Universally Accessible Order Book Style Decentralised Exchange

Student Name: Mahmut Baran Turkmen
Supervisor Name: George Mertzios
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—The increased trade of *cryptocurrencies* with real dollar value resulted in the high usage of exchanges. These exchanging services can be divided into two main classes: (1) centralised exchanges (CEX) and (2) decentralised exchanges (DEX). Existing CEX services maintain the control of user coins and work with order books making the trades more precise for the two parties (buyer and seller), whereas existing DEX services let the user maintain the control of their coins and work with vaults (two or more coins held in smart storage known as *pools*). This project aims to combine the advantages of both CEXs (where users can trade on order books) and DEXs (where users have complete control over their coins and do not connect to a centralised server) and create a new DEX such that it operates with order books where all the orders are saved on a blockchain. We aim to code this DEX in Solidity, and it was picked due to Ethereum's popularity and availability of running functions. In terms of data structures and algorithms, similar ones used in current CEXs are implemented in our application in terms of data structures and algorithms. The result is a fully functioning, responsive, universally accessible order book style DEX.

Index Terms—centralisation/decentralisation, Ethereum, exchange data structure design and analysis, user experience

1 Introduction

The aim of creating Bitcoin was to be an alternative to the current financial system [1]. After the 2007-2008 financial crisis, people all around the world, particularly the United States (US), were affected by the worsening economy [2]. The central triggering point of the crisis was banks taking huge risks by loosely lending money for housing loans [3]. These policies were approved by the board members of the banks and not everyday people, yet the people who suffered the most were the ones not participating in the decision making [2]. One of the aims of Bitcoin is to change this, a financial system for people made by the people, with no intermediary.

Bitcoin tries achieving this aim with the help of *blockchains*. The word blockchain derives from "chain of blocks", which can be compared to an online ledger that many people own and agree to add new transactions to the ledger based on some rules (the *Proof-of-work* consensus protocol) [4]. What makes blockchains different from bank ledgers is these blockchains can be stored at any machine (known as *nodes*). The result is a *peer-to-peer* (P2P) network used to transfer a *virtual currency* (known as *cryptocurrencies*, *coins*, *tokens*) [5].

In 2013 (5 years after the Bitcoin white paper was published), the Ethereum white paper [6] was published by Vitalik Buterin. The core difference between Ethereum from Bitcoin is the programming language *Solidity*. In Solidity, custom functions (known as *smart contracts*) can be deployed and run on the blockchain [7]. This brings a new level of functionality, whereas, in the Bitcoin network, a user could only transfer a virtual currency. Some examples are creating custom tokens or creating a *Decentralised Autonomous Organisation* (DAO, a self-governing organisation where proposals are voted for with tokens) [8]. Instead of coding a new token and a network from scratch, token creators started using the Ethereum network to launch their own token. This led to many tokens having a price and a significant trading market.

With the increasing amount of new tokens, the total market cap of cryptocurrencies skyrocketed [9]. These trades were done through both CEXs and DEXs.

A simple way of trading on DEXs was posting a token listing (token name, selling price and the sold amount) on a forum, and a buyer would message you to trade. However, a significant problem is trusting the seller (or the buyer) and whether they will send the tokens and complete the trade instead of running away after one party sends their tokens; thus, CEXs gained popularity.

1.1 Challenges of Modern Exchanges

One major problem with exchanges is that CEXs want users to deposit their tokens into the exchange's wallets. This is required since it is easier for exchanges to assign user balances instead of handling every user with their own wallet. From the user's perspective, depositing their coins to another wallet is risky since the user loses complete control over their tokens once the tokens leave their wallet. There have been incidents in the past where big exchanges were hacked (e.g., Mt. Gox) and resulted in millions of dollars worth of user funds being transferred to the hacker's wallet [10]. One of the most famous hacks is the "Mt. Gox Hack" [10]. At the time, roughly 460 million USD worth of Bitcoins were stolen [10]. The hacker acquired Mt. Gox's private keys (that are used for performing transfers from a cryptocurrency wallet) and transferred traders' Bitcoins to their own wallet.

Another major problem with CEXs is that exchanges or governments can block access to the service. For example, after the "2022 Russian invasion of Ukraine" was declared [11], CEX services such as Coinbase closed 25000 Russian accounts [12]. Before this event, some CEXs like Binance were banned in the US [13], which could have led to Binance users not being able to withdraw their funds.

In response to these problems, a group of developers started searching for an alternative method that does not require depositing funds into a wallet and where the access to the exchange was not easily blockable; thus, the solution that they have devised is the *Automated Market Maker* (AMM). AMMs are DEX services coded on one or more smart contracts where sellers provide liquidity (how readily an asset can be converted into cash) by locking their tokens in the smart contract and then collecting fees based on buy trades [14]. There were four aims when creating AMMs: (1) being able to sell or buy tokens without going on a P2P marketplace, (2) the user having the full ownership of their coins, (3) performing a safe transaction where you minimise the chances of getting scammed, and (4) the buyer not needing to match the price of the seller.

Whilst there are clear benefits of AMMs, there are also limitations; these are exemplified within the first AMM, Uniswap, which was founded by Hayden Adams in 2018 [15]. The major problem with Uniswap and other AMMs is how the selling price of a token is determined. Uniswap V2 uses the *constant product market maker formula* [16] $x * y = k$ where x is the amount of *token A* (one of the tokens in the pool) and y is the amount of *token B* (the other token in the pool). Providers (known as *farmers*) deposit Token A and token B into a vault (known as *pools*) with the aim to collect trading fees (known as *farming*). An example for providing token A is shown in Figure 1 [17]. When x is added to the pool, pool participation % of y decreases, making y more expensive (more x is required to get the same amount of y).

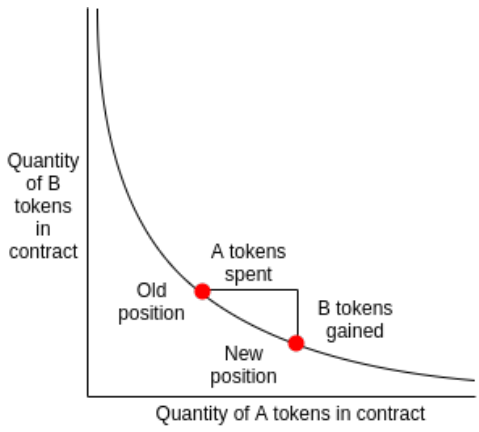

Fig. 1. Token A is deposited into the pool, and token B is withdrawn in return. As a result, the new position has more token A and less token B.

The limitation of the constant product formula is *impermanent loss*. Impermanent loss is a type of loss that only affects the token farmers. We further elaborate on this loss with an example. Imagine the current price of 1 ETH (the token "Ethereum") is 50 USDb (a token named "USD on the blockchain". USD does not exist on the blockchain thus we invented a token called USDb where 1 USDb is worth 1 USD. USD and USDb will be used interchangeably throughout this paper). A token provider would like to participate in the ETH-USDb pool in Uniswap V2 to collect fees from trading activities in that pool. To do that, the provider needs to supply the same amount for both tokens. For example, when they plan to deposit 1 ETH, they also need to deposit 50 USDb making the whole deposit worth 100 USDb (or 2 ETH). After the provider deposits 1 ETH and 50 USDb, a few ETH-USDb trades occur. Respectively to the trades, the reserves of ETH increased by 20% and USDb decreased by 20% change. The change happened because ETH was bought with USDb. As a result, the ETH reserves increased, and the USDb reserves decreased. Since ETH was bought, the price of ETH has increased, making it 60 USDb per ETH. When we compare the current shares of the farmers to the time before they entered, they made a profit of 8 USDb. This can be calculated by deducting the worth of current investment from the initial investment.

Respectively, if the provider did not deposit their tokens into the pool, they would have held 110 USD worth of ETH and USD. This can be calculated with the same formula above without deducting the initial investment. Therefore, 2 USD of impermanent loss has negatively affected the provider. The argument of accumulated fees balancing out the impermanent loss is not valid because 100 USD worth of deposit is a very substantial share of the pool, making the collected fees a few cents or lower. To elaborate on this, currently the worth of provided USDC (a token similar to USDb) and ETH in the USDC-ETH pool is 274 million USD. Therefore 100 USD owns a very substantial amount in the entirety of the pool, making it collect a tiny percentage of fees [18].

1.2 Project Objectives and Achievements

This project aims to combine the advantages of both CEXs (where users can trade on order books) and DEXs (where users have complete control over their coins and do not connect to a centralised server); thus, the research question **"Can a universally accessible order book style DEX be created and easily used?"** will be answered.

The objectives are divided into three categories: (1) basic, (2) intermediate and (3) advanced.

1.2.1 Basic Objectives

The basic category involved creating the stepping stones for intermediate and advanced deliverables. The first main objective of the basic category was to create tokens that were transferable to a smart contract. A prime example of these tokens is the ERC-20 (Ethereum Request for Comments 20) standard [19]. This standard is made to create tokens that are safe from exploitation (e.g., tokens getting transferred without the owner's approval). We followed the OpenZeppelin [20] documentation for the ERC-20 code and created two tokens with the symbols ETH and USDb. We needed a blockchain network to perform operations with these two tokens; thus, the second basic objective was to "Fork the Ethereum network". When a blockchain network is forked, it can be locally hosted. Initially, forking was done with the help of *Hardhat framework* [21]. However, it was later identified that forking the Ethereum network was unnecessary

because a new Ethereum blockchain (which only included the first block) satisfied the requirements of running our proposals. Furthermore, performing unit tests were easier with *Remix IDE* [22]; thus, Remix was used instead of the Hardhat framework. The third and last basic objective was to create a modifier that only allowed whitelisted users (users who have access to call a function) to call a special function. A trading (buying or selling) fee was going to be implemented, which only the deployer of the vault should be able to collect; thus, the whitelisting feature was required. Similarly to the ERC-20 code, an OpenZeppelin library was used. The name of the contract module is *Ownable*. It helps to set an owner to the deployed contract and creates a modifier named *OnlyOwner* that strictly limits a function to be run by the owner. Overall, these three basic deliverables were planned to be completed from the first week of November to the second week of December. These objectives were successfully delivered.

1.2.2 Intermediate Objectives

The intermediate category involved the main functionalities of the exchange application. The first objective of the intermediate category was to code an *exchange smart contract* where two ERC-20 tokens could be deposited. This smart contract consists of 3 minor contracts. They acted as support libraries that manage data structures (e.g., linked lists). In contrast, the exchange smart contract handled the business logic of placing and deleting an order. The initial plan of the research question involved saving and computing everything on the blockchain. Due to the high cost of computing, some functions on the blockchain (e.g., iterating a list) were implemented on the client-side application. This reduced the transparency of the application. The second objective of the intermediate category was to code the *factory smart contract*. The factory smart contract is used to instantiate multiple exchange smart contracts with different ERC-20 pairs (see Figure 2). The third objective of the intermediate category was to collect trading fees. The fourth and last objective of the intermediate category was to create a *graphical user interface* (GUI). The plan was to deliver them from the second week of December to late February. Unfortunately, the exchange smart contract protocol took more time than expected. Therefore, these deliverables were completed by late March.

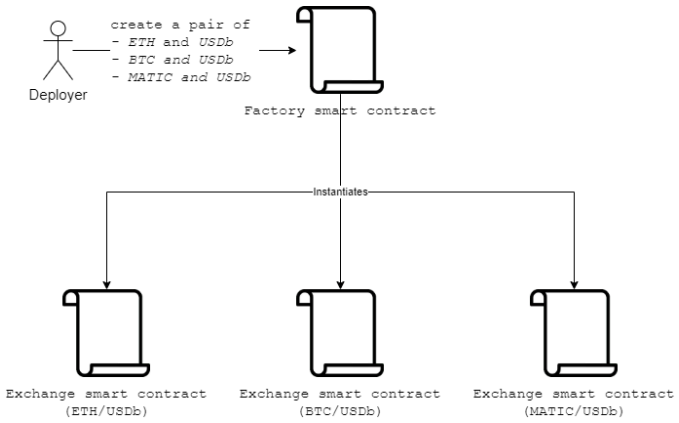

Fig. 2. The deployer instantiates multiple exchange smart contracts.

1.2.3 Advanced Objectives

The advanced category involved adding helpful indicators for the trader. We planned to implement a $time * price$ (known as *price chart*) graph where users could see the price history of sell and buy prices, and a *price oracle* which estimates the current price of an ERC-20 token in USD. The price oracles we used are operated by Chainlink [23]. However, only the price oracle indicator was implemented due to limited time constraints.

2 Related Work

A significant amount of exchanges have been launched in the last 12 years. Almost all were centralised until the past 5 years. From 2017 to 2022, the decentralised ones only started to gain popularity. The main reason for DEXs gaining popularity this late is the launch of Ethereum and smart contracts in 2015 [6].

2.1 The Bitcoin Forum

After the original Bitcoin protocol was proposed in 2008 [1], the creator *Satoshi Nakamoto* posted the Bitcoin project on a public forum named *P2P Foundation forum* [24]. This forum was the first website to share the project with the public [24]. Bitcoin gained attention in this forum. With the increased attention and to discuss only about Bitcoin, *Bitcoin Forum* was created.

The first post on Bitcoin Forum was posted on Dec. 9, 2009 [25]; almost a year after the post on the P2P Foundation forum. This forum became the home of Bitcoin and continues to this day. One of the most remarkable features of this forum is the *Marketplace* board. Under this board, there are many boards classified as *child boards*. Some examples of child boards are *Lending*, *Gambling* and *Currency exchange*. Under the Currency exchange child board (a screenshot of the website is in Figure 3), users traded Bitcoin to fiat currencies (e.g., USD, CHF, GBP) and vice versa. Many old posts can be found dating back to Dec. 14, 2010 (only 1 year and 5 days after the first post on the forum). These exchanges were mainly done in a P2P fashion where buyers wired money through a bank account, and the seller would send their Bitcoins after receiving the money. This method was particularly risky to the trade initiator that first sent the funds since there was no protection on either side. Another way was to meet at a place and exchange cash. This method was safer with the trade-off of revealing more personal information about the buyer or the seller. Another downside of this method was that the buyer and the seller needed to be at the exact location, making it harder to perform the exchange. Because of the limitations of these two trading methods and poor user experience (UX), an existing approach for ongoing trading services of *order books style exchanges* was adopted for trading Bitcoin.

The user interface (UI) of Bitcoin Forum is similar to traditional forums. On the main board, there are many headlines. Each headline is a post containing a title, a writer, and information about the last date it was modified (this includes when a new comment is made). The design and interface of these forums are not suitable for trading. The reason is there is no standard for posting a sell or a buying

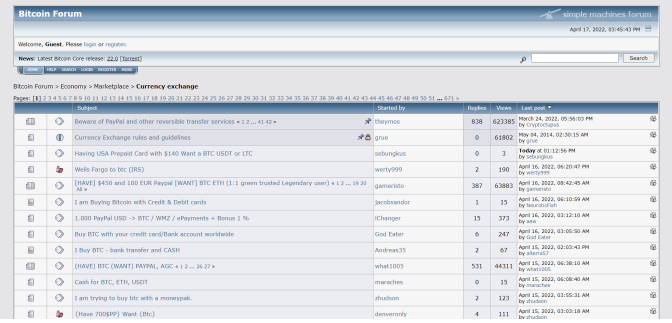

Fig. 3. This page shows the Currency exchange child board under the Marketplace board. The listings are ordered by the newest comment on a listing, making it harder to distinguish the side (sell or buy), the trading price, the trading size, the payment method and the trader's rating.

listing, thus, making it hard for traders to find a listing. These listings include a price per Bitcoin and trading size. Since these listings are not easily identifiable, the possibility of finding counterparty decreases, resulting in less amount of trades. Order book style exchanges introduce a component that standardises trading: the order book. As a result, order books improve the UX.

2.2 Mt. Gox

Due to risks surrounding payment safety, lack of accessibility and poor UX on Marketplace on Bitcoin Forum, an order book style exchange service was launched in 2010 [26]. Mt. Gox was the first CEX at the time. The main idea was to improve the feasibility of buying and selling (trading) Bitcoins. The traders entered the amount of Bitcoin (BTC) they wanted to sell or buy and the price per coin (see Figure 4). If there were an existing order for the given price, the order would get matched, and both traders would receive their funds. Mt. Gox gained popularity due to its ease of use and handled over 70% of all BTC transactions by 2014 [27]. However, the service had to shut down in 2014 after a major hack and the loss of Bitcoins.

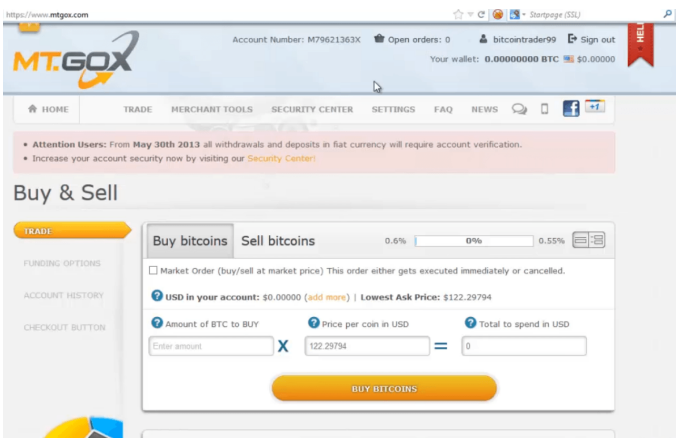

Fig. 4. Mt. Gox's "Buy & Sell" panel [28]. There is no price chart.

The reason users lost their funds was how Mt. Gox's exchange protocol worked. First, Bitcoins had to be deposited into Mt. Gox's Bitcoin wallet for the user to be able to trade. Then Mt. Gox would assign a balance of how much the user deposited. This protocol was disliked among users since once the coins were deposited, the user no longer had complete control over their coins. As a result, the phrase "Not your keys, not your coins" became popular, and once again, users followed the old P2P cash-exchange methods.

2.3 The First Organised DEX: LocalBitcoins

LocalBitcoins is a DEX found in 2012 [29]. Since it was easier to exchange currencies on Mt. Gox, LocalBitcoins was less popular at the time. However, after the hack and loss of user funds in Mt. Gox Hack, users started searching for P2P alternatives where users kept their Bitcoins in their wallets at all times (i.e., users do not need to deposit their coins in an exchange's wallet to trade).

The trading methods on LocalBitcoins are very similar to the Marketplace board on Bitcoin Forum: users can buy and sell Bitcoin with cash or by bank transfer. However, the main difference between LocalBitcoins from Bitcoin forum is (1) the UX, (2) liquidity levels (i.e., more Bitcoin is sold and bought on LocalBitcoins) and (3) trade protection. A screenshot of the current UI is shown in Figure 5.

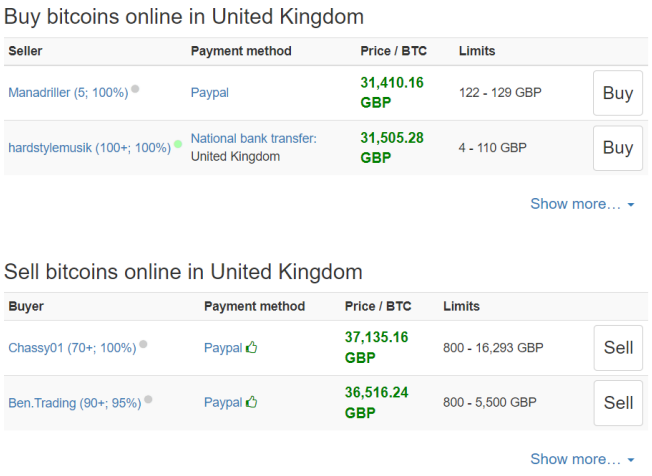

Fig. 5. UI of buying and selling page on LocalBitcoins. A listing row includes the trader's name, rating, the payment method (e.g., PayPal, cash, bank transfer), the price per BTC and maximum trade size.

In terms of UX, LocalBitcoins has a more organised website than Bitcoin Forum. The main reasons are the user can select filters such as the country they are performing the exchange in, the trading price per Bitcoin, the trading side and the payment method.

The liquidity level on LocalBitcoins was higher. This means that more Bitcoin was sold and bought for a fair price on LocalBitcoins than on Bitcoin Forum; thus, the fair price and the number of active listings lead to higher usage of the service LocalBitcoins.

The trade protection on LocalBitcoins ensures the two parties receive their funds after wiring the cash (or transferring the coin). An example of a buying flow is as follows. First, the buyer initiates a chat with the seller. Then, the seller sends their bank details. Sometimes the sellers can ask for additional information from the buyer (e.g., their driving license). This process ensures that the name on the bank transfer matches the name on the ID. After that, the seller

transfers the agreed amount of BTC to the LocalBitcoins trade. Once the buyer wires the agreed amount of cash and notifies the seller, the temporarily locked BTC in the trade gets unlocked, and the buyer receives their coins. This process has many steps that can be automated (i.e., the seller and the buyer does not have to chat, one party should not wait for another party to confirm they have sent first, or sensitive information should not be requested whenever initiating trade with a new seller or buyer) and involves dealing with sensitive information; thus, another solution was found and became more popular.

2.4 The First "Smart" DEX: EtherDelta

When Ethereum launched in 2015 [6], it drastically changed the scene of exchanges. Ethereum introduced *smart contracts*. A smart contract is one or more user-defined functions saved on the blockchain. Deploying a smart contract is the equivalent of hosting an API, and an API can be used instead of going through the process of trading manually like in LocalBitcoins. To interact with an API, the user needs to connect to a *node* (a server that hosts a particular blockchain and is connected to other nodes with the same state). These nodes are similar to servers where the back end is hosted. The most significant difference between blockchain nodes from centralised servers (e.g., Amazon's Elastic Compute Cloud) is the blockchain node can be hosted on any machine. Once a machine starts hosting the blockchain, they become a node and host all the smart contracts. The data saved in the blockchain is referred to as *on-chain*, and the data saved elsewhere (e.g., Amazon's Elastic Compute Cloud) is referred to as *off-chain*.

EtherDelta, founded by Zachary Coburn, published its first exchange smart contract in 2016 [30]. It uses an order book style where some data is saved off-chain, and the rest is on-chain. Keeping data on-chain is the equivalent of making it universally accessible. As long as the user can connect to one of the blockchain nodes, they can access the data. Unless the connection to all the nodes is blocked (by a government or the internet service providers), the data is always accessible. Blocking the access to all the Ethereum nodes is very hard since there are 5861 nodes actively running and in sync with the latest block [31]. Not only the number of nodes but new nodes are set up every week [32]. This is where our solution differs from EtherDelta; we host all the necessary data (to process the order book) on-chain.

EtherDelta stopped its operations in 2018 due to facing a charge from the SEC for "operating an unregistered exchange" [30].

2.5 Automated Market Makers

Uniswap, founded by Hayden Adams, is the first *automated market maker* (AMM) that was launched in 2018 [33]. The four advantages are explained in section 1.1. To summarise the advantages are the following:

1) Being able to sell or buy tokens without going on a P2P marketplace.
2) The user keeping the full ownership of their coins.
3) Performing a safe transaction where you minimise the chances of getting scammed.
4) The buyer not needing to match the price of the seller.

EtherDelta DEX already covered the first three points. An important difference between EtherDelta and Uniswap is that EtherDelta did not save every data on-chain; meanwhile, Uniswap did.

In time, the trading protocol of Uniswap got updated. The reason for updating the protocol was providers were losing funds because of impermanent loss (explained in section 1.1). The current trading protocol of Uniswap V3 is similar to order books. To elaborate, providers still provide token A and token B into a pool; but they also provide a range where the trading fees will be collected. This new feature is introduced as *concentrated liquidity* [34]. Some users saw this new protocol as an opportunity and decided to perform an attack known as "Just-In-Time Liquidity Attack" [35]. The attack is performed as follows: a buyer wants to trade a significant amount of token A to B. The buyer initiates the swap (or trade) by sending the transaction. As soon as token A leaves the buyers account, the bots listening to the *pending transaction pool* (where transactions first enter before getting confirmed) find out that there is a vast swap that will happen in a few seconds. These bots instantly provide a significant amount of tokens A and B with a narrow trading range focusing on the range where the trade will occur. Thus, the attacker aims to collect most of the trading fees by providing liquidity to a specific price range. The buyer does not get affected since the trading fees they pay do not change, and their buy price is the same; however, other providers earn substantially less trading fees because the attacker owned the majority of the pool for the given range when the trade occurred. The order book style exchanges prevent this attack since it eliminates the incentive for farming.

2.6 Second wave of CEXs

In Dec. 2017, the price of Bitcoin surged 35% in 2 weeks [36]. At the time, cryptocurrencies were popular, and many exchanges were used. One of them was Binance. Binance's UI, shown in Figure 6 was one of the significant influences on this paper's UI due to its popularity and ease of use.

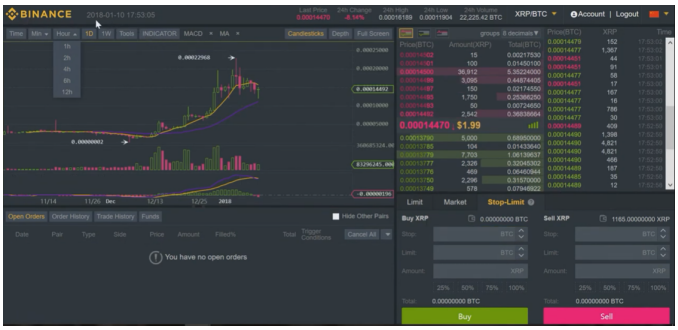

Fig. 6. UI of buying and selling page on Binance in 2017 [37].

Six main components can be seen on the trading page.

1) The top bar where time-based details can be learned from the coin (i.e., 24-hour price change).

2) The price chart indicates the previous trades (buy and sell) and their timestamp.
3) The component below the price chart is where the active orders rely on.
4) Next to the price chart is the order book. Prices and the size of total orders are shown.
5) Next to the order book, previous trades are shown in a list (instead of a price chart).
6) Below the order book and list of previous trades is the limit order panel. Users enter a price and an order amount and click a button to buy or sell the provided coin on the current website.

This interface was later updated to Figure 7. This was the main influence for creating the components.

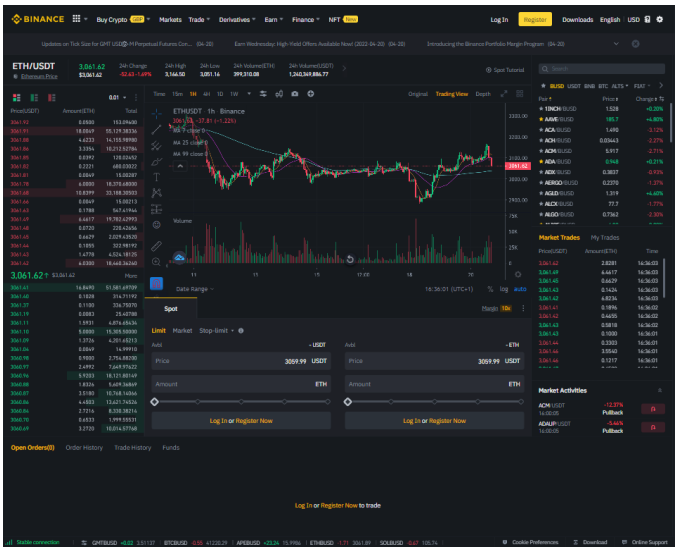

Fig. 7. UI of buying and selling page on Binance in 2022. The colours are darker and more distinct. The price chart and the trade panel are centred.

3 Methodology

The solution designed to build the universally accessible order book style DEX is presented in this section. We start by providing problems and solutions instead of giving the whole architecture.

Due to the nature of the project, we split the subsections into two main sections: (1) the back end and (2) the front end.

3.1 Back End

In this subsection, we explain how and why the exchange protocol works.

3.1.1 Hosting the Service

One of the project's aims was to make the exchange service as accessible as possible. This meant users should be able to use the exchange from any computer or phone connected to the internet. Unfortunately, hosting the exchange on a centralised service allows governments and ISPs to ban the service, making it almost impossible to use. VPNs are a solution, but once the government bans an exchange, the users can no longer deposit or withdraw their coins from that exchange. The reason for this is the bank accounts refuse to accept deposit or withdraw requests to the exchanges bank account. Another problem with centralised exchanges is they do not give *private keys* to user wallets. In this context, private keys are needed to execute transactions on a blockchain (e.g., transfer coins). If the user does not have their private keys, the user does not have full ownership of their coins.

We chose to host the service on a publicly accessible blockchain network to solve the problems above. As of now, there are countless amounts of different blockchain networks. The primary requirement to host service was to host a piece of code on the network. After finding a dozen networks, we decided that security was critical; thus, we started looking for networks hosted by the most servers. The most hosted public blockchain network where code can be hosted was Ethereum; hence this was selected.

The Ethereum network is not just a *public database* (or a *ledger*), it also executes codes. These codes bundled in *smart contracts* get executed in the *Ethereum Virtual Machine* (EVM) [6]. EVMs define the rules of computing transactions that change *states* (data objects that are permanently saved). A rule for running code in EVM is the code needs to be in byte-code.

Different programming languages to work with EVMs compiled down to byte-code were created [38]. The two main languages in which a smart contract can be coded are (1) Vyper and (2) Solidity.

According to Figure 8, Vyper accrues less interest [39].

Fig. 8. Red represents Solidity, and blue represents Vyper. A value of 100 is the peak popularity; meanwhile, a value of 0 means there was not enough data for this term.

We looked at the smart contracts that were most used based on spent *gas fees* (*gas* is the necessary cost to complete a transaction on the Ethereum network). These smart contracts were written in Solidity. We also searched for tutorials online and found more resources for Solidity. These two reasons were why we went with it instead of Vyper.

3.1.2 Features

We came up with a set of features to include in our exchange.

1) Order panel where the user places new buy and sell orders.
2) Order book where the total size of the order for prices is listed.
3) Active orders where the user's active orders are listed.
4) Price chart where the user can see the past buy and sell trading prices.

To build such a set of features, we had to develop our data structures. To keep track of which user address deposited how many coins, we created a mapping (i.e., hash table) variable named *deposits*. Deposits take a parameter with the type *address* and map to the amount with the type *uint256*. With *deposits*, the smart contract can keep track of the amount of coin deposits per-user address and allow them to cancel orders and withdraw the correct amount of tokens. However, there was a missing piece, in case of a deposit of *20*, the smart contract would not know if the deposited token was 20 ETH (Ethereum token) or 20 USDb (USD on the blockchain token); thus an additional mapping was created.

3.1.3 Keeping a Track of Deposits

The current state of *deposits* is it accepts an *address* and returns an amount (*uint256*) (see Figure 9). Instead of creating two different mapping with similar names, we created a new mapping in *deposits*. First, *deposits* accept one of the token addresses, then an *address*, and then it returns the amount.

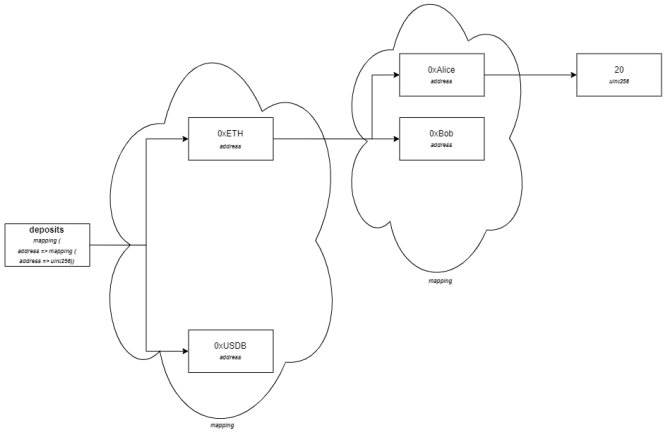

Fig. 9. *deposits* accept two different *address*, the address of the Ethereum coin or address of the USDb (USD on the blockchain) coin. In this example the Ethereum address accepts two mappings, addresses of Alice or Bob. When address of Alice is provided, the amount of 20 is returned. This indicates that Alice has deposited 20 ETH to this smart contract.

Since we save the deposits, now we can start saving the incoming orders.

3.1.4 Keeping a Track of Orders

Order is a data structure that saves two variables, *address* of the seller and a *uint256* of the sold or bought amount (see Figure 10).

Order	
seller *address*	amount *uint256*
0xAlice	20

Fig. 10. Example of an *Order*. The seller's address is 0xAlice and the amount is 20 ETH or 20 USD. Even though one of the property's name is *seller*, this might be a buy order.

The reason for not including the side (buy or sell) is that it costs 32000 gas to perform a CREATE operation [6]. One of the aims is to perform transactions with minimum gas; thus, we looked for an alternative solution and followed a similar solution to *deposits*; by creating an additional mapping variable named *orderBook* (see Figure 11).

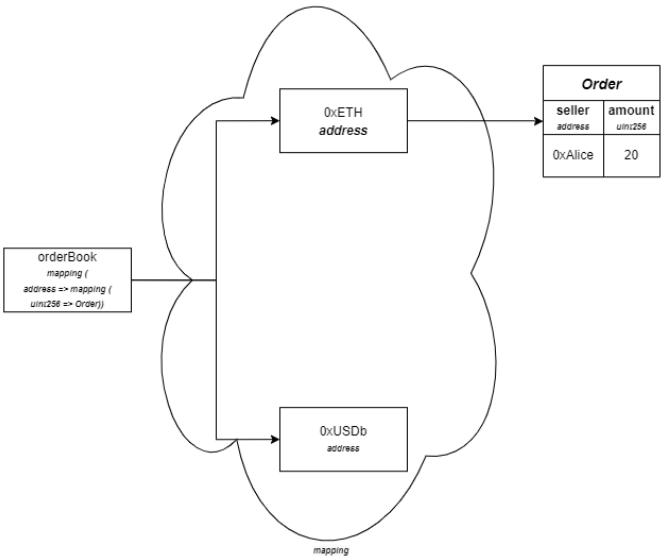

Fig. 11. The mapping variable is named *orderBook*. ETH means *sell* order, and USDb means *buy* order. In this example, a sell order of 20 ETH is placed by Alice.

However, the current order book lacks one of the main features: the (selling or buying) *price*. The solution we came up with was to create an additional mapping of prices (see Figure 12).

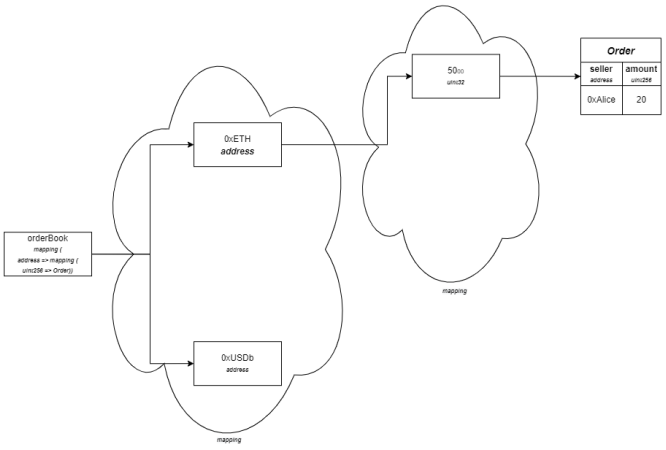

Fig. 12. An extra mapping of prices (in type *uint256*) has been added. In this example, a sell order of 20 ETH by Alice for a price of 50 USD has been placed.

The problem with current *orderBook* is only one order per price can be saved. Initially, we created a list of orders; however, orders were traversed to cancel an order. The time complexity of searching a given order is $O(n)$ where n is the number of orders for a price (see Table 2). The solution we came up with was to create a mapping where every order was given a unique *order ID*.

TABLE 1
Time Complexity

Data Structure	Search
Array	O(n)
Hash table (or mapping)	O(1)

3.1.5 Sequential Orders

These order IDs (in *bytes32*) are guaranteed to be unique. They are created by hashing numerous arguments such as the trader's address, trade amount and the timestamp of the current block of the network (see Figure 13).

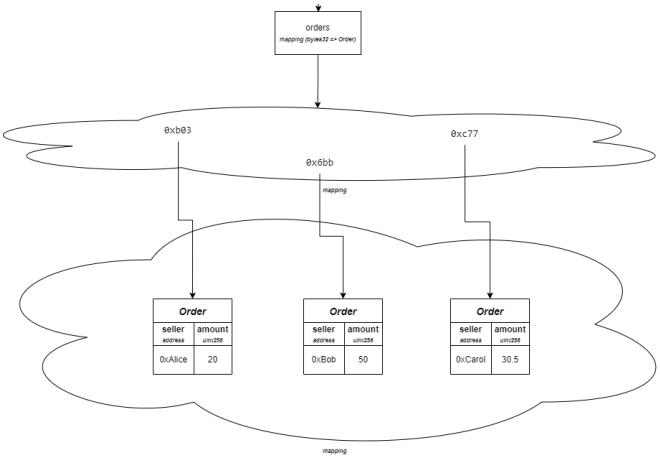

Fig. 13. Example of *orders*. In this example, there are 3 transactions (whose first symbols are 0xb03, 0x6bb, 0xc77) for the selling price of 50 USD.

A significant issue with this solution is filling the orders. For example, a buy order of size 30 ETH for a price of 50 USD has been placed. The first order of size 20 ETH will get filled, and the second order of size 50 will get partially filled. How does the algorithm know that the sell order with size 50 is just after the sell order with size 20? It does not; thus, we linked orders together by creating a linked lists library in Solidity (see Figure 14).

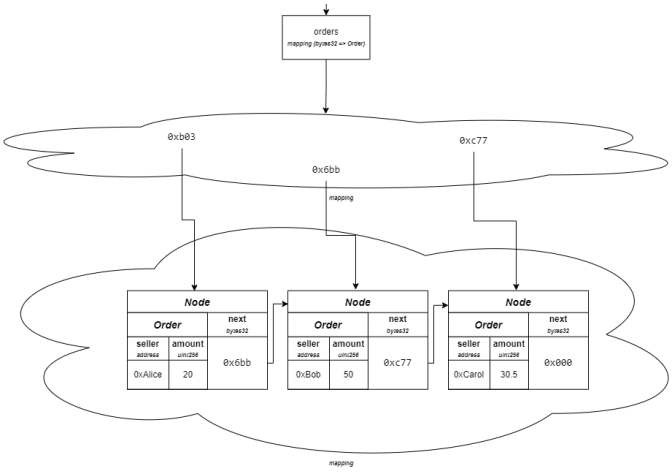

Fig. 14. Similar figure to Figure 13 but the main difference is each *Node* contains an *Order* and the hash of the next order.

Figure 14 is very close to the current *orderBook*. As of right now, traders can place orders on a side for some price. When new orders get placed on the same side and for the same price, they get linked. A problem occurs when an order is placed on the opposite side for the same price. Orders cannot be matched since the first order is not known. This happens due to the mappings' nature. A solution to fix this is to create a new data structure named *LinkedList* where the head *Node* is saved. We also decided to save some additional properties of the *orders* such as the length and the tail (see Figure 15).

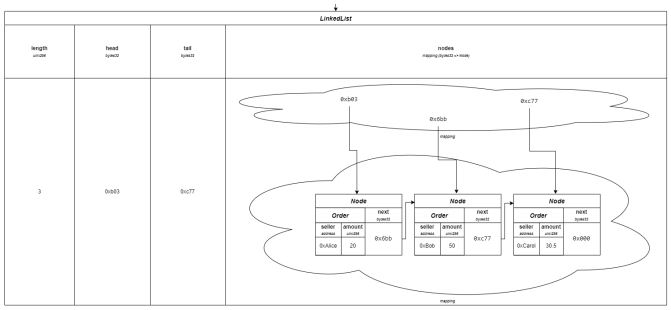

Fig. 15. In this example, there is a *LinkedList* data structure where the length is 3, the head hash starts with 0xb03, and the tail hash starts with 0xc77. Name of *orders* have changed to *nodes* due to storing *Node* instead of *Order*.

The length is needed to traverse through the linked list and match orders. The head is needed to define the first order. The tail is required to flag the *PVnode* (more on this later) that a *PVnode* for the given price exist. If the flag is not equal to 0, a new linked list is not initialised for that price and the current head node is modified; else, a new linked list is initialised.

This is an unoptimised implementation since traversing on-chain costs a large amount of gas compared to only working with one mapping. This unoptimisation problem was discovered in the later stages of the project when it was decided to implement some of the functions off-chain on the front end. More on this is explained in the evaluation section.

This is the current state of *orderBook*. Traders can place an order on some side for some price, and these orders get linked to the previous ones. It is also possible to match opposite orders (on the opposite side for the same price) since traversing through the linked list is possible.

With the current data structures, the order book panel (where the active orders for some price with some size) are not easily retrievable. Since the order prices are mappings, every number from 1 to 2^{32} (since the type of *price* is *uint32*) should be given as an input for some side. This calling operation needs to be done not only once but twice (due to *orderBook* having two sides of buy and sell). This "calling the total trading size for some price" is optimised by introducing a new data structure named an array of *PVnode*.

3.1.6 The Order Book

PVnode stands for "Price and Volume Node". Each *PVnode* stores a price and the total volume (or total size or total amount) of orders. Due to two sides existing in the order book, two array of *PVnode* were initialised: (1) *sellOB* and

(2) *buyOB*. The idea was to call these two arrays, retrieve all the total volumes for all prices, and get only the top N prices. The ordering was going to be determined by the most relevant order. In this context, the most relevant orders for selling are the *PVnode* with the lowest price; for buying is the *PVnode* with the highest price. To ensure this optimisation, *max heap* and *min heap* binary trees were implemented (see Figure 16).

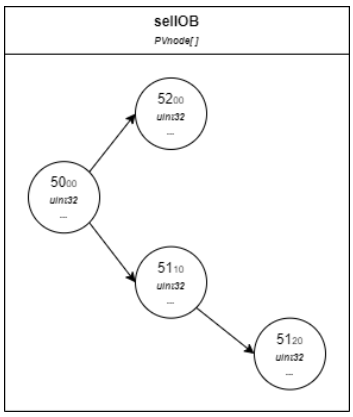

Fig. 16. In this example, there are 4 *PVnode* with prices (in *uint32*) of 50, 51.10, 52.20, 52 USD. Prices in the nodes are represented without a dot because Solidity does not support floating-point numbers. The pricing convention used throughout the project was to add two 0's as if they were cents. The "..." represent the total volume for the corresponding price.

In the worst case, it takes $O(\log n)$ time to insert an element to these trees. The main problem is that when inserting a node, nodes are replaced iteratively. To perform this, many on-chain operations are invoked. This increases the total gas cost to insert and delete *PVnode*. After implementing max heap, this solution was revoked, and a simple array was used to save many *PVnode*.

When an order is placed with a price that has not been set before, a new *PVnode* is initialised and appended for both *sellOB* and *buyOB*. Appending at both arrays ensures the stability of the index and helps when performing operations. This can be further explained by giving an example when placing a new sell order. When a sell order is placed for some price, *sellOB* is searched. If a *PVnode* does not exist for the given price, a new *PVnode* with that price is appended. If a *PVnode* does exist for the given price, the same index in *buyOB* is searched. If the total volume is greater than 0, there are active orders for the given price.

3.1.7 Active Orders

Once a user places a new order, the order is saved in *orderBook*. After placing the order, the order ID is no longer easily accessible. To easily access order IDs for a given user, we created two structures: (1) *_sellOrders* and (2) *_buyOrders*. These structures work as a *set*. *_orders* is a mapping that accepts user address and returns a list of *OPVnode* (Order ID, Price and Value Node) and *_indexes* keep track of that order IDs index in the returned list.

A use case will elaborate further. For example, Alice has placed a sell order for a price of 50 USD and a size of 1 ETH (see Figure 17).

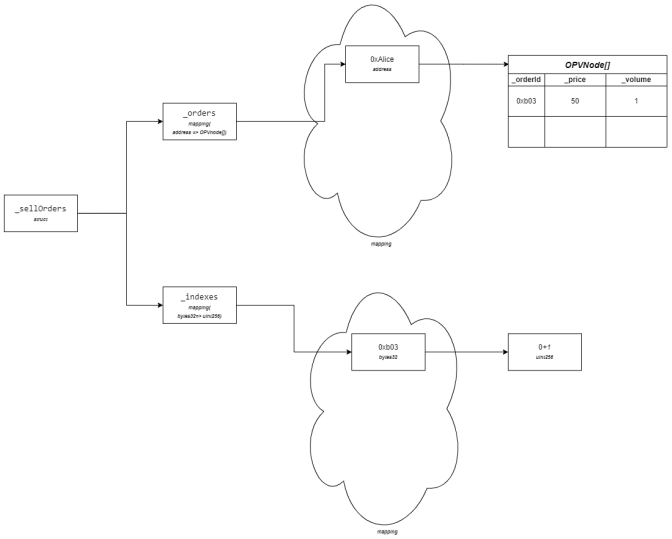

Fig. 17. Alice has an active sell order for a price of 50 USD and a size of 1 ETH. The index of this order is 1 and not 0 because 0 indicates null in a Solidity mapping; thus, a different convention of indexing is used.

They would like to cancel this order. Some arguments need to be given to the smart contract to cancel an order. We chose order ID as the primary key. *_sellOrders* accepts a user address and returns a list. To locate the index of the order in $O(1)$ time, the index is retrieved from *_indexes*. After that, the last order in the list of active orders is replaced with the cancelled order. This leaves no null memory locations in the list of *OPVnode*. The reason for not preserving the list of orders is not required; thus, an optimal solution of "delete and replace" has been used.

3.1.8 Trading Fees

Trading fees are set to 0.1%. This fee is deducted when any token is deposited. This includes placing a new buy and sell order.

Fees are collected with a function that can only be invoked by the deployer of the exchange smart contract. This way, no other user can withdraw the accumulated fees. The library used to initialise the owner (and allow it to be modified) is created by OpenZeppelin [20].

3.2 Front End

In this subsection, we showcase the UI. This includes the design process and the implementation.

3.2.1 Design

The application that was used to prototype the project was Figma. It is a free design application relatively easy to use compared to its competitors. We watched a few tutorials to start using it and continued by designing the first draft (see Figure 18).

- The top bar has a dropdown menu to select a token pair (e.g., ETH-USD or BTC-USD) and a price oracle. The price oracle is a result of a group of machines that verify each other based on a consensus made by Chainlink [23]. They agree on a price, and that price is updated every 30 seconds.

Fig. 18. The first draft of the UI. The level of detail is low. Most of the colours are not set. Buttons and other bars do not have a standard width or height.

- The main panel on the left is where the order is placed. To place a ”Limit order”, the trade’s size (either ETH or USD) and the price per ETH are entered. We also added a ”Market order” option where the front end would detect the highest buy price to sell (bid price) and the lowest sell price to buy (ask price).
- The main panel on the right is where the order book is placed. We can see the sell prices coloured in red and the buy prices coloured in green. The coloured numbers on the left indicate the price, and the numbers on the right indicate the total size. There is a mid bar where the spread $(highest\ sell - highest\ buy)$ is shown.
- The bottom panel indicates the active orders and the trade history. Each row on active orders consists of the side, size of ETH, price per ETH and a cancel button to cancel the order.
- There are two buttons at the very top. The left one opens settings, and the right one connects a third-party wallet. This wallet accepts (or creates) private keys and signs transactions for the user. They make the user’s life easier when signing and sending transactions.
- The price chart was going to be added, but at the time, there were doubts about whether there would be enough time to implement this advanced requirement; thus, it was never included in the design process.

After finalising the initial, low detailed draft, we continued to the high-detailed draft. First, we looked at the most popular CEXs and DEXs, such as DYDX, Binance and Coinbase. Then, we were influenced by their choice of colour and component placement. DYDX was the most considerable influence in terms of colours and Binance in terms of components (see Figure 19).

- The ”market order” tab has been removed, and the trade button has been replaced with ”buy” and ”sell” buttons.
- The ”fee” feature has been added just above the trade buttons.
- ”Settings” has been removed since no setting was set to configure the service.
- The ”pool” tab has been added at the very top left. This tab allows users to create their pair of tokens (pool). For example, the current pool on this page is ETH-USD.

3.2.2 *Implementation*

The scripting languages of choice to design the website were HTML and CSS for the implementation. Most components and colours were already finalised in the design; thus, turning the components into code was a matter of fact.

As an intermediate objective, React was planned to be used as the front end framework. Due to not having sufficient knowledge of React and JavaScript, we went through a series of tutorials on JavaScript. HTML and CSS took one week to complete. After finishing that, we started adding the functionality. It took a week to get familiarised with the language and another week to meet most of the requirements.

As the wallet, MetaMask functionality was implemented. The main reason for implementing MetaMask was due to its popularity as of now [40]. The main library to interact with the Ethereum network was Ethers [41]. It is a compact library, and due to our prior experience, it was used in this project.

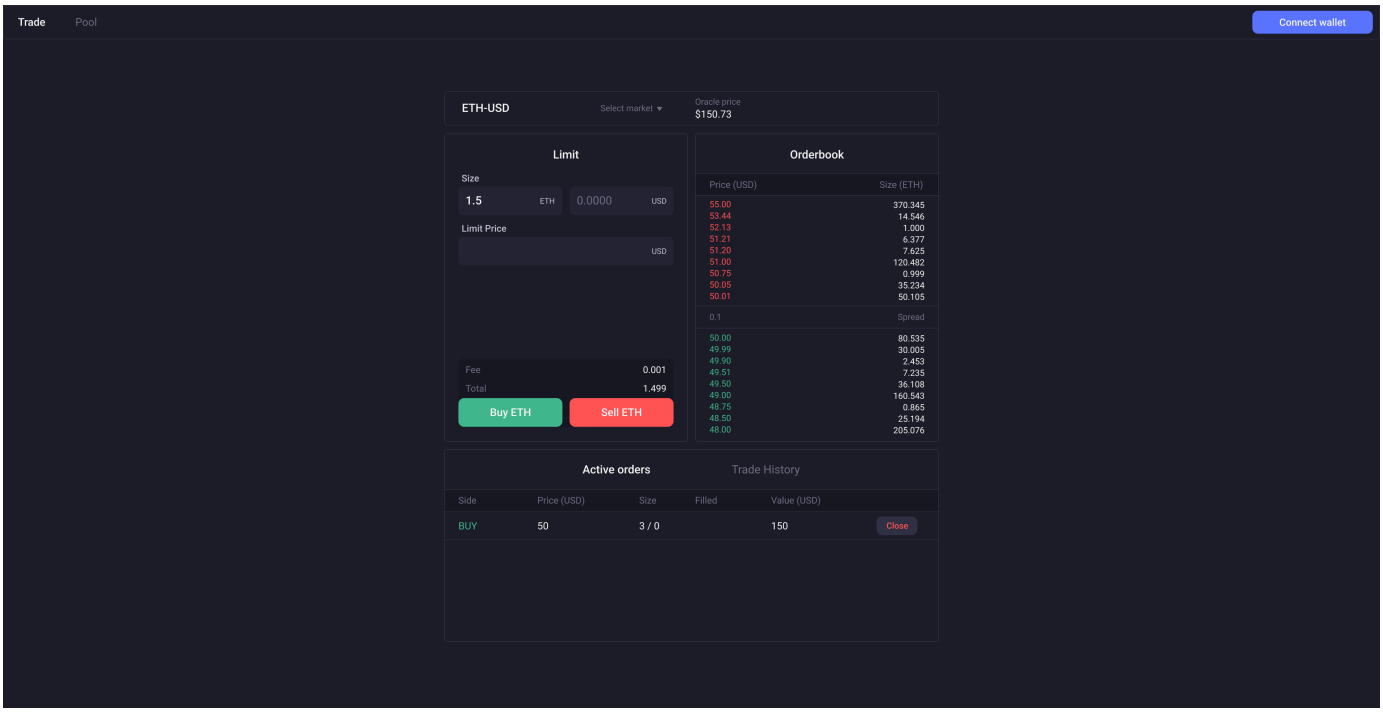

Fig. 19. The second draft of the UI. The level of detail is high. The colours are set. Buttons and other bars have a standard width and height.

One of the most tricky parts was to reload the website after every new block. The reason for this is a block is mined every few seconds. Each block contains a set of transactions. Some of them are deposits, placing an order and cancelling an order. These transactions get finalised when the block is mined; thus, in case of a change in the state of the order book, the website has to be refreshed, and the data has to be retrieved from the blockchain. We used a custom JavaScript event that listens to new blocks. When a new block is mined, the website retrieves the data on the network. We made sure that the application did not make too many requests to the node where it retrieved the data, or else the machine could have been blocked. This was one of the implementation choices. Only X number of requests are sent after every new block to retrieve all the necessary data.

4 Results

Our DEX is deployed at "https://on-chain-dex.vercel.app/". A screenshot of the trading page is provided in Figure 20.

4.1 Developing the Environment

4.1.1 Back end

Two ERC20 tokens were coded. OpenZeppelin library named "ERC20.sol" was used [42]. We coded the two from scratch and discovered the library later on.

We added the "whitelisting" feature early on. This feature was needed since only the deployer of the exchange smart contract should be able to collect the trading fees. This was successfully implemented with an OpenZeppelin library named "Ownable.sol" [43].

Instead of coding the factory smart contract first, we coded the exchange smart contract. The limit order protocol took the most time (close to 150 hours). There are three actions that the user can make: (1) initialise a *PVnode*, (2) place a new limit order and (3) cancel an existing order. The required gas, the cost in USD and time complexity are shown in Table 2. The gas price was assumed to be 50, which is the approximate gas price on Polygon [44]. The current price of Polygon (MATIC) is 1.46 USD [45]. The formula of for calculating how much each action cost is the following:

$$Cost\ of\ Action = \frac{Required\ Gas * 50 * 1.46}{10^9}$$

The division is needed due to the conversion of gas prices from WEI to GWEI. There are two different time complexities. On-chain time complexity is the time it takes for the action to be completed on-chain. Off-chain time complexity is the time it takes to compute on the front end. n represents the number of nodes (i.e., *PVnode* and *LinkedListLib.node*).

The component which allowed users to place a "market order" was not built due to limited time constraints.

The factory smart contract was coded after the exchange smart contract. The factory produces multiple exchange smart contracts. The only requirement to instantiate an exchange smart contract from a factory smart contract is a pair of token addresses. Once the token pairs are provided, a new exchange smart contract can be deployed. Users should have been able to deploy their exchange smart contracts through a new UI tab; however, this feature was not implemented due to limited time constraints.

A price oracle of the price of Ethereum was implemented. Chainlink's price oracles were used. This price oracle helps the users by providing the current agreed price of Ethereum in USD.

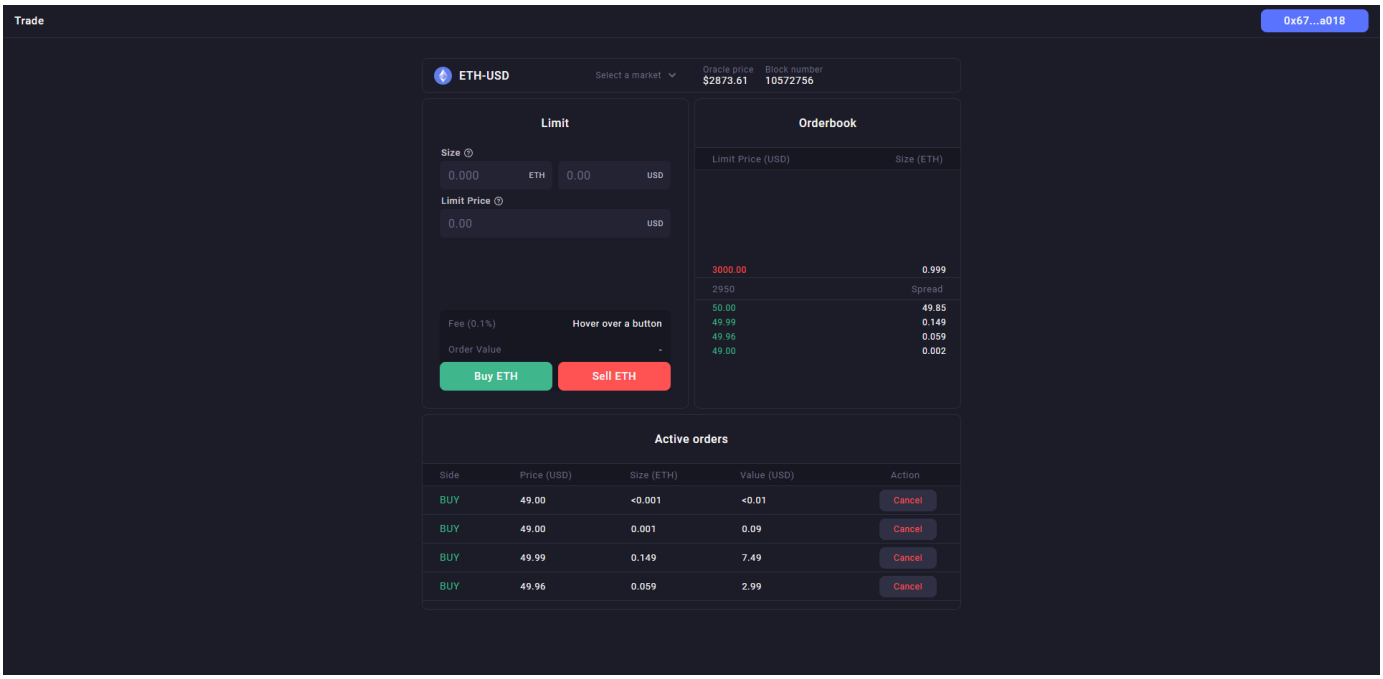

Fig. 20. A screenshot of the app logged in with the address 0x67C61e4C0e954B3a1589612510c5ecF0dF8a018C. The order book on the right has 5 rows with different prices. There are 4 active orders placed by this address.

TABLE 2
Details of Actions

Action	Required Gas	Cost	On-chain Time Complexity	Off-chain Time Complexity
Approve a token	44643	0.003	$\Theta(1)$	$\Theta(1)$
Initialise a *PVnode*	144079	0.010	$\Theta(1)$	$O(n)$
Place a new order	>333629	0.024	$O(n)$	$O(n)$
Cancel an order	>88911	0.006	$O(n)$	$O(n)$

One of the aims of decentralisation is to to make the exchange as transparent as possible. A step to achieve this was made by verifying the smart contracts on a blockchain explorer called *Etherscan*. The next step is to make the GitHub repository open-source.

4.1.2 *Front end*

The UI was designed and coded in 100 hours. HTML, CSS and JavaScript were used. Many features were considered to increase the UX. DYDX majorly influenced all of the colour choices. Flash notifications, which popped up after every successful or failed action, were added. Tooltips were added for elaborating financial keywords. CSS was modified to make the website more responsive (e.g., mobile view).

Extended statistics like a price chart was going to be added, but this feature was not implemented due to limited time constraints.

4.2 Performing Tests

The exchange went through 18 unit tests, where 4 of these tests were stress tests, and succeeded without any errors. We aimed for zero functional bugs for maximum robustness. We mention the test cases that start with a sell order; however, both buying and selling tests were run. The following tests exclude the fee mechanism to provide a clearer explanation.

4.2.1 *Full Match*

This test is classified as *full match* since the sell size was equal to the buy size. A sell order of 2 ETH at 50 USD per ETH was placed. After placing the sell order, a buy order of 2 ETH at 50 USD per ETH was placed. After the buy order had been placed, the exchange occurred: the seller received 100 USD from the exchange smart contract, and the buyer received 2 ETH from the exchange smart contract. Therefore, no active order remained in the order book.

4.2.2 *Partial Match*

This test is classified as *partial match* since the buy size was lower than the sell size. A sell order of 2 ETH at 50 USD per ETH was placed. After placing the sell order, a buy order of 1 ETH at 50 USD per ETH was placed. After the buy order had been placed, the exchange occurred: the seller received 50 USD transferred from the exchange smart contract, and the buyer received 1 ETH transferred from the exchange smart contract. A single sell order of 1 ETH at 50 USD per ETH made by the seller remained in the order book.

4.2.3 *Reverse Partial Match*

This test is classified as *reverse partial match* since the buy size was greater than the sell size. A sell order of 2 ETH at 50 USD per ETH was placed. After placing the sell order, a

buy order of 4 ETH at 50 USD per ETH was placed. After the buy order had been placed, the exchange occurred: the seller received 100 USD from the exchange smart contract, and the buyer received 2 ETH from the exchange smart contract. A single buy order of 2 ETH at 50 USD per ETH made by the buyer remained in the order book.

4.2.4 Multiple Full Matches

This test is classified as *multiple full matches* since there was more than one sell order and the buy size was equal to the sell size. This test aws the first stress test. Two sell orders of 2 ETH at 50 USD per ETH were placed. After placing the sell order, a buy order of 4 ETH at 50 USD per ETH was placed. After the buy order had been placed, the exchange occurred: the seller received 200 USD from the exchange smart contract, and the buyer received 4 ETH from the exchange smart contract. Therefore, no active order remained in the order book.

4.2.5 Multiple Partial Matches

This test is classified as *multiple partial matches* since there was more than one sell order and the buy size was less than the sell size. Two sell orders of 2 ETH at 50 USD per ETH were placed. After placing the sell order, a buy order of 3 ETH at 50 USD per ETH was placed. After the buy order had been placed, the exchange occurred: the seller received 150 USD from the exchange smart contract, and the buyer received 3 ETH from the exchange smart contract. A single buy order of 1 ETH at 50 USD per ETH made by the seller remained in the order book.

4.2.6 Multiple Reverse Partial Matches

This test is classified as *multiple reverse partial matches* since there was more than one sell order and the buy size was greater than the sell size. Two sell orders of 2 ETH at 50 USD per ETH were placed. After placing the sell order, a buy order of 6 ETH at 50 USD per ETH was placed. After the buy order had been placed, the exchange occurred: the seller received 200 USD from the exchange smart contract, and the buyer received 4 ETH from the exchange smart contract. A single buy order of 2 ETH at 50 USD per ETH made by the buyer remained in the order book.

4.2.7 Rapid Full Matches

This test is classified as *rapid full matches* since there was more than one buy order and the buy size was equal to the sell size. This test was the second stress test. A sell order of 2 ETH at 50 USD per ETH was placed. After placing the sell order, two buy orders of 1 ETH at 50 USD per ETH were placed. After the buy order had been placed, the exchange occurred: the seller received 100 USD from the exchange smart contract, and the buyer received 2 ETH from the exchange smart contract. Therefore, no active order remained in the order book.

4.2.8 Overspending

The seller owned 2 ETH in their wallet. They tried to place a sell order of 5 ETH at 50 USD per ETH. The sell function in the smart contract returned an error since the amount of ETH that the seller owned was less than what they were trying to sell.

4.2.9 Cancelling an Order

The seller had no active sell orders. They tried to cancel a sell order which was placed by another seller. The cancel function in the smart contract returned an error since the function caller had not placed that sell order.

4.3 Deploying the DEX

In this paper, we initially planned that the app was going to be tested on a local copy of the Ethereum network. Instead of doing that, we used Remix. Remix has a feature where a local blockchain can be spun up. This was used since unit testing was more flexible on Remix.

After performing the unit tests, the smart contracts were deployed on the Rinkeby network. Rinkeby network is one of the Ethereum test networks (known as *testnets*). The reason why we chose the Rinkeby testnet was this testnet was started by the Ethereum team in 2017 [46], it is immune to spam attacks [46], and a new block is mined every 15 seconds [46].

The exchange smart contract was deployed on the Rinkeby testnet and the address of the smart contract is "0x2551b4246b6f25212a576d48f610b7e7b204dd42". The source code was verified on Etherscan. Etherscan is a block explorer where the source code of verified contracts can be viewed, and past interactions with the smart contract are saved. The verification process was done through Hardhat. The exchange smart contract's address can be queried on Etherscan, where the source code and the past interactions with the smart contract can be read in detail.

Git was used for version control. With Git, GitHub was used to save the progress on the cloud in case the local folder was lost. The progress is saved on a private folder (known as *repository*, *repo* in short) on GitHub called "on-chain-dex". A continuous integration and continuous deployment (CI/CD) flow was followed with Vercel. The reason for following a CI/CD flow was to automatically deploy the front end after every commit.

5 Evaluation

5.1 On-chain vs Off-chain

The project's initial aim was to implement every action on-chain, making it 100% on-chain exchange. Storing all the data on-chain would benefit not trusting anything but the exchange smart contract. However, while working on the linked list library, it was discovered that off-chain functions could have been implemented without the drawback of the data being altered. The reason for choosing to implement off-chain functions over on-chain functions is that every writing operation costs gas; thus, real money. The data not being modified was ensured by passing and checking extra arguments after invoking a function in a smart contract.

An example of this is when working with prices and volumes. Before placing an order, the price of that *PVnode* is required. This is needed since the total volume for that price will be increased. Instead of traversing all the *PVnode* on-chain to find the correct *PVnode*, it is traversed off-chain, reducing the action to 0 gas. Then, the index of that *PVnode* is provided as an argument. After all of these, the order is ready to be placed. When the arguments (price, index of the

price node, size of the order) are passed, the index of the price node is checked if it is correct. This check is done in the smart contract ensuring $\Theta(1)$ is spent on-chain.

5.2 Possible Optimisations

Some of the actions could have been better optimised in time complexity or required gas. When removing a node in the order book, the whole linked list is traversed. Instead of traversing the entire linked list on-chain, either (1) they could have been traversed off-chain or (2) a reverse linked list could have been implemented. Both of these solutions would reduce the on-chain time complexity from $O(n)$ to $\Theta(1)$. The drawback would be saving the index in a linked list resulting in higher overhead (i.e., gas costs).

5.3 Collecting Feedback

A friendly user experience was one of the aims of this paper. To ensure this, the exchange was tested by 5 traders who had a prior experience in trading on CEXs (i.e., Binance) and DEXs (i.e., Uniswap). Some of the verbal feedback we received are the following:

1) "Why are prices in the order book listed randomly?"
2) "I want to match the order in the order book very fast."
3) "I do not understand what *Limit Order* means."
4) "My order is too small. I cannot see the size."
5) "I like the flash notifications."
6) "The app does not work on my phone!"
7) "Why cannot I see my trade history?"
8) "Why do I have to click the sell button twice when selling?"

All of these problems were addressed, and only 2 were not solved.

5.3.1 Sorting the Orders

Due to not implementing heaps in the exchange smart contract, the prices were saved in first-come-first-serve order. Therefore, we sorted the prices (from highest to lowest if it was the buy-side, from lowest to highest if it was the sell-side) after retrieving them from the smart contract. The sorting algorithm we used was the built-in JavaScript sorting method. The sorting algorithm differs from browser to browser, but all take $O(n\ log\ n)$ time. Since we have 2 sides (buy and sell) in the order book, sorting is performed twice; however, the performance is not degraded massively since $2 * O(n\ log\ n) = O(n\ log\ n)$.

5.3.2 Faster Order Placement

To place trades faster, two users wanted the feature of clicking on the row to copy the limit price on the order book to the limit price input box (see Figure 21). We asked the users which exchange they saw this feature on, and they replied saying this exchange was Binance.

5.3.3 Tooltips

Tooltips were added for providing extra clarity (see Figure 22).

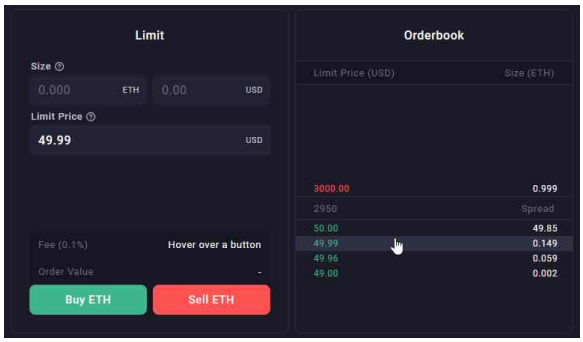

Fig. 21. A screenshot of the mouse clicking on the buy row with the price of 49.99 USD. After pressing the row, the price is copied to the limit price input box.

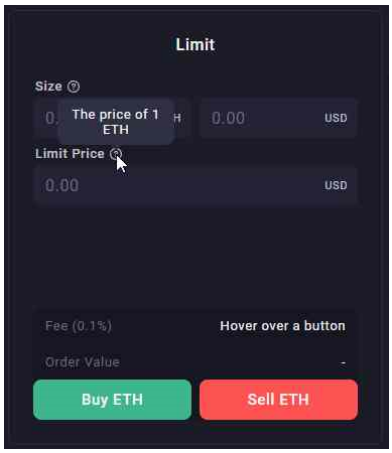

Fig. 22. A screenshot of the mouse hovering over a tooltip. The limit price tooltip shows "The price of 1 ETH".

5.3.4 Small Orders

Orders less than 0.001 ETH are currently displayed as $<$0.001 (see Figure 23). This was not the case since orders with size less than 0.001 would be rounded to 0.

Fig. 23. The active orders are included in the screenshot. The first row is a buy order with a size of $<$0.001 ETH at 49 USD. The second row is also a buy order with a size of 0.001 ETH at 49 USD.

5.3.5 Flash Notifications

Flash notifications are displayed after every action that involves calling the smart contracts (see Figure 24).

5.3.6 Responsiveness

The app was not designed for mobile view; however, after receiving multiple feedbacks surrounding this point, we designed a mobile view and coded it.

5.3.7 Trade History

The reason why we could not retrieve trade history was the lack of *events* in the exchange smart contract. Events work as

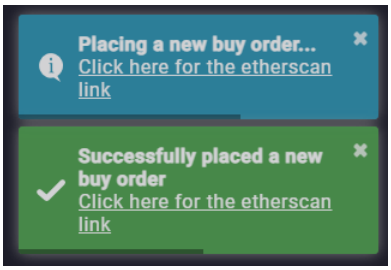

Fig. 24. The notification on the top is an "info" notification. It is displayed after every action. The notification on the bottom is a "success" notification. It is displayed after every successful action.

a log and store the passed arguments. These events could be added in the future in a new contract, and these events could be retrieved using the front end by querying the blockchain. However, this could be difficult since there are lots of events that need to be filtered.

5.3.8 Clicking the Trading Buttons Twice

When placing an order, two different confirmations are be asked: (1) initialising a *PVnode* and (2) placing the order. The reason for this is the design structure of the order book. The prices of different orders and sizes are saved in a list. If there have never been any orders for a particular price, that *PVnode* was not initialised; thus, the smart contract needs the user to initialise that particular *PVnode* with the given price. This can be fixed by the developer initialising all possible *PVnode*s in a certain range of price values when the apps is launched, so that regardless of the where the price is currently trading at, there is a *PVnode* available. This means that the user will only have to click on one button instead of two, increasing the UX, however this comes with the trade-off of paying a batch of initialisation fees.

5.4 Possible Extra Improvements

The overall user feedback was positive, and all 5 users said they would start using the exchange after it was deployed on a cheap network (e.g., layer 2 like Arbitrum One or a sidechain like Polygon), was audited by a smart contract company, and the pools had enough liquidity.

Another future improvement that was not mentioned was the front end. The front end does not use any framework making it less future proof and less clean. To elaborate, if a new front end developer is going to be hired in the future, they will spend a lot of time trying to understand the code, whereas if a framework were used, they would have understood it faster. As a result, the front end can be coded from scratch using React.

6 Conclusion

We designed and coded a universally accessible, order book style DEX that is user friendly.

The introduction covers the challenges of modern exchanges and the project objectives. Current exchanges are split into 2: (1) CEX and (2) DEX. The project objectives were divided into 3: (1) basic, (2) intermediate and (3) advanced. We completed all the basic and intermediate objectives and almost all advanced objectives.

In the methodology, we cover one order book protocol, but more optimisations (in terms of lower time complexity and functions that require lower gas) can be implemented. We also mention the design and the implementation process of the front end.

The most significant outcome in the results sections is the number of extensive tests. Many tests were performed to make sure the application was robust. Needless to say, the exchange passed all the tests.

In the evaluation, we mention how the exchange can be improved, what were the things we did correctly, what were the things we did wrong, and how we can solve these problems in the future.

This application is not deployed on the Ethereum mainnet network, making it only available on testnets for experimental purposes. In case of deployment on the Ethereum network, the smart contracts need to be audited by a security company specialised in Solidity to ensure no user action can be exploited.

Another step in the future would be to deploy it on layer 2 networks. These networks are built on top of the Ethereum mainnet network, making it cheap to deploy and operate. For comparison, the current price to swap tokens on Ethereum is 22.09 USD, whereas it costs as low as 1.04 USD on Optimism and Arbitrum One [47].

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, accessed on: Oct. 28, 2021. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] M. Singh, *Financial Crisis in Review*. Investopedia, 2021, accessed on: Oct. 30, 2021. [Online]. Available: https://www.investopedia.com/articles/economics/09/financial-crisis-review.asp

[3] A. Field, *What caused the Great Recession?* Insider, 2021, accessed on: Oct. 30, 2021. [Online]. Available: https://www.businessinsider.com/what-caused-the-great-recession?r=US&IR=T

[4] A. Rosic, *Blockchain Consensus: A Simple Explanation Anyone Can Understand*. Blockgeeks, 2020, accessed on: Oct. 31, 2021. [Online]. Available: https://blockgeeks.com/guides/blockchain-consensus/

[5] J. Frankenfield, *Peer-to-Peer (Virtual Currency)*. Investopedia, 2020, accessed on: Oct. 30, 2021. [Online]. Available: https://www.investopedia.com/terms/p/ptop.asp

[6] V. Buterin, "Ethereum whitepaper," 2021, accessed on: Oct. 31, 2021. [Online]. Available: https://ethereum.org/en/whitepaper/

[7] N. Szabo, "The idea of smart contracts. nick szabo's papers and concise tutorials," 1997, accessed on: Oct. 30, 2021. [Online]. Available: https://bit.ly/3EJOYb5

[8] C. Hackl, *What Are DAOs And Why You Should Pay Attention*. Forbes, 2021, accessed on: Oct. 31, 2021. [Online]. Available: https://www.forbes.com/sites/cathyhackl/2021/06/01/what-are-daos-and-why-you-should-pay-attention/

[9] B. CoinGecko, *Overall cryptocurrency market capitalization per week from July 2010 to October 2021*. Statista, 2021, accessed on: Oct. 31, 2021. [Online]. Available: https://www.statista.com/statistics/730876/cryptocurrency-maket-value/

[10] J. Tuwiner, "What was the mt. gox hack? buybitcoinworldwide," 2021, accessed on: Oct. 30, 2021. [Online]. Available: https://www.buybitcoinworldwide.com/mt-gox-hack/

[11] P. of Russia. (2022, Feb.) . Kremlin.ru. [Online]. Available: http://kremlin.ru/events/president/news/67828

[12] "Crypto platform blocks thousands of russia-linked wallets," *BBC News*, 2022, accessed on: Apr. 16, 2022. [Online]. Available: https://www.bbc.co.uk/news/technology-60661763

[13] R. Haar, "Binance.us review 2022: Low fees, but investors should take a pass," *NextAdvisor*, 2022, accessed on: Apr. 16, 2022. [Online]. Available: https://time.com/nextadvisor/investing/cryptocurrency/binance-us-review/

[14] *What is Uniswap? A 3-minute guide to the token swap exchange*. Decrpyt, 2020, accessed on: Oct. 31, 2021. [Online]. Available: https://bit.ly/3Moh9iv

[15] O. Fernau, *Uniswap Spearheaded the First AMM Era; its V3 May Usher in a New One*. The Defiant, 2021, accessed on: Oct. 31, 2021. [Online]. Available: https://bit.ly/3OzAleY

[16] D. P. Yi Zhang, Xiaohong Chen, "Formal specification of constant product (x * y = k) market maker model and implementation," 2018, accessed on: Apr. 16, 2022. [Online]. Available: https://github.com/runtimeverification/verified-smart-contracts/blob/uniswap/uniswap/x-y-k.pdf

[17] C. Younessi, *Uniswap — A Unique Exchange*. Medium, 2018, accessed on: Apr. 16, 2022. [Online]. Available: https://medium.com/scalar-capital/uniswap-a-unique-exchange-f4ef44f807bf

[18] "Uniswap v2 usdc-eth pair statistics," accessed on: Apr. 24, 2022. [Online]. Available: https://v2.info.uniswap.org/pair/0xb4e16d0168e52d35cacd2c6185b44281ec28c9dc

[19] "Erc-20 token standard," *Ethereum*, Dec. 2021. [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

[20] "Openzeppelin." [Online]. Available: https://openzeppelin.com/

[21] Hardhat. [Online]. Available: https://hardhat.org/

[22] "Remix ide." [Online]. Available: remix.ethereum.org

[23] A. J. Steve Ellis and S. Nazarov, "Chainlink a decentralized oracle network," Sep. 2017. [Online]. Available: https://whitepaper.io/document/387/chainlink-whitepaper

[24] S. Nakamoto, *Bitcoin open source implementation of P2P currency*. P2P Foundation forum, 2009. [Online]. Available: http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source

[25] *Development Technical Discussion in Bitcoin Forum*. Bitcoin Forum. [Online]. Available: https://bitcointalk.org/index.php?board=6.12280

[26] Y. B. Perez, *Mt Gox: The History of a Failed Bitcoin Exchange*. CoinDesk, 2015. [Online]. Available: https://www.coindesk.com/markets/2015/08/04/mt-gox-the-history-of-a-failed-bitcoin-exchange/

[27] M. K. Vaidya, *Computer Security - ESORICS 2014*. 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part I, 2014. [Online]. Available: https://books.google.co.uk/books?id=sOhKBAAAQBAJ&q=mt.+gox+70&pg=PA314&redir_esc=y#v=onepage&q=mt.\%20gox\%2070\&f=false

[28] B. M. Easy. How to create a mt. gox bitcoin trading account. Youtube. [Online]. Available: https://www.youtube.com/watch?v=CwFYKJxuqew

[29] "Localbitcoins." [Online]. Available: https://localbitcoins.com/about

[30] O. Knight, *Can Uniswap achieve what EtherDelta couldn't?'*. Yahoo Finance, 2020. [Online]. Available: https://finance.yahoo.com/news/uniswap-achieve-etherdelta-couldn-t-100043329.html

[31] "Ethereum mainnet statistics." [Online]. Available: https://ethernodes.org/

[32] "Ethereum node tracker." [Online]. Available: https://etherscan.io/nodetracker

[33] *The Uniswap V1 Protocol*. Uniswap Docs, 2018, accessed on: Oct. 31, 2021. [Online]. Available: https://hackmd.io/@HaydenAdams/HJ9jLsfTz?type=view

[34] *Introducing Uniswap V3*. Uniswap Blog, 2021, accessed on: Oct. 31, 2021. [Online]. Available: https://uniswap.org/blog/uniswap-v3/

[35] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais, "On the just-in-time discovery of profit-generating transactions in defi protocols," 2021. [Online]. Available: https://arxiv.org/abs/2103.02228

[36] CoinMarketCap, *Price of Bitcoin*. [Online]. Available: https://coinmarketcap.com/currencies/bitcoin/

[37] V. Loh. How to use binance exchange tutorial fully explained no music binancevideo. Youtube. [Online]. Available: https://www.youtube.com/watch?v=PULSPYH1t0U

[38] N. Hein, *Solidity vs Vyper*. QuickNode, 2022. [Online]. Available: https://www.quicknode.com/guides/vyper/solidity-vs-vyper

[39] trends.google.com. (2021) Google trends. Accessed on: Apr. 18, 2022. [Online]. Available: https://trends.google.com/trends/explore?cat=31&q=vyper,solidity

[40] OtherWayRound, *MetaMask Revenue and Usage Statistics*. OtherWayRound, 2022. [Online]. Available: https://otherwayround.net/metamask-statistics/

[41] "Ethers." [Online]. Available: https://docs.ethers.io/v5/

[42] OpenZeppelin, *ERC 20*. [Online]. Available: https://docs.openzeppelin.com/contracts/2.x/api/token/erc20

[43] ——, *Access Control*. [Online]. Available: https://docs.openzeppelin.com/contracts/2.x/access-control

[44] "Polygon gas tracker," accessed on: Apr. 22, 2022. [Online]. Available: https://polygonscan.com/gastracker

[45] CoinMarketCap, *Price of Polygon*, accessed on: Apr. 22, 2022. [Online]. Available: https://coinmarketcap.com/currencies/polygon/

[46] medvedev1088, "Comparison of the different testnets," 2017. [Online]. Available: https://ethereum.stackexchange.com/a/30072/79733

[47] "L2 fees," accessed on: Apr. 22, 2022. [Online]. Available: https://l2fees.info/