A Food Cooperative on Blockchain

1043531

University of Oxford

Final Honour Schools of Computer Science Part B

Trinity Term 2023

Word count: 4888

Acknowledgements

Many thanks to my supervisor Alastair Janse van Rensburg for guiding me through the project and helping me with technical questions in our weekly meetings. I also want to thank my tutor Thomas Melham for co-supervising and providing invaluable advice throughout the project.

Abstract

Blockchain technology holds the potential to transform the cooperative movement. Food cooperatives face significant challenges in the transparency and security of their economic activities and this emerging technology provides a solution, bringing these organisations in line with their founding values.

In this project, we draw upon the founding principles formulated by the International Co-operative Alliance (2015) to construct an effective blockchain system for such activities. While previous work has explored the use of blockchain in other areas of the cooperative economy and in non-cooperative marketplaces, limited research has been done on its direct application in food cooperatives.

This project contributes to the growing research into the intersection of blockchain with the cooperative economy by highlighting its potential for real-world usage.

We present the design and analysis of an original smart contract system for bulk wholesale ordering, as well as guidance for integration with existing cooperatives. Finally, we conclude the project with a presentation of our affordable calculated gas costs.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to the problem

With severe food insecurity affecting 11.7% of the population, and food resources shrinking globally, the impact of rising food costs on communities will only grow. Last year, 112 million more people were unable to afford a healthy diet, bringing the total number of affected individuals to 3.1 billion (UNICEF et al. 2022). Addressing this issue is critical, particularly in the worst affected low-income countries. A global solution will require wide-ranging reforms and will be enabled by the use of novel technologies (Mc Carthy et al. 2018). Food cooperatives are a community-level solution to the affordability and accessibility of food produce. Bulk group orders placed directly with farmers provide food at cheaper rates than traditional supermarkets (Deller et al. 2009). The governance of cooperatives is typically organized through the election of governing bodies, voted for using the 'one-member one-vote' principle (Berge, Calwell, and Mont 2016). Despite this, the existence of a restricted committee inevitably takes the activities of the cooperative out of the direct control of members. Various solutions, such as a rotating committee membership (ibid.), have been proposed to address the exclusivity of the governing bodies. In this project, we propose a novel blockchain system to address the economic activities of these organizations.

1.2 Cooperatives

Cooperatives are jointly owned and run enter'prises, proving a space for members to work towards common economic, social, and cultural needs. A food cooperative is a type of consumer cooperative, in which membership is defined by economic participation in food purchasing.

Illustrated below is the ordering and distribution process for a typical food cooperative (Deller et al. 2009).

1. Members submit their individual orders for a particular food product or group of food products.
2. These requests are combined to form a singular bulk wholesale order, which is sent to the appropriate provider.
3. Completed orders are delivered to a central distribution point where members collect their purchases.

There is no universally accepted definition of a cooperative due to the wide range of functionalities such organisations fulfil. The International Co-operative Alliance (2015) have derived seven generally accepted (Deller et al. 2009) principles from the from the founding of the cooperative movement (Fairbairn 1994).

We define a successful project as one that implements blockchain technologies to align food cooperatives more closely with these principles (International Co-operative Alliance 2015). A system that enables the activities of these organizations to better follow these principles will be more effective in facilitating cooperative wholesale purchasing.

In the sections below, we analyse the guidance (ibid.) in the context of food cooperatives.

(P1) Voluntary and Open Membership

Members must have made an active and free decision to participate in activities. Transparency is crucial for individuals to make an informed choice about joining the organisation. To ensure open membership without discrimination, economic participation in produce ordering should be the primary constriction for membership.

As cooperatives grow in size, transparency becomes more challenging due to logistical issues with a larger membership.

(P2) Democratic Member Control

Members should control the cooperative's activities, in particular the choice of produce available and the formation of bulk orders. Ensuring true democratic control is increasingly difficult as membership grows because an effective democracy must be open, transparent, and accountable. This is challenging in a larger cooperative's commercial activities where some level of privacy is required. For example, bank account login details should not be available to every member, especially if we have fulfilled P1.

(P3) Member Economic Participation

The purpose of food cooperatives is not to create profit for individual members. If there is any surplus value, it should be redirected to development and any money invested should be used to fund the activities. Because of this unusual economic activity, cooperatives can face regulatory and tax challenges.

(P4) Autonomy and Independence

Food cooperatives should operate autonomously and avoid any external influence that undermines the members' democratic control. This can become a risk if a food cooperative is reliant on services from a small number of suppliers. These third parties could influence activities by limiting the available produce.

(P5) Education, Training, and Information

Education should be provided to cooperative members, especially about the benefits cooperatives offer. In particular, educating young people often leads to new innovations and ensures the cooperative sustainability for the next generation. It is essential to educate and train members on the use of any new technologies implemented.

(P6) Cooperation among Cooperatives

Members should strive to benefit their own organisation and the entire movement. By working together, organisations can share innovations, costs, and resources. When utilizing novel technologies, systems should be designed in a way that encourages collaboration and partnership.

Cooperation among cooperatives is often hindered by the difficulty of communicating and sharing power between governing bodies.

(P7) Concern for Community

Cooperatives should strive to improve the local and membership communities. We can break these considerations down into prioritizing the local environment, enabling social progress and ensuring economic stability.

Food cooperatives, by definition, are for the purchasing of local and organic foods (Little, Maye, and Ilbery 2010). When making financial decisions, members must carefully consider the economic and social needs of the entire community.

1.3 Requirements

We define a successful project as one that implements blockchain technologies to align food cooperatives more closely with the seven guiding principles (International Co-operative Alliance 2015). A system that enables the activities of these organizations to better follow these principles will be more effective in facilitating wholesale purchasing by the food cooperative.

1.4 My contribution

Blockchain offers major benefits to food cooperatives but further research is required to see if this can be successfully utilized (Rocas-Royo 2021).

This project is an original design and prototype implementation of a wholesale order system on blockchain, presenting a practical use case in food cooperatives. This enables members to collectively order produce in bulk from farmers, streamlining the purchasing process. Through our work, we contribute to the ongoing research into the use of blockchain in the food supply chain and we highlight the potential for this technology to address pressing global issues.

Chapter 2

Background

2.1 Blockchain

The blockchain is a decentralized, distributed ledger that provides a secure and tamper-proof record of digital transactions. The network uses cryptographic proofs and consensus mechanisms to agree on the state of the blockchain, replacing the need for a trusted centralized authority. Originally designed for Bitcoin payments (Nakamoto 2009), the blockchain has expanded to provide secure and transparent records for other digital transactions. A notable extension is Ethereum, a decentralized platform with an embedded Turing complete programming language (Buterin 2013).

As of September 2022, Ethereum uses the proof-of-stake (PoS) consensus mechanism to maintain a consistent blockchain state among nodes in the network (Foundation 2021). Validator nodes execute and validate transactions, combining these into blocks that are broadcast to the network. Validators are randomly selected based on their staked Ether (ETH), the native currency of Ethereum. As the block propagates through the network, other validator nodes verify the included transactions, ensuring a consistent transaction history.

2.2 Smart contracts

Programs that run on the Ethereum blockchain are called smart contracts (Buterin 2013). Smart contracts are computerized transaction protocols that execute the contract terms (Szabo 1996). On Ethereum they take the form of immutable blocks of code, executing specific actions when function preconditions are met. Smart contracts exhibit deterministic behaviour and can be called by any user of the network, meaning code behaviour is expected, reproducible and verifiable.

Smart contracts, like traditional programs, require computational resources to run. The Ethereum protocol charges gas fees which sent to the block validator to execute the code for each transaction. To encourage inclusion into blocks, it is possible to pay additional gas fees.

2.3 Solidity

Solidity is a popular object-oriented programming language for smart contracts on Ethereum (Parizi, Amritraj, and Dehghantanha 2018). Contract inheritance enables libraries such as OpenZeppelin (2023) to provide secure, audited and inheritable smart contracts.

Solidity contracts have three key language features: functions, variables, and events. Variables are used for on-chain data storage, whereas functions execute logic and enable state changes. Events signal these changes to off-chain applications and are stored in blocks with their originating transactions.

Mappings and arrays are used for on-chain lists. Mappings are more efficient and less expensive to operate on but cannot be iterated through. Array iteration must be handled with caution, as exceeding the gas limit can result in a Denial of Service attack (Modi 2019).

2.4 Oracles

Oracles are third parties that provide off-chain data onto a blockchain system. (Al-Breiki et al. 2020). There are a wide variety of oracle providers available such as Chainlink, which provides a time-stamp oracle (Kaleem and Shi 2021). The 'Oracle problem' of obtaining external data without losing the benefits of decentralization is a widely discussed issue (Caldarelli 2020).

2.5 Security

The immutability of smart contracts pose unique security challenges since they cannot be modified once deployed. Smart contract vulnerabilities resulted in an estimated $1.57 billion lost up until May 2022 (Zhang et al. 2023).

2.5.1 Denial of Services

Denial of Services (DoS) attacks aim to make network resources unavailable. In smart contracts this can be achieved using block gas limit exceptions or an unexpected function failure. If the block gas limit set by validators (Ethereum 2023) is exceeded, transactions will be blocked. This could compromise the trust of a system, especially in cases where consumers are expecting refunds (Consensys 2023).

2.5.2 Re-entrancy

The DAO Hack is an example of a re-entrancy attack. This attack resulted in the theft of more than US$50 million worth of Ether (Price 2016). Re-entrant procedures can be interrupted in the middle, initiated over, and both runs can complete without any errors. In combination with recursive calls, the DAO attacker drained the entire balance from the vulnerable smart contract (Sayeed, Marco-Gisbert, and Caira 2020).

2.5.3 Transaction front-running

Network integrity relies on proper transaction ordering. Malicious validators can exploit their position by changing the order of transactions from a single contract (Luu et al. 2016). This is a serious concern in a first past the post reward system. An analysis of 200,000 front-running attacks uncovered a total profit of US$18.41 million (Torres, Camino, and State 2021).

2.5.4 Timestamp dependency

Timestamp dependency is when smart contract conditions are structured on the block timestamp. This usually applies to critical checks that result in a transfer of ether (Dika and Nowostawski 2018a). With a proof-of-work (PoW) consensus, miners could manipulate block timestamps by small amounts (Dika and Nowostawski 2018b). However, the Ethereum Merge minimizes this issue, as slots are filled every 12 seconds (Foundation 2021).

2.5.5 The Sybil attack

A Sybil attack is where an entity assumes a subset of identities in order to manipulate the system (Douceur 2002). Decentralized systems are particularly vulnerable due to the absence of a trusted central authority to verify users identities. It is apparent the danger of such an attack in a decentralised voting system.

2.5.6 Other security concerns

Smart contracts cannot prevent themselves receiving Ether from blockchain account so cannot rely on the zero balance assumption (SWC-Registry n.d.). Additionally, external calls to unknown smart contracts carry the risk of executing malicious code (Dika and Nowostawski 2018a).

Chapter 3

System Design

The project consists of three contracts: the market, produce, and group order. While combining these contracts would have reduced the risks of external calls, it would have also increased complexity and the risk of code vulnerabilities. In order to also promote extensibility (P6), our system is designed to be modular.

The market contract is used to record available produce and allow produce to be listed. When a produce is uploaded to the market, a new contract is deployed with identifying information for the off-chain produce.

In parallel, consumers can deploy group order contracts that combine orders and submit if an order threshold is met. Individual consumers contribute funds that are later transferred to the farmer or refunded. Each group order is associated with a single produce contract upon creation.

This framework allows any user (P1) to participate if they meet the function prerequisites. By automating this process, problems with increased membership can be avoided if the smart contracts scale effectively. Members can agree upon and enforce rules for their activities by expressing them as executable code, enabling democratic decision-making and decentralized governance (P2). Members are actively participating in economic activities (P3) by tracking inventory and making payments . A decentralized platform ensures that the cooperative is not reliant on a central authority (P4). Through using blockchain, we prioritize enhanced security and transparency to better serve the members of the cooperative community (P7). By minimizing gas costs, we ensure that produce is affordable and that the platform is accessible to more members (P1, P7).

3.1 Using the system

Figure 3.1 depicts when consumers orders have been accepted. Figure 3.2 demonstrates the case where the farmer rejects the order after it has been submitted. Figure 3.3 and Figure 3.4 showcase the situations that lead to an internal cancellation. Figure 3.3 shows when consumers decide to cancel their orders after submission. In this scenario all participating members should have equal voting power (P2). Figure 3.4 depicts when a group order fails to gather enough members within the time limit. This should be proposed on deploying the group order contract to obtain participant consent (P1, P2)

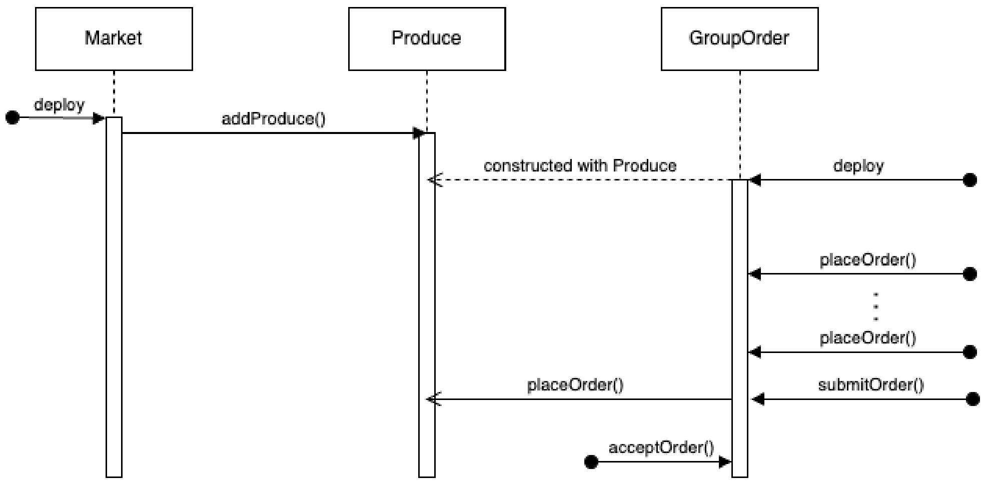


Figure 3.1: Successful group order

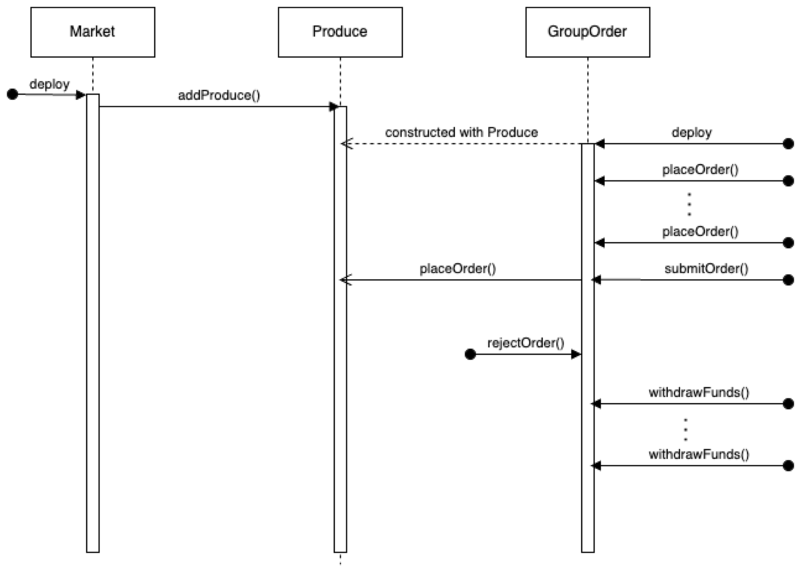


Figure 3.2: Unsuccessful group order: order rejected by farmer

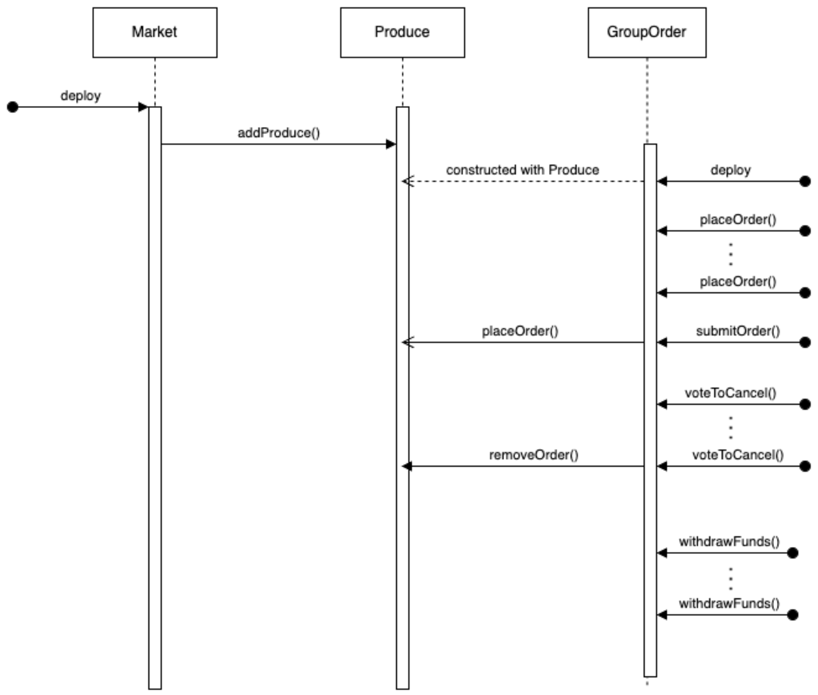


Figure 3.3: Unsuccessful group order: voted to cancel

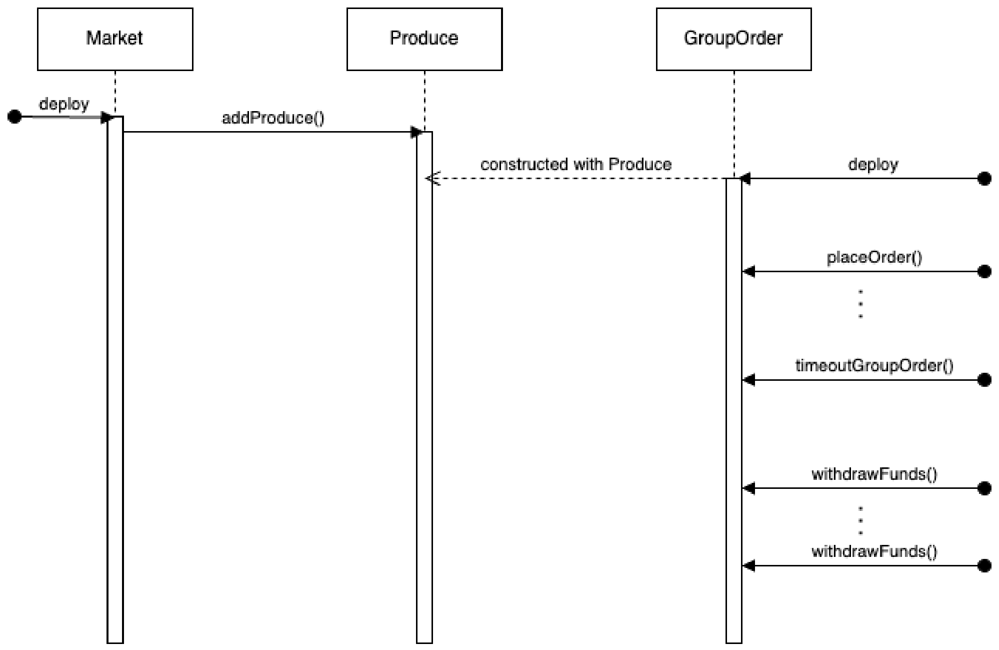


Figure 3.4: Unsuccessful group order: timed out

Chapter 4

Implementation Choices and Analysis

4.1 Programming languages

We chose Solidity for our smart contracts as it is popular and familiar to developers, due to a JavaScript-style syntax. While alternatives like Vyper exist, Solidity's features, such as class inheritance, make it the most suitable option for our project (Vyper Development Team 2023).

Using a well-documented programming language and the largest blockchain platform (CoinMarketCap accessed on April 18, 2023) with security-audited libraries will reduce our risk of security vulnerabilities (P7)

4.2 Libraries and Frameworks

We used two external tools for this project: Remix IDE and the OpenZeppelin library. Remix IDE allows us to test and run Solidity, and has a user interface for calling functions (see Appendix B). OpenZeppelin provides security audited contracts from which we inherit ownership and cloning functionality (OpenZeppelin 2023).

4.3 Smart Contracts

4.3.1 Market

The market provides a record of available produce contract addresses and allows farmers to list products, through the deployment of produce contracts.

We must be careful to prevent malicious actors implementing different methods under the same class name. To mitigate this we use the factory pattern (4.4.1), the check-effects-interaction pattern (4.4.2) and place external calls within a try-catch block. To address the high gas fees from creating new contract instances, we use smart contract cloning (4.4.3).

4.3.2 Produce

This contract is deployed by farmers via the market. It stores produce information to help consumers make informed decisions and records pending orders.

To ensure scalability and extensibility (P1, P6), the functionality to close the produce is directly contained within the contract. Only the owner (4.4.5), in this case the farmer, can close the contract to prevent malicious actors from closing all contracts. The state machine pattern (4.4.6) is used to control the behavior and prevent orders from being placed on closed contracts. Although we could achieve this through other means, such as contract self-destruction, it is important not to undermine the transparency of previous orders.

4.3.3 Group order

This contract enables bulk orders, ensuring a verifiable record of each participant's order, using the state machine pattern (4.4.6) to control behaviour.

Members should not be able to join an order unless they are economically participating (P3) and farmers should not be transferred funds until they have agreed to fulfil the order (P2). A smart contract acting as an escrow (4.4.7) will fulfil these requirements.

An order will be rejected if the farmer rejects it, the minimum order size is not met, or if members decide to cancel the pending order. Refunds must be handled carefully using the withdrawal pattern (4.4.9) to avoid a denial-of-service (DoS) attack.

When the minimum order is not met, consumers who have already sent their funds cannot retrieve their money and the farmer cannot reject the order. We will address this using a time limit (4.4.8) for incomplete orders. A voting system could be used here, but with a small initial membership, this could easily be hijacked by a Sybil attack (2.5.5).

To allow economic participants to reject a pending order and ensure P2, we implement a one-member-one-vote system for economic participants(P3). This is a resource demanding task that prevents Sybil attacks. There is a risk of users dropping offline and not returning but we assume consumers have sufficient economic incentive in recovering pledged money. Furthermore a voter threshold means we do not require complete participation. This should be set sufficiently high to reduce the risk of a small subset of users controlling the system.

4.4 Security

4.4.1 Factory patter

The factory pattern provides control over the class of object that is created. An on-chain contract creates contract instances with uniform functionality from a template. This prevents malicious actors from implementing different methods under the same class method names.

Members trust the cooperative to distribute the food so we argue no further trust is placed in the organisation by allowing it to communicate the verified markets and farmers. This should be democratized among all members in existing cooperative meetings (P2).

It is not scalable to use this verification method on produce contracts. As membership grows, the number of such contracts will increase significantly hence sufficient participation in voting will become infeasible.

4.4.2 Check-effects-interaction pattern

The check-effects-interaction pattern is used to prevent re-entrancy attacks (Wohrer and Zdun 2018). Upon an external smart contract call, there is a risk of the called contract hijacking the code or control flow. To prevent this, it is important to check function pre-conditions first, then make changes to the internal state, and finally implement external calls.

```
function closeProduce() external onlyOwner {
        require(state == State.OPEN, "only open produce can be closed");
        state = State.CLOSED;
        Market(market).removeProduce();
}
```

4.4.3 Cloning

To obtain lower gas costs for the repeated deployment of produce contracts through the market we use the OpenZeppelin Cloning library. The function logic is shared between cloned contracts but the data is stored independently for each instance.

4.4.4 Data storage

Only essential data should be stored on chain since storage is financially and computationally expensive (Leeming, Cunningham, and Ainsworth 2019).

To ensure transparency (P1, P2) we must verify identifying produce data on-chain. This means we cannot reduce the volume of verifiable information to reduce gas costs. Data involved in the system logic should also be stored on chain.

The identifying details are lengthy because they define what contract interactions imply off-chain. To address high gas fees a cryptographic hash of the data can be used (Hepp et al. 2018). A cryptographic hash function is a one-way function that condenses data into a fixed-size block. As long as a pre-image and collision resistant function is chosen, members can verify the data integrity and authenticity by computing the hash and comparing values.

To promote P6 and P4, secure hash functions are chosen independently of the system and only the hash values are stored on-chain.

An array could be used to store the lists of produce and group orders but an unbounded array could result in a DoS attack. We can prevent this by limiting pending orders but a better solution would be to use mappings and events. We suggest an off-chain system to monitor events and a mapping to cheaply verify the off-chain list. This fulfils transparency and affordability requirements (P1, P2, P7).

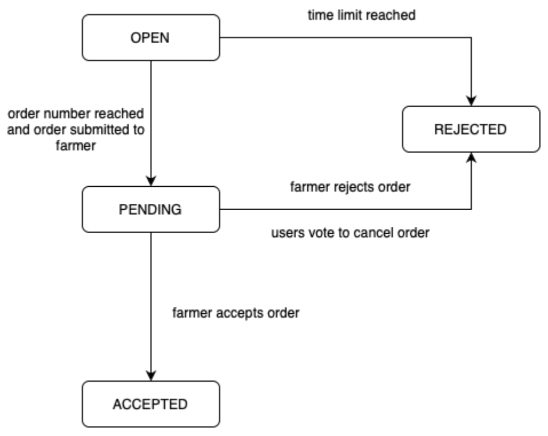
Below is an example of our use of events for placing orders in the produce contract.

```
function placeOrder() external {
    require(state == State.OPEN, "produce not open for ordering");
    require(
        msg.sender != owner(),
        "farmers cannot place orders for their own produce"
    );
    require(
        !orderList[msg.sender],
        "this order has already been placed"
    );
    orderList[msg.sender] = true;
    emit OrderQueued(msg.sender);
}
```

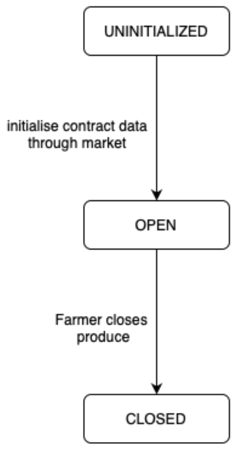
4.4.5 Ownership

We use the Ownable contract from OpenZeppelin (2023) for access control.

The farmer must be the only user able to take orders from and close the produce contract. This stops falsely accepted orders or available produce contracts being closed by malicious actors (P7).



(a) Group order contract



(b) Produce contract

Figure 4.1: State transition diagrams

To avoid unnecessary centralization of the system, we place full responsibility of the produce contract with the farmer (P4, P6) Additionally, this aligns with the real-life ownership of product increasing intuition for users (P5).

4.4.6 State Machine

The operation of group orders and produce contracts can vary depending on their state (Figure 4.1). Since both contracts have unique states with transitions between them, a state machine pattern is an appropriate choice (Wöhrer and Zdun 2018). An alternative, such as using multiple conditional statements, would resulted in complex and less understandable code that is more prone to vulnerabilities, undermining P5 and P7.

To understand the importance of different states, consider the case where a farmer has run out of stock. Without enabling a change in behavior, group orders can be placed indefinitely and without the closing of the produce contract, members placing orders have to wait for a rejection vote to be refunded. This inefficiency is not aligned with P7.

4.4.7 Escrow

The group order contract is an escrow between the farmer and group order members. An escrow is a third party that holds funds until requirements have been met. In typical cooperatives, this role is performed by the governing body but having an exclusive group with access to the funds contradicts P2.

In our system, users pledge funds for their orders (P3), increasing the agreed-upon state variable. Once the required size is met any user can submit the order (P2).

If the farmer accepts an order, the payment is transferred to their account, representing an off-chain agreement between the cooperative and farmer. To avoid relying on the Zero Balance assumption (2.5.6), only the agreed-upon amount is transferred. If the farmer rejects the group order a transition to the rejected state occurs.

Although the produce contract could have been chosen as the escrow, it would require extra mapping and a transfer back of funds if the order is rejected. More complicated logic incurs higher gas fees and we also aim to reduce the dependency between contracts (P6).

4.4.8 Timeout

We can access time on-chain through the block timestamp or a trusted off-chain oracle service. Chainlink provides a source of current time (Kaleem and Shi 2021) but our contracts should not rely on third party organisations (P4).

Using block timestamp means there is scope for manipulation by validators. Although this is improved upon by the transition to proof-of-stake (PoS) from proof-of-work (PoW), we will assume PoW in the worst case.

There is varying guidance as to the precision of block timestamps. Smart contracts that maintain integrity within 30 seconds are considered safe but no precision of greater than 15 minutes is guaranteed (Dika and Nowostawski 2018b).

For complete safety, our order time-out will be a minimum of 1500 minutes. This means timestamp manipulation will only affect consumers who join the order in the final 1% of order time. It is reasonable to assume that this will not be the majority of consumers therefore attacks will be mostly redundant. 1500 minutes is a sensible minimum time limit for a food ordering system.

This vulnerability is dangerous when the transfer of Ether is dependent on the timestamp condition (Ma et al. 2021) so no transfers occur in timestamp dependent functions.

4.4.9 Withdrawal pattern

The withdrawal pattern is used to mitigate DoS attacks on group order refunds. The pattern separates out the logic of transferring Ether and prevents gas limit exceptions.

Rather than refunding all users at once, users must call the withdrawFunds() function, as demonstrated below.

```
function withdrawFunds() external {
    require(state == State.REJECTED, "order has not been rejected");
    require(
        orders[msg.sender],
        "customer has no outstanding balance to refund"
    );
    orders[msg.sender] = false;
    payable(msg.sender).transfer(produce.price());
}
```

4.4.10 Front running

When demand for a product is high and supply is low, front running can be used to gain queue priority, preventing other members from joining the group order. To minimize this we restrict each consumer to a single portion. Even if an unfair consumer can always secure a place in a high-demand group order, the majority of places are still available.

Chapter 5

Interacting with the system

5.1 Offchain interactions

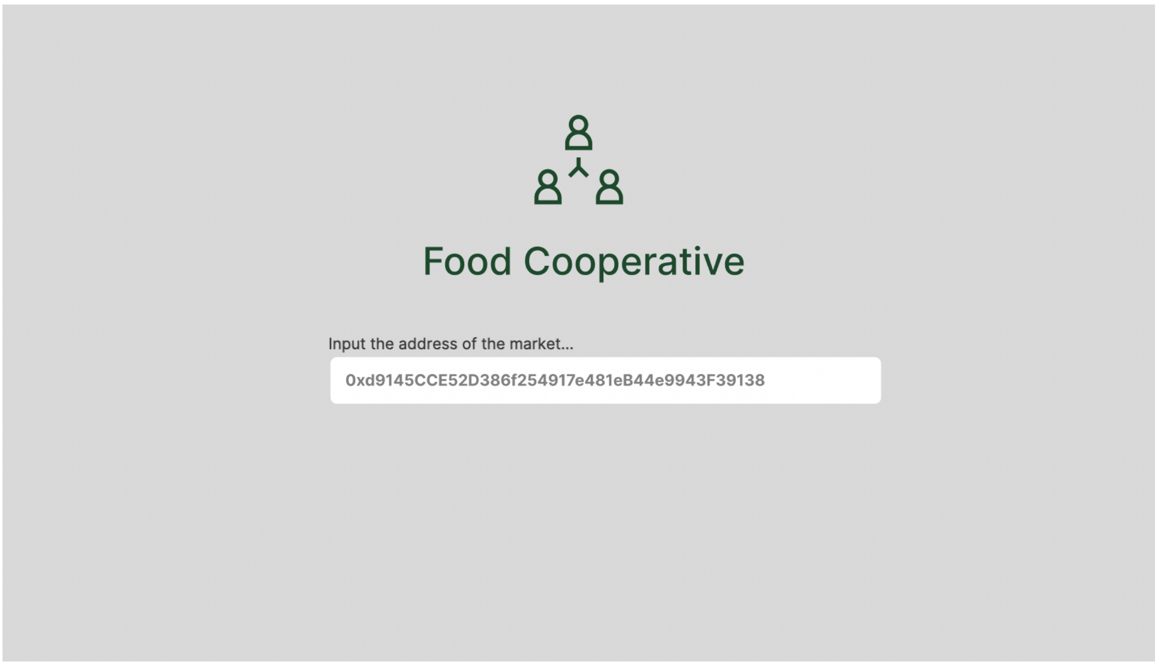
Delivery and distribution are not performed on the blockchain, but they can still take advantage of its benefits.

Each group order is linked to a cooperative that recommends farmers and markets and on-chain records can be used for proof of purchase and dispute resolution. In-person distribution reduces the risk of Sybil attacks (2.5.5) because members collecting multiple orders will have their identities discovered.

Consumers could recruit other users to falsely pick up their orders, but there is no theoretical distinction between this and the recruited users being genuine members.

5.2 User Interface Designs

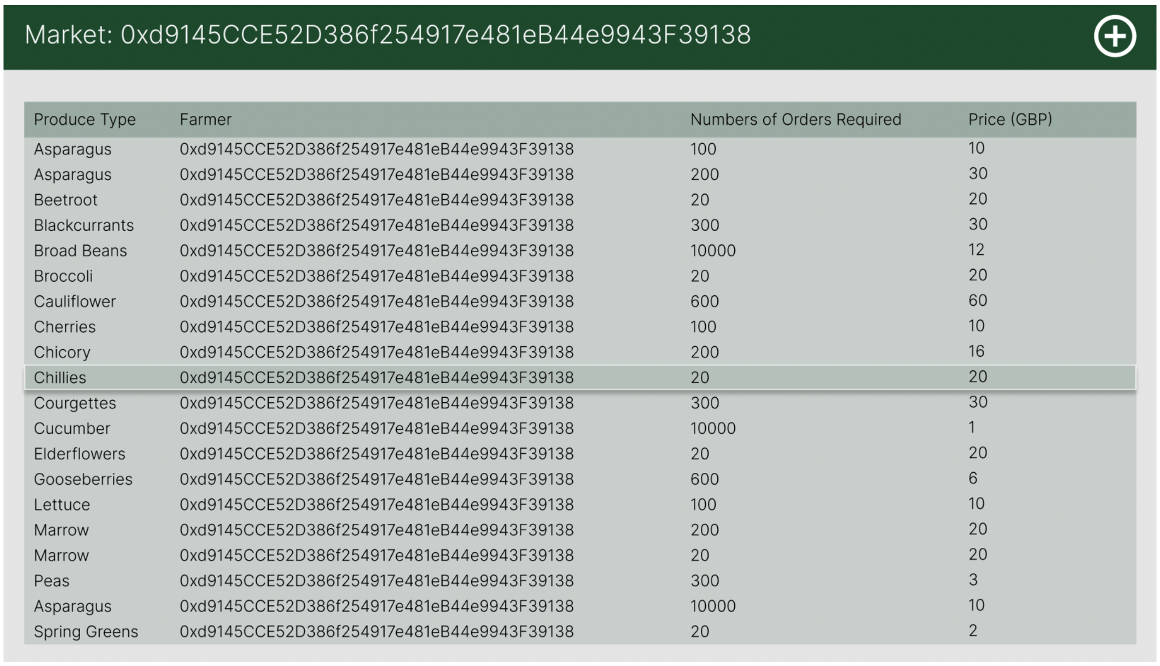
The use of blockchain allows members of the cooperative to be educated about novel technologies (P5). The designs below indicate a user interface for an application front-end. Showing the contract addresses allows members to gain an intuition of the blockchain system.


Food Cooperative

Input the address of the market...

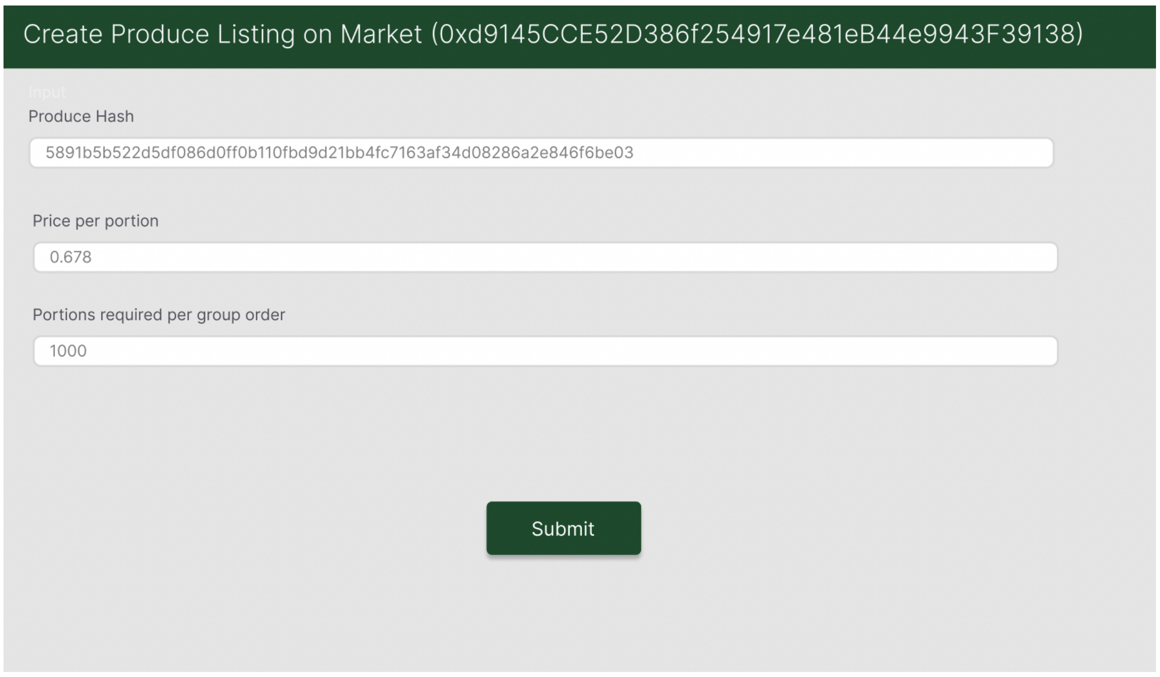
0xd9145CCE52D386f254917e481eB44e9943F39138

Figure 5.1: Start screen


Market: 0xd9145CCE52D386f254917e481eB44e9943F39138

Produce Type	Farmer	Numbers of Orders Required	Price (GBP)
Asparagus	0xd9145CCE52D386f254917e481eB44e9943F39138	100	10
Asparagus	0xd9145CCE52D386f254917e481eB44e9943F39138	200	30
Beetroot	0xd9145CCE52D386f254917e481eB44e9943F39138	20	20
Blackcurrants	0xd9145CCE52D386f254917e481eB44e9943F39138	300	30
Broad Beans	0xd9145CCE52D386f254917e481eB44e9943F39138	10000	12
Broccoli	0xd9145CCE52D386f254917e481eB44e9943F39138	20	20
Cauliflower	0xd9145CCE52D386f254917e481eB44e9943F39138	600	60
Cherries	0xd9145CCE52D386f254917e481eB44e9943F39138	100	10
Chicory	0xd9145CCE52D386f254917e481eB44e9943F39138	200	16
Chillies	0xd9145CCE52D386f254917e481eB44e9943F39138	20	20
Courgettes	0xd9145CCE52D386f254917e481eB44e9943F39138	300	30
Cucumber	0xd9145CCE52D386f254917e481eB44e9943F39138	10000	1
Elderflowers	0xd9145CCE52D386f254917e481eB44e9943F39138	20	20
Gooseberries	0xd9145CCE52D386f254917e481eB44e9943F39138	600	6
Lettuce	0xd9145CCE52D386f254917e481eB44e9943F39138	100	10
Marrow	0xd9145CCE52D386f254917e481eB44e9943F39138	200	20
Marrow	0xd9145CCE52D386f254917e481eB44e9943F39138	20	20
Peas	0xd9145CCE52D386f254917e481eB44e9943F39138	300	3
Asparagus	0xd9145CCE52D386f254917e481eB44e9943F39138	10000	10
Spring Greens	0xd9145CCE52D386f254917e481eB44e9943F39138	20	2

Figure 5.2: List of produce



Create Produce Listing on Market (0xd9145CCE52D386f254917e481eB44e9943F39138)

Input

Produce Hash

5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03

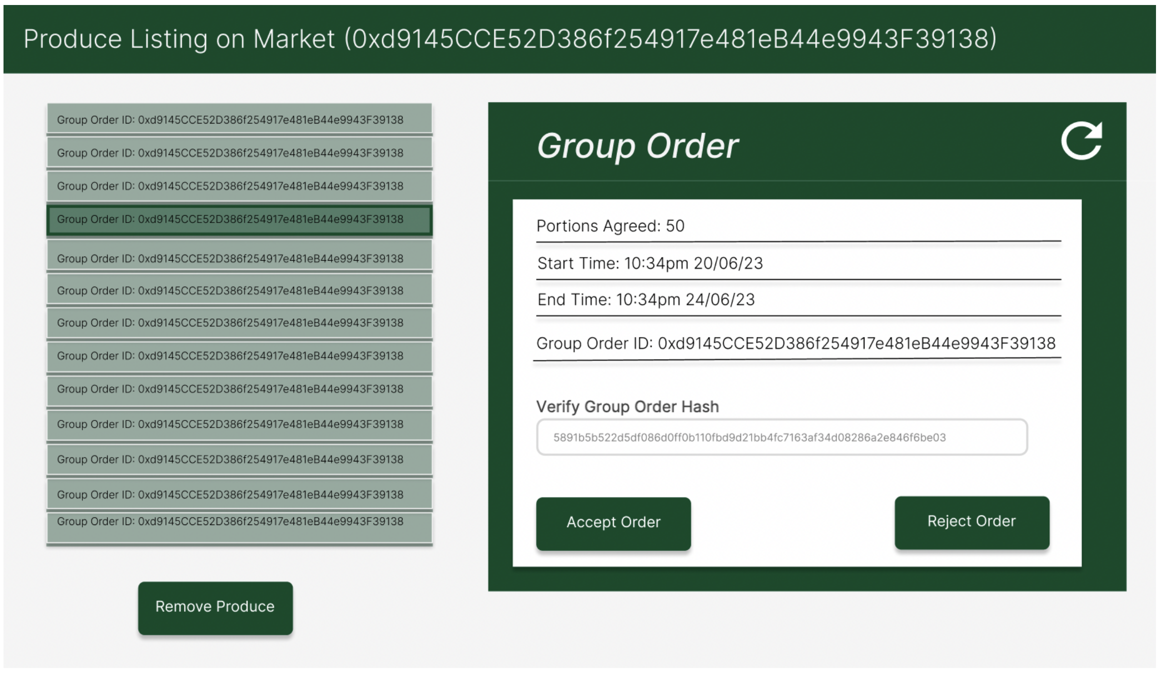
Price per portion

0.678

Portions required per group order

1000

Figure 5.3: Create produce



Produce Listing on Market (0xd9145CCE52D386f254917e481eB44e9943F39138)

Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Group Order

Portions Agreed: 50

Start Time: 10:34pm 20/06/23

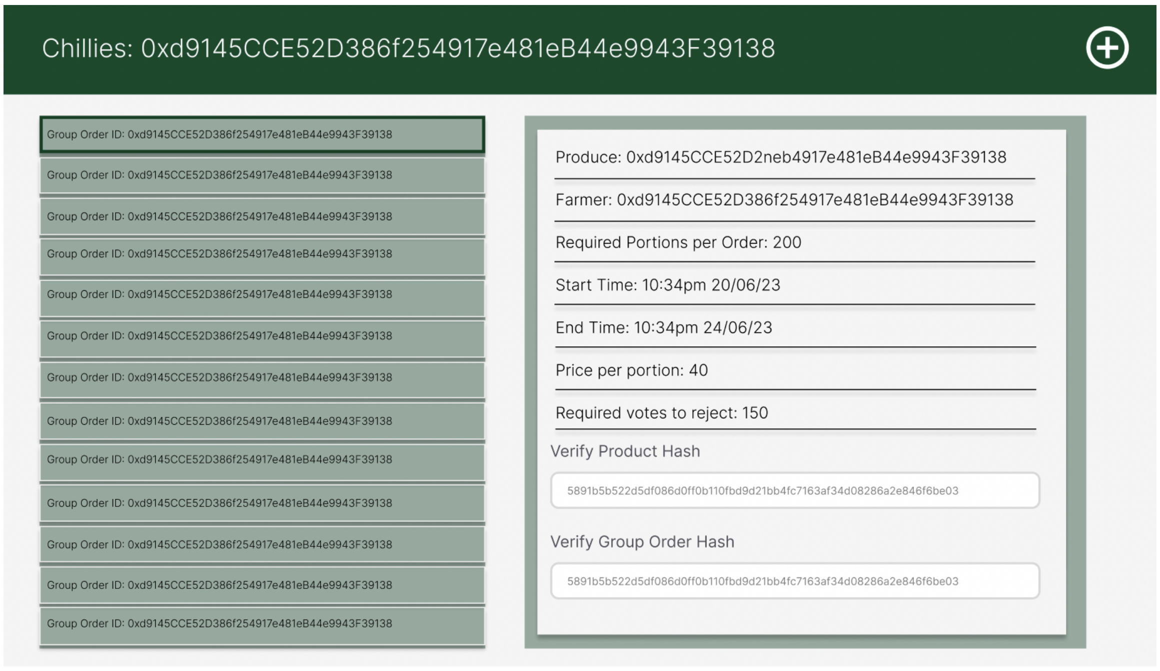
End Time: 10:34pm 24/06/23

Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Verify Group Order Hash

5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03

Figure 5.4: Interface for farmer to accept or reject pending group orders


Chillies: 0xd9145CCE52D386f254917e481eB44e9943F39138

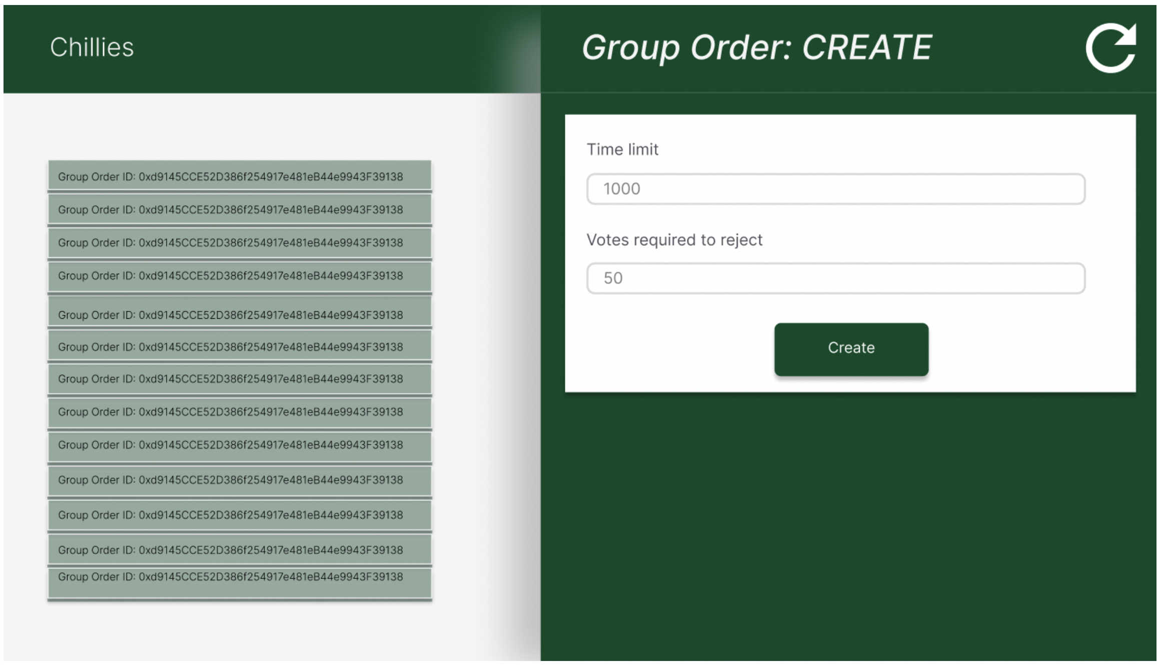
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Produce: 0xd9145CCE52D2neb4917e481eB44e9943F39138
Farmer: 0xd9145CCE52D386f254917e481eB44e9943F39138
Required Portions per Order: 200
Start Time: 10:34pm 20/06/23
End Time: 10:34pm 24/06/23
Price per portion: 40
Required votes to reject: 150

Verify Product Hash
5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03

Verify Group Order Hash
5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f6be03

Figure 5.5: List of current group orders for chillies


Chillies

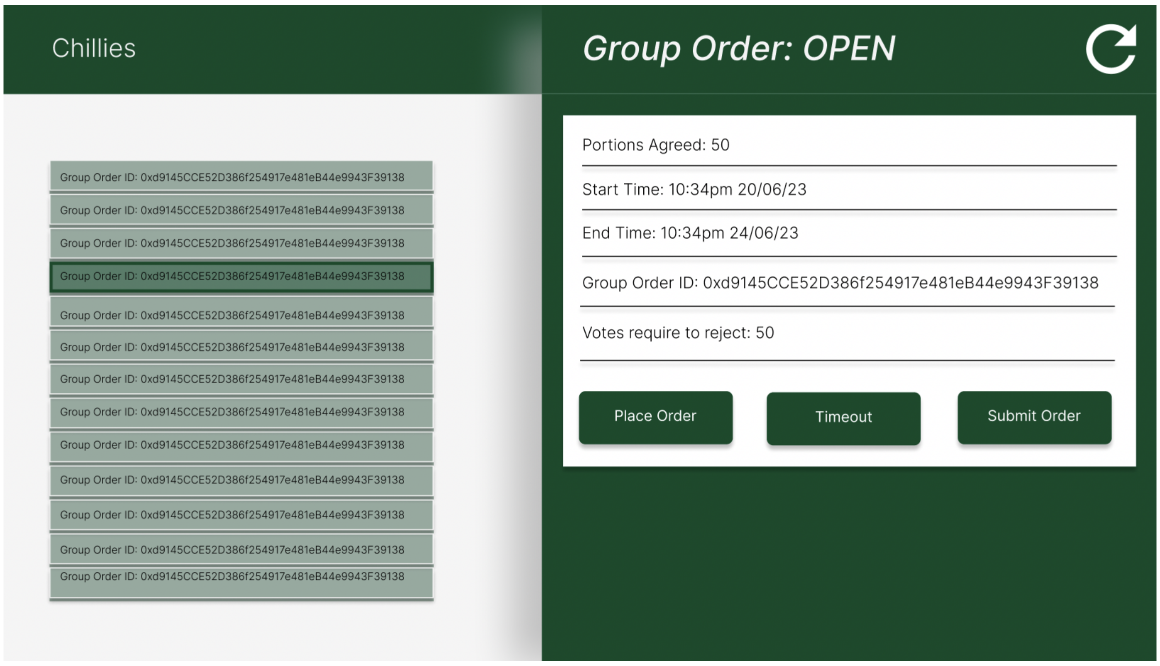
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Group Order: CREATE

Time limit
1000

Votes required to reject
50

Figure 5.6: Interface to create a group order

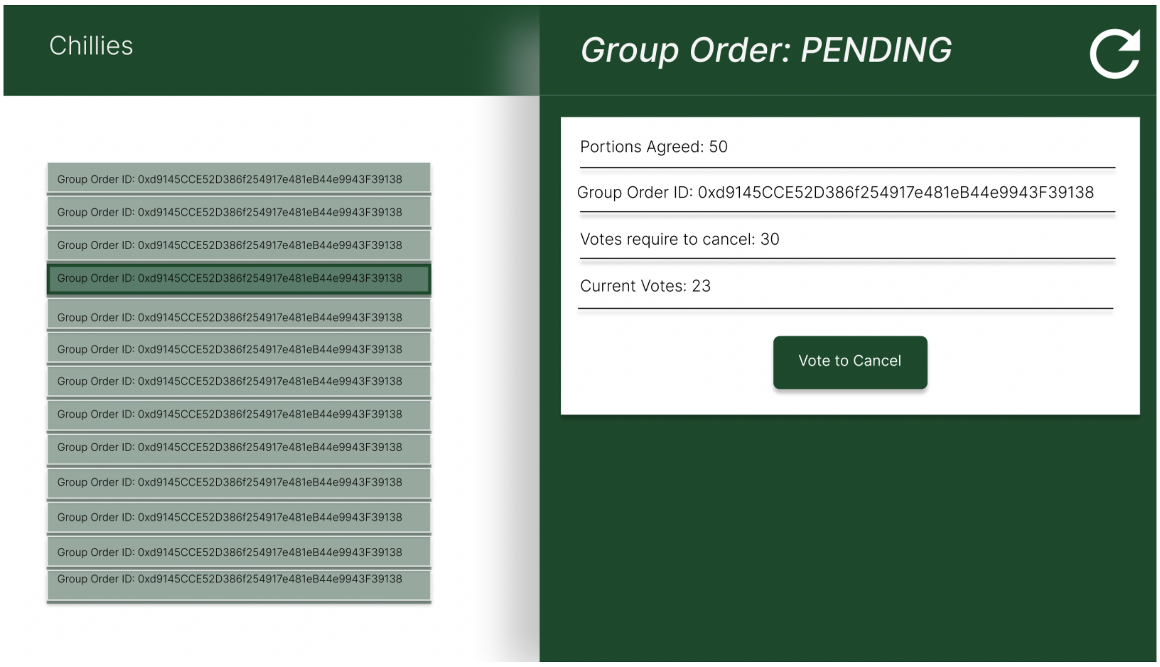

Chillies

Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Group Order: OPEN

Portions Agreed: 50
Start Time: 10:34pm 20/06/23
End Time: 10:34pm 24/06/23
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Votes require to reject: 50

Figure 5.7: Interface for open group order

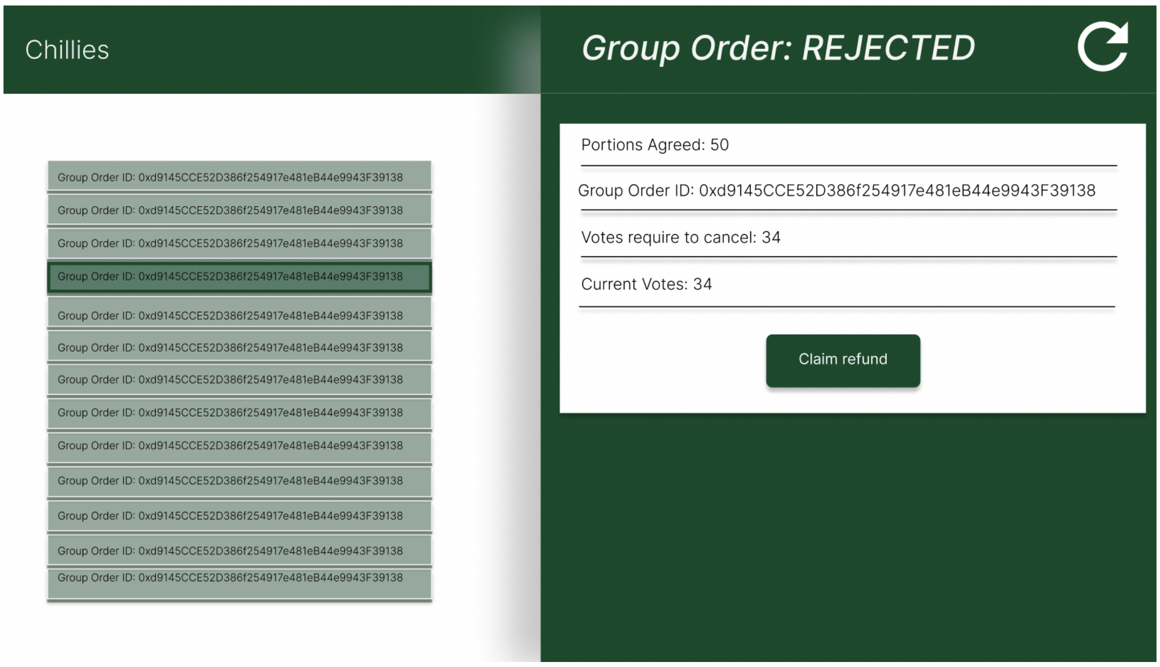

Chillies

Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Group Order: PENDING

Portions Agreed: 50
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Votes require to cancel: 30
Current Votes: 23

Figure 5.8: Interface for pending group order



Chillies

Group Order: REJECTED

Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138
Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Portions Agreed: 50

Group Order ID: 0xd9145CCE52D386f254917e481eB44e9943F39138

Votes require to cancel: 34

Current Votes: 34

Figure 5.9: Interface for rejected group order

Chapter 6

Evaluation and Discussion

The project successfully meets the requirements outlined in Chapter 1.1. This is apparent through the in-depth analysis of smart contract methods in Chapter 4 and through the system design decisions made in Chapter 3, during which every choice brought us further in line with the seven cooperative principles. The gas fees below demonstrate the potential for real-world usage of blockchain in food cooperatives.

We conducted extensive testing to validate the robustness of our system, using Remix IDE to deploy and interact with contracts. The table presents the gas cost in Gwei, one billionth of an ETH, and current GBP-ETH conversions (as of 08/05/2023) that were calculated during these tests.

Function	Gas cost (Gwei)	Gas cost (GBP)
Deploy Market	1340881	2.06
Create Produce	248942	0.38
Close Produce	38500	0.06
Deploy Group Order	1806342	2.77
Place order	85706	0.13
Accept order	54146	0.08
Vote to cancel	50240	0.08
Withdraw funds	41296	0.06
Timeout Group Order	47217	0.08
Reject Group Order	38566	0.07

6.1 Reflection

Initially, I planned to implement both the front-end and back-end systems but quickly realized that writing secure and effective smart contracts required a deep understanding of blockchain. Therefore, a more focused implementation was needed to maximize the impact of my contribution.

I began learning and experimenting with web3.js, the JavaScript library used for interacting with Ethereum. However, it became clear that this would require a significant amount of time, which could be better spent working on the on-chain components that directly address the project's goals. After extensive background reading and experimentation, I determined that using Remix IDE was the most effective solution. The front-end of the application was designed to concentrate efforts on addressing the principles of the project.

Time was spent exploring system components that were not included in the final implementation. To keep smart contracts simple and secure, only functionality that would benefit from moving on-chain was included.

For example, adding the food delivery service on-chain was considered, this would have been effective for transparent tracking of orders. Since the project context is for use in a preexisting organization, delivery could be coordinated at a cheaper cost off-chain.

If I were to complete this project again with my current knowledge, I could significantly reduce the time spent identifying problems that can be effectively solved using blockchain. Initially, I had to balance learning about this new technology with making implementation decisions. It was only after completing background research that I realized that it was not appropriate to have certain components implemented on-chain.

Chapter 7

Related Work

There is a significant body of academic work on the use of blockchain technology in the cooperative economy to replace traditional for-profit third parties. Examples include decentralised car sharing (Zhou et al. 2020; Kim et al. 2021; Valaštín et al. 2019), AirBnb-like home sharing (Schneck, Tumasjan, and Welpe 2020) and a second hand goods rental system (Bogner, Chanson, and Meeuw 2016; Tiansong and Yu 2020).

Tiansong and Yu (2020) addressed a lack of data transparency in the second-hand clothing rental market through an on-chain clothes renting system.

Blockchain has been analysed in depth in the agriculture industry, although primarily in supply chain traceability (Li, Lee, and Gharehgozli 2021; Pakseresht et al. 2023; Duan et al. 2020). Supply chains are widely considered an effective use of blockchain technology, although still in early stages it is thought this technology will ease pressure on global supply chains and increase sustainability (Saberi et al. 2019).

There is limited research into use of blockchain in agricultural trading, Leduc, Kubler, and Georges (2021) addresses the gap in research by proposing a marketplace platform between farmers and third party stake holders. But they assume these third parties to be other farmers or retailers, hence does not facilitate purchasing directly with consumers.

Kabi and Franqueira (2019) demonstrates a proof of concept system for a decentralised market place. They successfully reduce the gas costs to lower levels that traditional fees charged on Amazon and eBay. Their system relies on users placing individual orders, following the structure of a traditional market rather than a cooperative.

Rocas-Royo (2021) analyses the uses of blockchain in the context of four cooperatives, identifying, as we also do in this paper, that internal systems already exist for managing trust that do not necessarily benefit from moving on chain. The paper also highlights where blockchain would provide an advantage, for example in time banking working hours, supply-chain automation, democratic governance and cooperative activities.

Our project presents a use case that connects research on decentralized marketplaces with the use of blockchain technology in the cooperative economy. Our prototype implementation supports the recommendations made in Rocas-Royo (2021), further highlighting the advantages of bringing cooperative activities onto the blockchain.

Chapter 8

Conclusion

8.1 Summary

We designed, implemented and analysed a smart contract system for bulk wholesale ordering in food cooperatives (International Co-operative Alliance 2015). Throughout the application, we have carefully incorporated the principles of the cooperative movement. We have also conducted extensive testing, showing the system gas costs are within appropriate ranges for real-world usage. The project contributes to a growing collection of work demonstrating the practical applications of blockchain in the cooperative economy and highlights the potential of blockchain solutions in addressing pressing global issues.

8.2 Limitations

Little research has been done on how the security vulnerabilities of smart contracts change with the Ethereum Merge. Most literature discusses PoW when mitigating security issues in smart contracts, but it's unclear in PoS how timestamp dependency and front running will affect the system's security. Ethereum PoS is new so there are few papers exploring its specific security considerations for smart contracts. This is expected since the Merge happened part way through the project (15 September 2022).

The system relies heavily on the trust of the external cooperative organisation, and does not operate autonomously from this. This means that certain security considerations, such as preventing Sybil attacks, have been made under this assumption.

8.3 Further Extensions

The system has been designed for use with pre-existing cooperative organizations. However, we could extend the project by working towards a more independent system.

One step towards this would be implementing an on-chain reputation system for farmers and market contacts. Several papers have explored blockchain-based reputation systems (Dennis and Owen 2015; Arshad et al. 2022; Schaub, Bazin, and Brunie 2016), all identifying the extra care that must be taken with Sybil attacks.

We could use oracles to provide verification of food distribution on-chain. This would be similar to supply chain tracking, but on a more local level. Possible implementations include a decentralised voting oracle (Adler et al. 2018) for dispute resolution, or an on-chain delivery tracking system with IoT tags (Caro et al. 2018).

Rocas-Royo (2021) identifies other effective use cases for blockchain in food cooperatives that could be explored, such as tracking working hours. In the wider scope of the cooperative movement, we could explore similar organizational activities that could be moved on-chain.

Appendix A

Smart Contracts

A.1 Market Contract

```
pragma solidity ^0.8.0 .0;
import "./Produce.sol";
import "@openzeppelin/contracts/proxy/Clones.sol";

contract Market {
    event ProduceAddedToMarket(address produce, address farmer);
    event ProduceDeletedFromMarket(address produce);

    //This is the immutable definition of produce in this market
    address immutable produceImplementation;

    // This mapping stores avaliable produce
    mapping(address=> bool) public produceList;
    // This mapping stores farmers who have produce on the Market
    mapping(address=> bool) public farmerList;

    constructor() {
        produceImplementation = address(new Produce());
    }

    function addProduce(
        bytes32 _produceHash,
        uint256 _price,
        uint256 _orderSize
        ) external {
            address newProduceClone = Clones.clone(
               produceImplementation);
            Produce(newProduceClone).initilize(
                 msg.sender,
                 _produceHash,
                 _price,
                 _orderSize
            );
            produceList[address(newProduceClone)] = true;
            farmerList[msg.sender] = true;
            emit ProduceAddedToMarket(address(newProduceClone), msg.
                sender);
    }

    function removeProduce() public {
        require(produceList[msg.sender], "This produce is not listed
            on the Market");
        produceList[msg.sender] = false;
        emit ProduceDeletedFromMarket(msg.sender);
    }


}
```

A.2 Produce Contract

```
pragma solidity ^0.8.0;
import "./Market.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract Produce is Ownable{
    event OrderQueued(address groupOrder);

    address public farmer;
    address public market;
    bytes32 public produceHash;
    uint256 public price;
    uint256 public orderSize;

    mapping(address=> bool) public orderList;

    enum State {
        UNINITIALIZED,
        OPEN,
        CLOSED
    }
    State private state;

    constructor() {
        state = State.UNINITIALIZED;
    }

    function initilize(
        address _farmer,
        bytes32 _produceHash,
        uint256 _price,
        uint256 _orderSize
    ) external {
        require(state==State.UNINITIALIZED, "This produce has
            already been initilized");
        _transferOwnership(_farmer);
        market = msg.sender;
        produceHash = _produceHash;
        price = _price;
        orderSize = _orderSize;
        state = State.OPEN;
    }

    /// Place an order an order of the produce
    function placeOrder() external {
        require(state == State.OPEN, "produce is not open for
           ordering");
        require(
            msg.sender != owner(),
            "farmer cannot place order for their own produce"
        );
        require(!orderList[msg.sender],  "order has already been
           placed");
        orderList[msg.sender] = true;
        emit OrderQueued(msg.sender);
    }

    function removeOrder() external {
        require(orderList[msg.sender], "no order has been placed");
        orderList[msg.sender] = false;
    }

    function closeProduce() external onlyOwner {
        require(state == State.OPEN, "produce is already closed");
        state = State.CLOSED;
        Market(market).removeProduce();
    }


}
```

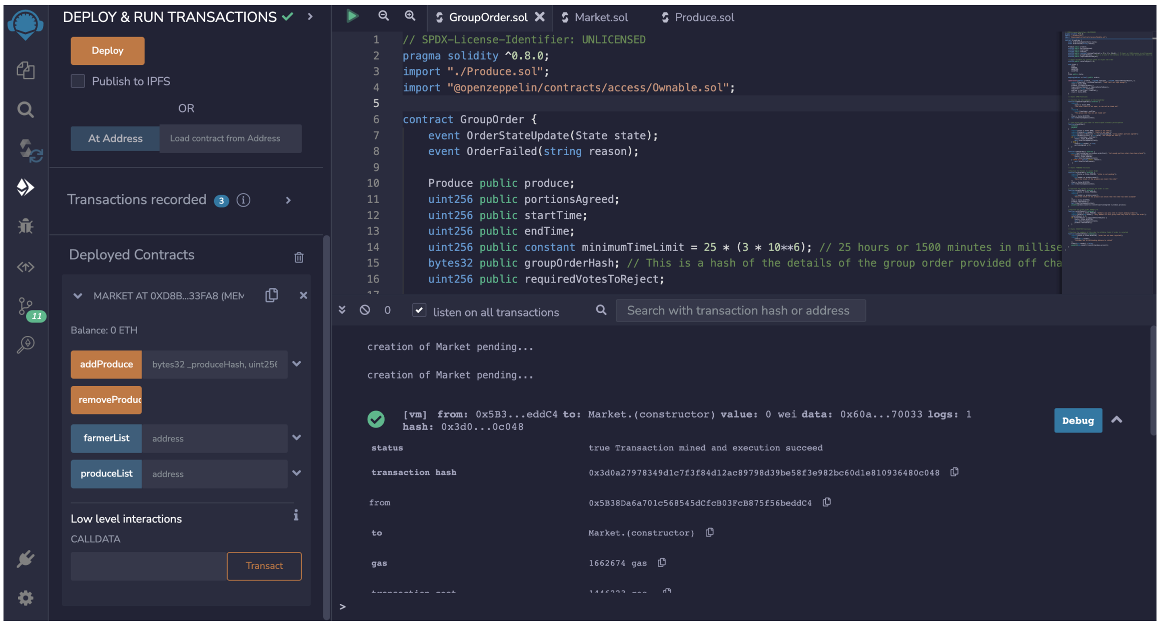
A.3 Group Order Contract

```
pragma solidity ^0.8.0;
import "./Produce.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract GroupOrder {
    event OrderStateUpdate(State state);
    event OrderFailed(string reason);

    Produce public produce;
    uint256 public portionsAgreed;
    uint256 public startTime;
    uint256 public endTime;
    uint256 public constant minimumTimeLimit = 25 * (3 * 10**6); //
       25 hours or 1500 minutes in milliseconds
    bytes32 public groupOrderHash; // This is a hash of the details
       of the group order provided off chain such as delivery
```

```
        location and pick up instructions
    uint256 public requiredVotesToReject;

    // State variable for counting votes to reject the order
    uint256 public votesToReject = 0;

    enum State {
        OPEN,
        PENDING,
        REJECTED,
        ACCEPTED
    }
    State public state;

    mapping(address => bool) public orders;

    constructor(address produce_, uint256 timeLimit_, uint256
        requiredVotesToReject_) {
        require(timeLimit_ > minimumTimeLimit, "time limit not high
            enough");
        state = State.OPEN;
        produce = Produce(produce_);
        requiredVotesToReject = requiredVotesToReject_;
        startTime = block.timestamp;
        endTime = startTime + timeLimit_;
        state = State.OPEN;
    }

    // State: OPEN Functions

    // function for the timeout of the GroupOrder
    function timeOutGroupOrder() external {
        require(
            state == State.OPEN,
            "The order state is not open, so can not be timed out"
        );
        require(
            block.timestamp > endTime,
            "The group order has not yet timed out"
        );
        state = State.REJECTED;
        emit OrderStateUpdate(state);
    }

    /// One person gets one order to ensure equal economic
        participation
    function placeOrder()
        external
        payable
    {
        require(state == State.OPEN, "state is not open");
        require(!orders[msg.sender], "state is not open");
        require(produce.orderSize() - 1 >= portionsAgreed, "wrong
            number portions agreed");
        require(produce.price() <= msg.value, "not enough wei sent")
            ;
        if (block.timestamp > endTime) {
            state = State.REJECTED;
            emit OrderStateUpdate(state);
        } else {
            orders[msg.sender] = true;
            portionsAgreed += 1;
        }
    }

    function submitOrder() external {
       require(portionsAgreed == produce.orderSize(), "not enough
          portion orders have been placed");
        try produce.placeOrder() {
            state = State.PENDING;
            emit OrderStateUpdate(state);
        } catch Error(string memory reason) {
            emit OrderFailed(reason);
         }
    }

    // State: PENDING Functions

    //Function for Farmer to reject Order
    function rejectOrder() external {
        require(state == State.PENDING, "state is not pending");
        require(
            msg.sender == produce.owner(),
            "Only the farmer of the produce can reject the order"
        );
        state = State.REJECTED;
        emit OrderStateUpdate(state);
    }

    // Function for farmer to notify the order is sent
    function acceptOrder() external {
        require(state == State.PENDING);
        require(
            msg.sender == produce.owner(),
            "Only the farmer of the produce can notify that the
                order has been accepted"
        );
        state = State.ACCEPTED;
        produce.removeOrder();
        emit OrderStateUpdate(state);
        payable(produce.owner()).transfer(portionsAgreed * produce.
            price());
    }

    //Function for group order members to vote to cancel the order
    function voteToCancel() external {
        require(state == State.PENDING, "members can only vote to
            reject pending orders");
        require(orders[msg.sender], "only members of this group
            order may vote to reject the order");
        votesToReject += 1;
        if (votesToReject > requiredVotesToReject) {
            state = State.REJECTED;
            emit OrderStateUpdate(state);
            produce.removeOrder();
        }
    }

    // State: REJECTED Functions

    //Function for members of the order to withdraw funds if order
        is rejected
    function withdrawFunds() external {
        require(state == State.REJECTED, "order has not been
            rejected");
        require(
            orders[msg.sender],
            "customer has no outstanding balance to refund"
        );
        orders[msg.sender] = false;
        payable(msg.sender).transfer(produce.price());
    }
```

Appendix B

Remix IDE Screenshot



```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;
import "./Produce.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

contract GroupOrder {
    event OrderStateUpdate(State state);
    event OrderFailed(string reason);

    Produce public produce;
    uint256 public portionsAgreed;
    uint256 public startTime;
    uint256 public endTime;
    uint256 public constant minimumTimeLimit = 25 * (3 * 10**6); // 25 hours or 1500 minutes in millise
    bytes32 public groupOrderHash; // This is a hash of the details of the group order provided off cha
    uint256 public requiredVotesToReject;
```

```
creation of Market pending...

creation of Market pending...

[vm] from: 0x5B3...eddC4 to: Market.(constructor) value: 0 wei data: 0x60a...70033 logs: 1
hash: 0x3d0...0c048
status                  true Transaction mined and execution succeed
transaction hash        0x3d0a27978349d1c7f3f84d12ac89798d39be58f3e982bc60d1e810936480c048
from                    0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                      Market.(constructor)
gas                     1662674 gas
```

Figure B.1: Interface for interacting with deployed market contract

References

Adler, John et al. (2018). “Astraea: A Decentralized Blockchain Oracle”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1145–1152. DOI: 10.1109/Cybermatics_2018.2018.00207.

Arshad, Junaid et al. (2022). “REPUTABLE–A Decentralized Reputation System for Blockchain-Based Ecosystems”. In: *IEEE Access* 10, pp. 79948–79961. DOI: 10.1109/ACCESS.2022.3194038.

Berge, Simon, Wayne Calwell, and Phil Mont (2016). “GOVERNANCE OF NINE ONTARIO FOOD CO-OPERATIVES”. In: *Annals of Public and Cooperative Economics* 87.3, pp. 457–474. DOI: https://doi.org/10.1111/apce.12134. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/apce.12134.

Bogner, Andreas, Mathieu Chanson, and Arne Meeuw (2016). “A Decentralised Sharing App Running a Smart Contract on the Ethereum Blockchain”. In: IoT’16. Stuttgart, Germany: Association for Computing Machinery. ISBN: 9781450348140. DOI: 10.1145/2991561.2998465. URL: https://doi.org/10.1145/2991561.2998465.

Al-Breiki, Hamda et al. (2020). “Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges”. In: *IEEE Access* 8, pp. 85675–85685. DOI: 10.1109/ACCESS.2020.2992698.

Buterin, Vitalik (2013). “Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform”. In: URL: https://github.com/ethereum/wiki/wiki/White-Paper.

Caldarelli, Giulio (Oct. 2020). “Understanding the Blockchain Oracle Problem: A Call for Action”. In: *Information* 11.11, p. 509. ISSN: 2078-2489. DOI: 10.3390/info11110509. URL: http://dx.doi.org/10.3390/info11110509.

Caro, Miguel Pincheira et al. (2018). “Blockchain-based traceability in Agri-Food supply chain management: A practical implementation”. In: *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*, pp. 1–4. DOI: 10.1109/IOT-TUSCANY.2018.8373021.

CoinMarketCap (accessed on April 18, 2023). *Chain Ranking*. https://coinmarketcap.com/chain-ranking/.

Consensys (2023). *Smart Contract Best Practices*. [Online; accessed May 1, 2023]. URL: https://consensys.github.io/smart-contract-best-practices/attacks/denial-of-service/.

Deller, Steven et al. (2009). *Research on the Economic Impact of Cooperatives.* Madison: University of Wisconsin Center for Cooperatives.

Dennis, Richard and Gareth Owen (2015). "Rep on the block: A next generation reputation system based on the blockchain". In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pp. 131–138. DOI: `10.1109/ICITST.2015.7412073`.

Dika, Ardit and Mariusz Nowostawski (2018a). "Security Vulnerabilities in Ethereum Smart Contracts". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 955–962. DOI: `10.1109/Cybermatics_2018.2018.00182`.

— (July 2018b). "Security Vulnerabilities in Ethereum Smart Contracts". In: pp. 955–962. DOI: `10.1109/Cybermatics_2018.2018.00182`.

Douceur, John R (2002). "The sybil attack". In: *International Workshop on Peer-to-Peer Systems.* Springer.

Duan, Jiang et al. (2020). "A Content-Analysis Based Literature Review in Blockchain Adoption within Food Supply Chain". In: *International Journal of Environmental Research and Public Health* 17.5. ISSN: 1660-4601. DOI: `10.3390/ijerph17051784`. URL: `https://www.mdpi.com/1660-4601/17/5/1784`.

Ethereum (2023). *Gas and Transaction Fees on Ethereum.* `https://ethereum.org/en/developers/docs/gas/`. [Online; accessed 13-April-2023].

Fairbairn, Brett (1994). *Meaning of Rochdale: The Rochdale Pioneers and the Co-operative Principles.* Occasional Paper Series 94.02. Includes bibliographical references. Centre for the Study of Co-operatives, University of Saskatchewan. ISBN: 978-0-88880-317-7.

Foundation, Ethereum (Dec. 2021). *The Merge.* `https://ethereum.org/en/eth2/the-merge/`.

Hepp, Thomas et al. (2018). In: *it - Information Technology* 60.5-6, pp. 283–291. DOI: `doi:10.1515/itit-2018-0019`. URL: `https://doi.org/10.1515/itit-2018-0019`.

International Co-operative Alliance (2015). *ICA Guidance Notes.* `https://ica.coop/sites/default/files/2021-11/ICA\%20Guidance\%20Notes\%20EN.pdf`. Accessed on April 23, 2023.

Kabi, Oliver R. and Virginia N. L. Franqueira (2019). "Blockchain-Based Distributed Marketplace". In: *International Conference on Blockchain and Trustworthy Systems.* Springer, pp. 210–224.

Kaleem, Mudabbir and Weidong Shi (2021). "Demystifying Pythia: A Survey of ChainLink Oracles Usage on Ethereum". In: *Financial Cryptography and Data Security. FC 2021 International Workshops.* Ed. by Matthew Bernhard et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 115–123.

Kim, Myeonghyun et al. (2021). "Design of Secure Decentralized Car-Sharing System Using Blockchain". In: *IEEE Access* 9, pp. 54796–54810. DOI: `10.1109/ACCESS.2021.3071499`.

Leduc, Guilain, Sylvain Kubler, and Jean-Philippe Georges (2021). "Innovative blockchain-based farming marketplace and smart contract performance evaluation". In: *Jour-*

nal of Cleaner Production 306, p. 127055. ISSN: 0959-6526. DOI: `https://doi.org/10.1016/j.jclepro.2021.127055`. URL: `https://www.sciencedirect.com/science/article/pii/S0959652621012749`.

Leeming, Gary, James Cunningham, and John Ainsworth (2019). "A Ledger of Me: Personalizing Healthcare Using Blockchain Technology". In: *Frontiers in Medicine* 6. ISSN: 2296-858X. DOI: `10.3389/fmed.2019.00171`. URL: `https://www.frontiersin.org/articles/10.3389/fmed.2019.00171`.

Li, Kunpeng, Jun-Yeon Lee, and Amir Gharehgozli (2021). "Blockchain in food supply chains: a literature review and synthesis analysis of platforms, benefits and challenges". In: *International Journal of Production Research* 0.0, pp. 1–20. DOI: `10.1080/00207543.2021.1970849`. eprint: `https://doi.org/10.1080/00207543.2021.1970849`. URL: `https://doi.org/10.1080/00207543.2021.1970849`.

Little, Ruth, Damian Maye, and Brian Ilbery (2010). "Collective purchase: moving local and organic foods beyond the niche market". In: *Environment and Planning A* 42.8, pp. 1797–1813.

Luu, Loi et al. (2016). "Making Smart Contracts Smarter". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. Vienna, Austria: Association for Computing Machinery, pp. 254–269. ISBN: 9781450341394. DOI: `10.1145/2976749.2978309`. URL: `https://doi.org/10.1145/2976749.2978309`.

Ma, Fuchen et al. (2021). "Security reinforcement for Ethereum virtual machine". In: *Information Processing Management* 58.4, p. 102565. ISSN: 0306-4573. DOI: `https://doi.org/10.1016/j.ipm.2021.102565`. URL: `https://www.sciencedirect.com/science/article/pii/S0306457321000674`.

Mc Carthy, Ultan et al. (2018). "Global food security – Issues, challenges and technological solutions". In: *Trends in Food Science Technology* 77, pp. 11–20. ISSN: 0924-2244. DOI: `https://doi.org/10.1016/j.tifs.2018.05.002`. URL: `https://www.sciencedirect.com/science/article/pii/S0924224417305125`.

Modi, Ritesh (June 2019). *Solidity Programming Essentials - Second Edition*. 2nd. Packt Publishing.

Nakamoto, Satoshi (Mar. 2009). "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *Cryptography Mailing list at https://metzdowd.com*.

OpenZeppelin (2023). *OpenZeppelin: A framework to build secure smart contracts on Ethereum*. `https://openzeppelin.com/`. Accessed on April 13, 2023.

Pakseresht, Ashkan et al. (2023). "The intersection of blockchain technology and circular economy in the agri-food sector11This work was supported by the Swedish University of Agricultural Sciences, Sweden, by a scholarship from L Nannesson's foundation, grant number Dnr SLU.ua.2019.3.1.5-617." In: *Sustainable Production and Consumption* 35, pp. 260–274. ISSN: 2352-5509. DOI: `https://doi.org/10.1016/j.spc.2022.11.002`. URL: `https://www.sciencedirect.com/science/article/pii/S2352550922002986`.

Parizi, Reza M., Amritraj, and Ali Dehghantanha (2018). "Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security". In: *Blockchain – ICBC 2018*. Ed. by Shiping Chen, Harry Wang, and Liang-Jie Zhang. Cham: Springer International Publishing, pp. 75–91.

Price, Rob (2016). *Digital Currency Ethereum is Cratering amid Claims of $50 Million Hack.* URL: https://www.businessinsider.com.au/dao-hacked-ethereum-crashing-in-value-tens-of-millions-allegedly-stolen-2016-6?r=UK&IR=T (visited on 10/05/2022).

Rocas-Royo, Marc (2021). "The Blockchain That Was Not: The Case of Four Cooperative Agroecological Supermarkets". In: *Frontiers in Blockchain* 4. ISSN: 2624-7852. DOI: 10.3389/fbloc.2021.624810. URL: https://www.frontiersin.org/articles/10.3389/fbloc.2021.624810.

Saberi, Sara et al. (2019). "Blockchain technology and its relationships to sustainable supply chain management". In: *International Journal of Production Research* 57.7, pp. 2117–2135. DOI: 10.1080/00207543.2018.1533261. eprint: https://doi.org/10.1080/00207543.2018.1533261. URL: https://doi.org/10.1080/00207543.2018.1533261.

Sayeed, Sarwar, Hector Marco-Gisbert, and Tom Caira (2020). "Smart Contract: Attacks and Protections". In: *IEEE Access* 8, pp. 24416–24427. DOI: 10.1109/ACCESS.2020.2970495.

Schaub, Alexander, Rémi Bazin, and Lionel Brunie (May 2016). "A Trustless Privacy-Preserving Reputation System". In: pp. 398–411. ISBN: 978-3-319-33629-9. DOI: 10.1007/978-3-319-33630-5_27.

Schneck, Patrick, Andranik Tumasjan, and Isabell M. Welpe (2020). "Next Generation Home Sharing: Disrupting Platform Organizations with Blockchain Technology and the Internet of Things?" In: *Blockchain and Distributed Ledger Technology Use Cases: Applications and Lessons Learned.* Ed. by Horst Treiblmaier and Trevor Clohessy. Cham: Springer International Publishing, pp. 267–287. ISBN: 978-3-030-44337-5. DOI: 10.1007/978-3-030-44337-5_13. URL: https://doi.org/10.1007/978-3-030-44337-5_13.

SWC-Registry (n.d.). *SWC-132: Unexpected Ether balance.* https://swcregistry.io/docs/SWC-132. Accessed: April 16, 2023.

Szabo, Nick (1996). "Smart contracts: building blocks for digital free markets". In: *First Monday* 2.9. URL: http://firstmonday.org/ojs/index.php/fm/article/view/548/469.

Tiansong, Li and Liu Yu (2020). "Design and Implementation of Second-Hand Goods Renting System Based On Ethereum Smart Contract". In: *Proceedings of the 4th International Conference on Intelligent Information Processing.* ICIIP '19. China, China: Association for Computing Machinery, pp. 346–351. ISBN: 9781450361910. DOI: 10.1145/3378065.3378131. URL: https://doi.org/10.1145/3378065.3378131.

Torres, Christof Ferreira, Ramiro Camino, and Radu State (2021). "Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain". In: *CoRR* abs/2102.03347. arXiv: 2102.03347. URL: https://arxiv.org/abs/2102.03347.

UNICEF et al. (2022). *The State of Food Security and Nutrition in the World 2022. Repurposing food and agricultural policies to make healthy diets more affordable.* Rome: FAO. ISBN: 978-92-5-136499-4. DOI: https://doi.org/10.4060/cc0639en.

Valaštín, Viktor et al. (2019). “Blockchain Based Car-Sharing Platform”. In: *2019 International Symposium ELMAR*, pp. 5–8. DOI: `10.1109/ELMAR.2019.8918650`.

Vyper Development Team (2023). *Vyper Documentation.* `https://docs.vyperlang.org/en/latest/index.html`. Accessed on April 13, 2023.

Wohrer, Maximilian and Uwe Zdun (2018). “Smart contracts: security patterns in the ethereum ecosystem and solidity”. In: *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pp. 2–8. DOI: `10.1109/IWBOSE.2018.8327565`.

Wöhrer, Maximilian and Uwe Zdun (2018). “Design Patterns for Smart Contracts in the Ethereum Ecosystem”. In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1513–1520. DOI: `10.1109/Cybermatics_2018.2018.00255`.

Zhang, Zhuo et al. (2023). “Demystifying Exploitable Bugs in Smart Contracts”. In: *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE.

Zhou, Qihao et al. (2020). “A Decentralized Car-Sharing Control Scheme Based on Smart Contract in Internet-of-Vehicles”. In: *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pp. 1–5. DOI: `10.1109/VTC2020-Spring48590.2020.9129439`.