Eyethereum: Tools to Gain Insights into the Ethereum Blockchain.

Ruairi Casey

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8RZ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at the University of Glasgow

01/09/2023

Abstract

The emergence of Bitcoin in the aftermath of the 2008 financial crisis reshaped the financial landscape, introducing the world to the potential of blockchain technology and digital 'cryptocurrencies'. While Bitcoin established the blockchain industry, Ethereum expanded its scope through various innovations such as smart contracts and decentralised applications, giving rise to a new sector known as decentralised finance. In recent years, trading and investing in cryptocurrencies on platforms called decentralised exchanges has exploded in popularity. However, the space is largely unregulated and anonymous, providing opportunities for malicious actors. Consequently, scams on decentralised exchanges, like Uniswap, are pervasive. This project aims to address some of these challenges by developing tools for users to navigate the 'Wild West' of decentralised finance and the Ethereum blockchain in a safer, more informed way.

This dissertation details the development of a full-stack, blockchain-based web application called Eyethereum with two core features: a token checker, and a wallet tracker. The token checker aims to provide helpful information on a given cryptocurrency, and the wallet tracker aims to utilise the transparency of the blockchain to enable users to follow the activity of any Ethereum account. The requirements for Eyethereum were gathered through extensive background research, the study of existing competitors, and a survey completed by users with knowledge of and experience with Ethereum.

The application was developed primarily in TypeScript, using ReactJS and the NextJS framework for the front end; NodeJS and the incorporation of tools and libraries such as Ethers for the back end; and MongoDB for database integration. Eyethereum was tested comprehensively to validate the accuracy of the results, primarily through extensive comparison with leading competitors. Additionally, a beta testing survey was completed by a small number of trusted participants to gain feedback on the usability of the product and receive suggestions for improvement. A number of suggestions were able to be implemented, however there were still a range of improvements to be made. The final chapter of this dissertation summarises the achievements of the project in successfully delivering a minimum viable product (MVP) with a functioning token checker and wallet tracker, before detailing future work to build on the progress made.

The final version of Eyethereum was publicly deployed with Vercel, and is available at the following domain: https://eyethereum.vercel.app/

A video demonstration of Eyethereum was deployed on YouTube and is available at the following link: https://www.youtube.com/watch?v=DCr_dQFMs3Y

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Ruairi Casey Signature: Ruairi Casey

Acknowledgements

First and foremost, I would like to thank my supervisor, Sham, for allowing me to choose a project in an area that I have a strong interest in. Building a blockchain-based product from scratch and writing a dissertation on an emerging technology has been both incredibly challenging and rewarding. Ultimately, I have produced a final product that I am proud of, learning many lessons and new skills along the way. I would also like to thank my parents, Gary and Janet, and my girlfriend Caris. I would not have made it this far without any of you.

Contents

Chapter 1 Introduction

1.1 Bitcoin and the Blockchain

In the wake of the 2008 financial crisis, a digital currency called Bitcoin was created as an alternative to traditional monetary systems. The idea of Bitcoin began to spread after Satoshi Nakamoto, a mysterious, anonymous figure published a whitepaper titled "Bitcoin: A Peer-to-Peer Electronic Cash System." The paper describes the main purpose of Bitcoin as the ability to allow two willing parties to make online transactions directly without the need for a third party, such as a government or a bank, which inherently requires an element of trust in these intermediary institutions [1]. Satoshi argues that this is the root problem with traditional monetary systems and 'fiat' money, as historically, banking institutions have repeatedly breached this trust. Instead, Bitcoin transactions rely on cryptographic proof, a concept which popularised the term 'cryptocurrency' to refer to digital assets powered by blockchain technology. In addition to decentralization, other key features of Bitcoin include a limited supply, only 21 million Bitcoin can ever be mined, and privacy, allowing users to maintain a degree of anonymity despite all transactions being recorded publicly.

The technological foundation of Bitcoin is the blockchain. This underlying technology is a distributed ledger of transactions comparable to a public check book, where every transaction is publicly recorded and it is impossible to change previous entries. The blockchain can be described as a decentralised database made up of a series of 'blocks' which contain a list of transactions made within the block. Upon completion, each block is added to a 'chain' of previous blocks – the blockchain - which contains the entire transaction history of the system. The integrity and security of the Bitcoin blockchain is ensured by a consensus mechanism known as Proof-of-Work (POW). In this system, specialised participants called miners compete against each other to solve complex mathematical problems. When a miner solves one of these puzzles, all the transactions within the block are validated, the block is added to the blockchain, and the miners are rewarded with some Bitcoin, incentivizing their continued participation [2]. Bitcoin was the first application of blockchain technology, however many experts believe the use cases could extend well beyond the creation of cryptocurrencies, and wider potential applications are still being explored. For example, in 2019 the UK HM Land Registry put blockchain technology to the test. They simulated the purchase and sale of a house that had been completed in the real world on the 6th of March 2019. The simulation used video chat and blockchain and was completed in 10 minutes - compared to 22 weeks in the real-world example. This successfully demonstrated proof of concept and the HM Land Registry concluded that blockchain could enable quicker property transactions with a greater degree of trust, security, and transparency [3]. Furthermore, companies like IBM offer blockchain solutions for a variety of industries, including banking, healthcare, and government [4]. While Bitcoin pioneered the concept of cryptocurrencies and blockchains, subsequent projects, such as Ethereum, have expanded on this foundation.

1.2 Exploring Ethereum

Vitalik Buterin, the founder of Ethereum, published the Ethereum Whitepaper in 2014 and subsequently launched the project in 2015. Like Bitcoin, Ethereum is a blockchain with a native cryptocurrency, 'Ether' or 'ETH', however there are key differences between them. For example, while Bitcoin's primary purpose was to serve as an electronic cash system, Ethereum was developed to serve as a 'world computer', a global computing infrastructure that enables developers to build decentralised applications (dApps), and execute programs called smart contracts [5]. Furthermore, while Bitcoin relies on a Proof of Work consensus mechanism, Ethereum transitioned to Proof of Stake (POS) in 2022, a significantly less energy intensive system. In POS, specialised participants called validators are chosen in a lottery-style system to validate new blocks and add them to the blockchain. The more ETH a validator has 'staked' – locked up for an agreed duration – the more likely they are to be chosen to validate a block of transactions. Validators are rewarded with transaction fees in the form of ETH, and their staked capital is at risk of being destroyed if they attempt to defraud the network, together incentivising their honest participation in the system [6].

Just as a bank account has a unique card number or account number, every Ethereum account has a unique identifier referred to as an address, consisting of a set of 40 letters and numbers, prefixed by '0x'. Ethereum addresses may belong to individual users, or self-executing smart contracts. Both types have the ability of holding, sending, and receiving ETH or Ethereum-based tokens, and interacting with smart contracts [7]. An Ethereum address belonging to a user is often referred to as an externally owned account, or a wallet, allowing them to securely participate in the Ethereum network. There is no personal information required or inherently associated with an Ethereum account or wallet, providing a degree of privacy to users. Each wallet has an associated public key and private key which allow exclusive access to the wallet. Metamask is widely regarded as the leading cryptocurrency wallet provider with a self-reported 30 million users, available as a browser extension or a mobile application, and supporting multiple blockchains [8].

Transactions on Ethereum refer to actions taken by externally owned accounts that are transmitted and recorded by the Ethereum blockchain. Transactions are the only thing that can result in a state change in Ethereum's global singleton state. The simplest transaction is a basic transfer of ETH from one account to another. Just as I can transfer £10 to another person with their bank details, I can transfer any amount of ETH to another person if I know their Ethereum wallet address. Each transaction has various properties, including but not limited to: the 'from' address; the 'to' address; the 'value' in ETH; a transaction fee paid in ETH, commonly referred to as the 'gas' fee; and a transaction 'hash', a unique identifier that is cryptographically generated upon completion [9]. All transactions on Ethereum are permanently and publicly recorded on the blockchain, making it possible to track all activity that occurs on the chain. Etherscan is a web based blockchain explorer which makes use of this transparency, recording all activity that occurs on Ethereum and displaying it to the public in a readable, searchable format [10]. Theoretically, if an aspiring trader found an Ethereum wallet with a 100% win rate, they could follow its activity on Etherscan. The wallet tracker feature of Eyethereum aims to simplify this process in an organised, user-friendly way.

A core element at the heart of the Ethereum ecosystem is the Ethereum Virtual Machine (EVM). Somewhat analogous in certain respects to the Java Virtual Machine [11], the EVM is a runtime environment that: provides a layer of abstraction, eliminating the need for developers to write machine-specific code; executes EVM bytecode that is compiled from specialised programming languages such as Solidity; and allows smart contracts to be executed in isolation from the rest of the network, ensuring that they run in an efficient and secure way without any interference [12, 13]. As discussed previously, smart contracts are a type of Ethereum account that are not controlled by a user, however they must be created and deployed by a user with a special contract creation transaction before they can execute [14]. Smart contracts are essentially computer programs consisting of code and data, that self-execute on the Ethereum blockchain exactly as programmed. Once created, smart contracts cannot be deleted, and any interaction with them cannot be reversed. Smart contracts are often compared to digital vending machines, where specific inputs and requirements being met results in a pre-determined, immutable output, eliminating the need for an intermediary [15]. Smart contract code is typically written in a specialised high-level programming language, of which Solidity is the most widely used [16]. An application binary interface (ABI) defines how code and data are accessed by machine code, as opposed to an application programming interface (API) which defines this access in human-readable formats known as source code. In Ethereum, smart contract ABI's define the callable functions within the smart contract, the arguments they accept, and the results they will return, and are responsible for encoding and decoding data into and out of machine code for the EVM. Crucially, the creator of the smart contract has the exclusive ability to call functions that can change its behaviour [14]. Smart contracts have various applications, including the creation of Ethereum-based tokens. Anyone can create and deploy a smart contract, so long as they have enough ETH to pay the transaction fees. One of the implications of this is that anyone can create an Ethereum token anonymously, in a largely unregulated industry. Consequently, scams are widespread in the decentralised finance space, and this is the problem that the token checker aims to address.

1.3 Aims and Objectives

The aim of the project is to build a web application consisting of two main features: an Ethereum token checker, which will allow users to receive various information which may help to inform them on the likelihood that it is a scam; and an Ethereum wallet tracker, which will enable users to create a curated pool of tracked Ethereum wallets to monitor and receive live updates for. The overarching ambition behind both functionalities is to provide users with tools to navigate the Ethereum blockchain in a more informed way.

1.4 Project Overview

Chapter 2 explores the background behind the problem the product aims to address in greater depth and details the requirements gathering process. Chapter 3 documents the project's development and implementation. Chapter 4 details the testing of the product accompanied by a critical evaluation. Chapter 5 reflects on the achievements and challenges of the project and explores potential areas for future work.

Chapter 2 Analysis and Requirements

2.1 Background

Ethereum supports the creation of tokens that can be categorised into two main types based on their fungibility – ERC-20 and ERC-721. Commonly referred to as non-fungible tokens (NFTs), ERC-721 tokens are unique in nature [17]. Each NFT has a 'tokenId' property which serves as a globally unique identifier. An NFT is not directly interchangeable with any other NFT, even if they belong to the same collection. This property of NFTs allows them to represent ownership of a wide range of digital and non-digital assets, leading to adoption in areas such as art, real estate, and even legal identification [18, 19, 20]. In contrast, ERC-20 tokens are fungible, meaning each token belonging to the same smart contract are identical and directly interchangeable [21]. Just as one £1 coin can be directly exchanged for and maintains the same value with another £1 coin, for example. There are many ERC-20 tokens, with a recent high-profile example being the announcement by PayPal of the launch of their own 'stablecoin', a digital asset pegged to the price of 1 US dollar, on Ethereum [22]. The fungible nature of ERC-20 tokens enables them to be traded on various platforms, such as centralised and decentralised exchanges.

Price speculation is an undeniable aspect of cryptocurrencies, which have garnered a reputation as being a 'high risk, high reward' asset class due to their volatility and in some cases, large returns on investment. For example, Ethereum itself held an initial coin offering (ICO) from July 22nd 2014 until September 2nd 2014 which was open to the public and only available to buy with Bitcoin. The price started at 2000 ETH per BTC and gradually declined to 1337 ETH per BTC by the end of the ICO. At the time, this resulted in an average price of around $0.31 per ETH token [23, 24, 25]. At the time of writing, the price of one ETH token is approximately $1840 [10], representing a price increase of over 59,000% from the ICO. Such historical returns on investment have attracted traders and investors to the cryptocurrency industry over the years, and the two primary vehicles for their participation are cryptocurrency exchanges.

As mentioned previously, cryptocurrency exchanges fall into two distinct categories, centralised and decentralised exchanges (CEXs and DEXs). CEXs operate with a central authority that oversees operations and typically require customers to provide personal information and legal, government-issued identification to participate, a process referred to as 'know your customer' or KYC [26, 27]. On the other hand, DEXs operate without a trusted third-party intermediary and do not require any personal information. All that is required for user participation is that they have a crypto wallet that can connect to the DEX's site. Advantages of DEXs include transparency and non-custody, meaning every trade is publicly verifiable and the DEX is never in possession of a user's funds [28]. The leading DEX platform, Uniswap, uses an automated market maker (AMM) protocol which uses smart contracts and an algorithm to pool liquidity and enable the permissionless, peer-to-peer trading of cryptocurrency tokens [29, 30]. In 2020, Uniswap had a trading volume of over $58 billion, up from $390 million in 2019 [31]. Since then, the platform has continued to grow, regularly outperforming Coinbase, one of the largest CEXs, this year with

trading volume as high as $70 billion in March [32]. Uniswap currently supports 8 blockchains [33], however the vast majority of the platforms volume comes from Ethereum, more specifically the trading of ERC-20 tokens. Furthermore, according to Dune analytics, Uniswap V2 has approximately 50% more monthly active unique users than V3, and over double the quantity of trades [34]. The trading of ERC-20 tokens on Uniswap V2 will therefore be the focal point of the functionalities implemented in this project.

As well as traders, investors and innovators, the cryptocurrency industry has also attracted a myriad of bad actors seeking to exploit it. Since its inception, the industry has been plagued by high profile scams, with a recent example being the collapse of FTX, at the time one of the top CEXs. The US Commodity Futures Trading Commission (CTFC) charged FTX with fraud in December 2022 and estimated that over $8 billion in customer funds were lost [35]. According to a Crypto Crime Report published by Chainalysis, an estimated $9.7 billion was lost to scams and theft alone in 2022 [36]. The anonymous, permissionless nature of Uniswap and the Ethereum blockchain means that virtually anyone can create an ERC-20 token and list it on Uniswap. This has provided new opportunities for traders, investors, and unfortunately, malicious actors. To illustrate the scale of this problem, a landmark study published in 2022 by researchers from the University of Pompeu Fabra and the University of Barcelona analysed all tokens listed on Uniswap from 04/05/2020 to 03/09/2021. Out of 27,588 tokens in the data set, 26,957 were labelled as scams or 'rug pulls' - a staggering 97.7% [37]. Faced with such overwhelming odds, navigating the 'Wild West' of DeFi safely can be a daunting prospect, especially for newcomers. However, equipping oneself with the knowledge and tools to identify and avoid scams can help.

Once a malicious actor has created an ERC-20 token contract, they can list it on Uniswap by providing liquidity, typically in the form of ETH. In doing so they create a liquidity 'pool' consisting of the token and ETH, thus making the token tradable. Once the token is live on Uniswap, there are two main types of scams they can employ, rug pulls and honeypots. A rug pull is characterised by the token creator who provided the liquidity, removing the liquidity. When a user swaps ETH for a token on Uniswap, all that ETH – excluding the transaction gas fee, Uniswap's swap fee, and any potential 'buy tax' - gets added to the liquidity pool, which is under the control of the creator. Therefore, if the token creator removes the liquidity from Uniswap after investors have bought the token, they not only render the token untradable, but also steal the ETH that was used to buy it in the process. To instil trust and prevent rug pulls, projects often 'lock' or 'burn' the liquidity. Locking the liquidity makes it inaccessible to the owner for a specified period of time, and burning the liquidity involves sending it to the 'dead' or 'zero' addresses, making it permanently unrecoverable [38]. In a typical honeypot, the owner will create the token to be sellable initially, and then after investors have bought the token, they will call a function that renders holders unable to transfer the token from one address to another and thus unable to sell it. Sometimes, the token will still technically be sellable, but the owner will call a function that increases the 'sell tax' on each sell transaction to a very high number. For example, if the owner set the sell tax to 99%, when a user sells 1 ETH worth of the token, their wallet would receive 0.01 ETH and 0.99 ETH would be transferred to a wallet specified by the owner. To prevent the owner calling malicious functions, ownership of a smart contract can be 'renounced' by transferring the ownership to the dead or zero addresses [39].

2.2 Similar Existing Products

2.2.1 Honeypot.is

Honeypot.is simulates buy and sell transactions to determine whether a token is a honeypot or not and highlight any potential anomalies. The website has a user-friendly interface with a long rectangular box to input a token contract address, and a 'check for honeypot' button to return the results. Currently the only supported blockchains are Ethereum and Binance Smart Chain (BSC) [40].

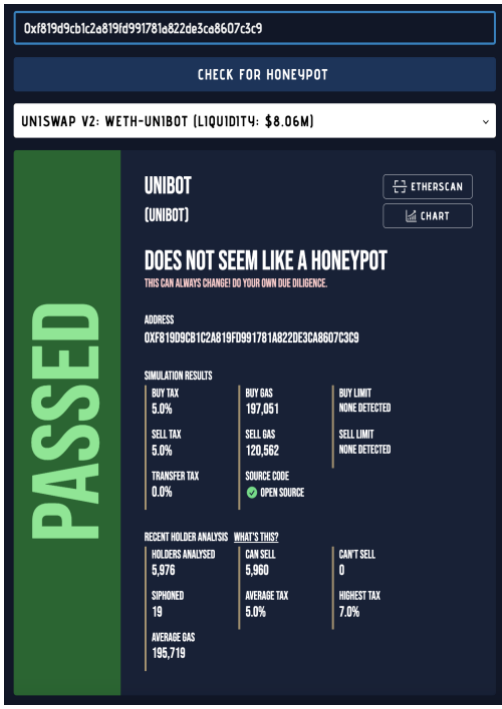

Figure 2.1: Ethereum token passes the honeypot detector.

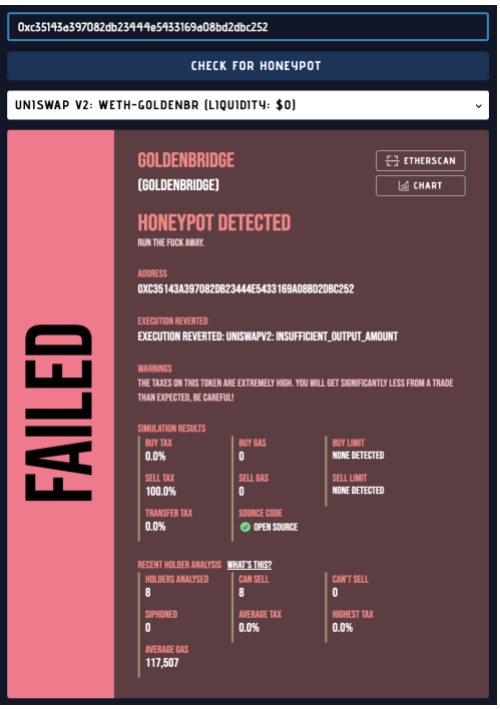

Figure 2.2: Ethereum token fails the honeypot detector.

The results display a large green or red banner to clearly indicate whether the test was passed or failed. Additional information is provided, including: buy tax, sell tax, transfer tax, buy gas, sell gas, buy limit, sell limit, and whether the source code is open or closed, also called verified or unverified. Furthermore, buy and sell simulations are run for each holder and the average tax, along with the number of holders who cannot sell is displayed – a possible reason for this is to detect 'blacklisting'. For example, out of all a token's holders, if some wallets can sell but some cannot, it suggests the owner has called a 'blacklist' function, only allowing some wallets to sell, and rendering the token a honeypot for others.

2.2.2 Token Sniffer

Token Sniffer is a comprehensive smart contract checker that supports 13 blockchains, including Ethereum [41]. The popular web application has been cited in official testimony on the floor of the US Senate Committee on Banking, Housing, and Urban Affairs [42]. The website has a text box labelled 'Search tokens by name or address'. When a contract address or name is entered, a drop-down list of matching tokens is displayed, which a user must then click to see the results.

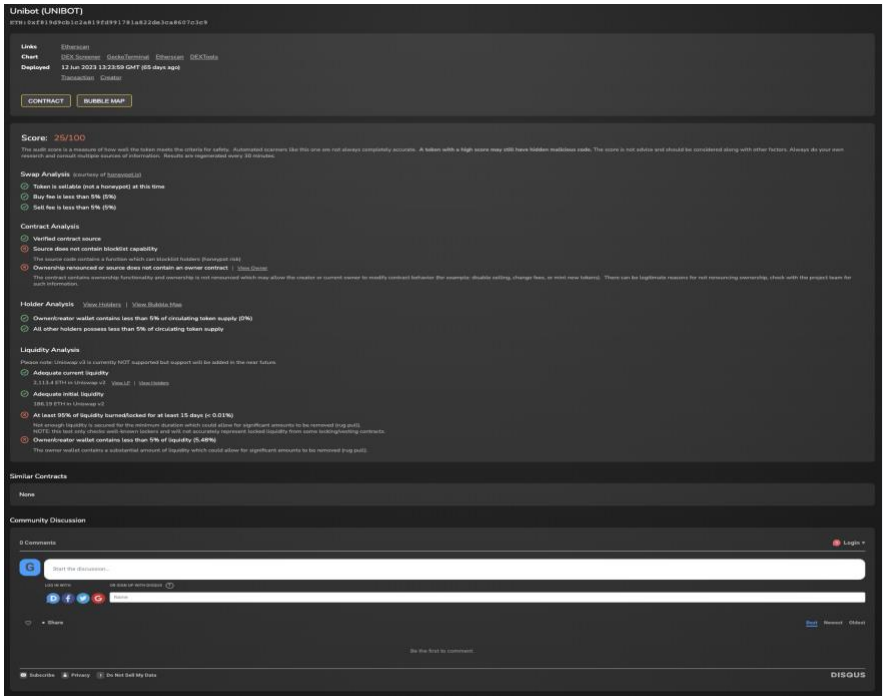

Figure 2.3: Token Sniffer results for an Ethereum token.

The result displays output including an overall score out of 100, swap analysis, contract analysis, holder analysis, liquidity analysis, similar contracts, and community discussion. The swap analysis results use Honeypot.is on the backend, returning honeypot status, buy tax and sell tax. Contract analysis returns whether the smart contract source code is verified, the code contains blacklisting functions, and the ownership has been renounced. Holder analysis returns the percentage of the supply held by the owner, and whether any wallet holds over 5%, which would be dangerously high. The liquidity analysis returns the current and initial liquidity value in ETH, and the percentage of the liquidity that has been burnt or locked by the owner. It also notes that Uniswap V3 is not supported yet. Finally, any similar contracts are identified and there is a section where users can add any comments. Importantly, both Token Sniffer and Honeypot.is make clear that the results are never conclusive and that users should always exercise caution and do their own due diligence.

2.2.3 Etherdrops Tracking Bot

EtherDrops is a Telegram Bot with various functionalities, including wallet tracking, supporting 8 blockchains [43]. The wallet tracker allows users to add or remove wallet addresses to track, give custom names to each wallet, filter what type of transactions they want to track, and receive real time notifications via Telegram messages for each transaction.

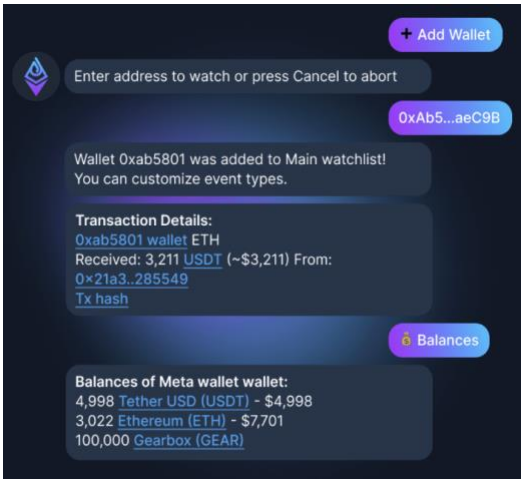

Figure 2.4: Visual representation of Etherdrops' Wallet Tracker feature.

2.3 Requirements Gathering Survey

Following on from extensive research of existing products, a user survey was carried out by creating a Google form and sharing it in the Discord communities of two major Web3 educational platforms, Alchemy University and LearnWeb3 [44, 45]. The form was to be completed anonymously with the purpose of informing the MoSCoW statements for each feature, and providing suggestions for any extra features that were not mentioned. Eyethereum's MoSCoW statements and user stories can be found in Appendices A and B. A link to the survey can be found in Appendix E and full results can be found in the Supplementary Material submitted in conjunction with this dissertation.

2.4 Ordered List of Features and Justification

2.4.1 Token Checker

- **Contract verification** If the token smart contract is not verified, then the smart contract code cannot be read. It is therefore impossible to detect any malicious functions in the contract.
- **Ownership renounced:** If the ownership of a token contract is not renounced, the user may be able to call malicious functions.
- **Liquidity locked or burnt:** If the liquidity is not locked or burnt, the owner can remove it at any point and rug pull.
- **Honeypot status:** Honeypot status it is essential to determine whether the token is sellable or not.
- **Buy and sell transaction taxes:** Without knowing the buy and sell taxes, a user may unknowingly lose their funds in either transaction.

2.4.2 Wallet Tracker

- **Create pool:** Before a user is tracking any wallets, they must be able to create a new 'pool' of tracked wallets to add to and remove from, exclusively linked to their Eyethereum account.
- **Add and remove wallets from pool:** Users must be able to add new wallets to their tracked pool, as well as remove existing wallets from the pool.
- **Delete pool:** Users should have the option to remove all the wallets from their pool in one action and start again.
- **Enable Tracking:** Users must be able to enable tracking of their pool of selected wallets.
- **Receive live and historical updates:** Users must receive real-time updates on the website, including live transactions and transactions made since they last checked.

2.4.3 General Features

- **Authentication:** Users must be able to register an account, log in, and log out. This enables a personalised user experience for the wallet tracker.
- **Deployment:** Eyethereum must be deployed on a public internet domain to enhance accessibility for users.
- **User Interface:** Eyethereum must have a consistent colour theme and an intuitive, user-friendly design.

Chapter 3 Design and Implementation

3.1 System Architecture

The development of Eyethereum, a full-stack web application, involved four main layers: client, server, database, and external APIs.

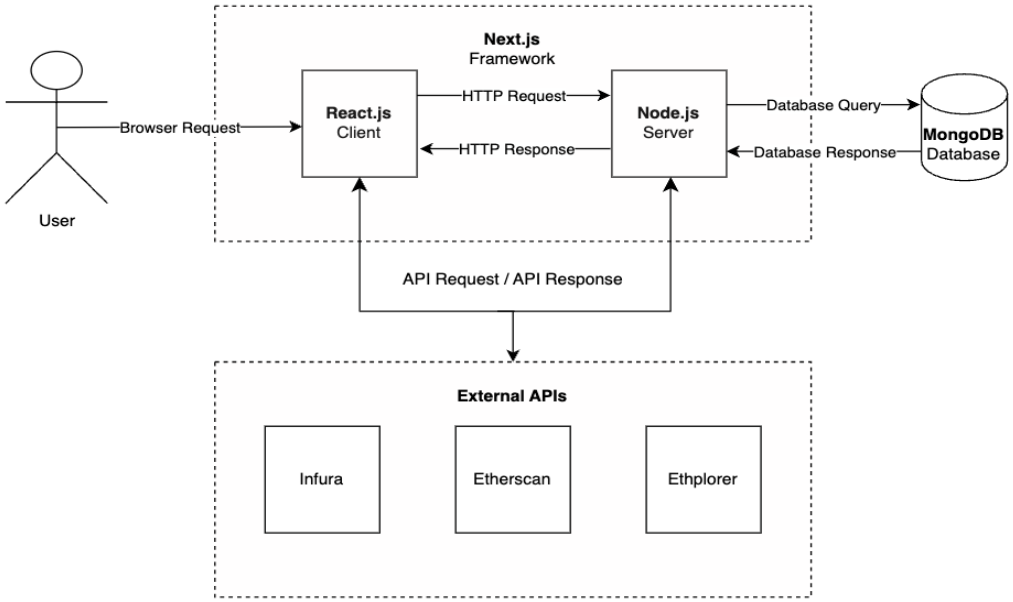

Figure 3.1: System Architecture Diagram.

Eyethereum was primarily written in the programming language TypeScript, a superset of JavaScript. TypeScript retains the many advantages of JavaScript, such as versatility and ubiquity in web development, while also conferring additional features such as type safety and enhanced refactoring capabilities and readability. This enables programmers to write code that is more maintainable and understandable before it is transpiled to JavaScript for execution [46]. Node.js is a mature and widely used runtime environment that allows for the execution of JavaScript code on the server-side layer [47]. Node.js provides efficiency, scalability, and access the largest software registry in the world, the 'npm' ecosystem of libraries and tools, such as Ethers and Ganache, which were integral to the development process in this project [48].

React.js is a popular JavaScript library for front end development which allows for the creation of reusable user interface components, making development more scalable and maintainable [49]. Furthermore, React has built-in features known as 'hooks', such as useEffect and useState, which offer enhanced state management capabilities and ensure a smooth user experience by keeping the UI in sync with the desired underlying state [50, 51]. Next.js is a tailor-made framework for web development with React that offers features such as client and server-side rendering, data fetching, and API route handling [52]. The Next.js framework was selected for various advantages such as faster page loading, more responsive user experience, and the simplification of API routing endpoints. The API routing features of Next were utilised extensively in the wallet tracker functionality when interacting with the MongoDB Atlas database.

3.2 Database Design

Structured Query Language (SQL) is a programming language that was designed to allow developers to create, manage, and interact with relational databases. SQL databases such as MySQL, also referred to as relational databases, store data in structured tables which must adhere to a strict pre-defined schema. Furthermore, normalisation processes must be undertaken to avoid redundancy and ensure integrity. On the other hand, NoSQL databases, also referred to as non-relational databases, can store data without requiring a rigid schema or normalisation processes, and support a variety of different formats, from graph-based to document-based [53]. NoSQL databases represent an alternative that offers greater flexibility, quicker set up, and are particularly advantageous when dealing with a high volume of write operations or rapidly changing data. Given the dynamic nature of the data Eyethereum interacts with, the frequent write operations associated with the wallet tracker, the time constraints and continuously evolving nature of development, a NoSQL database was determined to be the best solution for this project, specifically MongoDB.

MongoDB is a leading document-oriented NoSQL database which provides high speed performance and a cloud-based database solution, Atlas [54]. Atlas eliminates operational requirements involved in the setting up, configuration and maintenance of local databases, and allows for greater scalability. Atlas also has built in features which ensure compliance with high standards of data privacy and security [55]. Furthermore, MongoDB is highly compatible with JavaScript development. Firstly, it stores data in a JSON-like format, BSON. This facilitates seamless read and write data operations between the client and the server without any need for parsing [56]. Secondly, the availability of an official npm library 'mongodb' provides a rich set of features to facilitate the direct interaction between Node.js applications and MongoDB databases [57]. This ensures developers can rely on official documentation with consistency and up-to-date features. In summary, this multitude of features made MongoDB an ideal choice for the database infrastructure of Eyethereum.

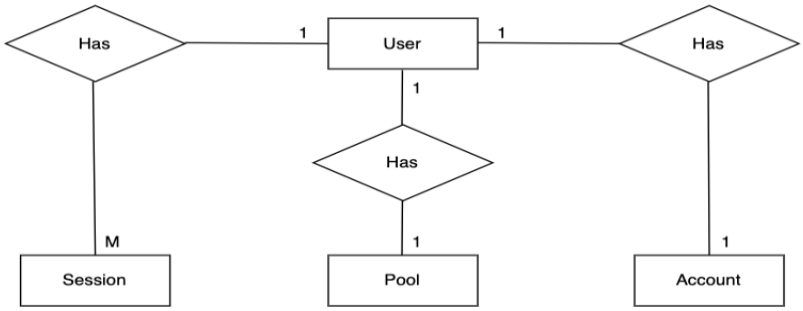

Figure 3.2: ER Diagram in Compressed Chen Notation.

As illustrated in Figure 3.2, each session is uniquely associated with one user, however over time a user can have multiple sessions. A user can only have one account and one pool, and they in turn can only belong to one user. It is important to note that, as a non-relational database, this is a conceptual representation, and the schema is dynamic and flexible. The application code ensures that certain relationships are consistent as the database itself does not enforce them. For example, when a user deletes a pool their 'poolId' attribute is reset to an empty value, 'poolCreated' is set to false and the pool document is deleted. Full attributes for each entity can be found in Appendix C.

3.3 Tools and APIs

Ethers.js: Ethers is a comprehensive npm library, written in TypeScript, for interacting with the Ethereum blockchain [58, 59]. Ethers offers a wide range of functionalities such as unit formatting, smart contract creation, event listeners, and transaction analysis and execution. A 'provider' is an Ethers object which enables read-only interactions with the blockchain. To execute write operations that change the state of the blockchain a 'signer' object is required, which must be instantiated with a provider.

Infura API: Infura offers a range of tools for Web3 development, including Ethereum nodes as a service [60]. A JSON-RPC is a remote procedure call which uses JSON as a shared format for requests and responses, facilitating communication between different software systems [61]. In this context, it serves as an interface for interacting with the Ethereum network. Infura's JSON-RPC API enables developers to read from and write to the Ethereum blockchain in real time without setting up and running their own node. A free API key was generated for Eyethereum and used to instantiate an Ethers provider object to query the Ethereum 'mainnet', the real blockchain in its current state.

Ganache: Ganache is a blockchain simulator tool developed by Truffle Suite [62]. This enables developers to build and test dApps and smart contracts in a safe, virtual environment without the necessity of using and risk of losing any personal funds. Ganache offers a range of powerful features such as: forking the current state of the Ethereum mainnet; impersonating any real or virtually created Ethereum account; JSON-RPC support, allowing it to serve as an Ethers provider; and an npm library which enables programmatic use in Node.js. A Ganache provider was used to instantiate an Ethers 'signer' object. This enabled the simulation of buy and sell transactions from virtually created wallets to determine the buy and sell taxes and honeypot status of a token.

Etherscan API: Etherscan is the leading blockchain explorer for Ethereum [10]. The API provides endpoints for developers to analyse Ethereum data such as transactions and smart contracts [63]. Eyethereum used the API to check if a smart contract was verified.

Ethplorer API: Ethplorer is an 'all-in-one platform for the world of the Ethereum blockchain' that offers a set of tools and APIs [64]. Eyethereum used Ethplorer's purpose built Bulk Monitor API free plan to implement the wallet tracker functionality [65].

Axios: Axios is a promise-based HTTP client for JavaScript which can be used in the browser and in Node.js programs, available as an npm library [66]. In Eyethereum, Axios was used to execute GET and POST calls to the Ethplorer and Etherscan APIs.

qs: qs is an npm library that enables the parsing and stringifying of query strings to handle complex data during API interactions [67]. Eyethereum used qs to stringify data for Ethplorer API calls into the correct format, 'x-www-form-urlencoded' content type.

3.4 Authentication and MongoDB Configuration

Eyethereum implemented the NextAuth.js example template for authentication. NextAuth.js is a runtime-agnostic, open-source authentication solution for Next.js applications with built in session management and sign-in support for popular providers such as Google, as well as password-less email sign in, which was configured for this project. There is also built in support for popular database providers, including MongoDB [68]. The project was initiated by following the instructions on the official NextAuth GitHub to set up the next-auth-example template and configure MongoDB [69, 70, 71]. Another advantage of starting with this template is that it comes with example files to demonstrate various Next.js features, such as how to handle API routes and sessions.

3.5 Token Checker

3.5.1 Contract ABIs

The token checker functionality was implemented in the 'token-checker.tsx' file. The first step was to import the necessary tools and libraries and initialise the constants and variables required for development. This included the ABIs for the Uniswap V2 'router' and 'factory', two of the core smart contracts that power the Uniswap protocol. The factory is responsible for handling liquidity pairs, and the router is responsible for facilitating swaps and other trading functionalities [72]. The addresses for each smart contract were sourced from the official Uniswap documentation [73, 74]. Subsequently, each contract address was searched for on Etherscan where the respective ABIs were then copied and saved in a dedicated 'abi' directory as '.json' files. This allowed them to be imported into the 'token-checker.tsx' file. Furthermore, the ABI for a basic ERC-20 token was initialised as an array of strings in the `erc20ABI` constant [75]. An illustration of the ERC-20 functions called in this project are shown in Table 3.1. These steps were required as in Ethers, a new `Contract` object must be passed an ABI parameter to be instantiated, and this determines what functions it can call in the program.

ERC-20 Standard Functions	Description
`totalSupply`	Returns the total units of a token.
`balanceOf`	Returns the token balance of a given address for a given token.
`approve`	Authorises a given address to execute transactions for a given token.
`name`	Returns the name, eg USD Coin.
`symbol`	Returns the symbol, eg USDC.
`decimals`	Returns the decimals used to divide token units to its user-readable form.

Table 3.1: ERC-20 ABI functions utilised in Eyethereum's implementation [75].

3.5.2 Verified Status

The verification status of a token is determined by the `isContractVerified` function which interacts with the Etherscan API. This function and all similar token checker functions take a `contractAddress` parameter representing the Ethereum address of the token being checked by the user. A `url` string to get the

contract ABI from Etherscan is constructed with the `contractAddress` and Etherscan API key and passed as a parameter to an Axios GET request. A successful request returns a response with a status of '1' and a message of 'OK', meaning the token contract is verified [76]. Therefore, if both conditions are met, then the contract is determined to be verified, and vice versa.

3.5.3 Ownership Renounced

To assess the ownership status of the token, the `isOwnershipRenounced` function first creates a new Ethers `Contract` object [77]. Subsequently, the ERC-20 function `owner()` is called and the result is saved in `owner` [75]. A user can renounce ownership of a smart contract by transferring ownership to the 'dead' or 'zero' addresses, Ethereum accounts specifically created to have no owner and be incapable of executing any transactions [78, 79]. If `owner` is either of these addresses, ownership is determined to be renounced, and vice versa.

3.5.4 Liquidity Burnt or Locked Percentage

To determine the percentage of a token's liquidity that has been 'burnt', permanently removing control from the owner, the `getBurnPercentage` function was designed. This function firstly instantiates new Ethers `Contract` objects for the Uniswap V2 factory and router with their respective ABIs passed as parameters. The factory ABI's `getPair` function is then called to retrieve the Ethereum address of the Uniswap liquidity pool (LP) tokens for the ERC-20 token being checked [73]. This address is then used to instantiate a new Ethers Contract, this time passing with the basic ERC-20 ABI as a parameter. This allows the `totalSupply` function to be called, followed by the `balanceOf` function to find the total supply of LP tokens and the amount held by the 'zero' or 'dead' addresses, respectively [75]. This data is then formatted so it can be used in mathematical operations and used to calculate the `burnPercentage`, which is then returned by the function.

The same logic is followed in the `getLockPercentage` function, with the primary distinction being that the final calculation is based on the balance of LP tokens held by the designated smart contract addresses for Unicrypt and Team Finance, the two leading liquidity lockers in the DeFi ecosystem [37, 80, 81].

3.5.5 Honeypot Status and Buy and Sell Taxes

To determine the honeypot status and buy and sell taxes of given token, the `simulateSwap` function simulates a buy and sell transaction on Uniswap V2 using the Ganache blockchain simulator. When the program is executed, Ganache generates 10 new virtual wallets each funded with 100 ETH [62]. Firstly, this array of accounts is retrieved from the Ganache `Provider` and the first account in the array is isolated as `address1`, which is then used to instantiate a new `Signer` object [82].

The next step is to instantiate new Ethers `Contract` objects for the Uniswap V2 router and the token being checked, with the newly created `Signer` object passed as the last parameter, thus enabling them to execute transactions within the

program. The 'expected amount' of tokens to be received from a buy transaction of 0.1 ETH – without any taxes – is then found by using the `getAmountsOut` function of the Uniswap V2 router contract [83].

Subsequently, a buy transaction is executed by calling the `swapExactETHForTokensSupportingFeeOnTransferTokens` function of the Uniswap V2 router contract with the 'to' address set to `address1` and the ETH 'value' set to 0.1 [74]. The `buyTaxPercentage` is then calculated by finding the difference between the expected amount out and the final balance of the token held by `address1` after the simulated buy transaction has completed, then dividing the result by the expected amount out and multiplying by 100.

The function then calculates the sell tax, firstly ensuring the token can be sold by calling the `approve` function [75]. The expected amount of ETH to be received by selling the full token balance held by `address1` - without any sell tax - is then calculated using `getAmountsOut` with different parameters. Subsequently, the `swapExactETHForTokensSupportingFeeOnTransferTokens` function is called with the 'to' address set to `address1` and the 'value' set to its token balance [74]. The difference between the ETH balance of `address1` before and after the sell transaction is then calculated and saved as `accountIncrease`. Importantly, a gas fee is deducted from the ETH balance of `address1` during the execution of the sell transaction. This value is extracted from the Ethers `transactionReceipt` and added to `accountIncrease` to find `inreasePlusFee` [84]. Without accounting for the gas fee, the sell tax calculation may be inaccurate. Finally, the `sellTaxPercentage` is calculated by finding the difference between the expected ETH amount out and `inreasePlusFee`, dividing the result by the expected ETH amount out, and multiplying by 100.

At the beginning of the function a Boolean variable to denote honeypot status, `HP`, is set to false. If the buy or sell tax is greater than 90% or the sell transaction fails, `HP` is set to true. If the buy or sell tax is between 10% and 90%, a warning message is issued to the user, but `HP` remains false. In any condition, the function returns an array containing `buyTaxPercentageString`, `sellTaxPercentageString`, and `HP` as the values which are then displayed on the front-end.

3.6 Wallet Tracker

3.6.1 Ethplorer API Calls

The Ethplorer Bulk Monitor API allows users to monitor multiple Ethereum addresses simultaneously and offers a range of endpoints. Eyethereum primarily uses the following POST endpoints: 'Create pool' initialises a new empty pool and returns a unique 'poolId' which is required for all subsequent API calls; 'Delete pool' removes an existing pool; 'Add addresses to pool' adds specified addresses to an existing pool; 'Remove addresses from pool' deletes specified addresses from an existing pool; and 'Clear pool' removes all addresses from a given pool. The only GET endpoint used in Eyethereum is 'Get pool updates' which returns a list of transactions and operations made by the tracked wallets in the pool within a

given period of time, with a maximum 'period' of 2592000 seconds, equivalent to 30 days. In the context of the Ethplorer API, 'transactions' refer to basic ETH transfers and 'operations' refer to any transactions involving ERC-20 tokens such as basic transfers, or swaps on Uniswap [65].

Corresponding functions for each endpoint were created in the 'wallet-tracker.tsx' file, using the axios library to make HTTP requests to the API and the qs library to stringify the request data to adhere to the format required by Ethplorer.

3.6.2 MongoDB Integration

Eyethereum's wallet tracker seamlessly integrates with MongoDB by utilising Next.js API routes to facilitate read and write operations to the database. This integration is crucial to the wallet tracker functionality as it enables multiple users to maintain their own unique pool of tracked wallets which persists across multiple sessions, by associating a user with a poolID and a pool with a userID. The application code ensures that the Ethplorer pool data is kept consistent with the corresponding pool collections in the database and there are no discrepancies between them.

The 'wallet-tracker.tsx' file contains a multitude of functions that perform read and write operations with the database by sending POST requests to internal API routes in the 'pages/api' directory. On the server side, the API routes handle these requests firstly by verifying that an authenticated user is logged in, and then performing the desired database operation using helper functions such as `updateOne` and `deleteOne` that are imported from the 'lib/dbHelper' file.

```
async function deletePool() {
  try {
      const response = await fetch('/api/deletePool', {
          method: 'POST',
          headers: {
              'Content-Type': 'application/json',
          },
      });
      const data = await response.json();

      if (data.success) {
          setPoolId(null);
          setPool([]);
          setPoolCreated(false);
          await deleteEthplorerPool(poolId, ETHPLORER_API_KEY);
      } else {
          throw new Error(data.error || "Failed to delete pool");
      }
  } catch (error) {
      console.error("Error deleting pool:", error.message);

  }
}
```

Figure 3.3: 'wallet-tracker.tsx' function.

```
import { getServerSession } from "next-auth"
import { authOptions } from "./auth/[...nextauth]"
import type { NextApiRequest, NextApiResponse } from "next"
import { deleteOne, updateOne } from "../../lib/dbHelper";

export default async function handler(
  req: NextApiRequest,
  res: NextApiResponse
) {
  const session = await getServerSession(req, res, authOptions);
  if (!session || !session.user || !session.user.email) {
    return res.status(401).json({ error: "Not authenticated" });
  }
  const sessionEmail = session.user.email;
  try {
    // Delete the pool document
    await deleteOne("pool", { userEmail: sessionEmail });
    // Update the user's document to reflect the pool deletion
    await updateOne("users", { email: sessionEmail },
    { $set: { poolID: "", poolCreated: false } });
    res.status(200).json({ success: true });
  } catch (error) {
    res.status(500).json({ error: "Database error" });
  }
}
```

Figure 3.4: 'api/deletePool.ts' file.

The example shown in Figures 3.3 and 3.4 demonstrates the synergy between these intricately linked operations. The `deletePool` function is triggered when the user clicks the 'Delete Pool' button on the web page. The function first sends a POST request to the corresponding API route to execute the deletion of the pool collection in the database. The API route achieves this by finding the pool document whose `userEmail` attribute matches the email from the session, deleting it, and updating the `poolID` and `poolCreated` attributes for the user document. If the deletion is successful, the local state variables are updated to reflect the changes on the front-end and the Ethplorer pool is deleted by calling the `deleteEthplorer` pool function. Throughout the entirety of the application,

data consistency is enforced between the Ethplorer pools, database, and front-end state variables. This provides a personalised and persistent experience for multiple users across multiple sessions.

3.6.3 Wallet Tracker Logic

Eyethereum's wallet tracker functionality is underpinned by robust logic that enables the real-time tracking of Ethereum wallet transactions and formatting of the raw data into readable, user-friendly outputs. The function responsible for this is `startTracking`, which is triggered when the user clicks the 'Enable Tracking' button on the web page and executes every minute on a loop until it is stopped. Firstly, before the loop begins: the local state variable `isTracking` is set to true; the values of the local variables used within the loop such as `previousTransactions` and `previousOperations` are updated with the most up to date information from the database by calling the `fetchUserPoolStatus` function; and the current timestamp is saved in `trackingEnabledTimestamp`, which allows the function to make the distinction between live transactions made in real-time and transactions made since the user last checked.

Inside the loop, `getPoolUpdates` is called to fetch the Ethplorer data from the user's pool of tracked wallets [65]. The total number of transactions and operations from the data are extracted and saved as `currentTransactions` and `currentOperations` before being compared to `previousTransactions` and `previousOperations`. The 'previous' values represent transactions which have already been detected and displayed on the wallet tracker page. The function then checks if the number of current transactions is greater than `previousTransactions` and that the transaction hash has not already been processed. These steps ensure that only new transactions are displayed, and that multiple updates are not displayed for the same transaction, respectively. If both conditions are met, the transaction data is extracted and used to construct a `newLog`, a string representing the notification displayed to the user on the web page. The transactions are then formatted based on whether the ETH transfer is to or from the tracked address and whether it is a live update or occurred since the user last checked. Finally, the `newLog` is added to the user's array of logs, `previousTransactions` is set to `currentTransactions`, and the transaction hash is added to the array of processed hashes. All changes are reflected both on the front end and in the database to ensure consistency.

The logic for detecting and displaying new operations is mostly the same as for transactions, however a distinction must be made between ERC-20 transfers and swaps on Uniswap. The transfer and swap functions for ERC-20 tokens each have a unique identifier known as a 'keccak256 hash', which can be found using Ethers as shown in Figure 3.5 [85]. For each new operation detected, the Ethers `transactionReceipt` is obtained and the 'log' is extracted from the receipt, which contains an array of 'topics'. If the first element of the topics array matches the keccak256 hash for a transfer or a swap, then the operation is formatted accordingly.

```
// Keccak 256 hashes for swaps and transfers
const swapKeccak = ethers.keccak256(ethers.toUtf8Bytes("Swap(address,uint256,uint256,uint256,uint256,address)"));
const transferKeccak = ethers.keccak256(ethers.toUtf8Bytes("Transfer(address,address,uint256)"));
```

Figure 3.5: Keccak256 hashes for transfers and swaps.

The loop interval is stored in the `trackerInterval` variable which aids in the management of the tracking functionality. For example, when the user clicks the 'Stop Tracking' button, the `stopTracking` function is called which sets `trackerInterval` to null and sets `isTracking` to false.

3.7 User Interface

The name Eyethereum encapsulates its mission to offer users valuable insights into the Ethereum blockchain. This branding was further emphasised through a logo designed using Microsoft Designer and Canva [86, 87].

Figure 3.6: Eyethereum logo.

The Eyethereum logo was inspired by the famous Eye of Providence, also referred to as the All-Seeing Eye, present on US dollar bills [88]. Instead of an eye inside a pyramid, it is inside a golden Ethereum emblem, encircled by a gold coin and set against a black backdrop. Subsequently, black and gold was chosen as the colour theme for the user interface (UI). The 'Lovelo' font used in the logo was downloaded and used consistently on the website for a cohesive look [89].

The home page consists of the logo in the centre and a succinct description of the website. For unauthenticated users, a welcome message informs them that they are not logged in, accompanied by a reminder to check their junk and spam inboxes for the sign in link. If logged in, the welcome message includes the user's email to provide a personalised experience. Furthermore, authenticated users gain exclusive access to the links for the token checker and wallet tracker pages. A banner is displayed at the top of every page with a link to sign in for unauthenticated users, and vice versa.

The token checker page consists of a text box that prompts the user to enter an Ethereum token contract address, and a button to fetch the results. While the results are loading, the Eyethereum logo rotates in the centre of the screen and a waiting message is displayed. The results are then displayed in white bold text inside dark gold rectangular components, ensuring clear visibility. The wallet tracker page only displays the create pool button if the user has not created one. If a pool has been created, additional functionalities become accessible along with details about the user's pool. Transactions from tracked wallets are displayed in a similar style to the token checker results, with a pagination feature limiting the display to 10 transactions per page. React's `useState` and `useEffect` hooks enabled the seamless integration between the back-end results and the UI [50, 51]. Screenshots of the final UI are displayed in Appendix D.

Chapter 4 Testing and Evaluation

4.1 Manual Testing

The decision to prioritise manual testing for Eyethereum was made due to the unique challenges and intricacies of working with the Ethereum blockchain and the decentralised finance ecosystem. The Ethereum blockchain is still evolving, with new updates, protocols, and smart contract functionalities emerging regularly. Ensuring that an automated testing strategy remains up to date with these rapid changes may be challenging, and failure to do so may result in missing critical nuances. While automated testing can identify logical or syntactical errors in individual functions, in the context of this project they would not be able to validate whether the results themselves were accurate. Competitor benchmarking allows for a more flexible and direct comparison to established products and the identification of any anomalies or discrepancies in real time. In conclusion, the unique, uncertain, and dynamic nature of blockchain development required a more hands on, flexible, and intuitive approach.

For this project, it was determined that the optimal approach was to compare Eyethereum's results with the leading competitors identified in Chapter 2 – Honeypot.is and Token Sniffer for the token checker and Etherdrops Tracking Bot for the wallet tracker - and record whether the results matched. Acceptance tests devised for this process can be found in Appendix E. Given the large number of tokens that were continuously tested over the course of the project, qualitative statements such as 'in every case' and 'in most cases' were used rather than quantitative values.

4.1.1 Token Checker

For the token checker, the results for the name, symbol, buy tax, sell tax, and honeypot status matched the results for Honeypot.is and Token Sniffer in every test case. The results for liquidity locked or burnt also matched Token Sniffer's results in every case. However, discrepancies were present for the ownership renounced and contract verified results. Token Sniffer would often declare that 'ownership renounced or source does not contain an owner contract' was true and Eyethereum would not. It is suspected that Eyethereum's results were inaccurate in these cases as the `isOwnershipRenounced` function does not check if the source code does not contain an owner contract. However, there were also cases where Eyethereum's results appeared to be more accurate than the competitors. For example, for the following token contract address - `0x38029c62dfa30d9fd3cadf4c64e9b2ab21dbda17` – Eyethereum's results matched Honeypot.is, except that Eyethereum determined the contract was verified and Honeypot.is did not [90]. Closer investigation on Etherscan confirmed that the token contract was verified [91]. Furthermore, Token Sniffer returned an empty page with the message, 'Token is pending review', for the same contract address [92].

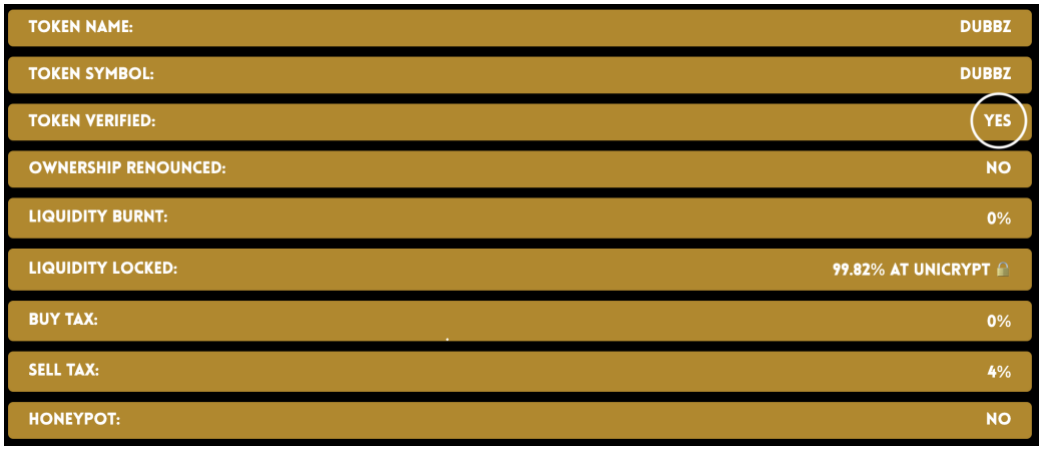

Figure 4.1: Eyethereum determines that the token contract is verified.

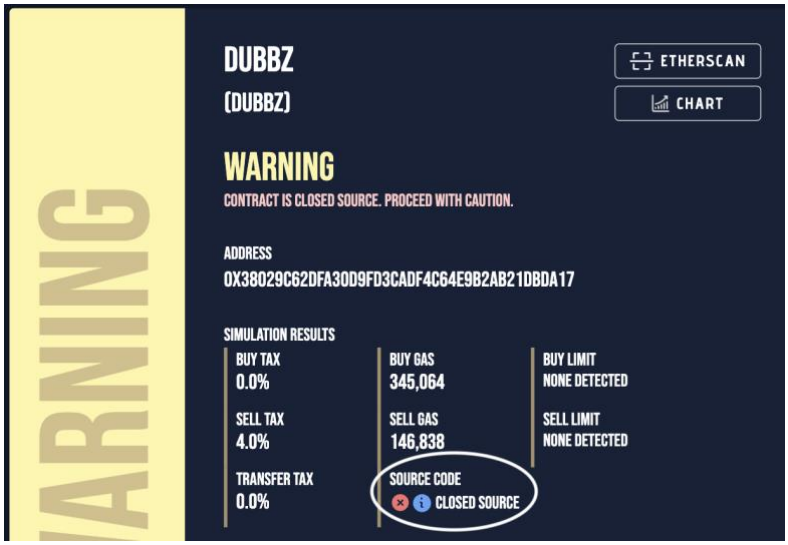

Figure 4.2: Honeypot.is determines that the token contract is not verified [90].

Figure 4.3: Etherscan confirms that the token contract is verified [91].

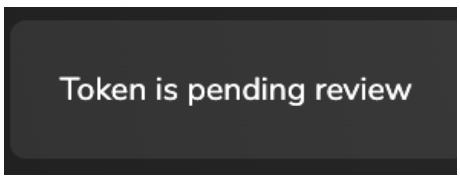

Figure 4.4: Token Sniffer has no results available for the token [92].

Demonstrating any potential edge in any test case over established products is a promising achievement for Eyethereum in its early stages of development.

4.1.2 Wallet Tracker

Manual testing for the wallet tracker compared Eyethereum's results with Etherdrops Tracking Bot to validate their accuracy [43]. The same Ethereum address was tracked on both applications, and the results for ETH transfers, ERC-20 transfers, and swaps on Uniswap V2 were compared. As shown below, the wallet tracker demonstrated strong proof of concept, with the results for live transactions matching in most cases during manual testing.

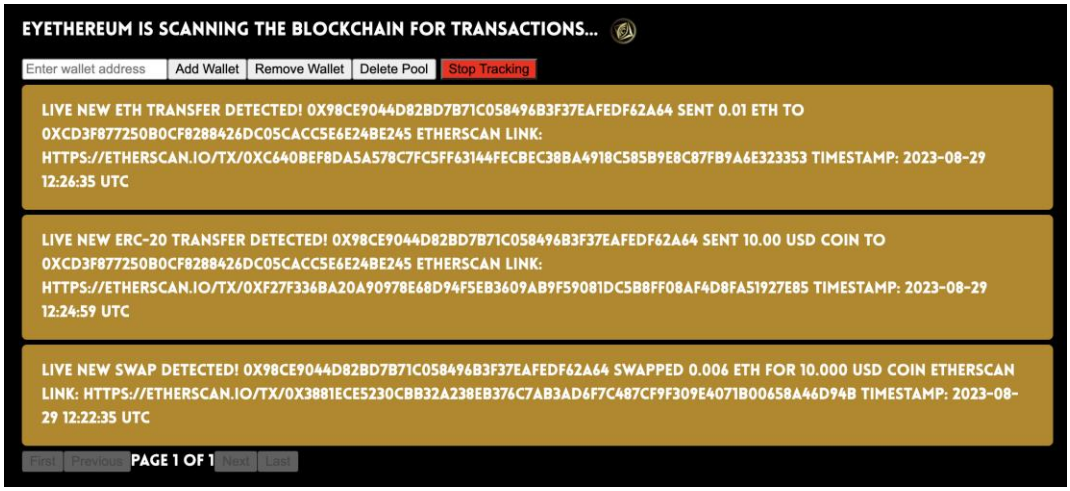

Figure 4.5: Eyethereum's wallet tracker notifications for test transactions.

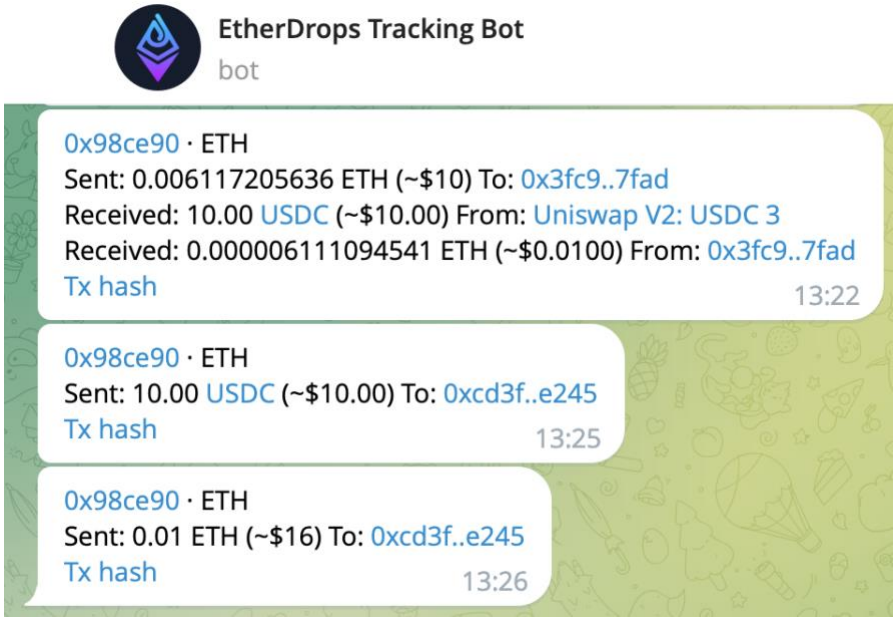

Figure 4.6: Etherdrops' Tracking Bot notifications for the same transactions.

The only test cases for live transactions that did not match Etherdrops occurred when the Ethplorer pool had expired. Deleting the pool and creating a new one in these cases would solve the issue. Furthermore, transactions made since the user last checked would only display on Eyethereum if they had occurred within the last 24 hours due to the limits of the Ethplorer API [65]. In contrast, Etherdrops would notify the user of all transactions as they happened, whether the user was online or not. Results for manual acceptance tests can be found in Appendix E.

4.2 Environment Testing

Eyethereum was run on both MacOS and Windows computers and the manual acceptance tests were carried out on Safari, Firefox, Google Chrome, and Microsoft Edge. The application performed as intended in every case. However, Eyethereum was also tested on iPhone devices, and while the web pages would render, the token checker and wallet tracker functionalities did not work.

4.3 User Testing and Evaluation

4.3.1 Deployment

Initially, it was proposed that beta testing participants clone the Git repository, install the required packages and dependencies, and run Eyethereum locally. However, this approach involved significant time cost and a steep learning curve for participants with no computer science background. It was therefore determined that deploying Eyethereum on a publicly available website was a high priority. Subsequently, Eyethereum was deployed on Vercel, a cloud platform specifically designed to host Next.js projects [93, 94].

https://eyethereum.vercel.app/

Figure 4.7: Public internet domain for Eyethereum.

4.3.2 Limitations

When crafting the beta testing survey, it was crucial to consider the limitations of the product and the participants level of blockchain knowledge when designing the process. For example, the token checker functionality can only calculate the buy and sell taxes and honeypot status of tokens which have liquidity on Uniswap V2. Furthermore, the wallet tracker feature relies on Ethplorer's Bulk Monitor API free plan which has various usage limits. For example, only 5 addresses in total can be tracked and there is a limit of 1000 'ticks' per month. Each transaction is worth 1 tick and actions such as adding a smart contract address to a pool costs 100 ticks [65]. To avoid exceeding these limits, a small number (n=3) of trusted participants were selected for beta testing and given a set of specific instructions to follow.

4.3.3 Beta Testing Survey

Beta testing is a process whereby real users can use a product in its pre-release 'production' environment to test and evaluate its overall usability and effectiveness, uncover any potential bugs, and provide feedback for improvement before its general release [95]. The primary objectives of the beta testing phase were to find out whether core functionalities worked successfully for other users on their own machines and receive general feedback and suggestions for improvement. Firstly, the users were requested to provide some basic information about their experience with and knowledge of computer science and blockchain technologies. They were then fill in an array of forms associated with a specific set of tasks, this included:

- Authentication: register an account, log in and log out.
- Token Checker: enter the two contract addresses provided and compare Eyethereum's results with Honeypot.is and Token Sniffer.
- Wallet Tracker: perform basic functions and indicate whether the application responds as it should.

Finally, the participants were asked for any suggestions or general feedback for how Eyethereum could be improved.

4.3.4 Summary and Evaluation

A total of 3 participants completed the beta testing form. One user had no experience with computer science or the Ethereum blockchain; one user had computer science experience but no experience with Ethereum; and one user had experience with both computer science and Ethereum. This range of experience levels provided the opportunity to receive feedback from diverse perspectives. All the authentication and wallet tracker tasks were successful for each participant. The token checker page returned results that matched Honeypot.is and Token Sniffer in every case except for the 'ownership renounced' check for token 2 – Token Sniffer said it was renounced whereas Eyethereum did not. This was the same discrepancy that was observed in manual testing. A link to the survey can be found in Appendix E and full results can be found in the Supplementary Material submitted in conjunction with this dissertation. With respect to feedback and suggestions, the participants raised a variety of issues, such as:

- User interface was very basic and not visually appealing.
- The token checker took a long time to load the results.
- The token checker and wallet tracker pages were still visible after logging out.
- The sign in email was sent to one user's junk email inbox.
- The user should have the option to delete their account.

4.3.5 Actions Taken

In response to the beta testing feedback, some suggestions were able to be implemented to improve the user experience. Due to the intensive back-end development required to power Eyethereum's core functionalities, front-end development was given lesser priority and the user interface was still at its most basic during beta testing. As discussed in Chapter 3, UI screenshots for the final product are shown in Appendix D. Another common theme was that the token checker results took a long time to load, 10 - 20 seconds was the most reported response by participants, compared to 1 – 10 seconds for Token Sniffer and less than 1 second for Honeypot.is.

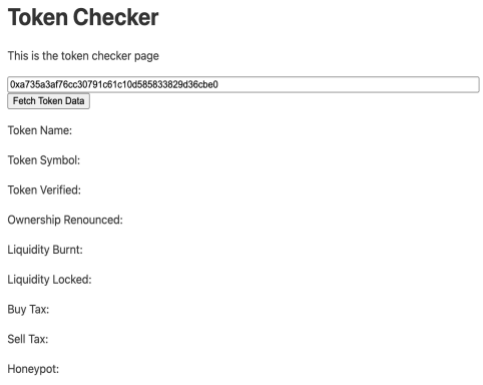

Figure 4.8: Token Checker page for results before user feedback.

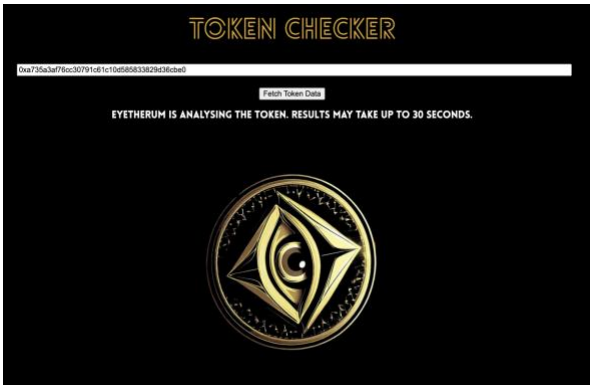

Figure 4.9: Token Checker page waiting for results after user feedback.

To address this issue, a loading graphic of the logo rotating was added to the page while the results were pending, accompanied by a message stating that the results could take up to 30 seconds to display.

Highlighting the benefits of beta testing in spotting flaws and edge cases, one user noted that the token checker and wallet tracker pages were still accessible even after signing out, with the token checker remaining fully functional. To solve this, the 'Access Denied' component that was included in the next-auth template was implemented on both pages if there was no active session, as demonstrated in Figure 4.10.

Figure 4.10: Token checker page while signed out, after feedback.

However, not all suggestions were able to be implemented, such as the option for a user to delete their account. Furthermore, the sign in email was sent to the junk inbox of one participant. An attempt was made to rectify this by modifying the message displayed to the user after the sign in email is sent to contain a warning and reminder to check their junk or spam inboxes. The relevant file was found and edited, and the solution was successful while testing on the local development server.

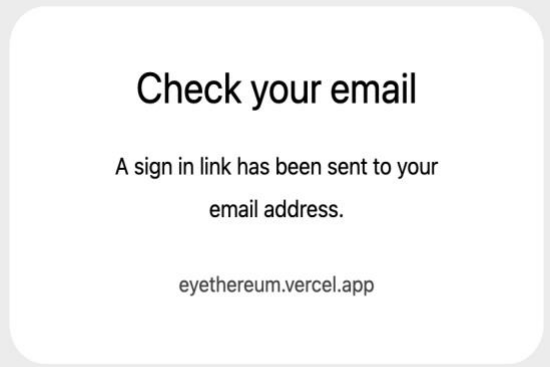

Figure 4.11: Message displayed after sign in email, before feedback.

Figure 4.12: Unsuccessful attempt to solve the issue after feedback.

However, technical challenges were encountered when attempting to push the changes to Git so they could be reflected on the publicly deployed website. The file that had to be modified was in the 'node_modules' directory which is included in the '.gitignore' file. It is generally considered bad practice to push the node modules directory to Git due to the large size and potentially serious ramifications if any mistakes are made. Due to time constraints and the unforeseen technical complexity involved in this task, an alternative solution of adding a warning message to the home screen for unauthenticated users was implemented. This challenge serves as an example of a potential disadvantage of using a template such as next-auth and represents just one of many improvements that could be made to Eyethereum in future work.

Chapter 5 Conclusion

5.1.1 Project Achievements

The aim of this project was to develop a web application to help users gain valuable insights into the Ethereum blockchain with two core features: a token checker, and a wallet tracker. To determine the requirements for a minimum viable product, extensive research into similar existing products was carried out and a requirements gathering survey was sent to target users with existing knowledge of the Ethereum blockchain. This research informed the MoSCoW statements and user stories that were produced to designate a priority level for each feature, as shown in Appendices A and B.

MoSCoW Priority	Features Implemented
Must have	15/17
Should have	0/4
Could have	0/4

Table 5.1: Number of features implemented per MoSCoW priority.

Nearly all the "Must have" features were successfully implemented, however none of the "Should have" or "Could have" features were, providing plenty of opportunities for future work. Overall, the Eyethereum token checker represents a viable product that could be ready for a limited public release to gather more feedback and suggestions for improvement. The wallet tracker functionality demonstrated strong proof of concept, however the reliance on the Ethplorer API limited its scalability and it would not yet be able to handle widespread adoption.

5.1.2 Future Work

At this stage, the token checker's `simulateSwap` function can only return the buy and sell taxes for Ethereum tokens with a liquidity pool on Uniswap V2. Further development could provide support for Ethereum tokens on Uniswap V3, as well as other popular Ethereum-based decentralised exchanges. In addition to Ethereum tokens, support could be extended further to multiple blockchains. Furthermore, a reoccurring piece of feedback from beta testing was that the results were not displayed as quickly as similar existing products. An update was made to add a graphic and message to inform the user that the results are actively loading. However, with more time it would be possible to investigate the issue more deeply and try to quicken the response time, potentially solving the issue rather than just mitigating for it.

For the wallet tracker, there are a wide range of improvements that should be made going forward. Firstly, having the tracker run continuously without the user being on the website would allow for 24/7, real-time notifications via text, Telegram, email, or any other preferred medium of communication that a user could opt in for. Allowing users to add customised names for each tracked wallet would also be a desirable feature that would provide a more personalised experience and help users tracking multiple wallets to curate their pool in a coherent, memorable way. Just as with the token checker, extending support to multiple DEXs and different blockchains would be an exciting prospect.

As discussed previously, there are limits to Ethplorer's Bulk Monitor API free plan that prevented Eyethereum from scaling the product. A potential short-term solution could be to consider one of the paid plans, or search for competitors. However, with more time the optimal solution would be to remove the reliance on the Ethplorer API. Originally, the approach was to use an Ethers event listener that listened to every transaction made in every block, identify transactions that matched a desired set of conditions, and then format the results accordingly.

```
// Instantiate a new provider with Infura
const infuraProvider = new Ethers.JsonRPCProvider(api_key);

// Listen to every transaction made in every block
infuraProvider.on('block', async (blockNumber)=> {

// Get the current block
 const block = await infuraProvider.getBlock(blockNumber);
// Extract all the transactions from the current block
 const blockTransactions = block.transactions;
// Initialise array outside the loop
 let transactionReceipts = [];

// Get the transaction receipts from the block's transactions
 for (const transaction of blockTransactions) {
  const receipt = await
infuraProvider.getTransactionReceipt(transaction);
// Add the receipts to the array
  transactionReceipts.push(receipt);
};
// Loop through all the transaction receipts
for(const transaction of transactionReceipts) {
  // Find transactions that match specified conditions and
  // format the output accordingly
```

Figure 5.1: Original wallet tracker approach.

There are approximately one million Ethereum transactions per day [96]. As such, the method shown in Figure 5.1 was highly data intensive and inefficient, and warnings from Infura were promptly received about exceeding the limits of their JSON-RPC API. With more time, a more data efficient solution could be investigated. If developed, this would represent an opportunity for the wallet tracker to mature into a scalable product capable of handling widespread use.

Beyond the two core features, there are many more opportunities to improve Eyethereum. For example, environment testing found that Eyethereum was not fully compatible with iPhone devices. Ensuring this issue was rectified would enhance the usability and expand the potential user base considerably. Going further, with more time a custom-designed mobile application could be developed and released on the App Store to compliment the web application. This could help Eyethereum stand out from competitors and appeal to users that would prefer to use this type of product on their mobile devices. Furthermore, a comprehensive automated testing strategy could be devised to compliment the manual testing and further validate the accuracy of Eyethereum's results. A more robust testing strategy, combined with more user feedback, bug fixes, and an extensive security review would pave the way towards a final product ready for general release.

Chapter 6 References

[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. United States Sentencing Commission. 2008. Available at: https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar/2018/Emerging_Tech_Bitcoin_Crypto.pdf

[2] A. Antonopoulos. *Mastering Bitcoin 2nd Edition: Programming the Open Blockchain*. 2017. Available at: https://github.com/bitcoinbook/bitcoinbook

[3] L. Tombs. *Could blockchain be the future of the property market?* HM Land Registry. 2019. Available at: https://hmlandregistry.blog.gov.uk/2019/05/24/could-blockchain-be-the-future-of-the-property-market/

[4] IBM. *Benefits of blockchain.* No date. Available at: https://www.ibm.com/topics/benefits-of-blockchain

[5] V. Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralised Application Platform*. ethereum.org. 2014. Available at: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf

[6] Ethereum.org. *PROOF-OF-STAKE (POS).* 2023. Available at: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/#:~:text=Ethereum%20switched%20on%20its%20proof,proof%2Dof%2Dwork%20architecture.

[7] Ethereum.org. *ETHEREUM ACCOUNTS.* 2023. Available at: https://ethereum.org/en/developers/docs/accounts/

[8] Metamask. *A crypto wallet & gateway to blockchain apps*. No date. Available at: https://metamask.io/

[9] A. Antonopoulos. *Mastering Ethereum: Building Smart Contracts and Dapps: Transactions.* 2018. Available at: https://github.com/ethereumbook/ethereumbook/blob/develop/06transactions.asciidoc

[10] Etherscan.io. *The Ethereum Blockchain Explorer.* 2023. Available at: https://etherscan.io/

[11] T. Lindholm., et al. *The Java Virtual Machine Specification Java: SE 20 Edition.* Oracle. 2023. Available at: https://docs.oracle.com/javase/specs/jvms/se20/jvms20.pdf

[12] Ethereum.org *ETHEREUM VIRTUAL MACHINE (EVM).* 2023. Available at: https://ethereum.org/en/developers/docs/evm/

[13] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger: Berlin Version*. 2022. Available at: https://ethereum.github.io/yellowpaper/paper.pdf

[14] A. Antonopoulos. *Mastering Ethereum: Building Smart Contracts and Dapps*. 2018. Available at: https://github.com/ethereumbook/ethereumbook/blob/develop/07smart-contracts-solidity.asciidoc

[15] Ethereum.org. *INTRODUCTION TO SMART CONTRACTS*. 2023. Available at: https://ethereum.org/en/developers/docs/smart-contracts/

[16] Github. *The Solidity Contract-Oriented Programming Language*. 2023. Available at: https://github.com/ethereum/solidity#readme

[17] Ethereum.org. *ERC-721 NON-FUNGIBLE TOKEN STANDARD*. 2023. Available at: https://ethereum.org/en/developers/docs/standards/tokens/erc-721/#:~:text=What%20is%20ERC%2D721%3F,something%20else%20like%20its%20visual.

[18] L. Kugler. *Non-fungible tokens and the future of art*. Communications of the ACM. 2021. Volume 6, Issue 9. Pages 19-20. Available at: https://doi.org/10.1145/3474355

[19] J. Haji. *Guide To Using NFTs In Real Estate*. Forbes. 2022. Available at: https://www.forbes.com/sites/forbesbusinesscouncil/2022/08/04/guide-to-using-nfts-in-real-estate/

[20] RNS. *Legal DID*. 2023. Available at: https://rns.id/

[21] Ethereum.org. *ERC-20 TOKEN STANDARD*. 2023. Available at: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

[22] PayPal Newsroom. *PayPal Launches US Dollar Stablecoin*. 2023. Available at: https://newsroom.paypal-corp.com/2023-08-07-PayPal-Launches-U-S-Dollar-Stablecoin

[23] V. Buterin. *Laumching the Ether Sale*. Ethereum Foundation Blog. 2014. Available at: https://blog.ethereum.org/2014/07/22/launching-the-ether-sale

[24] S. Malwa. *Ethereum ICO Participant Transfers $116M ETH After 8 Years of Dormancy*. CoinDesk. 2023. Available at: https://www.coindesk.com/tech/2023/07/19/ethereum-ico-participant-transfers-116m-eth-after-8-years-of-dormancy/#:~:text=The%20ether%20was%20purchased%20for%2031%20cents%20a%20token%20during%20the%20ICO.

[25] Coincodex. *Ethereum (ETH) ICO*. 2023. Available at: https://coincodex.com/ico/ethereum/

[26] Coinbase. *Identity Verification FAQ*. No date. Available at: https://help.coinbase.com/en/coinbase/managing-my-account/update-my-account/identity-verification-faq

[27] Binance Academy. *Know Your Customer (KYC).* No date. Available at: https://academy.binance.com/en/glossary/know-your-customer

[28] A. Aspiris., et al. *Decentralised exchanges: The "wild west" of cryptocurrency trading.* International Review of Financial Analysis. 2021. Volume 77. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1057521921001782

[29] H. Adams., et al. *Uniswap v3 Core.* 2021. Available at: https://rholang.io/community-site/whitepaper-v3.pdf

[30] Uniswap. *Uniswap Overview*. No date. Available at: https://docs.uniswap.org/concepts/overview

[31] Uniswap Labs Blog. *Uniswap's Year in Review: 2020.* 2020. Available at: https://blog.uniswap.org/year-in-review

[32] L. Ledesma. *Decentralised Exchange Uniswap Trading Volume Outpaces Coinbase for 4th Consecutive Month.* CoinDesk. 2023. Available at: https://www.coindesk.com/markets/2023/05/11/decentralized-exchange-uniswap-trading-volume-outpaces-coinbase-for-4th-consecutive-month/

[33] Uniswap Help Center. *Networks on Uniswap*. 2023. Available at: https://support.uniswap.org/hc/en-us/articles/14569415293325-Networks-on-Uniswap

[34] Dune. *Uniswap V2/V3 Comparison*. 2023. Available at: https://dune.com/mtitus6/Uniswap-V2V3-Comparison

[35] Commodity Futures Trading Commission. *CTFC Charges Sam Bankman-Fried, FTX Trading and Alameda with Fraud and Material Misrepresentations.* 2022. Available at: https://www.cftc.gov/PressRoom/PressReleases/8638-22

[36] Chainalysis. *The 2023 Crypto Crime Report*. 2023. Available at: https://go.chainalysis.com/rs/503-FAP-074/images/Crypto_Crime_Report_2023.pdf

[37] B. Mazorra., et al. *Do Not Rug On Me: Zero-Dimensional Scam Detection*. 2022. Available at: https://arxiv.org/pdf/2201.07220.pdf

[38] C. Torres., et al. *The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts.* USENIX. 2019. Available at: https://www.usenix.org/conference/usenixsecurity19/presentation/ferreira

[39] P. Xia., et al. *Trade or Trick? Detecting and Characterising Scam Tokens on Uniswap Decentralised Exchange.* Proceedings of the ACM on Measurement and Analysis of Computing Systems. Volume 5, Issue 3. Pages 1-26. 2021. Available at: https://dl.acm.org/doi/10.1145/3491051

[40] Honeypot.is. *Ethereum Honeypot Detector*. 2023. Available at: https://honeypot.is/ethereum

[41] Token Sniffer. 2023. Available at: https://tokensniffer.com/

[42] A. Goldstein. *Stablecoins: How Do They Work, How Are They Used, and What Are Their Risks?* Committee on Banking, Housing, and Urban Affairs, United States Senate. 2021. Available at: https://www.banking.senate.gov/imo/media/doc/Goldstein%20Testimony%2012-14-21.pdf

[43] DropsTab. *EtherDrops Tracking Bot*. No date. Available at: https://dropstab.com/products/etherdrops-bot

[44] Alchemy University. 2023. Available at: https://university.alchemy.com/

[45] LearnWeb3. 2023. Available at: https://learnweb3.io/

[46] TypeScript. *TypeScript is JavaScript with syntax for types.* 2023. Available at: https://www.typescriptlang.org/

[47] NodeJS. *About Node.js*. No date. Available at: https://nodejs.org/en/about

[48] NPM. *About npm*. No date. Available at: https://www.npmjs.com/about

[49] React. *React: The library for web and native user interfaces.* 2023. Available at: https://react.dev/

[50] React. *useEffect*. 2023. Available at: https://react.dev/reference/react/useEffect

[51] React. *useState.* 2023. Available at: https://react.dev/reference/react/useState

[52] NextJS. *The React Framework for the Web*. 2023. Available at: https://nextjs.org/

[53] M. Stonebraker. *SQL databases v. NoSQL databases.* Communications of the ACM. Volume 53, Issue 4. Pages 10-11. 2010. Available at: https://doi.org/10.1145/1721654.1721659

[54] MongoDB. 2023. Available at: https://www.mongodb.com/

[55] MongoDB. *MongoDB Atlas. The multi-cloud developer data platform*. 2023. Available at: https://www.mongodb.com/atlas

[56] S. Bradshaw., et al. *MongoDB: The Definitive Guide.* O'Reilly Media, Inc. 2019. Available at: https://books.google.co.uk/books?hl=en&lr=&id=pIrCDwAAQBAJ&oi=fnd&pg=PR2&dq=MongoDB&ots=bmVxiHwmFm&sig=o7JReYw5aHtelPHKcTYWyde5hDQ&redir_esc=y#v=onepage&q=javascript&f=false

[57] NPM. *MongoDB Node.js Driver*. 2023. Available at: https://www.npmjs.com/package/mongodb

[58] Ethers. *Documentation*. 2023. Available at: https://docs.ethers.org/v6/

[59] NPM. *The Ethers Project*. 2023. Available at: https://www.npmjs.com/package/ethers

[60] Infura. *Unleash the Full Potential of Web3*. 2023. Available at: https://www.infura.io/

[61] JSON-RPC.info. *JSON-RPC*. No date. Available at: https://json-rpc.info/json-rpc/

[62] Github. *ganache*. 2023. Available at: https://github.com/trufflesuite/ganache

[63] Etherscan. *Introduction: Welcome to the Etherscan APIs documentation*. 2022. Available at: https://docs.etherscan.io/

[64] Github. *Ethplorer API*. 2023. Available at: https://github.com/EverexIO/Ethplorer/wiki/Ethplorer-API

[65] Ethplorer. *Bulk API Monitor Guide (0.5.0)*. 2023. Available at: https://docs.ethplorer.io/monitor?from=apiDocs

[66] NPM. *Axios*: *Promise based HTTP client for the browser and node.js*. 2023. Available at: https://www.npmjs.com/package/axios

[67] NPM. *qs: A querystring parsing and stringifying library with some added security*. 2023. Available at: https://www.npmjs.com/package/qs

[68] NextAuth.js. *Authentication for Next.js*. 2023. Available at: https://next-auth.js.org/

[69] Github. *next-auth-example*. 2023. Available at: https://github.com/nextauthjs/next-auth-example

[70] Auth.js. *Official MongoDB adapter for Auth.js / NextAuth.js*. 2023. Available at: https://authjs.dev/reference/adapter/mongodb

[71] Github. *Example app using MongoDB*. 2023 Available at: https://github.com/vercel/next.js/tree/canary/examples/with-mongodb

[72] Uniswap. *Smart Contracts*. No date. Available at: https://docs.uniswap.org/contracts/v2/concepts/protocol-overview/smart-contracts

[73] Uniswap. *Factory*. No date. Available at: https://docs.uniswap.org/contracts/v2/reference/smart-contracts/factory

[74] Uniswap. *Router02*. No date. Available at: https://docs.uniswap.org/contracts/v2/reference/smart-contracts/router-02

[75] F. Vogelsteller., V. Buterin. *ERC-20 Token Standard*. Ethereum Improvement Proposals. 2015. Available at: https://eips.ethereum.org/EIPS/eip-20

[76] Etherscan. *Contracts*. 2022. Available at: https://docs.etherscan.io/api-endpoints/contracts

[77] Ethers. *class: Contract*. 2023. Available at: https://docs.ethers.org/v6/api/contract/#Contract

[78] Etherscan. 2023. Available at: https://etherscan.io/address/0x000000000000000000000000000000000000dead

[79] Etherscan. 2023. Available at: https://etherscan.io/address/0x0000000000000000000000000000000000000000

[80] UNCX Network. *Secure Defi Infrastructure*. 2023. Available at: https://uncx.network

[81] Team Finance. 2023. Available at: https://www.team.finance

[82] Ethers. *interface: Signer*. 2023. Available at: https://docs.ethers.org/v6/api/providers/#Signer

[83] Uniswap. *getAmountsOut*. 2023. Available at: https://docs.uniswap.org/contracts/v2/reference/smart-contracts/library#getamountsout

[84] Ethers. *class: TransactionReceipt*. 2023. Available at: https://docs.ethers.org/v6/api/providers/#TransactionReceipt

[85] Ethers. *Hash Functions: keccak256*. 2023. Available at: https://docs.ethers.org/v6/api/crypto/#keccak256

[86] Microsoft Designer. 2023. Available at: https://designer.microsoft.com/

[87] Canva. 2023. Available at: https://www.canva.com/

[88] J. Koenderink. *The All Seeing Eye?* Perception. Volume 43, Issue 1. Pages 1–6. 2014. Available at: https://doi.org/10.1068/p4301ed

[89] FontFabric. *Lovelo*. 2023. Available at: https://www.fontfabric.com/fonts/lovelo/

[90] Honeypot.is. *Ethereum Honeypot Detector*. 2023. Available at: https://honeypot.is/ethereum?address=0x38029c62dfa30d9fd3cadf4c64e9b2ab21dbda17

[91] Etherscan. 2023. Available at: https://etherscan.io/address/0x38029c62dfa30d9fd3cadf4c64e9b2ab21dbda17#code

[92] Token Sniffer. 2023. Available at: https://tokensniffer.com/token/eth/0ovf4xnohbgokn2z9g0x2h4ze20b92izsu85bksahljfn983i4jn62j2l96q

[93] Auth.js. *Deployment*. 2023. Available at: https://authjs.dev/guides/basics/deployment

[94] Vercel. 2023. Available at: https://vercel.com/

[95] M Fine. *Beta Testing for Better Software*. Robert Ipsen. 2002. Available at: https://books.google.co.uk/books?hl=en&lr=&id=XoKrMjrsU8EC&oi=fnd&pg=PR3&dq=what+is+beta+testing+software&ots=akr4gNKqxs&sig=rDJyRafozB35N_lcPRLL8HmGubg&redir_esc=y#v=onepage&q=what%20is%20beta%20testing%20software&f=false

[96] Etherscan. *Ethereum Daily Transactions Chart*. 2023. Available at: https://etherscan.io/chart/tx

Appendix A MoSCoW Statements

Must Have

- Users must be able to register an account.
- Users must be able to sign in and sign out.
- Users must be able to access the website on a public domain.
- The token checker must check if a contract is verified.
- The token checker must check if ownership of a contract is renounced.
- The token checker must check what percentage of the liquidity is locked.
- The token checker must check what percentage of the liquidity is burnt.
- The token checker must check the buy tax and sell tax.
- The token checker must check the honeypot status.
- The token checker must support Uniswap V2.
- The wallet tracker must be able to add wallets to track.
- The wallet tracker must be able to remove wallets to track.
- The wallet tracker must be able to filter updates by the type of transaction.
- The wallet tracker must be able to track swaps on Uniswap V2.
- The wallet tracker must be able to track swaps on Uniswap V3.
- The wallet tracker must be able to track ETH and ERC-20 transfers.

Should Have

- The token checker should check for similar contracts.
- The token checker should check if any wallets hold a large percentage of the supply.
- The token checker should support Uniswap V3.
- The token checker should support other DEXs.

Could Have

- The wallet tracker could be able to give wallets customised names.
- The wallet tracker could be able to track swaps on other DEXs.
- The wallet tracker could be able to track token approvals.
- The wallet tracker could be able to receive live updates via email, text, telegram or other mediums.

Would Have

These features were not identified originally but have been identified by the end of the project as work that would be undertaken in the future.

- Users would be able to delete their account.
- Users would be able to set a name and profile picture for their account.
- The token checker would match Token Sniffer's ownership renounced result in every case.
- The token checker and wallet tracker would support multiple blockchains.
- The wallet tracker would become scalable by not relying on external APIs.
- Eyethereum would be fully compatible with mobile devices.
- A custom-made mobile application would be developed for Eyethereum.

Appendix B User Stories

User Story	Priority: 1 Highest – 3 Lowest
As a user, I want to register an account.	1
As a user, I want to sign in and sign out.	1
As a user, I want to access the website on a public domain.	1
As a user, I want to check if an Ethereum token contract is verified.	1
As a user, I want to check if the ownership of an Ethereum token contract has been renounced.	1
As a user, I want to check what percentage of an Ethereum token's liquidity has been locked.	1
As a user, I want to check what percentage of an Ethereum token's liquidity has been burnt.	1
As a user, I want to check the buy tax and sell tax of an Ethereum token.	1
As a user, I want to be able to check the honeypot status of an Ethereum token.	1
As a user, I want to check Ethereum tokens on Uniswap V2.	1
As a user, I want to add Ethereum wallets to my tracked pool.	1
As a user, I want to remove Ethereum wallets from my tracked pool.	1
As a user, I want to filter wallet tracker updates by the type of transaction.	1
As a user, I want to track swaps on Uniswap V2 made by tracked wallets.	1
As a user, I want to track swaps on Uniswap V3 made by tracked wallets.	1
As a user, I want to track ETH and ERC-20 transfers made by tracked wallets.	1
As a user, I want to check Ethereum tokens for similar contracts.	2
As a user, I want to check if any wallets hold a large percentage of an Ethereum token's supply.	2
As a user, I want to check Ethereum tokens with liquidity on Uniswap V3.	2
As a user, I want to check Ethereum tokens with liquidity on other DEXs.	2
As a user, I want to give tracked wallets customised names.	3
As a user, I want to track swaps made on other DEXs by tracked wallets.	3
As a user, I want to track token approvals made by tracked wallets.	3
As a user, I want to receive live updates for tracked wallets.	3

Appendix C Entity Attributes

Attribute	Type
_id	ObjectId
provider	String
type	String
providerAccountId	String
access_token	String
token_type	String
scope	String
userId	ObjectId

Table C.1: Accounts

Attribute	Type
_id	ObjectId
poolID	String
walletsTracked	Array
previousTransactions	Int32
previousOperations	Int32
processedTransactionHashes	Array
processedOperationHashes	Array
userEmail	String
logs	Array
lastCheckedTimestamp	Int32

Table C.2: Pool

Attribute	Type
_id	ObjectId
name	String
email	String
image	String
emailVerified	Null
poolCreated	Boolean
poolID	String

Table C.3: User

Attribute	Type
_id	ObjectId
sessionToken	String
userId	ObjectId
expires	Date

Table C.4: Session

Appendix D User Interface Screenshots

Figure D.1: Home page before logging in.

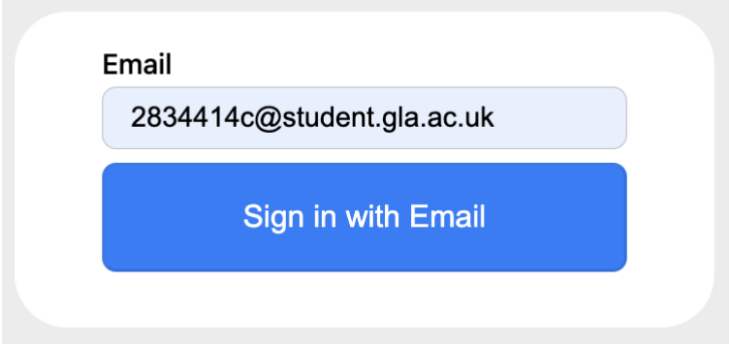

Figure D.2: Sign in page.

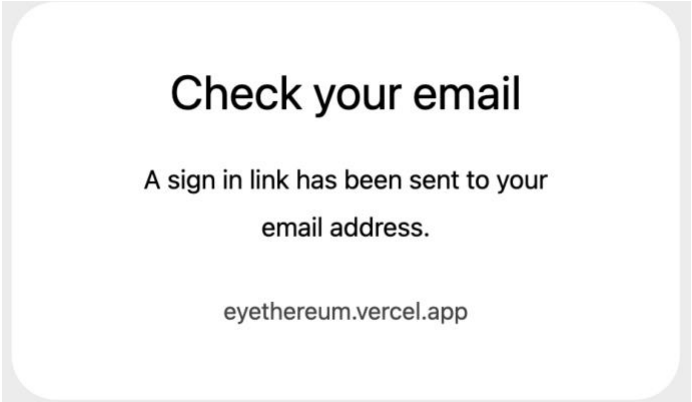

Figure D.3: Sign in page after email has been sent.

Figure D.4: Home page after logging in.

Figure D.5: Token Checker page before results.

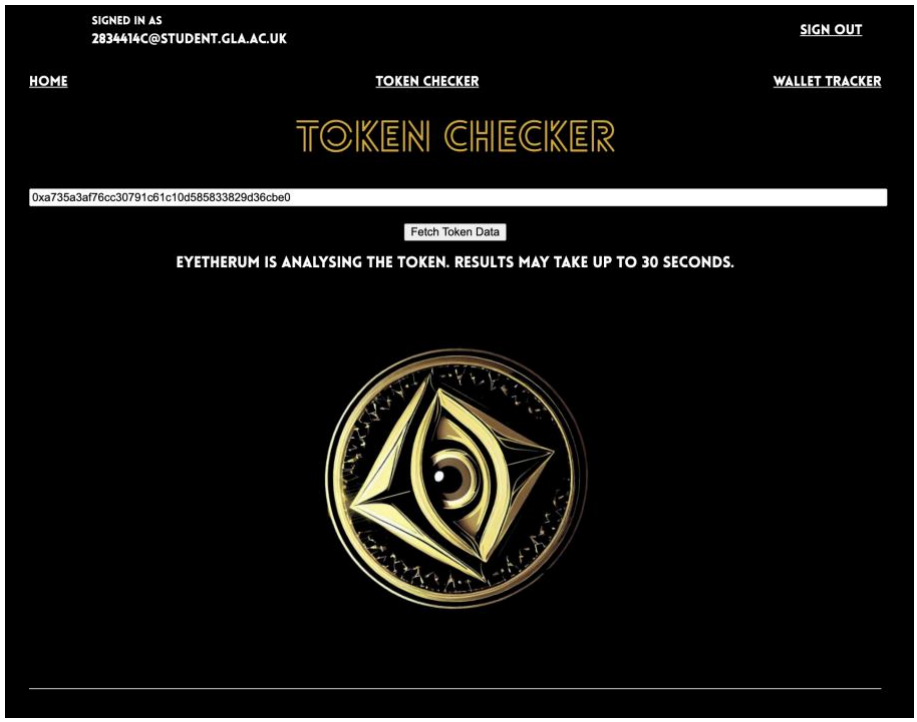

Figure D.6: Token checker page loading results.

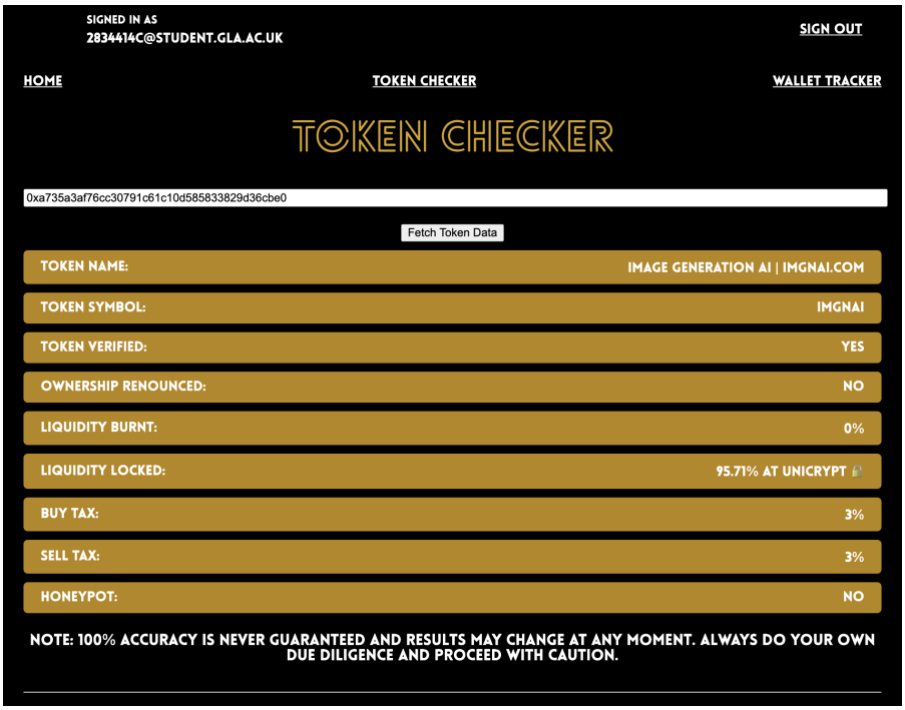

Figure D.7: Token checker page with results.

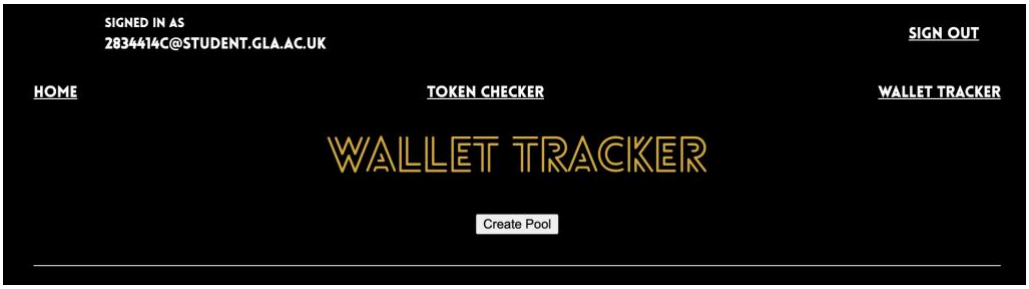

Figure D.8: Wallet tracker page before pool created.

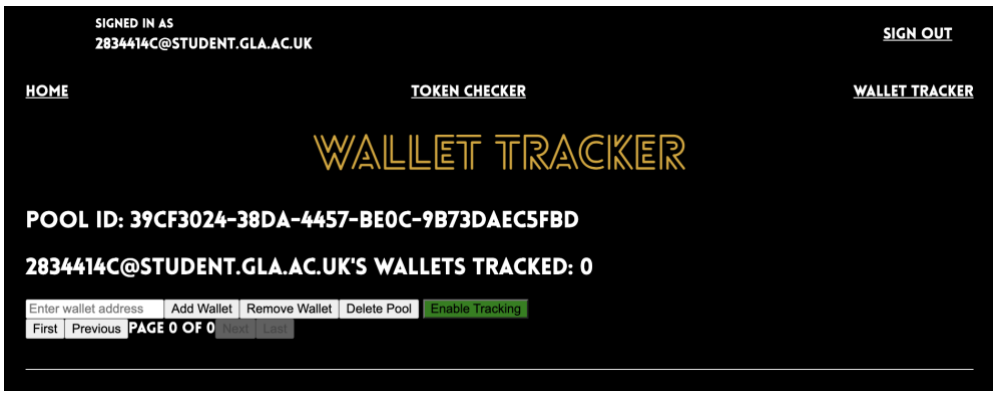

Figure D.9: Wallet tracker page after pool created with no wallets added.

Figure D.10: Wallet tracker page with a wallet added to the pool.

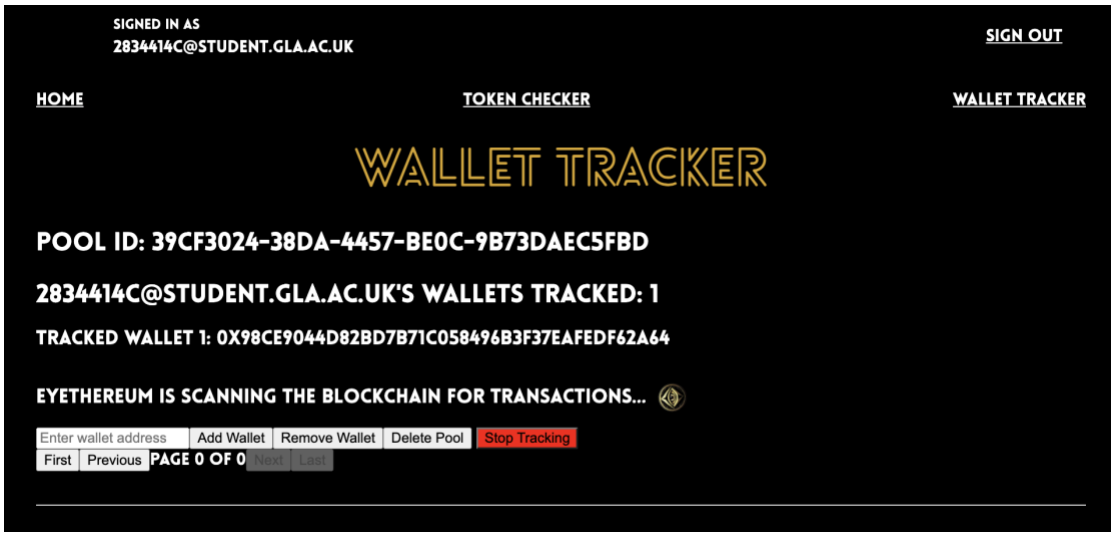

Figure D.11: Wallet tracker page with tracking enabled.

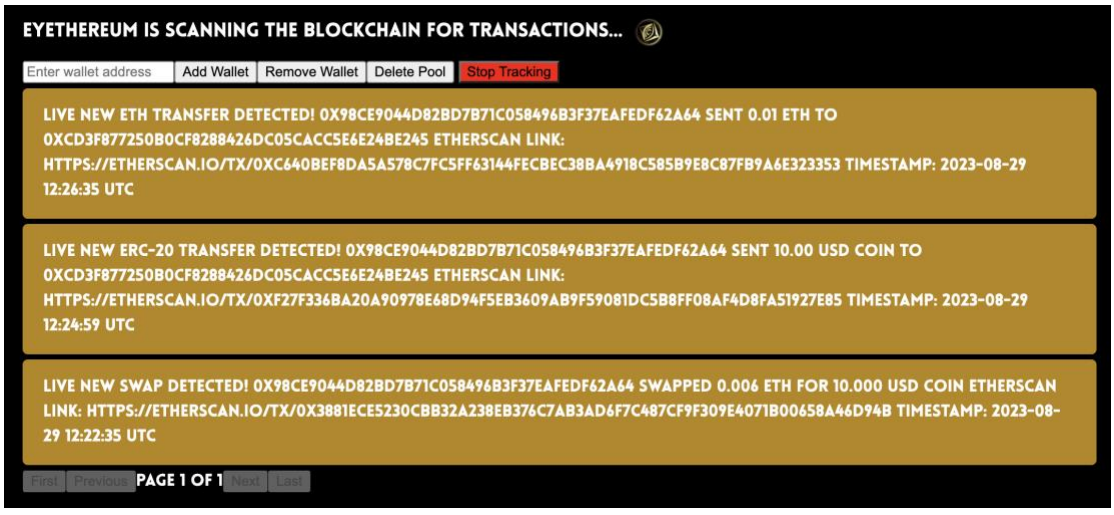

Figure D.12: Wallet tracker page with transactions displayed.

Appendix E Manual Acceptance Tests

Token Checker

Test Case	Passed
The name and symbol of a token should match the results for Honeypot.is and Token Sniffer.	In every case.
The contract verified result should match the results for Honeypot.is and Token Sniffer.	In most cases.
The buy tax, sell tax, and honeypot status should match the results for Honeypot.is and Token Sniffer.	In every case.
The ownership renounced result should match Token Sniffer's results.	In most cases.
The liquidity locked and burnt results should match Token Sniffer's results.	In every case.

Wallet Tracker

Test Case	Passed
Live ETH transfers should match results displayed by Etherdrops.	In most cases.
Live ERC-20 transfers should match results displayed by Etherdrops.	In most cases.
Live swaps should match results displayed by Etherdrops.	In most cases.
ETH transfers made since the user last checked should match results displayed by Etherdrops.	In some cases.
ERC-20 transfers made since the user last checked should match results displayed by Etherdrops.	In some cases.
Swaps made since the user last checked should match results displayed by Etherdrops.	In some cases.

Appendix F Useful Links

Eyethereum Website

https://eyethereum.vercel.app/

Eyethereum Video Demonstration

https://www.youtube.com/watch?v=DCr_dQFMs3Y

Eyethereum Requirements Gathering Survey

https://forms.gle/NgfU2e4QWfGJ2Apt5

Eyethereum User Testing Survey

https://forms.gle/EiKU32kkrrRYtyhQ9