University of Trento
Department of Computer Science
Blockchain

UNIVERSITÀ DI TRENTO

Report — Blockmeat

Sebastiano Cassol `sebastiano.cassol@studenti.unitn.it`
Marcel Pascal Stolin `marcelpascal.stolin@studenti.unitn.it`

19th August 2023

1 Introduction

Nowadays Blockchain isn't limited to financial applications as intended originally. With the increasing popularity of Decentralized Applications (dApps), both in industry and by consumers, the area of application of Blockchain technology has emerged significantly.
One area is the food industry, which is considered a *behind-the-scenes* business. Consumers don't know, who is involved and why certain steps are necessary, in the processing. Applying Blockchain technology provides a solution to this problem by providing more transparency. It will make producers more accountable and force them to deliver higher quality products by providing more safety for workers and the environment (including animals, farms, etc.). Another advantage of the Blockchain is, that it removes the need for intermediaries, as it provides for trust between involved entities through cryptography. This will give producers the ability to apply better and more fair prices for all involved stakeholders because no fees have to be paid to intermediaries like vendors or supermarkets.

This project introduces *Blockmeat*, a decentralized application (dApp) that targets to increase transparency during the buying and processing phase of meat products. One part is to provide a crowdfunding platform for customers, to buy high-quality and environmentally sustainable meat directly from the producer. Secondly, after a crowdfunding campaign has been funded successfully, supply chain management targets to make the process of the meat product, from producer to consumer, more transparent.

Blockmeat is developed as an exam condition for the course Blockchain, taught by Luigi Telesca and Mahfuzul Md. Islam at the University of Trento during the second semester of the academic year 2022/23. The goal is to present skills and expertise learned during the lectures of the course.

The report takes usage of the methods of software engineering for Blockchain applications introduced by Marchesi et al[1] for documentation purposes.

2 Business Idea

This section describes the business idea of this project. The problem statement is described in Section 2.1, and the proposed solution at Section 2.2. Next, a short analysis of the market is given in Section 2.3. Furthermore, potential users are introduced in Section 2.4. Lastly, Section 2.3 shows how revenue is generated using this project.

2.1 Problem Statement

As mentioned previously, the meat industry is a *behind-the-scenes* business, which rarely provides transparency over the process from the producer to the end consumer. This results, that consumers decide not to buy meat products from vendors that can't provide more transparency.

The 2019 Food & Health Survey[2] shows, that for 52% of asked consumers, the origin is an important factor when buying food, and 54% of consumers consider environmental sustainability as an important attribute. However, the problem becomes more clear when considering, that 63% of the consumers agree to the statement, that it is hard to know whether food choices are environmentally sustainable. Considering that, 63% would prefer sustainable food choices if it would be easier to know if the food is environmentally sustainable.

The targets of this project are the consumers mentioned in this survey, especially meat consumers. The value proposition canvas, attached at Section A, gives more details about the needs and the problems of the targeted audience, as well as a proposed solution (introduced next at Section 2.2) to provide for the needs of the meat consumer.

The targeted audience is any person, who plans to have a healthier meat diet and desires to have a clearer understanding about how meat choices are being produced. The benefits that the consumer wishes for, are a healthier diet and more sustainable shopping behavior. This can be reached with a more transparent way of checking where the food comes from and who was involved in the process. Additionally, the customer desires an easier way to buy the meat, without the involvement of an intermediary, like a supermarket. The jobs of the customer mostly involve going to the supermarket, trying to figure out what is the highest quality meat, checking if it is produced sustainably, and buying the meat. As introduced previously, understanding if the meat choice is environmentally sustainable, is usually not an easy task to solve. As a negative effect, it is possible, that the customer gives up and buys random meat or no meat, without satisfaction of the actual needs.

2.2 Proposed Solution

As explained previously, the main problem is the *behind-the-scenes* business of the meat industry. To target this problem, this project takes advantage of Crowdfarming[1] and Crowdbutching[2]. Crowdfarming is a way to reduce a supply chain to its essential parts. The overall goal is to eliminate intermediaries within the supply chain like vendors and supermarkets and provide a way where a consumer can directly buy from the producer. This way, a producer has more control over the price, no food waste, promotes more sustainable agriculture, secures better prices, and better social conditions.
Crowdbutching is a combination of Crowdfunding and Crowdfarming. The idea is, that a group of people buy a whole animal, and the animal is only processed after it has been sold completely. This means, that all usable parts of the animal have been sold. This way, the producer is allowed to grow a crop because it knows someone will consume it and therefore

[1] `https://www.crowdfarming.com/en/manifesto`
[2] `https://www.grutto.com/uk/crowdbutching-or-crowdbutchering/`

has no food waste.

To implement this solution, this project makes use of Blockchain technology, which allows to provide for more transparency for the consumer. The overall advantages of using a Blockchain are the following:

- Peer-to-peer payments without any intermediary
- Secured by cryptography
- Open to anyone

First, using a Blockchain allows the consumer to directly pay the producer, instead of any vendor. This allows the producer to have more control over fair prices for himself and all other entities involved. Secondly, a Blockchain takes advantage of cryptography to secure transactions. Once a transaction is performed and mined, it cannot be changed. Additionally, the data is open to anyone. This forces everybody involved in the supply chain to be more responsible, as the consumer can view the data. Additionally, it increases the transparency of the process and guarantees non-repudiation, which means that once a transaction has been made, it cannot be denied.

Furthermore, a dApp is implemented that contains smart contracts, which automatically execute the conditions of Crowdfarming and Crowdbutching. Additionally, it provides more transparency to the consumer, because the consumer can have insights into the process over a website. The consumer can view detailed information about the Crowdfunding of an animal and decide if it complies with the needs according to quality and environmental sustainability. After a campaign was funded successfully, the consumer is allowed to observe the supply chain via the website. Furthermore, this will provide for more trust between the consumer and the producer.

A problem is, how producers can provide proof that they deliver high-quality and sustainable processed meat. At first, when creating a new Crowdfunding campaign, the owner has to provide information about the animal, the farm, and the involved stakeholders (introduced in Section 2.4). The information about the stakeholders can involve the company name, an address, and a website link. Then, the consumer can research if these farmers and companies align with personal preferences according to sustainability. Furthermore, the farmer has to provide the ear-tag of the animal, an example is shown in Figure 1. The ear-tag uniquely identifies cattle, pigs, and sheep and is required by law in the European Union. Furthermore, bovine animals also require a passport that includes the ear-tag as identification [3, 4].

With the usage of a Blockchain, all previously mentioned information is stored on the Blockchain as transactions and cannot be changed. Each transaction will be digitally signed by one of the stakeholders (see Section 2.4 for further information about which stakeholder is responsible for what part), which provides, as mentioned previously, proof and non-repudiation. This means the stakeholder who has carried out the transaction cannot deny having made the transaction.

One problem remains, which involves the fact, that if a transaction on the Blockchain has been attached, it does not mean that the included information is true. Additionally, no legal binding between all involved entities exists. This problem is further explained at Section 5.2.

Figure 1: Ear-tag of a cattle in the netherlands

2.3 Market

As introduced previously in Section 2.1, the main target audience for this application are people who prefer to buy high-quality transparently produced meat with a positive impact on the environment. The 18th annual Power of Meat report released by the Meat Institute[3] and The Food Industry Association (FMI)[4] provides valuable statics about the north American meat market. It is stated that 71% of shoppers believe meat belongs in the daily diet, 78% of consider themselves as a meat eater, and 7% as vegan or vegetarian. More than half, 83% of meat eaters consider at least one *better-for* attribute when buying meat (*better for me/my family/animals/planet/farmers/workers*). Furthermore, 63% prefer private brands and manufacturers for fresh meat and 70% for processed meat[5].
When it comes to the European Market, sustainability will become a more prominent role for both producer and consumer in the meat market, as more modern equipment will lead to an environment-friendly meat production[6].

2.3.1 Competitors

To the knowledge of the authors, there is no direct competitor to this project that provides the same features and targets the same market. This means, a decentralized application including a frontend application, and a crowdfunding and supply chain smart contract.

IBM Food Trust[5] is considered an indirect competitor because it targets to solve a similar problem. It is a supply chain management application for the food industry based on Blockchain, in more detail. The main goal of IBM Food Trust is to increase the transparency of the supply chain for all kinds of food, from the producer to the consumer. The difference between IBM Food Trust and this project is, that IBM Food Trust directly targets vendors as their main consumers. For example, Walmart uses IBM Food Trust, to provide more

[3] `https://www.meatinstitute.org/`
[4] `https://www.fmi.org/`
[5] `https://www.ibm.com/products/supply-chain-intelligence-suite/food-trust`

transparency to their customers about the origin of their food[6].

2.4 Users

The target users of this dApp can be categorized as meat consumers and stakeholders.

2.4.1 Meat Consumers

As mentioned previously, meat consumers are the targeted users for this project. They interact through the website (frontend application) with smart contracts. Interaction involves searching for a campaign and buying a box of meat using cryptocurrency (Ether).

The typical meat consumer is any person who prefers to eat high-quality meat products. To see if meat is of high quality, a consumer prefers meat products with a highly transparent supply chain.

2.4.2 Stakeholders

Stakeholders are all users involved in supply chain management. The term stakeholders derives from the fact, that a stakeholder holds a potential stake after a campaign was funded successfully. Each stakeholder is being paid its share, after the supply chain has finished. The stakeholders include the campaign owner, the farmer, the butcher, and the delivery service.

Campaign Owner The campaign owner starts a campaign and manages it after it was funded successfully. This means a campaign owner has to provide the physical addresses of the consumers to the delivery service. It is responsible to define the stakeholders, of a campaign and their share of the total collected amount. Additionally, the campaign owner defines the meat boxes a consumer is allowed to buy. Furthermore, the owner is allowed to stop a campaign at any time, if the campaign has not been funded.

Farmer The farmer owns and takes care of the animal. It is very likely, that the farmer takes over the role of the campaign owner as well. After a campaign was funded successfully, the farmer is responsible to deliver the animal to the butcher, for further processing

Butcher The responsibility of a butcher is to process the animal after it was handed over by the farmer. Additionally, the butcher prepares the meat boxes defined by the campaign owner. Finally, the butcher will provide the boxes to the delivery services.

Delivery Service A delivery service is a third-party delivery company, that delivers all meat boxes to the consumers.

2.5 Monetization

On a Blockchain assets are bought using cryptocurrencies. This project is based on Ethereum, and therefore uses Ether[7]. As mentioned previously, consumers are allowed to buy boxes of meat using Ether. After a campaign was funded successfully, and the corresponding supply chain has finished, all stakeholders will receive their share of the total collected amount of Ether of the campaign (see Section 3 for more details).

[6] `https://pixelplex.io/blog/walmart-strives-for-food-safety-using-blockchain/`
[7] `https://ethereum.org/en/eth/`

3 Conceptual Design

This section explains the conceptual design of the decentralized application. It consists of two smart contracts (see Section 3.1) and a frontend application (see Section 3.2).

3.1 Smart Contract

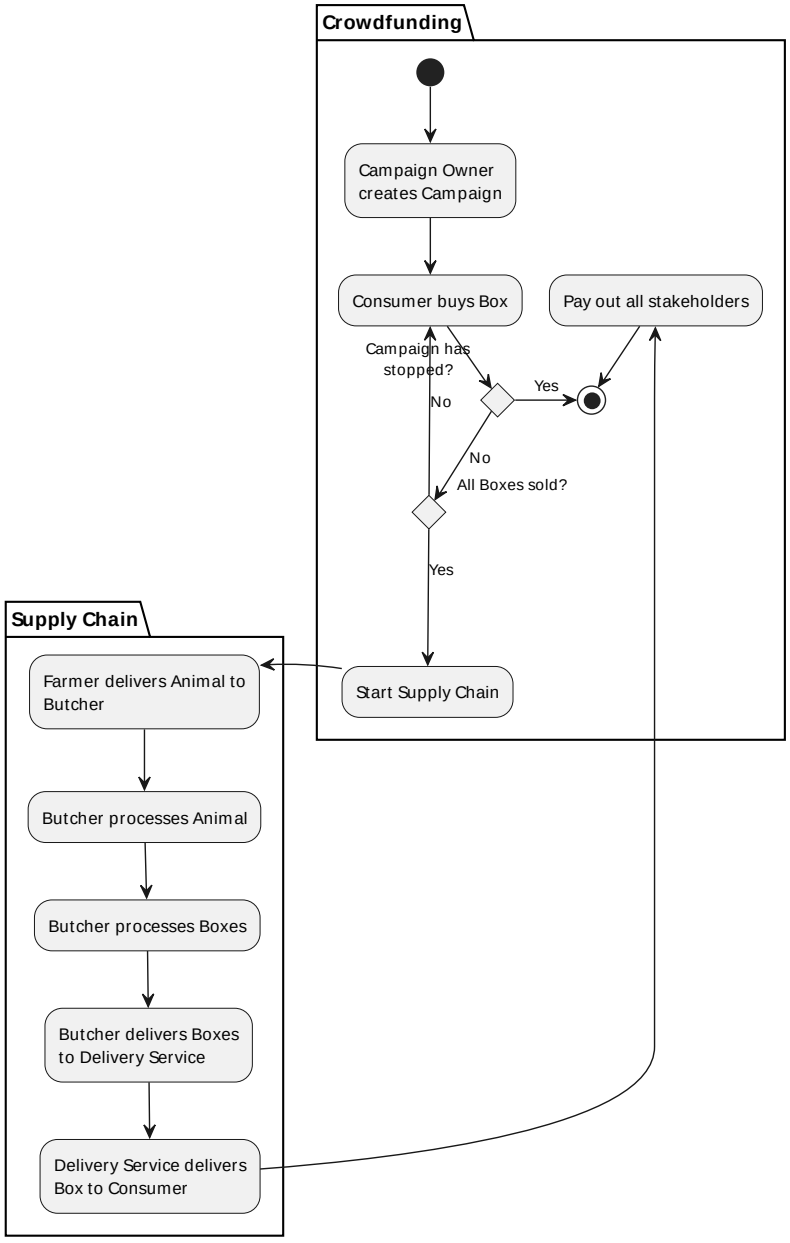

Figure 2: Overall activity diagram for the interaction with the Smart Contracts

Figure 2 shows the activity diagram, which illustrates the interaction between the users (introduced in Section 2.4) and the smart contracts. The overall process is divided into two sub-processes, the crowdfunding, and the supply chain part.

First, a campaign has to be created by a campaign owner. Then, consumers can buy boxes as long as the campaign is still active and boxes are available. As long as the campaign is still active, the campaign owner is allowed to stop a campaign for any arbitrary reason, which is further explained in Section 3.1.1. If all boxes are sold, the campaign will

automatically stop and start a supply chain for the campaign.

Next, the first part of the supply chain involves the farmer delivering the animal to the butcher. Then, the butcher has to prepare the animal and additionally prepare the boxes. Afterward, the butcher has to deliver the boxes to the delivery service, which is lastly responsible to deliver the boxes to the consumers. By delivering all boxes to the consumers, the supply chain part ends. Lastly, the crowdfunding smart contract is responsible to pay out all stakeholders.

3.1.1 Crowdfunding

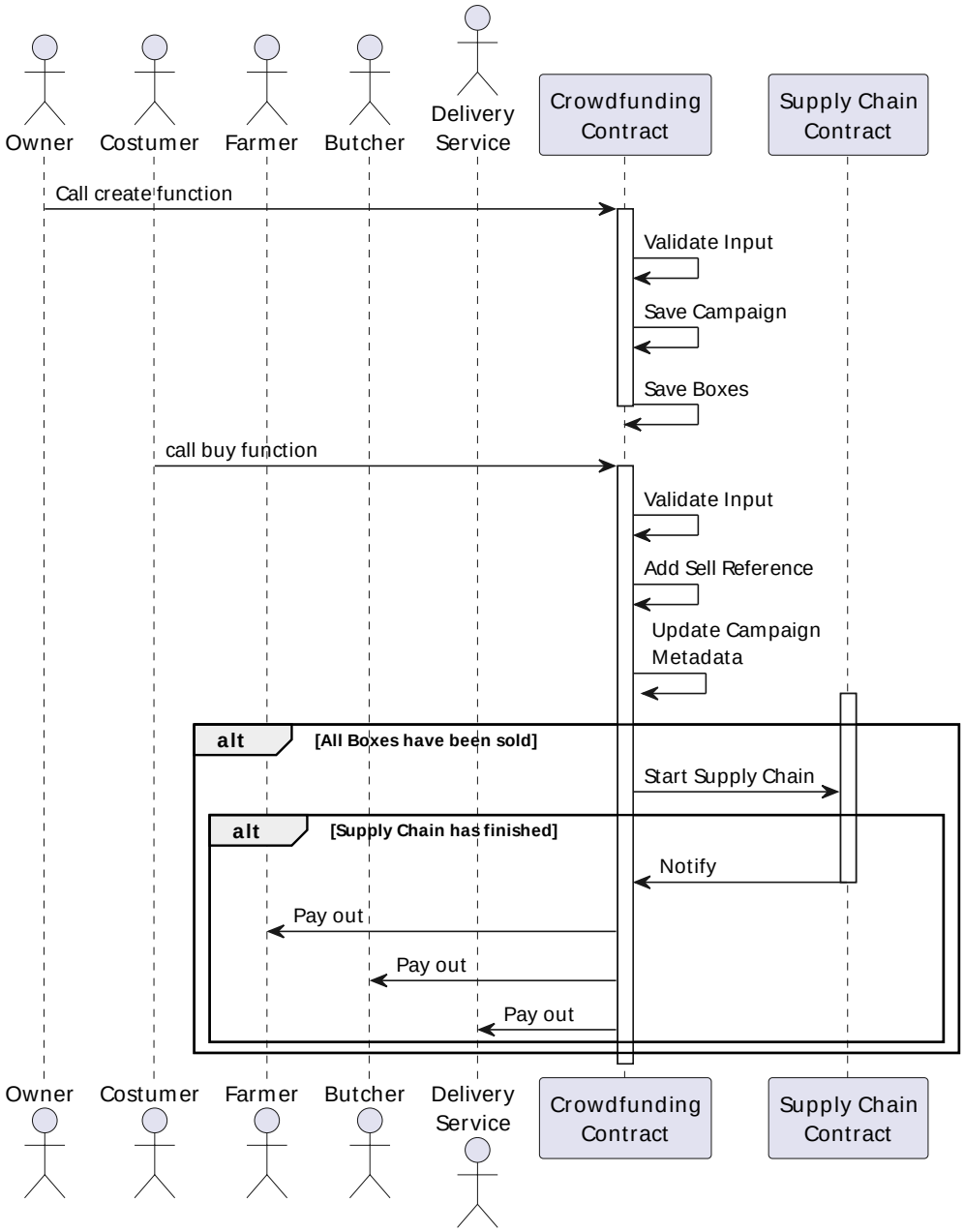

Figure 3: Crowdfunding sequence diagram

Figure 3 illustrates the sequence diagram of the crowdfunding smart contract. It involves all users introduced in Section 3.1.1, and the supply chain smart contract.

First, a campaign owner has to create a new campaign by calling the corresponding function of the Smart Contract. Then, the Smart Contract needs to validate the given data, save a campaign instance, and create the available box instances. Next, a consumer can buy a box by calling the *buy function* of the Smart Contract. Then, the input has to be validated (e.g. *is the box still available?*, *is the given amount of Ether enough?*), a reference of the sell has to be saved, and the campaign metadata needs to be updated. If all boxes have been sold, the crowdfunding Smart Contract will call the supply chain smart contract to start a new supply chain for the funded campaign. Lastly, after the supply chain smart contract has notified the crowdfunding Smart Contract (see Section 3.1.2 for details), it will start to pay each stakeholder its share over the overall collected amount.

3.1.2 Supply Chain

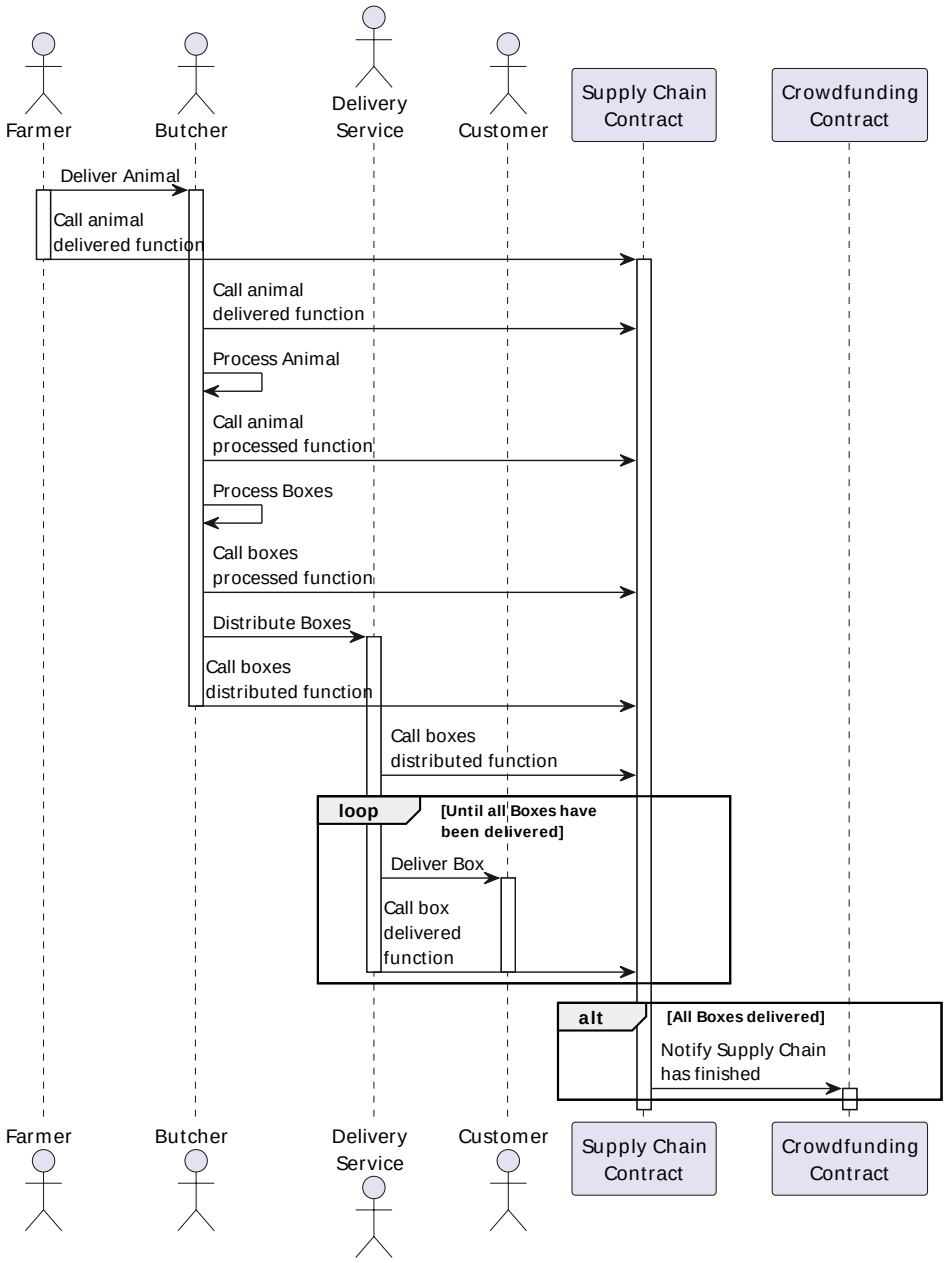

Figure 4: Supply chain sequence diagram

The sequence diagram of the supply chain is illustrated in Figure 4. It involves the same

entities as the crowdfunding sequence diagram, introduced in Section 3.1.1.

As mentioned in Section 3.1.1, the supply chain is started by the crowdfunding smart contract after all boxes have been sold. Then, the farmer is responsible to deliver the animal to the butcher. Afterward, both the farmer and butcher need to confirm the delivery of the animal by calling a designated function of the smart contract. This approach is inspired by the separation of duties concept[8] and is required to increase trust about the need that every stakeholder fulfills their duty. This problem is further explained at Section 5.2. Next, the butcher is required to process the animal and process the boxes. After each step, the butcher calls the corresponding functions offered by the smart contract, to update the state of the supply chain. After the butcher has fulfilled all steps, the boxes have to be delivered to the delivery service. This step has to be confirmed by the butcher and the delivery service respectively. After the delivery service has received the boxes, it is responsible to deliver the boxes to the consumer. It has to confirm the delivery of each box. After all boxes have been delivered, the crowdfunding smart contract gets notified about the status of the supply chain.

3.2 Frontend Application

The frontend application acts as the user interface for all users introduced in Section 2.4 to interact with the smart contracts introduced previously.

It provides the following functions:

- Login via Metamask
- Create a new campaign through a web form
- See all active and inactive campaigns
- See the details and status of a campaign through a campaign detail-page
- Buy a box over the campaign detail-page
- See the status of a sold box

3.2.1 Create Campaign Form

A campaign can be created through a form that provides a campaign owner to provide all the necessary details. This involves the information about a campaign (name, description, deadline), information about the animal (name, age, farm, ear-tag), information about all stakeholders (EOA address, company information), and the boxes (name, description, price in WEI).

3.2.2 Campaign Detail-Page

The detail page of a campaign contains information about the animal, information about the stakeholders, information about the campaign, and the status of the campaign supply chain. The supply chain status is only visible if the campaign was funded successfully. Additionally, interaction elements (e.g. buttons) are shown, to interact with the supply chain smart contract to update the status respectively. Additionally, it lists all available boxes and provides interaction elements to buy a box. Furthermore, if available, it shows all sold boxes with a link to a sell detail page.

[8] `https://en.wikipedia.org/wiki/Separation_of_duties`

3.2.3 Sold Box Detail-Page

Through the detail page of a sold box, a buyer can observe its delivery status. Additionally, the delivery service can update the delivery status through an interaction element. Furthermore, the campaign is allowed to decrypt the physical address of the buyer through a form.

4 Implementation

This section explains the details of the smart contract and frontend application implementation. The source code is available at GitHub via `https://github.com/mstolin/146157_Blockchain`.

4.1 License

To keep this work free, it is licensed under the terms of the GNU Affero General Public License[9] (AGPL). Due to its copyleft, this license forces all derivative works to be published under the same terms. It has been chosen because AGPL also considers network access as a form of distribution, which applies to both smart contracts and the frontend application.

4.2 Blockchain Test Environment

This project relies on the Truffle Suite[10] for the development of smart contracts.
Ganache[11] is used to simulate a Blockchain, in more detail on the ganache-cli. The directory `ganache/` contains two scripts: `start-ganache.sh` and `start-ganache-test.sh`. The first script starts Ganache with a fixed mnemonic and a path to store permanent data to create a simulated and persistent Blockchain on the host. This is required since this Ganache instance is used for the development of the frontend application instead for testing the smart contracts. The second script will start a non-permanent Ganache instance, used to test the smart contracts.

Furthermore, Truffle is used to compile, test, and deploy smart contracts. The directory `smart_contracts/` contains the source code, migration scripts, and test suites, as well as the generated Application Binary Interface (ABI).

4.3 Development Environment

A Development Container[12] is used for the development of this project, which guarantees the portability of the project across different hosts and additionally simplifies the setup of the development environment. Furthermore, the Development Container relies on VSCode[13] to install several plugins to support the Truffle Suite features, as introduced in Section 4.2.

4.4 Crowdfunding Smart Contract

The crowdfunding smart contract implements the crowdfunding concept introduced in Section 3.1.1.

Figure 5 demonstrates the UML class diagram of the Crowdfunding smart contract. To reduce to costs of gas as much as possible, the contract only implements the most basic features. For example, the contract does not offer any function to retrieve a single campaign or a stopped campaign. It implements a function `getCampaigns` that returns all campaigns instead, as it is the responsibility of the frontend application (introduced at Section 4.6) to implement further business logic.

4.4.1 Creating a Campaign

New Crowdfunding campaigns are created using the `createCampaign` function. It takes a title, description, duration, information about the campaign owner, information about the stakeholders, information about the animal, and a list of available boxes as input

[9] `https://www.gnu.org/licenses/agpl-3.0.en.html`
[10] `https://trufflesuite.com/`
[11] `https://trufflesuite.com/ganache/`
[12] `https://containers.dev/`
[13] `https://code.visualstudio.com/`

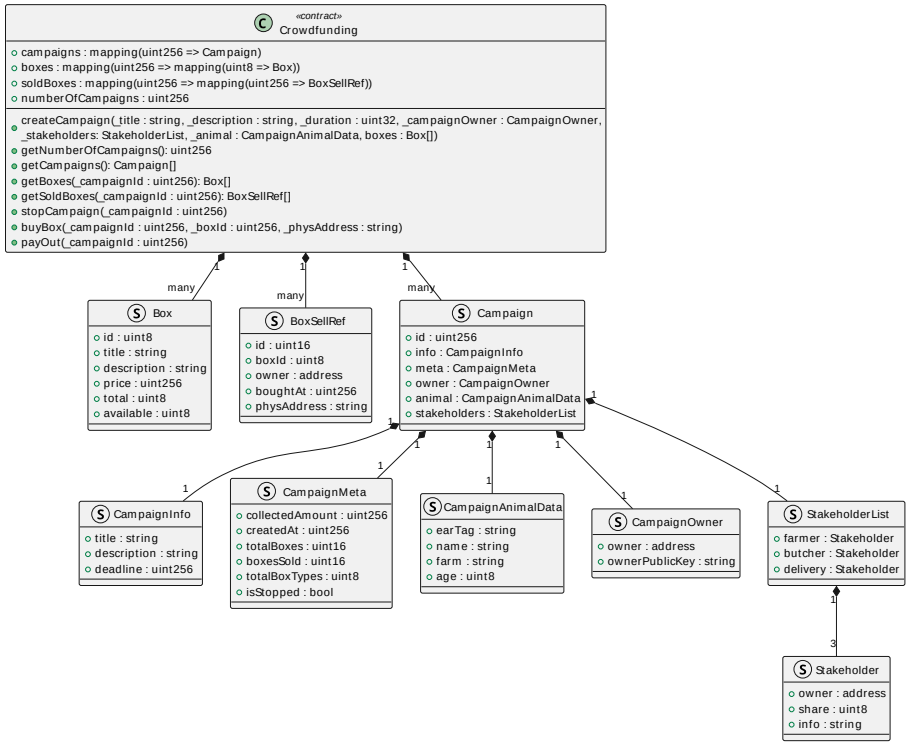

Figure 5: Crowdfunding smart contract class diagram

parameters. After the function is called, all parameters will be validated first, to make sure no false data will be stored on the Blockchain. Campaigns are represented by the `Campaign` struct. A `mapping(uint256 => Campaign)` called `campaigns` is used to store campaigns on storage. The `uint256` type is used, to allow $2^{256} - 1$ campaigns to be stored. One problem is, looping through a mapping isn't easily possible in Solidity, because it does not provide a way to retrieve the length of such. To solve this problem, the smart contract saves the total number of campaigns in a property called `numberOfCampaigns`, which can be used as the upper threshold to loop through the `campaigns` mapping.

When a campaign is created, the corresponding boxes are created as well. Boxes are represented by the `Box` struct and saved as storage in a `mapping(uint256 => mapping(uint8 => Box))` called `boxes`. The first `uint256` represents the key of the parent campaign and `uint8` is the id of the box. The usage of `uint8` allows each campaign to contain $2^8 - 1 = 255$ different types of boxes. Additionally, only 255 pieces of a single box can be available as well. To allow looping through the mapping of boxes (see Section 4.4.4 for an example), a campaign saves the number of different box types in a property called `totalBoxTypes` as metadata. Respectively, an additional property called `boxesSold` is saved to allow looping through the mapping of sold boxes (see Section 4.4.4).

4.4.2 Retrieving Campaigns

To retrieve a campaign, the dApp has to call the `getCampaigns` function. It returns all campaigns as an Array of `Campaign` structs, regardless if they are active or not. Campaigns are stored as storage in a mapping and have to be converted to memory before returning them.

4.4.3 Stopping a Campaign

As mentioned in Section 2.4.2, the campaign owner is allowed to stop a campaign, by calling the `stopCampaign` function. First, the function checks if the sender address is equal

to the address of the campaign owner (saved as metadata), to prevent unauthorized calls from stopping a campaign. Next, the `isStopped` metadata property of a campaign is set to `true`. Lastly, the smart contract automatically sends all Ether back to previous buyers of a box.

4.4.4 Retrieving Boxes

The `getBoxes` function is used to retrieve all box types of a specific campaign from the smart contract. It takes the `campaignId` as an argument to identify the targeted campaign. Then, the boxes have to be converted from storage to memory as well, to return them (as mentioned in Section 4.4.2). A `Box` struct includes a property called `available`, the number of available boxes of its kind, that can be used in a frontend application to check if this box is still available.

Sold boxes are saved as `BoxSellRef` structures in a different mapping called `soldBoxes` with the type `mapping(uint256 => mapping(uint256 => BoxSellRef))`. The first `uint256` represents the parent campaign id, and the second `uint256` represents the id of the sell. This allows to store $2^{256} - 1$ references to box sells. A sell-reference can be received using the `getSoldBoxes` function, which requires the campaign ID as a parameter. Furthermore, before returning them, sell references need to be converted from storage to memory as well.

4.4.5 Buying a Box

A box can be bought by calling the `buyBox` function. It is only allowed to buy one box at a time. It takes the ID of the campaign, the ID of the box, and the already encrypted private address of the buyer as parameters. The address must be encrypted before, as it cannot be encrypted using the smart contract. This would result, that the clear address would be available in the transaction data and therefore readable to everyone. The encryption process is further explained at Section 4.6.3, and a problem statement of this approach is given at Section 5.1.

First, the function checks if the campaign is still active. Next, it has to verify that the box is still available and that the sent amount of Ether is enough to buy the box. If the verification succeeds, a `BoxSellRef` including important metadata is added to the `soldBoxes` mapping. Lastly, the metadata of the campaign and the box is updated as well (decrease the number of available boxes, increase the number of sold boxes, and increase the collected amount).

4.4.6 Pay out all Stakeholders

The `payOut` function will pay out the share of each stakeholder over the overall collected amount after the supply chain has been completed. To prevent any unauthorized calls, this function can only be called by the supply chain smart contract.

4.5 Supply Chain Smart Contract

The supply chain smart contract implements the supply chain concept introduced in Section 3.1.2.

Figure 6 demonstrates the UML class diagram of the Supply Chains smart contract. Again, to reduce the costs of gas as much as possible, the contract only implements the most basic features. For example, the contract does not offer any function to retrieve a single supply chain or a single box status. It implements a function called `getSupplyChains` which returns all supply chains instead. The frontend application is responsible to implement further business logic.

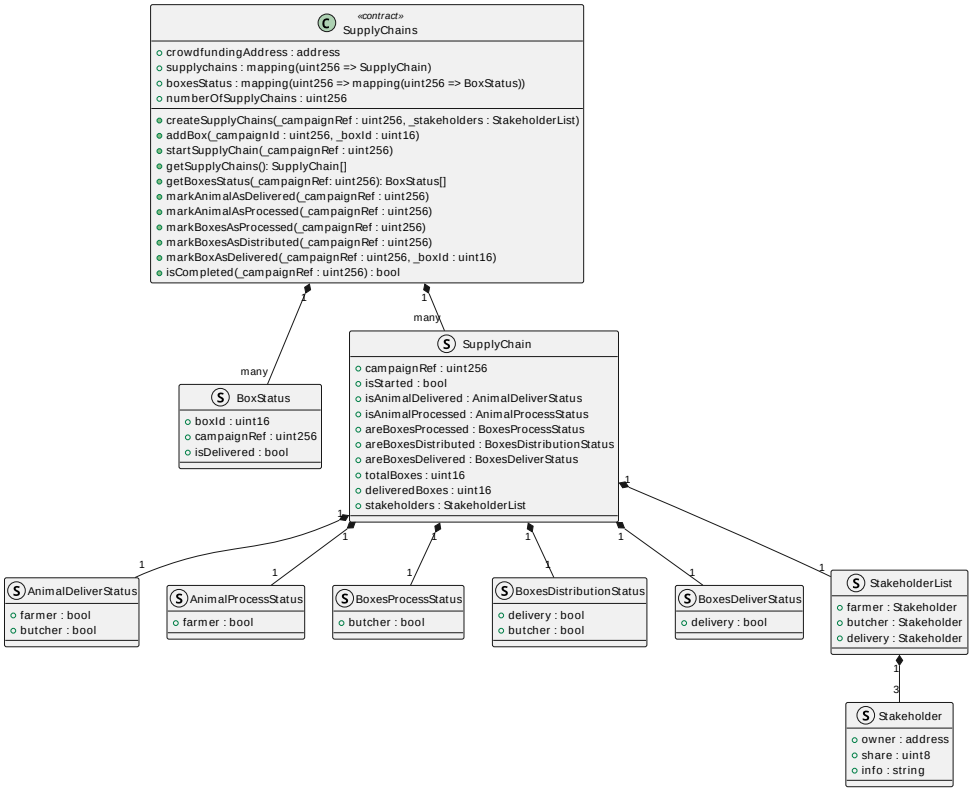

Figure 6: Supply chain smart contract class diagram

4.5.1 Creating a Supply Chain

New supply chains are created using the `createSupplyChain` function. To prevent unauthorized calls, this function is external and can only be called by the crowdfunding smart contract. The function only request necessary data, such as the campaign identifier (as a reference to the campaign) and the stakeholder list (for validation purposes through the supply chain process). Supply chains are represented by the `SupplyChain` struct and use a `mapping(uint256 => SupplyChain)` to store supply chains on storage, equally to the crowdfunding smart contract as introduced at Section 4.4.1. A property called `numberOfSupplyChains` is required, to allow looping through the `supplychains` mapping.

Once a campaign is created, the crowdfunding contract calls the `createSupplyChain` function. Since the supply chain requires sold boxes, no data about boxes is stored during this phase. After a box is bought and the respective `BoxSellRef` is created, the crowdfunding contract calls the `addBox` function, specifying the box identifier. Boxes statuses are represented by the `BoxStatus` struct and saved as storage in a `mapping(uint256 => mapping(uint16 => BoxStatus))` called `boxesStatus`. The first `uint256` represents the key of the parent campaign and `uint16` the ID of the sold box. To allow looping through the mapping of box statuses, the supply chain saves the number of total boxes in a property called `totalBoxes`. Finally, when all boxes are sold and the campaign is stopped, the crowdfunding contract calls the `startSupplyChain` function, that saves the state of the supply chain in a property called `isStarted`.

4.5.2 Retrieving Supply Chain Status

To retrieve the status of a supply chain, the dApp has to call the `getSupplyChains` function, which returns all supply chains as an array of `SupplyChain` structs. Supply chains are stored as storage in a mapping and have to be converted to memory before returning them.

Once a supply chain is retrieved, the `SupplyChain` struct provides several properties to keep track of the progress and determine the actual status: `isAnimalDelivered` that determines if the animal is delivered or not, `isAnimalProcessed` that determines if the animal is processed or not, `areBoxesProcessed` that determines if the boxes are processed or not, `areBoxesDistributed` that determines if the boxes are distributed or not, and finally `areBoxesDelivered` that determines if all the boxes are delivered or not. Moreover, a function called `isCompleted` is provided to let the crowdfunding smart contract know if the supply chain is actually completed or not. Thus, the function is external.

4.5.3 Retrieving a Box Status

To retrieve the status of a box, the dApp has to call the `getBoxesStatus` function, which returns all the box statuses as an array of `BoxStatus` structs. Box statuses are stored as storage in a mapping and have to be converted to memory before returning them.
As mentioned in Section 3.1.2, a box can be delivered regardless from the others. Once a box status is retrieved, the `BoxStatus` struct provides a property called `isDelivered` that determines, if a certain box is delivered or not.

4.5.4 Update a Supply Chain

As shown in Figure 4, many steps are required to complete the supply chain. Thus the smart contract provides several functions for each step.
Initially, once the supply chain is started, the animal can be marked as delivered by calling the `markAnimalAsDelivered` function. To prevent any unauthorized call, the function can only be called once by the farmer and the butcher. Then the animal can be marked as processed by calling the `markAnimalAsProcessed` function and the boxes can be marked as processed by calling the `markBoxesAsProcessed` function. These functions can only be called once by the butcher. Afterward the boxes can be marked as distributed by calling the `markBoxesAsDistributed` function. The function can only be called once by the butcher and the delivery service. Finally, each box can be marked as delivered by calling the `markBoxAsDelivered` function. The function can only be called one time for each box by the delivery service. Every function requires that the previous step is correctly done. When the last box to deliver is marked as delivered, the supply chain contract automatically calls the crowdfunding contract to perform the payment towards each stakeholder, with the `payOut` function (introduced at Section 4.4.6).

4.6 Frontend Application

As described at Section 3.2, the frontend application serves as the interface for users to interact with smart contracts. It is implemented using *Angular*[14] and *Bootstrap*[15]. To interact with the smart contracts, it uses the *Web3.js*[16] framework.

4.6.1 Crowdfunding Service

Angular uses the concept of services to interact with third-party services/data. Therefore, a service called `CrowdfundingService` was implemented to interact with the crowdfunding smart contract. It implements all publicly available methods introduced in Section 4.4 and additional business logic. For example, the crowdfunding service also implements a method to receive single campaigns by ID or a list of stopped campaigns.

4.6.2 Supply Chain Service

A service called `SupplyChainService` was implemented to interact with the supply chain smart contract, that implements all the public methods introduced in Section 4.5 and

[14] `https://angular.io/`
[15] `https://getbootstrap.com/`
[16] `https://web3js.org/`

additional business logic. For example, the supply chain service implements methods to receive single supply chains or single box statuses by ID. Moreover, the service also implements simpler but useful methods, for example to determine if an animal is delivered by both stakeholders.

4.6.3 Encryption and Decryption of private Addresses

As mentioned in Section 4.4.5, consumers are required to provide their physical address to the campaign owner for the delivery of the meat boxes. The problem is, once a transaction is performed, the physical address is written to the Blockchain as part of the transaction data, and publicly available to everyone. To mitigate this issue, the address needs to be encrypted in a way, that only the campaign owner is allowed to read it.

An advantage of Blockchain is, it takes significant usage of public key cryptography (PKC). With PKC it is possible to provide for confidentiality by encrypting a message with the public key of the receiver and the receiver can decrypt the message with its private key. This guarantees that only the owner of a private key to the corresponding public key can read the sensible information.

Metamask provides an API to receive the public key of an account as well as for encryption and decryption. Whenever a campaign is created, Metamask is used to get the public key of the campaign owner, which is then saved to the campaign data on the Blockchain. Later, when a consumer buys a box, the campaign owner's public key is fetched and used to encrypt the address before calling the smart contract function. This way, it is guaranteed that the address in the transaction data is already encrypted. Afterward, the campaign owner can view the *sell data* of a box and decrypt the address with its private key using Metamask as well.

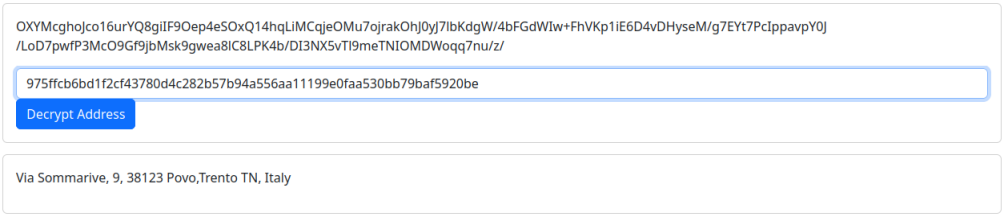

Figure 7: UI of the frontend showing the process of encrypting a consumers physical address

Figure 7 illustrates how the frontend application presents the encrypted and decrypted address of a consumer. On the top, the encrypted address is shown with an additional input field for the private key. Whenever the campaign owner provides its private key and presses *Decrypt Address*, Metamask will pop up, decrypt the address, and show the true address instead, as shown on the bottom.

5 Problems

This section provides insights about the encountered problems during the implementation of the proposed project, as well as possible solution statements.

5.1 Encryption of private Information

As introduced in Section 4.6.3, the frontend application takes use of Metamask encryption features to hide the private address of a consumer against unauthorized access. The reason is, that data stored on the Ethereum Blockchain is readable by everyone. The proposed solution is based on public key cryptography, where the address is encrypted using the Campaign owner's public key before being stored on the Blockchain. This results, that the address is only readable by the campaign owner.

Figure 8: Warning showing the deprecation of encryption features

This solution works to hide the address, however, the encryption features provided by Metamask are deprecated and not supported in future versions[17]. Figure 8 presents the warning, shown in the debugger console of Firefox. This problem does not affect the smart contract implementation, because the smart contract only saves the public key of the campaign owner and is not responsible for encryption. However, the frontend application relies on Metamask and a cumbersome solution will be, that the frontend application has to be extended in a way to provide the public key manually. Additionally, another library than *@metamask/eth-sig-util*[18] must be used for encryption and decryption.

Decentralized Identity[19] is another possible solution for this problem. Then, the consumer saves their personal information (in this case the address) securely using Blockchain technology and can control the flow of their personal information.

5.2 Trustworthiness of provided Information

As mentioned in Section 2.2, the presence of a transaction does not prove that the included data is true. It just proves that someone has performed a transaction on the Blockchain. Regarding this project, the problem is that every stakeholder (introduced in Section 2.4.2) can act falsely. The farmer can lie about the animal data, the butcher can lie about the processing of the animal and the boxes, and the delivery service can lie about delivering the boxes to the consumer. In general, there is no solution like a unified database (e.g. for all member states of the EU), where farmers are forced by law to provide information about livestock animals, identifiable by ear-tag.

A possible solution was introduced in Section 3.1.2, based on the separation of duties concept. It involves that stakeholders verify other stakeholders in the process chain, to make sure everybody gets paid at the end. However, this does not avoid that all stakeholders act falsely together as a group, or that all stakeholders are controlled by a single unit that acts falsely.

The overall problem is, there is no legal binding between the consumer and the campaign owner, as well as between the campaign owner and all stakeholders. Smart contracts are

[17] `https://medium.com/metamask/metamask-api-method-deprecation-2b0564a84686`
[18] `https://www.npmjs.com/package/@metamask/eth-sig-util`
[19] `https://ethereum.org/en/decentralized-identity/`

not legally binding which results that there is no legal negative effect of stakeholders acting falsely. A possible solution is the usage of a legal smart contract between all entities, to make sure all stakeholders fulfill their duty. An additional solution is, to issue a Ricardian Contract to the consumer, to confirm the legal ownership of a box.

Another solution is to take advantage of zero-knowledge proofs[20] (ZK proofs). In general, ZK proofs are used to provide higher privacy by proving that a statement is true, without revealing the information of the statement itself. However, a ZK proof protocol still requires a witness to verify that a statement is true by providing the information to the Prover. For example, this would force the delivery service to prove a box has been delivered to a consumer, e.g. through a signature.

[20] `https://ethereum.org/en/zero-knowledge-proofs/`

6 Conclusion

To conclude, this project provides an application to buy meat directly from the producer. It includes a dApp that offers crowdfunding campaigns and supply chains. To enable decentralization, it is based on the Ethereum Blockchain, which additionally increases trust between the consumer and the producer. Furthermore, it includes crowdfunding and a supply chain smart contract. The crowdfunding smart contract allows the consumer to buy high-quality products directly from the producer and the supply chain contract allows for more transparency and control between the consumer and the involved stakeholders. Additionally, a mechanism based on public key cryptography to securely save the physical address of a consumer on the Ethereum Blockchain was introduced.

References

[1] M. Marchesi, L. Marchesi, and R. Tonelli, "An agile software engineering method to design blockchain applications," in *Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia*, ser. CEE-SECR '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3290621.3290627

[2] "2019 food & health survey," accessed: 02/08/2023. [Online]. Available: https://foodinsight.org/wp-content/uploads/2019/05/IFIC-Foundation-2019-Food-and-Health-Report-FINAL.pdf

[3] "Identification and registration of certain kept terrestrial animals," accessed: 31/07/2023. [Online]. Available: https://food.ec.europa.eu/animals/identification_en

[4] S. Pavon, "Animal identification in the european union," accessed: 31/07/2023. [Online]. Available: https://www.icar.org/wp-content/uploads/2015/09/Pavon-AI-in-EU_EN1.pdf

[5] "18th annual power of meat report: meat purchases rise above pre-pandemic levels, consumers look for value," accessed: 31/07/2023. [Online]. Available: https://meatinstitute.org/press/18th-annual-power-meat-report-meat-purchases-rise-above-pre-pandemic-levels-consumers-look

[6] "Eu agricultural outlook 2021-31: consumer behaviour to influence meat and dairy markets," accessed: 31/07/2023. [Online]. Available: https://agriculture.ec.europa.eu/news/eu-agricultural-outlook-2021-31-consumer-behaviour-influence-meat-and-dairy-markets-2021-12-09_en

Appendix

A Value Proposition Canvas