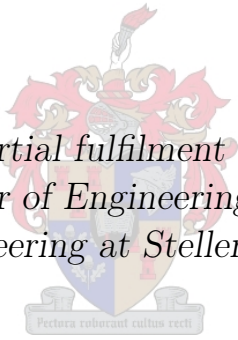
Bitcoin payment framework on a social media platform

by

Wessel Wessels



Thesis presented in partial fulfilment of the requirements for the degree of Master of Engineering (Electronic) in the Faculty of Engineering at Stellenbosch University

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Supervisor: Prof. G. van Rooyen

December 2015

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2015/12/10 .....

Abstract

Bitcoin payment framework on a social media platform

W. Wessels

Department of Electrical and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MEng (E&E)

December 2015

English abstract to be written

Uittreksel

Bitcoin betalingsraamwerk op 'n sosiale media platform

("Bitcoin payment framework on a social media platform")

W. Wessels

Departement Elektries en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MIng (E&E)

Desember 2015

Afrikaanse uittreksel wat nog geskryf moet word

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations ...

Dedications

Hierdie tesis word opgedra aan ...

Contents

List of Figures

List of Tables

Nomenclature

No nomenclature yet.

Chapter 1

Introduction

Bitcoin [1] is a peer-to-peer decentralised digital payment mechanism that was introduced in a paper published by a person or group called Satoshi Nakamoto in 2008. Bitcoin aims to be an alternative to traditional centralised online payment mechanisms like credit cards and PayPal [2].

We focus on testing the viability of Bitcoin as payment mechanism on social media platforms, specifically on a mobile platfom.

1.1 Background

1.2 Related Work

1.3 Objectives

Chapter 2

Background

2.1 History

Chapter 3

System Design

3.1 Framework

In this project we test the viability of a payment framework on a mobile social media platform. The framework will consist of several independant but connected pieces:

- REST API for payments
- Wallet Application
- Bitcoin Interface
- Use-case Application

A summary of what is required from the system can be seen in figure 3.1.

3.1.1 REST API for payment management

A REST (Representational State Transfer) API [3] was chosen for the main interface for developers to use Bitcoin without running a Bitcoin node or having experience with Bitcoin. REST was chosen because it is a commonly used architecture, it is easy to use and understand and it does not constrain the user's choice of programming langauge or environment.

The purpose of the REST API is to let developers make payment requests and check if a payment has been made, without dealing with the low-level Bitcoin transactions directly. Thus, our system should generate a new Bitcoin address on request.

Our requirements from the REST API are:

- New Bitcoin address for each payment
- Verify payment

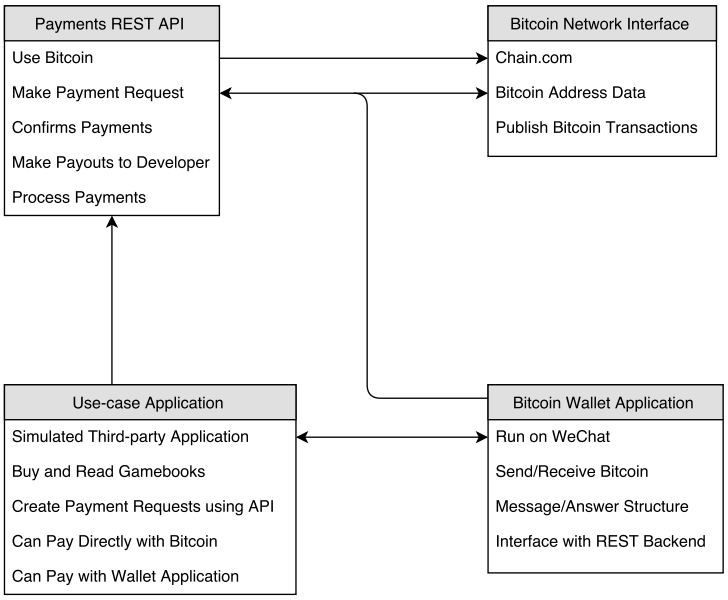


Figure 3.1: Summary of Framework

- Check total balance of developer
- Witdraw available Bitcoin of developer

3.1.1.1 The concept of the Bitcoin payment

This is a high-level explanation of how to receive verifyable payments with Bitcoin. With Bitcoin, unlike a traditional bank account, you don't have a single "account" where people can make payments to and you can verify that the payment came from them. With Bitcoin it is trivially easy to make a new Bitcoin address, and it can be generated without being connected to the Internet or the Bitcoin network.

Since the entire Bitcoin blockchain is public, a single address is not sufficient to receive multiple payments. With a single address, it is not easy to verify that a spesific person has made a payment, since there may be several payments of the same amount happening in short succession.

The sollution to the problem is generating a new address for every payment, and requesting that the user make the payment to that address. Since the newly generated address is not yet present on the blockchain, when a payment to that address of the requested amount occurs, we can be certain that the person in question made the payment. When the payment is complete, the Bitcoin in that address can be transferred to a central address, and the original address can be discarded.

From our requirements for the REST API, we clearly require (at least) the following methods:

- A payment method
- A balance method
- A payout method

3.1.1.2 The /payment method

The /payment method is the core of the REST API. It is used to make a payment request with a specified amount of Bitcoin and a description of the transaction. The /payment method returns a Bitcoin public address and a payment ID.

The user can then pay to the Bitcoin address using any standard Bitcoin payment method, or can pay directly from the Bitcoin wallet that will run on WeChat and will be connected to the payment infrastructure.

3.1.1.3 /payment/{PUBLIC_ADDRESS} and /payment/{ID}

These two methods are conceptually the same, but they take in two different arguments. The one takes the Bitcoin address to be queried, and the other takes the payment ID. The method returns all the data about the transaction, including the status of the transaction.

The main purpose of this method is to verify that a transaction has been completed by the user. It can also be used to give the payment details to to user again.

3.1.1.4 /balance

The /balance method gives the developer the balance of all the available Bitcoin from all the received transactions. The method also returns a flag that says if there is enough Bitcoin to make a payout.

3.1.1.5 /payout

The /payout method is used by the developer to transfer all of the available Bitcoin to a specified Bitcoin address.

3.1.2 Wallet Application

The Wallet Application is a Bitcoin wallet implemented on the WeChat platform. The WeChat platform uses a simple message-answer structure. A user sends a message in the Wallet Application. The message is then sent to WeChat that sends it to a third party server controlled by the developer. The server then sends a reply to WeChat that is then forwarded to the user.

In this manner, a fully functional Bitcoin wallet is realised. The third party server stores the private keys and processes the Bitcoin transactions on commands from the user.

The advantage of using the WeChat platform is the security built in to the platform, as well as an existing userbase.

The Wallet Application will be directly connected to the back-end of the REST API. Thus, payment requests will be referable directly from the Wallet Application without needing to reference the Bitcoin Address. It will be able to reference the request using the payment ID mentioned in 3.1.1.2

3.1.3 Bitcoin Interface

To connect to the Bitcoin peer-to-peer network, a Bitcoin client is needed. The standard way of doing this is running the Bitcoin open source software on a server. This is very network and processor intensive. For development and testing, it will be quite expensive to run the Bitcoin software. Thus, an alternative is required for interfacing with the Bitcoin network. Fortunately, there are services that provide access to most of the Bitcoin operations using their API's.

We require the following from such a service:

- Get the balance from an address,
- Get unspent outputs from an address,
- Post a signed Bitcoin transaction to the network,
- It must be able to use the Testnet

The details of the chosen service is covered in chapter 4.

3.1.4 Use-case Application

To use the payment framework, a Gamebook application is created to read Choose Your Own Adventure style books on the WeChat platform. User-created books can be sold by using the Bitcoin payment framework and the author can potentially earn Bitcoin.

For each sale, the Gamebook application creates a payment request using the REST API. The user can then pay using the Wallet Application or any Bitcoin payment mechanism. The Gamebook application can the query the API to confirm that the payment is received.

Chapter 4

Detail Design

4.1 Back-end

To implement the payment framework, we need a back-end server and a back-end web framework.

4.1.1 Back-end Server

We chose an Amazon Elastic Cloud Computing (EC2) instance for the back-end server. EC2 gives us access to a virtual machine (VM) where we can run our own software, including a publicly accessible website. The micro instance is deployed in Singapore in the Asia Pacific region. The reason for this is to have the server as close as possible to the WeChat servers in China, since our servers must connect to WeChat's servers.

4.1.2 Back-end Web Framework

We decided to use XAMPP (Apache, MySQL, PHP and Perl) for the back-end development environment. XAMPP is a full stack development environment. It includes an HTTP server (Apache), a database server (MySQL) and a scripting language (PHP). XAMPP is free and easy to deploy, and is also used because the author is familiar with it.

4.2 Social Media Platform

We chose WeChat for our social media platform. WeChat works on most smartphones, already has a userbase and it has a third-party API with a development sandbox feature.

On WeChat, a third-party application is known as an "Official Account". We registered for a sandbox Official Account that only allows 20 users and is not searchable on WeChat.

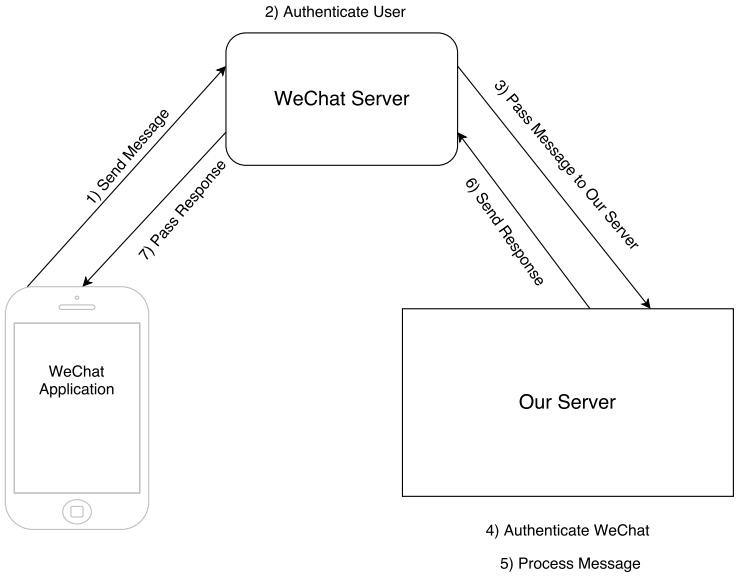


Figure 4.1: Interaction with WeChat

WeChat acts as an intermediary between the user and our server as seen in figure 4.1.

4.2.1 WeChat Security

WeChat uses a shared token hashing scheme to authenticate itself on our server. We provide WeChat with a unique string to use for authentication purposes. That string is then used in every request that is made to our server. WeChat uses the string we provided, a random nonce and the UNIX timestamp to create a signature. It sorts the string, timestamp and nonce and forms a single string from these three values. This single string is then hashed using the SHA256 hashing algorithm to generate a signature. When a request is made to our server, WeChat provides the nonce, the timestamp and the signature The message does not contain the unique string. Since we know the unique string, we can use the nonce and timestamp to also generate a signature. If the message comes from someone that possesses the same unique string, the signature that is provided must match the signature that we calculated.

Thus, we can be reasonably certain that any request that is made to our server with a signature that is verified comes from WeChat and not an attacker. If our unique string is compromised somehow, someone will be able to make false requests to our server that appear valid. If this would happen, we can give WeChat a new unique string.

4.2.2 WeChat Interface

The WeChat Official Account interface works on a message-answer basis. Any message that the user sends in the Official Account dialog is forwarded to our server. When we configure our Official Account, we provide WeChat with a URL that points to the script that handles all messages from and to Wechat. This script verifies any incoming messages as described in section 4.2.1.

The incoming message is in XML format. It contains a unique identifier for every subscriber to the WeChat Official Account. It is important to note that the unique identifier is only unique for the spesific Official Account. The identifier is not a global unique identifier for the entire WeChat platform. Thus, the same user will have two different identifiers in two different Official Accounts. This is important to remember, since we will be using two seperate Official Accounts.

The messages that are received must be interpreted and responded to accordingly. Since we will have applications that rely on previous messages and results, we need something to keep track of the user's current state. A state machine is required to look at the current state the user is in, look at the message they sent and accordingly determine what to reply and to what state to move to.

To keep track of the user's state, we will use a MySQL database as described in section 4.1.2. Thus, for each message received, we will read the user's current state from the database and apply the state machine logic to their message. We will then update their state in the database and reply with the message that the state machine determined.

4.2.3 Gamebook Design

The use-case application we chose to demonstrate the payments framework is a Gamebook application. A Gamebook is a non-linear book that tells a story based on the user's input. In physical books, this is done by giving the user a choice and then telling them what page to go to based on their choice. In the digital realm, we can simply provide the user a choice and give them the corresponding text. The software keeps track of what options links where.

In a dedicated Gamebook reader, a common method of generating a Gamebook is using a scripting language like ChoiceScript [4]. Since WeChat doesn't have any client-side scripting, using a scripting language like this is not possible.

We decided to create our own system of storing and reading Gamebooks. We use the MySQL database as the method of storing and organising the Gamebooks.

ID	Text	Choice 1	Choice 2	Choice 3	Choice 4	Choice 5

Figure 4.2: Gamebook Database Table

4.2.3.1 Gamebook Database Design

Every book in our design is a table, and every page is an entry in the table. Each entry has a unique id, the text of the page and then the references of each of the choices from that page. We chose five to be the maximum amount of choices on each page. The database resembles a linked list, where each entry has pointers to the next corresponding entry. The design of the Gamebook table is seen in figure 4.2.

The values of the fields choice 1 - 5 will have the corresponding ID's of the "page" they link to. If an entry only has two choices, the rest of the options will have the value null to indicate that it is not a valid choice.

We chose this method of storing the Gamebooks, because it is easy to implement, easy for a writer to visualise and allows us to have sircular stories and also allows us to visit multiple branches of the story.

We will also need two more tables: one to store a list of all the books available and another to keep track of books purchased by individuals.

4.2.3.2 Gamebook Author Platform

As stated in section 4.2.3, we are not going to use a scripting language to write the Gamebook in. It doesn't fit the design of the application that we are building, and has a learning curve for people that want to start writing books and are not familiar with scripting languages. We decided to use a Graphical User Interface for writing the Gamebooks.

We decided to use a website as the author platform. We used commonly used web technologies to build the platform. For the front-end, we used HTML, JavaScript with jQuery, and Twitter's Bootstrap CSS framework.

Using AJAX, we connect the front-end with the back-end PHP scripts that connects to the MySQL database. We also decided to use an OAuth login mechanism to facilitate logins, rather than writing our own login system. A big motivation for doing this is the authors desire to get experience with OAuth. We decided to use Google's OAuth platform, because it widely used and well documented. By using OAuth, we are enabled to have unique, secure logins without having to design all the security measures to protect the user. It also makes it easier for the user, because they can log in with a single click.

Since the Gamebook resembles a tree, we decided to display the tree using an organisational chart. We used Google's JavaScript Chart API to map the Gamebook. The Chart API is free, simple, powerful and well documented. To display the story as a tree makes it easier for the writer to navigate through the book and to get a bigger picture of the story. An example of such a chart

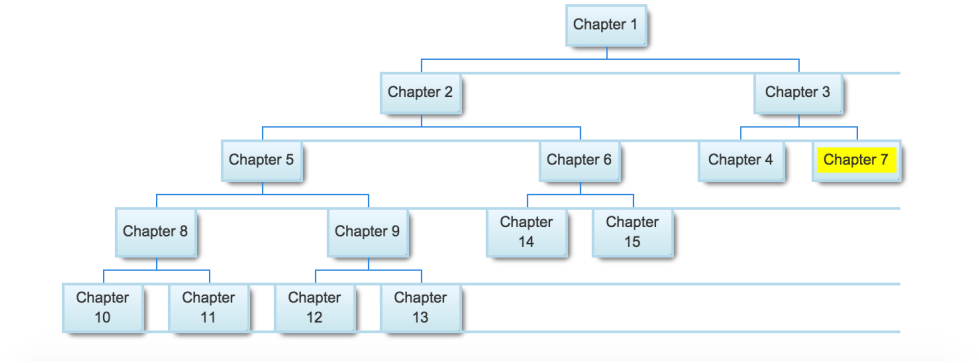


Figure 4.3: Gamebook Tree Representation

can be seen if figure 4.3. Using a chart makes it easier for the author to see where the story needs more work.

4.3 WeChat Wallet Design

As stated before, the WeChat Official Account doesn't allow us to run client-side code. This means that the WeChat Wallet will have to be a hosted Bitcoin wallet, with the Official Account interfacing with the hosted wallet.

To have a WeChat wallet, we need a Bitcoin address. When hosting a Bitcoin wallet, there are two main method of keeping addresses. The first method is where a user gets a single address or adresses and these addresses are used only by the single user. The user may or may not have access to the private key, but the wallet is associated with only that user. This allows the user to verify the balance and transactions by checking the blockchain.

The second method is where the user has an account, and can have a receiving address, but the user is not directly in control of the address. The users' balance is not verifiable by using the Bitcoin blockchain. This is because the server keeps track of the balance of the user, and makes a payment from other addresses when a user wants to make a payment.

We chose the first method of storing the address, because it is simpler and has a more direct feeling for the user that he is using a real Bitcoin wallet, and not just something that arbitrarily keeps track of the balance.

4.3.1 Bitcoin Interface Design

Thus, we need to generate a Bitcoin address for each user. To do this, we use a a popular open-source PHP Bitcoin library called bitcoin-lib-php. One of the features of Bitcoin is that generating a Bitcoin wallet can be done locally. That means that the bitcoin-lib-php library generates a Bitcoin address on our server without connecting to the Bitcoin network. This has many advantages,

including it doesn't use network bandwidth, it is fast and the private key never leaves our server.

We chose bitcoin-lib-php as our library, because it satisfies our requirements perfectly, it is open-source (and thus verifiable) and is decently documented.

As mentioned in section 4.3, by creating an address for each user, the user can more directly monitor his funds and transactions since it can be independently verified by using any software that is connected to the Bitcoin blockchain.

The Bitcoin library allows us to create an address, create a Bitcoin transaction and sign a transaction. These are the core functions of Bitcoin. However, a transaction can't be created without information about the address or addresses that want to create the transaction. This information is called "unspent outputs" and, as the name suggests, are previously received transactions that are not yet spent. These unspent outputs contain all the information to build a transaction, including the value of the output, the transaction hash and the script type etc.

These unspent outputs are only acquirable with access to the Blockchain and we do not run the Bitcoin software. Thus we do not have direct access to these outputs. From our requirements in section 3.1.3, we require a third-party Bitcoin interface to provide these outputs among other things. From the requirements listed in section 3.1.3, we decided to use chain.com. Chain.com satisfies all our requirements and is free at the time of writing.

During the progress of this project, chain.com has release version 2 of their API with the biggest change being that they can also sign a transaction for you. When we started, we had to sign the transaction ourselves. Signing the transaction ourselves exposes us to a smaller risk of compromising the security of the users' private key. We choose to still use version 1 of the API, because updating to version 2 will not have a significant effect on our proof of concept's overall outcome. The summary of the Bitcoin payment flow can be seen in figure 4.4.

4.3.2 WeChat Wallet Features Design

The main function of a Bitcoin wallet application is to send and receive Bitcoin. A Bitcoin address is a string of 26 to 35 alphanumeric characters. Reading an address to someone is not practical due to the long length. A very common method of representing a Bitcoin address is by using a graphical QR-code. Another user can then scan a QR-code and receive the address to make the payment to.

4.3.2.1 Address to QR-code

Our first design specification is to represent the user's alphanumeric address as a QR-code image. There are several ways we can do this. The first method

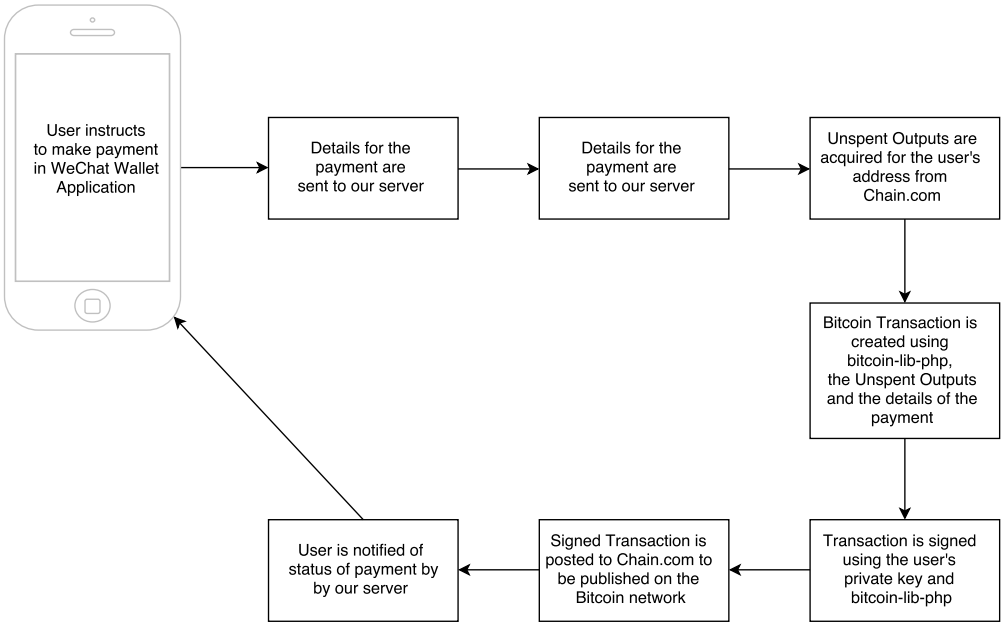


Figure 4.4: Bitcoin Payment Flow

is to generate the QR-image on our server and send the image to the user on WeChat. The problem with this method is that our server will be responsible for generating images, and this is computationally intensive. The second method is using a webpage that generates the QR-code client-side using JavaScript. The WeChat Official Account allows us to open a link directly in WeChat. Thus, we can send the user a link of a webpage that will generate the QR-code in WeChat's built-in browser. With this method, computation is spread to the users and not a central server.

We chose the webpage method of generating the QR-codes. We wrote our own webpage that uses an Open Source JavaScript library to generate the QR-code. We encode the address to be generated in a GET request and send the link to the user. The link will look like this:
https://domain.com/qr?address=1Bd5wrFxHYRkk4UCFttcPNMYzqJnQKfXUE
This QR-code can then be generated when a user wants to receive a payment from someone. An example of a generated QR-code can be seen in figure 4.5.

4.3.2.2 QR-code to Address

Our second design spesifiction is to allow a user to scan a QR-code to make a payment to. Although the WeChat application has a QR-code reader built-in, it doesn't allow the user to scan a QR-code directly from an Official Account. An Official Account does allow a user to send an image to our server. This will allow us to decode the QR-code on our server. For simplicity, we decided to use a third-party service called goQR.me to decode the QR-code. WeChat

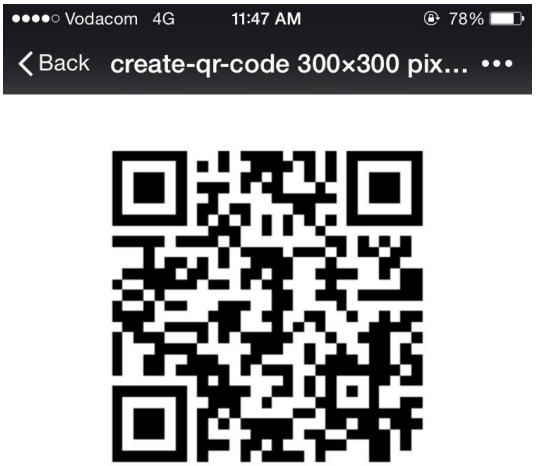


Figure 4.5: Generated QR-code

sends our server a link of the image that the user sent. This link of the image is used by the third-party service and responds with the decoded text. The bitcoin-lib-php software can verify that the decoded text is a valid Bitcoin address.

This method of decoding the QR-code is very computationally- and bandwidth efficient for our server, since the image itself is not sent through our server. Just like the Bitcoin Interface in section 4.3.1, we chose to use a third-party service for our proof of concept. If we were to build the service for production scale, we would rather exchange the third-party service for our own software. In the case of the QR-code service, it would be a major security concern to use a third-party service. A malicious third-party can exchange the correct address with their own address to receive Bitcoin that is not intended for them.

4.3.2.3 User Balance

Getting the balance of the user's address is the second core feature that a Bitcoin wallet application requires. Calculating an address' balance involves summing all the user's unspent outputs. Fortunately, the Chain.com API allows us to retrieve the balance of any Bitcoin address directly using a single method call. The result of getting the balance can be seen in figure 4.6.

4.3.2.4 Make A Payment

The core process of making a payment is handled in section 4.3.1 and can be seen in figure 4.4. The most important part of the payment flow is that the user's Bitcoin private key never leaves our server, even though a third-party service is used to publish the transaction.

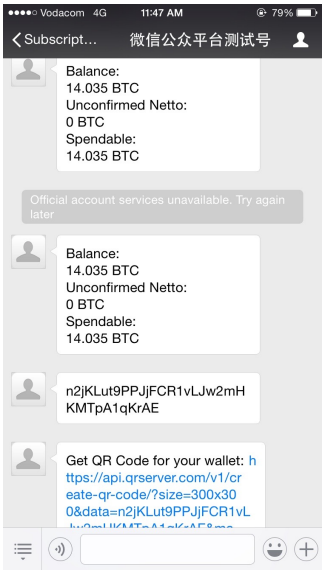


Figure 4.6: WeChat Balance

4.4 REST API Design

There are many ways to implement a REST API. This is because of the "black-box" approach of REST. As such, the first major design decision that needs to be made is whether to use a REST framework or two build it from scratch. Using a REST framework would be easier and standardized and if the framework is maintained it would also be secure.

In spite of the advantages of using a REST framework, we decided to build it ourself. The motivation for choosing this method is that the author wanted to get a better understanding of how a REST API works on the low-level and felt that building it from scratch would give a better understanding. Thus, we will use the same tools to build the other parts of our system, ie. PHP and MySQL.

4.4.1 REST Endpoints

A PHP script is usually called in the following format: "url/script.php". However, for a REST API we need a clean URL in the format: "url/script". Since we are using an Apache server, we need to modify the ".htaccess" file with regular expressions.

Thus, we have different regular expressions in the .htaccess file that will call a single script with the corresponding parameters from the request. The .htaccess file will only relay the parameters of the url to the PHP script. The PHP script will then apply logic to determine what function should be executed.

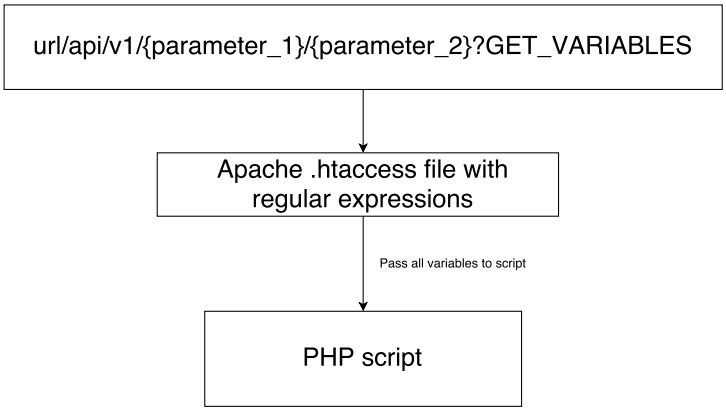


Figure 4.7: REST URL Flow

Flag	Description
NC	No Case. Doesn't differentiate between upper- and lower case.
QSA	Query String Append. Adds the variables to the existing GET variables.
L	Last. If the rule matches, no further rules will be processed.

Table 4.1: Table of selected .htaccess flags

The PHP script will firstly authenticate the developer, followed by the do the corresponding external API calls, database entries, response HTTP status code and JSON response data.

The following is an excerpt from the .htaccess file that is responsible for the routing of the REST endpoints:

```
RewriteRule ^api/v1/([A-Za-z0-9-]+)/([A-Za-z0-9-]+)/?$
/chaintest/my_api.php?first_query=$1&second_query=$2 [NC,QSA,L]

RewriteRule ^api/v1/([A-Za-z0-9-]+)/?$ /chaintest/my_api.php?first_query=$1 [NC,QSA,L]

RewriteRule ^api/v1/?$ /chaintest/my_api.php
```

The first RewriteRule matches queries with two parameters in the URI and turns the parameters into GET variables first_query and second_query. The my_api.php script will then be able to access these variables like normal GET variables.

Chapter 5

Tests

5.1 Quantitative

This project is an implementation of a platform on top of an existing social media platform WeChat. We do not have access to the back-end systems of WeChat. This makes it hard to do many of the quantitative tests required for the application itself. They will require qualitative tests. The REST API however, can be tested thoroughly using unit tests.

5.1.1 REST API unit tests

There are tools and frameworks available for testing REST API's. Just like when making the REST API, we decided to rather write the unit tests ourselves without using a framework in order to get a better understanding of how unit tests work.

Thus, we wrote the entire REST API unit tests in a PHP script that calls all our REST endpoints with inputs that we know what the outputs should be. Each test should also confirm that the correct HTTP status code, as shown in table 5.1, is returned. In total, there are 39 different tests that excecute.

Status Code	Text	Description
200	OK	Request succeeded
400	Bad Request	Request not understood due to malformed syntax
401	Unauthorized	The request requires user authentication
404	Not Found	Request does not exist

Table 5.1: Table of selected HTTP status codes

5.1.1.1 Testing Authentication

In order to test the authentication of the developer with our REST API, we wrote a simple endpoint called /test that returns an HTTP status code "200 OK" if the authentication was successful and the corresponding status code if it is unsuccessful.

The following are the tests that should not authenticate:

- Duplicate Request
- Missing api_key
- Invalid api_key
- Missing nonce
- Missing timestamp
- Missing signature
- Invalid signature

Since all of these tests fail to authenticate, they should all return an HTTP status code "401 Unauthorized".

All further tests assume that the authentication happens successfully.

5.1.1.2 Testing /payment

To test the /payment endpoint, we make a payment request with an amount and a description. For the result, we expect a valid Bitcoin address with the same amount and description that we provided. We uses a function in bitcoin-lib-php library to validate the Bitcoin address.

The following are tests that should fail:

- Wrong Method (GET instead of POST)
- amount_sat smaller or equal to 0
- amount_sat not an integer
- Missing description
- Description too long

These tests should all return an HTTP status code "400 Bad Request".

5.1.1.3 Testing /payment/{TRANSACTION_ID}

To test this method, we need to call it with a valid transaction ID that corresponds to the api_key that created the transaction. If the ID does not correspond with the api_key, the test should return an HTTP status code "401 Unauthorized".

5.1.1.4 Testing /payment/{PUBLIC_ADDRESS}

Testing this method is very similiar to the /payment/{TRANSACTION_ID}. The public address must correspond with the api_key that created the transaction.

The following tests should fail:

- Wrong Public Address (401 Unauthorized)
- Invalid Public Address (400 Bad Request)

5.1.1.5 Testing /balance

This method only has no arguments. Thus, if the authentication succeeds, it should return an HTTP status code "200 OK" and the balance.

5.1.1.6 Testing /payout

This method will always return an HTTP status code "200 OK" with the result in the JSON response.

The following tests should fail:

- No Address Given
- Invalid Address

5.1.1.7 Testing Invalid API Method

All these tests should return an HTTP status code "404 Not Found":

- No Method Given
- Testing "/"
- Testing /asdfasdf
- Testing /asdfasdf/asdfasdf
- Testing /balance/asdfasdf
- Testing /payment/asdfasdf

5.2 Qualitative

Chapter 6

Conclusion

6.1 The Conclusion

Appendices

Appendix A

No appendix yet

Bibliography

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Consulted*, pp. 1–9, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] PayPal, "PayPal," 2015. [Online]. Available: https://www.paypal.com/za/webapps/mpp/about

[3] Oracle.com, "What Are RESTful Web Services?" [Online]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html

[4] C. o. G. LLC, "Introduction to ChoiceScript." [Online]. Available: https://www.choiceofgames.com/make-your-own-games/choicescript-intro/