Carbon Zero

A new blockchain platform for carbon emissions trading

UNIVERSITY OF BIRMINGHAM

Student name:

David-Daniel Candreanu

Student ID:

2038036

Degree Course:

BSc Computer Science
with Digital Technology Partnership

Supervisor:

Dr Mohammed Bahja

Word count:

9688

0. Abstract

Global warming is one of the biggest and most concerning problems we are confronted with. It has irreversible consequences and will eventually affect all forms of life if not managed properly. Greenhouse gases are among the biggest polluters in today's world, and governments are looking into solutions to reduce their emissions. Different protocols and mechanisms have been put in place to tackle this issue, the most well known being the Kyoto Protocol.

Currently, there are several platforms put in place to conduct emissions trading, the most popular being traditional web2 systems such as the European Energy Exchange and the Intercontinental Exchange. The current systems, however, lack security and transparency. To address this issue, blockchain solutions have also been proposed. Blockchain offers transparency, security, and immutability.

Even though adopting blockchain technology is a step in the right direction, there is still plenty of work to be done. Solutions introduced in written literature are incomplete and lack implementation, and the blockchain projects that have been published are insufficient and lack critical functionality.

The goal of this project was to create a complete platform that enables governments to assign allowances and oversee trading, and empowers companies to trade and manage their emissions allowance portfolio. A complex system was developed that encompasses trading and token assignment functionality on EVM blockchains, a functional web app and a server that maintains an off-chain order book in a database. The system was developed using SCRUM methodology, and good software engineering practices were respected. Every component was thoroughly tested using manual and automated tests, and user evaluation was conducted.

The resulting platform could have significant impact in today's carbon trading markets. If adopted, it is designed to be a tool that helps governments and companies collaborate better in creating a greener, less polluted world.

0.1 Acknowledgements

I'd like to firstly give thanks to God for always being by my side. I'd also like to thank my family for their unconditional love and support, and for always being there to listen. Special thanks to my supervisor, Dr. Mohammed Bahja for his constant support, valuable advice and for always being available. I'd also like to thank Dr. Shan He for his useful feedback.

Table of contents

0. Abstract **2**

0.1 Acknowledgements 2

1. Introduction **6**

1.1 The Kyoto Mechanisms explained 6

1.1.1 Kyoto protocol mechanisms 7

1.2 What is the Blockchain and how can it help 8

2. Related Work **8**

2.1 Literature review 8

2.2 Solutions review 10

2.3 Review verdict and proposed innovation 11

3. System Architecture **12**

3.1 Target demographic and end users 12

3.2 Rationale for Choosing a 3-Component System 13

4. System Components **14**

4.1 The Smart Contracts 14

4.1.1 Choosing a blockchain and the development tools 14

4.1.2 Smart contract design choices 15

4.1.2.1 Function visibility and access modifiers 15

4.1.2.2 Gas optimisation 15

4.1.2.3 Events 16

4.1.2.4 Contract modularity 17

4.1.3 Security 17

4.1.4 Functional requirements 18

4.1.4.1 Sign Up Contract 18

4.1.4.2 Trading Platform Contract 19

4.1.5 Important functionality 21

4.1.5.1 Minting tokens 21

4.1.5.2 Creating buy or sell orders 22

4.1.5.3 Buying tokens 23

4.1.5.4 Selling tokens 23

4.1.6 Unit testing....24
4.2 Server and Database....25
4.2.1 Prerequisites and development tools....25
4.2.2 Server....26
4.2.2.1 Processing past events....26
4.2.2.2 Processing live events....26
4.2.3 Database....26
4.2.3.1 Collections and documents format....26
4.2.3.2 Security....28
4.2.3.3 Real-time updates....28
4.2.4 Functional requirements....29
4.2.4.1 Server....29
4.2.4.2 Database....29
4.2.5 Testing....30
4.3 Web App....31
4.3.1 Prerequisites and development tools....31
4.3.2 Software design choices....32
4.3.2.1 InterPlanetary File System (IPFS) integration....32
4.3.2.2 Accounts, roles and menus....32
4.3.2.3 Asynchronism and data refreshing....33
4.3.2.4 Modals....33
4.3.2.5 User Experience....33
4.3.3 Functional Requirements....33
4.3.3.1 Web App....33
4.3.3.2 Log In page....34
4.3.3.3 Navigation Bar....34
4.3.3.4 Active Companies page....34
4.3.3.5 Guide page....35
4.3.3.6 Account page....35
4.3.3.7 Trading Platform page....36
4.3.3.8 Admin Dashboard page....36
4.3.4 Functionality and components....37

4.3.4.1 Admin Dashboard page 37
4.3.4.2 Account page 39
4.3.4.3 Guide page 40
4.3.4.4 Active Companies page 40
4.3.4.5 Trading Platform page 42
4.3.5 Security features 43
4.3.6 App-wide manual testing 43
5. Project management **44**
6. Evaluation **46**
7. Future Work **47**
8. Conclusion **48**
9. References **49**
10. Appendix **52**
Appendix A. Smart contracts unit testing results table 52
Appendix B. Smart contracts unit tests 55
Appendix C. Web app manual testing results table 56
Appendix D. Screenshot of Ganache Blockchain 62
Appendix E. User Testing Questionnaire 63
Questions about the user's background 64
Questions about Nielsen's Usability Heuristics 64
Open feedback 66

1. Introduction

Climate change is one of the biggest global issues we are now facing as a society. Global warming is becoming increasingly severe and apparent all around us. Lately, big natural disasters have influenced the world to pay more attention to this issue and to find different solutions to tackle it, by addressing the root issues. The main cause of global warming is greenhouse gases, especially carbon dioxide emissions.

Among the biggest polluters are large companies and countries that emit large quantities of carbon dioxide and other harmful gases in the atmosphere. An estimate made by the International Energy Agency says that the United States of America and China are the number one emitters of greenhouse gases, emitting together around 35% of the total greenhouse gases and 40% of global carbon dioxide gases (Mohajan, 2017). Whenever an entity takes any action that burns fossil fuels, it will emit greenhouse gases.

One of the big impacts of global warming is the increase of the ocean's level, and it is estimated that, as a result, most coastal areas will be underwater by 2050 (Mohajan, 2017) and some insects and animals will go extinct. Whenever harmful gases are released into the atmosphere, they trap heat and harm the planet in the long term. Effects such as hotter temperatures, more severe storms, droughts, and forest fires are more prevalent, causing harm to both humans and animals.

In the search for solutions to limit the emissions of greenhouse gases (GHG), the United Nations Government passed the 'Kyoto Protocol' (United Nations, *What is the Kyoto protocol?*) in Kyoto, Japan, in December of 1997. Currently, there are 192 Parties involved (United Nations, *What is the Kyoto protocol?).* The main idea behind the protocol was to propose a reduction of emissions through the establishment of a flexible market, mechanism which is comprised of three parts: International Emissions Trading, Clean Development Mechanism (CDM) and Joint Implementation (JI). In short, each country that is targeted by the protocol, also known as 'Annex 1 Parties' has a capped allowance of GHG emissions, and they have to stay within limit by either reducing their emissions through different green projects or trading in order to gain more allowance.

The biggest carbon trading system in the world is currently The European Union Emissions Trading System, which was created by the European Union and was their implementation of the commitments made in the Kyoto Protocol (Skjµrseth and Wettestad, 2016). Under the EU ETS, a companies have a limited amount of GHG they can emit in a specific area, and they can also trade those rights. This Trading System covers 45% of the EU's GHG emissions and limits GHG emissions from institutions such as large industrial plants and airlines (EPA, *The EU Emissions trading system)*. The EU ETS scheme is divided into trading periods that have the length of a few years, when parties can trade emission rights. So far, three trading periods, or phases, have ended (Phase 1: 2005-2007, Phase 2: 2008-2012, Phase 3: 2013-2020) and a fourth one is in progress (EC, *Revision for phase 4*) (Phase 4: 2021-2030). Each of these periods has a proposed target for the reduction of greenhouse gases. So far, the EU ETS scheme has found great success and has managed to reduce emissions by about 35% between 2005 and 2019 (EC*, EU Emissions Trading Systems)*.

1.1 The Kyoto Mechanisms explained

This mechanism is also explained by Sadawy and Ndiaye (2022) and the structure of the explanation is inspired by it. The aim of the protocol was to reduce the greenhouse gas emissions of the committed developed countries, known as the 'Annex B Parties' (United Nations, *What is the Kyoto protocol?*), by placing a limit on them and ensuring that each party respected it and stayed within its allowance. The parties in that can implement this system could be countries, companies etc. Each party is assigned an initial cap, which is the number of tones of carbon

dioxide it may emit during a certain period (UN, *Targets for the first commitment period)*, and has the responsibility to ensure that at the end of the period, their emissions will fall within their allowance. In the event that they foresee that emissions will exceed the limit, there are two options: the parties can either decrease their emissions or increase their allowance. For some, it might be cheaper to implement different green solutions and to emit less, while for others it might make more sense to stay at their current rate of emissions and to buy more allowance.

In order to increase their allowance, a trading system is put into place. There, parties can both buy and sell emission rights to fulfil their needs. The initial cap put on each party consists of a number of assigned amount units (AAUs). Each of these units represents the right to emit one metric tone of carbon dioxide in the atmosphere. The trading system works in such a way that parties that have a surplus of emission units (lower CO2 emissions than their allowance) can sell their spares to parties that are over their limits. When a transaction is taking place, the transacted AUUs are subtracted from the seller's emission limit and added to the limit of the buyer.

1.1.1 Kyoto protocol mechanisms

1. Emissions Trading Mechanism (UN, *Emissions trading)*: the trading scheme allows countries that are a part of the Kyoto Protocol to trade parts of their allowance. The units being traded are called Assigned Amount Units (AAUs, sometimes called Kyoto units), and each unit stands for one tonne of CO2. Countries might have excess emissions (permitted but not used) and they are allowed to sell them to countries that need them. Other units traded using this mechanism are removal units (RMU), emission reduction units (ERU), and certified emission reduction units (CER). Each of these additional units also equals to one tonne of CO2 and can be generated by activities such as reforestation or different emission reduction projects.
2. Joint Implementation (UN, *Joint implementation)*: this mechanism empowers parties in the scheme to buy emission reduction units (ERU) from green projects located in a different country that's member of the protocol. Organisations can develop projects that reduce emissions (remove greenhouse gases from the atmosphere). For each tonne of CO2 removed as a result of such a project, an ERU is generated, and parties that still need to reduce their emissions can buy these ERUs from projects in other countries.
3. The Clean Development Mechanism (UN, *The Clean Development Mechanism*): this mechanism empowers countries with Kyoto commitments to implement their own emission reduction projects in developing countries. For every tonne of CO2 that is removed from the atmosphere or reduced with such a project, a certified emission credit (CER) is earned by the investing party.

The Kyoto emission budget is calculated by a regulatory party before the trading (commitment) period commences. The allowance calculation is based on multiple factors, including the **base-year emissions** of a party (how much they would emit under normal circumstances and without trying to reduce, usually equal to 1990 emissions), the **target reduction percentage** (percentage of how much emissions need to be reduced), and the **length of the trading period**.

We get the following formula for the Kyoto allowance:

Kyoto allowance = [base-year emissions] x [target reduction percentage] x [years of commitment period] (EEA, 2010)

For example, if Company A makes a pledge to reduce emissions by 9% for a 5 year trading period, they would have to make the following calculation:

Emissions allowance for Company A = [base-year emissions of Company A] x 91/100 x 5

The result will represent the total number of assigned amount units (also known as Kyoto Units) that the party receives, which is equal to the number of tonnes it is allowed to emit or trade with.

1.2 What is the Blockchain and how can it help

Blockchain is a type of digital ledger technology that enables safe, open, and transparent transaction recording. Instead of a central computer or database, transactions are stored on a network of computers, also referred to as nodes. According to Narayanan and Clark (2017), it is a database that is duplicated over several computers and uses cryptographic protocols to assure transaction security and transparency. Functionality can also be developed on blockchains, in the form of smart contracts. According to Khan *et al.* (2021), they are transparent, tamper-proof executable codes that run on top of the blockchain and facilitate trust-less transactions between parties, without an intermediary. Although the creation of cryptocurrencies such as Bitcoin (Nakamoto, 2008) is its most well-known application, blockchain may also be useful for carbon emission market mechanisms.

Despite all of the success demonstrated by the EU ETS, the Kyoto mechanisms have their drawbacks. The protocol, even though it encourages CO2 reduction and is helpful in reducing greenhouse gases emitted in the atmosphere, has flaws, such as non-transparent trading, difficulty in reporting and monitoring emissions and allowance administration, and carbon credit fraud.

This is where the characteristics of blockchain can show their use. It can empower the building of a distributed ledger of all transactions that happen on the emissions market. All transactions are transparent and can be followed (Zhou and Zhang, 2022), with a clear source of carbon credits, and a public destination. It can help regulators oversee how different parties manage their allowances, and it allows the companies to store and share their emission registry. Thus, regulators can check if the involved parties trade in good faith and that their emissions do not exceed their allowance. In the case that emissions exceed the participant's allowance quota at the end of a trading period, they can be held accountable, with clear proof. Blockchain can also help in building a decentralised trading platform, where no intermediary is needed, and parties can buy and sell emission rights and emission removal units directly, following a public and transparent protocol. Blockchain can also help the trading participants manage their portfolios, with a clear view of their usage, which can be recorded automatically, and of their allowance. This way, corporate entities or countries can plan their trading or emission reduction and removal strategies and make decisions backed by data.

2. Related Work

2.1 Literature review

Blockchain technology has been thoroughly researched, and its features, such as immutability, decentralisation, security and privacy have already been found useful, with applications (Jaoude and Saade, 2019) in fields such as IoT, energy, finance, government, and others.

There have been several attempts to improve the Emission Rights Trading System with the help of blockchain, and several research papers have been published. Diverse solutions have been proposed, each with their own particularities, and it is important to look at the work that's been done so far, in order to draw conclusions and to propose new solutions. Research has been made on several aspects of the Trading System, from the architecture of the system itself, to different trading strategies and governance solutions.

Pan *et al.* (2018) introduced in a research paper a single system that encapsulates three separate components used in today's emissions trading industry: registration, management, and trading. It mentions the advantages of such a blockchain system and why it is better than the traditional approach, but it does not provide any actual implementation, smart contracts or web app. There was more work done by Rawat *et al.* (2022) to build on this 3 piece system, and a more complex solution was devised, that encompasses accounting, trading, and governance. Smart contracts were described with the functionality to buy offset tokens from carbon removal projects, a Decentralised Autonomous Organisation (DAO) was proposed for validating new green projects, instead of a single admin entity; and stakeholders can vote on different issues using tokens. A "buy and burn" mechanism was also introduced, meaning that the credits bought are "burned" to offset the consumed CO2. The solutions devised in this paper also has its limitations: emitters can only buy offset tokens from green projects and do not have the functionality to trade their allowances amongst themselves, no actual smart contracts are presented and no architecture for a decentralised web app is introduced.

Another paper (Al Sadawi and Ndiaye, 2021) proposed a system to "elevate governance" and transparency. It describes the implementation of a system comprised of several smart contracts, that has one administrator and provides the functionality to register buyers and to trade units. It also provides a study on the cost of running the contracts on the Ethereum blockchain. However, it has a few critical drawbacks: it doesn't provide the architecture for a decentralised web app to interact with the smart contracts, and the trading functionality is very limited (even though there can be multiple buyers, there can only be a single seller which sells emission units and a single project which sells offset units per contract).

There have also been attempts to create reputation-based trading systems. Khaqqi *et al.* (2018) proposed a system where buyers had access to better or worse offers, based on their reputation. It achieves this by using Multichain, a blockchain platform that allows users to create their own blockchain and has the functionality to restrict certain users (in this case, based on their reputation) from accessing functionality, unless given permission by the administrator. This system, however, does not provide the source for the user's reputation, nor does it provide a web app's architecture or smart contracts. Another reputation-based trading system was proposed by Muzumdar, Modi and Vyjayanthi (2022). This paper also introduces a formula to calculate a buyer's reputation based on their clean energy usage and efforts to reduce emissions, and buyers with a higher reputation have access to better offers. It uses a permissioned blockchain, but it doesn't provide any smart contract architecture.

Another paper that describes a system similar to the one based on reputation was published by Zhao, Sheng and Yan (2021). It describes how blockchain could help elevate the emissions trading system in China using an "incentive and punish mechanism" that rewards truthful emission reporting and innovation and looks into simplifying the trading procedure by using inter-blockchain technology. However, it does not describe the trading mechanisms, or any smart contract architecture.

Sadawi *et al.* (2021) propose a system comprised of several layers. The bottom layer contains IoT devices that are used to collect CO2 emissions data from companies, and the top handles token allocation and trading. It goes more in depth in how governments could track emissions and allocate tokens, but it does not provide any implementation or smart contract architecture, and the system is complex and difficult to implement in a real-life scenario.

A common point we can see in the literature that has been published so far is the lack of a complete system that encompasses the architecture for a decentralised web app and a comprehensive smart contract specification and architecture. With small exceptions, there aren't any actual implementations of the trading functionality on the blockchain, and where an implementation is introduced, it is rudimentary and limited. In addition, there is little focus on a mechanism for a regulatory entity to assign the initial token balance to parties that are active in a Kyoto Protocol trading period.

2.2 Solutions review

In order to develop an appropriate blockchain solution to the problem of carbon emission trading, we need to look at what software or systems are currently available and in use. A complete system should have functionality for assigning tokens to trading parties following the Kyoto Protocol formula, as well as functionality for trading those tokens. In addition to this, it should leverage the characteristics of the blockchain and offer transparency and immutability in records.

At the moment, the most widely adopted solutions are not blockchain based. Systems such as those offered by EEX (European Energy Exchange), CME Group, and the Intercontinental Exchange are the main tools used in several European and United States trading schemes and provide functionality for trading carbon emission rights.

According to their website, the EEX was appointed as a “common auction platform by the European Commission” (European Energy Exchange, *EU ETS Auction*), and is also involved in emissions trading in China (European Energy Exchange, *Chinese Carbon Market*) and New Zealand (European Energy Exchange, *NZ ETS Auctions)*. It is one of the main platforms used in the European Trading Scheme. Its purpose is to enable companies to trade their allowance after they have been assigned a quota following the Kyoto Protocol, and to acquire additional allowance. The assignment of the initial allowances happens externally, and the EEX is used to buy and trade after that initial assignment has taken place. Companies can modify their allowances using two mechanisms supported by the platform: the EU ETS Auctions (European Energy Exchange, *EU ETS Auctions)* and EU ETS Spot Futures Options (European Energy Exchange, *EU ETS Spot Futures Options)*. By participating in the auctions, allowance can be bought directly from what governments are auctioning off at different points in time, for the current or upcoming trading period. By using the spot futures options platform, allowances can be bought directly from or sold to other companies, with dynamic pricing and offerings. Based on their strategy, companies can choose to acquire emissions allowances using either platform. Although this platform offers a comprehensive trading solution, it does not cover the issue of enabling a regulatory body to allocate the initial allowance, which is done externally. Also, transparency is not a major point of focus. It does not provide a way to see how each company manages their accounts.

Other big platforms, such as the CME Group (CME Group, *Trading Energy Emissions)* and Intercontinental Exchange (InterContinental Exchange, *Carbon Futures)* operate in a similar fashion to the EEX, by allowing companies to trade emission rights in different reduction schemes. Both platforms allow spot and futures options for multiple trading schemes designed for the European Union, the United States, and China, however they lack the functionality to empower regulators to assign the initial quota, and they lack transparency.

Blockchain based solutions have also been devised as an attempt to improve and upgrade the trading mechanism. AirCarbon Exchange (AirCarbon, n.d.) is a marketplace that provides trading functionality for carbon credits in the aviation industry, as well as other carbon offset projects. CarbonEX (CarbonEX, n.d.) and Toucan (Toucan Protocol, n.d.) are also two platforms that allow the trading of offset carbon credits. While Toucan uses an Automated Market Maker mechanism in order to create a pool of tokens with a dynamic price, CarbonEX provides a more general trading platform for ERC20 tokens, with a hybrid trading mechanism that combines an Order Book with an Automated Market Maker. A common point for these blockchain platforms is that they lack the functionality for an authority to assign an initial allowance of carbon credits. In addition, they are mainly focused on trading the offset units that are generated by green projects, instead of the allowance that is initially assigned to companies at the beginning of a trading period.

2.3 Review verdict and proposed innovation

After looking at the literature that has been published on this matter and at the current software solutions, we can see that a complete system that contains the functionality to assign and trade emission rights in an open, transparent, and fraud-free way is needed. The literature proposes incomplete solutions, with a good background but lacking a complete system architecture and implementation. Traditional emissions trading systems are opaque and do not contain functionality to assign allowances in a transparent way, and the currently used blockchain platforms are incomplete, also lacking the allowance assignment functionality.

The rest of this paper will be focused on the architecture and implementation of a new blockchain system that proposes a novel solution to the current faults of the emissions trading mechanisms.

The novelty stands in the ability of a regulatory entity to approve companies to join the platform and to assign them an initial trading allowance, following the Kyoto Protocol formula described earlier. The regulator can also set the parameters of a new trading period and manage it. Companies will be able to trade their emissions directly through peer-to-peer transactions. The system will be secure, with all the trading functionality done on the blockchain, but also fast and inexpensive to operate because of the off-chain order book approach.

Instead of taking the most commonly used approach for a blockchain trading mechanism, which is the automated market maker, this system will solely rely on an order book. This solution was chosen in order to allow direct matching between two trading parties. Therefore, instead of having a pool of tokens to buy and sell, companies will be able to create either custom future orders that will be added to the order book or spot orders that will be filled automatically from what is available in the order book. All transactions will be direct and peer-to-peer, transferring tokens to the buyer and funds to the seller without any intermediaries.

Storing data on the blockchain is expensive, and every operation has a fee attached to it, so reducing operating costs is an important aspect of creating a blockchain application. In order to reduce costs and speed the order matching process, the order book will be held off-chain, and only critical data and the trade execution will be held on-chain. This will spare us from saving large quantities of data on-chain, and the computationally intensive operations of matching orders will have no costs attached to them. Only the data and operations that will assure the integrity of the platform will be held on the blockchain.

A custom token, representing the right to emit one tonne of CO2, will be used on the platform. In the Kyoto Protocol, these are known as the Kyoto units. All of the transactions will also be public and viewable on the platform. Any user will be able to see how a company manages their allowance, and it will also make it simpler for regulators to keep trading parties in check.

3. System Architecture

In order to achieve the goals set for our system, it needs to implement several important features. It needs to offer a way for an administrator to manage a trading period and to assign an allowance to companies, and companies need to be able to trade part of their allowance. To better understand our requirements, we can look at the use case diagram of the entire system in Figure 1. We will have different types of users: unauthorised users, authorised users, and an admin.

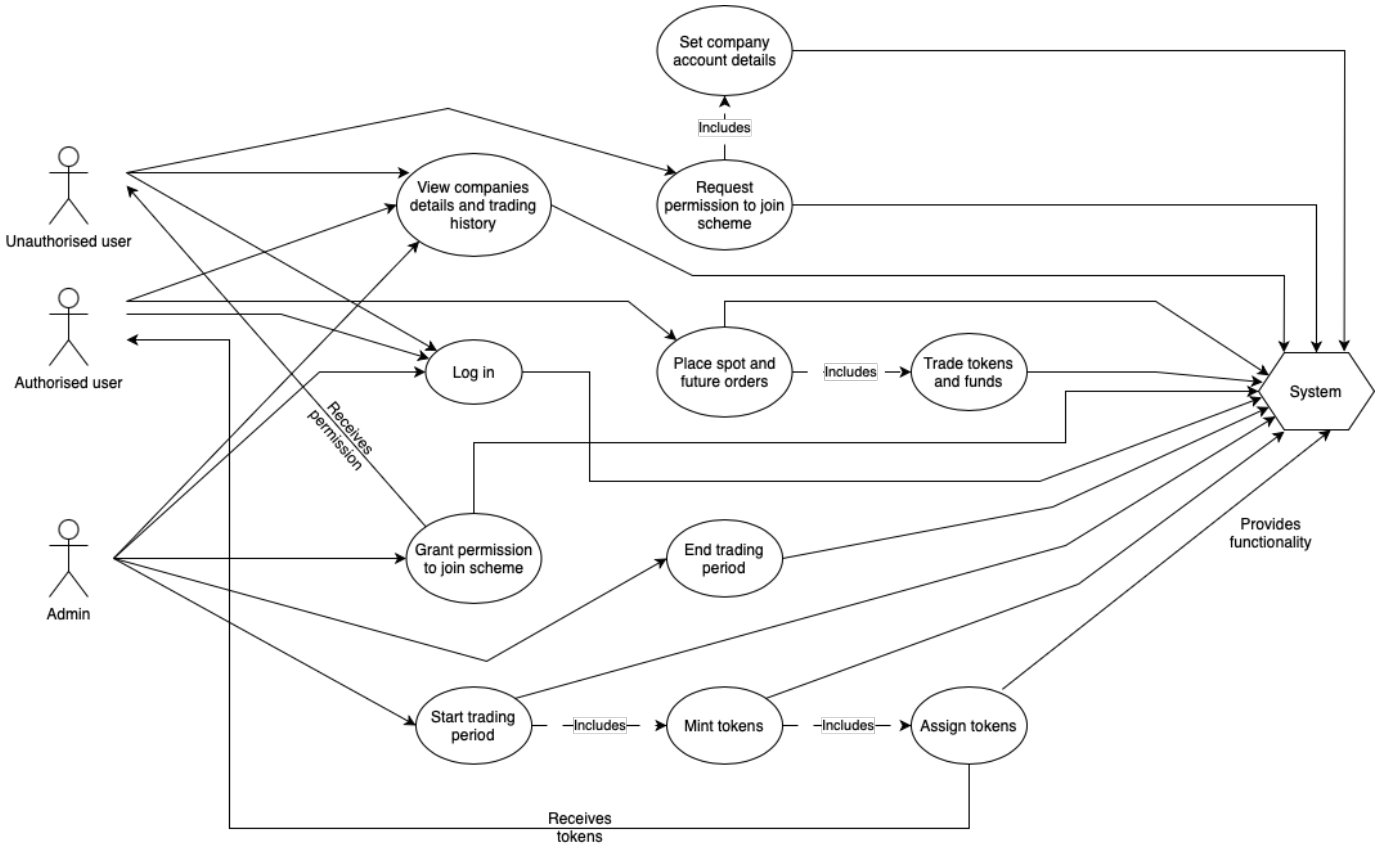


Figure 1. Use case diagram for the entire system

3.1 Target demographic and end users

The platform is aimed at being used by regulatory entities to assign trading allowances to companies and for companies to trade. In addition, it can also be used by other users to get a view of the trading market. By their level of authorisation, three different types of users emerge: unauthorised users, authorised users, and admin users.

Unauthorised users can be normal people who are simply interested in how the emissions market works and how companies manage their portfolio. Unauthorised users can also become authorised users, if they receive permission from the admin. This might be the case for companies that have decided to join the trading scheme and are waiting to gain access to the trading functionality of the platform.

Authorised users have already received permission from the admin to join the trading platform. These are the companies that will be able to place trades when the trading period is open.

The **admin user** represents the regulatory entity. It will approve or decline join requests, oversee the trading period, decide and assign the period parameters, and assign a trading allowance to companies.

The admin will approve a join requests based on interactions they have had with the company outside of the platform. We will make the assumption that the companies have shared their details with the admin before applying to join the platform, and the admin will allow their registration if the details such as their blockchain address match.

3.2 Rationale for Choosing a 3-Component System

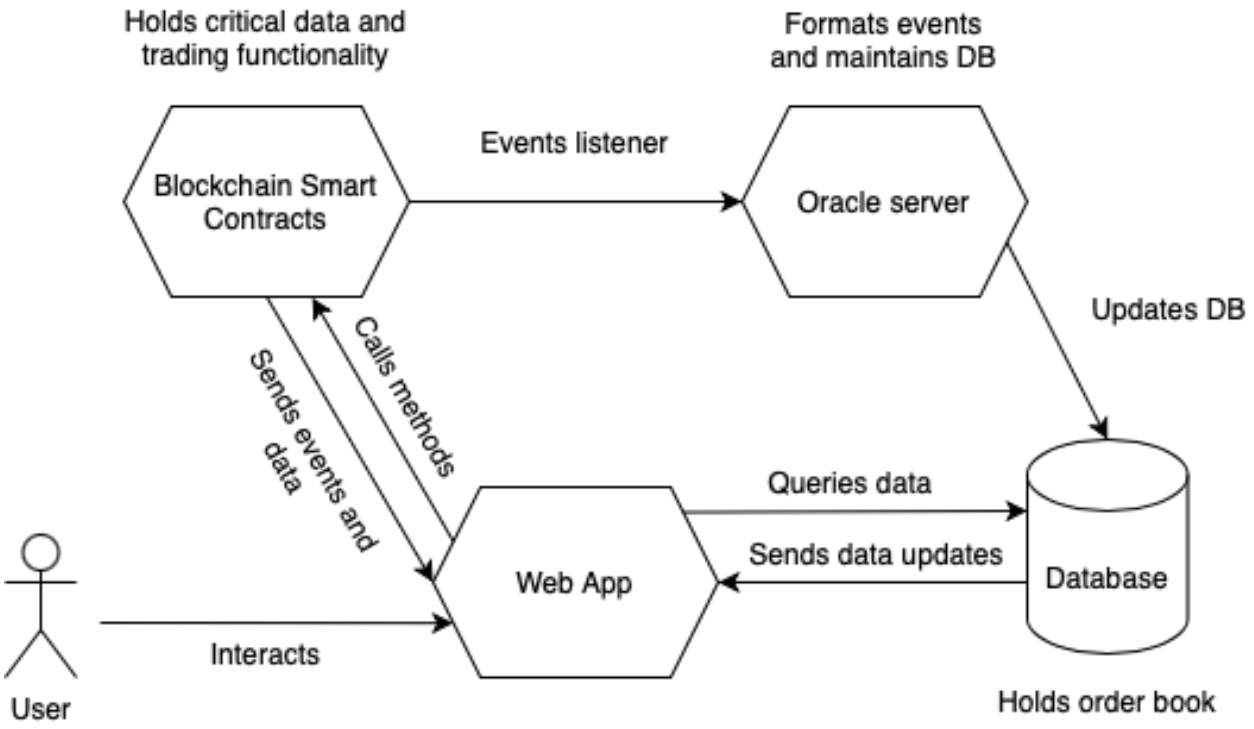


Figure 2. System components and how they interact

In order to enable users to trade allowances on the blockchain, that functionality needs to be created in the form of **smart contracts**. They will be running on a blockchain of choice, and each user will get access to different types of functionality, based on their assigned role.

To interact with the smart contracts in a simple and intuitive way, users need a graphical interface. It is not feasible to have a company connect directly to the blockchain and sign their own cryptographic transactions because of the high level of complexity. Therefore, a **web app** is developed that has access to the system's smart contracts. It will enable users to easily see the data on the blockchain and access its functionality. The app will also receive new data and updates from the smart contracts, in the form of events. We will go more in depth into what events are in the section of the report that covers the design of the smart contracts.

An important point of focus is the optimisation of what is saved and computed on the blockchain. Every action that we make in a smart contract has a fee attached to it, so it is necessary to only keep what is critical on-chain. The approach that we have chosen for matching peer-to-peer

transactions is an order book, and for cost reduction reasons we will keep it off-chain, in an external database.

In order to maintain the database, a server is used to scrape the events emitted by the smart contract and read the data attached to them. The server will format old events and update the database, as well as listen to events emitted while it is running and add them to our storage. We will consider the **server** and the **database** that holds the order book as a single component, because they work closely together and are of no use on their own.

A more detailed explanation of how each component works will follow in the next part of the report.

4. System Components

4.1 The Smart Contracts

4.1.1 Choosing a blockchain and the development tools

Our system is based on smart contracts. They allow us to access the functionality and the qualities of the blockchain.

The smart contracts were designed to be deployed on EVM compatible blockchains for multiple reasons. They are the most popular blockchains that support complex smart contracts, with high availability. Also, depending on the desired characteristics, the contracts can be deployed on several blockchains without any changes to the code being needed. This allows for future multiple blockchain integrations. The most popular EVM blockchain is Ethereum, however, it might not be the first choice to deploy on because of the high operating fees. Other options are available, such as the Binance Smart Chain, Polygon, or Avalanche, and each of them has different features, such as the consensus mechanism.

Another reason for choosing to develop on an EVM blockchain is the high quality of the available documentation. Security needs to be a top priority when developing code on the blockchain, because even the smallest vulnerability can have a devastating effect if it leads to stealing a user's funds or tokens. Smart contracts for the EVM are written in Solidity, and their vulnerabilities have been thoroughly researched and documented.

For the purpose of this project, the Solidity smart contracts will be deployed on Ganache (Appendix D), which is a local testing blockchain that is run on the user's computer. It encompasses all the features of an EVM blockchain, but with no running costs attached to it. Also, the Truffle development framework was used because of its compatibility with Ganache and the wide range of development tools that it offers. It offers the functionality to compile and deploy smart contracts on a blockchain of choice using Javascript code and to also write unit tests for the deployed contracts. When running unit tests, Truffle creates a separate, clean testing chain where all the testing functionality is run in an isolated environment. Truffle was chosen over other frameworks such as Hardhat because of its simplicity and built-in features, such as the ability to manage multiple contracts and deploy to different networks.

4.1.2 Smart contract design choices

4.1.2.1 Function visibility and access modifiers

Depending on the purpose of the smart contract functions, there are restrictions on how they can be called. Functions are restricted in two different ways: by **visibility** and by **user permission**.

For **visibility** there are, for example: 'private' functions that can only be called by other functions inside the smart contract, 'external' functions that can only be called from separate contracts and others.

Different users can access and call different functions on the contracts depending on their **permission** level. The user with the highest permission, the admin, has the most flexibility. When the admin deploys the contract, their address receives owner status and can edit the status of other users addresses. Unauthorised users can request to receive a higher status from the owner and get rights to access more functionality, such as the trading mechanisms. Unauthorised users have an address status equal to **0**, users pending authorisation have a status equal to **1**, authorised users have **2** and the admin has a status equal to **3**.

In order to achieve this restriction functionality, modifiers were used. Modifiers are Solidity functions that can restrict access to different functions. They can be predefined (used for the visibility restrictions) or custom (used for the user restrictions). A custom modifier can be seen in the Figure 3 below, which restricts function access to authorised users only:

```
modifier is_approved() {
    require(address_account_status[msg.sender] == 2);
    _;
}
```

Figure 3. Custom function modifier

4.1.2.2 Gas optimisation

Gas is the currency of operating on the blockchain. In order to maintain the network, a fee is paid from the caller's funds whenever any operation is done in smart contracts. The most expensive operation is storing data on chain. In order to reduce the amount of gas paid, several optimisation techniques were used:

-use of events to save data instead of saving it to the blockchain storage;
-avoiding unnecessary event emissions;
usage of mappings instead of arrays;
-packing smaller variables (such as uint64) together;
-batching function calls of the same type;
-saving only one 32-byte variable per buy or sell order (instead of two separate variables for amount and price).

More complex gas optimisation techniques were also used, as seen in the following section:

Short error codes: in order to save gas, errors with short codes will be emitted in case of failure. This is done to avoid storing long strings on the blockchain. They can be mapped using the following table:

Error code	Error message
E1	Function caller does not have appropriate permissions.
E2	Function caller is already Signed Up.
E3	Mint address list contains addresses without Trading Permission.
E4	The trading period is not open.
E5	Sell order amount is bigger than address allowance.
E6	Address already has an active sell order open.
E7	The amount or price parameters of the buy function are incorrect.
E8	The buy order amount is higher than the amount the seller has on sale.
E9	Not enough funds sent to the buy function to complete transaction.
E10	Failed to send Ether.
E11	Not enough funds were sent to the function to open a buy order for the specified amount and price.
E12	Address already has an active buy order open.
E13	The amount of price parameters of the delete buy order function are incorrect.
E14	The amount or price parameters of the sell function are incorrect.
E15	The sell order amount is higher than the amount the buyer wants to buy.
E16	Your allowance amount is lower than the amount you are trying to sell.
E17	Contract funds can only be withdrawn when the trading period is closed.

Table 1. Smart contract error codes and what they represent

IPFS integration: because storing data on the chain is expensive, long strings, such as company details will be stored on the InterPlanetary File System. Only the generated hash will be stored on-chain, and that can be used to access the desired details on the IPFS. More information about the IPFS will follow in the report section about the web app.

4.1.2.3 Events

Events are a way for the smart contracts to communicate with elements outside of the blockchain by sending notifications when different actions happen. For example, a custom event can be emitted when a function in a smart contract is done running. When events are emitted, they can have different parameters. Once they are emitted, they are permanently stored on the blockchain.

Events can be used as a cheaper form of immutable storage on the chain. Although past events cannot be accessed by functions inside the smart contract, they can be accessed from outside the contracts.

In order to lower operating costs, the proposed system uses different types of events to log data.

For example, when a buy order is created, an event is emitted with the order parameters, as seen in Figure 4. The other system components are listening to this type of events. When emitted, the events are processed and added to the database (in the order book, for example) or used inside the web app. If needed, past events can also be accessed, because they are permanently stored on the chain.

```
emit buy_order_event(msg.sender, _amount, _price);
```

Figure 4. Buy order creation event

When the same buy order is deleted, a different event of the same type is emitted which contains the same blockchain address (msg.sender), but the other parameters are empty, as seen in Figure 5.

```
emit buy_order_event(msg.sender, 0, 0);
```

Figure 5. Buy order deletion event

The smart contracts also emit other types of events for minting, changing a user's status, trading tokens and so on.

4.1.2.4 Contract modularity

The code deployed on the blockchain is immutable. This means that once a contract has been compiled and added to the chain, it can no longer be edited. In order to allow some flexibility in development, the smart contracts were deployed in a modular way: the sign up contract is separate from the trading contract. The trading contract extends functionality from the sign up contract. In case changes need to be made to the trading contract, it can be redeployed and linked to the old sign up contract.

4.1.3 Security

Security is crucial when developing smart contracts. A vulnerable contract could allow an attacker to steal funds or tokens from an address. Most security vulnerabilities are tied to mistakes in the logic of the functions, and in the following part we will look at the most common ones and how the proposed smart contracts are secure from them.

Re-entrancy is a vulnerability often found when users can withdraw funds. Calling the same functions that withdraws funds multiple times before it finishes running might enable an attacker to withdraw the amount multiple times. A place where this attack might occur would be in the 'delete_buy_order' function, because when prospective buyers delete their orders, the amount of 'WEI' they have locked in the buy order (which was sent to the smart contract address) is returned to them. However, since this function does not make any external function calls, there is no

opportunity for an attacker to repeatedly call the same function before previous invocation has completed. Additionally, once the transfer of Ether is done, the buy order is deleted, which further reduces the potential for a reentrancy attacks.

Arithmetic overflows and underflows used to be a big problem in older versions of Solidity. Checks had to be put in place to make sure that arithmetic operations returned the correct results. However, this issue has been fixed in the later versions of the programming language. Since the system's smart contracts are using a pragma newer than Solidity 8.0, the compiler checks for overflows or underflows automatically.

The code is also safe from **replay attacks**. This is a vulnerability where an attacker can replay an identical transaction multiple times.

When creating a buy or sell order, the parameters of the order are hashed using the safe keccak256 function and stored on the blockchain. Later, when calling the buy or sell function, the correct parameters need to be provided for the order that is selected to be fulfilled, and they are hashed to check for equality with the hash that has already been stored on chain. The transaction happens only if the hashes match.

On every successful transaction, the hash changes, because the amount and price parameters are changed. An attacker also does not have access to the hash, because it is private and can only be viewed by the functions inside the contract. Therefore, an attacker can not replay the same transaction multiple times.

4.1.4 Functional requirements

For the purpose of this project, two smart contracts were developed: one for sign up and permission management, and one for trading. Each contract needs to implement complex functionality, and the MoSCoW prioritisation technique (Tudor and Walter, 2006) was used to devise functional requirements. The ranking splits requirements into several priority levels: must-have, should-have, could-have, wouldn't-have. The must-have requirements should be implemented to have a usable first version of the contracts.

4.1.4.1 Sign Up Contract

This smart contract provides the functionality to sign up for the platform as a user, and it enables the admin (the owner of the contract) to approve or decline sign up requests. It also allows the user to edit their details or delete their account.

Number	Functional requirement	MoSCoW ranking
1	The contract must grant owner status to deployer.	M
2	It must set the status of the deployer's address to 3 by default.	M
3	It must ensure that only the owner can set the account status of any address.	M
4	It must set the status of a user's account to 0 by default.	M
5	It must allow a user to only request access for their own account.	M
6	It must allow a user to only be able to delete their own account.	M

Number	Functional requirement	MoSCoW ranking
7	It must set the user's account status to 0 when they delete their account.	M
8	It must set the user's CID and emissions to the values given as arguments when they request access.	M
9	The contract should allow users to get the status of an address.	S
10	It should allow users to get the CID of an address.	S
11	It should allow users to get the emissions of an address.	S
12	It should emit events when the status of an account is modified.	S
13	The functionality to set an account status, delete an account or request access should only be callable from an external source.	S
14	It should be of abstract type.	S
15	The contract could allow users to update the CID or the emissions of their own address.	C
16	The contract won't have any functionality related to transferring funds or tokens.	W

Table 2. Sign Up Contract Functional Requirements

4.1.4.2 Trading Platform Contract

This is the smart contract that enables the admin of the platform (the owner of the contract) to set the trading parameters and mint tokens. It also allows approved users to manage their orders and trade tokens. It extends the Sign Up Contract, in order to access the status of each account and their details.

Number	Functional requirement	MoSCoW ranking
1	The contract must extend SignUpContract.	M
2	It must grant owner status to deployer.	M
3	It must only allow the owner to set the trading open status.	M
4	It must allow anyone to see the trading open status.	M
5	It must only allow the owner to mint tokens.	M
6	It must allow the owner to set the length and target reduction of the trading period during token minting.	M
7	It must assign tokens to addresses following the rules set by the owner.	M
8	It must emit an event whenever a mint happens.	M

Number	Functional requirement	MoSCoW ranking
9	It must only allow users with approved addresses to create a sell or buy order.	M
10	It must only allow users to create a sell or buy order only if trading is open.	M
11	It must only allow users to create a sell order for less tokens than their current allowance.	M
12	It must only allow users to create a buy order if they send enough funds to make the desired transaction.	M
13	It must transfer the funds to buy the desired amount of tokens from the buyer's address to the contract's address when a buy order is created.	M
14	It must emit an event whenever a user creates their sell or buy order.	M
15	It must allow users to delete their open sell or buy orders.	M
16	It must return the funds from the contract's address back to the user's address when they delete their buy order.	M
17	It must emit an event whenever a user deletes their sell or buy order.	M
18	It must allow a user to only create or delete a sell or buy order for their own address.	M
19	It must allow any approved users to buy tokens from any seller with an open sell order.	M
20	It must allow any approved users to sell tokens to any buyer with an open buy order.	M
21	It must emit buy or sell events whenever a buy or sell action takes place.	M
22	It must only allow buy or sell actions when trading is open.	M
23	The contract should raise custom error codes for different types of possible errors.	S
24	The contract should allow the owner to specify a percentage of the minted tokens to be allocated to a spare allowance pool.	S
25	It should emit an event whenever the trading open status is changed.	S
26	It should allow anyone to see the allowance of an address.	S
27	It should open trading whenever a mint happens.	S
28	The functionality to create and delete sell and buy orders, buy or sell should only be accessible from an external source.	S
29	It should allow users to only have one open sell order at a time.	S
30	It should allow users to only have one open buy order at a time.	S
31	It should allow users to delete their buy or sell orders even when trading is not open.	S
32	It should send extra funds sent by buyers to the contract's address.	S

Number	Functional requirement	MoSCoW ranking
33	The contract could save the total sum of assigned tokens during minting.	C
34	It could allow users to see the total amount of funds belonging to the contract's address.	C
35	It could allow the owner to withdraw extra funds sent to the contract's address after trading is closed.	C

Table 3. Trading Platform Contract Functional Requirements

4.1.5 Important functionality

This section will provide an overview of the most important functionality of the contracts: minting tokens, creating orders and trading tokens.

4.1.5.1 Minting tokens

Each custom token represents one tonne of CO2 that a regulator assigns to a company as an emission allowance. Although the initial plan was to implement ERC20 or ERC721 tokens, they were not suitable because of the order book trading mechanism used. The order book should be able to automatically match buyers with sellers, and vice-versa. Once matched, the transactions should automatically compile. In order to achieve this, a custom type of token had to be created, which implements custom buy and sell functions.

The initial allowance of tokens happens when the mint function (Figure 6) is called. It has the purpose to create and assign the tokens to authorised companies.

```
ftrace | funcSig
function mint (address[] memory _pending_addresses⬆,
    uint8 _trading_period_length⬆, uint8 _target_reduction⬆) external only_owner {
    uint64 total = 0;
    uint64 company_allowance = 0;
    for (uint i = 0; i < _pending_addresses⬆.length; i++) {
        require(get_account_status(_pending_addresses⬆[i]) == 2, "E3");
        company_allowance = get_account_base_year_emissions(_pending_addresses⬆[i]);
        company_allowance = company_allowance - (
            (company_allowance * _target_reduction⬆) / 100);
        company_allowance *= _trading_period_length⬆;
        allowance[_pending_addresses⬆[i]] += company_allowance;
        total += company_allowance;
    }
    uint64 spare_allowance = (total * 5) / 100;
    total += spare_allowance;
    allowance[msg.sender] += spare_allowance;
    total_allowance += total;
    set_trading_open(true);
    emit minted(_trading_period_length⬆, _target_reduction⬆, block.timestamp);
}
```

Figure 6. Mint function

The function to mint tokens can only be called by the owner of the contract, which is the admin of the platform. Instead of assigning tokens to each user separately, it uses batching to assign tokens to all the approved users at the same time. It follows the Kyoto Protocol formula to assign an allowance. When called, it also assigns the given parameters to the trading period and starts it.

A small fraction of the total of assigned tokens is also sent to the owner's address. This can be used to assign allowance to companies that join the scheme after the initial mint took place.

4.1.5.2 Creating buy or sell orders

When the trading period is live authorised users can create buy or sell orders, which are added to the order book.

When creating a buy order (Figure 7), the user engages to buy an amount of tokens at a set price. The user needs to own enough funds for the transaction to take place. The chosen amount and price parameters are hashed and added to the chain as a single object. The funds required for the transaction to go through are sent to the smart contract's address. When a separate user wants to fill this buy order they need to provide the correct amount and price parameters, which can be taken from the order book.

```
ftrace | funcSig
function create_buy_order (uint64 _amount⬆, uint256 _price⬆) external payable is_approved {
    require(trading_open, "E4");
    require(msg.value >= (_amount⬆ * _price⬆), "E11");
    require(buy_orders[msg.sender] == 0, "E12");

    buy_orders[msg.sender] = keccak256(abi.encodePacked(_amount⬆, _price⬆));

    emit buy_order_event(msg.sender, _amount⬆, _price⬆);
}
```

Figure 7. Create buy order function

If the order is deleted before being fulfilled, the sent tokens are returned to the creator of the order.

The function to create a sell order (Figure 8) works in a similar way. However, it is not required to send any funds. In order to ensure that a user can not oversell from their allowance, only one sell order can be opened at one time. This is also the case for the buy orders.

```
ftrace | funcSig
function create_sell_order (uint64 _amount⬆, uint256 _price⬆) external is_approved {
    require(trading_open, "E4");
    require(allowance[msg.sender] >= _amount⬆, "E5");
    require(sell_orders[msg.sender] == 0, "E6");
    sell_orders[msg.sender] = keccak256(abi.encodePacked(_amount⬆, _price⬆));
    emit sell_order_event(msg.sender, _amount⬆, _price⬆);
}
```

Figure 8. Delete buy order function

4.1.5.3 Buying tokens

Tokens are bought when a user selects a sell order available in the order book and decides to fulfil it. The correct parameters for the sell order are checked to ensure that enough funds are transferred. The selected sell order is either deleted or updated if it is not fully completed, if, for example, the buyer required less tokens than available in the order.

The buy function can be seen in Figure 9 below.

```
ftrace | funcSig
function buy (address payable _seller⬆, uint64 _amount⬆,
    uint64 _on_sale⬆, uint256 _price⬆) external payable is_approved {
    require(trading_open, "E4");
    require(keccak256(abi.encodePacked(_on_sale⬆, _price⬆)) ==
        sell_orders[_seller⬆], "E7");
    require(_amount⬆ <= _on_sale⬆, "E8");
    require(msg.value >= (_amount⬆ * _price⬆), "E9");

    allowance[_seller⬆] -= _amount⬆;
    allowance[msg.sender] += _amount⬆;
    uint64 amount_left = _on_sale⬆ - _amount⬆;

    if(amount_left == 0) {
        sell_orders[_seller⬆] = 0;
        emit sell_order_event(_seller⬆, amount_left, 0);
    } else {
        sell_orders[_seller⬆] = keccak256(abi.encodePacked(amount_left, _price⬆));
        emit sell_order_event(_seller⬆, amount_left, _price⬆);
    }

    emit buy_event(_seller⬆, msg.sender, _amount⬆, _price⬆);

    (bool sent,) = _seller⬆.call{value: msg.value}("");
    require(sent, "E10");
}
```

Figure 9. Buy function

4.1.5.4 Selling tokens

The sell function (Figure 10) works in a similar way to the buy function. A buy order needs to be selected from the order book, the correct parameters for that order need to be given as arguments to the function, and the order is partially or fully completed. At the end, the selling user receives funds, and the buying user, whose order was selected from the order book, receives the tokens.

```
ftrace | funcSig
function sell (address _buyer⬆, uint64 _amount⬆,
    uint64 _on_request⬆, uint256 _price⬆) external is_approved {
    require(trading_open, "E4");
    require(keccak256(abi.encodePacked(_on_request⬆, _price⬆)) ==buy_orders[_buyer⬆], "E14");
    require(allowance[msg.sender] >= _amount⬆, "E16");
    require(_amount⬆ <= _on_request⬆, "E15");

    allowance[msg.sender] -= _amount⬆;
    allowance[_buyer⬆] += _amount⬆;

    uint64 amount_left = _on_request⬆ - _amount⬆;

    if(amount_left == 0) {
        buy_orders[_buyer⬆] = 0;
        emit buy_order_event(_buyer⬆, amount_left, 0);
    } else {
        buy_orders[_buyer⬆] = keccak256(abi.encodePacked(amount_left, _price⬆));
        emit buy_order_event(_buyer⬆, amount_left, _price⬆);
    }

    emit buy_event(msg.sender, _buyer⬆, _amount⬆, _price⬆);

    address payable to = payable(msg.sender);
    (bool sent,) = to.call{value: (_amount⬆ * _price⬆)}("");
    require(sent, "E10");
}
```

Figure 10. Sell function

4.1.6 Unit testing

Extensive unit testing was performed using the Truffle framework, to ensure that all of the functionality of the smart contracts works as expected. All of the functions were tested and multiple tests were written for the critical functionality.

Test number	Test description	Expected result	Actual result
1	Check if a non-owner address can change the status of an address	The smart contract method should revert	PASS - the smart contract method fails and reverts
2	Check if the method to request access sets the company details correctly	The request_access() method should set the cid, status and description to the input arguments	PASS - the method correctly sets the company details
3	Check if the owner can set the status of an account correctly	The account's status should be changed to the argument value, when set_account_status() is called by the owner.	PASS - the smart contract method successfully sets the new status

Test number	Test description	Expected result	Actual result
4	Check if an account can successfully change its CID	The set_account_cid() method should correctly set the CID of the caller's address to the input argument	PASS - the smart contract method successfully sets the new CID
5	Check if an address can not delete the account of a different address	The delete_account() method should fail and revert when called for an address different than the one of the caller	PASS - the smart contract method reverts when it is called for an address different to the caller's
6	Check if an address can delete their own account	The status of the caller's address should be set to 0	PASS - the status of the caller's address is set to 0
7	Check if the trading period is closed before mint	The trading_open boolean should be set to FALSE by default	PASS - trading_open is set to FALSE by default.
8	Check if calling the mint function sets the correct trading parameters	The target reduction and period length parameters should be correctly set to the mint() method arguments	PASS - the mint() function sets the correct parameters
9	Check if the trading period is open after mint	The trading_open boolean should be set to TRUE after calling the mint() method	PASS - trading_open is set to TRUE by mint()
10	Check if deploying the smart contracts assigns the correct status to the owner's address	The owner's address status should be set to 3 after the smart contract was deployed	PASS - the owner's status equals 3

Table 4. Smart contracts unit testing results

For more Unit Tests, see Appendix A or access the carbon_zero/src/truffle/test folder.

All 80 Unit Tests written for the Smart Contracts are passing, as seen in Appendix B.

4.2 Server and Database

The server has the purpose of listening to smart contract events then formatting them and adding them to the database. The database, maintained by the server, holds data from different types of events and the order book of the platform. It sends real-time updates to the web app whenever data is updated.

4.2.1 Prerequisites and development tools

The web server is written in the Node.js JavaScript runtime environment. The reasons for this choice were its compatibility with Web3.js and its event-driven architecture. The latter is particularly useful for our functionality of listening to smart contract events.

The database of choice is Firebase Firestore because of several reasons: it has the functionality to send real-time, fast and efficient updates to a web app, it has extensive query functionality and customisable security rules.

4.2.2 Server

The server's sole purpose is to maintain the database.

4.2.2.1 Processing past events

The server might not be always available. In order to make sure that the database is up to date, the server needs to account for events that were emitted while it was down.

To do this, the server cleans the database on every new run and re-logs every event that was emitted before the current run to the database.

4.2.2.2 Processing live events

While running, the server is constantly listening to new events that are emitted. Depending on the type of event it has to take different actions. For order events, it has to update the order entry of the emitting blockchain address inside the database. For transaction or mint events, it has to add all new entries to the database.

Whenever an event is caught, it is written to the console and then added to the database (Figure 11).

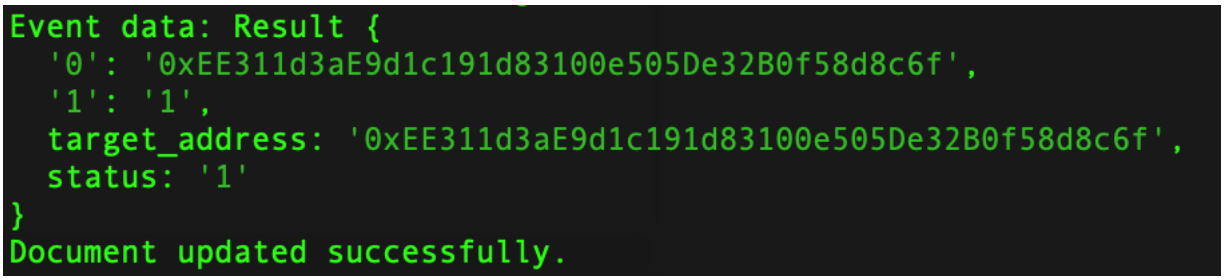


Figure 11. Server console output for smart contract event

4.2.3 Database

4.2.3.1 Collections and documents format

A Firebase Firestore Database is formed of collections, and each collection is formed of documents. Each document can contain multiple data fields of different types.

There are several collections inside the system's database and each of them holds a different type of events. The available collections are as seen in the following figure:

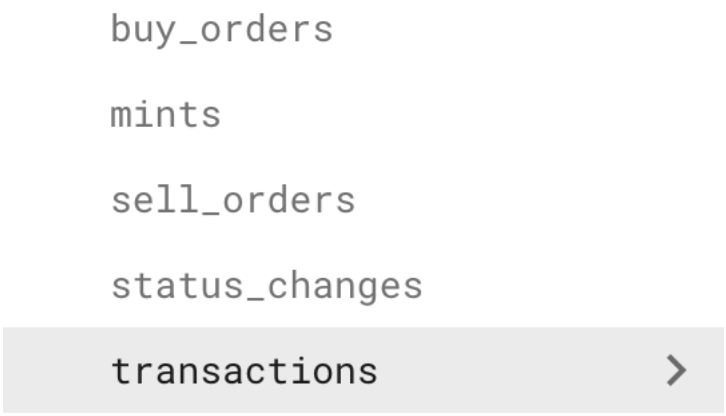


Figure 12. Database collections

The ‘buy_orders’ and ‘sell_orders’ hold the order books. Whenever a user creates an order, the document which has the ‘address’ field equal to the user’s blockchain address is updated with the parameters of the new order. The database order documents contain the following fields: ‘amount’ (number of tokens in the order), ‘price’ (price per token in WEI) and a ‘log_order’ (use to query orders in chronological order). An example of an order document can be seen Figure 13:

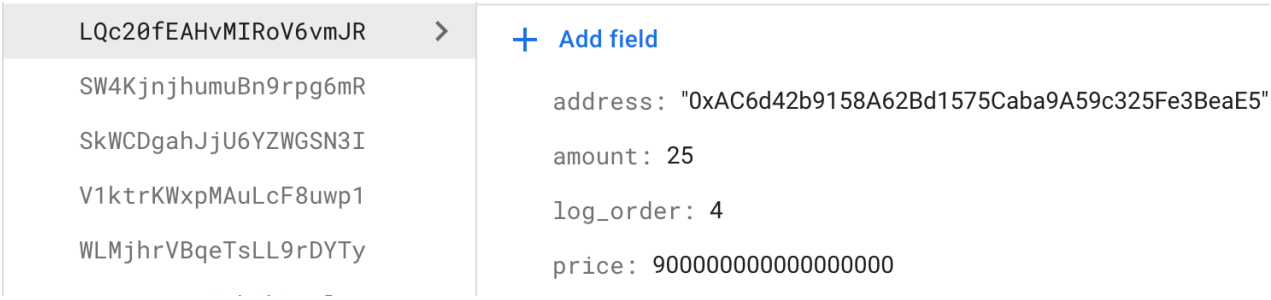


Figure 13. Order document example

The ‘mints’ collection contains the events that were emitted every time a mint has taken place. The ‘timestamp’ field represents the timestamp at which the event was emitted (Figure 14).

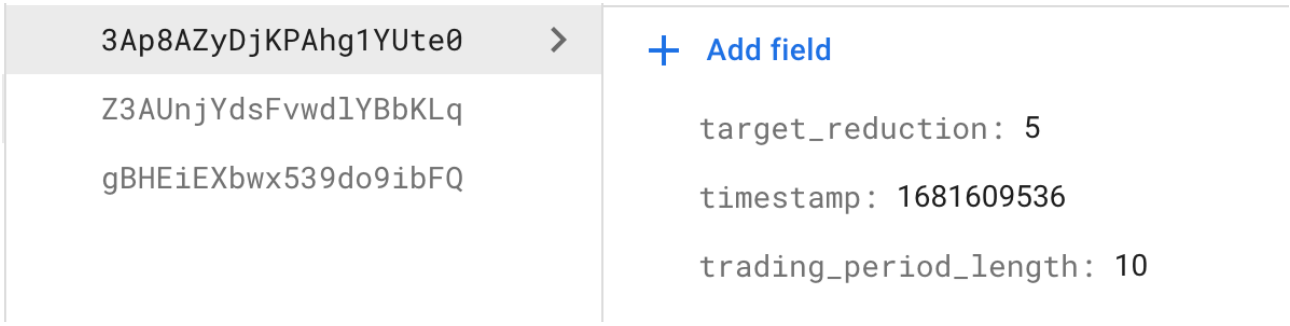


Figure 14. Mint document example

The ‘status_changes’ collection holds events that were emitted whenever the status of an account was changed. The ‘status’ field represents the new status of the address (Figure 15).

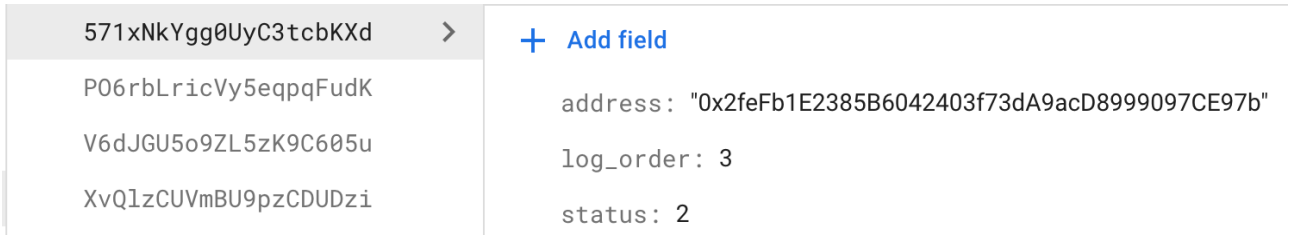


Figure 15. Status change document example

The ‘transactions’ collection holds all token transactions that have taken place on the system’s smart contracts. The fields of each transaction document represents the parameters of each trade, such as the buyer and seller’s address, amount and price (Figure 16).

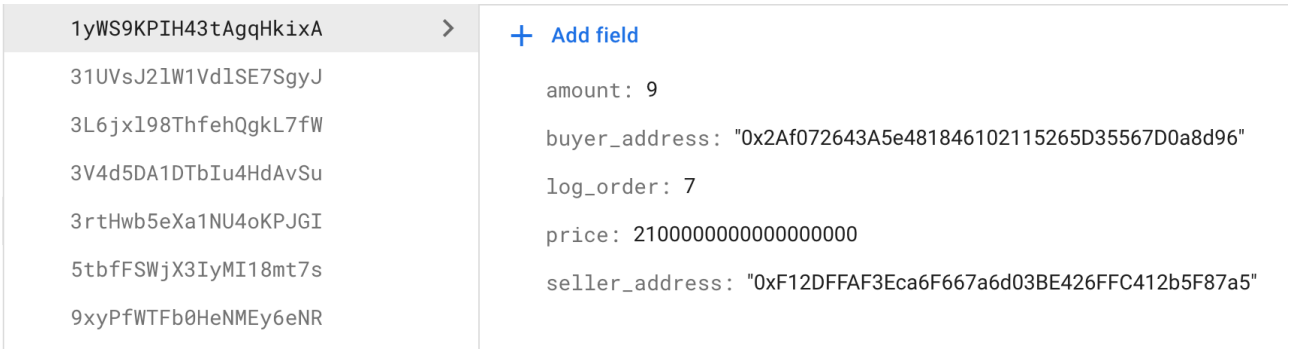


Figure 16. Transaction document example

4.2.3.2 Security

Firestore allows developers to set security rules for their database. For example, there is the functionality to only authorise certain users to make writes or reads. This can be useful in our system.

We might want to allow anyone to read from the database because it holds data that is already publicly available on the blockchain, but only allow our server to write data to the database.

4.2.3.3 Real-time updates

FIrebase Firestore has the ability to send fast, reliable data updates to our web app. By using the **onSnapshot** function, any data that we have on our web app will be instantly refreshed when updated in the database. This functionality is crucial in the trading mechanism.

4.2.4 Functional requirements

4.2.4.1 Server

Number	Functional requirement	MoSCoW ranking
1	The server must listen to all mint, transaction, buy or sell orders and status change events emitted by the smart contract.	M
2	It must ensure that all events are logged, including those triggered when the server was down.	M
3	It must maintain an off-chain order book in the database containing all buy and sell orders created on the smart contract.	M
4	It must update buy and sell orders inside the off-chain order book in the database when buy or sell events are emitted by the smart contract while the server is running.	M
5	It must log to the database all transaction, status change and mint events emitted by the smart contract while the server is running.	M
6	It should delete old data inside the Firebase Firestore database on new runs and repopulate with past events, to keep the database complete and fresh.	S
7	It could provide support for additional smart contracts to log their events to the Firebase Firestore database.	C
8	It could log the block number of the last logged event to resume logging past events from that point in time, rather than repopulating the entire database with all the events emitted in the past.	C

Table 5. Server Functional Requirements

4.2.4.2 Database

Number	Functional requirement	MoSCoW ranking
1	The database must be secure and only allow writes from permitted users.	M
2	It must return an error code and description when there is a failure submitting information.	M
3	It must return all data queried by the web application.	M
4	It should allow any user to view data on it.	S

Table 6. Database Functional Requirements

4.2.5 Testing

Several checks were conducted to make sure that the server and the database function correctly.

Test number	Test description	Expected result	Actual result
1	Delete the old collections	The server should delete all the contents of the current collections inside Firestore on every start	PASS - the server deletes all collections on start
2	Populating collections with old events	The server should repopulate all collections with old events (emitted before the current time) on every start, after emptying the collections	PASS - the server correctly repopulates all collections with old events on every start
3	Listening to status change events	The server should listen to all status change events emitted by the smart contract and correctly update the status of the event address inside the Firestore collection with the new value	PASS - the server listens to status change events and correctly updates the status on Firestore
4	Listening to mint events	The server should listen to all mint events and add them to the ‘mints’ Firestore collection	PASS - the server adds mint events to Firestore
5	Listening to transaction events	The server should listen to all transaction events (when 2 companies exchange allowance) and add them to the Firestore ‘transactions’ collection, with the log_order field of the last added transaction event + 1	PASS - the server adds transaction events to Firestore
6	Creating a buy order	The server should create a new document inside the ‘buy_orders’ collection whenever a company first creates a future buy order, and input the correct data inside the document fields from the parameters of the event	PASS - the server successfully adds new buy orders to Firestore
7	Creating a sell order	The server should create a new document inside the ‘sell_orders’ collection whenever a company first creates a future sell order, and input the correct data inside the document fields from the parameters of the event	PASS - the server successfully adds new sell orders to Firestore

Test number	Test description	Expected result	Actual result
8	Updating buy orders	The server should update the input fields with the correct event parameters inside the buy order document for the address that emitted the event when an event which updates their buy order is emitted	PASS - the server correctly updates a buy order inside the 'buy_orders' collection when an event is emitted
9	Updating sell orders	The server should update the input fields with the correct event parameters inside the sell order document for the address that emitted the event when an event which updates their sell order is emitted	PASS - the server correctly updates a sell order inside the 'sell_orders' collection when an event is emitted
10	Deleting buy orders	The server should set the amount and price fields to 0 inside the buy order document of an address when the smart contract emits an event for deleting their buy order	PASS - the server correctly deletes a buy order
11	Deleting sell orders	The server should set the amount and price fields to 0 inside the sell order document of an address when the smart contract emits an event for deleting their sell order	PASS - the server correctly deletes a sell order
12	Console log	The server should log to the console the parameters of an event whenever it is emitted by the smart contract	PASS - the server logs events to the console

Table 7. Server and database tests

4.3 Web App

The purpose of the web app is to provide a graphical interface to our system. It represents an intuitive and simple way to use the functionality provided by the smart contracts.

4.3.1 Prerequisites and development tools

HTML and CSS were used to create the design of the web app. To create functionality, the ReactJS library for JavaScript was used. There are multiple reasons why React was chosen, instead of other alternatives like Vue.js, Next.js or Angular. React has a component-based architecture, which enables developers to reuse UI components. Also, it creates a virtual DOM, which allows to render only parts of the web page, instead of reloading the entire page.

The Web3.js library was used to connect to the smart contracts and to access their functionality. It was selected over alternatives such as Ethers.js because it supports more features of EVM networks and it has a higher quality documentation.

In order to connect to a blockchain and to sign transactions, the MetaMask wallet was used. It allows users to select the blockchain which holds the smart contracts and to log in to the web app using their blockchain address.

4.3.2 Software design choices

4.3.2.1 InterPlanetary File System (IPFS) integration

The IPFS is a distributed and decentralised file system that can be used as a form of storage (Kumar and Tripathi, 2020). It provides qualities such as decentralisation and immutability, and it comes with a lower associated cost than saving data on a blockchain. For these reasons, long strings containing user details were stored on the IPFS, with a reference to them stored on chain.

Infura was used to connect to the IPFS. It provides an API service, which makes it simpler to access the functionality of the IPFS, without having to run your own node.

4.3.2.2 Accounts, roles and menus

The web app can only be accessed if users connect with their MetaMask wallet. The system then constantly checks inside the smart contract the account status of their connected blockchain address. Variating functionality inside the web app can be accessed depending on the permission level of the user.

Depending on their permission there are three types of user roles, and they can access different pages inside the navigation menu:

User role	Menu
Unauthorised user	Active Companies, Trading Platform, Account, Guide
Authorised user	Active Companies, Trading Platform, Account, Guide
Admin user	Active Companies, Trading Platform, Admin Dashboard

Table 8. Account roles and the pages they can access

Even though authorised and unauthorised users have access to the same pages, some functionality might be restricted from the unauthorised users, such as creating trades (Figure 17).

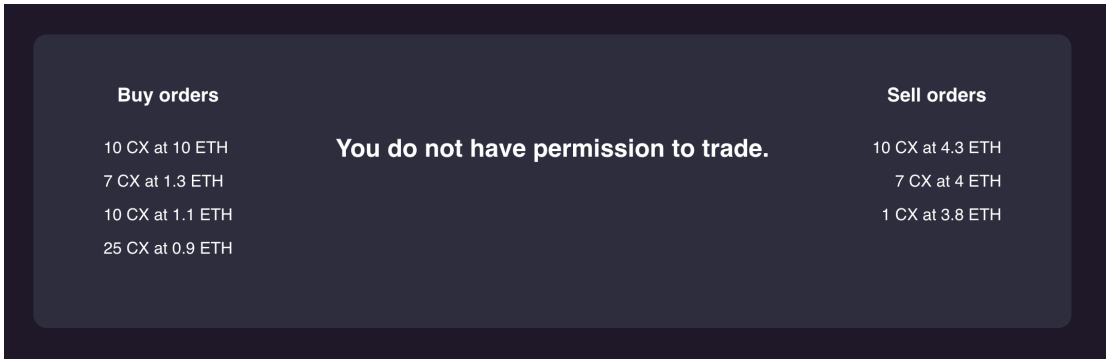


Figure 17. Unauthorised user restriction message

4.3.2.3 Asynchronism and data refreshing

Cutting short the loading times of the app was a constant focus over the course of the development process. In the first iteration, data from the blockchain used in a page was loaded every time the page was accessed. In order to harvest the capabilities for asynchronism of ReactJS, a new solution was devised using a different type of hook.

As an improvement, data used in the entire app is pulled from the blockchain once when the user first accesses the web app. The user can browse different pages of the app while data is being loaded. Also, after the initial load, the data is kept fresh by listening to events emitted by the smart contracts. Even if the user is on a different page, data on all pages is constantly updated in the background by listening to events.

The web app also pulls data from the Firebase Firestore Database. It is pulled once when the page is loaded and then constantly updated and refreshed using the **onSnapshot** function provided by Firestore. By using this function, the Database can notify the web app whenever data is updated and refresh it instantly for the user. Where possible, data is pulled from the Database instead of the blockchain, for the shorter query time.

4.3.2.4 Modals

Modals were used to keep the user informed of the status of the platform. Loading screens are displayed while data from the blockchain is being loaded. Also, one-time messages are displayed when various actions are completed, informing the user of the success or failure of their operation.

4.3.2.5 User Experience

The design of the web app is simple and consistent. The layout of the components is well spaced and uncluttered. The app has a responsive design, and components will be well aligned if the window is resized up to a point.

The web app does not support well mobile screen size, because it is not intended to be used on mobile devices, such as phones or tablets. Its functionality is linked to MetaMask, which is not yet available as an extension on mobile browsers.

4.3.3 Functional Requirements

4.3.3.1 Web App

Number	Functional Requirement	MoSCoW ranking
1	The web app must only allow logged in users to progress beyond the Log in page.	M
2	The web app must trigger the Metamask wallet for method calls to the smart contract.	M
3	The web app must retrieve relevant data from Firebase Firestore and from the Inter Planetary File System (IPFS).	M

Number	Functional Requirement	MoSCoW ranking
4	The web app must keep the user informed about the success or failure of their operations.	M
5	The web app should display the address of the current account on every page.	S
6	The web app could provide news and updates about the carbon market	C
7	The web app could integrate different external wallets, other than Metamask.	C
8	The web app could provide a chart of historical trading data.	C
9	The web app won't allow users to trade other cryptocurrencies or tokens.	W

Table 9. Web App Functional Requirements

4.3.3.2 Log In page

Number	Functional Requirement	MoSCoW ranking
1	It must allow the user to log in with their Metamask wallet.	M
2	It could allow the user to log in with other wallets.	C

Table 10. Log In page Functional Requirements

4.3.3.3 Navigation Bar

Number	Functional Requirement	MoSCoW ranking
1	It must contain relevant navigation tabs for the different kind of users (normal user or admin).	M
2	It must allow the user to Log out of the web app.	M
3	It must allow the user to navigate between tabs.	M

Table 11. Navigation Bar Functional Requirements

4.3.3.4 Active Companies page

Number	Functional Requirement	MoSCoW ranking
1	It must retrieve relevant company data from the smart contract.	M

Number	Functional Requirement	MoSCoW ranking
2	It must display all companies that have a trading account on the app.	M
3	It must display all trades performed by every company, as well as their relevant details and token balance.	M
4	It must update the token balance for the company if it is changed on the smart contract, by listening to events.	M
5	It could allow the user to see images of each company.	C

Table 12. Active Companies page Functional Requirements

4.3.3.5 Guide page

Number	Functional Requirement	MoSCoW ranking
1	It must display instructions about how to operate the web app.	M
2	It must contain a table of contents with links to the appropriate instructions.	M

Table 13. Guide page Functional Requirements

4.3.3.6 Account page

Number	Functional Requirement	MoSCoW ranking
1	It must allow the user to sign up.	M
2	It must keep the user informed of the status of their request to sign up.	M
3	It must fetch from the blockchain and IPFS the current details of the user's account and display them.	M
4	It must update the user's details if they are updated in the smart contract by listening to events.	M
5	It could allow the user to edit their account details or delete their account.	C

Table 14. Account page Functional Requirements

4.3.3.7 Trading Platform page

Number	Functional Requirement	MoSCoW ranking
1	It must only allow registered users to place buy or sell orders and make trades.	M
2	It must only allow trading to take place if the trading period is open.	M
3	It must fetch and update in real time orders from the Firebase Firestore order book.	M
4	It must allow users to place Future and Spot orders.	M
5	It must display the allowance of the current account.	M
6	It must trigger the Metamask wallet whenever a trade or an order is placed.	M
7	It must show the current open buy or sell orders of the user.	M
8	It must allow the user to close their open buy or sell orders.	M
9	It should inform the user of the success or failure of their transactions.	S
10	It should display the past 10 transactions	S
11	It should not allow the admin to place trades.	S

Table 15. Trading Platform page Functional Requirements

4.3.3.8 Admin Dashboard page

Number	Functional Requirement	MoSCoW ranking
1	It must allow the admin to set the mint parameters.	M
2	It must allow the admin to mint tokens and start the trading period.	M
3	It must allow the admin to view the parameters of the ongoing trading period and to to end the trading period.	M
4	It must allow the admin to view pending sign up requests for the trading platform and to approve or deny them.	M

Table 16. Admin page Functional Requirements

4.3.4 Functionality and components

The web app contains multiple components in the form of pages. Their purpose is to create an interface for users to view blockchain data (company details and their trading history), and access the smart contracts functionality.

The following section will introduce some of the main components of the web app:

4.3.4.1 Admin Dashboard page

This page (Figure 18) is where the admin can set the parameters of a trading period, start it, and mint tokens (Figure 19). If the trading period is live the admin can see its details. The page also displays a list of pending accounts. The admin has the ability to approve or deny their sign up request (Figure 20).

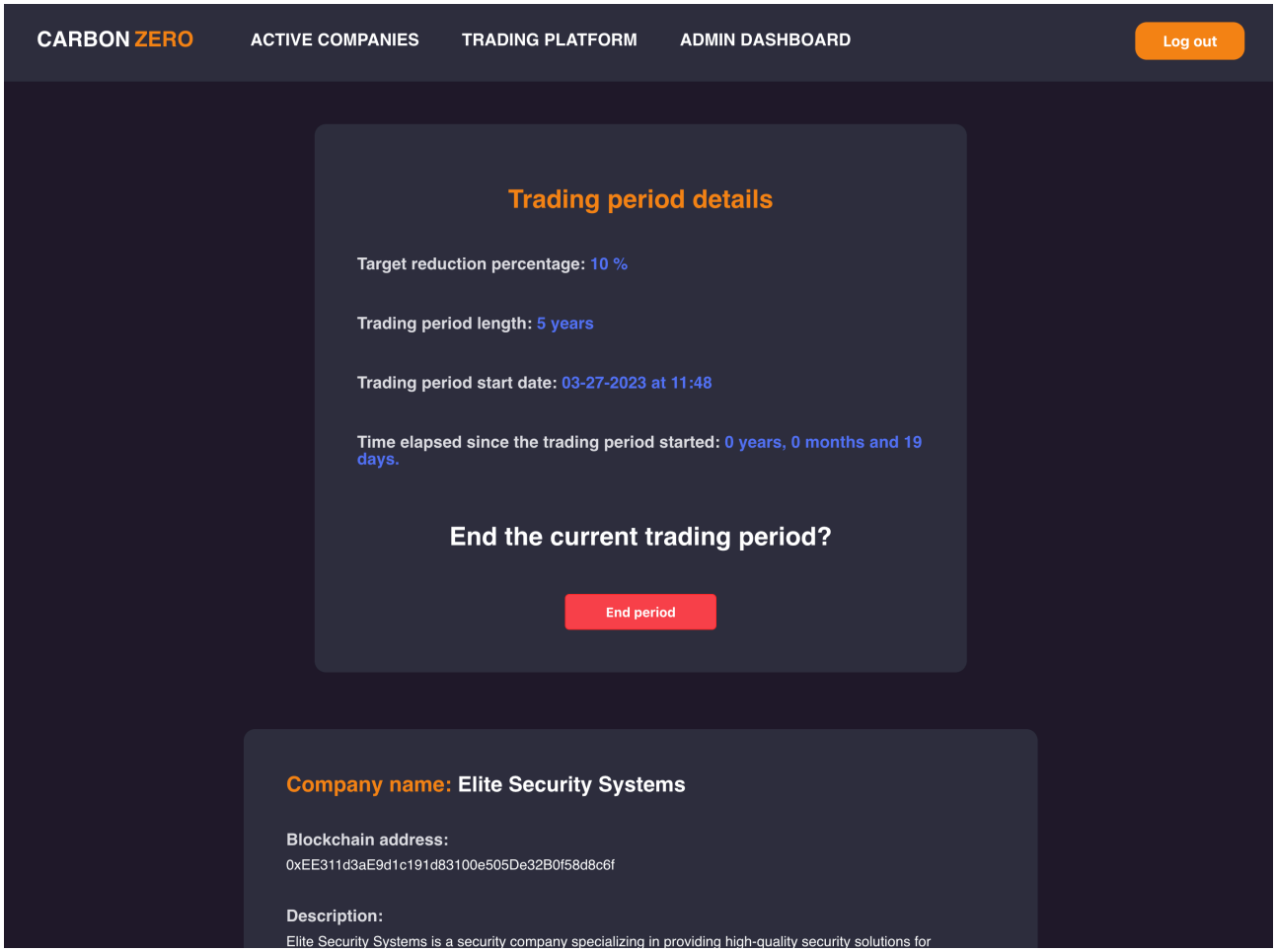


Figure 18. Admin Dashboard page

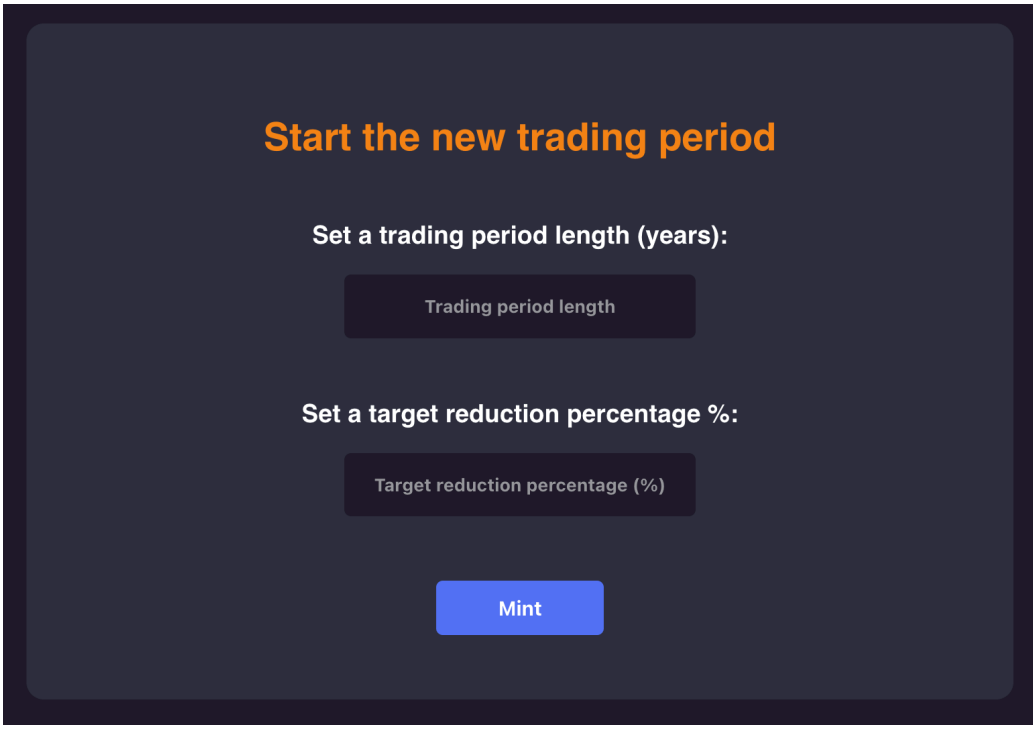


Figure 19. Setting the parameters of the new trading period

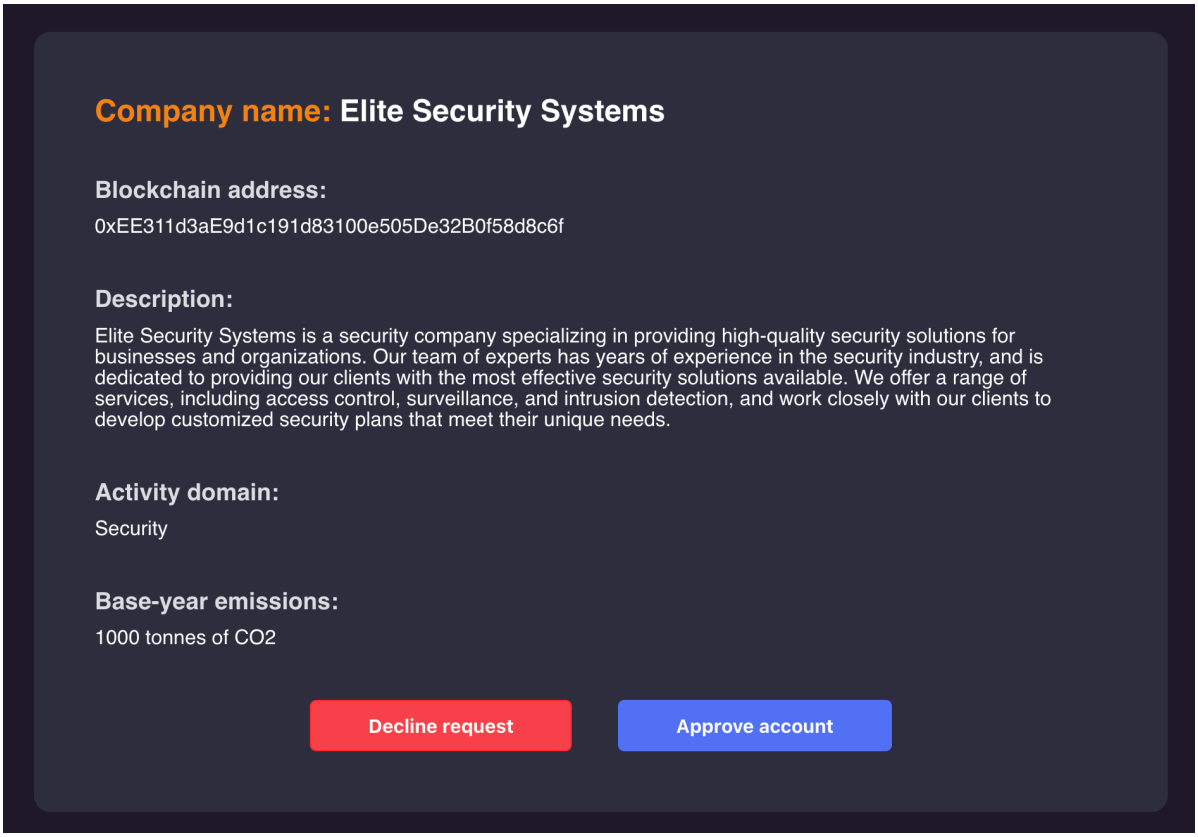


Figure 20. The admin can approve or deny sign up requests

4.3.4.2 Account page

This component provides the functionality for users to send their details to the admin and request to join the platform (Figure 21) or view their details if they have already been authorised (Figure 22).

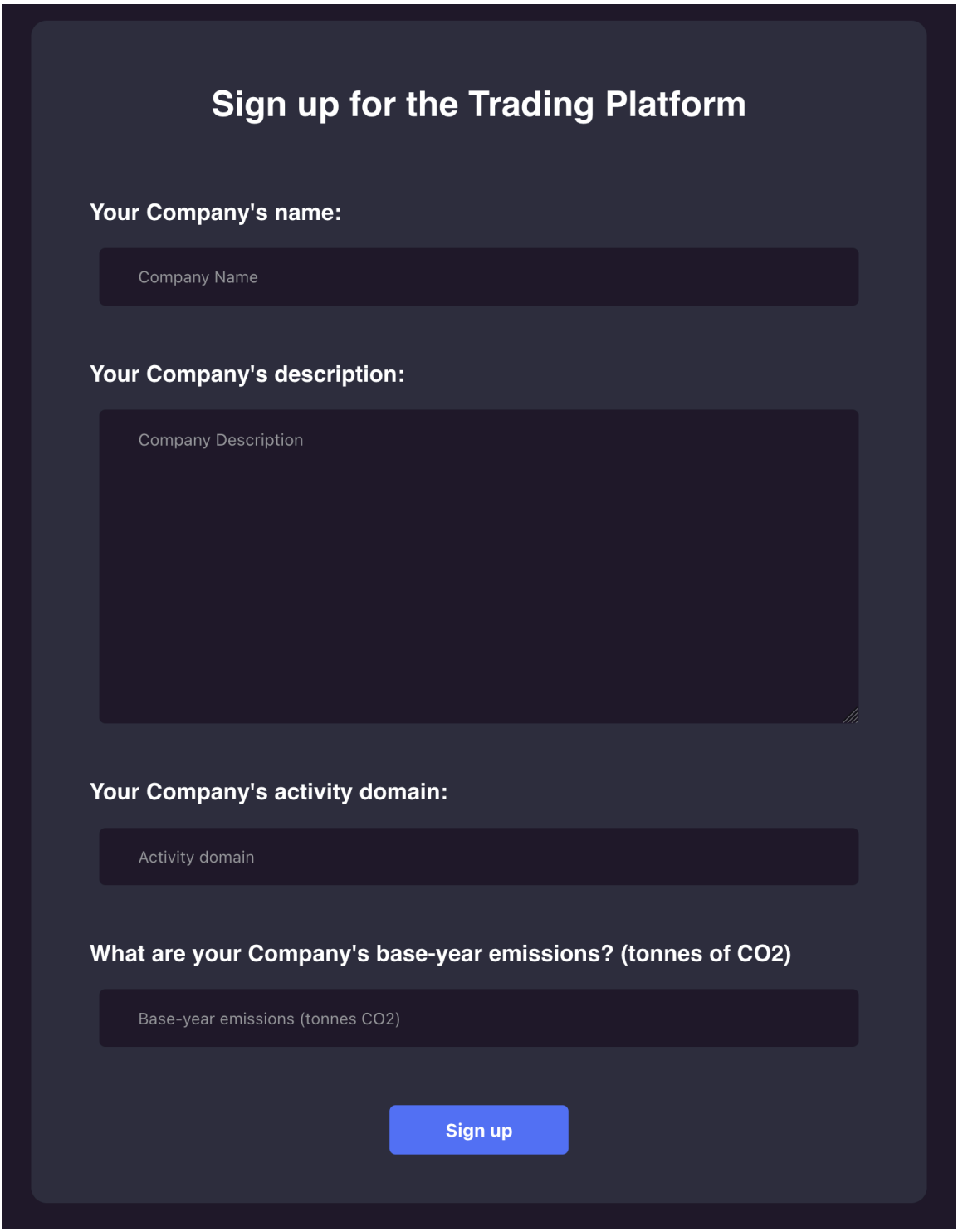


Figure 21. Sign up form for unauthorised users

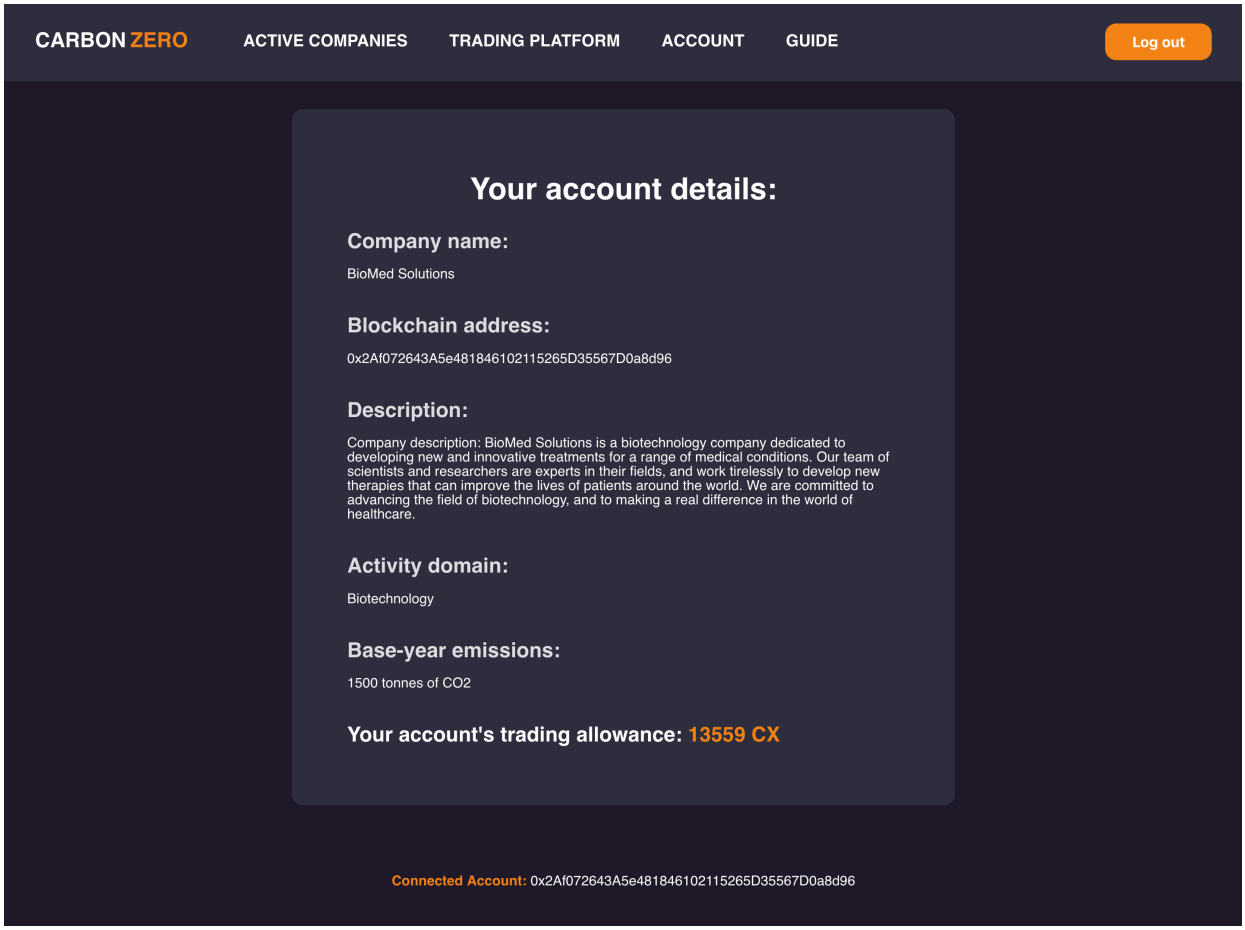


Figure 22. The Account page where authorised users can view their details

4.3.4.3 Guide page

Provides a set of instructions on how users can operate the platform.

4.3.4.4 Active Companies page

This page provides a full view of what data is saved on the blockchain. It contains a list of all the active companies that have signed up to the platform. It displays all of their details, using combined data from both the chain and the IPFS. It also displays a full list of all of the token transactions that each company has made and their up to date allowance. This is where users can go to inspect how a company manages their emission allowance portfolio.

The details and trades of each company appear on a card, as seen in Figure 23.

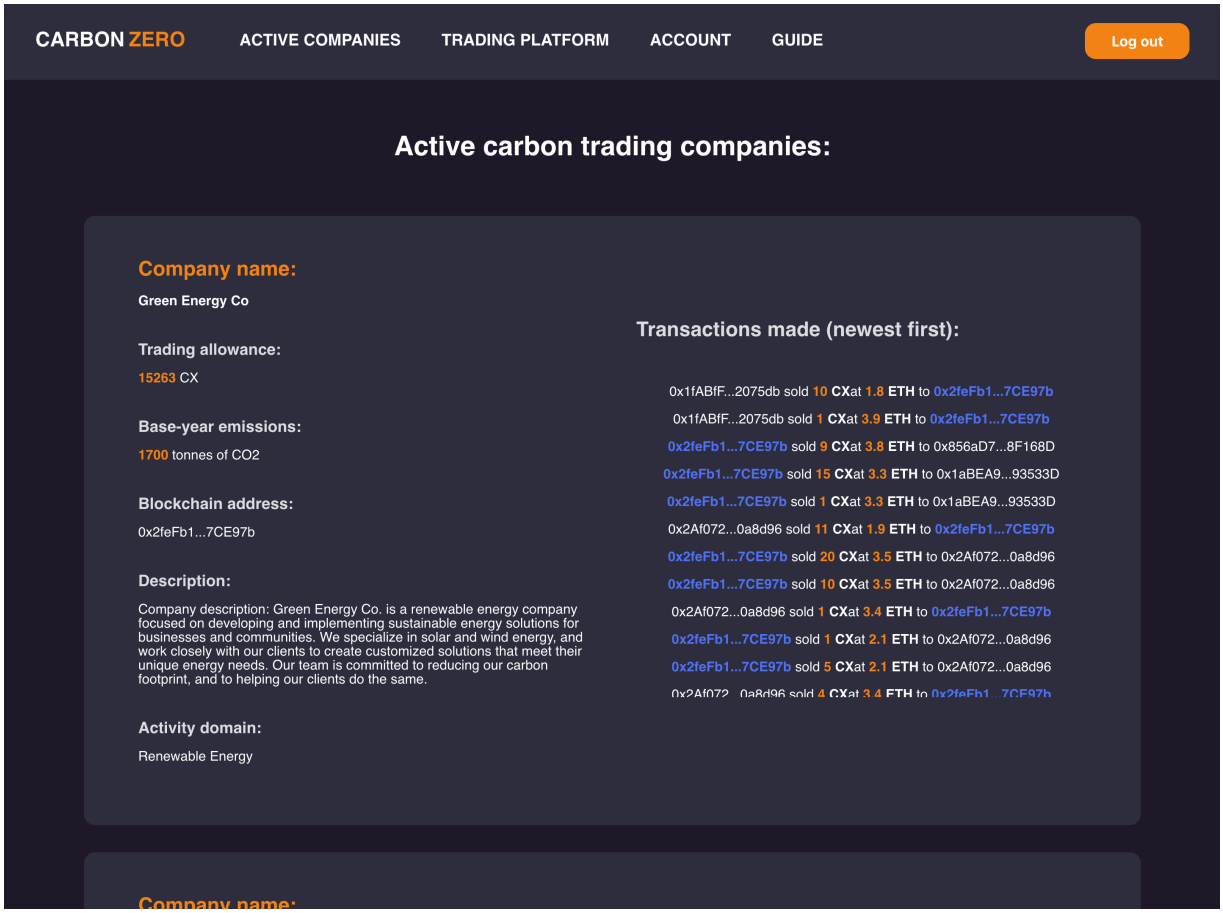


Figure 23. Details of active company on Active Companies page

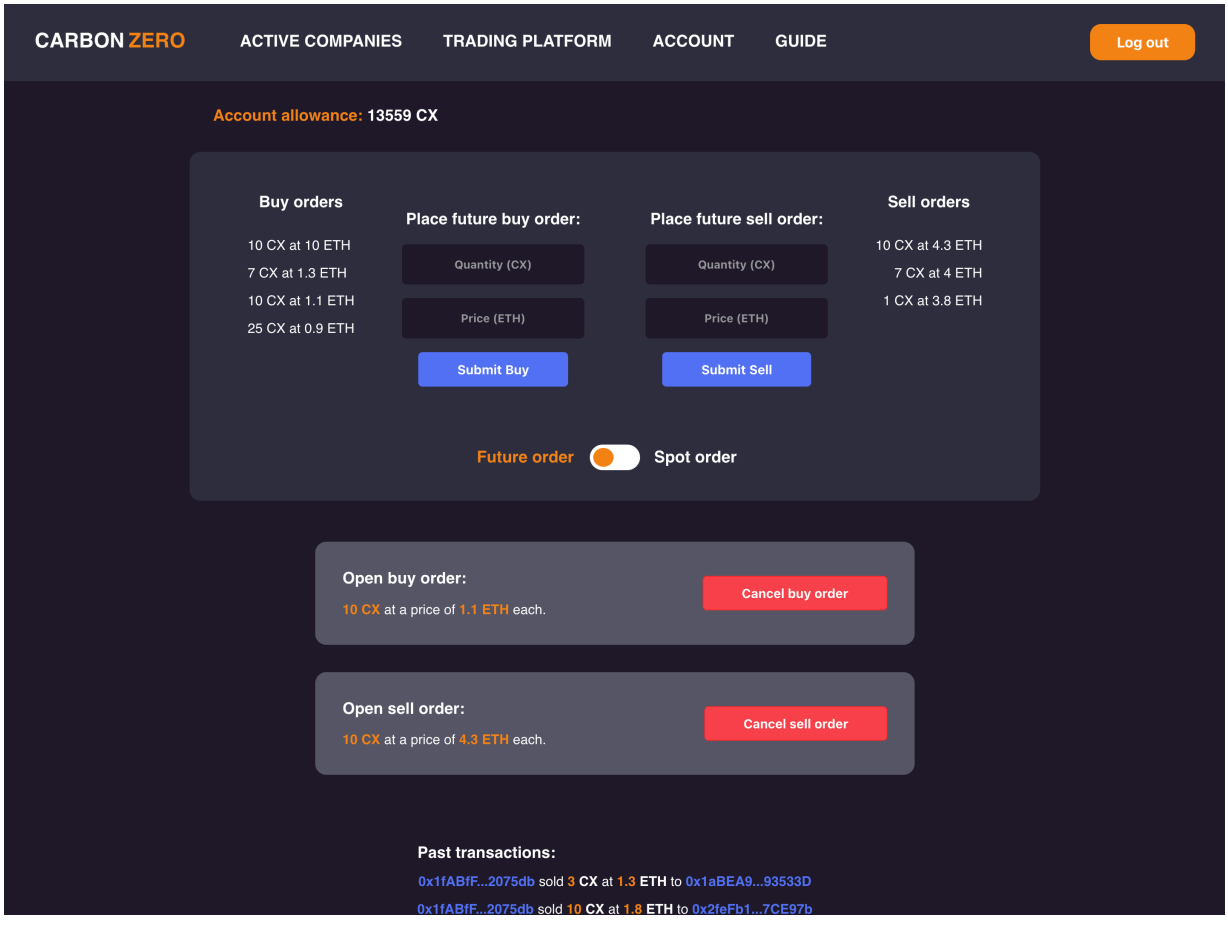


Figure 24. Screen capture of the Trading Platform page

4.3.4.5 Trading Platform page

This component (Figure 24 above) holds the trading functionality of the app. Users can add Future orders (Figure 25) that will be added to the order book, and Spot orders (Figure 26) that will be fulfilled from the order book.

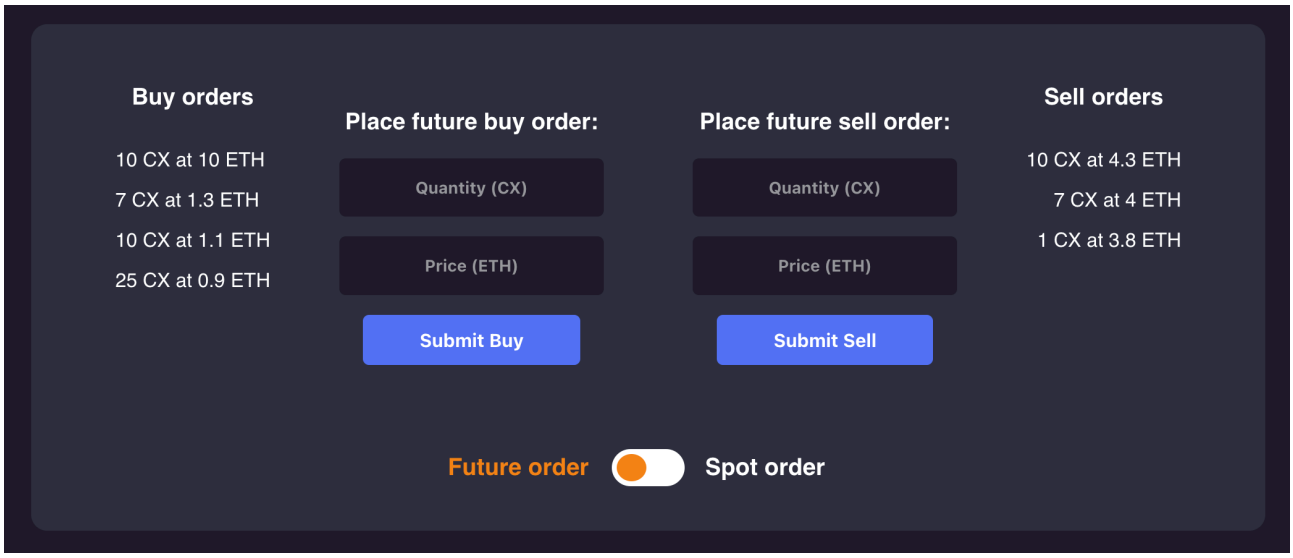


Figure 25. Users can place Future orders in the Trading Platform

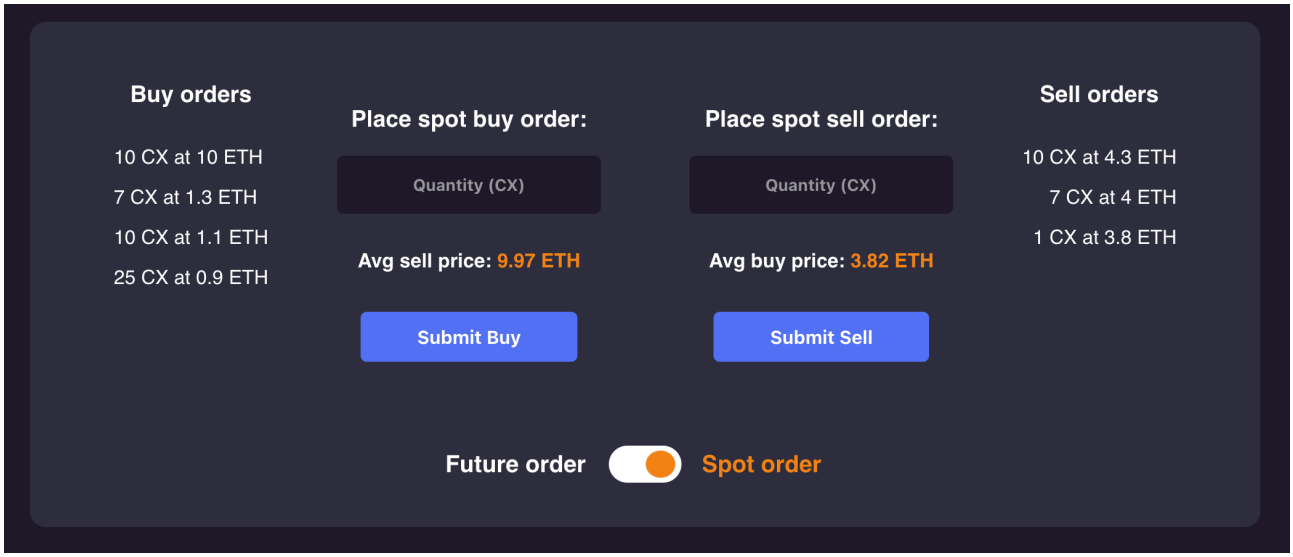


Figure 26. Users can place Spot orders in the Trading Platform

When creating a Future order, the user agrees to sell or buy a certain amount of tokens at a set price in the future. When creating a Spot order, the user's order will be matched with the Future orders readily available in the order book. A Spot sell will be matched with Future buy orders sorted decreasingly by price and a spot buy will be matched with future sell orders sorted increasingly by price. Therefore, a seller will get to sell to the best price available, and a buyer will get to buy from the lowest price on offer.

If the Spot order amount can not be fulfilled with only one Future order, multiple successive transactions will be triggered, and the order will be filled from several Future orders until an error is reached or the order is fulfilled.

Users can also see and delete their open Future orders (Figure 26).

Figure 27. Users can see the details of their open Future order and cancel it

4.3.5 Security features

The app has several security features. Authentication is achieved through the use of the MetaMask wallet to access functionality and sign blockchain transactions. Therefore, an attacker does not have the ability to pretend to be a different user, unless they get access to their wallet.

Users with no permissions are blocked from accessing the Admin page by the way the app is routed. In the unlikely event that an unauthorised user manages to get access to components that were designed for users with higher authorisation, they will be unable to use their functionality because of the architecture of the smart contracts. Modifiers inside the smart contracts will provide another layer of security.

The web app sanitises every piece of input that is fed into its input fields. Users are not allowed to send any blank forms or provide data of the wrong type.

4.3.6 App-wide manual testing

Extensive manual testing was performed on all the web app's functionality, to make sure that the expected behaviour is achieved. All of the buttons and front-end functionality, as well as the content that is displayed were rigorously tested using both manual and user tests.

Test number	Test description	Expected result	Actual result
1	Right pages for different user types	The navigation bar displays the correct menu for Admin user and for normal users	PASS - the navigation bar displays the correct components
2	Log in button triggers Metamask wallet	The metamask wallet should be activated and set to Log In when the Log in button is pressed	PASS - the Metamask wallet is correctly triggered

Test number	Test description	Expected result	Actual result
3	Correct blockchain address is displayed	The web app should display the correct blockchain address that is currently connected at the bottom of each page	PASS - the web app displays the correct address
4	Correct blockchain address is displayed when user switches accounts	The web app should display the new blockchain address at the bottom of each page when the user changes accounts on Metamask	PASS - the web correctly displays the new address
5	Redirect when user changes accounts	The web app redirects the user to the Active Companies dashboard page when the blockchain address is changed from Metamask	PASS - the web app successfully redirects the user
6	Navigation bar buttons redirect to the right pages	Clicking on any menu button on the navigation bar redirects the user to the correct page	PASS - the navigation buttons redirect to the correct pages
7	Log out	Clicking on the Log out button on the Navigation bar redirects the user to the Log in page	PASS - the user is successfully redirected
8	Guide page	The Guide page correctly displays the instructions for using the web app	PASS - the Guide page is successfully displayed
9	Blocked access for non-admin users	Non-admin users should be blocked from using the restricted pages that are reserved for Admin only (accessing them by page link)	PASS - non-admin users are blocked and redirected from Admin page
10	Correct Account page content for the status of the current account	The Account page should display either a Sign Up form, a Pending request frame or the Account details, depending of the current status of the user's account	PASS - the Account page displays the correct content

Table 17. Manual testing results

See Appendix C for the rest of the manual tests and results.

5. Project management

The SCRUM Agile framework was used to manage the workflow of the project in order to create such a complex system in the limited given time-frame. SCRUM is a framework that promotes flexiblity in development, continuous improvement and collaboration. It is focused on splitting the work into short blocks called sprints.

The development of the entire system was split into a total of 7 sprints. Each sprint took between 2 to 4 weeks to complete, depending on the complexity of the tasks that needed to be fulfilled. Towards the end of each sprint a list of tasks for the upcoming sprint was devised.

The meetings with the Project Supervisor took place towards the end of each sprint. They were useful in receiving feedback on the tasks that were already achieved and input for the tasks that were planned for the upcoming sprint.

The following table shows each sprint that was completed and the tasks were achieved:

Sprint number	Tasks
Sprint 1	-Research the carbon emissions trading system -Literature and software solutions review -Choose the tech stack for the system and set up the web development environment
Sprint 2	-Understand the basics of the blockchain and how EVM Smart Contracts work -Learn the basics of ReactJS -Research how the emission trading system can be implemented on the blockchain -Research on the off-chain order book implementation -Devise a system architecture
Sprint 3	-Set up of the Ganache blockchain -Set up the smart contracts development environment -Begin implementation of the Sign Up smart contract -Create the first React page and connect to the Database -Set up a React router to manage multiple pages -Begin implementation of the Trading Platform Page and Admin Dashboard Page front end
Sprint 4	-Create more front end components of the Web App (Account page and Company Dashboard) -Advance in the development of the functionality of the web app components that were created in the past sprint -Wrap up implementing the Sign Up contract -Begin implementation of the Trading Platform Smart Contract -Begin implementation of the Server

Sprint number	Tasks
Sprint 5	-Finish developing the Server -Finish implementing the smart contracts -Finish the functionality of the web app -Create the Guide page of the web app
Sprint 6	-Refactoring of smart contracts for gas optimisations -Work on the styling of the web app -Write unit tests for the smart contracts -Run manual tests for the web app and server
Sprint 7	-Conduct user evaluation -Devise future plans and write a conclusion -Finish writing the report

Table 18. SCRUM sprints and their achieved tasks

6. Evaluation

In hindsight, the development process went well and most of the plan was followed. There were times where important changes had to be made to what was planned, but every aspect was taken into account and the most appropriate solution was taken (eg. building an off-chain order book instead of the planned on-chain order book). The process of production was always focused on feedback and constant improvement. The performance of the app was constantly improved in ways such as modifying the way data was pulled from the blockchain, and gas optimisations to lower operating costs.

Besides unit testing and manual testing, user testing was conducted to evaluate the created system. A set of tasks was given to 10 subjects to test a number of scenarios from the perspective of two types of end users: companies and the admin. The evaluation was conducted in 1-on-1 meetings. After completing the tasks, a questionnaire was provided.

The first part of the questionnaire covered the background of the questionnaire subjects. An important aspect that was pursued was to receive feedback from persons with mixed backgrounds, to account for the types of end users that will utilise the platform. Because 80% of the subjects had Software Engineering experience, their input was crucial to test the quality of the software. The non-technical subjects provided a good view of how usable the app was to an inexperienced user. Most of the persons asked had heard about emissions trading before and could express educated opinions on the subject. An important question was about their experience with different blockchains, and EVM blockchains ranked high in the usage list.

The second part of the questionnaire was focused on Nielsen's (1994) Ten Usability Heuristics:

-Aesthetic and minimalist design;
-Consistency and standards;
-Recognition rather than recall;
-Error prevention;

-Help users recognise, diagnose, and recover from errors;
-User control and freedom;
-Flexibility and efficiency of use;
-Visibility of system status;
-Match between system and the real world;
-Help and documentation.

Feedback from the users (see in Appendix E) shows that most of the Usability Heuristics were respected. The only heuristic that could not be tested was 'Help users recognise, diagnose, and recover from errors', because no errors were reached.

The final part of the questionnaire was open ended and focused on general feedback. Everything that was suggested here was later implemented in the app, such as renaming several pages and changing some of the web app's design.

As a conclusion, feedback from users was positive. The functionality of the app is clear, no errors were reached and the design is pleasant and aesthetic.

7. Future Work

The achievement of this project is the creation of a secure and transparent platform that has the functionality to assign and trade carbon emission rights. It was inspired by the traditional platforms used today, such as the European Energy Exchange or the Intercontinental Exchanged, and also by the literature written for blockchain solutions. It encompasses two of the three systems introduced by Pan *et al.* (2018) (registration and trading) and implements functionality based on permissions, such as what was described by Khaqqi *et al.* (2018).

The current version of the platform can be used as a complete carbon trading system, however there is plenty of future work that can still be done to improve and take it to the next level:

Future targets	Value increase
External security audit	An external security audit would bring more trust to the platform. Although a full security audit was conducted, a common practice in smart contract development is to hire external help to conduct additional security checks. Having a different developer than the one who wrote the code perform an audit can make it easier to spot logic flaws.
Government partnership	In order to actually implement the system in today's trading market, support from the governments is needed. The system could be fine-tuned for their needs, and it would attract companies to our trading platform.
Multiple blockchain implementation	It would create the ability for potential customers to use a blockchain of their choosing to create an account and trade. This could increase adoption and ease of use.
Corporate emission management system	At the moment, the platform contains two out of the three systems introduced by Pan *et al.* (2018). Adding the management system would give companies the ability to securely log their emission levels and it would make it easier for regulators to oversee allowance offset.

Future targets	Value increase
More token types	Besides the Kyoto units currently implemented, other types of carbon tokens can be added to the platform, such as emission reduction units or certified emission credits. This would make it easier to support and invest into green projects. Having more types of tokens also implies having more flexibility in trading and portfolio management.
Pool of funds	The functionality to create and donate to a pool of tokens could be implemented in the app. This would make it easier for normal customers to donate to green projects and help save the environment.
More types of orders and trading tools	The platform currently implements spot and futures orders. More types of orders (such as stop-order, stop-limit orders etc) would give users more flexibility in trading. Combining additional orders with additional trading tools (such as graphs) would create the ability to devise and follow more complex trading strategies.

Table 19. Future work that can be done to improve the project and the value it would bring

8. Conclusion

The final result of this final year project is a complex platform that allows for a better, more transparent carbon emission market. Many important design choices had to be made and new challenges came up along the way. The end product has complex functionality and it represents a more refined blockchain solution to assign and trade carbon emissions, taking the current innovation in the field even further. For me, it was an immense learning experience. It took me from being a complete blockchain novice to writing my first complex smart contracts. I learned a lot about web development, self management, and writing a project report. In the end, by improving the carbon trading mechanisms, this project is a step forward towards a greener, less polluted environment.

9. References

AirCarbon (n.d.) AidCarbon
Available at: https://www.aircarbon.co/carbon-assets
Accessed: 20 April 2023

Al Sadawi, A. and Ndiaye, M. (2022) “Blockchain-based carbon trading mechanism to elevate governance and smartness.” 14th International Conference on Theory and Practice of Electronic Governance (ICEGOV ’21). Association for Computing Machinery, New York, NY, USA, 34–43.
Available at: https://doi.org/10.1145/3494193.3494198

CarbonEX (n.d.) CarbonEX
Available at: https://carbonex.co/
Accessed: 20 April 2023

CME Group (n.d.) Trading Energy Emissions
Available at: https://www.cmegroup.com/trading/energy/emissions/?redirect=/trading/energy/emissions/index.html
Accessed: 20 April 2023

Environmental Protection Agency (n.d.) The EU Emissions trading system
Available at: https://www.epa.ie/our-services/licensing/climate-change/eu-emissions-trading-system-/
Accessed: 19 April 2023

European Commission (n.d.) EU Emissions Trading Systems
Available at: https://climate.ec.europa.eu/eu-action/eu-emissions-trading-system-eu-ets_en
Accessed: 19 April 2023

European Commission (n.d.) Revision for phase 4
Available at: https://climate.ec.europa.eu/eu-action/eu-emissions-trading-system-eu-ets/revision-phase-4-2021-2030_en
Accessed: 19 April 2023

European Energy Exchange (n.d.) Chinese Carbon Market
Available at: https://www.eex.com/en/markets/environmental-markets/china-carbon
Accessed: 20 April 2023

European Energy Exchange (n.d.) EU ETS Auction
Available at: https://www.eex.com/en/markets/environmental-markets/eu-ets-auctions
Accessed: 20 April 2023

European Energy Exchange (n.d.) EU ETS Auctions
Available at: https://www.eex.com/en/markets/environmental-markets/eu-ets-auctions
Accessed: 20 April 2023

European Energy Exchange (n.d.) EU ETS Spot Futures Options
Available at: https://www.eex.com/en/markets/environmental-markets/eu-ets-spot-futures-options
Accessed: 20 April 2023

European Energy Exchange (n.d.) NZ ETS Auctions
Available at: https://www.eex.com/en/markets/environmental-markets/nz-ets-auctions
Accessed: 20 April 2023

InterContinental Exchange (n.d.) Carbon Futures
Available at: https://www.theice.com/market-data/indices/commodity-indices/carbon-futures
Accessed: 20 April 2023

Khan, S.N. et al. (2021) "Blockchain Smart Contracts: Applications, challenges, and future trends," Peer-to-Peer Networking and Applications, 14(5), pp. 2901–2925.
Available at: https://doi.org/10.1007/s12083-021-01127-0

Khaqqi, K.N. et al. (2018) "Incorporating seller/buyer reputation-based system in blockchain-enabled emission trading application," Applied Energy, 209, pp. 8–19.
Available at: https://doi.org/10.1016/j.apenergy.2017.10.070

Kumar, R. and Tripathi, R. (2020) "Chapter 2 - Blockchain-Based Framework for Data Storage in Peer-to-Peer Scheme Using Interplanetary File System",
Handbook of Research on Blockchain Technology, 2020, Pages 35-59, ISBN 9780128198162.
London, United Kingdom: Academic Press.
Available at: https://doi.org/10.1016/B978-0-12-819816-2.00002-2

Mohajan, Haradhan (2017) "Greenhouse Gas Emissions, Global Warming and Climate Change." 15th Chittagong Conference on Mathematical Physics. March 2017. University of Chittagong, Chittagong, Bangladesh. Jamal Nazrul Islam Research Centre for Mathematical and Physical Sciences (JNIRCMPS).
Available at: https://www.researchgate.net/publication/315668490_Greenhouse_Gas_Emissions_Global_Warming_and_Climate_Change

Mohajan, Haradhan (2017) "Greenhouse Gas Emissions, Global Warming and Climate Change." 15th Chittagong Conference on Mathematical Physics. March 2017. University of Chittagong, Chittagong, Bangladesh. Jamal Nazrul Islam Research Centre for Mathematical and Physical Sciences (JNIRCMPS).
Available at: https://www.researchgate.net/publication/315668490_Greenhouse_Gas_Emissions_Global_Warming_and_Climate_Change

Muzumdar, A., Modi, C. and Vyjayanthi, C. (2022) "A permissioned blockchain enabled trustworthy and incentivized emission trading system," Journal of Cleaner Production, 349, p. 131274.
Available at: https://doi.org/10.1016/j.jclepro.2022.131274

Narayanan, A., Clark, J. (2017). "Bitcoin's academic pedigree"
Communications of the ACM.
Volume 60, Number 12 (2017), Pages 36-45

Available at: https://dl.acm.org/doi/fullHtml/10.1145/3132259

Nielsen, J. (1994) “Enhancing the explanatory power of usability heuristics,” Conference companion on Human factors in computing systems - CHI '94 [Preprint].
Available at: https://doi.org/10.1145/259963.260333

Pan, Y. et al. (2019) "Application of blockchain in carbon trading," in Energy Procedia, 2019, vol. 158, pp. 4286-4291, doi: 10.1016/j.egypro.2019.01.509.
Available at: https://doi.org/10.1016/j.egypro.2019.01.509

Rawat, V. et al. (2022) “A blockchain-based decentralized framework for carbon accounting, trading and governance,” 2022 8th International Conference on Computer Technology Applications [Preprint].
Available at: https://doi.org/10.1145/3543712.3543755

S. Nakamoto (2008) Bitcoin: A Peer-to-Peer Electronic Cash System
Available at: https://bitcoin.org/bitcoin.pdf
Accessed: 20 April 2023

Sadawi, A.A. et al. (2021) “A comprehensive hierarchical blockchain system for carbon emission trading utilizing blockchain of things and smart contract,” Technological Forecasting and Social Change, 173, p. 121124.
Available at: https://doi.org/10.1016/j.techfore.2021.121124

Skjµrseth, J.B. and Wettestad, J. (2016) EU emissions trading: Initiation, decision-making and implementation. London: Routledge.

Toucan Protocol (n.d.) Toucan
Available at: https://toucan.earth/carbon-projects
Accessed: 20 April 2023

Tudor, D. and Walter, G. A. (2006) “Using an agile approach in a large, traditional organization” AGILE 2006 (AGILE'06), Minneapolis, MN, USA, 2006, pp. 7 pp.-373, doi: 10.1109/AGILE.2006.60.
Available at: https://ieeexplore.ieee.org/abstract/document/1667600

United Nations (n.d.) Kyoto Protocol - Targets for the first commitment period
Available at: https://unfccc.int/process-and-meetings/the-kyoto-protocol/what-is-the-kyoto-protocol/kyoto-protocol-targets-for-the-first-commitment-period
Accessed: 19 April 2023

United Nations (n.d.) Emissions trading
Available at: https://unfccc.int/process/the-kyoto-protocol/mechanisms/emissions-trading
Accessed: 19 April 2023

United Nations (n.d.) Joint Implementation
Available at: https://unfccc.int/process/the-kyoto-protocol/mechanisms/joint-implementation
Accessed: 19 April 2023

United Nations (n.d.) The Clean Development Mechanism
Available at: https://unfccc.int/process-and-meetings/the-kyoto-protocol/mechanisms-under-the-kyoto-protocol/the-clean-development-mechanism
Accessed: 19 April 2023

United Nations (n.d.) What is the Kyoto Protocol?
Available at: https://unfccc.int/kyoto_protocol
Accessed: 19 April 2023

Zhao, N., Sheng, Z. and Yan, H. (2021) “Emission trading innovation mechanism based on Blockchain,” Chinese Journal of Population, Resources and Environment, 19(4), pp. 369–376.
Available at: https://doi.org/10.1016/j.cjpre.2022.01.010

Zhou, Q. and Zhang, Q. (2022) “Simulation research on carbon emissions trading based on blockchain”, Journal of Environmental Engineering and Landscape Management, 30(1), pp. 1-12. doi: 10.3846/jeelm.2022.15107.

10. Appendix

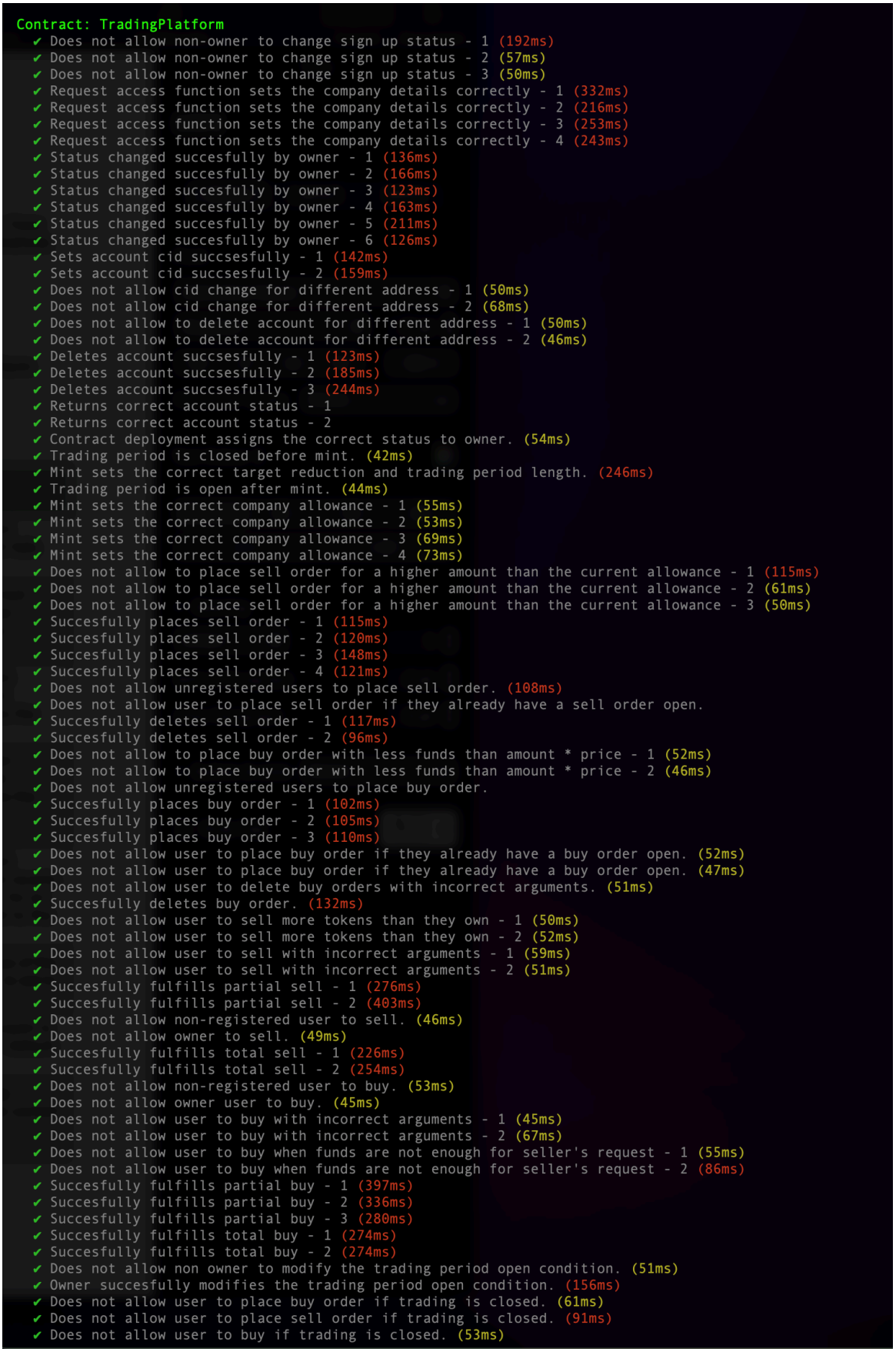
Appendix A. Smart contracts unit testing results table

Test number	Test description	Expected result	Actual result
11	Check if trading allowance was correctly assigned to the target addresses after mint	Each address that is allowed to trade should have a correct trading allowance after the mint() method was called by the owner	PASS - the mint() method assigns the correct allowance to all target addresses
12	Check that unregistered users are not allowed to place sell orders.	The create_sell_order() method should revert when called by an unregistered user (address with status different to 2)	PASS - the create_sell_order() method reverts
13	Check that user cannot place sell order for a token amount higher than their allowance	The create_sell_order() method should revert when the user tries to create an order for an amount of tokens higher than their allowance	PASS - the create_sell_order() method reverts
14	Check that user cannot place a sell order if they already have an open sell order	The create_sell_order() method should revert when the user already has an open sell order	PASS - the create_sell_order() method reverts

Test number	Test description	Expected result	Actual result
15	Check if user can successfully delete their sell order	The delete_sell_order() method sets the address's active sell order to 0 and emits the correct event and parameters	PASS - the delete sell order method emits the correct events and correctly deletes sell order
16	Check if the user can successfully place a sell order	The create_sell_order() should change the open sell order of the address to the correct value and emit the correct sell order event and parameters	PASS - the method sets the correct sell order for the caller's address and emits the correct event
17	Check that unregistered users are not allowed to place buy orders	The create_buy_order() method should revert when called by an unregistered user (address with status different to 2)	PASS - the create_buy_order() method reverts
18	Check that it does not allow user to place buy order with less funds than amount * price	The create_buy_order() method should revert if the user does not send enough funds to match the order parameters when calling the method.	PASS - the create_buy_order() method reverts when not enough WEI is sent
19	Check if the user can successfully place a buy order	The create_buy_order() should change the open buy order of the address to the correct value and emit the correct buy order event and parameters	PASS - the method sets the correct buy order for the caller's address and emits the correct event
20	Check if user can successfully delete their buy order.	The delete_buy_order() method sets the address's active buy order to 0 and emits the correct event and parameters	PASS - the delete buy order method emits the correct events and correctly deletes buy order
21	Check that user cannot place a buy order if they already have an open buy order	The create_buy_order() method should revert when the user already has an open buy order	PASS - the create_buy_order() method reverts
22	Check that a user cannot spot sell more tokens than they own	The sell() method should revert if the amount of tokens to sell argument is higher than the allowance of the caller's address	PASS - the sell() method reverts if the seller tries to sell more tokens than they own
23	Check that a user cannot spot sell with incorrect arguments	The sell() method should revert if it is called by the seller (selling address) with incorrect arguments (amount to buy and price) for the buyer's address given as argument	PASS - the sell() method reverts if the seller inputs incorrect arguments for the buyer's open buy order
24	Check if the smart contract successfully fulfils a partial spot sell	The smart contract should correctly adjust the open buy order for the buyer address with the new parameters, increase the buyer's allowance, transfer the WEI funds to the seller and emit an events with the correct parameters	PASS - the sell() method correctly adjusts the open order, emits the correct event and transfers the WEI and allowance

Test number	Test description	Expected result	Actual result
25	Check if the smart contract successfully fulfils a total spot sell	The smart contract should delete the open buy order for the buyer address, increase the buyer's allowance, transfer the WEI funds to the seller and emit an events with the correct parameters	PASS - the sell() method correctly deletes the open order, emits the correct event and transfers the WEI and allowance
26	Check that a user cannot spot buy with incorrect arguments	The buy() method should revert if it is called by the buyer (buying address) with incorrect arguments (amount on sale and price) for the seller's address given as argument	PASS - the buy() method reverts if the buyer inputs incorrect arguments for the seller's open sell order
27	Check that the smart contract reverts if the buyer does not provide enough WEI funds to fulfil the seller's request	The smart contract should reverts if the WEI funds sent when the buy() method is called are not higher than the buy amount argument times the price set by the seller	PASS - the buy() method reverts if the WEI funds provided are not enough
28	Check if the smart contract successfully fulfils a partial spot buy	The smart contract should correctly adjust the open sell order for the seller address with the new parameters, increase the buyer's allowance, transfer the WEI funds to the seller and emit an events with the correct parameters	PASS - the buy() method correctly adjusts the open order, emits the correct event and transfers the WEI and allowance
29	Check if the smart contract successfully fulfils a total spot buy	The smart contract should delete the open sell order for the seller address, increase the buyer's allowance, transfer the WEI funds to the seller and emit an events with the correct parameters	PASS - the buy() method correctly deletes the open order, emits the correct event and transfers the WEI and allowance
30	Check that the smart contract does not allow any trades to be created when the trading period is open	Any functions related to selling, buying and setting new orders should be reverted if called if the trading period is called	PASS - all buy, sell or order creation functions called when trading is called are reverted.

Appendix B. Smart contracts unit tests



```
Contract: TradingPlatform
  ✔ Does not allow non-owner to change sign up status - 1 (192ms)
  ✔ Does not allow non-owner to change sign up status - 2 (57ms)
  ✔ Does not allow non-owner to change sign up status - 3 (50ms)
  ✔ Request access function sets the company details correctly - 1 (332ms)
  ✔ Request access function sets the company details correctly - 2 (216ms)
  ✔ Request access function sets the company details correctly - 3 (253ms)
  ✔ Request access function sets the company details correctly - 4 (243ms)
  ✔ Status changed succesfully by owner - 1 (136ms)
  ✔ Status changed succesfully by owner - 2 (166ms)
  ✔ Status changed succesfully by owner - 3 (123ms)
  ✔ Status changed succesfully by owner - 4 (163ms)
  ✔ Status changed succesfully by owner - 5 (211ms)
  ✔ Status changed succesfully by owner - 6 (126ms)
  ✔ Sets account cid succsesfully - 1 (142ms)
  ✔ Sets account cid succsesfully - 2 (159ms)
  ✔ Does not allow cid change for different address - 1 (50ms)
  ✔ Does not allow cid change for different address - 2 (68ms)
  ✔ Does not allow to delete account for different address - 1 (50ms)
  ✔ Does not allow to delete account for different address - 2 (46ms)
  ✔ Deletes account succsesfully - 1 (123ms)
  ✔ Deletes account succsesfully - 2 (185ms)
  ✔ Deletes account succsesfully - 3 (244ms)
  ✔ Returns correct account status - 1
  ✔ Returns correct account status - 2
  ✔ Contract deployment assigns the correct status to owner. (54ms)
  ✔ Trading period is closed before mint. (42ms)
  ✔ Mint sets the correct target reduction and trading period length. (246ms)
  ✔ Trading period is open after mint. (44ms)
  ✔ Mint sets the correct company allowance - 1 (55ms)
  ✔ Mint sets the correct company allowance - 2 (53ms)
  ✔ Mint sets the correct company allowance - 3 (69ms)
  ✔ Mint sets the correct company allowance - 4 (73ms)
  ✔ Does not allow to place sell order for a higher amount than the current allowance - 1 (115ms)
  ✔ Does not allow to place sell order for a higher amount than the current allowance - 2 (61ms)
  ✔ Does not allow to place sell order for a higher amount than the current allowance - 3 (50ms)
  ✔ Succesfully places sell order - 1 (115ms)
  ✔ Succesfully places sell order - 2 (120ms)
  ✔ Succesfully places sell order - 3 (148ms)
  ✔ Succesfully places sell order - 4 (121ms)
  ✔ Does not allow unregistered users to place sell order. (108ms)
  ✔ Does not allow user to place sell order if they already have a sell order open.
  ✔ Succesfully deletes sell order - 1 (117ms)
  ✔ Succesfully deletes sell order - 2 (96ms)
  ✔ Does not allow to place buy order with less funds than amount * price - 1 (52ms)
  ✔ Does not allow to place buy order with less funds than amount * price - 2 (46ms)
  ✔ Does not allow unregistered users to place buy order.
  ✔ Succesfully places buy order - 1 (102ms)
  ✔ Succesfully places buy order - 2 (105ms)
  ✔ Succesfully places buy order - 3 (110ms)
  ✔ Does not allow user to place buy order if they already have a buy order open. (52ms)
  ✔ Does not allow user to place buy order if they already have a buy order open. (47ms)
  ✔ Does not allow user to delete buy orders with incorrect arguments. (51ms)
  ✔ Succesfully deletes buy order. (132ms)
  ✔ Does not allow user to sell more tokens than they own - 1 (50ms)
  ✔ Does not allow user to sell more tokens than they own - 2 (52ms)
  ✔ Does not allow user to sell with incorrect arguments - 1 (59ms)
  ✔ Does not allow user to sell with incorrect arguments - 2 (51ms)
  ✔ Succesfully fulfills partial sell - 1 (276ms)
  ✔ Succesfully fulfills partial sell - 2 (403ms)
  ✔ Does not allow non-registered user to sell. (46ms)
  ✔ Does not allow owner to sell. (49ms)
  ✔ Succesfully fulfills total sell - 1 (226ms)
  ✔ Succesfully fulfills total sell - 2 (254ms)
  ✔ Does not allow non-registered user to buy. (53ms)
  ✔ Does not allow owner user to buy. (45ms)
  ✔ Does not allow user to buy with incorrect arguments - 1 (45ms)
  ✔ Does not allow user to buy with incorrect arguments - 2 (67ms)
  ✔ Does not allow user to buy when funds are not enough for seller's request - 1 (55ms)
  ✔ Does not allow user to buy when funds are not enough for seller's request - 2 (86ms)
  ✔ Succesfully fulfills partial buy - 1 (397ms)
  ✔ Succesfully fulfills partial buy - 2 (336ms)
  ✔ Succesfully fulfills partial buy - 3 (280ms)
  ✔ Succesfully fulfills total buy - 1 (274ms)
  ✔ Succesfully fulfills total buy - 2 (274ms)
  ✔ Does not allow non owner to modify the trading period open condition. (51ms)
  ✔ Owner succesfully modifies the trading period open condition. (156ms)
  ✔ Does not allow user to place buy order if trading is closed. (61ms)
  ✔ Does not allow user to place sell order if trading is closed. (91ms)
  ✔ Does not allow user to buy if trading is closed. (53ms)
```

Appendix C. Web app manual testing results table

Test number	Test description	Expected result	Actual result
11	Account page displays the correct company displays	The Account page should display the correct details for the company of the Account that is currently logged int	PASS - the Account page displays the correct content
12	Non-registered users are not allowed to place trades	If the user's status is not approved by the admin, the functionality for placing trades should not be available	PASS - trade functionality unavailable for non-registered users
13	Input sanitation for the Sign Up form	The input fields on the Sign Up form should check and not allow the form to be submitted if any field is empty, or if the data is not of the correct type.	PASS - the Sign Up form does not allow empty or wrong type submissions
14	Sign Up button correctly triggers Metamask wallet	The Sign Up button should call the request_access() smart contract method with the input data as arguments and trigger the Metamask Wallet	PASS - the Sign Up button successfully triggers Metamask
15	Correct Trading Allowance	The web app displays the correct trading allowance for the user on all pages	PASS - the correct trading allowance is displayed
16	Dashboard contains all approved companies	The Active Companies dashboard page should contain all companies that have been approved to trade by the Admin and own a trading allowance	PASS - dashboard displays all companies
17	Correct details for each companies	Each company slot on the Active Companies dashboard page contains the correct details for the respective company	PASS - all companies display the correct details
18	Correct trades for each company	Each company slot on the Active Companies dashboard page contains the correct trades made by the respective company (either buy or sell) in inverse chronological order (most recent first)	PASS - all companies display the correct trades
19	Web app correctly loads data	The 'Loading' frame should be displayed on each page that loads data from the blockchain, and then the correct data should be displayed once loaded	PASS - web app correctly loads data

Test number	Test description	Expected result	Actual result
20	Correct order of orders	The Buy and Sell orders on the Trading Platform are displayed from most expensive to the least expensive.	PASS - orders are displayed in the correct order
21	Correct order contents	The lists of Buy and Sell orders on the Trading Platform match the orders on Firebase Firestore	PASS - the correct orders are displayed
22	Orders are successfully updated	Any change made to the Firebase Firestore collection of orders is also instantly visible on the Trading Platform lists of orders	PASS - orders are successfully updated
23	Company status update	The status of every company should be update in real time on the Web App whenever it is update on Firestore	PASS - company status is updated from Firestore
24	Future order open by default	The default state for the order type selection button is Future	PASS - Future order selected by default
25	Order type switch	The switch to change the type of order between Future and Spot changes the input fields and Submit buttons from Future to Spot and back, when pressed	PASS - the order type switch correctly changes the Trading Platform contents when pressed
26	Correct average price for orders by type	The correct average price of all Buy and Sell orders is displayed when the switch is set to Spot order	PASS - the correct average price is displayed
27	Open orders for current account are displayed	If the current account has any open future Buy or Sell order, it is displayed under the Trading tool on the Trading Platform	PASS - the platform correctly displays the current account's open orders
28	Close open order functionality	Pressing the close order button for the open buy or sell order of the current account should call the delete_buy_order() or delete_sell_order() method and trigger the Metamask wallet	PASS - the close order buttons call the correct functions and trigger the Metamask wallet with a transaction
29	Correct transactions list	The list at the bottom of the Trading Platform page should display the latest 10 transactions made on the app	PASS - the correct transaction list is displayed
30	Empty input submit	Submitting an order with an empty input field should trigger an error and display an alert on screen	PASS - submitting any order with an empty field displays an alert

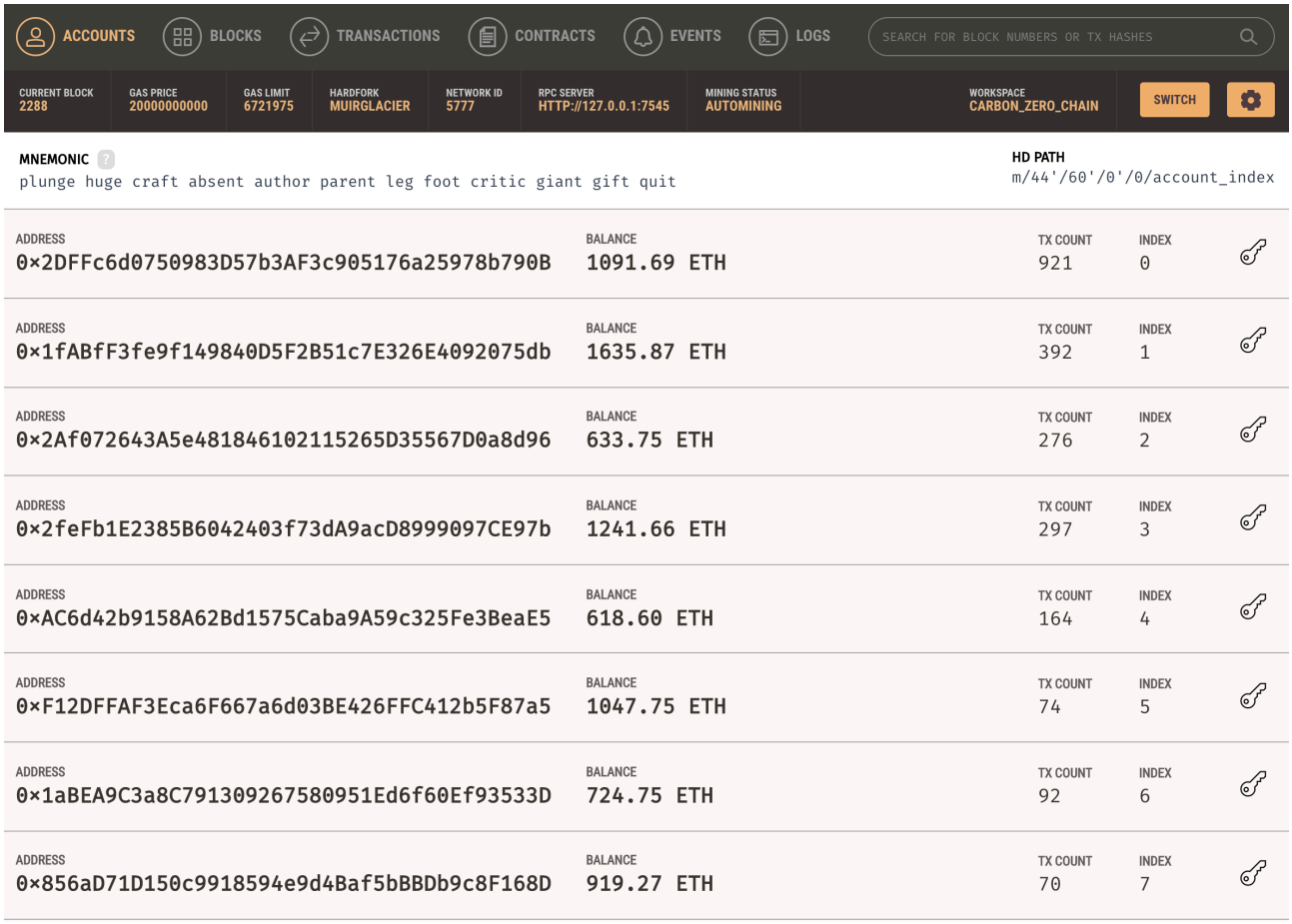
Test number	Test description	Expected result	Actual result
31	Wrong data type submit	Submitting any order with data in the input fields with a different type than numerical should trigger an error and display an alert on screen	PASS - submitting any order with non-numerical data displays an alert
32	Submitting a future buy order triggers the wallet and calls the smart contract method	Submitting a future buy order should trigger a transaction on the Metamask wallet and call the smart contract create_buy_order() method with the correct arguments, taken from the Quantity and Price input fields. Metamask call should fail if the arguments are invalid (functionality triggered by the smart contract method)	PASS - submitting the buy order triggers Metamask and calls the correct method. Invalid arguments trigger an error
33	Submitting a future sell order triggers the wallet and calls the smart contract method	Submitting a future sell order should trigger a transaction on the Metamask wallet and call the smart contract create_sell_order() method with the correct arguments, taken from the Quantity and Price input fields. Metamask ca;; should fail if the arguments are invalid (functionality triggered by the smart contract method)	PASS - submitting the sell order triggers Metamask and calls the correct method. Invalid arguments trigger an error
34	Spot buy does not match with the sell order made by the current account	Spot buy order is not matched to the open sell order that belongs to the current account, but is matched to and fulfilled with orders opened by different accounts	PASS - spot buy is not matched to a sell order belonging to the current account
35	Spot sell does not match with the buy order made by the current account	Spot sell order is not matched to the open buy order that belongs to the current account, but is matched to and fulfilled with orders opened by different accounts	PASS - spot sell is not matched to a buy order belonging to the current account
36	Spot buy is matched to the sell order with the lowest asking price	A spot buy order should be matched to and fulfilled from the sell order in the order book with the lowest asking price. A Metamask transaction should be triggered and the buy() method should be called with the correct arguments	PASS - buy operation is matched to the order with the lowest price in the sellers order book, and the correct method is called

Test number	Test description	Expected result	Actual result
37	Spot sell is matched to the buy order with the highest asking price	A spot sell order should be matched to and fulfilled from the buy order in the order book with the highest asking price. A Metamask transaction should be triggered and the sell() method should be called with the correct arguments	PASS - sell operation is matched to the order with the highest price in the buyers order book, and the correct method is called
38	Placing a spot buy order with a quantity that cannot be fulfilled from a single future sell order	Placing a spot buy order with a quantity that is higher than the first sell order in the order book, should trigger multiple transactions, until the entire order is fulfilled. The buy() method should be called sequentially for each available future sell order in the order book, until the spot buy quantity is fulfilled, or until an error. The Metamask walled should be triggered for each transaction. In case the entire quantity has been fulfilled, a success message should be displayed. If an error happens, an error message should be displayed and the execution should stop	PASS - the spot buy quantity is fulfilled sequentially from future sell orders available in the order book. The Metamask wallet is triggered successfully for each transaction. The execution runs until the entire order is fulfilled or until error. Success or error message is displayed
39	Placing a spot sell order with a quantity that cannot be fulfilled from a single future buy order	Placing a spot sell order with a quantity that is higher than the first buy order in the order book, should trigger multiple transactions, until the entire order is fulfilled. The sell() method should be called sequentially for each available future buy order in the order book, until the spot sell quantity is fulfilled, or until an error. The Metamask walled should be triggered for each transaction. In case the entire quantity has been fulfilled, a success message should be displayed. If an error happens, an error message should be displayed and the execution should stop	PASS - the spot sell quantity is fulfilled sequentially from future buy orders available in the order book. The Metamask wallet is triggered successfully for each transaction. The execution runs until the entire order is fulfilled or until error. Success or error message is displayed
40	Placing a spot buy order for a quantity higher than what is available in total on the order book	Placing a spot buy order for a quantity higher than what is available in total on the future sell order book should fill part of the order sequentially, until everything that was available was bought and end in an error	PASS - buying more than what is available should end in an error

Test number	Test description	Expected result	Actual result
41	Placing a spot sell order for a quantity higher than what is available in total on the order book	Placing a spot sell order for a quantity higher than what is available in total on the future buy order book should fill part of the order sequentially, until everything that was on ask was fulfilled and end in an error	PASS - selling more than there is ask for should end in an error
42	Empty input fields for admin token mint	Clicking ‘mint’ if any of the input fields for the minting details is empty should trigger an error and display an error message on the screen	PASS - minting with empty input arguments triggers error
43	Invalid type data in input fields for admin token mint	Clicking ‘mint’ if any of the input fields for the minting details has invalid data (different type from numerical) should trigger an error and display an error message on the screen	PASS - minting with invalid data type input arguments triggers an error
44	Minting tokens	Pressing the mint button with valid form input arguments for the trading period details triggers the Metamask wallet and calls the mint() method with the correct arguments	PASS - minting successfully triggers the Metamask wallet and calls the correct smart contract method with the given arguments
45	Correct Admin page contents 1	The mint parameters form is displayed on the Admin dashboard if the trading period is not live	PASS - mint form is displayed if the trading period is not live
46	Correct Admin page contents 2	The correct trading period details (reduction, length and starting time) and a button to end the period are displayed if the trading period is live	PASS - correct trading details displayed if the trading period is live
47	Ending the trading period	Pressing the ‘End period’ button should trigger a Metamask transaction and call the set_trading_open(false) smart contract method	PASS - pressing the button triggers the wallet and calls the correct method
48	Pending accounts list	The details of each account that is pending approval from the admin to join the trading scheme should be displayed on the Admin Dashboard page	PASS - the correct details are displayed for each address
49	Updating the pending accounts list	The pending account list should be automatically updated whenever the status of an account is changed	PASS - the list is successfully updated

Test number	Test description	Expected result	Actual result
50	Approving account	Pressing the ‘Approve account’ button for a pending account should trigger a Metamask transaction, call the set_account_status() smart contract method with the correct arguments to set the address’s status to 2 and update the pending accounts list	PASS - pressing the approval button for any address by the admin triggers the wallet, calls the correct method and updates the list
51	Declining join request	Pressing the ‘Decline request’ button for a pending account should trigger a Metamask transaction, call the set_account_status() smart contract method with the correct arguments to set the address’s status to 0 and update the pending accounts list	PASS - pressing the decline button for any address by the admin triggers the wallet, calls the correct method and updates the list
52	Empty pending account list	If the list of pending accounts is empty, the Admin dashboard should display a message saying ‘No pending accounts.’	PASS - the correct message is displayed if the pending accounts list is empty

Appendix D. Screenshot of Ganache Blockchain



Appendix E. User Testing Questionnaire

To test the application, several tasks were given to users. There were 2 set of tasks (one for admin user testing and one for company user testing):

Tasks for company users:
-write the **allowance** for the company called NovoTech and their latest transaction
-sign up as a company (input test details and 1000 base-year emissions)
-submit a future sell order of 100 CX at 2 ETH
-submit a spot buy order of 15 CX
-cancel your future sell order
-write your company's current **allowance**
-write your company's **activity domain**
-log out

Tasks for admin users:
-write the **allowance** for the company called NovoTech and their latest transaction
-sign up as the admin
-**decline** the sign up request of a company
-**approve** the sign up request of a company
-set the parameters for a new **trading period** and start it
-end the current **trading period**
-log out

After completing the tasks, a Questionnaire was provided. The first part contained questions about the user's background, the second part contained questions about Nielsen's Usability Heuristics, and the final part contained an open section for feedback.

The questionnaire was completed by a total of **10 users**.

Questions about the user's background

Question	Possible answers	Received answers (no of users)
Do you have any experience in software development?	A. No B. 0-1 yrs C. 1-3 yrs D. 3+ yrs	A: 2 B: 5 C: 2 D: 1
Have you ever heard about carbon emission trading before?	A. Yes B. No	A: 6 B: 4
Have you ever done any transactions on the Blockchain before?	A. Yes B. No	A. 7 B. 3
If answer to previous question was yes, what Blockchains have you used?	Open ended	Bitcoin: 7 Ethereum: 7 Polygon: 3 Solana: 3

Questions about Nielsen's Usability Heuristics

Usability heuristic	Question	Possible answers	Received answers (no of users)
Aesthetic and minimalist design	The application's design and layout are aesthetically pleasing.	A. Strongly B. Disagree C. Neutral D. Agree E. Strongly Agree	A: 0 B: 0 C: 1 D: 2 E: 7
Aesthetic and minimalist design	The amount of information on the screen at any point was not overwhelming.	A. Strongly B. Disagree C. Neutral D. Agree E. Strongly Agree	A: 0 B: 0 C: 1 D: 4 E: 5
Consistency and standards	The design and layout of the web app felt consistent throughout use.	A. Strongly B. Disagree C. Neutral D. Agree E. Strongly Agree	A: 0 B: 0 C: 0 D: 2 E: 8
Recognition rather than recall	It was easy to find what I was looking for in the app.	A. Strongly B. Disagree C. Neutral D. Agree E. Strongly Agree	A: 0 B: 1 C: 1 D: 4 E: 4

Usability heuristic	Question	Possible answers	Received answers (no of users)
Recognition rather than recall	Placing an order is intuitive.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 3 E: 7
Recognition rather than recall	It was easy to understand the functionality of each button and element before using them.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 2 E: 8
Error prevention	The app prevented me from placing an order with invalid arguments.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 0 E: 10
Help users recognise, diagnose and recover from errors	The app provided enough guidance to recover from errors.	A. Strongly Disagree … E. Strongly Agree	N/A (no errors encountered during testing)
User control and freedom	Navigating the app felt seamless and intuitive.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 2 E: 8
Flexibility and efficiency of use	The app is performant, with minimum waiting time.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 4 E: 6
Visibility of system status	The app consistently kept me informed of its status at any point in time (loading, loaded etc).	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 1 D: 3 E: 6
Match between system and the real world	I could easily understand the purpose of the web application.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 0 E: 10
Match between system and the real world	The terminology and language used throughout the web app make sense to me in the context of carbon emission trading.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 1 D: 0 E: 9
Help and documentation	The guide and the documentation on the web app are clear and easy to understand.	A. Strongly Disagree … E. Strongly Agree	A: 0 B: 0 C: 0 D: 1 E: 9

Open feedback

Question	Answers
Do you have any suggestions and feedback to make the app more usable and intuitive?	-make order selection switch rounded -emphasise wether it's future or spot now -rename 'Place buy order' to 'Future / Spot order' -add 'CX' input placeholder -specify the order of transactions in dashboard -rename 'Dashboard Page' to something that is more telling about the contend (e.g 'Active Companies') -rename 'Profile' page to 'Account' page