

Application for cryptocurrency wallets management

(Aplikacja do zarządzania portfelami kryptowalut)

Sławomir Górawski

Praca inżynierska

Promotor: dr Leszek Grocholski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

23 czerwca 2019

Abstract

Cryptocurrencies are a practical application of the blockchain technology. Crucial operations conducted by owners of such digital assets include: creating wallets, tracking the amount of funds and making transactions. This thesis aims to deliver a method of carrying them out on different blockchains. The result is a program written in Python, integrated with Bitcoin and Ethereum, allowing users to perform financial operations on them in an unified way.

Kryptowaluty są praktycznym zastosowaniem technologii blockchain. Do głównych operacji przeprowadzanych przez posiadaczy tych cyfrowych dóbr, należą: zakładanie portfeli, śledzenie ilości środków oraz wykonywanie transakcji. Celem tej pracy jest dostarczenie metody ich realizacji na różnych blockchainach. Efektem końcowym jest napisany w Pythonie program, zintegrowany z Bitcoinem i Ethereum, umożliwiający użytkownikom wykonywanie na nich operacji finansowych w ujednolicony sposób.

Contents

1	Introduction	7
2	Description and analysis of the subject	9
2.1	Definition of the goal	9
2.2	Technology overview	10
2.2.1	Similarities	10
2.2.2	Differences	11
2.2.3	Conclusions	11
3	Comparison with other known solutions	13
3.1	Copay	13
3.2	My Ether Wallet	14
4	Description of applied solutions	17
4.1	Library	17
4.1.1	Creating wallets	18
4.1.2	Checking balances	18
4.1.3	Making transfers	19
4.1.4	Networking	19
4.2	Web application	20
5	For users	21
5.1	Installation	21
5.2	User guide	22
5.2.1	Accessing wallets	22

5.2.2	Creating or recovering wallets	23
5.2.3	Viewing wallets	24
5.2.4	Making transfers	25
6	For programmers	27
6.1	Library	27
6.1.1	Structure	27
6.1.2	Tools	28
6.2	Web Application	28
6.2.1	Structure	28
6.2.2	Tools	29
7	Summary	31
	Bibliography	33

Chapter 1

Introduction

Cryptocurrency is a digital asset designed to work as a medium of exchange, history of which can be traced back to 2008, when Satoshi Nakamoto's Bitcoin white paper was published [1]. Since that time the blockchain technology Bitcoin was based on has gained a significant popularity, giving birth to a huge number of different projects. The applications of it are various, this thesis will be focused on the most widespread one – cryptocurrency.

Since 2008 the technology has significantly evolved, and with three distinct blockchain generations being distinguished [2] and over 2300 various cryptocurrencies available over the internet [3], it may seem challenging to comprehend. With such a vast array of different digital assets for the users to exchange, wrapping one's head around them is likely a non-trivial task.

This project aims to provide a partial solution by creating an application for cryptocurrency wallets management – an unified interface for performing common financial operations across different blockchains. Chosen for this purpose were Bitcoin and Ethereum [4], belonging respectively to first and second generation, as popular choices among users [5] which together should provide a comprehensive overview of the technology as a whole.

Chapter 2

Description and analysis of the subject

To decide on the best approach to the challenge, first an analysis of desired properties and features of a solution shall be conducted. After that the technologies that the project will have to work with will be looked into, with the goal to determine how are they similar, and at the same time, how they differ.

2.1 Definition of the goal

To compile a list of features that ought to be demanded from a crypto wallet management application, we can compare it to services used by more traditional internet banking. As other applications of the blockchain technology, for example smart contracts, are out of this thesis' scope, focus is to be on financial management.

The core functionality that the project cannot be considered complete without is to be:

- creating cryptocurrency wallets,
- checking the amount of funds,
- making transfers.

All of the above should function in a similar way from the end user's perspective across all supported cryptocurrencies.

With these implemented an user should have sufficient control over their crypto assets, allowing him to participate in either personal or professional financial operations over the cryptocurrency networks.

Some additional features, that would be convenient to have, however will not be considered essential could be:

- saving wallets' balance history,
- insight into value of the crypto assets (e.g. in US dollars),
- ability to import wallets used previously with other services.

The project should provide all this functionality while remaining stable and portable, prove resistant to predictable user errors and ensure that managed assets' security will not be compromised. The user experience ought to be simple, and the interface should be clean and focused on the most important elements, with function taking precedence over form.

2.2 Technology overview

The project will have to integrate with two blockchain generations: first generation, represented by Bitcoin, the oldest and most known cryptocurrency, and second generation, represented by Ethereum. Although these cryptocurrencies have a lot in common – they both are based on blockchain, after all – they differ even in what they were to be.

Bitcoin was created as a peer-to-peer electronic cash system, aiming to provide an alternative to traditional banking – performed through financial institutions – with a concept of a global, distributed ledger of transactions, secured from being mangled with by proof-of-work in form of using computational power to mine blocks of transactions, thus preventing them from being retroactively changed.

The premise of Ethereum was a bit different, as it was focused more on distributed computation, aiming to deliver what can be thought of as a decentralized computer, with global state made of accounts containing information such as balance, being secured by the blockchain technology. The solution used currently for mining blocks is also proof-of-work, although that is said to change in near future.

2.2.1 Similarities

Both technologies are interacted with through asymmetric cryptography – users are identified by their addresses and authorize operations by signing data with their private keys. In the core essence, a cryptocurrency wallet is all about its private key. A public key is coupled with the private key, and then an address is derived from the public key as an additional layer of security. This is true for both blockchains – even though the format of keys and addresses differs, underlying principle remains the same.

2.2.2 Differences

As was hinted before, Ethereum accounts are stateful by nature, Bitcoin ones, however, are not. This difference is apparent when certain information needs to be recovered from the blockchain. It is often the case when it's saved as part of the Ethereum network's global state and the extraction is straightforward. Bitcoin is not based on a notion of any global state, so in theory crawling an entire blockchain might be required for reconstructing the current situation.

A good example of this is wallet balance – a part of the account state in Ethereum, easy to recover. Bitcoin does not however deal with accounts directly, it deals only with transactions; so this quite basic piece of information might require tracing all transactions somehow associated with a given wallet, possibly many years into the history of Bitcoin.

2.2.3 Conclusions

Seamless integration of the project with both blockchains, with details abstracted away under a shared interface might prove a nontrivial task, as there are quite substantial differences in how these function.

However, the similarities between Bitcoin and Ethereum make the same task seem sensible and worth the effort, as for an end user dealing with cryptocurrencies, both are in their essence just digital assets, so providing an unified way to manage them sounds like an useful idea.

Chapter 3

Comparison with other known solutions

Two applications that provide similar functionality and can be compared with this project are Copay [6] and MyEtherWallet [7]. Following sections will provide a brief overview of their features and how the project tries to improve on them.

3.1 Copay

Copay is a popular application for Bitcoin wallets management. It's an open source client created by BitPay Inc, currently featuring hundreds of contributors. By default, it utilizes Bitcore Wallet Service as its backend for connecting to the network, however it can be configured to integrate with user's custom nodes.

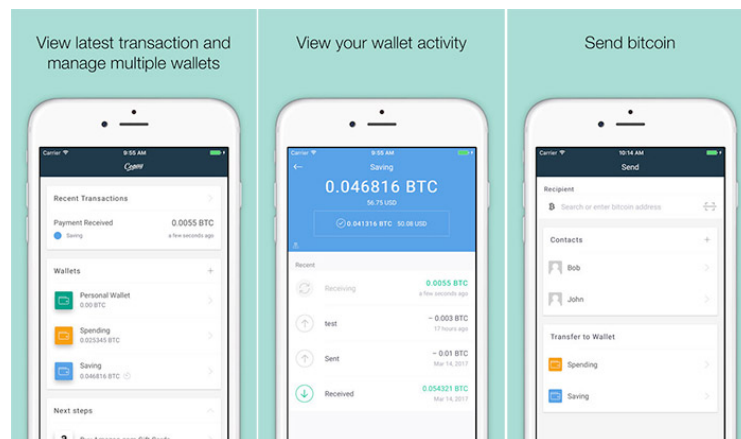


Figure 3.1: Copay interface on a mobile device

The application itself is multi-platform, written in JavaScript [9]. On desktop operating systems, it runs on top of Electron [10]. It allows for creating wallets, viewing balances and making transfers, so all the operations considered essential in this thesis are covered. However the supported currencies currently are Bitcoin and Bitcoin Cash, which is a fork of Bitcoin – so in essence just one type of blockchain. Apparently, the goal here was quality support for few currencies.

The multi-platform nature of the application is quite problematic on its desktop versions. The interface is clearly designed mobile-first and using it with keyboard and mouse is not comfortable. The screens that the user interacts with display very limited amounts of information at once, and accessing some of the less obvious options often requires traversing several views. While this kind of design may work well if used on a mobile device, it does not provide a good user experience for a desktop application.

An interesting fact about Copay is that it has quite recently been a target of a rather sophisticated attack [8], which aimed to steal private keys of users, who possessed more than 100 bitcoins on their wallets. That incident might have undermined users' trust in security of the application.

3.2 My Ether Wallet

My Ether Wallet is a wallet management service for Ethereum and some of the tokens available on its network. The client is a single page application created in JavaScript and available under the address <https://www.myetherwallet.com>. The entire project received a design makeover recently, however its new version does not seem to provide all of the features available in the previous one yet, so the comparison will be done in regard to the older version, which is still available at <https://vintage.myetherwallet.com>.

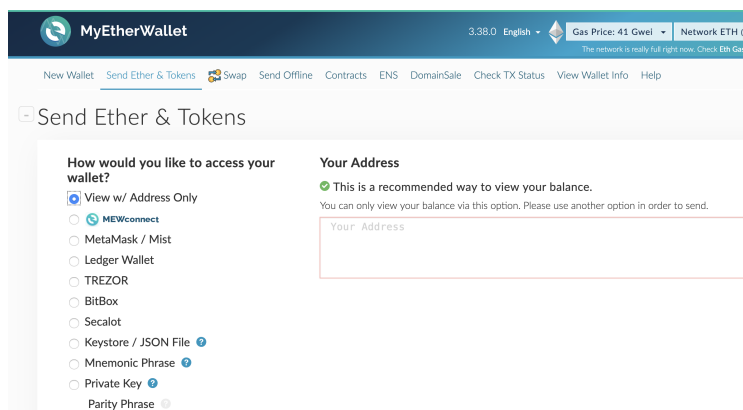


Figure 3.2: My Ether Wallet website

The application is quite feature-rich and allows for connecting with custom nodes and performing all the necessary operations. A major advantage is that everything works out of the box in browser, without the need of installing anything on user's device.

What can be disliked in this application is again the user interface, although for different reasons than Copay's one. While this one looks as if it was designed desktop-first, makes good use of the entire screen and can be comfortably operated with a keyboard and a mouse, it feels cluttered. There are too many options displayed on a single screen, and upon first access an alert with a rather invasive explanation of how the website works is displayed, and to make it not appear on user's device they are forced to read through several screens.

Another major issue with the MyEtherWallet user experience is that upon refreshing a page, the entire session is lost. While it might be a good security choice in context of accessing wallets by their private keys, it provides a rather poor experience while viewing wallet by its address. Not only is the user forced to input it again, they are also shown the intrusive alert described above and have to deal with it as well.

Chapter 4

Description of applied solutions

The entire project consists of two parts:

- a library, which provides a unified interface for connecting to and performing operations on the blockchains,
- a web application using that library, which provides a convenient interface for the users and a couple of extra features.

Such separation makes it easy to potentially change the user interface, e.g. to a command line one, while being able to preserve a convenient way of interacting with the blockchains. It also allows for a better control over the project's dependencies, – where the library isn't cluttered with requirements of any web framework – and makes testing easier. Core functionality of the library can be fully verified without being restricted by the user interface. Application tests, on the other hand, can have the library calls mocked out, enabling them to be performed without access to the cryptocurrency testnets, which makes setting up particular scenarios easier.

4.1 Library

The library is essentially a set of classes with an unified interface for typical wallet management operations, i.e.:

- creating a new wallet or retrieving one,
- checking balance on a given wallet,
- sending money to other wallets of given currency,

which shall be explained in more detail below.

4.1.1 Creating wallets

This operation is performed nearly the same way on either blockchain. First, a private key is either randomly generated, or recovered from a mnemonic phrase. Next, a public key is retrieved from the private key, and then an address is derived from it. The public key is not really relevant to the user, as the address is used in its stead to provide an additional layer of security [11]. Expected format of these pieces of data may differ (e.g. it's hexadecimal in Ethereum), but the underlying principle remains the same. Overall, the private key, the address and a mnemonic phrase allowing to easily recover this wallet later are returned.

A question may arise whether the mnemonic phrase is required, since a private key already provides control over a wallet? Essentially, it is not, and the information stored in it is equivalent, however its format makes it easier for humans to remember, than a key which is a long, seemingly random sequence of characters.

Theoretically, this whole procedure is independent from the blockchain, as a wallet is just a combination of keys and can be generated offline. In principle, that is the case, as it should be in a decentralized network. However, there is a catch – in the case of Bitcoin a wallet's address is imported by a node for indexing. This will be explained in the following subsection.

4.1.2 Checking balances

Retrieving the amount of money available on a given address works a bit differently in the two networks. The Ethereum accounts are stateful by nature, storing that information directly in themselves – all it takes to view it is a single remote procedure call to our node in the network.

In Bitcoin, however, this is not the case – only transactions are stored in the blockchain, balances are not. In order to determine how much money does one have at their disposal, they would have to crawl the chain tracing the flow of money related to their address, possibly many blocks in, even to the very beginning of Bitcoin.

In practice Bitcoin node implementations provide a solution for this, indexing transactions related to given addresses, which makes it possible to retrieve all unspent transactions outputs [12] for an address without crawling the blockchain every time. Each of these UTXOs (Unspent Transaction Outputs) can be used to make a transfer, which means that the balance of a wallet is a sum of their amounts.

4.1.3 Making transfers

The last operation is rather easy in the case of Ethereum, where a transaction is constructed by specifying sender, recipient, amount and the appropriate gas price. The latter serves (in a rather indirect way) as fee for block miners and can be inferred from network activity using a built-in procedure of the node. There is also *nonce*, which is like an account's transaction counter – its appropriate value is just a number of our unconfirmed transactions in the network.

When making a transfer to more than one recipient (let us say there are n), n transactions are created, with *nonce* iteratively incremented for each one. The downside of this simple approach is that the gas price must be provided separately, so we end up paying n miner fees for all of the transactions.

As was described earlier, Bitcoin deals with transfers, not with accounts, and this results in a more complex way of creating a transaction. Some of our unspent transaction outputs must be selected to serve as input to the new transactions, and they must be spent entirely. If their total value is too high, our own address is added as one of the recipients to receive change. After that step we end up with a transaction object with some inputs – worth in total x BTC – and recipients (including us) to receive in total y BTC. The difference between x and y serves as miners fee, like gas price in Ethereum. The good aspect of this procedure is that it is only paid once, no matter the number of recipients.

After creating a transaction object of either currency, it is signed with our wallet's private key and broadcasted to the network, waiting to be included in the blockchain.

4.1.4 Networking

The actual communication with the node of a given cryptocurrency's network is conducted via HTTP [13], using JSON [13] format for remote procedure calls. As it turned out, the JSON-RPCs [15] used by Bitcoin and Ethereum are standardized, sharing the same format for requests and responses and differing only in details. This allowed to encapsulate network communication details of both currencies in a single class.

For each required procedure call, a request is made to a specific node. The response is then checked for whether it is associated to this request (by an embedded ID). Lastly, in case of a success response relevant data is extracted from it and returned, and in case of an error response, the error is handled in an appropriate way.

4.2 Web application

The web application aims to provide a clean web interface for the users. It listens to HTTP requests and returns server-side rendered HTML [16] templates as responses. Users communicate with server using HTML forms [17], and their requests are handled by appropriate functions, which usually use the library's methods to interact with the blockchains.

Aside from allowing users to perform basic operations mentioned in the Library section, the application provides some extra functionality for better user experience [18]. First of all, the balance history for all wallets is saved in a database, allowing the user to see how it changed over time plotted on a graph. Secondly, the application provides insight into actual value of the crypto assets in dollars by integration with the Coinlayer [19] service. The prices are retrieved from their API [20] and cached in the database, to avoid flooding the service with requests. Wallet's balance value is also tracked, saved in the database and presented to the user in form of a graph.

Chapter 5

For users

5.1 Installation

The project's minimum requirement is Python, [21] version 3.7 or higher. For local testing it is recommended to use Docker [22] for running supporting services, such as blockchain nodes and database, along with Docker Compose utility.

After cloning sources it is recommended to create a local Python virtual environment [23], so as not to clutter the global installation with project's dependencies. All packages can be managed using official Python package installer `pip` [24], which comes bundled in with most Python distributions. First, the library must be installed for local use, and after that, the application itself.

```
$ pip install -e lib[dev]
$ pip install -e app[dev]
```

After installation, supporting services should be set up in the background; this can be done with `docker-compose`, i.e.:

```
$ docker-compose up -d
```

At this point, there should be a blank database available for connecting, which needs to be populated with required tables etc.:

```
$ flask db upgrade -d app/migrations
```

What is left is to spin up a web server, the development one bundled with Flask will do (specify port using the flag, default is 5000):

```
$ flask run --port 5000
```

The application can be accessed from any web browser now.

5.2 User guide

5.2.1 Accessing wallets

To view a wallet, click a button in the upper-right corner of the page.



Figure 5.1: Access to wallet

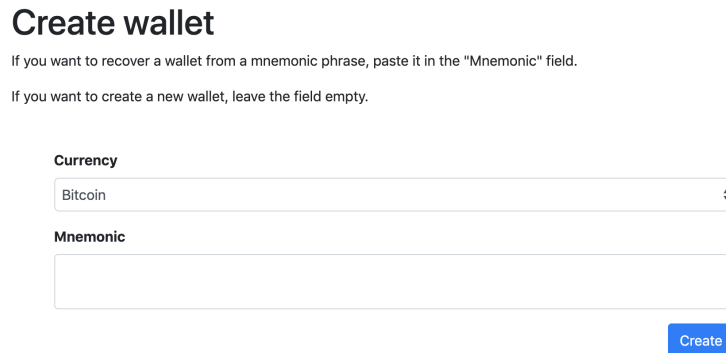
You will be redirected to a page containing a form resembling a sign-in from other web applications. One notable difference is that private key is not required at this point, so any wallet with known address can be viewed, not just current user's. This might not seem very intuitive at first, but it is designed in accordance with the very nature of the blockchain, where all such information is public.

View wallet

Figure 5.2: Login-like page

5.2.2 Creating or recovering wallets

If the user doesn't have any wallet to manage yet, they are shown an option to create a new one, or recover one from a mnemonic phrase. After clicking the button, they are redirected to another page with a simple form:



Create wallet

If you want to recover a wallet from a mnemonic phrase, paste it in the "Mnemonic" field.

If you want to create a new wallet, leave the field empty.

Currency

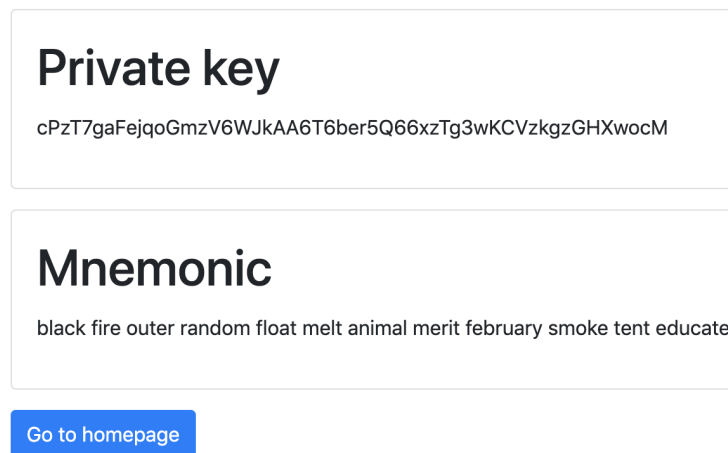
Bitcoin

Mnemonic

Create

Figure 5.3: Creating a new wallet

After a successful creation of a wallet, the user is presented with its private key and mnemonic phrase. It's crucial to save them at this point, because this is the last time such information will be available (for security reasons it is not saved anywhere in the system).



Private key

cPzT7gaFejqoGmzV6WJkAA6T6ber5Q66xzTg3wKCVzkgzGHXwocM

Mnemonic

black fire outer random float melt animal merit february smoke tent educate

Go to homepage

Figure 5.4: Wallet successfully created

After saving secret information, the user has an option to be redirected to homepage in context of the newly created wallet, as if they accessed it by its address.

5.2.3 Viewing wallets

When using site in context of a selected address, the user is presented with a balance over time chart and a short summary of the funds available, along with an option to make a transfer.

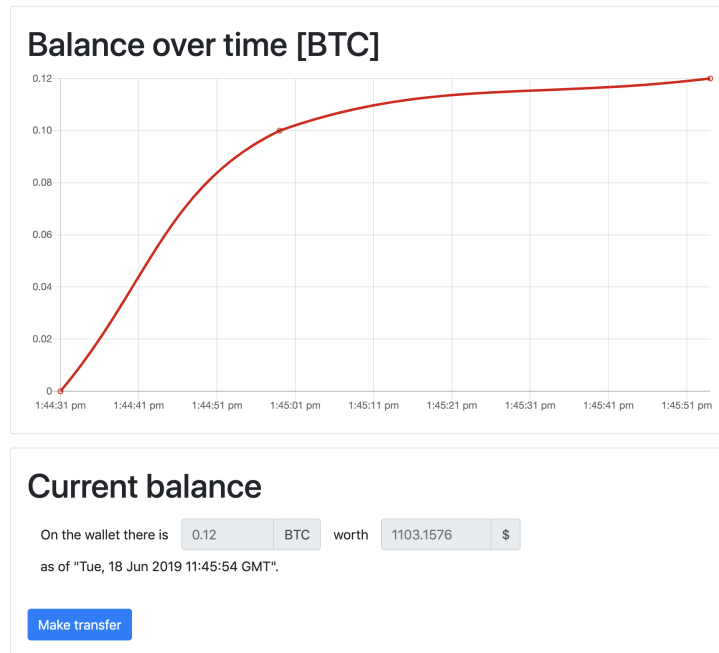


Figure 5.5: Homepage of a selected address

Below, on the same page, there is a similar chart presenting value of the user's assets in dollars over time, as it doesn't have to be correlated with the balance itself, and a table providing a more detailed insight into various useful data.

During this entire time, on all pages that current user can interact with, they are presented with the wallet context (currency and address) on the navigation bar. If they wish to exit it, the appropriate button is available in the upper-right corner, which is slightly similar to a logout on most websites.

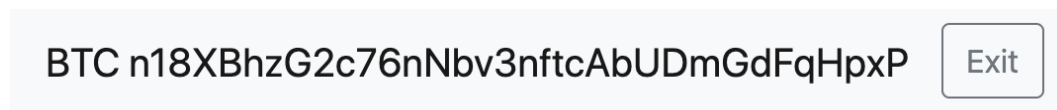
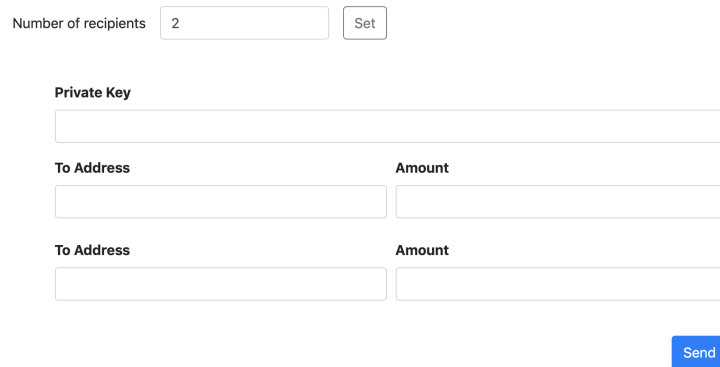


Figure 5.6: Wallet context

5.2.4 Making transfers

In order to make a transfer, the user ought to click the button described above. After that action, they are redirected to transfers page with a form to fill. Desired number of recipients should be set accordingly, and user wallet's private key must be provided for authorization. It will be used only for this one transfer, and will not be saved anywhere for security. Appropriate amounts to send to their respective addresses are to be filled in below.



The form is titled "Transfer" and contains the following elements:

- A label "Number of recipients" followed by a text input field containing the value "2" and a "Set" button.
- A label "Private Key" followed by a large text input field.
- A table with two columns: "To Address" and "Amount".
- Two rows of input fields for the table, each with a "To Address" field and an "Amount" field.
- A blue "Send" button at the bottom right.

Figure 5.7: Transfer

After a successful transfer, the user will be redirected to the homepage and provided with transaction IDs saved in the database (only one in Bitcoin, or n in case of Ethereum, n being the number of recipients). These can be used to check transaction statuses (e.g. numbers of confirmations) using online blockchain explorers.

Chapter 6

For programmers

The primary programming language used in the project is Python. It was chosen because of its conciseness, a vast array of open source third-party tools available, such as libraries and frameworks, and support for multiple programming paradigms, such as imperative and object-oriented, with functional and declarative elements.

The source code was kept in the Git [25] version control system and written using PyCharm [26] tool by JetBrains on an educational license, which can be too heavy for a small project, but provides excellent support for code navigation and refactoring in larger ones.

6.1 Library

6.1.1 Structure

The library consists of two major classes, named **Bitcoin** and **Ethereum**. They provide a unified interface for performing operations on respective blockchains by exposing the following methods:

- `create_wallet`,
- `get_balance`,
- `send_money`,

which should be self-descriptive.

There is also a class that provides an abstraction over the networking details, which are pretty much the same in both blockchains, as was described before. The class is called **RpcProxy** and upon being instantiated with an URL to an appropriate network node, is passed to **Bitcoin** or **Ethereum** by dependency injection [27]. This allows for significant code reuse between respective currencies' classes.

6.1.2 Tools

The following third-party dependencies were used to provide the library’s functionality:

- **pywallet** – a library used for creating both Bitcoin and Ethereum wallets, chosen because of its universality,
- **eth-account** – a collection of Ethereum-specific utilities, used mostly for signing transactions with private keys (in Bitcoin this was done by the node itself),
- **requests** – a tool used for making HTTP requests, offering a minimalistic interface better than its equivalents from the standard library; one of the most widely used Python libraries and a de facto standard in the industry.

For development, a unit testing framework **pytest** was also used, providing a clean, declarative style interface superior to the standard library. This one is not required for the library to work, though.

To test the integration with actual cryptocurrency networks, the nodes were run as Docker containers using the Docker Compose utility, exposing their ports for the manual and unit tests to use. For Bitcoin, the image used was an official implementation Bitcoin Core [28] available as a Docker image **ruimarinho/bitcoin-core**. Ethereum was tested against Ganache [29], a tool designed specifically for such use cases, available as a Docker image **trufflesuite/ganache-cli**.

The entire library is completely independent of the web application; it is packaged with **setuptools** to be used as if it were an external dependency.

6.2 Web Application

6.2.1 Structure

The application consists of:

- request handling functions,
- a module exposing the library classes dynamically by currency,
- a module for integration with the Coinlayer API,
- an ORM mapping database relations to Python objects,
- templates for response rendering,
- database migrations stored as Python modules.

6.2.2 Tools

The application was created using Flask [30], a popular Python web framework. It is a modular tool which does not enforce any opinionated project structure, which was desirable due to this project not being a typical CRUD backend. Said popularity was another major advantage, as there is now a vast array of guides and answered questions on the web.

The ORM and toolkit used for database operations is SQLAlchemy [31], also one of the most popular Python tools of its category. It allows for easy query construction and integrates seamlessly with Flask thanks to the `flask-sqlalchemy` plugin. Another plugin was `flask-migrate`, used to generate and apply database migrations with Alembic [32].

Web forms were created with the help of WTForms [33] and its respective plugin `flask-wtf`. This tool makes generating HTML easier, and it comes with built-in input validation.

The `requests` library described in 6.1.2 was used to fetch cryptocurrency prices from the Coinlayer API.

PostgreSQL [34] was chosen as a database for the project, due to popularity and its useful features. For development, it was set up as a Docker container along with testnet nodes and ran with Docker Compose.

HTML responses were generated with Jinja2 templating engine, which is bundled with Flask as a sensible default option. For styling and responsiveness Bootstrap [35] CSS [36] framework was used, chosen because of popularity, ease of use and good looking design.

Last but not least, charts for viewing balances and values over time were generated in JavaScript, with the help of the Chart.js [37] library.

Chapter 7

Summary

In the last chapter we shall consider some possible uses of the project and its potential for further development.

First and foremost, the application could be used by individual users who invest in cryptocurrencies for management of their personal wallets. The unified interface would provide a comprehensive overview of digital assets across different blockchains. To get there, the project would have to be deployed on a publicly available domain, along with a backend of several network nodes for different blockchains. Proper configuration of such service is likely a non-trivial task, but it has already been done by similar solutions, e.g. the ones discussed in chapter 3.

Some other improvements in this scenario could be:

- Support for more cryptocurrencies (many of them are based on Bitcoin or Ethereum, so it should not require any major changes).
- A more personalizable user interface, for instance providing an overview of all wallets of a given user on one screen (this would likely require storing user information in the database).
- Exchanging cryptocurrencies (likely requiring some legal approval).

As can be seen above, adapting the project for common use would require some work. There is however another scenario, for which the project could be suitable in its current form without a need for significant modifications – usage by development teams working in blockchain-related projects, such as trading platforms for instance.

Development and testing of such projects often requires dealing with cryptocurrency wallets and making real transactions. A simple, self-hosted tool such as this one could be quite helpful in the process. Managing all desired cryptocurrencies through a unified interface, with no unnecessary clutter (like user accounts) would likely provide a convenient way of collaboration between members of a team.

Additional features that would make the project better for such use are for example:

- Support for more cryptocurrencies (same as in first scenario).
- A more detailed insight into the blockchains, e.g. tracing desired transactions.

To conclude, I believe that the project shows potential for further evolution, although development teams might be a better target group for it than casual users. Either way, there are many factors to consider and many features possibly worth adding. A core foundation that can be built upon is there, and fulfills the requirements set at the beginning of the thesis.

Working on this project has been a valuable learning experience, allowing me to familiarize myself with technicalities of the blockchain technology and practise designing functional user interfaces. The ever-changing world of cryptocurrencies had seemed difficult to grasp at times, however in the end proved to be both a remarkable application of technology and a promising path for further study.

Bibliography

- [1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System.
<https://bitcoin.org/bitcoin.pdf>
- [2] Stephan Cummings. The Four Blockchain Generations.
<https://medium.com/altcoin-magazine/the-four-blockchain-generations-5627ef666f3b>
- [3] Cryptocurrency List by Coinlore.
https://www.coinlore.com/all_coins
- [4] Vitalik Buterin. A Next-Generation Smart Contract and Decentralized Application Platform.
<https://github.com/ethereum/wiki/wiki/White-Paper>
- [5] CoinMarketCap cryptocurrency statistics.
<https://coinmarketcap.com/all/views/all>
- [6] Copay: A secure Bitcoin and Bitcoin Cash wallet platform.
<https://github.com/bitpay/copay>
- [7] My Ether Wallet: Client-side interface helping you interact with the Ethereum blockchain.
<https://github.com/MyEtherWallet/MyEtherWallet>
- [8] John E. Dunn. JavaScript library used for sneak attack on Copay Bitcoin wallet.
<https://nakedsecurity.sophos.com/2018/11/28/javascript-library-used-for-sneak-attack-on-copay-bitcoin-wallet/>
- [9] Javascript, Mozilla Developer Network web docs.
<https://developer.mozilla.org/docs/Web/JavaScript>
- [10] Electron.js project website.
<https://electronjs.org>
- [11] Technical background of version 1 Bitcoin addresses.
https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses

- [12] Sudhir Khatwani. What Are Unspent Transaction Outputs (UTXOs)?
<https://coinsutra.com/unspent-transaction-outputs-utxos>
- [13] P. J. Leach, T. Berners-Lee, J. C. Mogul, L. Masinter, R. T. Fielding, J. Gettys.
Hypertext Transfer Protocol - HTTP/1.1, IETF RFC 2616.
<https://tools.ietf.org/html/rfc2616>
- [14] T. Bray. The JavaScript Object Notation (JSON) Data Interchange Format,
IETF RFC 7159.
<https://tools.ietf.org/html/rfc7159>
- [15] JSON-RPC 1.0 Specification.
https://www.jsonrpc.org/specification_v1
- [16] HTML5. A vocabulary and associated APIs for HTML and XHTML.
<https://www.w3.org/TR/2010/WD-html5-20100624>
- [17] HTML 5.2 Forms. W3C Recommendation, 14 December 2017.
<https://www.w3.org/TR/html52/sec-forms.html>
- [18] User experience definitions.
<http://www.allaboutux.org/ux-definitions>
- [19] Coinlayer API - Free, Real-time Crypto Rates API.
<https://coinlayer.com>
- [20] David Emery. Standards, APIs, Interfaces and Bindings.
<https://web.archive.org/web/20150116081559/http://www.acm.org/tsc/apis.html>
- [21] The official home of the Python Programming Language.
<https://www.python.org>
- [22] Docker: Enterprise Container Platform.
<https://www.docker.com>
- [23] Virtual Environments and Packages — Python 3.7.3 documentation.
<https://docs.python.org/3/tutorial/venv.html>
- [24] Pip: The PyPA recommended tool for installing Python packages.
<https://pypi.org/project/pip>
- [25] Git: A free and open source distributed version control system.
<https://git-scm.com>
- [26] PyCharm: the Python IDE for Professional Developers by JetBrains.
<https://www.jetbrains.com/pycharm>
- [27] James Shore. Dependency Injection Demystified.
<https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html>

- [28] Bitcoin Core official website.
<https://bitcoincore.org>
- [29] Ganache CLI: Fast Ethereum RPC client for testing and development.
<https://github.com/trufflesuite/ganache-cli>
- [30] Flask: A Python Microframework.
<http://flask.pocoo.org>
- [31] SQLAlchemy - The Database Toolkit for Python.
<https://www.sqlalchemy.org>
- [32] Alembic: A lightweight database migration tool.
<https://alembic.sqlalchemy.org/en/latest>
- [33] WTForms Documentation.
<https://wtforms.readthedocs.io/en/stable>
- [34] PostgreSQL: The World's Most Advanced Open Source Relational Database.
<https://www.postgresql.org>
- [35] Bootstrap: The most popular HTML, CSS, and JS library in the world.
<https://getbootstrap.com>
- [36] All CSS specifications.
<https://www.w3.org/Style/CSS/specs.html>
- [37] Chart.js: Open source HTML5 Charts for your website.
<https://www.chartjs.org>