other cryptocurrencies such as Ethereum [18]. The reasoning behind this difference lies in the design of the Bazo currency, which is intentionally private. This means, that a key pair is only then a valid account if an *AccTx* is executed. The registration of a public key therefore requires a transaction containing the public key of the new and root account. This transaction would then be signed with the private Key of the root account. Assigning new funds to a registered account requires that the financial service provider deducts the bonus points from the user and sends a *FundsTx* to the users public key. In order to determine if a user may participate in the program and to deduct the bonus points from the correct account, the user needs to manually obtain a token, which is then sent with each request. These operations would be performed in the backend of the financial service provider. The mechanism is further illustrated in Figures 3.2 and 3.3. Operations marked with an asterisk are scheduled to run periodically.



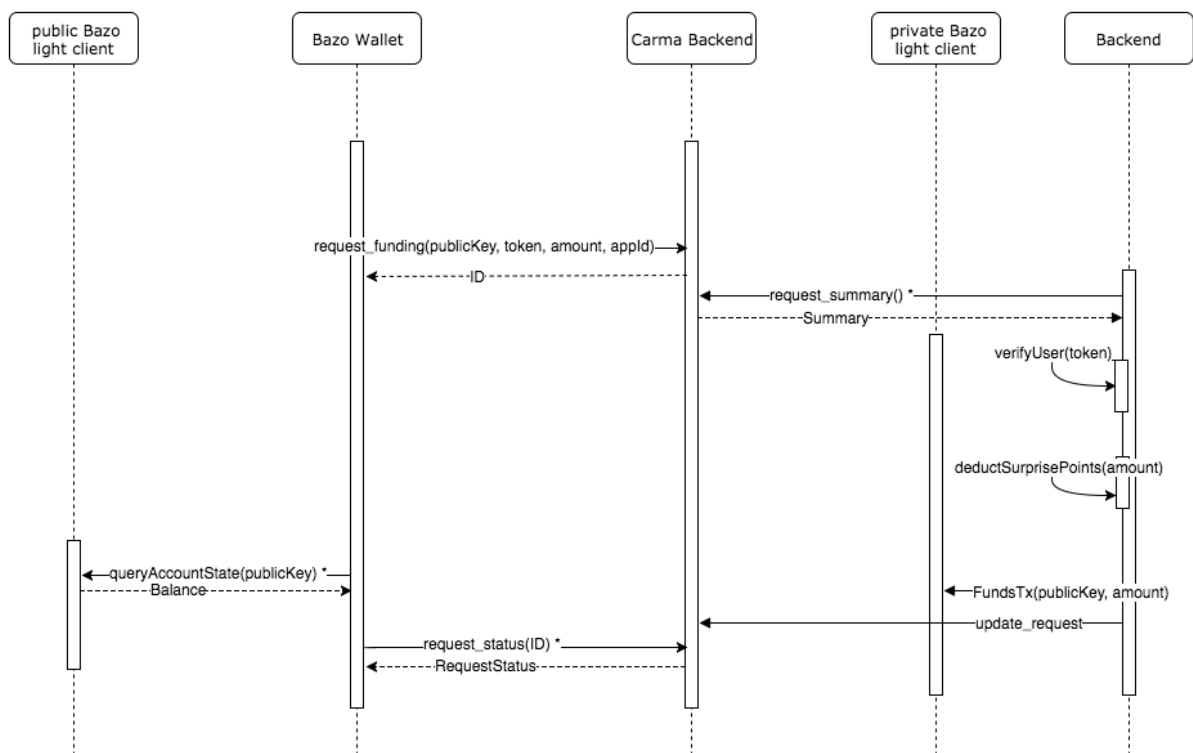


Figure 3.3: Process of requesting new Funds for an existing account.



# Chapter 4

## Implementation

Chapter 3 introduced the design of the Bazo Wallet in order to fulfill the requirements to such an application documented in Chapter 2.3. This Chapter documents the implementation phase of the application. Chapter 4.1 outlines the architecture, the application employed. The following sections explain how the individual functional requirements were implemented.

### 4.1 Architecture

The application was implemented as a Progressive Web Application. The characteristics of such an application have been outlined in Chapter 2.1.2. To comply as a PWA, a web application with the following features was developed:

- **Manifest** A manifest is a JSON file that describes metadata of a web application such as links to icons for different screen sizes. It is further used to configure the progressive web application. For example, the name that will be displayed in the app drawer on the phone once the app is installed, can be set here. It is important to note that only Google Chrome on Android supports manifest files. [12] Other platforms, such as iOS rely on meta tags in the main html file of the web application [13]. Implementing meta tags and manifest file was straight forward, the names and description of the application had to be given, and the static resources such as icons had to be linked correctly.
- **Service Worker and offline capability** A service worker is a method to enable background processing of certain actions. A common way how service workers are used is enabling offline capability by intercepting web requests. By caching static parts of the web application, offline availability is enabled [11]. The service worker used in the project is automatically generated at compile time by a webpack plugin. This makes it possible to flexibly configure the service worker and define all the static resources which are then cached for offline usage. The offline experience is further enhanced by an event listener, which is integrated in the routing of the SPA.

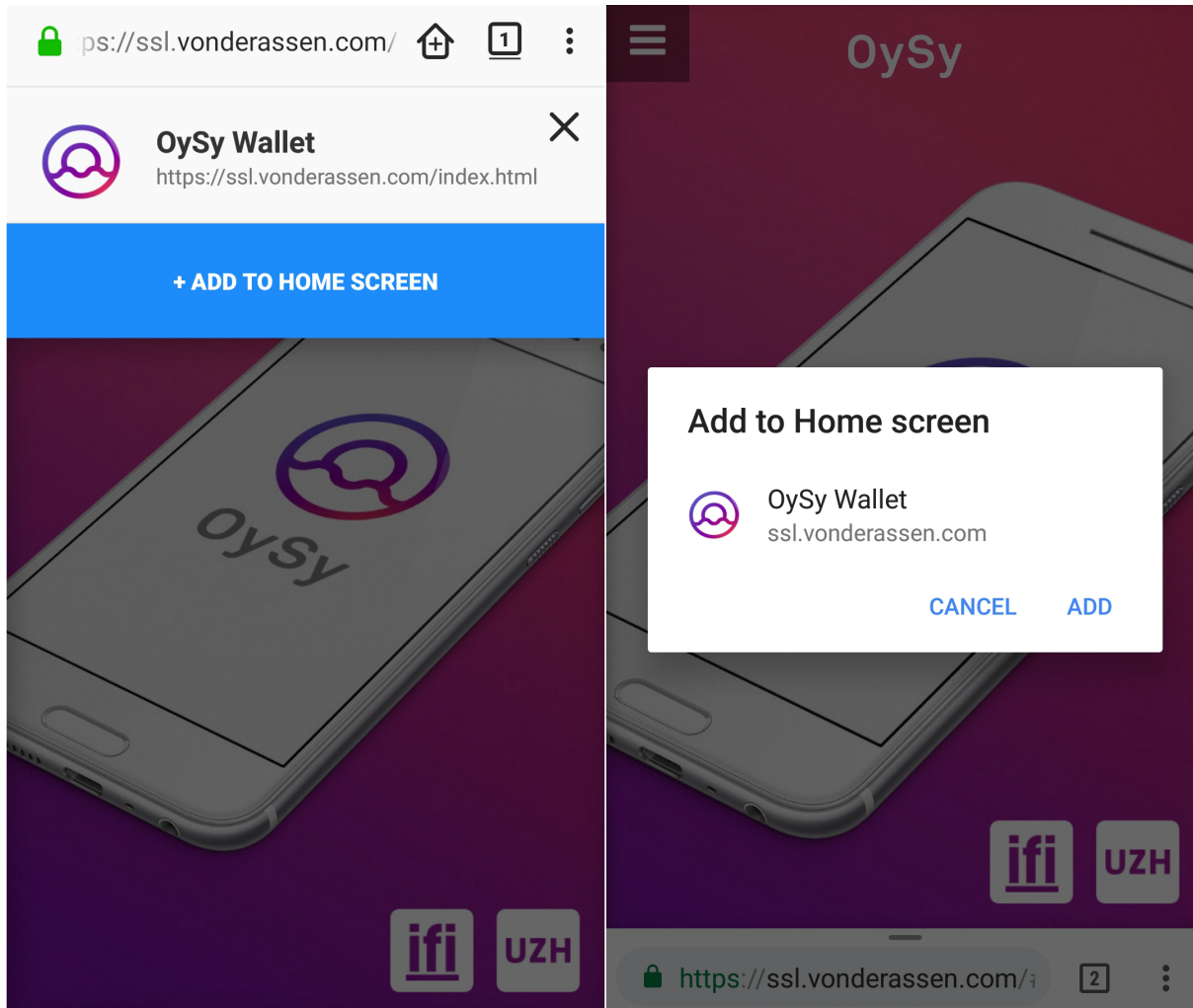


Figure 4.1: Comparison of the PWA installation prompt in Firefox (left) and Google Chrome (right) on an Android device.

If a user is on a page that requires network access, the application will then inform the user about the network loss and redirect to the Home page.

- **Security Constraints** User-agents such as Google Chrome pose certain security constraints to allow a web application to be installable. This restricts web application and all cross-origin requests to be in a secure context, such as secured http or WebSockets [12] [25].

When all characteristics for a PWA are implemented correctly, browsers such as Google Chrome and Firefox will display an installation prompt. This is visualized for both browsers in Figure 4.1.

#### 4.1.1 User Interface

The application was implemented as a Single-Page-Application. With SPAs, it is possible to structure content and application interface into pages without leaving the document.

This deals with the quality requirement, that the application should run in the browser with as little logic on the backend as possible. The Wallet was structured into the following pages, each accessible through a unique URI.

- **Home Page** This is where the user will be directed upon loading the application. If a user tries to access a page, he is not allowed to, he will be directed to this page. For example, if a user opens the link to the payment page of the application, but the network connection is lost, he will be temporarily redirected to the home page until network access becomes available again. An example of the Home Page with a stored account is shown in Figure 4.2.
- **Settings Page** On this page, the user can edit configurations such as showing or hiding advanced options on the other pages, which are directed at users in the role of a merchant. This would mean that the user has an additional field available to encode the POS ID into the transaction data. Further, the user is given the possibility to set the URI of a Bazo client web interface, which is then used for further communication with the client's web interface. The implementation of these options is shown in Figure 4.11.
- **Accounts Page** Whenever a user tries to access any of the other pages except for the Home and Settings Page, without an account being stored in the browser, he will be redirected to the accounts page. Here, it is possible to store new Bazo accounts, by supplying the public key and a name to identify the account. Every public key can only be stored once, so that the total balance can be computed from the set of accounts. All accounts stored in the browser are displayed in a table, along with information such as balance, name and address for every account. Each of the stored accounts can be inspected in the Bazo Block Explorer, see 2.2.4 and deleted from the storage. An account can also be selected to be the main account used. This account will then automatically get preselected for all other operations on other pages. This page can also be opened with an address supplied in the URI, which would then be filled into the form. This is employed to make the process of importing an existing account easier. Figure 4.5 gives an insight into how this Page is being displayed when an unconfirmed and a confirmed account are stored. Figure 4.6 shows how an account can be imported from a keyfile.
- **Requesting Funds Page** When a user wants to share transaction data with another user, he would use this page. Here the user can set the necessary information such as target address and transaction value. If the user has selected advanced options in the Settings Page, he may set an identifier for a POS system. The compiled transaction data can then be transferred to other users using as many of the transaction sharing methods, as supported by his device and browser. Only the sharing methods supported by the underlying platform are visible to the user, in order to progressively enhance the application. For each of the transfer methods, an overlay would be opened to instruct the transfer process and to indicate the state of the transfer. Figure 4.7 demonstrates an example of how payment information can be compiled. The transmission of this information via QR-Code can be seen in Figure 4.8.

- **Sending Funds Page** If a user opens transaction data which is encoded into a single URI, he would be taken to the Sending Funds Page. All transaction data supplied as parameters in the URI are then parsed and filled into the form. If a POS ID is present, see chapter 3.2, the transaction value is looked up. All of this information can also be manually supplied by filling out the form. In either case, the user needs to select an account from which he wants to deduct the transaction, given that multiple accounts are stored. For said source account, the maximal amount of coins to be spent is displayed on the page. If the transaction data is valid, the user can request the transaction to be initiated. Once the transaction is prepared, a modal is opened, where the user needs to enter his private key, which is used to sign the transaction. The process of preparing, signing and submitting a transaction is further described in 4.3. Figure 4.9 visualizes the process of entering payment information. The confirmation of the resulting payment is shown in Figure 4.10.
- **Obtaining new funds** This page allows users to inspect and submit new requests to deduct value from the bonus program and to top up a supplied Bazo account. The user interface consists of table, where previous fund requests are displayed along with their state, and a form to submit new requests. The user has the possibility to generate a new account to select as a recipient for the coins. In order to make a new request, the user needs to supply their token, target Bazo address and the amount of coins he wants to receive. If the user generates a new key pair, he is responsible for backing up the private key. This is promoted by structuring the form in a way that it is recognized by Password Managers. Further, the user has the possibility to download a keyfile. This keyfile contains the key pair, as well as an URL, which allows the user to import the public key into the Wallet with a single click. The process of inspecting an issuing a new funding request is demonstrated in 4.3. The user interface for the generation of a new account is displayed in Figure 4.4.

The described pages are always a standalone display, where it is not possible for a user to be on multiple pages. However, there are parts of the user interface that are shown on all of the pages. These parts along with the individual pages were logically structured into reusable components:

- **Navigation Bar** The navigation bar, located at the top of the screen holds multiple items. The icon of the project, a display of the total balance over all accounts and the currently selected main account are displayed. On devices which are less than 550 pixels wide, the total balance and main account are hidden.
- **Side Bar** The side bar is located at the left side of the document and can be used to navigate between the individual pages. If there is no account configured in the application, only the Settings and Account Pages are shown. If there is no network access, navigating between pages is restricted to the Home, Settings, Accounts and Requesting Funds Pages. The sidebar also contains a display of the total amount of balances over all accounts and the icon of the project. Users can select between German and English languages at the bottom of the Sidebar. Each text in the application is fully internationalized in these two languages. This part of the user interface can be seen at the top of Figure 4.2.

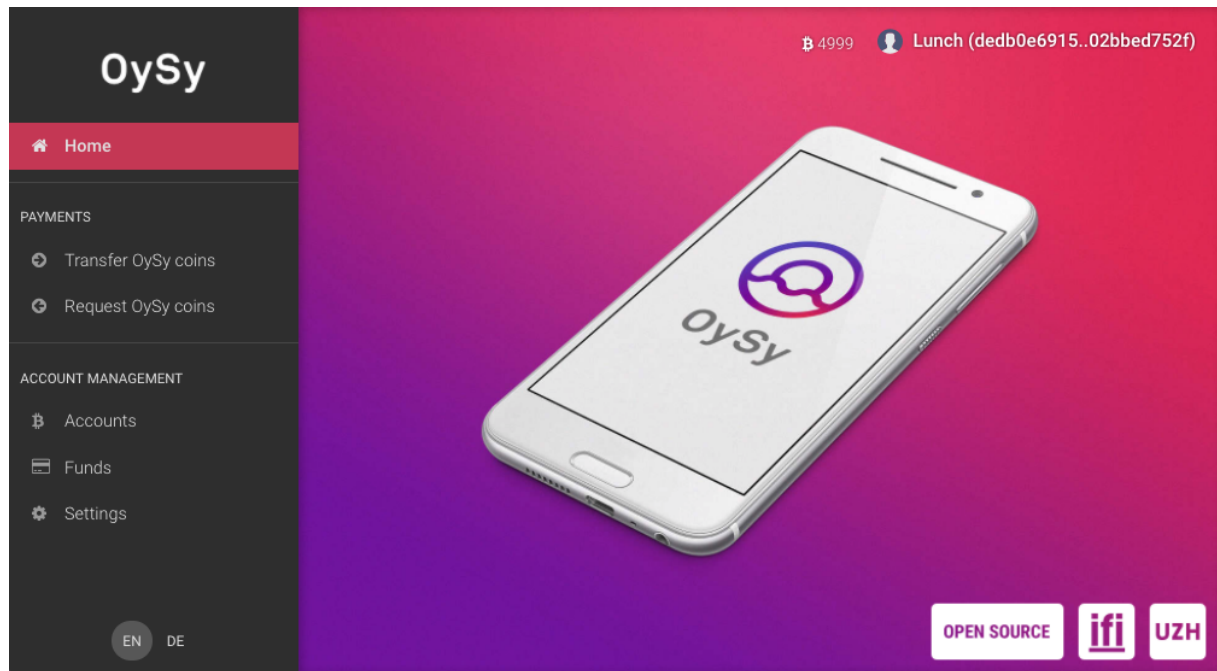



Figure 4.2: Home Page with Side Bar and Navigation Bar on a Desktop Device.

- **Offline status display** There is a background service, which is constantly checking if the application has network access. If this is not the case, the usage of the application is restricted to the Home, Settings, Accounts and Requesting Funds pages. If a user happens to be on any other page during connection loss, he is taken to the Home page where the Offline status display is shown. If the user does not navigate further he will be taken back to the original page, once network connection is obtained again.
- **Progress Bar** On each of the pages there is a thin progress bar at the very top of the document, above the header. This progress bar is used to indicate the progress of network requests made after the application is loaded.

Since the user interface of the coinblesk-frontend application was structured as a SPA with reusable components, it was possible to use this application as a base [40]. Since that application was designed as a client-server application, most of the logic had to be rewritten. However, it was possible to inherit the application structure as well as certain components. The Home Page, as well as the four components that are visible on all pages were reused. Few modifications such as styling had to be applied. Almost all individual pages had to be rewritten. When doing this, the same user interface framework was used, since it provided reusable components and a consistent look with other parts of the application. A webpack configuration was used to optimize and build the application.

#### 4.1.2 Storage and State management


In Chapter 2.3.2, the requirement for complete client-side operation and security requirements were introduced. This implied, that a storage solution local to the browser had to

 **OySy**

## Funds

Transfer your Surprise points to OySy coins.

Surprise Token	Volume	↕
token123	100	


 Reload

---

Surprise Token

Amount

Target account



Request OySy Coins

Figure 4.3: **Funds Page:** Inspect and create new funding requests from bonus points.



Your new OySy account

A new OySy account was generated.  
Submitting this form will add the account to the Wallet. **Make sure to store the private key in a secure place!** In order for the account to be valid, you need to request new OySy coins.

Name this account

miscellaneous

Store the following information securely

Public Key (Address) ⓘ

fc20d1eb78d4c2edc35ea86a06e33b7f808585ft

Private Key ⓘ

.....

Key file

cancel

Add to Wallet

Figure 4.4: **Funds Page:** Generate, save and export a new Bazo account for new funding requests.

≡

OySy

## Accounts

TOTAL ₮ 4999

You currently have 4999 OySy coins.

**Name**

**OyS**

miscellaneous	fc2
Lunch	ded

15 January 2018, 22:19:02

↻ Reload

OySy Address

Name

☐ Mark this account as your main account?

✓ All balances were updated.

This account is **not** yet registered in the Bazo network. You can register the account by requesting new OySy coins on the *Funds* page.

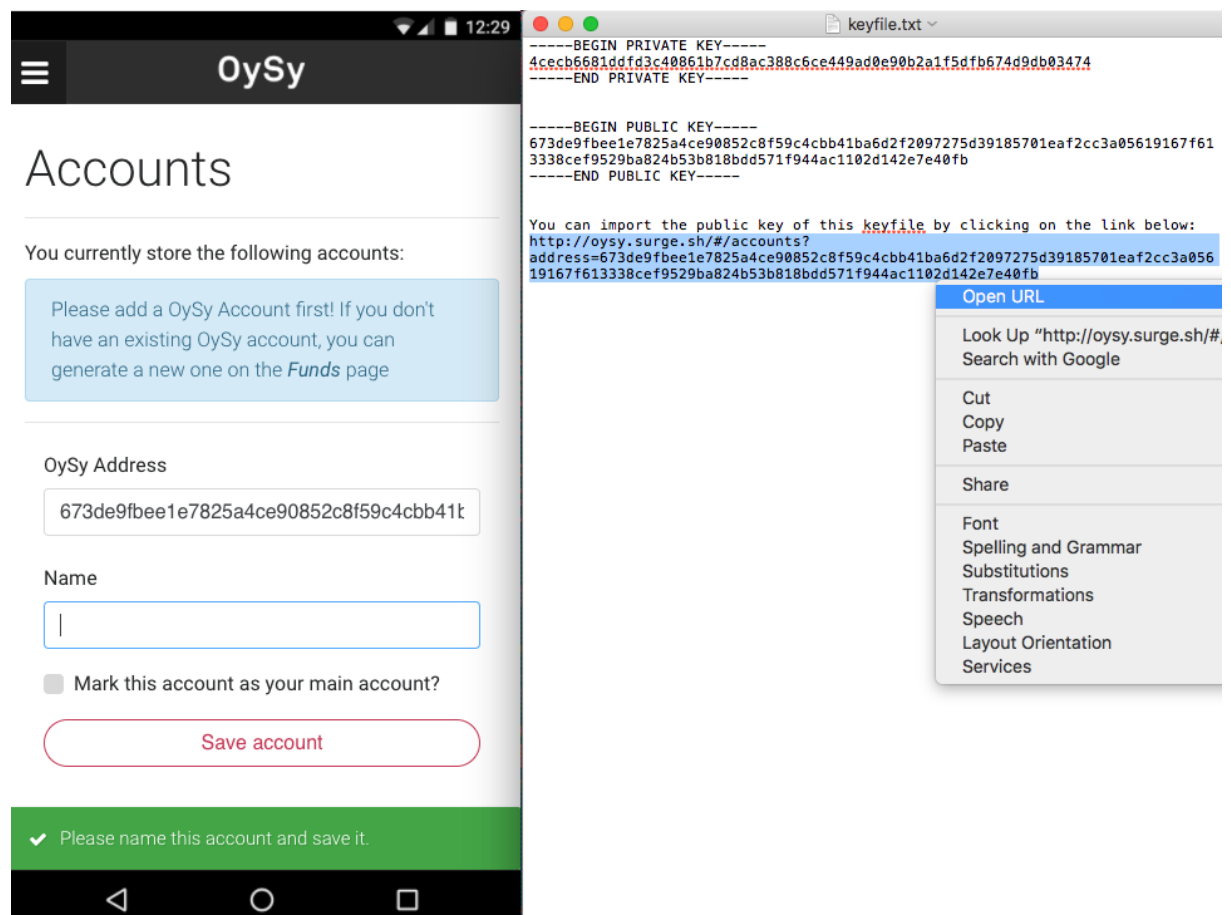


Figure 4.6: **Accounts Page:** Importing an account from a keyfile that was generated on the *Funds* page.

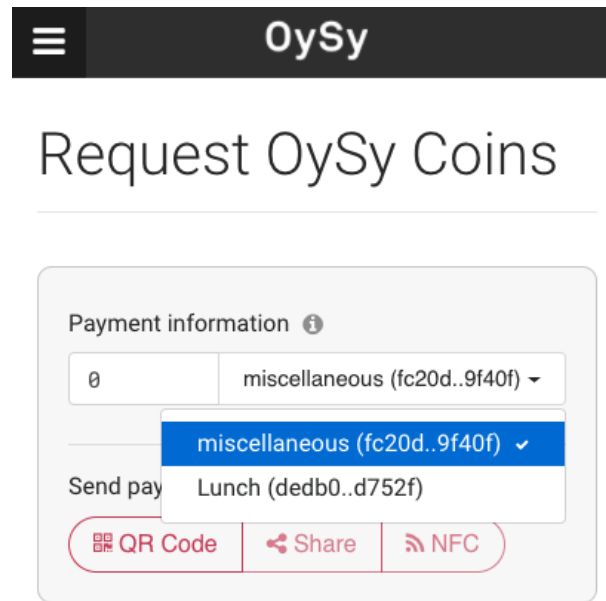



Figure 4.7: **Request Page:** Compile shareable payment information.



Figure 4.8: **Request Page:** Transfer payment information with one of the transfer methods.

 **OySy**

## Transfer OySy coins

Receiver (OySy address) ⓘ  
fc20d1eb78d4c2edc35ea86a06e33 ⓘ 📷

Amount  
10

Maximal amount ⓘ  
4999 Bazo

OySy Account ⓘ  
Lunch (dedb0..d752f) ▾

☒ Fees are included in the amount ⓘ

Send 10 OySy coins

✓ All balances were updated.

Figure 4.9: **Send Page:** Fill in payment information or obtain it through one of the transfer methods available.

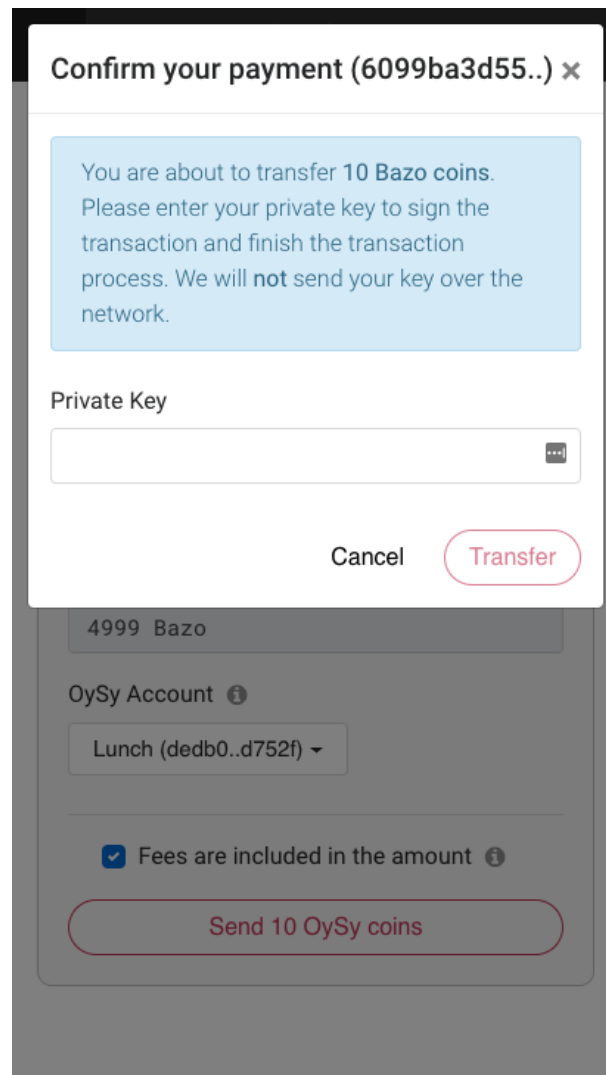


Figure 4.10: **Send Page:** Confirm and sign the payment.

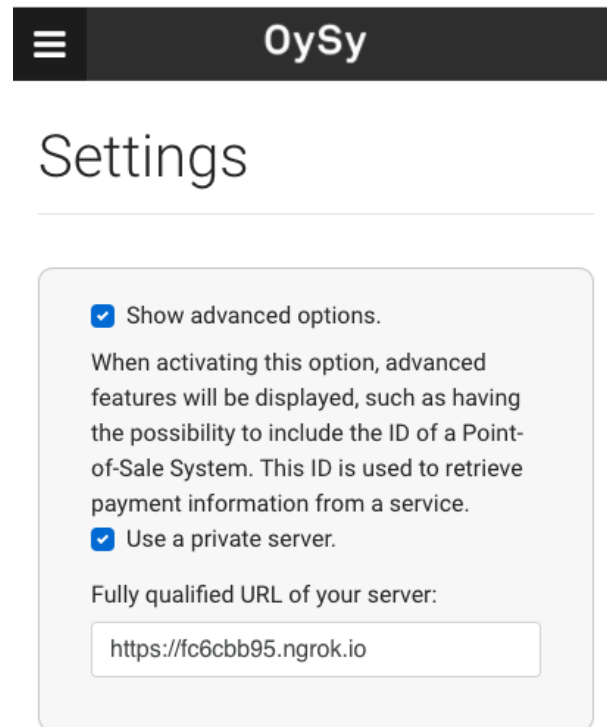


Figure 4.11: Settings Page in the Bazo Wallet.



be used. HTML5 Web Storage is a browser side storage solution, that is supported by the majority of browsers [14] [15].

In the Security part of the specification document, multiple issues with the security of Web Storage are explained [15]. With a new API for browser side encryption in discussion by the World Wide Web Consortium, mechanisms like symmetrically encrypting values in the storage could be leveraged. However, the API is only considered a recommendation of the consortium[16]. This makes local storage of security related data such as private keys unsuitable. Therefore, only user-data such as settings and public keys of Bazo accounts were stored in the Web Storage. For explicit state management vuex, a library for state management with Vue.js was used [17]. The subset of state that needed to be persisted in Web Storage was placed in this storage.

### 4.1.3 Network Communication

Multiple operations of the Bazo Wallet are performed using cross-origin http requests. These operations include payment preparation, payment submission and account balance querying. All of these operations are centralized in a single Service module, that can be shared between the individual components. This makes parametrizing network requests globally possible. This is a requirement, so that the URI used to perform the requests can be configured by the user. The actual communication with the endpoints of the Bazo client and their semantics are explained in 4.3.

## 4.2 Transaction Sharing

Requesting payments is a core requirement for the Bazo Wallet to qualify as a payment system for end-users. The design of the sharing process and the data model were explained in 3.2. This section explains how sharing transaction data across devices was implemented.

### 4.2.1 Web APIs and Browser Support

At the time of writing, more and more native APIs are being made available for the Web. However, the API specification for NFC connectivity in the browser is still in draft [2]. Partial implementations of the API are available in Google Chrome on Android devices. Other user-agents and platforms do not yet support the feature, although some have expressed intent to implement. The implementation state in Google Chrome is not documented thoroughly since there exists no implementation report yet, as they are available for other specifications [14][2]. This implied that the specification and Google Chrome's source code were the only sources to obtain information on the subject apart from testing. For the prototype, the feature was implemented for Google Chrome on Android. In order to allow web-nfc in Google Chrome, there need to be two flags set to enable the experimental implementation: *Experimental Web Platform features* and *webnfc*. The

first of these flags will show the `nfc` object in the navigator object of the browser. However, this does not mean that the hardware does support web-nfc. The latter flag can only be set on devices whose hardware actually supports NFC. This approach of hiding functionalities behind flags to let developers test the implementations, is called origin trials, in the Chromium Projects [28]. After extensive testing it became clear that the current implementation of Google Chrome only allows devices to use NFC functionalities such as reading and writing to NFC tags as a target. Performing these actions to active NFC devices such as a smartphone will throw the corresponding error according to the specification draft [2]. After discussion with members on the working group, they confirmed that targeting active NFC devices is not yet implemented. The reason being that these operations would require native UI elements from the Android operating system. However, they expressed intent on implementing it in the future [27].

Another challenge posed the creation and signing of Bazo transactions in the browser. The structure of a Funds Transaction can be seen in figure 4.2.1. Since not all data-types that are usually available in C-like structs are available in the browser, a bridging solution had to be implemented. The application was therefore designed as a Signing-Only Client, with the implication of designing a web interface into the light client as described in 3.1.2. This is a common approach when building web based Wallets that don't store the users private key on a server [23][26].

---

**Algorithm 1** FundsTx

---

```

1: type FundsTx struct {
2:   Header byte
3:   Amount uint64
4:   Fee uint64
5:   TxCnt uint32
6:   From [32] byte
7:   To [32] byte
8:   Sig [64] byte
9: };

```

---

Structure of a FundsTx in Bazo [1].

## 4.2.2 Web NFC

In section 3.2.2 the semantics of the data model of a transaction that can be shared among users in the form of a single URI was explained.

NFC also enables devices to share information at a distance of less than 10 centimeters. Users can share business cards, make transactions, access information from a smart poster or provide credentials for access control systems with a simple touch. NFC is a technology standard that allows contactless, two-way data transfers between devices that are within centimeters of distance [29]. Since the support for WebNFC is limited to Android devices, the functionality for writing and reading the transaction data is visible only to the users that have activated advanced options. The corresponding UI elements are therefore

hidden if the user has not activated advanced options. The same goes for browsers and devices that lack webnfc or nfc support. Using the API for the actual reading and writing operations was straightforward since the API consists of one method for each of them. The data type of the NFC record was set to URL. This way, the URL can be read from all types of devices. Android devices can therefore use the Android system or the Bazo Wallet to scan the Tag. If the Android system is used, the tag can be placed to the back of the device. If NFC is activated in the system settings and the user has unlocked the phone, the NFC record is parsed. Since the data type of the NFC record is a single URL, Android's intent system is used to ask the user which application to use for the URL. This should include Google Chrome for all standard installations and the Bazo Wallet itself, if it is installed to the homescreen.

### 4.2.3 NFC Bridge

Due to limited browser support for Native APIs such as webNFC, see chapter 4.2.1, a prototype was discussed and implemented in the scope of this thesis. The prototype involved a native Android application that should enable the web application to forward transaction information to NFC capable devices using the Android Beam technology. The design and requirements of such an application was outlined in 3.2.2 and the two main functional requirements for the application were presented. Reading NFC messages from passive and active devices was achieved without implementing it in the Android application. This was done by changing the data model for transaction information to one that can be expressed in a single URL. With this media type, it is possible to use the tag dispatching system in Android. This would work as follows:

1. An NDEF message is received when the Android device is unlocked and NFC enabled.
2. The NDEF message for this application contains just a single URI. Through Android's intent system the application that fits best is started.
3. If the Bazo Wallet PWA was previously installed on the Android device, an intent filter is present for the origin URI of the PWA. This will result in the PWA being opened with all transaction data passed as parameter.
4. If the Bazo Wallet PWA is not installed, there will be no intent filter requesting the data. The user is then prompted with all applications that can handle an URI, such as a Web browser. If a browser is selected, then the Bazo payment page is opened with all transaction data passed as a parameter.

Writing the transaction data to other Android devices was implemented using the Android Beam technology. The transaction data can be passed to the NFC Bridge by either manually entering it into a form or by passing it as data through an intent. Passing data to the application through parameters has the benefit that web applications can generate links that will take the user to the application. Such a link can contain further data, which would be set to the transaction data the user wants to share with another Android

device. If the Android phone is held against another Android Beam enabled device, the NFC Bridge application will automatically set the appropriate transaction data encoded as an NDEF message and push it to the other device.

At the time of writing, NFC support for iOS platforms was just released for their most recent publicly available version, iOS 11. The NFC functionality offered to iOS developers covered writing and reading to NFC tags on the iPhone 7 and iPhone 7 Plus [4]. Thus, an NFC Bridge application similar to how it was developed for Android would not extend the functionality of existing generic NFC applications. The NFC fallback capabilities with the Bazo Wallet on iOS are outlined in Chapter 4.2.6.

#### 4.2.4 Bluetooth Low Energy

Bluetooth Low Energy allows for devices to exchange data in short wireless connections. Devices will usually establish connections in a peer-to-peer topology [30]. Data can be structured into GATT-services which is then exposed to other BLE-capable devices. These devices may then perform actions such as reading and writing on these Services [31]. This makes the web-bluetooth API useful for web applications interacting with BLE-devices in proximity. However, to use the API for transaction sharing between two web pages on different devices, this would imply for one of the devices to take on the role of a server. This functionality is not provided in the web-bluetooth API, and therefore makes it unsuitable to solve the requirement [32].

Since the web-bluetooth API could not be used for data sharing, the websharing API was leveraged to provide easy sharing capabilities with low friction. Low friction for usage is given through the deep integration of the API with Android's intent system. By invoking the API through passing the relevant URI, all sharing options available to the device will be presented to the user [36]. This has two advantages:

1. Technologies such as Bluetooth, E-mail, SMS and even Android Beam are always available, even on systems without any third-party applications.
2. Sharing transaction data is not limited to functionalities provided by the Wallet. The user has the possibility to use any existing application, for example his favorite chat application, to transfer the URI. When new applications are installed, they are automatically registered and then considered by the Android's intent system for sharing.

The Web Share API is not considered a W3C standard, however it was assessed as an origin trial in Google Chrome on Android and is now normally available starting with version 61 [36][37].

#### 4.2.5 Quick Response Codes

Quick Response Codes were initially envisioned to be the fallback option to transfer transaction data for devices that either do not support NFC or in cases where the browser does

not support webNFC. With the data model explained in 3.2.1, all transaction data can be placed in a QR Code in the form of a single URI. This data model was chosen with the intent to support devices where the browser does not allow camera access in the browser over the webRTC API, which is still not widely adapted by browsers [42]. Testing Google Chrome and Safari on desktop and mobile systems showed that only Google Chrome has implemented the webRTC API for camera access. Safari has implemented webRTC into their most recent version 11, which was released to the public in September 2017 [9][10]. This implies that users with older versions of iOS will not have access to the API. Users with such a device can still read the QR Code using a generic native application. Since all information is encoded in a single URI the payment page can be opened with all necessary information prefilled. Google Chrome has already implemented the webRTC API, therefore multiple QR Reader libraries leveraging the API exist. The QR Scanner from the coinblesk-frontend project was a simple wrapper around an existing QR Reader library [33]. Due to the implementations in Safari version 11, the underlying library had to be modified so that certain attributes were set. This allowed to run the QR Reader in the Browser of iOS devices. The actual decoding of the image with the library is implemented in pure JavaScript. This has performance drawbacks, since there is no hardware support. Further, the library is quite large with around 1.5 MB size for a minified JavaScript file. Because of this, the ShapeDetectionAPI was evaluated, which promises certain optimizations, since the decoding is not run in JavaScript [35]. However, this new API is neither a W3c Standard or on track of becoming one [34].

#### 4.2.6 Fallback Solutions

Subsections of Chapter 4.2 outlined how existing native and web-enabled APIs were leveraged for the Bazo Wallet to share transaction data. This section documents the fallback solutions evaluated for devices or platforms that lack the support for the API, either natively or on the web. For Android, the complete functionality of NFC could be leveraged. Writing and reading NFC tags was implemented with the web-nfc API. Complete peer to peer communication was enabled by developing an API bridge, see 4.2.3. A similar functionality is also provided by the webshareAPI implemented for Bluetooth transferrals 4.2.4. Since iOS platforms only support native APIs when communicating over NFC and peer-to-peer mode has not yet been included, the fallback solution to use NFC on iOS consists of using a native application. Writing the NFC tag can be done with any generic NFC application or, on an Android device, with the Bazo Wallet itself. Initially a QR-Code was envisioned to be the fallback solution that should cover all devices. However, reading a camera video stream is only possible for the most recent version of iOS. This implies that this can not be used as a fallback solution for older iOS devices, although the same procedure as with NFC, where an existing native application can be used for scanning, is possible. Because of this limitation in iOS, a solution where the URI with all transaction data is being copied to the users clipboard was implemented for both systems. This way, iOS users have a convenient way of obtaining the URI and can then manually distribute the URI over their preferred channel. The same functionality is offered to Android users over the webshare API which has a better integration into the Android system, so that the URI is directly passed to Android's intent system for sharing, see 4.2.4.

## 4.3 Blockchain Interaction

Trustless network communication with the Bazo network posed the most difficult technical challenge for the application. The following operations and issues had to be evaluated:

1. Building the transaction data in the schema defined in the Bazo protocol, as described in chapter 4.2.1. Due to technical limitations of the browser, this application context is not suitable to fulfill the requirements of a light client implementation. Reasons for this are lack of support for tcp communication, appropriate storage solutions and data structures.
2. Signing the transaction data with the appropriate algorithms.
3. Verifying transaction and account information. In order to run the Bazo Wallet as trustless as possible, the application has to maintain at least certain parts of the blockchain to verify information. For the Bazo Wallet this would mean that the Bazo Wallet itself needs to verify blocks and transactions, or that the Bazo Wallet needs to communicate with a trustworthy Bazo client.

### 4.3.1 Bazo Client Web interface

In order to interact with the Bazo client, a web interface had to be created. The web service was designed as a RESTful API and implemented into the light client. The web service consisted of two endpoints for each type of transaction in the Bazo currency and one for the retrieval of account information such as balance and other parameters. For each of the transaction types the following two endpoints need to be used.

- The endpoint at `/create<Transaction Type>/` is used to create and obtain a transaction hash given all parameters for the transaction struct are sent. This solves the problem of lacking the appropriate data types in the browser. The returned transaction hash can then be signed on the client.
- The endpoint at `/send<Transaction Type>/` accepts the transaction signature, which was generated on the client, and matching transaction hash. With these informations, the transaction can be considered complete, so that the underlying Bazo client can distribute the transaction for further validation in the network. The web service does not validate the transaction and respond to the client.

### 4.3.2 Transaction Signing

Due to limitations of the browser for C-like structures, required to create a FundsTx, an interface to the Bazo client was developed. The process of obtaining transaction information, requesting a new transaction, signing and dispatching it into the network is shown in figure 4.12. The process can be described as follows:

1. The application queries the necessary account information to request a transaction struct in a later step. This is a necessary step, since the TxCount of the transaction has to be correct in order for the transaction to be valid.
2. When the account information is read, the web application sends a request to the web service to create a new FundsTx struct. The Bazo client creates the appropriate structure, hashes it using the *SHA3-256* algorithm and returns it to the Wallet.
3. The Wallet is now in charge of asking the user about his private key and signing the received transaction hash, using the *ECDSA* algorithm on the *p256* curve. The resulting signature and transaction hash pair then needs to be sent back to the Bazo client for distribution in the network.

This approach is comparable to the mechanism employed in the official JavaScript API of the Ripple currency [8]. Figure 4.13 demonstrates how a transaction can be sent with the Ripple cryptocurrency using the official library. The mechanism can be outlined as follows:

1. A connection to an interface is opened over a WebSocket.
2. A request is made to obtain relevant data and instructions for the payment.
3. The data returned from preparing the transaction is signed.
4. The API submits the signed transaction over the WebSocket.

Comparing the mechanism in Bazo, see Figure 4.12, and Ripple, 4.13, reveals certain differences, although the approach, with the respective consequences being similar.

- **Protocol mechanism** In Ripple, there exists a convenient method to prepare all data into a single JSON object which needs to be signed and submitted. This step is split up into multiple steps in the web interface for Bazo. The client is responsible for querying the account information and then requesting a transaction hash from said information.
- **Connection Type** Both cryptocurrencies use web interfaces for communication with a node in the network. Http is used in Bazo, while the Ripple library makes use of a WebSocket for two-way communication. Given that two-way communication is not necessary for the Bazo currency, the http approach was chosen, since it is supported by a broad range of devices and browsers.

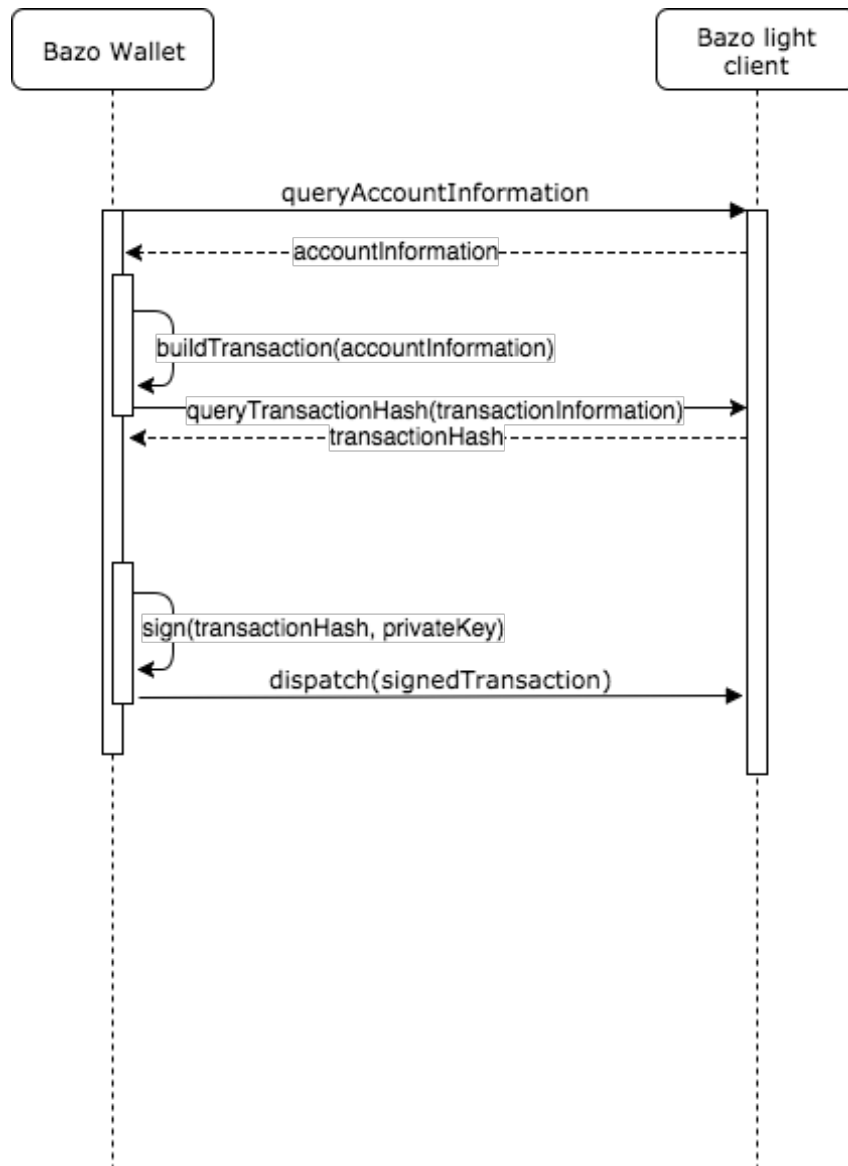


Figure 4.12: Requesting, building, signing and submitting payment information with the Bazo Wallet.



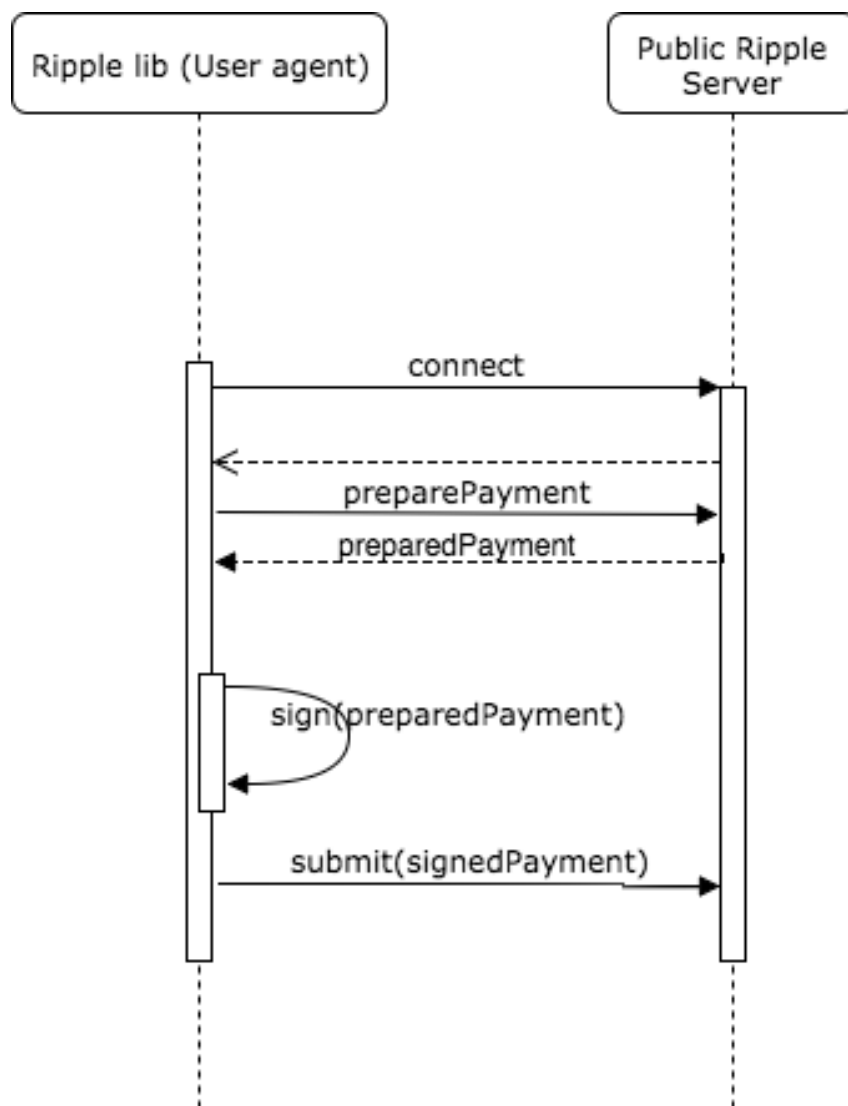


Figure 4.13: Programmatically creating, signing and sending a transaction with Ripple-lib [8].

### 4.3.3 Client Integration

Chapter 4.3.1 explained the interaction between the Bazo Wallet and the Bazo light client over a web interface. This chapter explains contexts in which the Bazo client can be run as well as their consequences. It further documents the efforts of porting the light client to mobile devices. Due to the security constraints one needs to consider when interacting with cross-origin resources in a PWA, the Bazo client web interface has to be in a secure context [25]. This would mean that an SSL certificate needs to be installed or that the server needs to be accessible through a secure reverse tunnel. This makes mobile devices as a target difficult. However, a Proof of Concept was developed to show that mobile devices are technically feasible to run the client.

The first approach for porting the light client, written in the Go language, evaluated the usage of bindings for Java. These could then be used in a native Android application. There exists a project, *go mobile*, that can be used to generate bindings from a package written in Go. However, the current state of the *go mobile* project does not support all networking and data structure requirements. Further, this would have required to restructure the *go* application as a package [5]. Programming to such bindings has the advantage that the Java application can directly call methods of the binded package. Since all communication between the actual Wallet application would go over the *http* interface, this would not pose an advantage.

The second approach focused on cross-compiling the Bazo client for the Android operating system running on ARMv5 architectures. To port the complete Bazo client without any modifications to the Android operating system, the application was compiled using the *clang* compiler. The compiler can be obtained through Android's official NDK or it can be compiled from scratch. The complete toolchain was compiled targeting the ARM architecture, and Android API version 21 for a darwin system which was the host for the compilation [38]. The *clang* compiler of the toolchain was then used to configure the *go* environment for cross-compilation. This involved configuring the *golang* environment for ARM architectures and to use the freshly compiled *clang* compiler [39]. The binary was then copied to an Android device and run in the scope of an existing application. This is a necessary step for binaries that are run on devices where root access is not given. The application performed as expected, so that *ngrok* was used to create a secure reverse tunnel. With this in place, the web application was able to interact with the Bazo client running on the same device but accessible through the secure context. Detailed instructions for installation can be found in A.3. This Proof of Concept could be leveraged to create an actual Android application wrapping the binary. However, a solution to configure SSL appropriately would need to be found.

# Chapter 5

## Evaluation

The ecosystem of the Bazo cryptocurrency can not be considered completed to a degree where a test run would be possible, at the time of writing. Reasons for this are that the development of the interfaces of the existing infrastructure is not complete. Since these interfaces were specifically designed for the Bazo Wallet during the course of this thesis, the Wallet itself can be considered complete. This chapter assesses the developed Wallet and explains limitations to the system. The assessment of the application is structured into two sections, where the first one focuses on performance aspects and the latter on qualitative metrics.

### 5.1 Quantitative Analysis and Optimization

In order to assess and optimize the quality of the developed web application in a quantitative manner, Lighthouse was used for the audit.

Lighthouse is an open-source, automated tool for improving the quality of web pages.

There exist various work flows to run the tool, however, running it directly from the Google Chrome Developer Tools is suggested by the authors. The audit can be performed against almost any web page and assesses the following five key factors of web applications: *Progressive Web App*, *Performance*, *Accessibility*, *Best Practices* and *SEO*. For each of the metrics, several audits are performed and then summarized to express the scoring as a percentage. It is possible to drill down and obtain detailed information on a failed audit. For some of the audits, the tool can even predict possible enhancements [43]. Running the tool against the deployed web application showed the strengths and weaknesses of the implementation. Initially, the application received the full score for *Progressive Web App* and *Accessibility*, which performs audits against the Progressive Web Application Checklist [44]. The biggest optimization potential was highlighted through the *Performance* metric, which obtained a result of 44. There were various issues that hindered the performance of the application. These issues and their solution are explained in the following.

- Render-blocking scripts:

The JavaScript library that contained the decoder for QR-Codes makes up a large part of the application's size. Loading the library from the start of the application had a negative impact on the loading time required until the application is drawn and interactive. This is an interesting fact, since the development of the Shape Detection API, introduced in 4.2.5, has the intention of avoiding the necessity to include a decoding library in web applications. This issue was solved by dynamically loading the library, once the user enters the actual page where the decoding capability is needed.

- Large images:

Images that are not properly sized can also increase the loading time, thus affecting when the application becomes interactive. The images, supplied by a Design agency were compressed, resized and converted to a more suitable format to decrease the loading and rendering time. By doing so, the size of the assets was decreased by around 80%.

- Rendering time:

The time required to render the home page was further optimized by making use of a server-side rendering approach. Since the home page contains mostly static content, it was ideal for this technique. Server-side rendering was achieved by incorporating an optimization plugin into the *webpack*-based build process.

The issues described above led to initial loading times of around five to seven seconds until the application becomes responsive. It is important to note that Lighthouse performs these tests under constrained settings. Network speeds are throttled to resemble mobile devices and caching is not leveraged. This testing environment is comparable to a mobile device with a slow network connection accessing the application for the first time. With the performance optimizations described above, the initial loading time was decreased to under two seconds. Subsequent loading times would take around 250ms until the application becomes responsive by leveraging the caching mechanisms. After optimizing the application, the *Performance* metric received a score of 77.

For the full score of the *Best Practises* metric, the application would have to be deployed to a server that supports HTTP/2. Since this is not part of the actual implementation of the web application and rather a question of deployment, this metric can not be improved with the code basis of the web application. Lighthouse assesses all audits from the user's computer. This might impact the validity of the Performance audit. Although this audit is already performed with a constrained network to simulate mobile environments, the audit could suffer from other impacting factors. For this reason, another popular performance testing service, Pingdom, was used to verify the results. Pingdom differs from Lighthouse in such a way that it runs the tests from a server, where an intercontinental grid of servers is used to determine the performance of the application [46]. The Bazo Wallet was tested from all the locations available in the tool and consistently obtained the highest performance grade. Figure 5.1 summarizes a performance audit with Pingdom for one location.

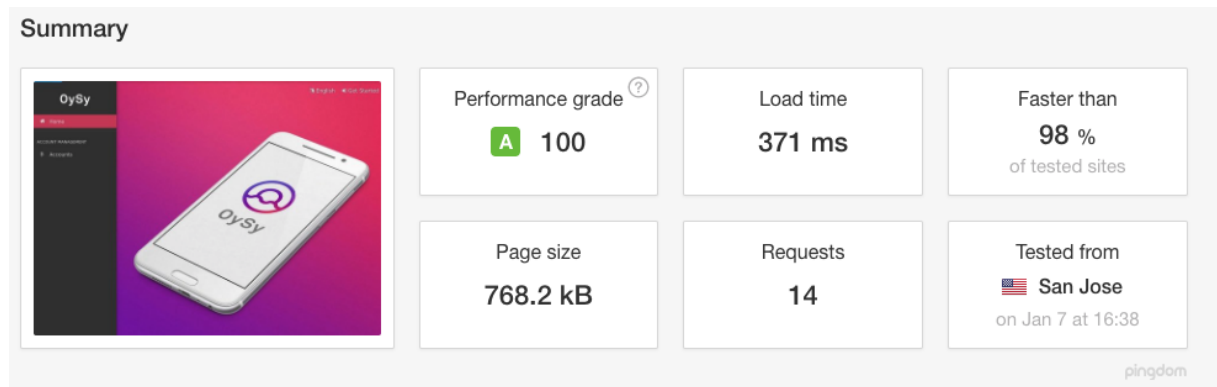


Figure 5.1: Summary of a Performance report using Pingdom.

Lighthouse specifically advises to conduct manual checks to ensure an appropriate user experience on multiple browsers and platforms. The same strategy is proposed for testing User Interfaces, with regard to page transitions and animations. Since the complete functionality with all branches can be tested relatively quickly, the complete app was tested for a variety of devices. The rest of this chapter gives insight into the testing results and the tested combinations of operating systems and browsers.

	Internet Explorer	Google Chrome	Safari	Firefox
macOS X (10.12.6)	no (deprecated)	yes	yes	yes
Windows 10 (14393.479)	yes	yes	no (deprecated)	yes
iOS (11.1.2)	no (not available)	yes	yes	yes
Android (8.0.0)	no (not available)	yes	no (not available)	yes

Table 5.1: Overview of tested browsers and operating systems.

Table 5.1 shows targeted platforms where the complete functionality of the Bazo Wallet was tested. The web application proved to work as expected from the evaluation of cross-browser support for the leveraged APIs as described in 4.2.1. Testing the web application in Safari on an iOS device revealed that there are still several issues with PWAs on iOS; The application behaves differently, depending on if it is run directly from Safari or from the home screen. Given that WebKit, the browser engine used by Safari, still does not completely support PWAs explains the lack of stability. Many of the technologies used with PWAs, such as Service Workers, are not completely implemented at the time of writing [45]. However, using the web application in Safari provided a consistent user experience. All elements that require unsupported features were hidden from the user.

All parts of the application that contain interaction with the Bazo network can be considered critical functionality. The same applies to other parts which are important for the security of user's funds. In order to simplify testing and improve reusability, all of these operations were bundled into a JavaScript library. Hence, this library consists of methods that wrap the web interface of the Bazo client, outlined in 4.3.1. The library further contains helper functions for operations, such as generating appropriate key pairs. The Bazo Wallet only needs to be able to create one of the three types of transactions

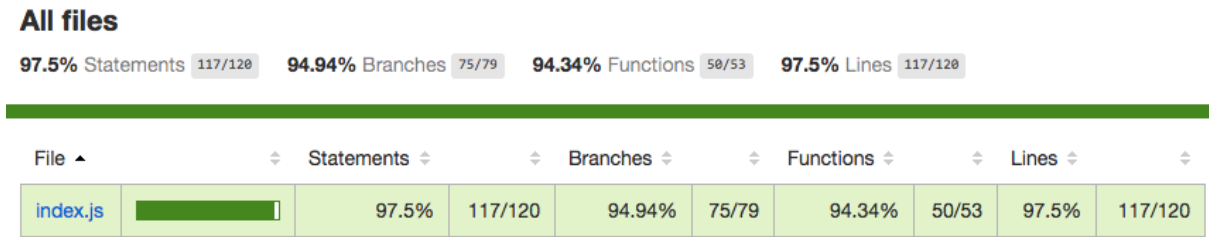


Figure 5.2: Coverage report of the Bazo API library.

that are possible within Bazo. However, to promote the library’s reusability, the scope of the library was extended for all transaction types and their respective helper methods. The library was tested using the *mocha* testing framework in conjunction with *istanbul* for code coverage analysis. 97.5% of all lines were covered by the unit tests, see Figure 5.2. Modularizing all payment related operations into a library further allows assessing the speed of a payment with the Wallet. The helper method of the JavaScript library, which allows to prepare, sign and distribute a transaction, was tested under constrained network access to simulate a mobile environment. Running the tests iteratively revealed a consistent performance for the operation, which took between 1500 and 2000 milliseconds. The duration of the operation can be explained by looking at the implementation, which consists of three synchronous http requests and the computation of the transaction’s signature. Each request had an impact of around 500 to 600 milliseconds to the operation. The unit tests of individual methods running the same requests on their own showed similar performance. Computing the transaction’s signature showed a performance impact of around 10 milliseconds. This result is comparable to the results of the benchmarking performed by the maintainer of the utilized JavaScript library [47]. Since most of the time for the operation is lost while connecting to remote systems, the mobile port of the Bazo client, described in 6, could significantly improve the performance of this operation.

## 5.2 Qualitative Evaluation

This section evaluates the developed P2P mobile payment application against key characteristics for such a solution. Qualitative Key characteristics of a mobile payment solution can be [22]:

1. **Payer control** The degree as where a user can select the most advantageous terms for his payment.
2. **Security** The degree of risk of having funds manipulated or stolen by engaging in the payment system is related to Security.
3. **Universality** This key characteristic deals with the importance of a payment system being accepted by a large user base to make the payment solution attractive.

Payer control with the Bazo currency has a mixed image. The user has the control to create transactions at terms he prefers, for example, the user can set the fee he is willing

to spend on the transaction. However, the clearance of the transaction highly depends on the network. That is the current amount of unconfirmed transactions, the rate at which transactions are validated and the fee the user has set. The user has little influence on these conditions.

Security, as in the definition above, should be given for all transactions signed by the user. Section 5.3 outlines cases where the user can be tricked into revealing his private key. From a technical point of view, the Wallet can be considered secure, as that it is not possible to steal a private key or manipulate transactions which would result in the loss of funds for the user.

Universality is not a strength of the Bazo currency. Since the Wallet is not compatible with other payment systems or applications, only users in the system can exchange funds. Since Bazo is a newly created cryptocurrency, there is no user base and users would have to be convinced to join the system. Given that the target users should already be in the bonus program, the usage of the Bazo currency could be offered an incentive. Since the mechanisms in the Wallet for communicating with the currency are very similar to the mechanisms in other currencies such as Ripple, it would technically be feasible to reuse the Wallet for other currencies and map the bonus points to said currency [8], thus leveraging an existing user base.

## 5.3 Limitations

This chapter introduces limitations of the developed systems and how they can be improved.

- **Trust** It is an objective of many cryptocurrencies to be as independent from third parties as possible. This should allow that assets can be traded in a trustless way. Since the application is designed as a Signing-Only Client, there needs to exist a certain amount of trust between the user of such a Wallet and the server he relies on [23]. With this architecture, it is technically possible to tamper with the information that is sent to the client, such as account balances. This is due to the client being unable to verify the transactions in the same manner that other applications can. However, it is not possible to modify outbound transactions and thus steal assets from the user of the developed system. This is ensured by having all transaction signing implemented in the browser. Since sharing transaction data is not performed over the same server, but rather on a P2P basis, it is also hard to trick the user into targeting the wrong address. Another solution to further reduce this threat is giving users the freedom to connect to a specific server. In the Bazo Wallet, the user has the possibility to set a URL on the *Settings* Page. This URL is persisted locally and used for all further communication. That way, users could deploy the light client to an https enabled server and use this node as a private backend. This approach is frequently employed when dealing with Signing-Only Clients [23]. It is also technically possible to run the light client locally on an Android phone, by running the binaries of the light client. This was tested and proven to work as with any other operating system or platform.

- **Phishing** Another risk is introduced with the unified data model of transaction data, since this points to the URL of the Bazo Wallet. One could trick a user into using a web application that looks like the Bazo Wallet, but has the single purpose of stealing the private key. This is a serious risk in cases where the user does not realize that the URL does not belong to the actual Wallet. One solution how this weakness can be handled is to install the progressive web application to the home screen. On Android systems, this will associate the installed application with the URL of the origin. Therefore, the application will only open if the host of the URL matches the origin of the installed web page.
- **Browser Support** Chapter 4.2 assessed the support of different operating systems for new, web-based APIs. Especially Safari on iOS devices lacks many of the newer features. These inconsistencies between browsers were handled with a progressive approach, where the User Interface only shows the available transferral types. For every device, a fallback solution ensures that there is some way of transferring transaction data. This limitation is unfavorable since the financial service provider estimates the majority of potential users to use an iOS device. Because of this, the development of a native iOS application, that would wrap the web application, is being discussed. However, the assessment of the technical feasibility and the actual implementation are not part of the scope of this thesis.



# Chapter 6

## Future Work

During the development of the Bazo Wallet, limitations with respect to the current implementation of applications in the Bazo ecosystem were observed. This chapter summarizes the issues and possible solutions.

### 6.1 Web-based Headers-only Client

Chapter 2.3.3 contained an analysis of the application that was to be developed later. Due to the elicited requirements and the design guidelines the application was developed as a Signing-Only Client. Considering only the resources available to such an application, it would be possible to develop a headers-only client for a blockchain based currency. For the Bazo currency, this was not applicable for two reasons. Firstly, a web application can not participate in the peer-to-peer network over a TCP connection. Secondly, a solution for using JavaScript or a similar mechanism to validate blocks would need to be found. The first of these issues could be solved by implementing a web service that allows a web application to download block headers over an https or wss connection. This web service could be in the form of a standalone application or integrated into the RESTful API of the Bazo Light client.

### 6.2 Native Mobile Client

Chapter 4.3.3 proved that it is technically possible to run the Bazo client and even the Bazo miner application on a mobile device running Android. This was done with the intention to allow the PWA to communicate with a trustworthy Bazo node. The necessity of this solution lies in the weakness of Signing-Only Clients which rely on a web service and thus on a third-party. To allow a user to use the compiled binaries in a comfortable manner, the following issues would have to be evaluated and solved:

- **Android wrapper application** The compiled binaries of the PoC can be run from the command line. However, an Android application that would wrap and run the binaries would need to be developed.
- **Communication between native and web application** Most of the communication between the two applications could be achieved through the RESTful API. However, running the two applications consistently would possibly require extension of the API. Another way how these types of applications can exchange data based on user input is through the intent system. This approach was also proved to work in the NFC Bridge application, see 4.2.3, and might be a more resilient channel of communication.
- **Network constraints** Chapter 4.3.3 outlined constraints posed on a PWA. This would imply for the PWA to establish a secure connection to the native application. Depending on the way how the two applications communicate, this could involve further efforts, such as enabling the native application with https.

## 6.3 Integrating Third-Party services

Chapter 3.2.3 and 3.2.4 introduced two processes dependent on interfaces and applications on the side of the financial service provider. The mechanisms, processes and interfaces required in the company's infrastructure were designed in partnership. All necessary operations in the Wallet have been implemented. At the time of writing, the necessary developments on the company's side have not yet been completed. This means that minor adjustments to the code base of the PWA, such as setting an URL for the submissions, are needed in the future. In order to simplify the development of the application required in the company's backend, the scope of the Bazo API library was extended for the transaction types that will be needed in the onboarding process. Thus, the developers on the company's side have two possibilities to execute the required functionality on the Bazo network. It is possible to run the operation directly by running the binary as a subprocess, or through the web API. If the latter approach is chosen, it is possible to make plain http requests to the API or reuse the Bazo JavaScript API.

# Chapter 7

## Summary and Conclusions

The overarching goal of this thesis was the enrichment of the Bazo ecosystem to a point, where the bespoke cryptocurrency can be used in a sandboxed environment for mobile payments. This was achieved by implementing a PWA-based Mobile Wallet with the respective interfaces to the system. Through this application, the financial service provider can take the first steps at evaluating the payment system and comparing it to traditional alternatives. Due to the strongly web-focused approach of the implementation, the current state of different Web APIs was explored, tested and documented. With the development of a PoC, an architecture where communication between native and web application is possible, was assessed. This revealed strengths and weaknesses of a web-based approach to the development of a Mobile Wallet. Another PoC lied in porting the existing applications to the Android operating system. With this, the technical feasibility of a completely trustless, web-based Wallet was demonstrated; a key factor that other popular Wallets are not capable of.

The introduction of the thesis lied in a motivation and description of work. In the subsequent chapter, related technologies, activities and the elicited requirements were explained to introduce the reader to the context. The design of the envisioned solution was then explained for the aforementioned requirements. The implementation of this design was then detailed in the fourth chapter, where the development of the required functionalities was presented. The Evaluation chapter highlighted strengths and weaknesses of the developed system. The following chapter outlined open questions that would ideally become the subject of further research.

The developed Bazo Wallet can be used as a fully functional Signing-Only Client for the cryptocurrency. Further, the application provides different features, so that it can be used as a mobile payment system. This condition is not met with other popular Wallets for comparable currencies. By employing a web-based approach to the development of such an application, different operating systems on mobile and desktop devices were targeted with a single code base. However, since the support for the aforementioned features varies on different platforms, the development of a native application wrapper of the Wallet is planned by the financial service provider. This architecture has been proven to work by the developed PoC, which was part of this thesis. By employing this hybrid architecture,

the company intends to deliver a unified customer experience across devices, while still leveraging the web-based code base as much as possible.

# Bibliography

- [1] Livio Sgier: Bazo - A Cryptocurrency from Scratch, 2017.
- [2] Kenneth Rohde Christiansen, Zoltan Kis, Alexander Shalamov: Web NFC API, <https://w3c.github.io/web-nfc/>, accessed 2nd January 2018.
- [3] Android NFC, <https://developer.android.com/guide/topics/connectivity/nfc/nfc.html>, accessed 10th December 2017.
- [4] Apple Inc: Core NFC, <https://developer.apple.com/documentation/corenfc>, accessed 26th November 2017.
- [5] Golang: gobind Documentation, <https://godoc.org/golang.org/x/mobile/cmd/gobind>, accessed 3th January 2018.
- [6] Nils Schneider, Matt Corallo: URI Scheme, <https://github.com/bitcoin/bips/blob/master/bip-0021.mediawiki>, accessed 20th December 2017.
- [7] Simon Bachmann: Proof of Stake for Bazo, 2018.
- [8] Ripple-lib, <https://github.com/ripple/ripple-lib/blob/develop/docs/samples/payment.js>, accessed 20th December 2017.
- [9] Apple Inc: Safari 11.0, [https://developer.apple.com/library/content/releasenotes/General/WhatsNewInSafari/Safari\\_11\\_0/Safari\\_11\\_0.html](https://developer.apple.com/library/content/releasenotes/General/WhatsNewInSafari/Safari_11_0/Safari_11_0.html), accessed 4th January 2018.
- [10] Apple Inc: iOS 11 available tomorrow, <https://www.apple.com/chde/newsroom/2017/09/ios-11-available-tomorrow/>, accessed 4th January 2018.
- [11] Alex Russell, Jungkee Song, Jake Archibald, Marijn Kruisselbrink : Service Workers, <https://www.w3.org/TR/service-workers-1/>, accessed 26th December 2017.
- [12] Marcos Caceres, Kenneth Rohde Christiansen, Mounir Lamouri, Anssi Kostiaainen, Rob Dolin: Web App Manifest, <https://w3c.github.io/manifest/>, accessed 26th December 2017.
- [13] Apple Inc: Supported Meta Tags, <https://developer.apple.com/library/content/documentation/AppleApplications/Reference/SafariHTMLRef/Articles/MetaTags.html>, accessed 26th December 2017.

- [14] World Wide Web Consortium (W3C): Implementation Report, <https://w3c.github.io/test-results/webstorage/all>, accessed 28th December 2017.
- [15] Ian Hickson: Web Storage, <https://www.w3.org/TR/webstorage/>, accessed 28th December 2017.
- [16] Mark Watson: Web Crypto API, <https://www.w3.org/TR/WebCryptoAPI/>, accessed 28th December 2017.
- [17] Vue.js: Vuex, <https://vuex.vuejs.org/>, accessed 10th January 2018.
- [18] MyEtherWallet.com, <https://www.myetherwallet.com/>, accessed 10th January 2018.
- [19] Terri Bradford, William R. Keeton: New Person-to-Person Payment Methods: Have Checks Met Their Match? <https://www.kansascityfed.org/VXnYf/publicat/econrev/pdf/12q3Bradford-Keeton.pdf>, p54, Federal Reserve Bank of Kansas City Economic Review 2012.
- [20] Leif Erik Kleivene: P2P Mobile Payments: Investigating the factors of adoption among students in Germany, <http://lipisadvisors.com/wp-content/uploads/2017/01/White-Paper-P2P-mobile-payments.pdf>, 2016.
- [21] Fiserv, Inc: The Four Pillars of Mobile Payments - Immediate Opportunities, [https://www.fiserv.com/resources/Four\\_Pillars\\_Mobile\\_Payment\\_1404.pdf](https://www.fiserv.com/resources/Four_Pillars_Mobile_Payment_1404.pdf), 2014.
- [22] Mobile Payments and Bitcoin: Concluding Reflections on the Digital Upheaval in Payments, in Gabriella Gimigliano: Bitcoin and Mobile Payments, p281, 2014.
- [23] Rostislav Skudnov: Bitcoin clients, [http://publications.theseus.fi:80/bitstream/handle/10024/47166/Skudnov\\_Rostislav.pdf](http://publications.theseus.fi:80/bitstream/handle/10024/47166/Skudnov_Rostislav.pdf), 2012.
- [24] Google: Progressive Web Apps, <https://developers.google.com/web/progressive-web-apps/>, accessed 30th December 2017.
- [25] Google: Prefer Secure Origins For Powerful New Features, <http://www.chromium.org/Home/chromium-security/prefer-secure-origins-for-powerful-new-features>, accessed 30th December 2017.
- [26] Mastering Bitcoin: Unlocking Digital Crypto-Currencies, O'Reilly Media, 2014.
- [27] WebNFC Community Group: Issues, <https://github.com/w3c/web-nfc/issues/139>, accessed 31th December 2017.
- [28] The Chromium Projects: Origin Trials, <https://www.chromium.org/blink/origin-trials>, accessed 2nd January 2018.
- [29] NFC Forum: About the technology, <https://nfc-forum.org/>, accessed 2nd January 2018.
- [30] Bluetooth SIG Inc: Low energy: Point-to-Point, BluetoothLE:Point-to-Point, accessed 2nd January 2018.

- [31] Bluetooth SIG Inc: GATT Overview, <https://www.bluetooth.com/specifications/gatt/generic-attributes-overview>, accessed 2nd January 2018.
- [32] François Beaufort, Google Inc: Interact with Bluetooth devices on the Web, <https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web>, accessed 2nd January 2018.
- [33] Coinblesk: coinblesk-frontend-instascan, <https://github.com/coinblesk/coinblesk-frontend-instascan>, accessed 3rd January 2018.
- [34] Miguel Casas-Sanchez (Google Inc.): Accelerated Shape Detection in Images, <https://wicg.github.io/shape-detection-api/>, accessed 3rd January 2018.
- [35] W3C: Shape Detection API Specification - Current Related Efforts and Workarounds, <https://github.com/WICG/shape-detection-api#current-related-efforts-and-workarounds-wrench>, accessed 3rd January 2018.
- [36] Paul Kinlan, Sam Thorogood (Google Inc): Introducing the Web Share API, <https://developers.google.com/web/updates/2016/09/navigator-share>, accessed 3rd January 2018.
- [37] Matt Giuca (Google Inc): Web Share API , <https://wicg.github.io/web-share/>, accessed 3rd January 2018.
- [38] Google Inc: Standalone Toolchains, [https://developer.android.com/ndk/guides/standalone\\_toolchain.html](https://developer.android.com/ndk/guides/standalone_toolchain.html), accessed 3rd January 2018.
- [39] Golang: Generate go files by processing source, [https://golang.org/cmd/go/#hdr-Generate\\_Go\\_files\\_by\\_processing\\_source](https://golang.org/cmd/go/#hdr-Generate_Go_files_by_processing_source), accessed 3rd January 2018.
- [40] Coinblesk: coinblesk-frontend, <https://github.com/coinblesk/coinblesk-frontend>, accessed 3rd January 2018.
- [41] Luc Boillat: Bazo Block Explorer, <https://github.com/lucBoillat/BazoBlockExplorer>, accessed 3rd January 2018.
- [42] World Wide Web Consortium: Real-time Communication Between Browsers: Implementation Report, <https://wpt.fyi/webrtc>, accessed 3rd January 2018.
- [43] Google Inc: Lighthouse, <https://developers.google.com/web/tools/lighthouse/>, accessed 8th January 2018.
- [44] Google Inc: Progressive Web Application Checklist, <https://developers.google.com/web/progressive-web-apps/checklist>, accessed 8th January 2018.
- [45] WebKit: Bug 174541: Service Workers, [https://bugs.webkit.org/show\\_bug.cgi?id=174541](https://bugs.webkit.org/show_bug.cgi?id=174541), accessed 16th January 2018.
- [46] Pingdom: Website speed test, <https://tools.pingdom.com/>, accessed 16th January 2018.

- [47] Fedor Indutny: Elliptic <https://github.com/indutny/elliptic>, accessed 21st January 2018.



# Abbreviations

AccTx	Transaction with the purpose of Account Registration [1]
BTLE	Bluetooth Low Energy
ECDSA	Elliptic Curve Digital Signature Algorithm
FundsTx	Transferring Funds Transaction [1]
HTTP(S)	Hypertext Transfer Protocol (Secure)
JSON	JavaScript Object Notation
NDEF	NFC Data Exchange Format
NDK	Native Development Kit
NFC	Near Field Communication
OySy	Oyster System - The name under which the Bazo project was communicated externally
P2P	Person-to-Person payment or Peer-to-Peer network
PoC	Proof of Concept
POS	Point of Sale
PWA	Progressive Web Application
QR Code	Quick Response Code
RTC	Real-Time Communication
SHA	Secure Hash Algorithm
SPA	Single Page Application
SSL	Secure Sockets Layer
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSS	WebSocket Secure



# Glossary

**Karma Backend** Bespoke application in the financial service provider's infrastructure.  
This allows third party applications to operate on customer data.

**User Agent** Client application for accessing remote services.

**Near Field Communication** Technology for proximity-based data transfers, typically  
for ranges under ten centimeters.



# List of Figures

3.1	Payment process using an NFC tag with static transaction data. . . . .	13
3.2	Process of generating and registrating a new account. . . . .	14
3.3	Process of requesting new Funds for an existing account. . . . .	15
4.1	Comparison of the PWA installation prompt in Firefox (left) and Google Chrome (right) on an Android device. . . . .	18
4.2	Home Page with Side Bar and Navigation Bar on a Desktop Device. . . . .	21
4.3	<b>Funds Page:</b> Inspect and create new funding requests from bonus points. . . . .	22
4.4	<b>Funds Page:</b> Generate, save and export a new Bazo account for new funding requests. . . . .	23
4.5	<b>Accounts Page:</b> Add and inspect existing Bazo accounts. . . . .	24
4.6	<b>Accounts Page:</b> Importing an account from a keyfile that was generated on the <i>Funds</i> page. . . . .	25
4.7	<b>Request Page:</b> Compile shareable payment information. . . . .	26
4.8	<b>Request Page:</b> Transfer payment information with one of the transfer methods. . . . .	27
4.9	<b>Send Page:</b> Fill in payment information or obtain it through one of the transfer methods available. . . . .	28
4.10	<b>Send Page:</b> Confirm and sign the payment. . . . .	29
4.11	Settings Page in the Bazo Wallet. . . . .	30
4.12	Requesting, building, signing and submitting payment information with the Bazo Wallet. . . . .	38
4.13	Programmatically creating, signing and sending a transaction with Ripple-lib [8]. . . . .	39

5.1 Summary of a Performance report using Pingdom. . . . . 43

5.2 Coverage report of the Bazo API library. . . . . 44

# List of Tables

2.1	Comparison of different client solutions existing in the Bitcoin network [23].	6
5.1	Overview of tested browsers and operating systems. . . . .	43





# Appendix A

## Installation Guidelines

### A.1 Wallet Application

In order to compile the source code of the Bazo Wallet application, *npm* needs to be installed on the system. If *node.js* is installed on the system, *npm* should be already included as well. Otherwise, download and install *npm* and the *node.js* runtime from the website:

```
https://www.npmjs.com/get-npm
```

After having the package manager installed, install *yarn* and then use it to fetch the dependencies from the *npm* registry as follows. From the command line, run the following to install *yarn* globally:

```
npm install -g yarn
```

This step may require root access to the system where *yarn* is installed. Upon installation, the dependencies can be fetched by running the following in the directory of the source code:

```
yarn install
```

With all the dependencies met, it is now possible to run an integrated development server on *localhost* without compiling and optimizing all source files:

```
yarn run dev
```

This server will not meet the requirement posed on Progressive Web Applications, namely that they need to be served from a secure context. So, in a production environment one should build the actual files using another supplied script:

```
yarn run build
```

This will build and transpile all source files. Further, the resulting project will be optimized for performance by splitting files into chunks and compressing them. The built project can then be found in the *dist* directory of the current working directory.

It is possible to run the built project locally by running:

```
node start-server.js
```

However, to deploy the project to a suitable production environment, the *dist* folder needs to be served from an http server. Any web server can be used to serve the project, however using a server where *https* is optimal. A simple solution to test-deploy the project to a server without the need to configure https is using a webservice like *surge*. Since npm should already be installed, installation is as simple as running:

```
npm install -g surge
```

Upon successful installation, the dist project can be deployed to an https enabled server by running:

```
surge -d 'https://<subdomain of choice>.surge.sh' -p 'dist'
```

This command will tell surge to deploy the directory *dist* to the supplied subdomain. If the subdomain is prefixed with *https* as with the command above, all incoming *http* requests will be redirected to the secure context. Once the project is deployed, the user can navigate to the URL, where he will be automatically asked if he wants to install the Progressive Web Application to the device.

## A.2 Installing and Running the NFC Bridge

The NFC Bridge PoC can be installed in two ways. For both methods, developer options have to be enabled on the Android device. Follow the instructions from the official Android Studio documentation to enable developer options:

```
https://developer.android.com/studio/debug/dev-options.html
```

Now, connect your device to the computer using e.g. an USB cable and follow one of the installation types below:

1. Importing, compiling and deploying the source code with Android Studio First, install a build of Android Studio Canary. Builds can be found at:

```
http://tools.android.com/download/studio/canary/latest
```

Start Android Studio, and import the NFC Bridge source code as a gradle project. Click on *Run* in the application menu and select *Run 'app'*. This should prompt a dialog with a list of connected Android devices. Select the target device and confirm by clicking *OK*. On some devices, the Android system will show a dialog and ask the user if *USB Debugging* should be allowed for the requesting computer. After this is granted, the application is automatically installed and opened.

2. Deploying the compiled *.apk* file. Connect your Android powered device to the computer and copy the NFCBridge *.apk* file to the device. The file can then be opened from the Android system. This will install the application permanently and run it. This installation type requires that unknown sources are trusted for installing Android applications. The process of granting this right is described here:

```
https://developer.android.com/distribute/marketing-tools/  
alternative-distribution.html#unknown-sources
```

## A.3 Installing and Running a Bazo node on Android

In order to run a complete Bazo node on Android device, several binaries have to be run. A full Bazo node consists of the Bazo miner and light client, of which the latter contains the web interface. This installation uses *Android Debugging Bridge* to obtain a shell session on the Android device. First, connect and configure your target Android device as described in the official documentation:

```
https://developer.android.com/studio/command-line/adb.html#wireless
```

Once the device is properly connected, the binaries can be pushed to the device. The following command is run from the root directory of the Android Port project:

```
adb push bin/bazo*-android-armv5 /data/local/bin
```

The target directory can be any directory with write access for the adb shell. Once the files are copied to the device, they can be run directly if root access is enabled on the phone.

```
adb shell /data/local/bin/<binary>
```

If this is not the case, the VM of an existing Android application can be used to run the binaries in. Open an interactive shell on the device:

```
adb shell
```

Now use the following command to attach to an installed application. The following command demonstrates this with the NFC Bridge application:

```
run-as <ch.uzh.ifn.nfcbridge>
```

Copy the binaries to the current working directory of the application:

```
cp /data/local/bin/bazo* .
```

After this step the binaries can be run from the command line. First, the provided root key which is associated with the root account that is also the beneficiary in the miner application, has to be used to create a new non-root user account:

```
./bazo_miner new_database :8000 &  
./bazo_client accTx <header> <fee> <privKey> <keyOutput>
```

With this, the Bazo miner binary is run in the background, while passing the name for a new database file. The last argument will tell the miner to listen on localhost:8000 for inbound connections. Then, the client can be used with the following parameters:

**header** Reserved header for later usage. E.g value: 0

**fee** The fee for the transaction. E.g value: 1

**privKey** Key file containing the key pair of a valid root account. E.g value: ../rootkey

**keyoutput** Name for the key file of the newly created account. E.g. value: new\_user

Now that a user account is available and the Bazo miner application is running in the background, the Bazo client application can be started. When no arguments are passed to the client denoting the transaction type, the client will create the web interface running on port 8001.

```
./bazo_client &
```

Now, the web service will be listening on port 8001 of the Android device, so that this URI can be used in the Wallet application. Navigate to the settings page of the Wallet web application to modify the URI that will be used for all interaction with the Bazo network.

Please note that the development efforts of the light client application have not been completed at the time of writing. The instructions for compiling the project can be used for later versions of the light client. However, the compiled binaries reflect the state of the light client at the time of writing.



# Appendix B

## Contents of the CD

- Bazo Wallet Source Code
- NFC Bridge Source Code
- Android ARMv5 Binaries of the miner and client application. Instructions for cross-compilation
- Source Code of the JavaScript API wrapper library
- Latex Source Code
- Final Thesis (PDF)
- Intermediate Presentation
- Final Presentation