Use Cases and Design of Web Applications in Decentralized Finance

Johannes Hüsers

BACHELORARBEIT

eingereicht am
Fachhochschul-Bachelorstudiengang

Software Engineering

in Hagenberg

im Februar 2021

Advisor:

DI Martin Harrer

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere. This printed copy is identical to the submitted electronic version.

Hagenberg, February 1, 2021

Johannes Hüsers

Abstract

—0.5 to 1 page—

Kurzfassung

—0.5 to 1 page—

Contents

Chapter 1

Introduction

1.1 Motivation

—1.25 pages—

1.2 Goals

—0.75 pages—

1.3 Structure of the Thesis

—0.5 pages—

Chapter 2

Fundamentals

2.1 Cryptography

2.1.1 Public Key Infrastructures

— 0.5 pages —

2.1.2 Elliptic Curve Cryptography

— 0.5 pages —

2.1.3 Hashing Functions

— 0.5 pages —

2.2 The Ethereum Blockchain

2.2.1 Blockchain Fundamentals

— 0.5 pages —

2.2.2 Clients

— 0.5 pages —

2.2.3 Wallets

— 0.5 pages —

2.2.4 Transactions

— 0.5 pages —

2.3 Smart Contracts

– 1 page —

2.4 Decentralized Finance

— 1 pages —

2.4.1 State of the Art

current applications, relevance on the market, technologies, ...

Chapter 3

Use Cases of Decentralized Finance

This chapter aims to introduce the five most relevant use cases of Decentralized Finance. Each use case is built on top of the previous one and rises in complexity. For example storing value in digital systems is quite easy nowadays but moving real physical assets such as gold or real estate into a blockchain is still pretty demanding. Note, that each type of financial service is not specific to Decentralized Finance, which means that they are applicable to every financial environment.

3.1 Classification of Financial Services

—0.25 pages—

3.2 Store of Value

No matter of the financial product or service, they all need one essential thing. In fact, our entire economic system depends on it: a currency, or in simpler terms, something to store value. Storing value is a fundamental trait of money alongside with being exchangeable and having a unit of account. The last two characteristics are comparatively easy to accomplish in the online world. Storing value in a digitalized way without having a central trusted authority, however, was a similar challenge as the people had to face in the early days when they started to switch from bartering to a real currency. A common consensus needs to be created where everyone can easily verify that a specific piece of money is authentic and wasn't created illegally. Even further, each person that uses this money needs a proof that it is storing real value. This wasn't possible until Satoshi Nakamoto, an anonymous person or group, published a specification [6] in 2008, on how digital money could be implemented. The first cryptocurrency was founded: Bitcoin.

3.2.1 Proof of Work

Bitcoin uses a mechanism called proof-of-work in order to give each block its value. This utilizes a cost-function which is designed to be easily verifiable but quite expensive to compute [3]. In order to create a new block, a value needs to be found that matches with the correct number of zero bits of the block hash. Once the correct solution has been found, the new block represents real value, because operating this computationally intensive task on a CPU costs a lot of electricity. The integrity of this block is guaranteed as well, because in order to change the history of the blockchain, each changed block needs to be re-computed. The longest chain of blocks on the network is accepted as the "truth", so if dishonest people want to cheat and change the history of the blockchain, they would need to create the longest chain of blocks which can be only achieved by having more than 50% of the computing power of the whole network. While these so-called 51% attacks are a threat on blockchains with a smaller number of network members, it is very unlikely to happen on a well established network like the Bitcoin blockchain [8]. The concept of proof-of-work in order to achive consensus in a decentralized manner became very popular and can be used in other areas of application too, such as elections, lotteries, asset registries, digital notarization and more [1].

3.2.2 Proof of Stake

A different approach to storing digital value is proof-of-stake. Instead of making the participants of the network solve computational expensive problems, the consensus is created by proving that someone owns specific funds. A blockchain that uses a proof-of-stake algorithm to achieve consensus has a set of validators. These validators are running a master node which gives them the opportunity to vote. In order to do so, the validators need to prove that they own a specific amount of funds. This is done by sending a special transaction, which locks away their currencies for a specific time. If the validators are honest and their vote corresponds to the result of the majority, they will get their funds back and an additional reward proportional to their deposited stake. If their vote gets rejected, the dishonest validator risks loosing his money [2].

3.2.3 Switching the Consensus Algorithm

At the time of writing this in December 2020, the Ethereum blockchain uses a similar proof-of-work algorithm like Bitcoin, called *Ethash*. However, Ethereum is about to switch to a proof-of-stake algorithm called *Casper* in the near future [22]. Algorithms that are based on proof-of-stake have the big advantage of less energy consumption because they are not focused on computational intensive tasks. The Bitcoin network, for example, has an annual electrical energy consumption of approximately 72.18 terawatt hours which is comparable to the consumption of Austria [21]. But proof-of-stake comes also with a caveat. Implementing incentives on a voting system which are based on the amount each validator is willing to stake means that the rich validators get richer and the poor validators stay poor.

The process of changing a consensus algorithm of a blockchain is not easy to accomplish. Because the network is decentralized, there is no single entity which can force all members to change the algorithm from proof-of-work to proof-of-stake. Just letting the people choose what they prefer to use is also not a good idea, because it is very unlikely that all agree on one type of algorithm. That would probably trigger a hard fork of the blockchain resulting in two separate networks, similar to what happened to Bitcoin in August 2017.

Bitcoin's network is by design very slow in verifying transactions. A block which is mined approximately every 10 minutes has a size of about 1 MB. Due to the small block size, Bitcoin is only capable of processing 7 transactions per second. Some people wished for a Bitcoin network which is more suitable for day to day payments. That is why *Bitcoin Cash* was created, emerged out of a hard fork of the Bitcoin network [20] [5]. In order to prevent a hard fork on Ethereum due to the transition from proof-of-work to proof-of-stake, the Ethereum protocol [10] has a built in mechanism called *difficulty bomb* which makes it more difficult over time to be profitable with mining[1] by increasing the size of the problem to solve. This ensures a smooth transition of all members to switch to proof-of-stake.

For the majority of users on the blockchain which are neither miners nor validators it does not matter which consensus algorithm is being used. In fact, it is not even important when developing decentralized applications based on smart contracts like it is done in chapter 4. The only thing that makes an impact is, that the blockchain is able to store real value on the network by reaching consensus by its members.

3.3 Payments

3.3.1 Bitcoin

—0.5 pages—

3.3.2 Stable Coins

—0.5 pages—

3.3.3 Dynamic Supply Protocols

—0.5 pages—

3.3.4 Stability Protocols

—0.5 pages—

3.4 Lending & Borrowing

—1.5 pages—

3.5 Exchanging

3.5.1 Characteristics of Decentralized Exchanges

With the arise of many new fintech enterprises in the last few years, online exchanges started to grew in importance on the market as well. While they all were very centralized in the first place, Decentralized Finance made it possible to establish new ways of exchanging money. Although each online exchange may behave very different, they

[1] used synonymously to proof-of-work

can be all categorized into three big types when it comes to their structure: Centralized, decentralized and non-custodial exchanges. Centralized exchanges (CEXes) are the traditional approach where all the power is centralized to one specific organization. Decentralized exchanges (DEXes), however, are the complete opposite where there is no single entity in control and decisions are made completely based on Smart Contracts. Non-custodial exchanges are somewhere inbetween but are often confused with decentralized exchanges. It is important to note, that none of the types are superior in comparison to the others. Each type of exchange has its own benefits and risks. In order to assess whether an exchange is suitable for a specific use case, it is crucial to know how to classify it.

Probably the most apparent indicator is the ownership of the private keys. On CEXes, the assets are coffered for the user. If someone buys Bitcoin on a centralized exchange, private keys are never an issue. On non-custodial and decentralized exchanges you have to manage your private keys by yourself. While this is usually a good thing, because you don't have to trust someone else[2], it comes also with the risk of losing all your assets if you forget your private key, since nobody can restore it for you.

Owning your private keys is a good indicator but by far not the only trait a decentralized exchange needs to have. When looking at the technical infrastructure of the exchange, many things could be structured in a centralized manner. While the code of a centralized exchange is usually proprietary, a DEX needs to have code that is licensed exclusively with an open-source licence, making it possible for anyone to fork the project in case the service is no longer available. That is also highly related to emergencies. How does an exchange react to bad or unexpected events such as hacking attacks? CEXes have the option to put their service temporarly offline until the issue is resolved, in order to save the funds of the users. On decentralized exchanges, there is no such thing and assets may be lost forever.

Another crucial topic to consider is censorship and geo blocking. If the service is not available in the whole world, it is not a DEX. Centralized exchanges always decide which coins are listed on the exchange, which is usually a pay to play model. Decentralized exchanges don't have such regulation. Anyone can add a new pair of coins to swap on a DEX, which often leads to a tremendously high amount of unknown and irrelevant listed coins.

The last key indicator on how to categorize an online exchange is the way how the order matching and the settlement works. Centralized exchanges use the same algorithm as stock exchanges, which utilizes an order book. Buy and sell orders are listed in a centralized ledger and the settlement happens when two orders match. Non-custodial exchanges usually use price oracles which try to get price information [11, p. 47f.]. Truly decentralized exchanges even go a step further and solve this problem without a third party price oracle. A DEX uses a concept that relies on Liquidity Pools, which will be discussed in detail in the next section.

3.5.2 Liquidity Pools

Every decentralized exchange needs liquidity. Because there is no order book, assets have to be already on the exchange before the user wants to swap two coins. In order

[2] often referred to as "not your keys, not your coins"

to achieve that, the DEX depends on Liquidity Providers, which add their funds to the exchange. Liquidity Providers get rewarded[3] by the DEX because nobody provides liquidity for free. The rewards can be taken from the exchange fees, which are usually still lower than on CEXes. For example, Uniswap, one of the biggest decentralized exchanges rewards each Liquidity Provider with 0.3% of each transaction taking place on that Liquidity Pool proportional to their share of the pool [24]. If there is a yearly transaction volume on the pair Ether/Tether USD of $17.7 billion and the Liquidity Provider owns a 0.1% share of this pool, he will be rewarded with $53.100 per year. The share is usually expressed through Pool Tokens, which can be traded as well.

Liquidity Providers can only provide liquidity in pairs. If someone wants to provide liquidity to the WBTC-ETH pair, which is Wrapped Bitcoin[4] and Ether, he needs to add the same value both in WBTC and ETH. People who want to swap their WBTC to ETH add only one asset of that pair and receive the other one, which creates an imbalance in that specific pool. The total value stays the same but the proportion of the assets change. In this example, there is more Ether compared to Wrapped Bitcoin after the swap. This makes it very attractive for the market to swap ETH back to WBTC, because the user will get proportional more WBTC for less ETH[5]. In high-volume markets this imbalance will be exploited very soon and the original proportion is restored.

If a Liquidity Pool of a specific trading pair is very small, there is a risk of clearing the entire pool by a single transaction, which would give away the associated asset almost for free in the next transaction. In order to prevent that, a user never gets the entire value in the swapped asset. The difference between the expected and the actual value is called slippage. That might seem annoying in the first place, but it is a crucial instrument for the exchange to work properly. This happens on exchanges with an order book too, because it is very unlikely that both a buy and a sell order have the exact same price. The deeper[6] the order book or the Liquidity Pool, the lower the percentage of the price slippage. If someone makes a high volume order on a small order book or Liquidity Pool, he will experience high slippage because his order wipes out a lot of opposite orders on a CEX and uses a high pool percentage on a DEX [23].

3.5.3 Risks

While providing liquidity to an exchange pool might look very profitable at first, there are also certain risks the Liquidity Provider has to deal with. Smart Contract risk is something which you are automatically exposed to when trusting software instead of people. Because software is still written by people and people make mistakes, there is no guarantee that the Smart Contract on a DEX works completely the way it should be. On a turing complete network such as Ethereum with programming languages like Solidity the risk is even higher because you can literally do anything in a Smart Contract. Because a Smart Contract is deployed only once, it is impossible to fix errors once it is public on a blockchain. This risk is far less present on networks which are not turing

[3] known as Liquidity Mining
[4] Bitcoin as a token on the Ethereum network
[5] this phenomenon is called arbitrage and applies to all efficient markets
[6] in terms of more orders and more liquidity

complete such as Bitcoin.

Even if the Smart Contract risk is eliminated as good as possible by writing a lot of tests, there is still the project risk, that the developers might deliberately implement errors and backdoors into their code in order to disburse liquidity to themselves. This is often the case when the project has poor auditing and if the founders are not known to the public.

Impermanent Loss is the last and most present risk liquidity providers are exposed to. The explanation of a Liquidity Pool in the last section is missing an important factor. Even if there is no one trading an asset pair, an imbalance can still occur because each asset is volatile by itself unless the asset pair consists of two stable coins pegged to the same fiat currency. Impermanent Loss always exists if one of the two assets has large price fluctuations in a short amount of time. And while it is usually good to diversify an investment, Impermanent Loss has an even greater impact on non-correlating assets, because the price difference can be much higher. The only way to reduce the risk of Impermanent Loss is to have a larger time horizon where you can choose a good time to cash out the liquidity. Unfortunately there is no such thing as Impermanent Win because it is always a loss once one of the two assets fluctuates. The loss is expressed through the loss in the Liquidity Pool compared to the price gains by holding each asset separately.

3.6 Investing

3.6.1 Tokens

—0.5 pages—

3.6.2 Derivatives

—0.5 pages—

3.6.3 Synthetic Assets

—0.5 pages—

Chapter 4

Design and Architecture

This chapter builds on top of the concepts and use cases of the previous chapters and lays out the architecture and design of a typical decentralized application by using a more practical approach. After a short introduction into the used technologies there will be a detailed view on each important part of a modern decentralized web application. To keep this chapter more concise, only selected code snippets will be shown to describe certain aspects. However, the entire code base of the working application can be found on Github [13].

4.1 Technology Stack

4.1.1 Solidity

The main components of decentralized web applications are Smart Contracts. Smart Contracts can be written in a few different programming languages such as LLL, Serpent, Solidity, Vyper and Bamboo. Solidity is by far the most popular and currently the de-facto standard in writing Smart Contracts [2]. Solidity is statically typed, has a JavaScript-like syntax and supports various popular programming concepts such as libraries and inheritance [7]. Because there are no classes in Solidity, it is considered not as an object-oriented but as a contract-oriented language [17].

4.1.2 Truffle

Truffle is a development environment which helps to develop decentralized applications on the Ethereum Virtual Machine [18]. It handles all the difficult parts starting from Smart Contract compilation, testing, linking, deployment and continuous integration. When developing more than just some simple Smart Contracts, Truffle is an important tool to save a lot of time.

4.1.3 Ganache

Because testing a decentralized application on the main Ethereum network would not be a good idea (and is expensive too), some sort of testing blockchain is needed. Ganache is a local blockchain for developers which provides everything to test a decentralized applications on the local machine [14]. It offers accounts with fake ether and works really well with Truffle, which is built by the same developers.

4.1.4 Web3

In order to link Smart Contracts to a graphical user interface such as a web client application, web3.js comes into play. It is a JavaScript library and provides all the necessary APIs in order to connect a blockchain backend to a web frontend [19]. It also manages connections to specific wallet providers such as MetaMask.

4.1.5 Mocha & Chai

Testing plays in blockchain development an even bigger role than in software development in general, because deployed Smart Contracts are immutable and bugs literally cost money. A flawed function with causes an endless loop whould burn the entire gas that was available for that function call without doing anything useful. Mocha is a JavaScript testing framework which makes it possible to test the functionality of each Smart Contract in a structured and automated way [15]. Together with the assertion library Chai [12] and the integration into Truffle, Mocha releases its full potential as a fully fledged blockchain testing environment.

4.1.6 React

What would be the best Smart Contract without a graphical user interfaces which gives people the opportunity to interact with the blockchain in an easy and intuitive way. React is a JavaScript library for building user interfaces for the web and by far the most popular web framework out there [16]. React is super fast because it uses a virtual DOM and provides the user a modern and intuitive experience.

4.2 The ERC20 Token Standard

The ERC20 standard makes it possible to build standardized tokens on the Ethereum protocol which can be integrated into the existing ecosystem [9]. This is achieved through a common interface each token needs to implement. The standard was initially introduced in 2015 as an Ethereum Request for Comments (ERC). It received its name due to the automatic issue assignment on Github, which had the issue number 20 [2] [4].

4.2.1 Example Implementation

The function `totalSupply` returns the maximum amount available of this token in *wei*, the smallest unit possible. The supply can be either static or dynamic. In this case, it is static assigned through the constructor when the smart Contract is created. The return type is a 256 bit unsigned integer which is the most common data type in Solidity.

```
uint256 private _totalSupply;

function totalSupply() public view returns (uint256) {
  return _totalSupply;
}
```

Program 4.1: The SafeMath library.

```
library SafeMath {
  function sub(uint256 _a, uint256 _b) internal pure returns (uint256) {
    assert(_b <= _a);
    return _a - _b;
  }

  function add(uint256 _a, uint256 _b) internal pure returns (uint256) {
    uint256 c = _a + _b;
    assert(c >= _a);
    return c;
  }
}
```

Each token is responsible to keep track of all addresses holding that token. This can be stored using the data type `mapping` which is a key value pair of addresses and token amounts. The function `balanceOf` returns the token balance for a specific address.

```
mapping(address => uint256) private balances;

function balanceOf(address _owner) public view returns (uint256 balance) {
  return balances[_owner];
}
```

The functionality of sending tokens to a specific address is implemented in the `transfer` function. It takes an address and an amount as parameters and returns a boolean if the transaction was successful. The `require` function checks for a specific condition to be true. If this is not the case, the function will be reverted and all the gas goes back to the sender, which can be evaluated in Solidity using `msg.sender`. In order to prevent over- or underflow errors when performing additions and subtractions, a small library `SafeMath` 4.1 is being utilized. If everything worked well, the `Transfer` event will be emitted.

```
function transfer(address _to, uint256 _value) public returns (bool success) {
  require(_value <= balances[msg.sender]);
  balances[msg.sender] = balances[msg.sender].sub(_value);
  balances[_to] = balances[_to].add(_value);
  emit Transfer(msg.sender, _to, _value);
  return true;
}
```

The function `transferFrom` is very similar to `transfer` with the only difference that the caller is not the one who sends the tokens. This is usually not a physical person but rather a Smart Contract of a decentralized application. Tokens can be sent only on behalf of someone else if the sender accepts the amount first. All funds that are already released for transactions are stored in another mapping specifying for each address which addresses are enabled to send what amount.

```
mapping(address => mapping(address => uint256)) private allowed;

function transferFrom(address _from, address _to, uint256 _value) public returns (
    bool success) {
```

```
  require(_value <= balances[_from]);
  require(_value <= allowed[_from][msg.sender]);
  balances[_from] = balances[_from].sub(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  balances[_to] = balances[_to].add(_value);
  emit Transfer(_from, _to, _value);
  return true;
}
```

If the caller is not the sender of a transaction, funds need to be approved first. That is exactly what the function `approve` does. It authorizes the spender to spend a specific amount of the tokens of the address which called the function. Old funds that are already authorized will not be accumulated but overwritten. This makes it possible to revert an approval by calling the `approve` function another time with the value of zero. This function emits the `Approval` event.

```
function approve(address _spender, uint256 _value) public returns (bool success) {
  allowed[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
}
```

Because the data structure holding the allowed funds is usually private, a public interface is needed to find out how much remaining tokens can be spent on behalf of someone else. The function `allowance` takes two parameters and returns the amount of tokens that the spender is authorized to spend from the owner's address.

```
function allowance(address _owner, address _spender) public view returns (uint256
    remaining) {
  return allowed[_owner][_spender];
}
```

The events `Transfer` and `Approval` play an important role in the interface of an ERC20 token. The `Transfer` event is emitted every time when a transaction is about to be executed. The `Approval` event is being called every time some tokens need to be approved. Both events trigger some sort of action in the linked wallet application of the user. For example, the browser wallet MetaMask opens a popup window asking the user for confirmation, once an event is emitted.

```
event Transfer(address indexed _from, address indexed _to, uint256 _value);
event Approval(address indexed _owner, address indexed _spender, uint256 _value);
```

4.3 Lending Implementation

— 1 page —

4.4 Exchange Implementation

— 1 page —

4.5 Staking Implementation

— 1 page —

4.6 Testing Smart Contracts

— 1 page —

4.7 Connecting Smart Contracts to Web Applications

— 1 page —

4.8 Smart Contract Deployment

— 1 page —

Chapter 5

Closing Remarks

5.1 Criticism

—0.5 pages—

5.2 Risks

—0.5 pages—

5.3 Prospective Impact

—2 pages—

Appendix A

Technical Details

Appendix B

Supplementary Materials

List of supplementary data submitted to the degree-granting institution for archival storage (in ZIP format).

B.1 PDF Files

Path: /

thesis.pdf . . . . Master/Bachelor thesis (complete document)

B.2 Code-Files

Path: /media

. . . .

B.3 Reference Files

Path: /online-sources

. . . .

Appendix C

Glossary

References

Literature

[1] Andreas Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain.* 2nd ed. Sebastopol: O'Reilly Media, Inc., 2017 (cit. on p. 5).

[2] Andreas Antonopoulos and Gavin Wood. *Mastering Ethereum: Building Smart Contracts and Dapps.* Sebastopol: O'Reilly Media, Inc., 2018 (cit. on pp. 5, 10, 11).

[3] Adam Back. *Hashcash - A Denial of Service Counter-Measure.* Aug. 2002. URL: http://www.hashcash.org/papers/hashcash.pdf (cit. on p. 4).

[4] Badr Bellaj, Richard Horrocks, and Xun Wu. *Blockchain By Example.* Birmingham: Packt Publishing Ltd., 2018 (cit. on p. 11).

[5] Brenn Hill, Chopra Samanyu, and Valencourt Paul. *Blockchain Quick Reference.* Birmingham: Packt Publishing Ltd., 2018 (cit. on p. 6).

[6] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* Oct. 2008. URL: https://bitcoin.org/bitcoin.pdf (cit. on p. 4).

[7] *Solidity Documentation.* Version 0.7.5. Oct. 2020. URL: https://docs.soliditylang.org/ (cit. on p. 10).

[8] Melanie Swan. *Blockchain: Blueprint for a New Economy.* Sebastopol: O'Reilly and Associates, 2015 (cit. on p. 5).

[9] Fabian Vogelsteller and Buterin Vitalik. *ERC-20 Token Standard.* A standard interface for tokens. Mar. 8, 2019. URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md (cit. on p. 11).

[10] Gavin Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger.* Version 3e2c089. Sept. 2020. URL: https://ethereum.github.io/yellowpaper/paper.pdf (cit. on p. 6).

[11] Xun Wu and Weimin Sun. *Blockchain Quick Start Guide.* Birmingham: Packt Publishing Ltd., 2018 (cit. on p. 7).

Software

[12] *Chai.* 2020. URL: https://github.com/chaijs/chai (cit. on p. 11).

[13] *CherryPool*. 2020. URL: https://github.com/johannesdominik/cherrypool (cit. on p. 10).

[14] *Ganache*. 2020. URL: https://github.com/trufflesuite/ganache (cit. on p. 11).

[15] *Mocha*. 2020. URL: https://github.com/mochajs/mocha (cit. on p. 11).

[16] *React*. 2020. URL: https://github.com/facebook/react/ (cit. on p. 11).

[17] *The Solidity Contract-Oriented Programming Language*. 2020. URL: https://github.com/ethereum/solidity (cit. on p. 10).

[18] *Truffle*. 2020. URL: https://github.com/trufflesuite/truffle (cit. on p. 10).

[19] *web3.js - Ethereum JavaScript API*. 2020. URL: https://github.com/ethereum/web3.js (cit. on p. 11).

Online sources

[20] *Bitcoin Cash*. Nov. 2020. URL: https://www.bitcoincash.org/ (visited on 11/24/2020) (cit. on p. 6).

[21] *Bitcoin Energy Consumption Index*. Oct. 2020. URL: https://digiconomist.net/bitcoin-energy-consumption (visited on 11/24/2020) (cit. on p. 5).

[22] *Ethereum Casper*. Sept. 2018. URL: https://twitter.com/i/events/1036281460704112645 (visited on 11/24/2020) (cit. on p. 5).

[23] *Liquidity Mining, risks and exchange function on a DEX explained*. Nov. 2020. URL: https://julianhosp.com/liquidity-mining-risks-and-exchange-function-on-a-dex-explained/ (visited on 11/24/2020) (cit. on p. 8).

[24] *Uniswap Pools*. Nov. 2020. URL: https://uniswap.org/docs/v2/core-concepts/pools/ (visited on 11/24/2020) (cit. on p. 8).

Check Final Print Size

— Check final print size! —

width = 100mm
height = 50mm

— Remove this page after printing! —