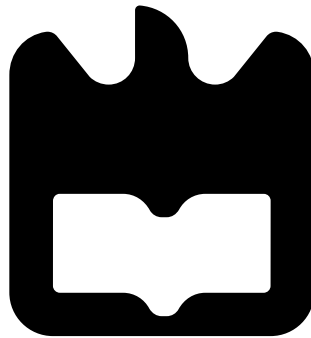




**Adriano  
Oliveira**

**Utilização de tecnologia Blockchain na  
contratualização de serviços de pastorícia  
Using Blockchain to manage contracts for land  
clearing with sheep**







**Adriano  
Oliveira**

**Utilização de tecnologia Blockchain na  
contratualização de serviços de pastorícia  
Using Blockchain to manage contracts for land  
clearing with sheep**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e telemática, realizada sob a orientação científica de Hélder Gomes e Pedro Gonçalves, professores da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro



**o júri / the jury**

presidente / president

**André Ventura da Cruz Marnoto Zúquete**

Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

**Filipe Alexandre Pais de Figueiredo Correia**

Professor auxiliar da Faculdade de Engenharia da Universidade do Porto

**Hélder José Rodrigues Gomes**

Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro



**agradecimentos /  
acknowledgements**

Quero agradecer a todas as pessoas que de alguma forma me ajudaram neste meu percurso. Nomeadamente aos meus orientadores, Hélder Gomes e Pedro Gonçalves pela sua compreensão e prestabilidade. À minha família que sempre me apoiou nos bons e maus momentos, a todos os meus amigos, principalmente ao Tiago Vieira e Manuel Alejandro com os quais partilhei momentos inesquecíveis durante todo o percurso académico. Ao Carlos Viana, e meus colegas da GoContact pela compreensão que tiveram comigo na fase final desta dissertação. À minha namorada Dora, que especialmente nesta fase nunca deixou de me apoiar e incentivar. Um obrigado também à associação de estudantes ESN por me ter dado a oportunidade de reinventar a minha experiência académica e assim criar grandes amizades.





## Resumo

Os serviços de limpeza de terreno estão rapidamente a ganhar popularidade, principalmente devido a requisitos legais impostos a donos de terrenos, que têm como objetivo a prevenção de eventuais fogos florestais. Estes serviços de limpeza são tradicionalmente feitos com recurso a maquinaria ou produtos químicos, o que acaba por ser um processo custoso e com impacto no ambiente. Uma solução alternativa é um processo que não é uma novidade por si só - o uso de animais de pastagem para a remoção das espécies vegetais indesejáveis. Esta é uma alternativa com um custo monetário e ambiental reduzido. Para aumentar a eficiência deste processo, o projeto SheepIT foi criado, e trouxe a possibilidade de exercer um melhor controlo sobre os animais através do uso de aparelhos eletrónicos, que simultaneamente permitem uma coleção de dados importantes. Esta dissertação acresce ao trabalho desenvolvido no projeto SheepIT, e tem como objetivo tornar-se uma ponte entre donos de terrenos que tencionam pagar por serviços de limpeza de terreno, e donos de rebanhos que atuam como prestadores desse serviço e estão à procura de clientes. O objetivo final é criar uma plataforma onde os utilizadores possam publicitar as suas intenções, contratualizar serviços de pastagem, e gerir os mesmos. A tecnologia blockchain é então usada para garantir a proveniência e integridade de dados importantes que passam pelo sistema, com o intuito de fomentar confiança entre os utilizadores da plataforma



## **Abstract**

Land clearing services are quickly spreading, mostly due to legal requirements imposed on field owners to force them to clean their terrains in order to prevent the spread of eventual wildfires. These land clearing services are traditionally made with the help of machinery or chemicals, which is costly and heavy on the environment. An alternative is a process which is not a novelty in itself - bringing grazing animals, usually sheep, to the terrain, and have them eat the unwanted weeds. This is a cheaper and more environmental-friendly way of conducting a land clearing service. In order to increase the efficiency of this process, the SheepIT project was created, which brought the possibility of exerting a better control over the animals with the use of smart devices, which also enable valuable data collection. This dissertation is built on top of the SheepIT project and aims at being the bridge between owners of fields who intend on paying for these services, and the owners of sheep acting as service providers and looking to find clients. The goal is to create a platform where they can publicize their intents, contractualize services and manage them. Blockchain technology was then used in order to guarantee the provenance and integrity of important data flowing through the system, in order to generate trust between all the users of the platform.



---

# Contents

---

<b>List of Acronyms</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.2 Objectives . . . . .	4
1.3 Document Structure . . . . .	4
<b>2 Related work</b>	<b>5</b>
2.1 Usage of animals for land clearing . . . . .	5
2.1.1 SheepIT . . . . .	6
2.1.2 Land clearing services . . . . .	9
Rent-a-Ruminant . . . . .	9
The Goat Lady . . . . .	10
2.2 Contract management platforms . . . . .	10
2.2.1 Airbnb . . . . .	11
2.3 Distributed Ledger Technology . . . . .	12
2.3.1 Blockchain . . . . .	12
2.3.2 Consensus Algorithms . . . . .	14
Proof of Work (PoW) . . . . .	15

Proof of Stake (PoS) . . . . .	16
2.3.3 Smart Contracts . . . . .	17
2.4 Platforms for blockchain application development . . . . .	19
2.4.1 Hyperledger . . . . .	19
Hyperledger Fabric . . . . .	20
Hyperledger Composer . . . . .	21
2.4.2 Ethereum . . . . .	24
2.4.3 Comparison between these platforms . . . . .	25
<b>3 Solution</b>	<b>27</b>
3.1 Requirements . . . . .	29
3.2 Stakeholders . . . . .	30
3.3 Data model . . . . .	31
3.4 Access control . . . . .	34
<b>4 Implementation</b>	<b>37</b>
4.1 Architectural choices . . . . .	37
4.1.1 Platform . . . . .	37
4.1.2 Blockchain configuration . . . . .	38
4.1.3 Hosting . . . . .	40
4.2 Building the prototype . . . . .	42
4.2.1 Creating the network . . . . .	42
4.2.2 Business logic . . . . .	54
<b>5 Results</b>	<b>59</b>
<b>6 Conclusions</b>	<b>83</b>
6.1 Future work . . . . .	85

---

## List of Figures

---

2.1	Sheep Carrying a Collar . . . . .	6
2.2	System architecture . . . . .	7
2.3	Location system operation . . . . .	8
2.4	Classification Rules . . . . .	9
2.5	Workflow of a Blockchain . . . . .	13
2.6	Differents kinds of accessibility for blockchain networks . . . . .	14
2.7	Countries with lower energy consumption than the Bitcoin network . . . . .	16
2.8	Architecture of HLC . . . . .	23
3.1	Data model . . . . .	33
4.1	Devised configuration of the blockchain network - (C-Channel, P-Peer, O-Orderer, CA-Certificate Authority, SO- Sheep Owner, FO- Field Owner) . . . . .	39
4.2	IBM Blockchain Platform pricing overview . . . . .	41
4.3	Business network successfully started . . . . .	52
4.4	Pinging the network . . . . .	52
4.5	REST endpoints for the Listing asset . . . . .	53
4.6	Web interface generated with Yeoman . . . . .	53
5.1	Network admin registering John . . . . .	60

5.2	Network admin registering Rick . . . . .	60
5.3	John was added to the Shepherd registry . . . . .	61
5.4	Rick was added to the Field owner registry . . . . .	61
5.5	Network admin issuing an ID to John . . . . .	62
5.6	Network admin issuing an ID to John . . . . .	62
5.7	Transaction that registers a gateway . . . . .	63
5.8	Gateway registered . . . . .	63
5.9	Network admin issuing an ID for the gateway . . . . .	64
5.10	John trying to issue a BNC for a gateway . . . . .	64
5.11	Available BNCs . . . . .	65
5.12	John registering an animal . . . . .	66
5.13	Animal instance created . . . . .	66
5.14	Rick registering a field . . . . .	67
5.15	Field instance created . . . . .	67
5.16	John submitting a CreateListing transaction . . . . .	68
5.17	John's listing available in the marketplace . . . . .	68
5.18	Rick sending a solicitation to John's Listing . . . . .	69
5.19	Solicitation available in the solicitation registry, only visible to the involved parties . . . . .	69
5.20	John accepting Rick's solicitation . . . . .	70
5.21	Land clearing contract created, visible only to the involved parties . . . . .	70
5.22	Transaction that signals the beginning of animal movement . . . . .	71
5.23	Transaction that signals the beginning of animal transportation . . . . .	71
5.24	Animal transportation instance created in the animalTransportation registry	72
5.25	animal movement entry added to the registry, in this case being viewed already by Regulator. . . . .	72
5.26	John invokes the initiate contract transaction . . . . .	73
5.27	Rick invokes the initiate contract transaction . . . . .	73
5.28	Land clearing contract status set to ONGOING . . . . .	73
5.29	Transaction that directs a gateway to a contract . . . . .	74
5.30	Gateway directed . . . . .	74



5.31	Animal enters field transaction, sent by gateway . . . . .	75
5.32	Pasture instance created in the pasture registry . . . . .	75
5.33	Animal leaves field transaction, sent by gateway . . . . .	76
5.34	Pasture instance finished . . . . .	76
5.35	Contract entry updated with the total time spent by the animal during the pasture . . . . .	77
5.36	John trying to tamper with the amountOfHoursWorked field . . . . .	77
5.37	Rick sends a finishContract transaction . . . . .	78
5.38	John sends a finishContract transaction . . . . .	78
5.39	Contract Finished . . . . .	78
5.40	John sends Rick his feedback about the finished contract . . . . .	79
5.41	Rick sends John his feedback about the finished contract . . . . .	79
5.42	John's profile updated with the feedback entry given by Rick . . . . .	80
5.43	Rick's profile updated with the feedback entry given by Rick . . . . .	80
5.44	John trying to send an AnimalEnterField transaction . . . . .	81
5.45	John trying to send an UpdateParticipant transaction . . . . .	81



---

## List of Acronyms

---

<b>ANOA</b>	Associação Nacional de Operadores Agrícolas	<b>HLC</b>	Hyperledger Composer
<b>ANOP</b>	Associação Nacional de Operadores de Pecuária	<b>HLF</b>	Hyperledger Fabric
<b>API</b>	Application Programming Interface	<b>JSON</b>	JavaScript Object Notation
<b>BNC</b>	Business Network Card	<b>PoW</b>	Proof of Work
<b>CA</b>	Certificate Authority	<b>PoS</b>	Proof of Stake
<b>DGAV</b>	Direção-Geral de Alimentação e Veterinária	<b>REST</b>	Representational State Transfer
		<b>RSSI</b>	Received Signal Strength Indicator



# CHAPTER 1

---

## Introduction

---

Nowadays, brush and weed control in fields is becoming a major concern, mainly due to the need to prevent fires from starting and spreading through forests, which cause enormous environmental and economic damage. To fight this, many countries have created legislation which require field owners to clean their terrains periodically, or risk incurring in a penalty, e.g. Portugal - (n.º 2 do artigo 15.º da Lei n.76/2017) [1].

However, this process is usually costly, since it is typically done by mechanical and chemical methods and needs to be repeated periodically. The mechanical approach involves using plows and cutters to remove herbs between plant rows, while the chemical method resorts to spreading herbicide between plant feet, in order to kill the weeds. Both methods are considered very aggressive for the cultures and environment, chemicals can even remain in the soil and contaminate water lines. These processes also entail a significant economic impact, with typical costs per hectare ranging from 80 €/ha up to 380 €/ha for agricultural terrains, while forest land can reach thousands of Euros per hectare.

An alternative is the use of animals for controlling weeds, which is an old method that has been proven successful in various regions, reducing the environmental and financial costs, while providing land fertilization at the same time. There have been recent experiments in Portugal to confirm the feasibility of this approach, such as “gestão de combustíveis florestais com recurso a cabras sapadoras” [2], where 30

sheep consumed around 11000kg of green matter over the course of 3 months in a 4000m<sup>2</sup> terrain in Santa Maria da Feira [2].

For this to be done efficiently though, there needs to be some sort of control over the animals, in order to ensure they stay in the appropriate area, and that they stay safe and do not cause unwanted damage.

Enter, the concept of Smart Farming, which represents the application of modern Information and Communication Technologies (ICT) into agriculture, “Smart Farming has a real potential to deliver a more productive and sustainable agricultural production, based on a more precise and resource-efficient approach” [3].

In particular, this dissertation is building up from the work developed in the SheepIT project, which had the goal of developing an IoT based system, able to control animal posture, and to deploy virtual fences to control animal feeding areas, making it possible to use herds of sheep for land clearing services [4].

Our vision is to create a marketplace to bring together sheep owners looking to provide land clearing in exchange for monetary compensation, with field owners in need of such services. They will be able to make use of our platform to look for opportunities and engage in land clearing contracts, which bind them to an agreement for the service to be held with the agreed conditions. To facilitate this, it is essential for both parties to trust each other since the service provider is interested in assuring the security of the animals, and that the payment will be made with the agreed conditions, while the owner of the field wants the field properly cleaned and his crops to remain secure. A rating functionality was integrated to incentivize users to deliver a good service and follow all rules, in order to build up a good reputation within our platform.

To complement this, some of the data flowing through our system benefits from having its integrity and provenance verifiable by the interested parties. To address this issue, blockchain technology proves to be quite promising, since it enables the various users in a network to trust the data present in it thanks to its inherent properties.

Blockchain was invented by Satoshi Nakamoto in 2008 to become the backbone of Bitcoin’s functionality [5], and quickly became more than that, as people started realizing this technology could be disruptive for many industries worldwide.

The first field where Blockchain irrupted is the financial sector. Thousands of cryptocurrencies were created since then and while a big chunk of them are not significant, the whole market capitalization, at the time of writing, is around 240 \$ Billion, [6] having reached its peak in early January 2018, at around 830 \$ Billion.

The potential of this technology stems from the fact that blockchain provides a way for business

partners to be able to trust each other, knowing that it safe to do so, without having to rely on a trusted third party for mediation purposes. In the cryptocurrency example, for instance, the blockchain enables individuals to digitally transfer value with one another over the internet, simply by trusting the underlying system to both enable that transfer and ensure sender authenticity and currency validity [7]. Whereas in a “centralized” system, a single third party (*e.g.* Bank) is needed to act as the intermediary who guarantees those two properties.

In the case of this dissertation, blockchain brings immutability to all the data related to land clearing contracts made between sheep and land owners, such is the case for the health status of the animals, the data generated during pastures, and even the rating data associated with each user, creating the potential for an increase of trust between parties.

## 1.1 Motivation

The use of sheep for land clearing purposes is a more efficient way to do so than the current traditional methods used, providing a more flexible, ecological and cheap alternative. This method can become widespread through upcoming business models, such as “Rent-a-sheep” or land clearing as a service, where sheep owners could become land clearing service providers, who could specify their own terms and conditions for their services, such as prices and deadlines. Contracts could then be made between them and Field Owners to officiate the service and ensure the conditions will be fulfilled by both parties. For this, live data from the sheep, provided by the SheepIT [4] infrastructure, such as their location and time spent inside the field can be used to monitor how the land clearing service is progressing.

But how to connect parties in order to engage in a contract? One option would be for a marketplace to be created through any kind of web platform, much like recent systems such as Airbnb and Uber, which disrupted the hospitality and private transportation sectors respectively. In our case, sheep owners will be able to create competitive offers, which will become visible for field owners to scroll through when searching for land clearing services. Participants will have an associated rating, reflective of any feedback gained from past contracts done through the system, in order to encourage people to trust each other beforehand and incentivize users to perform a good service.

Technologies such as Blockchain and smart contracts can then be added on top of this to guarantee that data is not manipulated in order to create unfair advantages to some party, that way enabling trust between parties.

Therefore, this project aims to create a system where these technologies are integrated, in order to

create a trustworthy marketplace designed to make the contractualization of land clearing services much more reliable and effective.

## 1.2 Objectives

The Objectives for this dissertation are the following:

- Analysis of similar projects or academic research in the area of sheep control and land clearing management in order to analyze the feasibility of different approaches to possible problems.
- Analysis of current Blockchain frameworks and selection of the one that fits the problem the best.
- Define a system in which all the participants involved in the land clearing business (field owners, animal owners, regulators, veterinaries) can retrieve information about other parties they might want to do business with, create contracts with them, check up on the state of these contracts and related information, while staying sure that all the available information is true and was not tampered.
- Create a working prototype of this system, choosing the best fitting technologies from all the analyzed possibilities.

## 1.3 Document Structure

This document is composed by the introductory concepts, followed by the Related work which will give an overview of the current relevant technologies being used in projects with similar goals or architectures. Then, a section showing the necessary components and requirements for the project to be built, followed by the explanation of the built solution. We will then analyze the obtained results and make the appropriate conclusions.



## CHAPTER 2

---

### Related work

---

In this chapter, we make an analysis of the problem of land clearing, how it can be achieved with the use of animals, and how technology improves the efficiency of this process. Then we analyze relevant projects which use these technologies, in order to gather valuable information about the feasibility of each studied approach. Afterwards, we make an analysis of distributed ledger technology, namely blockchain, and the different criteria that define it.

### 2.1 Usage of animals for land clearing

The use of animals for land clearing is a traditional method that works quite well [8]. They can easily clear land on steep hillsides and rough terrain and can go where people and machines cannot go safely [9]. So, in a matter of days, these animals can take a heavily infested with brush and weed area and turn it into a highly fertile field, with added protection against fire. This makes this approach a more versatile, cost effective and environmentally safe solution than the use of chemical or machinery for land clearing.

However, simply releasing animals in a field could come with risks, such as attacks from predators or animals getting lost, causing accidents and damaging valuable property, so, traditionally this has been

solved with wired or electrical fencing, but this can be an expensive and inflexible solution.

To tackle this problem, there have been a few recent projects which aimed at developing solutions to enable the monitoring of the animals, allowing their owners to exert a better control over their behavior, most importantly their location.

An example is the project that recently took place in Universidade de Aveiro, SheepIT.

### 2.1.1 SheepIT

The SheepIT project (Universidade de Aveiro, 2017-2019) aimed at developing a solution for monitoring and controlling grazing sheep in vineyards and similar cultures while also collecting data about sheep activity for logging and analysis purposes [10].

To achieve this, they developed a system which enabled sheep to be controlled in an efficient way. For this they conceptualized a way to deploy virtual fences surrounding a desired area, where the animals are allowed to freely roam, while wearing portable electronic collars, which not only permit the tracking of their location, but the automatic conditioning with the use of harmless stimuli, namely warning sounds and vibration, ultimately preventing the animals from stepping out of the desired perimeter.



Figure 2.1: Sheep Carrying a Collar [11]

It also allows the deployment of algorithms to process the data and detect abnormal situations, such as health conditions, lost animals or attacks from predators, generating automatically alarms when one of such events occurs [12].

The system architecture was designed in order to provide a flexible and adaptable solution regardless of vineyard's size and shape. Moreover, the human intervention is maintained at a low level, being only indispensable for setting up the devices [12].

These devices (collars), responsible for monitoring and controlling the animals' behavior, communicate with fixed nodes (beacons), which are equipped with GPS, and take care of interconnecting the network devices by relaying the data to an end node (gateway), which in turn aggregates data from all the beacons and uploads it to a cloud platform. Then, a cloud application is responsible for gathering all the data collected and presenting it to the human operator, ultimately allowing him to oversee the herd in an efficient way.

The architecture of the system is shown in the following figure:

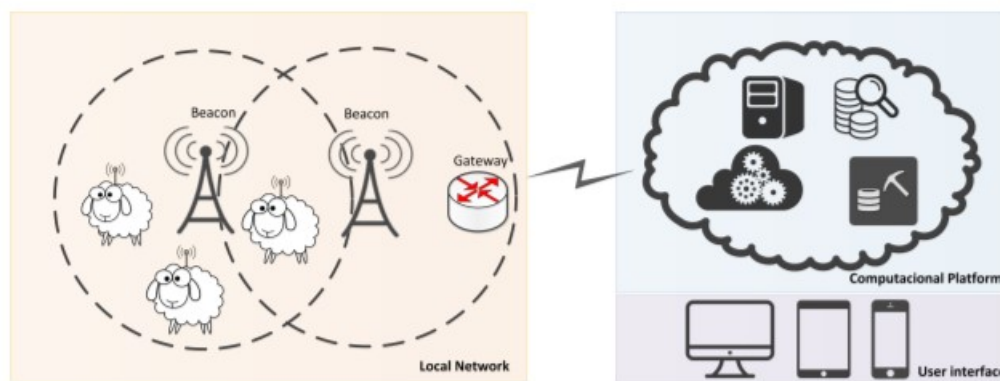


Figure 2.2: System architecture [13]

The important elements of the system are:

**Collars** - Worn by the sheep, these are composed of a radio, a micro controller, sensors, actuators, and a battery. Their purpose is to communicate with the beacons and control the sheep's behaviour.

**Beacons** - Composed of a radio, a micro controller and a GPS module, these are fixed nodes, installed by the shepherd, in order to provide coverage to the entire desired grazing area. They are responsible for the synchronization of the network and relaying the information collected from the collars to the gateway.

**Gateways** - gateways are the interfaces between the sensor network and the Internet. It is composed of a beacon and a computer connected to the internet. All beacons must be registered in the gateway in order to communicate with it. Since all the beacons direct the data to this node, it is the only one capable of knowing the global state of the network.

The location of the animals is obtained with the use of an approach proposed by José Pereira, which kept in mind the need for low energy consumption. It consists of a hybrid solution, using GPS to determine the absolute positions of the beacons, and the Received Signal Strength Indicator (RSSI) values (measure of the strength of a radio signal) acquired during the communication between collars and beacons, to estimate the position of the collar relative to the beacons. The RSSI value between a collar and a beacon

is relayed to the gateway. Then, having RSSI values respective to at least 3 beacons, a trilateration algorithm can be run on the gateway, to achieve an estimate of the absolute position of each collar, while keeping the power usage low, for extended battery life [13].

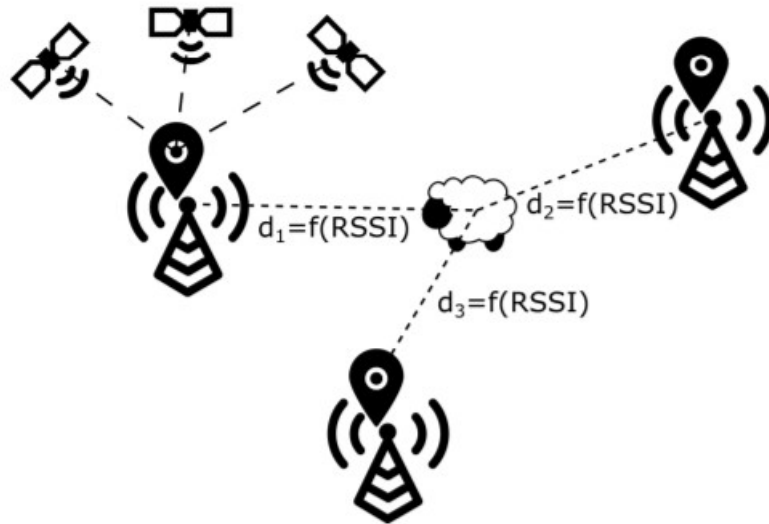


Figure 2.3: Location system operation [13]

Having access to the data about the real-time location of each animal, algorithms are run on the collars which can easily detect a fence infraction, triggering an audible warning signal. If during the subsequent communications the animal doesn't return to the inside of the perimeter, the system responds with an electrostatic stimuli. This has been proven to be an efficient and harmless way of exerting control over animals, not only in the context of SheepIT, but also other similar projects (Bishop-Hurley *et al.*) where a prototype virtual fence was set up, while allowing animals to walk through an alley contained in it. The results showed that the virtual fence was successful in modifying the behaviour of the sheep. The experiment demonstrated virtual fencing has potential for controlling sheep in extensive grazing systems [14].

The SheepIT project also enables the control over the posture of the animals, allowing even further control over their behavior, which can be a very important feature in cultivated fields that have valuable products at the animals' reach, where it is important to prevent the animals from eating them, which would result in production losses. This was achieved with a mechanism comprised of a set of sensors, to monitor the animal posture and movements; an algorithm executed by a microprocessor that analyzes the data gathered by sensors, applies sensor fusion and decides about the necessity of applying stimuli, along with a set of actuators that apply the actual stimulation to the animal, when commanded to do so [12].

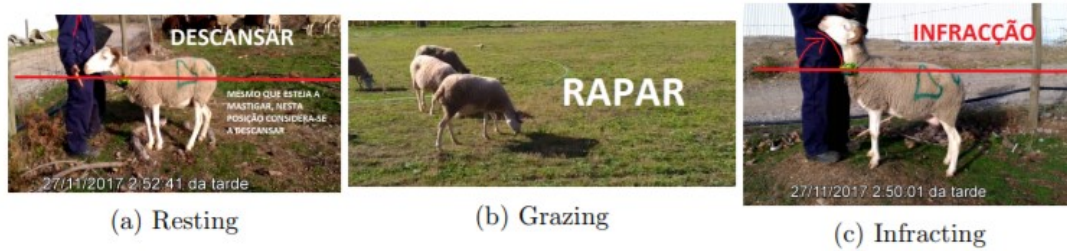


Figure 2.4: Classification Rules [11]

As can be seen in the picture above, the collar can detect when the animal has adopted a posture that likely means that it is feeding on the valuable fruits located higher up. With this information, it can use the actuators to deliver stimuli to the animal to correct the undesired behavior.

## 2.1.2 Land clearing services

There are several companies which use animals to provide land clearing services.

### Rent-a-Ruminant

Rent-a-ruminant is a company based in the United States, which provides land clearing services. They handle the transportation of the animals and installation of hardware such as the essential electric fence that contains the herd and identifies potential goat hazards such as high traffic, sheer drop-offs or poisonous plants [15].

Owners of sheep who are looking to provide land clearing services contact with Rent-a-ruminant in order to open a franchise. By doing that, they benefit from the popularity of the company, which ultimately helps them find customers who are looking for these services. Having secured a deal, generally, before any work begins, a goat wrangler will perform a visit to the client's site to evaluate it for access and safety of the goats, and how to best install the electrical fencing. Once that is done, they will begin the service by bringing the animals to the premises, where the animals graze for several days until the desired result has been achieved. It usually takes three to five days to clear one acre, and their prices for 15, 60 or 120-goat herds range from \$250 to \$725 a day.

A comparison between a field prior to the service, and afterward is shown in the following figures:



(a) Before a service



(b) After a service

---

## The Goat Lady

The goat lady is a company that specializes in residential and commercial brush removal using goats and sheep, working on very flexible conditions, with the possibility of the rental of small herds of as little as 2 animals, and up to 50, having delivered services with durations ranging from 1 day to a few months [16].

Their workflow starts with an inspection of the site to evaluate the conditions, with this, they give a free estimate to the client of the total value of the service. This will be the final price independently of anything that could happen to interfere with the work, such as weather conditions. If an agreement is reached, the company will bring the animals and the necessary hardware to keep the animals in the desired area. For bigger services, this will be an electric fence, while in smaller services there's the option of reducing costs by tethering the sheep to stakes. In this case, homeowners must provide water, keep the tethers untangled and move the goats to new grazing areas every few days [17].

For a herd of 25 goats, they charge \$350 for the first day, and \$325 for the subsequent days for residential rentals, while commercial rentals need to be bid according to the specifics of the work needed (size of area to be cleared, time needed for the job, hazardous vegetation removal, etc.) [16].

## 2.2 Contract management platforms

An important aspect of the desired system is a platform designed to connect the interested parties, allow them to make contracts with each other, and act as an unbiased intermediary to facilitate and guarantee the interests of all parties involved in a contract.

Examples of such platforms are the recent companies with business models giving rise to the ever-expanding shared economy of today. The most famous examples are Airbnb and Uber, which completely

disrupted the housing and transportation markets, respectively, by acting exclusively as an intermediary between service providers and clients.

## 2.2.1 Airbnb

Airbnb is an online marketplace that connects people who want to rent out a space, to people who are looking for a place to stay in a certain locale.

The basic workflow in Airbnb is the following: Someone who is interested in renting out a place through the platform must register in order to create a profile. With it, they can then publicize their place on the platform, where they present the space (pictures, location, description), the essential information such as price per night, the number of guests allowed, minimum number of nights, along with any other relevant details they desire, such as restrictions on pets, smoking or noise.

The listing is then created and visible to anyone using the platform, also displaying the availability of the listing through a calendar. Then, users who are looking for a place to stay can search in the platform for a place in a certain location, specifying the desired check-in and check-out dates. The platform shows them the available results, which they can analyze and evaluate based on their preferences.

If they find a listing which satisfies their interests, they can take the next step, which is the booking process. This consists of sending a booking request for a specific time period, which in turn needs to be accepted by the owner of the space. Airbnb also enables the hosts to activate the “instant book” option, which enables guests to instantly book their space without the need for the host’s confirmation. The company states that this option brings benefits such as increased earnings and boost in search results [18]. When a booking request is accepted, the deal is made, and the guests will be expected at the specified dates. After the stay, both parties can leave a feedback comment along with a 1 to 5-star rating representative of their satisfaction with the experience. This rating and previous comments are visible to anyone on the platform. This creates a reputation system in which the hosts are incentivized to deliver a good service, and guests to act according to the house rules, in order to strive to achieve a better reputation score, which ultimately greatly improves their chances of securing good deals through the platform in the future, since every user is more comfortable in doing business with someone who has a well-proven reputation.

The payment is made by the guest at the time of the booking, Airbnb acts as the middleman here, holding on to the money, and transferring it to the host 24 hours after the check-in date [19]. To note that the company charges a service fee to both host and guest based on the total price of the reservation, which is typically around 13% [20].

Airbnb’s users are comfortable doing business with other people using these platforms because there is a reputation system, in which the users’ reputation is built based on the positive/negative feedback they receive from service providers after having used their service, and the reputation of the service providers is built based on the feedback they receive from users that used their services. Both users and service providers trust in this reputation system because they trust the company in charge to be impartial, as to not have an interest in benefiting some users over others, and to be competent to prevent false feedback from being added.

Therefore, these are centralized reputation systems, which work nicely on their own, but are still susceptible to manipulation strategies from the mediating entity, for example, listing offers from “premium” users higher up, in order to increase revenue.

## 2.3 Distributed Ledger Technology

Distributed Ledger Technologies, as the name suggests, can be seen as a distributed database, where information is stored in a shared, synchronized, and replicated way, where information can be stored and accessed by all participants, independently of their geographical location. Being completely decentralized, there is no governing authority, so, the information to be added or discarded is chosen by a consensus algorithm, which prevents one single entity from being able to corrupt the ledger with false information.

### 2.3.1 Blockchain

Invented in 2008 by an unknown person or group of people using the pseudonym “Satoshi Nakamoto” to serve as the backbone of the world’s first cryptocurrency, Bitcoin, it was designed to serve as a public ledger, holding all the Bitcoin transactions ever made. The success of Bitcoin as a digital currency is debatable, but the underlying Blockchain technology is irrefutably useful because it has the capability of establishing trust between all the participating parties in a business ecosystem, with no need for a central authority to do so. It accomplishes this by replicating a common ledger across a high number of nodes (servers), which store the current state of all the information contained within it. The updates on this ledger are controlled through a *consensus protocol*, which ensures that no invalid information can be added to the ledger.

In order to update the ledger, a user needs to broadcast a transaction to the network which holds all the relevant information to update it. Then, the hash value of the transaction’s contents is calculated, along with the hashes of other transactions being broadcasted. The result is then fed into the header of a



bundle of transactions to be inserted in the ledger - a *block* (Figure 2.5). This is then combined with the hash value of the previous block and the current timestamp.

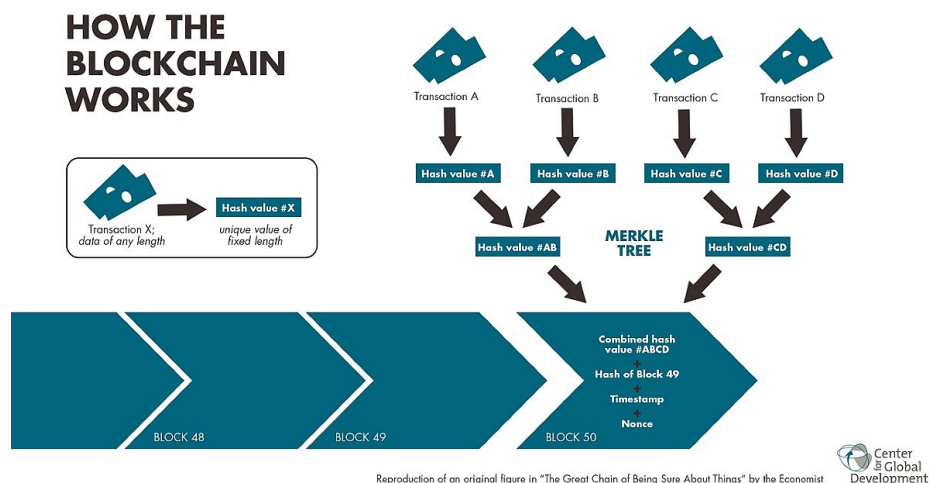


Figure 2.5: Workflow of a Blockchain [21]

The fact that the previous block’s hash value is used for the current block header, means that it is impossible for anyone to make changes in the chain and get away with it because the slightest modification would completely change the hash values of the preceding blocks. Thus, the integrity of the Blockchain and the data within it, are always preserved. This property also allows for the provenance of each transaction to be tracked all the way back to its origin, which can be extremely useful, for example, for law enforcement to track apprehended money used for illegal activities.

In terms of accessibility, a blockchain network can either be private (permissioned) or public (permissionless). This will define who can participate in the network, more specifically, who is allowed to write data onto the ledger.

In a **public** blockchain, anyone can download the protocol and read, write or participate in the network, this is the most widespread type since most cryptocurrencies use a public blockchain. On the other hand, in a **private** blockchain, restrictions are placed on who can participate in the network and interfere with the ledger.

Going into further detail, one can also define who can read the data of the blockchain, in an **open** blockchain, anyone can do so, while in a **closed** solution, only a restricted group are allowed. Figure 2.6 classifies a few common use cases of blockchain networks according to the parameters analyzed.

The figure below shows that different levels of accessibility make certain blockchain solutions more practical for certain use cases. This are just guidelines though, for example, “there’s a perception that

		<b>Ethereum Bitcoin</b>	
		<b>Public &amp; Closed</b>	<b>Public &amp; Open</b>
		<ul style="list-style-type: none"> <li>• Voting</li> <li>• Voting records</li> <li>• Whistleblower</li> </ul>	<ul style="list-style-type: none"> <li>• Currencies</li> <li>• Betting</li> <li>• Video Games</li> </ul>
		<b>Private &amp; Closed</b>	<b>Private &amp; Open</b>
		<ul style="list-style-type: none"> <li>• Construction</li> <li>• National Defence</li> <li>• Law enforcement</li> <li>• Military</li> <li>• Tax Returns</li> </ul>	<ul style="list-style-type: none"> <li>• Supply Chain</li> <li>• Government financial records</li> <li>• Corporate earning statements</li> </ul>
<b>Hyperledger R3 Corda</b>			

Figure 2.6: Different kinds of accessibility for blockchain networks [22]

public blockchain platforms like Ethereum can't be used to build permission scenarios or to control access to data. The truth is that they can, they just don't give you all the built-in tools that you can find on a private or permission blockchain platform. But basically, you can always use these open public platforms to build a permission solution, you just need to be aware that it's upon you, your architects and your developers to create that permissioning model, and that all starts with identity management system" [22].

## 2.3.2 Consensus Algorithms

Consensus algorithms play a very important role in the Blockchain technology, they are responsible for maintaining the integrity and security of these distributed systems [23]. They do this by ensuring that all the nodes in the system agree on the current state of the blockchain. However, reaching consensus on distributed systems safely and efficiently is an interesting challenge [24]. How can a distributed network of computer nodes agree on a decision, if some of them are likely to fail, or act dishonestly? This is the fundamental problem of the Byzantine Generals, which goes as follows: A number of Byzantine Generals plan to attack a city, they are encircling it on a radius big enough that instantaneous communication between them is impossible, so it must be carried out through messengers which take a considerable amount of time to carry the message.

The generals must attack together, at the same time, to achieve victory. To do this, they must find a way to signal the other generals about their plans, to reach a consensus about when to do it.

When receiving a message from the carrier, the generals can never trust it, because the communication

channel cannot be trusted since the message could have been written by a general who wants the plan to fail, or the messenger could have been compromised by the city, or even have ill intentions.

So, how can the generals come up with a way to reach a consensus about the time to attack?

Since Blockchain systems are inherently trustless, decentralized, and actors act independently and according to their interests and the information available to them, there would always be incentives for malicious attackers to broadcast false information for their benefit. Therefore, nodes must find a way to reach an agreement about which information is true, and which is not, assuming that there will always be ill-intended actors in the system. This is called **Byzantine Fault Tolerance (BFT)** and it is an important property that any consensus algorithm must address.

## Proof of Work (PoW)

This is the consensus algorithm used by Bitcoin, and it governs the way that blocks can be added to the chain. The general idea is for a system to require proof that some kind of work was put into any request that arrives to it. This is useful for example to deter Denial of Service (DoS) attacks, because the proof of work, in this case, is usually a significant amount of computing power that has to be put into the request.

In the case of bitcoin, the idea is very similar, a reward is allocated for every time a new block is to be added to the chain, this reward will be given to whoever is the first to find the solution to an extremely hard cryptographic puzzle. The idea is for the puzzle to be very hard to solve, but very easy to verify. For this, a hashing function is ideal - one that requires high amounts of computing power to brute force their way through millions of possibilities, to find the one that matches the solution. This process is called *mining*, and people who do these are subsequently *miners*.

In the early days of bitcoin, since there wasn't much competition, normal computers would do the trick, but nowadays, with the incentive of the reward [25], very expensive sets of hardware, some even purposely made to run the bitcoin hashing algorithm, with the possibility of crunching through millions of hashes per second [25], are needed to have any chance to be close to a reward.

The way this puzzle-solving brings consensus to the network is because, out of the many participants trying to add a new block (which contains transactions) to the chain, only the first to find the solution will be given the right to do it. This means that for an ill-intended individual to add blocks containing false information

And so, an ill-intended individual would have to own more than 50% of the hashing power in the whole

network to be able to double spend his currency.

This algorithm works quite well for generating consensus, but in some cases it can have one big flaw, for example, in the case of Bitcoin, absurd amounts of hashing power is being used by the miners to secure the network, which leads to a huge worldwide energy waste, especially since the computations are not useful in any sense other than securing the network. Some other projects have tried to tackle this problem, by using the computations to generate valuable work, while also securing the network, such is the case of Oyster, in which the computations are also used to upload files into a decentralized cloud storage network.

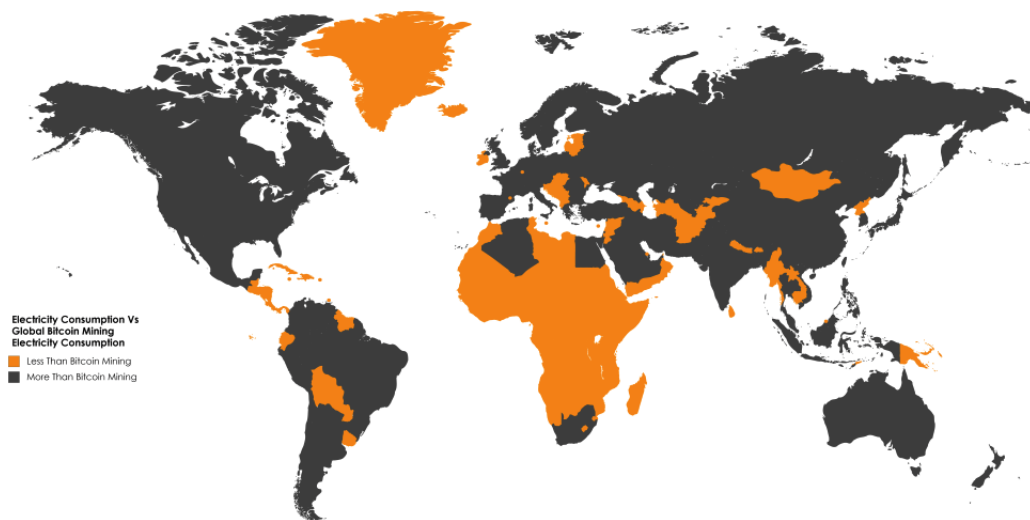


Figure 2.7: Countries with lower energy consumption than the Bitcoin network[26]

## Proof of Stake (PoS)

The Proof of Stake algorithm works quite differently from the PoW, and it is being studied and improved nowadays because it can bring important benefits, one of them being a great reduction in energy demand when compared to PoW. Ethereum, the 2nd biggest cryptocurrency to date, is set to switch their consensus algorithm from PoW to a PoS based solution [27]

“Each party has a certain amount of stake in a Blockchain (typically the amount of cryptocurrency). For each block there is a randomized leader election process; the party that gets elected can issue the next block. The more stake a party has, the more likely it is to get elected as a leader. Similarly to PoW, block issuing is rewarded (for example by collecting transaction fees from parties that want their data to be included).” [28]

This “leader” is usually designated as minter or validator. Their stake is locked in the Blockchain, and it is released only once the network has verified that the transactions inside the block validated by

the minter are legitimate. This means that if the minter tried to cheat by approving fake transactions, he will lose his stake.

From the algorithmic perspective, there are 2 major types of PoS: Chain-based and BFT-style. In **Chain-based** proof of stake, the algorithm, at the end of a certain period, will randomly choose a validator, and assigns them the right to create a single block, which must link to a previous block, belonging to the previously longest chain. This will make the blocks converge over time into a single constantly growing chain. This approach is preferably chosen for permissionless Blockchain implementations, as it brings suiting benefits such as more availability of validators.

**BFT-style** proof of stake, (Byzantine fault tolerance) the act of suggesting a block, and creating it are separated. Validators are randomly assigned the right to propose blocks, but the decision on which block is canonical is made through a multi-round voting process where each validator votes for a block in each round. This approach is more favored for permissioned implementations, as this brings more consistency to the system.

A **51% attack** is extremely improbable of happening, since it brings a monetary disadvantage to the attacker. They would need to own more than 50% of the respective currency, but once the attack had happened, it would be publicly visible, which would crash the value of the whole network, resulting in a huge loss for the attacker.

**Nothing at stake attack** “In PoS it is much easier to construct alternative chains. In PoW, a party would lose CPU time by working on an alternative chain, whereas in proof-of-stake, party seemingly does not lose anything by also mining on an alternative chain.” [28]

### 2.3.3 Smart Contracts

Smart contracts are one of the big use cases for DLT's, they consist of a computerized transaction that executes explicit terms of a contract. The objective is to enforce the conditions stated by all the parties engaged in a contract, (for example, payment terms, deadlines, confidentiality) while eliminating the need for a middleman, and minimize malicious or accidental breaches. This can bring economic and logistical benefits such as a higher level of efficiency, security, and transparency, and lowering the costs of transactions, arbitration, and enforcement.

The main principle of smart contracts can be compared to that of vending machines. All the instructions for how it should proceed are directly coded by its creator(s), then, the contract will execute and generate different outputs depending on its inputs.

The idea was originally explored in 1994 by computer scientist and former George Washington University law professor Nick Szabo [29]. He defined the main requirements for smart contracts to become a reality, but at the time there was no appropriate environment to realize them.

Since then, a lot has changed, mainly with the emergence of Blockchain technology, which laid the basis for setting up contracts on decentralized networks, where the contract participants agree on the terms of a contract by signing it with their private keys. The contract can then be deployed, and be distributed among the nodes of the platform, where its code will be immutable, and publicly visible to all the participants.

Thanks to their properties and functionality, smart contracts brings several advantages when compared to traditional methods of establishing contracts. Some of these are:

**Digital rarity and identity** - Some decades ago, owning a vinyl copy of a song, meant you owned that copy, and it had value. But with the advance of technology, the concept of digital uniqueness disappeared, because anyone could easily make as many copies of whatever they wanted. With Blockchain and smart contracts, the possibility of owning something unique in the digital world is back, with the added possibility of having no central authority issue the valuable items, but a completely decentralized smart contract doing so. Such is the case of Gods Unchained, an online trading card game where players can earn and trade cards for real money, which, at the time of writing, is experiencing a volume of 150000\$ being traded per day.

**Decentralized Autonomous Organization (DAO)** - A DAO is a thought experiment, based on the idea that, if the blockchain can remove the middleman from value transactions, then maybe an organization such as a company could one day operate without hierarchical management, but in a decentralized manner instead [30].

The basic rules for its functioning would be hardcoded from the get-go, then, according to external input, the rules would be automatically enforced by the smart contracts in the DAO. Many use cases can stem from this, a basic example could be a DAO which invests its capital on the most voted company by its users.

The most famous DAO experiment to date, unfortunately, didn't go well - the DAO incident [31], where a decentralized autonomous organization, with the intent of allowing participant companies to make proposals for funding. Hackers were able to find some mistakes made by the developers, which enabled them to empty the contract and steal the equivalent of 60\$ Million in Ethereum.

**Elections** - Even though the question of electronic voting is still in debate whether or not it is a good idea, Blockchain and smart contract technologies could certainly be a step forward in creating a voting

system that could be tamper-proof, where all the important data could be encrypted.

“In September 2016, Kaspersky Labs, industry leaders on computer security, and The Economist newspaper organized a case-study competition where MBA teams from the US and the UK had to implement voting systems using the Blockchain” [32]

“In this case, the nodes would be voting machines that have received permission to be nodes, creating a “permissioned Blockchain”. Votebook’s 19 system consists of voting machines resembling traditional voting machines that act as nodes on a Blockchain. Once polls open, voting machines would collect votes and organize them into a “block”, ready for verification on the network. The block would be broadcasted on the network and every other node will check the validity of the block’s components by using their public key that corresponds to the proposing node’s ID to decrypt the hash and check for a match” [32]

## 2.4 Platforms for blockchain application development

In this section, we will analyze two projects available that allow the creation and deployment of blockchain networks. As our criteria, we chose the two most popular and used platforms available at the time of writing [33].

### 2.4.1 Hyperledger

Hyperledger is a project developed by The Linux Foundation, with the aim of advancing cross-industry collaboration by integrating Blockchains and distributed ledgers in their business systems, with the focus on improving efficiency and reliability of global business transactions between worldwide companies. The project integrates use-specific modules, with independent open protocols and standards, including Blockchain solutions with their own consensus and storage routines, identity services, access control, and smart contracts.

Hyperledger’s design philosophy follows these principles: [34]

- **Modular** - Hyperledger is developing modular and extensible frameworks with common reusable building blocks that enable developers to experiment with different types of components and interchange them at will without compromising the rest of the system. This means that developers can create independent components that can be combined and reused to build distributed ledger solutions appropriate to different requirements.

- **Highly secure** - Security is extremely important for distributed ledgers, especially since many use cases involve high-value transactions or sensitive data. With large codebases, many networked nodes, and valuable data flows, distributed ledgers have become a prime target for online attackers. Distributed ledgers must provide a large set of features and functions while managing to resist adversaries. Hyperledger projects embrace security by design and follow the best practices specified by the Linux Foundation's Core Infrastructure Initiative. [34] As such, all Hyperledger algorithms, protocols, and cryptography are reviewed and audited by security experts, as well as the wider open source community, on a regular basis.
- **Interoperable** - In the future, many different Blockchain networks will need to communicate and exchange data to form more complex and powerful networks. Therefore, smart contracts and applications should be portable across many different Blockchain networks. This will help meet the increased adoption of Blockchain and distributed ledger technologies.
- **Cryptocurrency-agnostic** - Hyperledger projects are independent and agnostic of any cryptocurrencies whatsoever, however, the design philosophy includes the capability to create a token used to manage digital objects, which may represent currencies, although this is not required for the network to operate.
- **Complete with APIs** - All Hyperledger projects provide rich and easy-to-use Application Programming Interface (API)s that support interoperability with other systems. This enables clients and applications to connect easily with Hyperledger's core distributed ledger infrastructure and reap the benefits of doing so, even with little knowledge of the Blockchain architecture. These APIs support the growth of a rich developer ecosystem, and help Blockchain and distributed ledger technologies proliferate across a wide range of industries and use cases.

Hyperledger comes also with a set of tools designed to improve blockchain solutions, by providing extra functionalities or increasing developing efficiency.

## Hyperledger Fabric

Hyperledger Fabric (HLF) is one of the implementations of the Hyperledger project. It consists of a distributed ledger platform for running smart contracts, which makes use of familiar and proven technologies, along with a modular architecture, allowing pluggable implementations of various functions.

The **key features** of HLF are:

- **Channels** - A Channel is a private subnet of communication between two or more specific network



members, to conduct private and confidential transactions. Only peers connected to a channel will receive the transactions that pass through it. This is one way of guaranteeing the data confidentiality requirements of this project. The other way to do so is to control the access to the data at the client level, through the **access control list (ACL)** capabilities of Hyperledger.

- Peers - These are effectively the nodes which will hold the replicated ledger(s) and smart contracts. They are run by any organization which wishes to hold a copy of the ledger. **Endorsing Peers** also have to hold the smart contracts in them, to confirm the validity of a transaction by simulating the transaction with it [35].
- The endorsement policy, which states which peers must endorse a transaction for it to be valid, must be provided at the moment of the creation of the network.
- Ordering service - This is a service that will order the transactions into the correct order and is responsible for maintaining a consistent ledger state in all the peers in the network. An orderer is built on top of a message-oriented architecture. There are two options currently available to implement an ordering service [36]: **Solo**: Suitable for development but has a single point of failure. Solo should not be used for the production-ready network. **Kafka**: Suitable for production, HLF network uses Kafka as the Orderer implementation. Kafka is a messaging software that has high throughput and fault-tolerant features.
- Certificate Authority - This will provide identities to the participants, each organization should have its own CA, for all components of the network to become identifiable.

## Hyperledger Composer

Hyperledger Composer (HLC) is an extensive, open development toolset and framework build using the HLF infrastructure and runtime. Its primary goal is to accelerate time to value and make it easier to develop Blockchain applications and integrate them with existing business systems. [37] It can be used to model any business network, containing the existing assets, participants, and transactions related to them, create a secure network with this information and provide an endpoint for everyday applications to consume the data retrieved from the Blockchain application.

It comes with the tradeoff of being slightly less flexible than fabric, for example, features such as multiple channel configurations [38] and Private Data Collections [39] are only available in HLF.

HLC includes an object-oriented modeling language that is used to define the domain model for a business network definition, with it, business networks are modeled according to assets, transactions, and participants, and put into a .cto file.

All the transaction logic is then handled with a JavaScript file, containing the behavior for all the transactions defined in the .cto file, this allows for all kinds of operations related to the Blockchain, such as changing world-state values of assets or participants.

Access control rules are then defined on a .acl file, to determine which users/roles are permitted to create, read, update or delete certain elements. Hyperledger provides an access control language (ACL) do enable this.

Queries for data retrieval from the Blockchain are defined in a .qry file, using a language very similar to normal languages for managing databases.

These files are then packaged into a .bna file, which is ready to be deployed to the preferred platform.

These are the key concepts of HLC: [37]

- **Assets** - Tangible or intangible goods, services, or property which might be owned by certain participants. These are stored in registries. Assets can represent almost anything in a business network, from a physical asset such as a house to an intangible one such as a financial bond. Assets can contain whatever useful properties one might want to give it, along with a mandatory unique identifier. They may then be related to other assets or participants.
- **Participants** - Participants are members of a business network that interact with it in some way. They may own assets and submit transactions that can interact with them. Participants
- **Transactions** - Transactions are the mechanism by which participants interact with assets. They do this by changing world state values of properties in the assets, such as ownership (*e.g.* for a sale)
- **Blockchain State Storage** - All the transactions submitted on a business network are stored on the Blockchain ledger, while the current asset and participant state is stored in the Blockchain database. The Blockchain distributed the ledger and the state database across a set of peers and uses a consensus algorithm to ensure that updates to the ledger and state database are consistent across all peers.
- **Business Network Card (BNC)** - Business networks combine identities, connections profiles and helpful metadata to simplify the process of connecting to a business network and extend the concept of an identity outside the business network to a 'wallet' of identities, each associated with a specific business network and connection profile.
- **Connection Profiles** - Connection Profiles are used to define the system to connect to. A connection profile is a JSON document that is part of a BNC. These profiles are usually provided by

the creator of the system they refer to and should be used to create BNCs to enable participants to connect to that system.

- **Identities** - An identity consists of a digital certificate and private key, which is stored in a BNC and mapped to a participant. This allows the respective participant to transact on the business network and verify his identity.
- **Queries** - Queries are used to retrieve information about the Blockchain current world-state. They are defined within a business network and can include various parameters for customization. This provides a simple and easy way to extract data from the Blockchain network.
- **Events** - Events are defined in the business network definition to indicate to external systems that something of importance has happened to the ledger. Applications can then subscribe to emitted events through the composer-client API.
- **Access Control** - Access control allows for the clear definition of access privileges and control over participants through the use of a control language that is rich enough to specify sophisticated conditions such as “Only the owner of an animal can transfer the ownership of the animal”. Externalizing access control from transaction logic and processing makes it easier to inspect, debug, develop and maintain applications.
- **Historian Registry** - The historian registry is a specialized registry which holds all successful transactions, along with information about the participants and identities that submitted them.

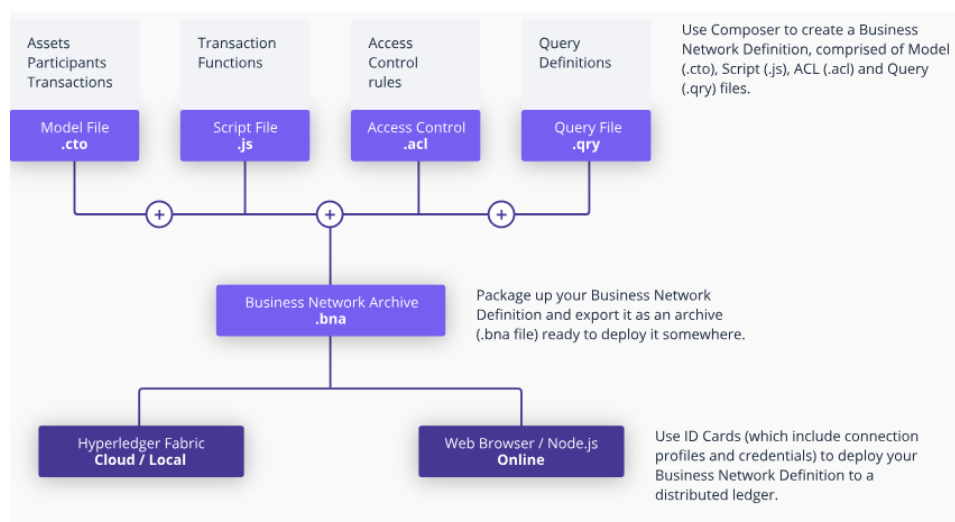


Figure 2.8: Architecture of HLC [37]

## 2.4.2 Ethereum

Ethereum is a public, open-source, distributed computing platform and operating system, providing a permissionless Blockchain solution that supports the development, deployment, and execution of smart contracts. It was initially described in a whitepaper written by Vitalik Buterin [40] in late 2013. Buterin argued that Bitcoin was lacking a scripting language for application development, so he took it into his own hands, and in 2014 released the working project.

Ethereum is the most generically and widely used smart contract platform, available for any kind of application to be designed over it. It uses a cryptocurrency called Ether to compensate mining nodes for computations performed, used to secure the network. An internal transaction pricing mechanism, “Gas”, is used to mitigate spam and allocate resources on the network.

Ethereum provides a decentralized Turing-complete virtual machine, the Ethereum Virtual Machine, which is the runtime environment for smart contracts. Every Ethereum node in the network runs an EVM implementation and executes the same instructions, forming the fundamental consensus mechanism for Ethereum. The EVM’s instruction set is kept minimal for simplicity, and to avoid implementation problems.

The world state is represented by **accounts**, which hold its address, made up of 20 bytes, and all the **transactions** leading up to the current state, which may transfer value or information.

There are two distinct types of **accounts**, although this distinction may be eliminated in a later version of Ethereum, these are:

**Externally Owned Accounts (EOA)** - EOA’s are owned by anyone who holds their private key, they have their address to be identifiable, which allows them to receive and send transactions, resulting in the current ether balance associated with the account. These EOA’s have no associated code.

**Contract Accounts** - Contracts are autonomous agents “living” inside the Ethereum execution environment, they function much like EOA’s, but they have associated code which makes some of their functions available to be called by incoming transactions when these include metadata identifying which function to call, and with which parameters. The contract code is then executed by the Ethereum Virtual Machine on each node participating in the network as part of their verification of new blocks.

On the other hand, **transactions** are signed data packages that store a message to be sent from an externally owned account to another account on the Blockchain.

These are composed of: The **recipient** of the message, A **signature** which identifies the sender and proves that they intend to send this transaction A **Value** field, which specifies the amount Ether to send,

along with optional data, normally used to trigger specific functions in smart contracts A **STARTGAS** value, representing the maximum number of computational steps the transaction execution is allowed to take, a **GASPRICE** value, representing the fee the sender is willing to pay to have his transaction be “mined”, the higher the value, the faster it will go through since miners always choose to clear the highest paying transactions first.

Ethereum blocks contain a copy of the transaction list and the most recent world state. These blocks are then chained together, to create Ethereum’s Blockchain, securing the whole network.

Smart contracts are written in a contract-oriented programming language called **Solidity**. It was created for developing smart contracts in blockchain platforms such as Ethereum. It was developed by Gavin Wood, Christian Reitwiessner, Alex Beregszaszi, Liana Husikyan, Yoichi Hirai and several former Ethereum core contributors.

### 2.4.3 Comparison between these platforms

A comparison between the two platforms analyzed above is now given. This comparison will be mainly based on some of the most important aspects a project team looks for when deciding between the different possibilities for developing a blockchain solution:

- Development process and costs - In terms of the development with Ethereum, costs can be avoided with the use of the Ropsten Testnet, which enables the deployment of smart contracts and all the interactions with it to be done with the use of fake Ether and Gas, that can be easily be obtained through a faucet [41]. The programming language (Solidity) is generally accessible, with good documentation, and thousands of active developers.

HLF also doesn’t come with any mandatory costs associated with the development process. The documentation is very good, as it is run by one of the biggest open source communities in the world, The Linux Foundation. The availability of frameworks such as Composer is extremely enticing for a regular developer, as these tools make the development and deployment process quite easy.

- Running costs - In terms of running costs, a running contract on the Ethereum platform requires the payment of fees for the deployment of the contract and every transaction invoked. The value of the fees is dependent on the congestion of the network, so this can create a certain unreliability, as users could be forced to pay substantial amounts to interact with the system.

On the other hand, a Hyperledger network can be running with only the costs of keeping the nodes online on the respective machines, no other costs are mandatory.

- Privacy - Ethereum’s permissionless and transparent nature also comes with the unavoidable lack of privacy, as all transactions flowing through the network are available to be viewed by anyone.

HLF, on the other hand, works on a consortium basis, where only the participating users can view the data in it.

- Performance and scalability - The high level of security obtained with consensus protocols such as Ethereum’s proof of work, where transactions need to be “mined” and validated before they can be added to the chain, comes with the tradeoff of slow transaction speeds, frequent congestion periods, and the consequent rise of transaction fees [42], as miners naturally give priority to the transactions which carry a bigger reward for themselves. Ethereum’s network is ready to handle up to transactions per second [43], transaction speeds depend on the GAS paid, but usually hover between 15 seconds and 5 minutes [44].

HLF’s permissioned approach allows for more modularized and extendible practices [45], such as customizable requirements for transaction validation, which can lead to a great gain in performance. Fabric’s performance numbers greatly depend on the configuration of the network, with theorized numbers of up to 50000 transactions per second [46]. In more realistic setups it has been tested to handle up to 3500 TPS, with negligible transaction speeds [47].

In summary, Ethereum is the most generically and widely used platform, available for almost any kind of application to be designed over it. HLF seems to be more business-oriented, being more appropriate to enterprise use cases, or in situations where data confidentiality or network performance are important.

## CHAPTER 3

---

### Solution

---

In this chapter, all the requirements needed to instantiate the desired system will be detailed and explained, these requirements are based on the vision that this system is to become a marketplace where the interested parties in the land clearing business can participate in order to engage in business with each other while relying on the users reputation and animal monitoring features provided by the system to be able to trust each other.

Let's assume John owns a herd of sheep and wants to generate profit with them by acting as a land clearing service provider on our platform. First, he must register in our system to be able to create his profile and be identifiable. After having done so, the next step is to publicize his intent of engaging in land clearing deals. He will do this by creating a listing, in which he must detail the relevant information about the service, such as the price per sheep per hour that he is wishing to receive, when the service will be available, and the number of available animals to allocate to this service.

Then, Rick, a field owner, after also having registered in the platform, can browse through these offers, and look for the ones that matches his interests the best. Having found one that interests him, he can then make a solicitation to the offer in question. While doing this, Rick has an overview of John's service availability, where he can check whether John is free or booked for a specific period. To send the solicitation, Rick must choose the dates in which he wishes to have the animals come to his terrain

for the land clearing service, and include any relevant information in it, such as the field's location, size, vegetation type, or restrictions he might have, *e.g.* in terms of the maximum number of animals, deadlines, contract finishing conditions and payment terms.

John would then receive this, and possibly more solicitations on his listing. He could browse through these and look for one that matches his interests. While doing this, John has access to information about the other party too. This will help with the decision making, since any participant will always feel safer when doing business with someone with a good reputation score, since this means that users who engaged in deals with them in the past, in general, have had good experiences.

Having found an interesting solicitation, John can choose to accept it. This will bind both parties to an agreement called a land clearing contract. This contract will include the properties stated in the land clearing offer, and the restrictions from the solicitation. The most important ones are the price per sheep per hour to be paid and the contract finishing condition. This condition can be a parameter that the system can automatically verify, such as the number of sheep hours to be worked in the field or a deadline, but also a more specific condition which might need to be confirmed by both parties manually, *e.g.* the desired height of the grass.

Having agreed on a starting date, John can then bring his sheep to Rick's terrain. Animal transportation is subject to some legal regulations (Reglamento (CE) n° 1/2005, de 22/12/2004) [48], so, this process needs to be registered in the system. For this, sheep owners must have acquired a transportation authorization from the regulating entity, which certifies that the vehicle conditions are apt for loading and unloading the animals, there is enough available space and driver formation is appropriate [49]. Then, this authorization must be sent along with the transaction that signals the beginning and end of the movement. These transactions will also include information about which, and how many animals are being transported, along with the location of departure and destination. This way, the regulator can verify that the all the specifics of the travel went according to the regulations.

Having arrived at the field, the necessary hardware from the SheepIT project will be installed, which will allow the system to collect the data generated by the collars worn by the sheep. For this data to be directed from the hardware to the blockchain, the SheepIT gateway needs to be configured to be connected to the blockchain network. We're assuming that the hardware will be purchased by the sheep owner who is providing the service. So, this needs to be done in such a way that the owner of the field trusts that the sent data was not tampered with to favor the sheep owner, we will refer to this as **Problem 1**, and we will detail our solution in the next chapter.

It is important to note that even though the sheep will be registered in the system individually, they will likely be tracked as a group during the pastures, because the collars will not necessarily be



bound to specific animals, instead, they can be randomly distributed among the animals, since it could be troublesome to ensure a fixed collar-sheep relationship (identifying each sheep before or after putting the collar on its neck, in order to associate the data obtained from each collar to each animal).

Thankfully, only group metrics concern both parties in a land clearing service, namely the total amount of time spent by the herd of sheep inside the field. This data will then be accessible for both parties to visualize, to know how the service is progressing. However, if the sheep owner desires, he can guarantee the fixed sheep-collar association, to visualize the data about specific animals.

Once the contract finishing condition has been reached, the contract will be closed, and the payment must be made. The payment process will not go through the system, it will be made through traditional methods since this is not the focus of the work. It would bring extra complexity, more legal requirements to be able to handle money through the platform, and it wouldn't bring much value to the solution when compared to the payment methods available today such as bank transfer or cash in hand. To note that the contract can end in a disagreement between both parties, and therefore not reach its finishing condition.

Both parties are then allowed to leave a feedback comment on the other's profile, along with an evaluation of how satisfied they are with all the stages of the exchange. This will generate a public rating for each user, which will be an average of all the evaluations obtained in previous services. With this system, all users are incentivized to deliver a good service and behave in a business-friendly way, to obtain a better rating, and therefore a better reputation.

The health status of the sheep is certified by veterinarians who will be interested in being registered in the system to acquire clients through it. Sheep owners can check the veterinarian's profile to view their contact information. The scheduling of the visit is then made outside of the platform, what matters to our system is the resulting report that the vet will release in respect to the sheep owner's animal(s). This will impact their health status in the system, which will be visible to anyone who has enough permissions to access them. Having the good health of animals certified is extremely beneficial for the chances of a sheep owner to secure a land clearing deal, so, it is in their best interest to do so as soon as they join the platform.

The data collected during the pastures and animal movements will also be available for a regulating entity to be able to verify that the corresponding legal requirements are being met.

## **3.1 Requirements**

To accomplish the vision above, the system must provide these features:

- Registration of participants and assets - The system should allow users to become registered in the system and associate their respective assets with their profile.
- Listing management - The system should allow sheep owners to create and manage their listings through the platform, and field owners to send solicitations to them.
- Contract management - The system should allow sheep owners and field owners to create and manage the state of their land clearing contracts through the platform.
- Interface with SheepIT - The system must be able to retrieve the data generated with the SheepIT hardware, and direct it to our platform.
- Access Control - Different participants have different roles and access levels in the system, and therefore, the functionalities that each end-user has access to should depend on the role associated with its identity.
- Reputation - All the participants should have an associated reputation score, which will be based on past reviews and feedback gained through previous services with other users. This reputation score should be visible to all the other participants, which will make all users strive for delivering a good service, in order to achieve better scores, which in turn will naturally make users more likely to do business with them.
- Data integrity - Users must be able to trust that all the data they see in the system was in fact generated according to the procedure and was not made up or tampered with. Example - A user should always be able to trust that data such as previous feedback given to another user, or pasture data generated during a land clearing service is, in fact, real and never was tempered with.

## 3.2 Stakeholders

These are the users who will make use of the system

- Sheep Owner - This is the party that owns sheep, and expects to contractualize services with field owners in order to have the sheep clear their land of unwanted and invasive plants and weeds, in exchange for monetary compensation.
- Field Owner - The owner of one or more fields, which will make contracts with sheep owners, in order to have the fields cleared.
- Veterinary - Credited veterinarians may register into the system, which will allow them to provide regular health checkups to animals, along with any extraordinary checkup requested by any other

party. The results and data collected will be distributed in the Blockchain network for everyone with the necessary credentials to visualize. They will be represented by Direção-Geral de Alimentação e Veterinária (DGAV)

- The Regulating Authority (DGAV) - The regulating authority will be able to access the information generated in the system and verify that legislation is being followed in all aspects.

### 3.3 Data model

This section will display what the different entities consist of and how they are connected.

The system will be comprised of:

- The end users will call the available functions in order to interact with the blockchain, these will be the Sheep Owners, Field Owners, Veterinarians, and Regulator: All of these will have their basic personal information associated, such as their name and address.

The main actors (Field and sheep owner) will also have their respective assets registered, along with a list of their previous feedback, and the resulting rating which will always be visible to everyone in the platform.

- Listing - The listing is the entry that the sheep owner creates in which he publicizes his intent of engaging in land clearing deals, specifying the terms they request for a deal to be made. These terms consist of the price per sheep per hour and the number of animals available for the service. Listings will also have an attached status field, which will signal if the offer is open or closed.
- Solicitation - After having found an interesting contract offer, A field or sheep owner will be able to send a solicitation to the offer creator, in order to communicate their interest in the offer in question. A solicitation object consists of a reference to the contract offer in question, a reference to its sender, an optional comment (message) to go along, and a response field, which will be filled out when the solicitation is answered.
- Land Clearing Contracts - After having received solicitation(s) on a listing, the listing creator will be able to accept one of these, which will in turn bind both participants in a land clearing contract. This contract will have all the contract information agreed in the respective offer, along with an array of pastures, through which all of these will be accessible to the respective participants. The contract will also display the current progress, based on the number of hours “worked” vs the agreed total number of hours.

- Pasture - when a land clearing contract is active, the sheep owner can begin the service by bringing the sheep inside the field. Each animal will create a pasture entrance, associated with its ID, and will specify the starting time and finish time, which will make it possible to count the hours worked so far, and how far the contract is to the finish.
- Animal - The sheep used for land clearing services, associated with the respective owner. It will also have an associated health status, the date of the last checkup, and an identifier of the pasture instance it is working on if that is the case.
- AnimalTransportation - When the animals are to be transported, an instance of this concept should be created, in order for the regulator to be able to inspect, and confirm that legal procedures are followed.
- Field - The fields where the land clearing will take place, associated with the respective owner. It has address and size associated, in order to give more information to future business partners and help them decide on the feasibility of a potential contract. If there is an active contract in a field, its ID will be associated in this field.
- Feedback - When the contract has reached its ending, both participants will be able to leave feedback on the other party's profile, these feedback instances will be recorded, and identify the sender and receiver, the contract in question, along with the comment and the rating (1-5).
- Veterinary checkup - Sporadic or scheduled veterinary appointments for health checkups or other issues of specific animals, it identifies the veterinary, the animal, the date, and the resulting report, this will, in turn, affect the health status of the inspected animal.

According to the specification above, the data model shown in Figure 3.1 was built, to show the connection between the different entities, and the main data fields they are composed of.

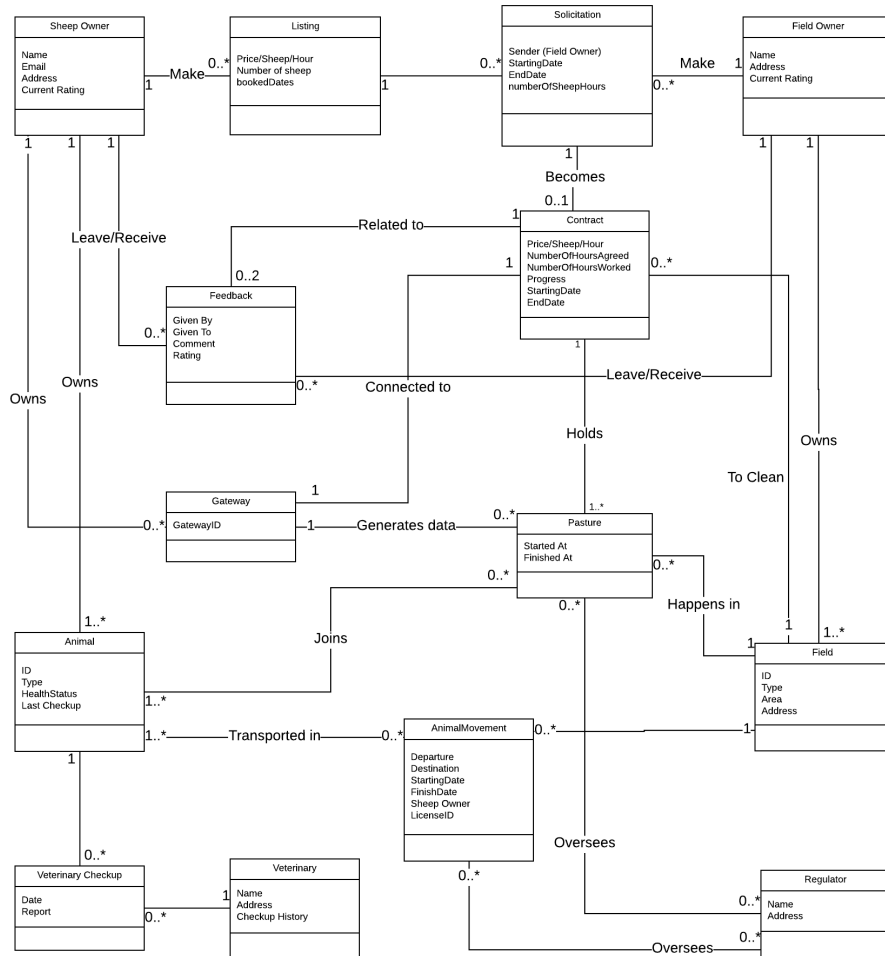


Figure 3.1: Data model

### 3.4 Access control

The information available to each participant should be given according to the principle of least privilege, in which each participant should only be given access to the information and resources that are necessary for its legitimate purpose [50]. With that in mind, different pieces of information given to and generated by this system should be available only to specific participants, depending on the case. The following table details these permissions according to the needs of our scenario.

	Net. Admin	Sheep owner	Field owner	Regulator	Gateways
Participants	RW - P	R - A	R - A	R - A	
Fields		R - A	R - A W - O	R - A	
Animals		R - A W - O	R - A	R - A	
Listings		R - A W - O	R - A		
Solicitations		R - P	RW - O		
Contracts		RW - O	RW - O		
Animal Mov.		RW - O		R - A	
Pastures		R - P	R - P	R - A	W - P
Gateways	RW - P	R - O			

Table 3.1: Devised access control permission table for our system

This table details what permissions each participant (on the top row) has over each data type that will be present in the system (left column), in the format of **ACTION - GROUP**, the available actions are **Read** (visualize), **Write** (create). These actions can be performed on one of three groups, **All** - meaning the action can be applied to any element of the data table, **Own** - meaning the action can only be applied to elements owned by the caller, and **Permissioned**, meaning the action can only be called if the caller meets certain conditions.

The following reasoning was used to come up with the permission table:

**Participant Information, Animal, and Field information:** this includes any details that are safe and in everyone’s best interest to share, in order to make use of the blockchain’s properties to guarantee the reputation of the participant. These details include the current rating, feedback history of participants, and public details about their owned assets (**animals** or **fields**). Some detailed information about these assets will still be permissioned, available only to the owner and possibly to the regulator and participants

engaged in contracts with the person to whom the information belongs.

The idea is to safely share as much information as possible, without sharing sensitive information that could hurt the owner. This will increase incentives to deliver a good service when a contract is agreed upon. Therefore all of the participants are able to (R)ead this information, while the sheep owner and field owner are also able to (W)rite animals and sheep respectively into their ownership.

About **Listings**, Sheep owners are allowed to (W)rite listings which will be associated with their ownership. Then, one might argue that these should be hidden from competitors - Should a sheep owner be able to see the listings made by other sheep owners? But in this case, a full disclosure approach is more reasonable, for all participants to be aware of the price offers at any moment, which will make the price to be dictated by supply and demand, which could, in turn, bring more users into the system looking for a good deal.

About **Solicitations** - The field owner is allowed to create them, and sheep owners allowed to see them, in the condition that they are the owner of the listing to which the solicitation was made to.

The most restrictive information is the Contracts, which are created when a sheep owner accepts a solicitation. The information about these contracts will be available only to the two parties involved in the contract, and no one else.

Animal movement entries are only able to be added by sheep owners, and they can be read only by themselves and the regulator, who will verify that legal standards are followed.

Pastures can only be written by the gateways, and the information about them is only visible to the regulator, and the sheep owner and field owner that belong to the contract associated with the pasture.

The gateways, as we said before, are also participants in our platform, as they will also directly interact with the network. As such, only the network admin is able to create them. They can then also be accessed by the sheep owner who becomes its owner.

The same goes for the information about the Pastures connected to these contracts, but the regulator will also have READ access to them in order to verify that the pastures are being held accordingly to legal standards.

The **Veterinary** - even though not shown in the table, is also given permission to WRITE or update the information about animals, mainly, the health status, but only the he is analyzing





# CHAPTER 4

---

## Implementation

---

In this chapter, we start by explaining the choices that we made in terms of technologies, architecture, and configuration to be used. Then, we show the work that was developed during the course of this dissertation.

### 4.1 Architectural choices

#### 4.1.1 Platform

Analyzing the options displayed in chapter 2, it shows that both Ethereum and Hyperledger bring their advantages and disadvantages, but looking into the general and technical requirements stated in the previous chapter, **Hyperledger**'s frameworks and tools make the most sense for the development of this project.

The main reasons for this start with the obvious lack of confidentiality in an Ethereum-based solution. If our solution was built in it, all the data flowing through it would be visible to anyone. This is a negative point because, for example, users would lose bargaining power when negotiating new deals, as the

prices of previously held contracts would be visible to the other party. On the other hand, Hyperledger's permissioned approach allows us to control who can access which data.

Another reason is the costs and ease of use associated with running the system. An Ethereum solution would require users to pay a fee for every single transaction they would make. To do that, every user would also need to set up an Ethereum wallet, which can be a challenging task for inexperienced users. On the other hand, Hyperledger's costs are only the ones associated with running the nodes on machines or servers. The onboarding of the users is a fairly easy process, and they do not need to know the technological implications or configurations necessary for them to be able to interact with the network, as all their transactions will be accessible with a regular REST API.

Ethereum is also known to be unpredictable because of its open nature. Spikes in network use can cause congestions or high transaction prices, as it has often happened in the past [51]. In this case, users could see their transactions stuck for hours or days, while only experienced users who would understand the problem behind the scenes could choose to pay exorbitant GAS fees to interact with the system, or simply wait for the average fee to lower to an acceptable amount. With Hyperledger this situation is a very unlikely possibility, as their throughput is more than enough for our requirements, and even on the occasion of congestion, the consequences would not be as serious.

That being said, the Hyperledger route is a better fit for our problem. With it, we can also make use of their tools, such as Composer, to facilitate the development process. This brings some limitations, such as the impossibility of having more than one channel holding the network [38], but we feel that even so, this is the best choice for proceeding with the development of the system.

## 4.1.2 Blockchain configuration

The configuration we aimed to build for our blockchain network is shown in Figure 4.1.

Taking into consideration our restrictions, a **single channel approach** will be taken for this project, where all the transactions will flow through. The confidentiality of the data will be guaranteed at the client level with **Access Control Logic**.

**3 Organizations** will take part, one representing the interests of the field owners, another representing the interests of sheep owners, we will refer to these as Associação Nacional de Operadores Agrícolas (ANOA) and Associação Nacional de Operadores de Pecuária (ANOP). And also DGAV which represents the Regulator and Veterinarian.

ANOA and ANOP will host 2 peers each for crash-tolerance purposes, while DGAV will host only one.

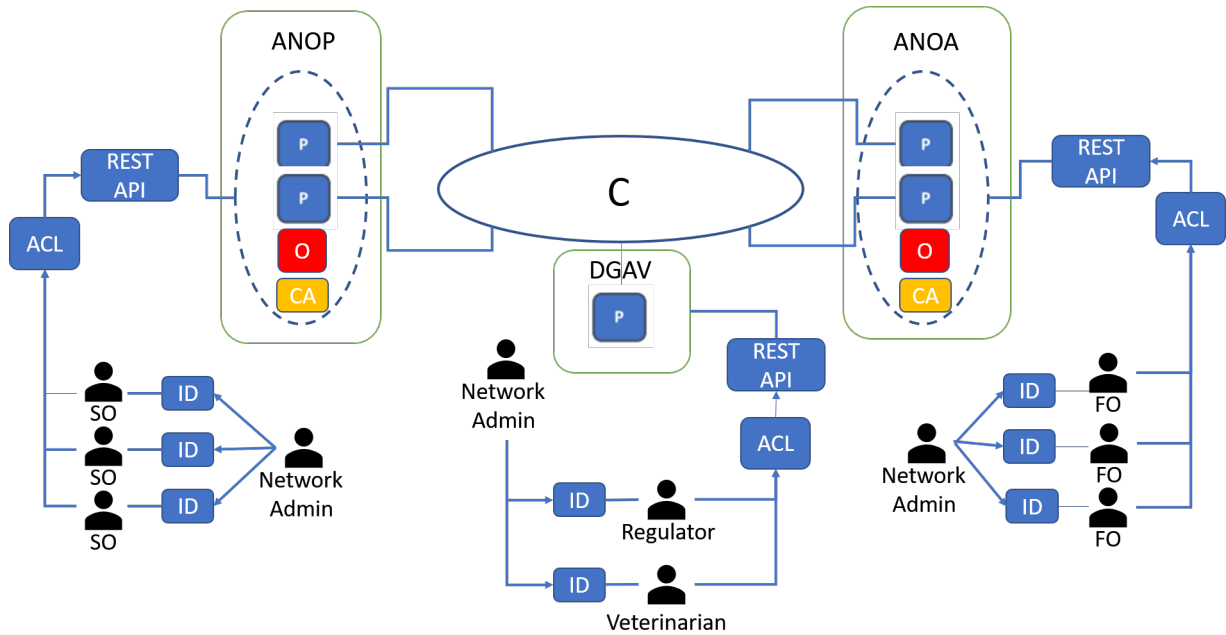


Figure 4.1: Devised configuration of the blockchain network - (C-Channel, P-Peer, O-Orderer, CA-Certificate Authority, SO- Sheep Owner, FO- Field Owner)

All of them will participate in the consensus process and hold a copy of the ledger.

The endorsement policy will state that, for any transaction to be validated, it will have to be endorsed by **one peer from each organization**. This way, no organization can manipulate the ledger by inserting fake transactions, since the peers of the opposing organization would instantly detect a false transaction and would not endorse it. Since all peers in a channel are running the same chaincode, a malicious organization would also not be able to purposely deny the endorsement of a valid transaction sent by the other.

**Ordering Service** - ANOA and ANOP will hold an ordering node that will participate in the ordering service, securing decentralization in the ordering process. The service will run on Kafka mode at the production stage in order to be fault-tolerant.

**A REST API** will be the interface from the blockchain to the end-users, this will provide a secure and efficient way for applications to retrieve and insert data into the blockchain.

**The End Users** - (Sheep owners, Field Owners, Regulator, Veterinarian) will only be able to join the network after obtaining a valid ID from the Network Administrator of their organization. After obtaining one, these clients will interact with the blockchain through web applications designed for their specific use cases, which in turn will interface with the blockchain through the endpoints provided by the REST API.

**Network Admins** - When a business network is started, the business network must be configured with a set of initial participants. These participants are responsible for bootstrapping the business network and onboarding other participants into the business network. In HLC, these participants are called network administrators.

These participants hold some administrative permissions, such as onboarding other participants by supplying them with a valid ID for joining the network. These ID's are generated from the Certificate Authority (CA) associated with their organization. Each organization will have its network administrator. HLC also grants the option for network admins to onboard more participants who hold permission to onboard other participants. This could be useful in case of the project evolving into a scale where more than one network admin might be needed in an organization.

The Network Admin should verify all the available information about the participant and the assets they own (criminal registry, health status of animals, etc.) before accepting them into the system. A network admin could possibly maliciously accept fraudulent participants into the system, but this would not be in their, or their organization's best interest, since it could be quickly uncovered, as soon as the fraudulent participant would participate in a contract, delivering a lower quality service than the other party would expect. For this reason, it is reasonable to assign this role to a person with a respectable level of responsibility in each organization, looking for their best interests.

**Certificate Authority** - Each organization will run a CA which will provide ID's which will undeniably identify anyone who wishes to participate in the network. This makes it so that access control is enforced, regardless of the location of the peer. A peer could be hosted in a physical machine owned by Org1, but if it was associated with Org2's CA, it would be still owned and only accessible by Org2.

**Peer Admins** - Peers admins are not participants, instead, they are the ones who take care of initializing the network on each organization. This role can add/remove peers, deploy chaincode, create and join channels, etc. on behalf of that organization [52]. This does not mean the peer admin could act maliciously though since they can only make changes in the configuration of their side of the network on their own, any global changes to the network, such as installing chaincode in a channel, need to be agreed by all the peers in the network [52]. For this reason, it does not matter who exactly will take the role of peer admin in their organization, as long as it is someone with the required technical skills.

### 4.1.3 Hosting

To bring up the components of the whole network, there are a few different options available:

- Hosting everything in the same machine - suitable for a proof of concept solution but shouldn't be used in a production environment.
- Hosting the components of each organization in separate machines - Suitable for production, this would be the way to go for each organization to contribute resources to the network and rest assured that these haven't been tampered with.
- Using the IBM cloud services to host the network. They provide solutions for different project requirements, where isolation is provided for the Fabric component nodes in a network, and each node executes in a secure Docker container that protects the execution environment [53]. These services come in a **Pay-As-You-go** business model, where more resources can be requested for a fee (Figure 4.2).

The **Starter Plan** is perfect for a proof of concept, even for simulating a multi-organization blockchain, but it comes with some limitations, such as only providing the network with one orderer node, which runs in SOLO mode, making the ordering service fault-tolerant [54].

The **Enterprise Plan** is a production-ready offering for organizations that would like to create or join a blockchain network for real businesses. The plan provides the key infrastructure along with tools and support to easily start a highly secure and production-ready network [53].

Pricing elements	Starter Plan cost per month	Enterprise Plan cost per month
Membership fee	\$250	\$1000
Peer fee	\$125	\$1000

Figure 4.2: IBM Blockchain Platform pricing overview [55]

**Membership fee** - Covers organization creation, access to ordering service and CA, and is charged on a per-instance basis. Included in this pricing element, the IBM Blockchain Platform handles the ordering service and the CA on the network's behalf. This fee is required to have access to a network built on the IBM Blockchain Platform.

**Peer fee** - Covers an organization's peers and is charged on a per-peer basis. This fee is required only if it is desired to have a peer maintain a copy of the ledger and validate transactions.

The **Starter Plan** allows for unlimited organizations per membership and the ability to switch between organizations in the Network Monitor. Because the Starter Plan is designed for development, test, and Proof-of-Concept environments, it is possible to simulate multi-organization environments. The total

network storage is capped at 20GB, including the components, chaincode, and ledger data. The simulated organizations share the 20GB storage in the blockchain network.

The **Enterprise plan** allows for a single organization per membership. Because Enterprise is designed for pilot and production environments, each account is linked to their specific organization.

Analyzing these options, it looks like any of the options from the IBM cloud services are more appropriate to a project with a business model generating enough profit to make the payment of the fees worth its value. In the case of this project, we will simply host everything in the same machine as a proof-of-concept, however, we will follow the same steps that would be taken in order to spread the network over multiple machines.

## 4.2 Building the prototype

### 4.2.1 Creating the network

Taking all the previous aspects into consideration, a proof of concept of the system shown in figure 4.1 was created. For this, we followed the Build Your First Network [56] and multi-organizations deployment [57] guides, which allowed us to quickly set up a local HLF instance, upon which we ran all of the necessary components inside Docker containers. With this approach, all of our organizations' (ANOA, ANOP and DGAV) fabric networks are running on the same machine as a proof of concept. In the real world, they would in separate IP networks or domains, or secure Cloud environments.

To start with, a JavaScript Object Notation (JSON) configuration file - the connection profile [58] - had to be created in which we detail all the necessary information for the network to be built *i.e.* the information about the channel, and which components are connected belong to it. The participating organizations, which peers they hold, and the associated CA that is used to onboard participants into it. And finally, the information about each of the components (orderer, peers, CAs), most importantly, the IP and ports that they use to communicate with each other using the gRPC [59] protocol.

```
1 {
2   "channels": {
3     "mychannel": {
4       "peers": {
5         "peer0.ANOP.example.com": {
6           "endorsingPeer": true,
```

```

7         "chaincodeQuery": true,
8         "eventSource": true
9     },
10    "peer1.ANOP.example.com": {
11        "endorsingPeer": true,
12        "chaincodeQuery": true,
13        "eventSource": true
14    },
15    "peer0.ANOA.example.com": {
16        "endorsingPeer": true,
17        "chaincodeQuery": true,
18        "eventSource": true
19    },
20    "peer1.ANOA.example.com": {
21        "endorsingPeer": true,
22        "chaincodeQuery": true,
23        "eventSource": true
24    },
25    "peer0.DGAV.example.com": {
26        "endorsingPeer": true,
27        "chaincodeQuery": true,
28        "eventSource": true
29    }
30 }
31 }
32 },
33 "organizations": {
34     "ANOP": {
35         "peers": [
36             "peer0.ANOP.example.com",
37             "peer1.ANOP.example.com"
38         ],
39         "certificateAuthorities": [
40             "ca.ANOP.example.com"

```

```

41     ]
42 },
43 "ANOA": {
44     "peers": [
45         "peer0.ANOA.example.com",
46         "peer1.ANOA.example.com"
47     ],
48     "certificateAuthorities": [
49         "ca.ANOA.example.com"
50     ]
51 },
52 "DGAV": {
53     "peers": [
54         "peer0.DGAV.example.com",
55     ],
56     "certificateAuthorities": [
57         "ca.DGAV.example.com"
58     ]
59 }
60 },
61 "orderers": {
62     "orderer.example.com": {
63         "url": "grpcs://localhost:7050",
64         "tlsCACerts": {
65             "pem": "ORDERER_CA_CERT"
66         }
67     }
68 },
69 "peers": {
70     "peer0.ANOP.example.com": {
71         "url": "grpcs://localhost:7051",
72         "tlsCACerts": {
73             "pem": "ORG1_CA_CERT"
74         }

```



```

75     },
76     "peer1.ANOP.example.com": {
77         "url": "grpc://localhost:8051",
78         "tlsCACerts": {
79             "pem": "ORG1_CA_CERT"
80         }
81     },
82     "peer0.ANOA.example.com": {
83         "url": "grpc://localhost:9051",
84         "tlsCACerts": {
85             "pem": "ORG2_CA_CERT"
86         }
87     },
88     "peer1.ANOA.example.com": {
89         "url": "grpc://localhost:10051",
90         "tlsCACerts": {
91             "pem": "ORG2_CA_CERT"
92         }
93     },
94     "peer0.DGAV.example.com": {
95         "url": "grpc://localhost:11051",
96         "tlsCACerts": {
97             "pem": "DGAV_CA_CERT"
98         }
99     }
100 },
101 "certificateAuthorities": {
102     "ca.ANOP.example.com": {
103         "url": "https://localhost:7054",
104         "caName": "ca-org1",
105         "httpOptions": {
106             "verify": false
107         }
108     },

```

```

109     "ca.ANOA.example.com": {
110         "url": "https://localhost:9054",
111         "caName": "ca-org2",
112         "httpOptions": {
113             "verify": false
114         }
115     },
116     "ca.DGAV.example.com": {
117         "url": "https://localhost:11054",
118         "caName": "ca-dgav",
119         "httpOptions": {
120             "verify": false
121         }
122     }
123 }
124 }

```

Snippet 4.1: The connection profile used to build up our network

We can see that in our proof-of-concept, all the components are running on the localhost, but in a scaled-up version, we would input the IP of the separate machines. We must note that we had problems configuring the ordering service in kafka mode as we devised in the architecture shown in figure 4.1, so we resorted to installing one single ordering service for this proof of concept.

The flags shown in the configuration of the peers mean that we are setting each peer to be able to endorse transactions (`endorsingPeer`), be able to handle chaincode query requests (`chaincodeQuery`), and set off events (`eventSource`) which we may want to consume in the future.

The content of the `tlsCACerts` tags are then substituted by CA certificates that can be generated with a `cryptogen` tool provided in the guide.

```

1 $ cryptogen/config/peerOrganizations/ANOA.example.com/peers/peer0.ANOA.
  example.com/tls/ca.crt

```

Snippet 4.2: Accessing the CA for ANOA

This is obviously only provided as a means of simplifying the setup for the guide, and never in a production context, as this would allow the peer admin to access other organization's secret key. In a

production environment, the certificates would be obtained instead through interacting with the Fabric CA.

Then, the BNCs for each of the organization's Network Admin were created using the `composer card create` command. These cards allow the admins to deploy the blockchain business network to their Hyperledger Fabric network.

```
1 $ composer card create -p /tmp/composer/ANOA/byfn-network-ANOA.json -u
PeerAdmin -c /tmp/composer/ANOA/Admin@ANOA.example.com-cert.pem -k /
tmp/composer/ANOA/*_sk -r PeerAdmin -r ChannelAdmin -f PeerAdmin@byfn
-network-ANOA.card
```

#### Snippet 4.3: Creating the BNC for the Peer Admin of ANOA

```
1 $ composer card create -p /tmp/composer/ANOP/byfn-network-ANOP.json -u
PeerAdmin -c /tmp/composer/ANOP/Admin@ANOP.example.com-cert.pem -k /
tmp/composer/ANOP/*_sk -r PeerAdmin -r ChannelAdmin -f PeerAdmin@byfn
-network-ANOP.card
```

#### Snippet 4.4: Creating the BNC for the Peer Admin of ANOP

With the cards created, they were imported into the Admin's wallets using the `composer card import` command.

```
1 $ composer card import -f PeerAdmin@byfn-network-ANOA.card --card
PeerAdmin@byfn-network-ANOA
```

#### Snippet 4.5: Importing the BNC for the Peer Admin of ANOA

```
1 $ composer card import -f PeerAdmin@byfn-network-ANOP.card --card
PeerAdmin@byfn-network-ANOP
```

#### Snippet 4.6: Importing the BNC for the Peer Admin of ANOP

Having done this, the business network could be installed onto all of HLF peer nodes. For this, first, all of the business logic needed to be created, and then bundled into a `.bna` file, ready to be deployed on the network. The necessary files to be created were: the **Model** file (`.cto`), the **Script** file (`.js`) and the **Access control** (`.acl`) file. Snippets of an entry from each of these files are given below, a more detailed specification of all the business logic work developed is given in the next section.

First off, the .cto file is where all the data must be modeled, along with the specification of the transactions that the network has (Inputs and Outputs).

```
1 asset AnimalTransportation identified by transportationID{
2   o String transportationID
3   --> Shepherd animalOwner
4   --> Animal[] animals
5   o String licenseNumber
6   o Location departureLocation
7   o Location destinationLocation
8   o DateTime startingTime
9   o DateTime finishingTime optional
10 }
```

Snippet 4.7: Model definition for the AnimalTransportation asset

The **Script(.js)** file, which holds all the functions that carry the logic that runs when the respective transaction is invoked. An example of a function is shown in the following snippet.

```
1 async function ShepherdCreateListing(transaction) {
2   const NS = "org.example.mynetwork";
3   const factory = getFactory();
4   const ListingRegistry = await getAssetRegistry(NS + ".Listing");
5
6   const listing = factory.newResource(NS, "Listing", generateID("1"));
7   const transactionSender = factory.newRelationship(NS,
8     getCurrentParticipant().getType(), getCurrentParticipant().
9     getIdentifier());
10
11   listing.listingCreator = transactionSender;
12
13   listing.pricePerSheepPerHour = transaction.pricePerSheepPerHour;
14   listing.numberOfSheepAvailable = transaction.numberOfSheepAvailable;
15   listing.solicitations = [];
16   listing.bookedDates = "";
17   listing.listingStatus = "OPEN";
18   await ListingRegistry.add(listing); // add the listing to the listing
```

```

17 Registry
18 }

```

Snippet 4.8: Code for the Create Listing transaction

The **Access Control (.acl)** file was then created to control what data or actions each different participant role is authorized to see or do. The following snippet shows an example of a rule imposed on the sheep owners.

```

1 rule ShepherdCanOnlySeeTheirContracts {
2     description: "Sheep owners can only see their own contracts"
3     participant(t): "org.example.mynetwork.Shepherd"
4     operation: READ
5     resource(v): "org.example.mynetwork.LandClearingContract"
6     condition: (v.shepherd.getIdentifier() == t.getIdentifier())
7     action: ALLOW
8 }

```

Snippet 4.9: ACL rule to make sure that Shepherds can only see their own contracts

The 3 files containing the definition of all of the other assets, transaction logic, and access control were then packaged together to create the **.bna** file, which was installed on the network with the command `composer network install`.

```

1 $ composer network install --card PeerAdmin@byfn-network-ANOA --
  archiveFile sheepIT.bna

```

Snippet 4.10: Installing the business network onto the peers of ANOA

```

1 $ composer network install --card PeerAdmin@byfn-network-ANOP --
  archiveFile sheepIT.bna

```

Snippet 4.11: Installing the business network onto the peers of ANOP

Afterwards, the endorsement policy had to be configured. This policy states the necessary condition for a transaction to be endorsed so that it can be committed to the blockchain.

```

1

```

```
2 {
3   "identities": [
4     {
5       "role": {
6         "name": "member",
7         "mspId": "ANOAMSP"
8       }
9     },
10    {
11      "role": {
12        "name": "member",
13        "mspId": "ANOPMSP"
14      }
15    },
16    {
17      "role": {
18        "name": "member",
19        "mspId": "DGAVMSP"
20      }
21    }
22  ],
23  "policy": {
24    "2-of": [
25      {
26        "signed-by": 0
27      },
28      {
29        "signed-by": 1
30      },
31      {
32        "signed-by": 2
33      },
34    ]
35  }
```

### Snippet 4.12: Endorsement policy for our network

In our case, we configured our policy to require 2 of the organizations to endorse a transaction in order for it to be accepted. This way no organization can dictate whether a transaction should be added, or blocked on its own.

The final step before initializing our network was to generate the security artifacts for our network administrators to be able to interact with the network once it is up. In our business network, both organizations have a network administrator. The admin for ANOA is called Alice, while the admin for ANOP is called Bob.

```
1 $ composer identity request -c PeerAdmin@byfn-network-ANOA -u admin -s
  adminpw -d alice
```

Snippet 4.13: Retrieving certificates for Alice to use as the business network administrator for ANOA:

```
1 $ composer identity request -c PeerAdmin@byfn-network-ANOP -u admin -s
  adminpw -d alice
```

Snippet 4.14: Retrieving certificates for Bob to use as the business network administrator for ANOP:

Then, the network could be finally started with the command `composer network start`.

```
1 $ composer network start -c PeerAdmin@byfn-network-ANOA -n sheepIT -V
  0.2.6-deploy.1 -o endorsementPolicyFile=/tmp/composer/endorsement-
  policy.json -A alice -C alice/admin-pub.pem -A bob -C bob/admin-pub.
  pem
```

### Snippet 4.15: Starting the network

This command successfully starts the network, figure 4.3 shows the output of the command, showing the creation of the 2 network admin cards, ready to be used to interact with the network.

```
adriano@adriano-laptop:~/fabric-dev-servers/fabric-samples/first-network$ composer network
start -c PeerAdmin@byfn-network-org1 -n sheepit -V 0.2.6-deploy.1 -o endorsementPolicyFile=
/tmp/composer/endorsement-policy.json -A alice -C alice/admin-pub.pem -A bob -C bob/admin-p
ub.pem
Starting business network sheepit at version 0.2.6-deploy.1

Processing these Network Admins:
  userName: alice
  userName: bob

✓ Starting business network definition. This may take a minute...

Successfully created business network cards:
  filename: alice@sheepit.card
  filename: bob@sheepit.card

Command succeeded
```

Figure 4.3: Business network successfully started

The connection can be tested with the `composer network ping` command, figure 4.4 shows the output.

```
adriano@adriano-laptop:~/fabric-dev-servers/fabric-samples/first-network$ composer network
ping -c alice@sheepit
The connection to the network was successfully tested: sheepit
  Business network version: 0.2.6-deploy.1
  Composer runtime version: 0.19.20
  participant: org.hyperledger.composer.system.NetworkAdmin#alice
  identity: org.hyperledger.composer.system.Identity#74ef49eceab083202b989c137a1a1ea4
1c9183b537485ab7332d222e1b5992e8
```

Figure 4.4: Pinging the network

The network was finally ready to be interacted with through the composer command line interface. With it, we also launched the REST API that enables our network to become accessible to outside applications. We did this with the command `composer-rest-server`, which easily takes care of generating the REST server, and makes it available on the `localhost:3000`, making all the defined transactions ready to invoked through automatically generated endpoints.

Figure 4.5 below shows an example of an endpoint available through the REST interface, in this case, related to the Listing asset.



### org\_example\_mynetwork\_Listing : An asset named Listing

GET	/org.example.mynetwork.Listing
POST	/org.example.mynetwork.Listing
GET	/org.example.mynetwork.Listing/{id}
HEAD	/org.example.mynetwork.Listing/{id}
PUT	/org.example.mynetwork.Listing/{id}
DELETE	/org.example.mynetwork.Listing/{id}

Figure 4.5: REST endpoints for the Listing asset

At this point, all the necessary components for the functioning of the network were available. The only missing component was a web interface to deliver to the end-users. Since this was not the main focus of our work, we opted to simply generate a skeleton web interface with the use of Yeoman, which is also integrated in the deployment pipeline of a HLC network. A run of the command `yo hyperledger-composer:angular` generates an interface on `localhost:4200` according to the specified REST API. As an example, the following figure shows the interface to be used for sheep owners to create a Listing asset.

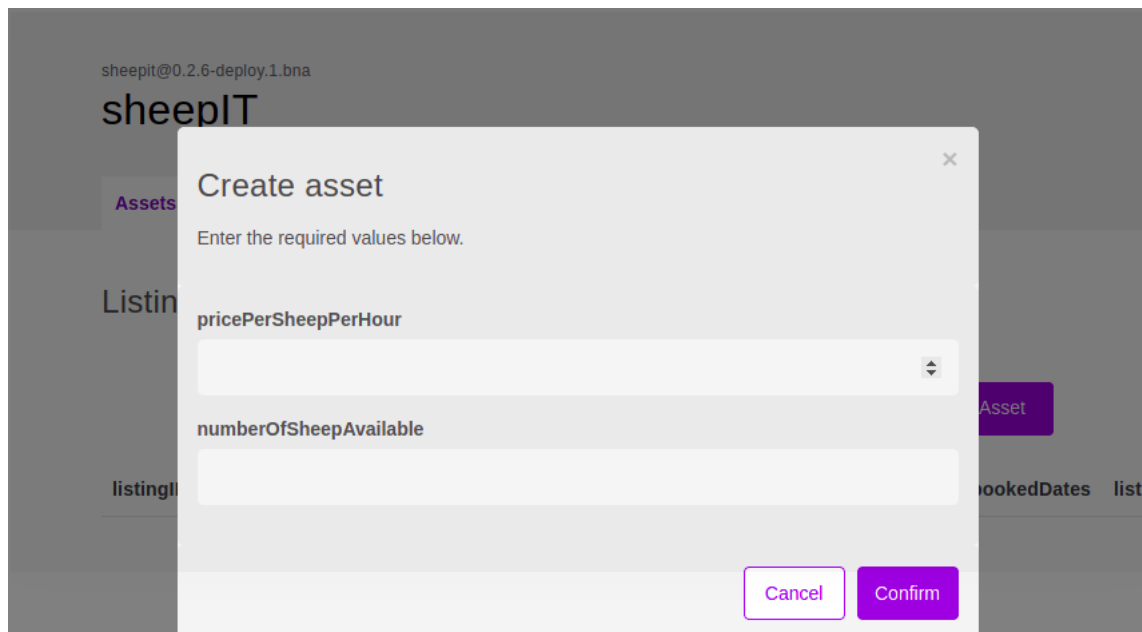


Figure 4.6: Web interface generated with Yeoman

## 4.2.2 Business logic

The business logic is the core of the functionality of the network since it defines what the network is composed of, and how it behaves with any stimulation. As we stated in the previous section, we needed to create a .bna file to start the network. This .bna file is composed of the model (.cto), script (.js) and access control (.acl) files.

We now detail the work we developed in all of these files by presenting the network model we built, in terms of which participants and assets it is composed of, followed by the description of all of our transactions, and who is allowed to invoke them, restricted by the access control logic we developed.

Starting with the participants, these are all the entities allowed to interact with the blockchain. For this, they need to be onboarded by their respective network admins, which will provide them a BNC which they can use to interact with the network.

- Shepherd - The sheep owner participant.
- FieldOwner - The field owner participant.
- Regulator - The regulator participant.
- Gateway - The sheepIt gateways. We treated them as participants as well, since they'll be invoking the transactions associated with the animal activity during the pastures.

The assets are the representation of tangible or intangible elements whose properties are affected by the transactions invoked by the participants, effectively representing the state of the network.

- Field - Field owners have field assets associated with them. Detailed information about these is only accessible to the owner.
- Animal - Sheep owner have animal assets associated with them. Detailed information about these is only available to the owners.
- Listing - The listings created by sheep owners. Visible to all the sheep owners and field owners.
- LandClearingContract - Contracts representing agreements between sheep and field owners, each contract is accessible only to the participants involved.
- Pasture - Pasture instances are created every time an animal enters a field, and are closed when it leaves. Pasture records are visible to the sheep owner and field owner involved in the respective contract, and also the regulator.

- AnimalTransportation - Every time an animal transportation happens, it must be reported with the respective transaction, creating an AnimalTransportation entry in the registry. These records are visible only to the owner of the animals and the regulator.

The transaction headers must be declared in this file, in which we specify the inputs it needs to receive. For simplicity's sake, we also detail here the functioning of each transaction, which was written in the .js file.

- RegisterField - This transaction receives a field asset as input, and sets its "owner" property to match the participant ID of the sender of the transaction. This transaction is available only to field owners.
- RegisterAnimal - This transaction receives an animal asset as input, and sets its "owner" property to match the participant ID of the sender of the transaction. This transaction is available only to sheep owners.
- RegisterGateway - This transaction receives the id of the gateway, along with the ID of the sheep owner to set as its owner. To address Problem 1, this transaction is available only to network admins. With this they are responsible for creating the Gateway participant and the respective BNC, and then install it onto the gateway. This configuration needs to be protected to not be accessible even by the owner of the gateway, because otherwise, he could send transactions on the gateway's behalf. By having the ID of its owner associated with it, the gateway will then be able to receive commands from this user only.
- DirectGateway - This transaction receives as inputs the land clearing contract ID, and the gateway ID. It sets the LandClearingContract field of the Gateway to the input parameter. This transaction is available only to sheep owners, and they can only direct commands to the gateways they own.
- ShepherdCreateListing - This transaction receives a pricePerSheepPerHour and a numberOfSheepAvailable as inputs, and it creates an entry in the listing registry with these parameters and sets its "creator" property to match the participant ID of the sender of the transaction. This transaction is available only to sheep owners.
- FieldOwnerSendSolicitation - This transaction receives the listing that the field owner is sending a solicitation to, and the field in which he intends on conducting the land clearing as inputs, along with the optional startingDate, endDate, and amountOfSheepHours fields. It creates a solicitation entry in the registry with the specified parameters and sets its "creator" property to match the participant ID of the sender of the transaction. This transaction is available only to field owners.

- ShepherdAnswerSolicitation - This transaction receives the solicitation asset and a boolean answer as inputs. It sets the status of the respective solicitation to accepted or rejected, depending on the answer. If the answer is positive, a LandClearingContract is created in the registry with the details of the listing associated with this solicitation, setting the FieldOwner field to the creator of the solicitation, and Shepherd field to the participant ID of the sender of the transaction. This transaction is available only to sheep owners.
- BeginAnimalTransportation - This transaction receives as inputs the licenseNumber of the license that the shepherd is animal transporter is required to have, the array of animals being transported, and the GPS coordinates of the departureLocation and destinationLocation. It creates an AnimalTransportation entry in the registry with the specified parameters set and sets the starting time to the moment the transaction was called, and the animalOwner to match the participant ID of the sender of the transaction. This transaction is available only to sheep owners.
- finishAnimalTransportation - This transaction receives only the AnimalTransportation ID as input. It sets its status to FINISHED, and the finishing time to the timestamp of the transaction. This transaction is available only to sheep owners.
- ShepherdInitsContract - This transaction receives as input the land clearing contract ID, it sets the SOInitiated flag of the contract to true, if the FOInitiated flag is already set to true, this transaction sets the status flag of the contract to ONGOING. This transaction is available only to sheep owners, and they can only be called upon contracts in which they are involved.
- FieldOwnerInitsContract - This transaction receives as input the land clearing contract ID, it sets the FOInitiated flag of the contract to true, if the SOInitiated flag is already set to true, this transaction sets the status flag of the contract to ONGOING. This transaction is available only to field owners, and they can only be called upon contracts in which they are involved.
- AnimalEntersField - Takes as input the ID of the collar that the animal that entered the field is wearing. This transaction creates a pasture instance in the registry, indicating that the collar entered the field at the time when the transaction was submitted. Only the gateway can call this transaction, which happens automatically when the gateway detects this event.
- AnimalLeavesField - Takes as input the ID of the collar that the animal that left the field is wearing. This transaction closes the pasture instance associated with the animal with the input collar ID. By doing so, the total time spent by the animal inside the field is calculated with the time difference between this and the AnimalEntersField transactions. This time difference is added to the variable amountOfSheepHoursWorked in the contract the gateway is associated with. Only the gateway can call this transaction, which happens automatically when the gateway detects this event.

- **ShepherdFinishesContract** - This transaction receives as input the land clearing contract ID, it sets the **SOFinished** flag of the contract to **true**, if the **FOFinished** flag is already set to true, this transaction sets the status flag of the contract to **FINISHED**. This transaction is available only to sheep owners, and they can only be called upon contracts in which they are involved.
- **FieldOwnerFinishesContract** - This transaction receives as input the land clearing contract ID, it sets the **FOFinished** flag of the contract to **true**, if the **SOFinished** flag is already set to true, this transaction sets the status flag of the contract to **FINISHED**. This transaction is available only to field owners, and they can only be called upon contracts in which they are involved.
- **ShepherdSendsFeedback** - This transaction receives as inputs the feedback rating, on a 0 to 100 scale, the comment, and the ID of the field owner they are sending the feedback to. It associates the rating and the comment to the field owner's profile. This transaction is available only to sheep owners, and they can only invoke this transaction once, on a contract they are involved, with the **FINISHED** status.
- **FieldOwnerSendsFeedback** - This transaction receives as inputs the feedback rating, on a 0 to 100 scale, the comment, and the ID of the sheep owner they are sending the feedback to. It associates the rating and the comment to the sheep owner's profile. This transaction is available only to field owners, and they can only invoke this transaction once, on a contract they are involved, with the **FINISHED** status.



## CHAPTER 5

---

### Results

---

In order to test our platform, we make use of the Composer Playground tool, which enables us to test all aspects of the network, by allowing us to easily create assets and participants, issue ID cards, and assign them to participants, which allow us to interact with the blockchain on their behalf. This way we can impersonate each user and invoke the available transactions for their role, and instantly check the resulting changes in the current state of assets and participants (state database) in accordance to the access control restrictions we created.

The system will be tested according to John and Rick's user story mentioned in chapter 3.

Here, the network admins onboard John and Rick onto the platform by invoking this Create New Participant transaction, where the user basic information is input.

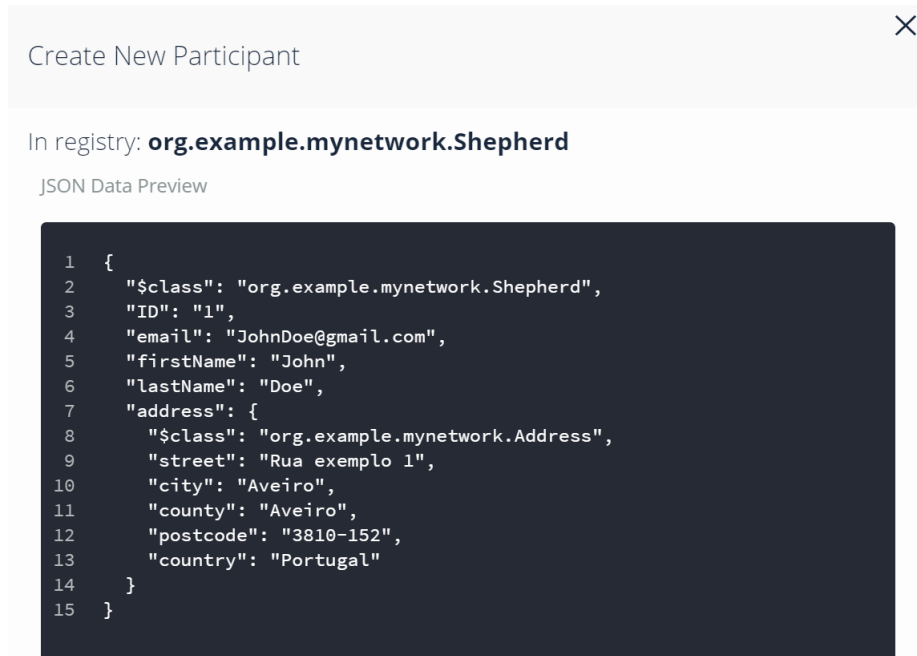


Figure 5.1: Network admin registering John



Figure 5.2: Network admin registering Rick



John and Rick are added to the participant registry, with the specified parameters from the transaction. The fields rating, feedbackHistory and activeContracts are also initialized to the values shown below.

Define Test admin

Participant registry for org.example.mynetwork.Shepherd + Create New Participant

ID	Data
1	<pre>{   "\$class": "org.example.mynetwork.Shepherd",   "animals": [],   "ID": "1",   "email": "JohnDoe@gmail.com",   "firstName": "John",   "lastName": "Doe",   "address": {     "\$class": "org.example.mynetwork.Address",     "street": "Rua exemplo 1",     "city": "Aveiro",     "county": "Aveiro",     "postcode": "3810-152",     "country": "Portugal"   },   "rating": 100,   "feedbackHistory": [],   "activeContracts": [] }</pre>

Collapse

Figure 5.3: John was added to the Shepherd registry

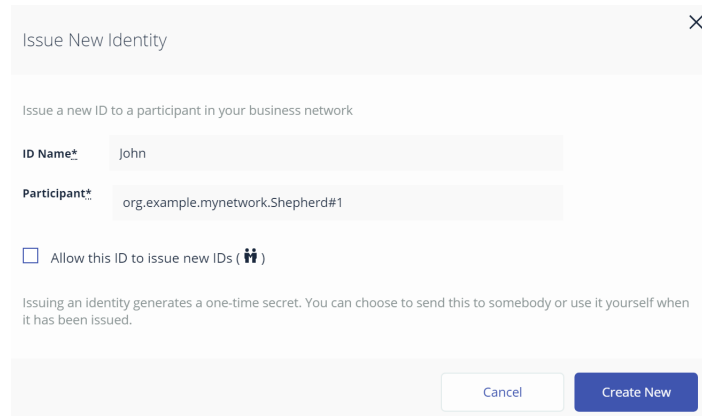
Define Test admin

Participant registry for org.example.mynetwork.FieldOwner + Create New Participant

ID	Data
1	<pre>{   "\$class": "org.example.mynetwork.FieldOwner",   "fields": [],   "ID": "1",   "email": "RickAstley@gmail.com",   "firstName": "Rick",   "lastName": "Astley",   "address": {     "\$class": "org.example.mynetwork.Address",     "street": "Avenida exemplo",     "city": "Porto",     "county": "Porto",     "postcode": "",     "country": "Portugal"   },   "rating": 100,   "feedbackHistory": [],   "activeContracts": [] }</pre>

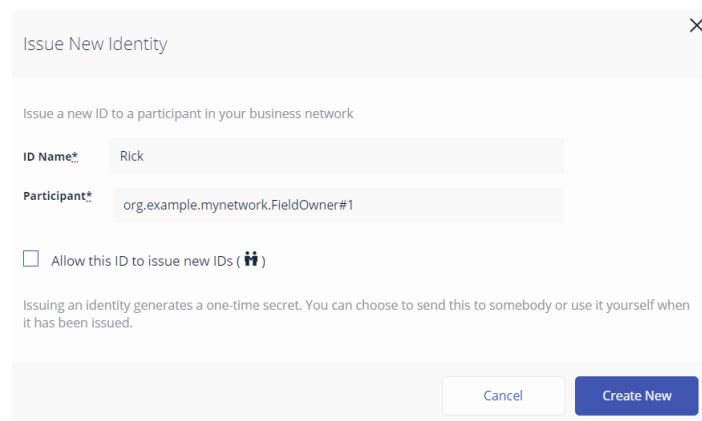
Figure 5.4: Rick was added to the Field owner registry

Then, the network admins of each organization issue ID's for John and Rick. They must provide the name of the identity, and the participant which this ID will be associated with. Also, a checkbox is available, which signals if the issued ID will be allowed to issue new IDs as well. In our case, we only want the network admins to be able to do that, so we'll leave it unchecked.



The screenshot shows a dialog box titled "Issue New Identity" with a close button (X) in the top right corner. Below the title is the instruction "Issue a new ID to a participant in your business network". There are two input fields: "ID Name\*" containing "John" and "Participant\*" containing "org.example.mynetwork.Shepherd#1". Below these fields is a checkbox labeled "Allow this ID to issue new IDs (👤)" which is unchecked. A note below the checkbox states: "Issuing an identity generates a one-time secret. You can choose to send this to somebody or use it yourself when it has been issued." At the bottom right, there are two buttons: "Cancel" and "Create New".

Figure 5.5: Network admin issuing an ID to John



The screenshot shows a dialog box titled "Issue New Identity" with a close button (X) in the top right corner. Below the title is the instruction "Issue a new ID to a participant in your business network". There are two input fields: "ID Name\*" containing "Rick" and "Participant\*" containing "org.example.mynetwork.FieldOwner#1". Below these fields is a checkbox labeled "Allow this ID to issue new IDs (👤)" which is unchecked. A note below the checkbox states: "Issuing an identity generates a one-time secret. You can choose to send this to somebody or use it yourself when it has been issued." At the bottom right, there are two buttons: "Cancel" and "Create New".

Figure 5.6: Network admin issuing an ID to John

As we can read in the image, "Issuing an identity generates a one-time secret. You can choose to send this to somebody or use it yourself when it has been issued". The admins would then send these credentials to John and Rick, which would finally allow them to interact with the blockchain as the respective participants created before. (Shepherd#1 and FieldOwner#1).

The gateway is also a participant from our system's perspective, as it will be the entity responsible for invoking the transactions related to the pastures, shown further ahead in this demo. The following transaction shows the RegisterGateway transaction, which is called by the network admin. In it he inputs the gatewayID and the ID of the sheep owner which will be its owner. This creates an entry in the gateway registry with the specified parameters.

```
1 {
2   "$class": "org.example.mynetwork.RegisterGateway",
3   "gatewayID": "gw12345",
4   "owner": "resource:org.example.mynetwork.Shepherd#1"
5 }
```

Figure 5.7: Transaction that registers a gateway

ID	Data
gw12345	<pre>{   "\$class": "org.example.mynetwork.Gateway",   "gatewayID": "gw12345",   "owner": "resource:org.example.mynetwork.Shepherd#1" }</pre>

Figure 5.8: Gateway registered

A BNC must be provided for the gateway to be able to interact with the blockchain. It is important to notice that we must prevent John from having access to this BNC even though he is the owner of the gateway. If he had access to it, he could invoke the transactions that are reserved for the gateway, with which he could act maliciously, for example by adulterating the record of the total time spent by his animals inside a field.

In the images below we show the network admin issuing the ID for the gateway, and also the result from John trying to do the same.

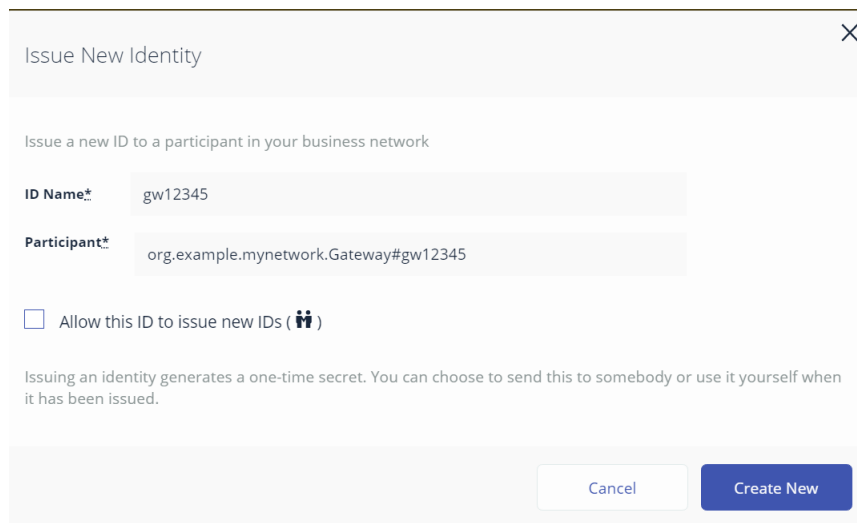


Figure 5.9: Network admin issuing an ID for the gateway

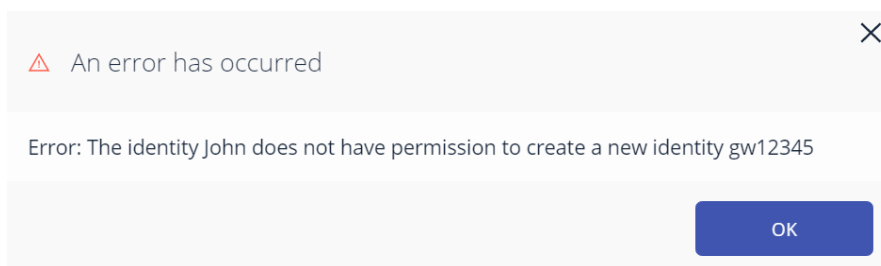


Figure 5.10: John trying to issue a BNC for a gateway

The participant creation and ID issuing process was also followed for a Regulator, so that we can visualize the system through his role. Then, having created all the ID's we will use, composer allows us to visualize all of their BNCs, and we can use them to connect to the network and interact with it on the behalf of the respective participant, while also being able to visualize the access control restrictions each identity is subject to.

With this, we can use these cards to impersonate each of these users to continue this demo.

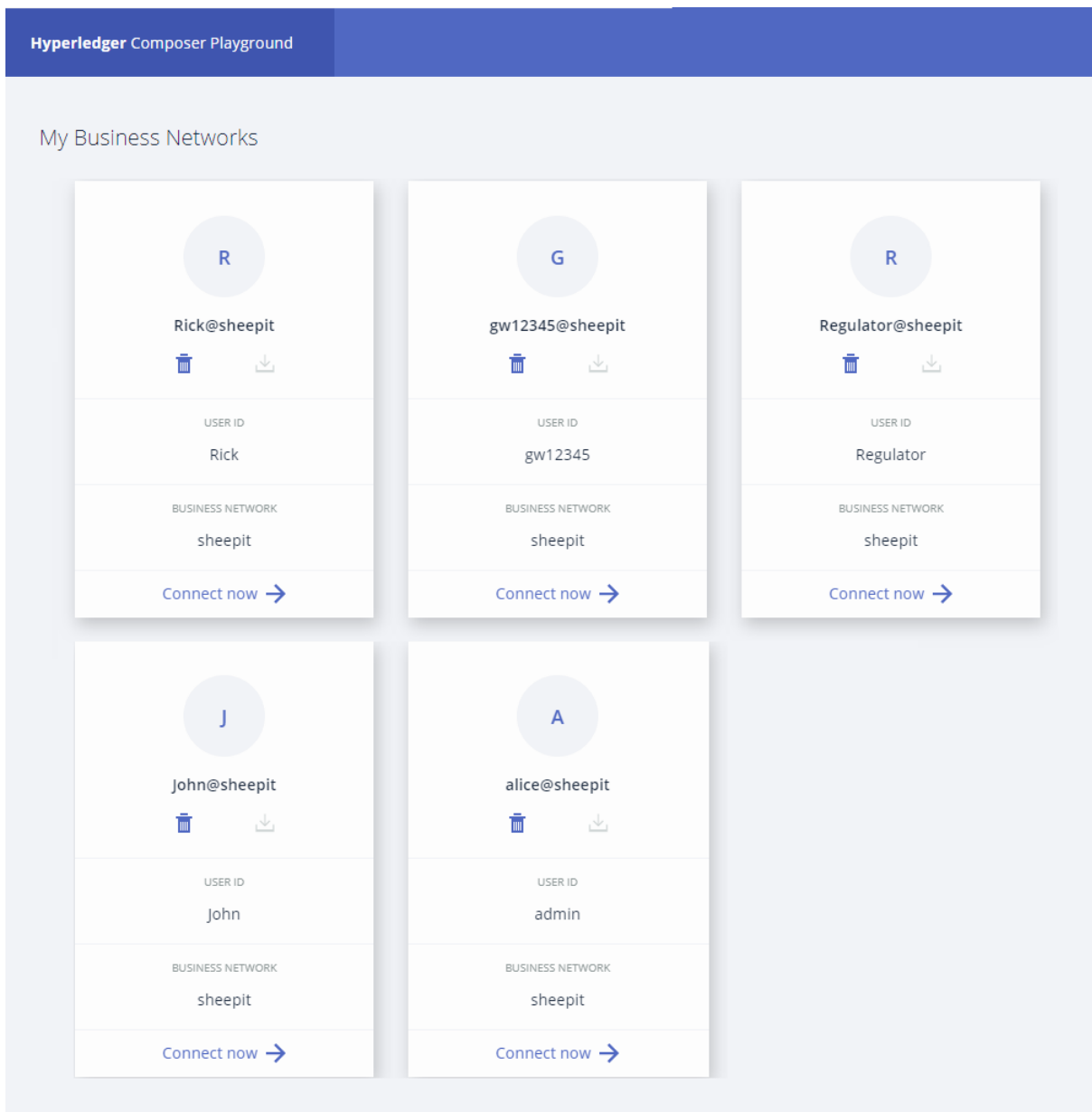


Figure 5.11: Available BNCs

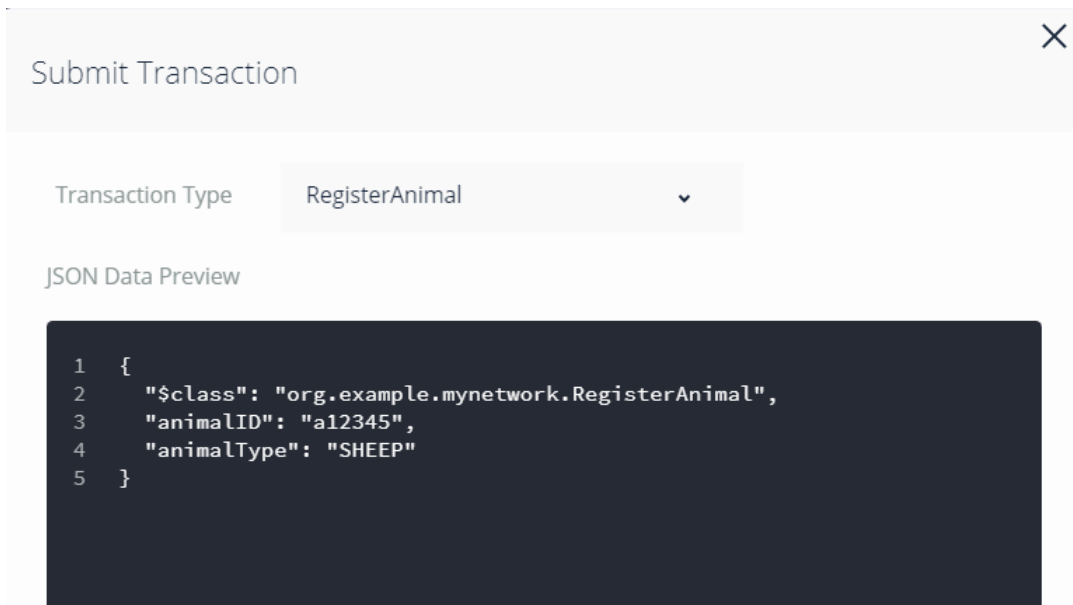


Figure 5.12: John registering an animal

Here, John invokes the RegisterAnimal transaction, where he specifies its ID and type, this creates an entry in the animal registry with the specified parameters, sets the owner field to the transaction sender's ID, and initializes the status and healthStatus to RESTING and UNVERIFIED respectively. The former is used to manage the status of the animal regarding its working status. Regarding the latter, it was thought to manage the health status which would be verified by the veterinarian, but no further work was developed regarding that aspect during this dissertation, as we chose to focus on other functionalities.



Figure 5.13: Animal instance created

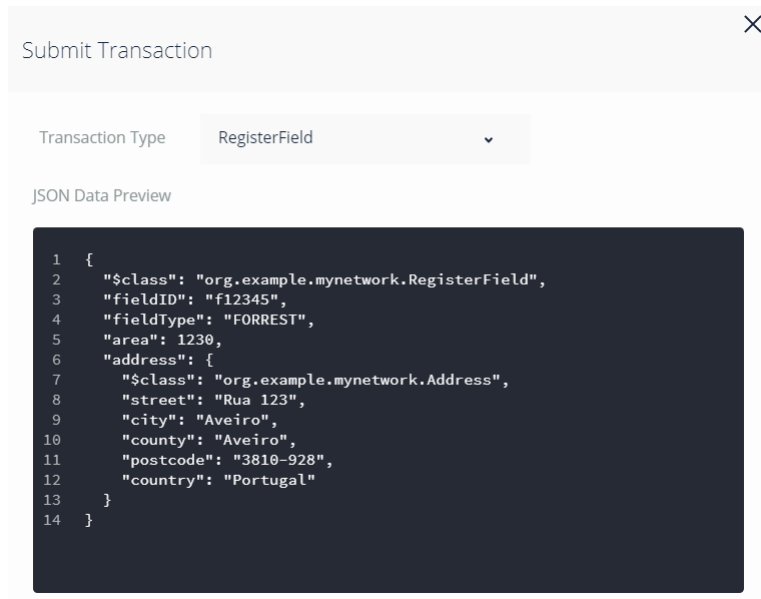


Figure 5.14: Rick registering a field

Rick invoking the transaction RegisterField, in which he provides all the details about it, most importantly the ID of the terrain. This creates an entry in the field registry with the specified parameters, sets the owner property to the transaction sender's Id, and initializes the animalsInside field to an empty array.

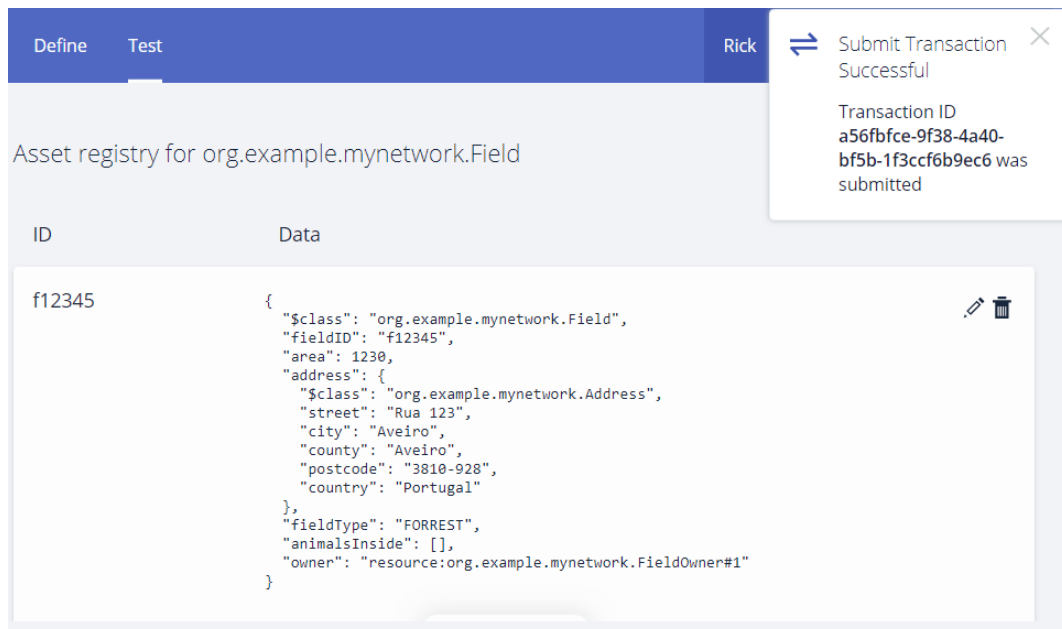


Figure 5.15: Field instance created

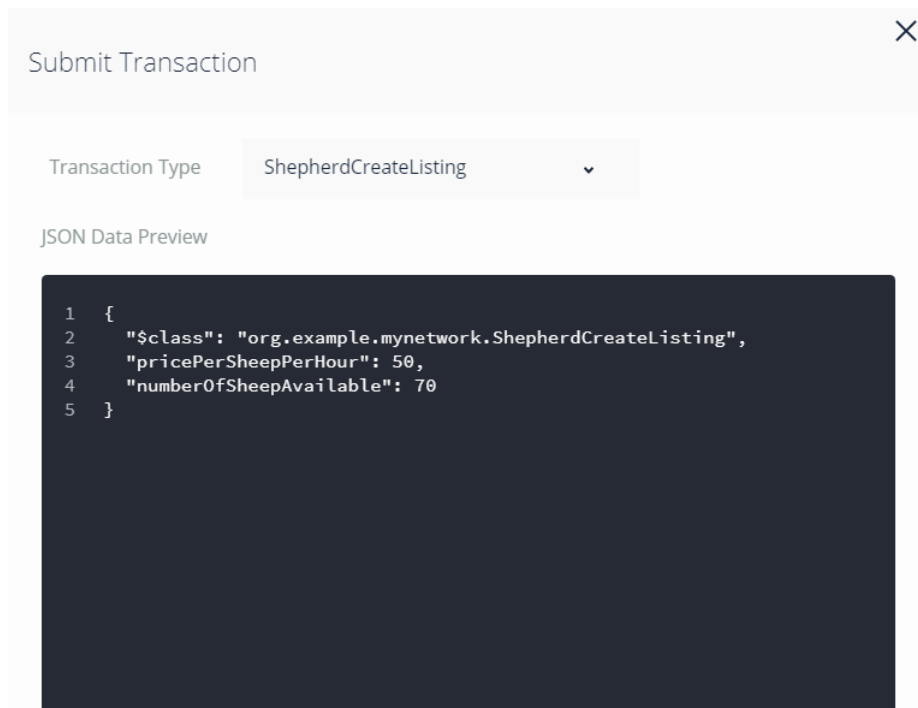


Figure 5.16: John submitting a CreateListing transaction

John invokes the ShepherdCreateListing, where he specified the pricePerSheepPerHour and the numberOfSheepAvailable. This creates an entry in the Listing registry with the specified parameters, it also generates a unique ID for the entry, sets the listingCreator field to the transaction sender's ID, and sets the listingStatus, solicitations, bookedDates to the initial values shown below.

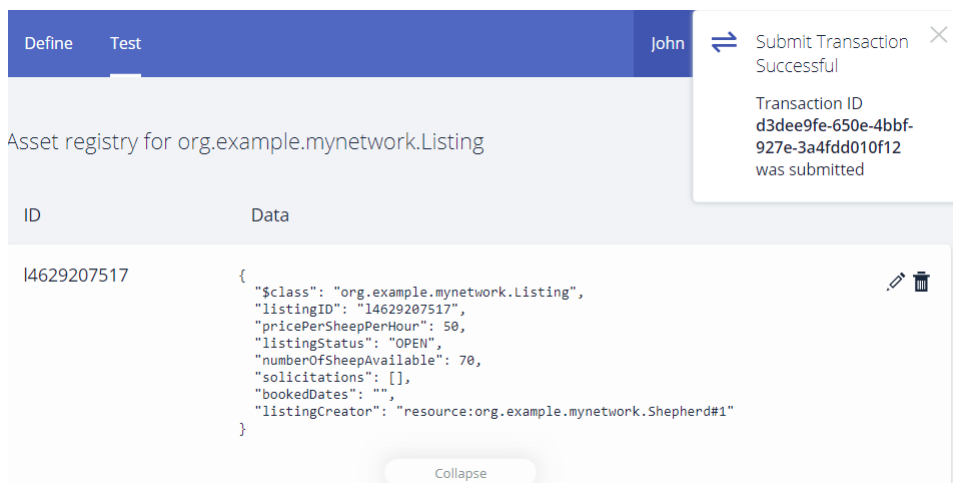


Figure 5.17: John's listing available in the marketplace





Figure 5.18: Rick sending a solicitation to John's Listing

Rick sends a FieldOwnerSendsSolicitation transaction in which he specifies the ID of the listing he is making a solicitation for along with the starting date, end date, and field to which he is requesting the land clearing service (he must be the owner of this field), This creates an entry in the Solicitation registry with the specified parameters, a txSender field set to the transaction sender, and generates a unique ID for this solicitation.

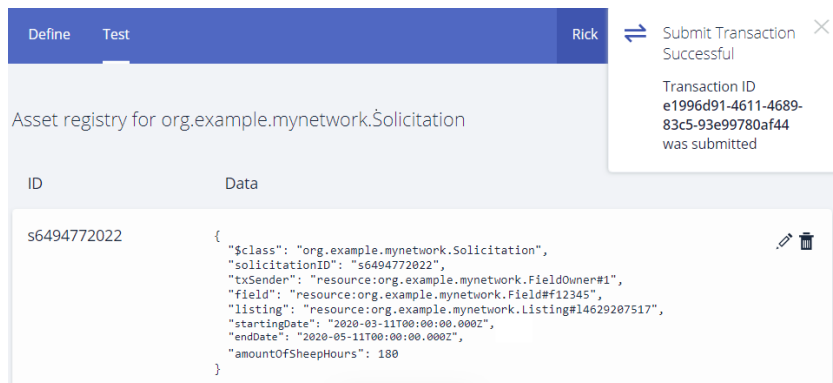


Figure 5.19: Solicitation available in the solicitation registry, only visible to the involved parties

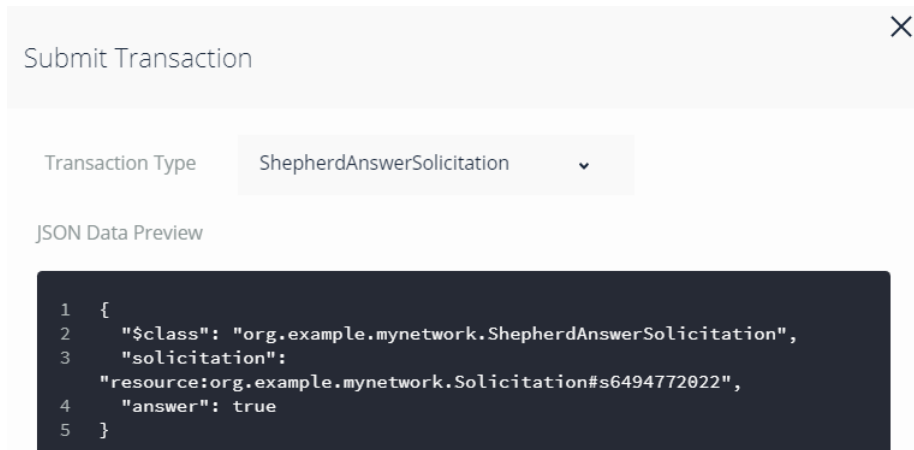


Figure 5.20: John accepting Rick's solicitation

John accepts the solicitation by sending a `ShepherdAnswerSolicitation` transaction, indicating the ID of the solicitation he's answering, and setting the boolean field `answer` to `true`. By doing this, an entry in the `LandClearingContract` is created with all the details from the listing and solicitation which generated it. A unique ID for the contract is generated, and the fields `status` and `amountOfSheepHoursWorked` are set to the initial values shown below. Even though the starting and end dates are specified (which were brought from the solicitation), these have no influence in the contract, and are just guidelines. The 4 flags (`SOInitiated`, `FOInitiated`, `SOFinished` and `FOFinished`) which were initialized to `false` are the decisive fields which change the status of the contract, as we will see in the following images.

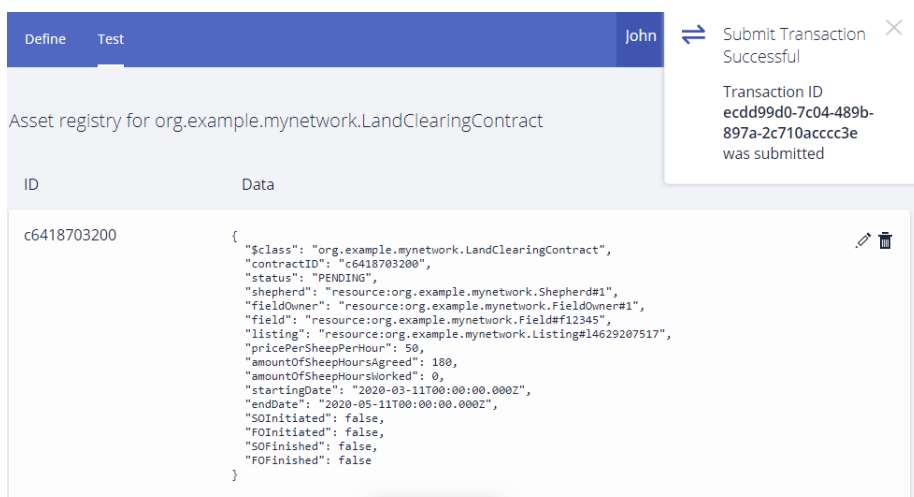


Figure 5.21: Land clearing contract created, visible only to the involved parties

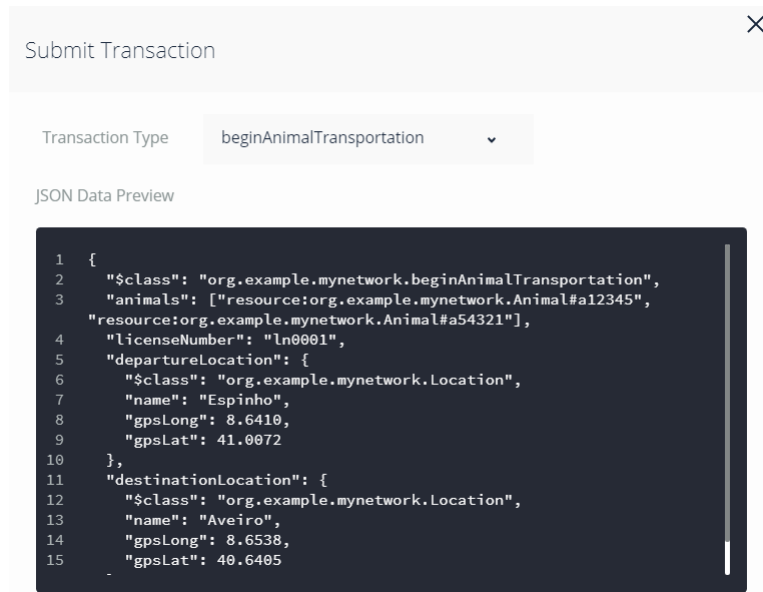


Figure 5.22: Transaction that signals the beginning of animal movement

In this transaction, John signals the beginning of the animal movement. he must specify the license number of the transporter, the animals he is transporting, and the information about the locations of departure and destination. This creates an entry in the animalTransportation registry with these specified details, generating a unique ID for it, setting the animalOwner field to the transaction sender's ID, and the startingTime to the timeStamp of the invocation of this transaction.

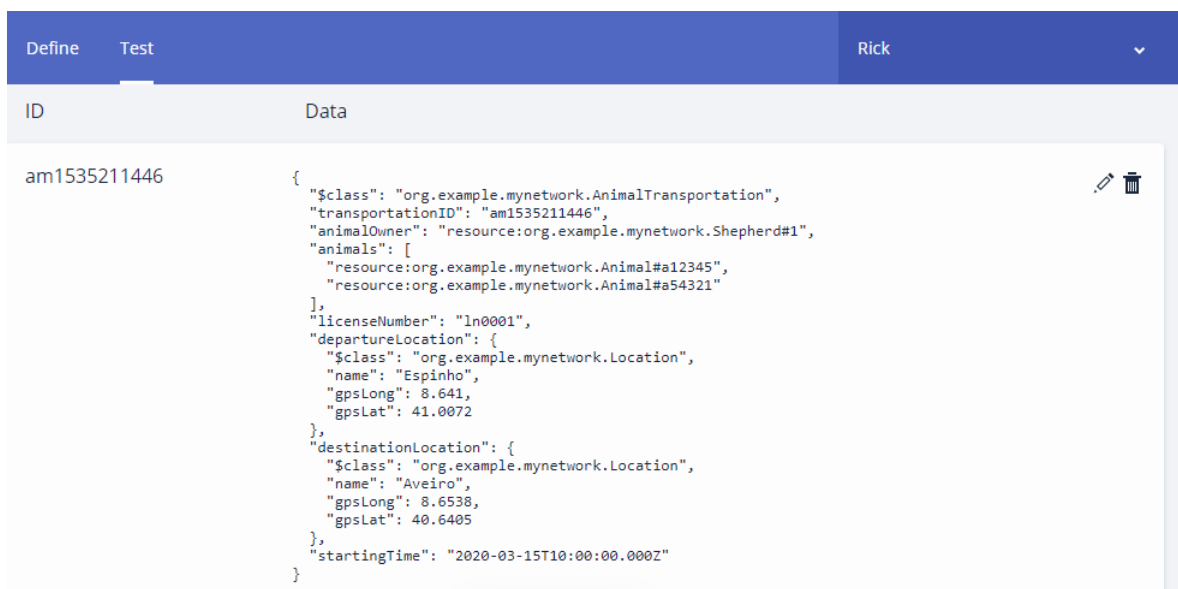


Figure 5.23: Transaction that signals the beginning of animal transportation

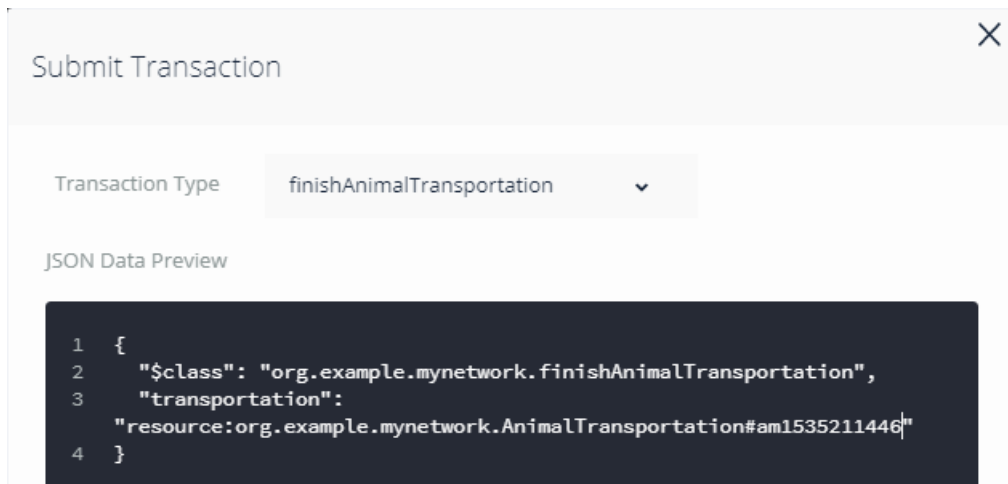


Figure 5.24: Animal transportation instance created in the animalTransportation registry

Once the transportation has completed, John must invoke the `finishAnimalTransportation` transaction, specifying the ID of the `animalTransportation` entry that was created before. This will finish the `animalTransportation` entry by setting the `finishingTime` field to the timestamp of the transaction call.

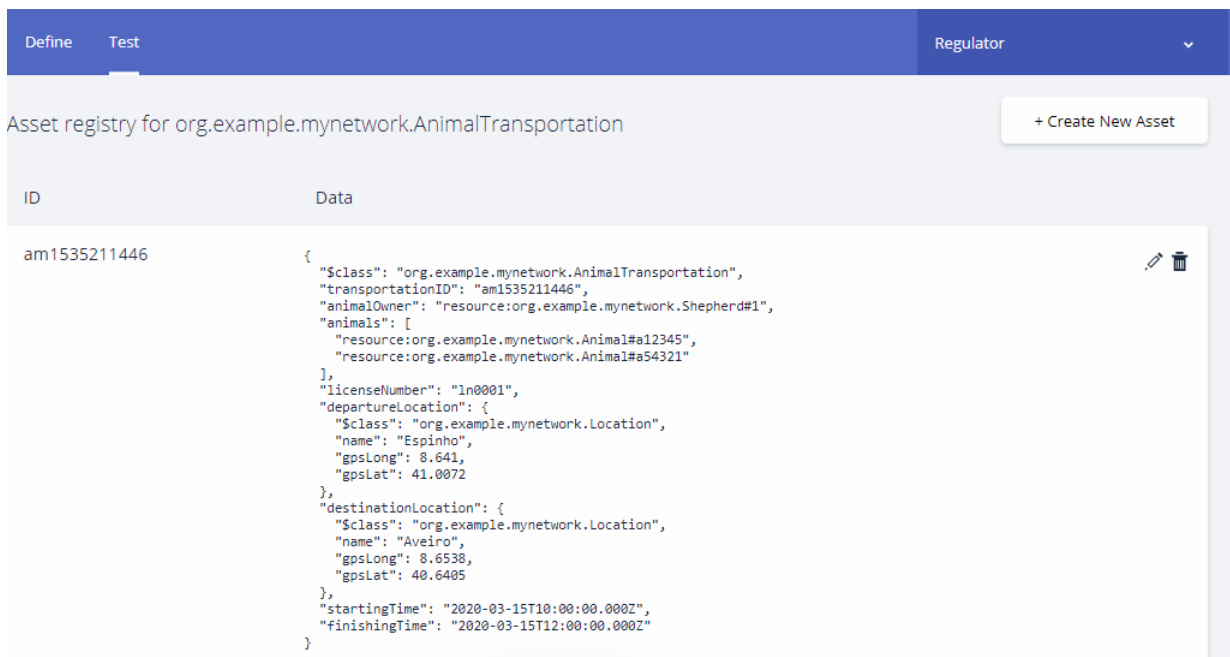


Figure 5.25: animal movement entry added to the registry, in this case being viewed already by Regulator.

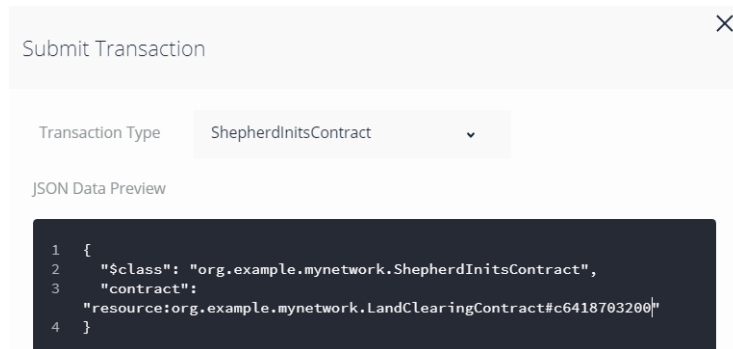


Figure 5.26: John invokes the initiate contract transaction

These 2 transactions are called when the participants wish to effectively start the contract. The only input is the ID of the contract. When both parties have called this transaction, the status field of the respective contract will be set to **ONGOING**, and the service can finally take place.



Figure 5.27: Rick invokes the initiate contract transaction

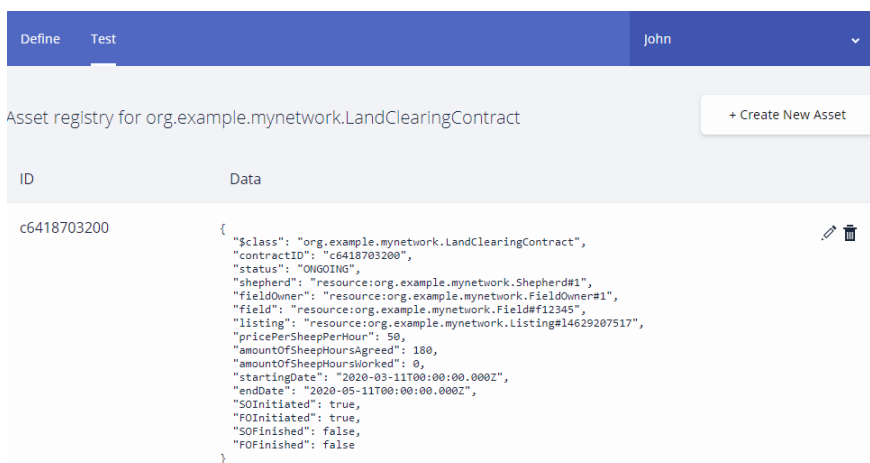


Figure 5.28: Land clearing contract status set to ONGOING

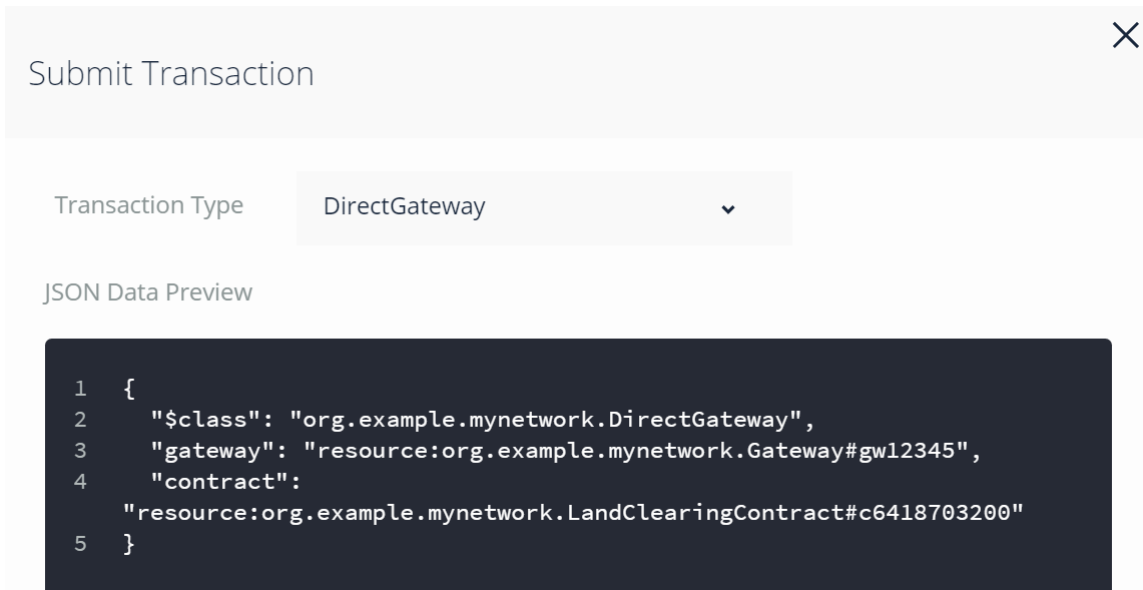


Figure 5.29: Transaction that directs a gateway to a contract

In this transaction, John tells the gateway the ID of the contract it should associate the information to. Only the owner of the gateway is allowed to call this transaction, in this case John.



Figure 5.30: Gateway directed

We can see that the gateway now has the currentContract field set to the land clearing contract that John input in the transaction.

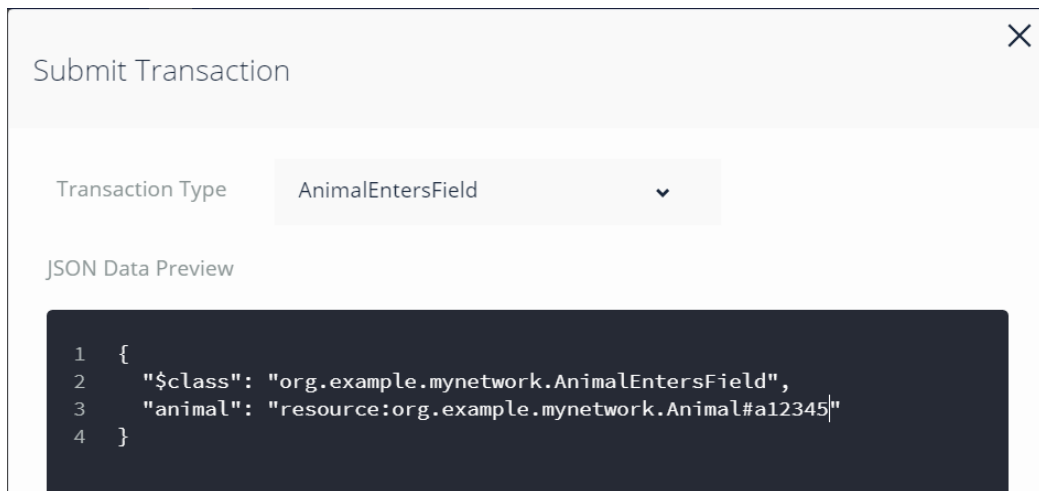


Figure 5.31: Animal enters field transaction, sent by gateway

When the gateway detects that an animal has just entered the field, it will send the above transaction, specifying the ID of the animal it has just detected. This in turn creates an entry in the Pasture registry, with the respective animal ID. The contract field is set to the contractID that is set on the gateway, which was configured in the directGateway transaction. A status field is also set to `ONGOING`, and the startingTime set to the timestamp at which the transaction was invoked.



Figure 5.32: Pasture instance created in the pasture registry

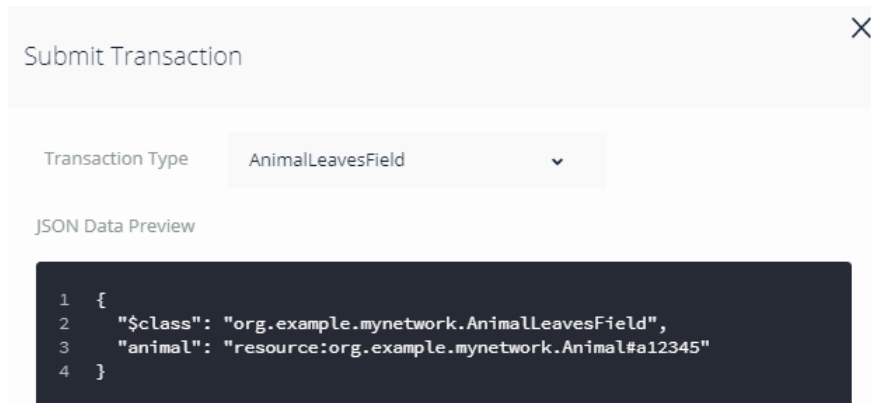


Figure 5.33: Animal leaves field transaction, sent by gateway

When the gateway detects that an animal has just left the field, it will send the above transaction, specifying the ID of the animal it has just detected. This in turn changes the status of the Pasture instance associated with the animal to **FINISHED**, and sets the finishingTime field to the timestamp at which the transaction was invoked.

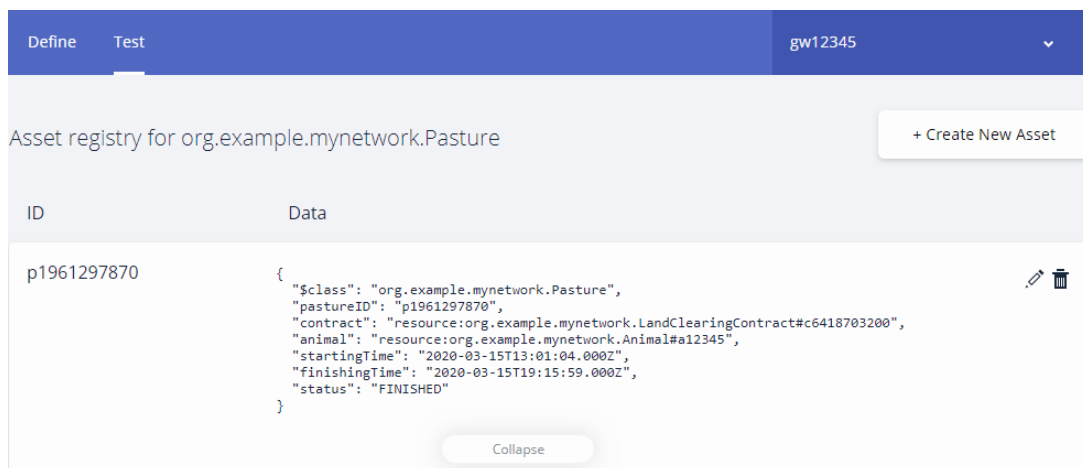


Figure 5.34: Pasture instance finished



The AnimalLeavesField transaction also updates the amountOfSheepHoursWorked field in the contract with which the pasture is associated, adding to it the difference between the startingTime and finishingTime fields of the pasture entry, rounded to 2 decimal places. In this case, we can see that the contract has now 6.25 hours worked in it, calculated from the entry and exit timestamps of 13:01:04 and 19:15:59 respectively. Only the gateway is allowed to create this asset, and if one of the participants tries to tamper with the data, they will be met with the error shown in figure 5.36.



Figure 5.35: Contract entry updated with the total time spent by the animal during the pasture

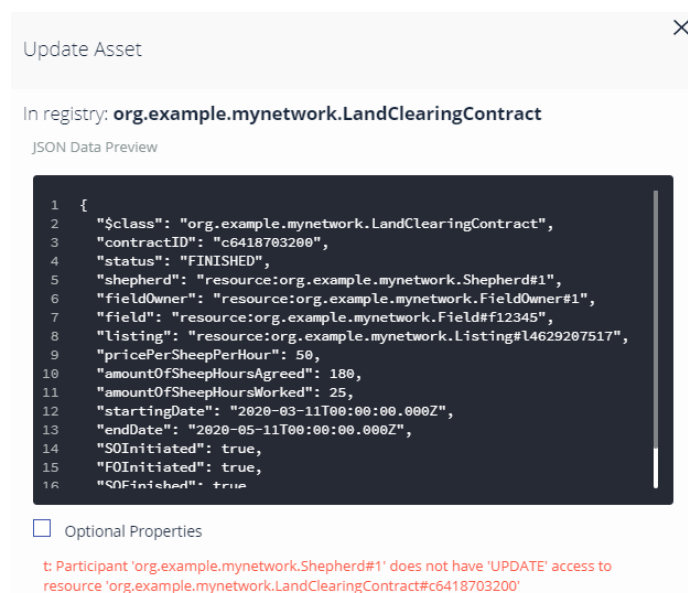


Figure 5.36: John trying to tamper with the amountOfHoursWorked field



Figure 5.37: Rick sends a finishContract transaction

These 2 transactions are called when the participants wish to effectively finish the contract. The only input is the ID of the contract. When both parties have called this transaction, the status field of the respective contract will be set to **FINISHED**.

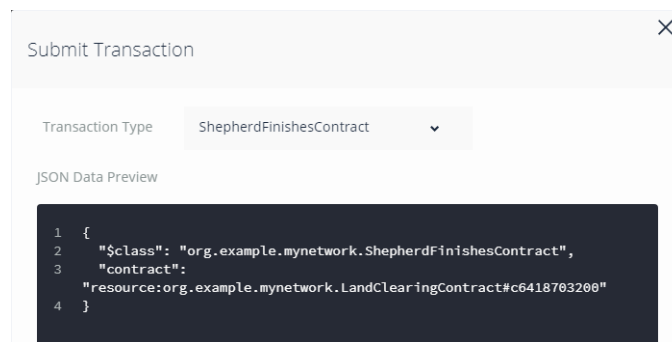


Figure 5.38: John sends a finishContract transaction

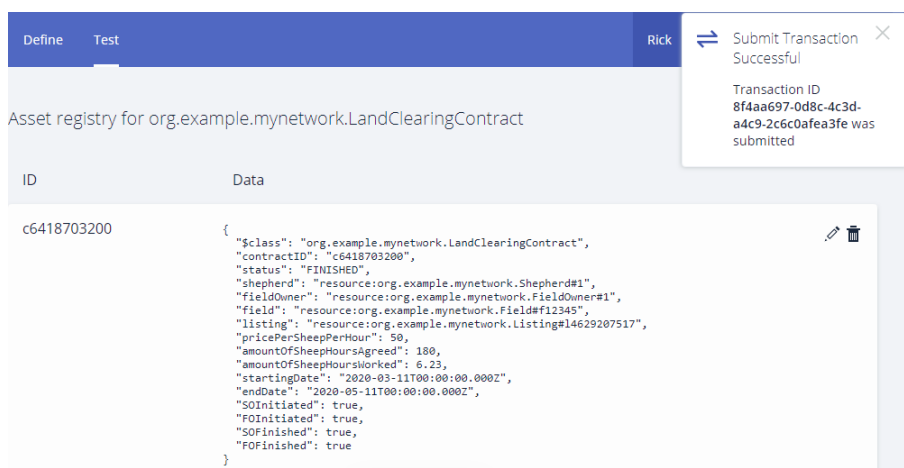


Figure 5.39: Contract Finished

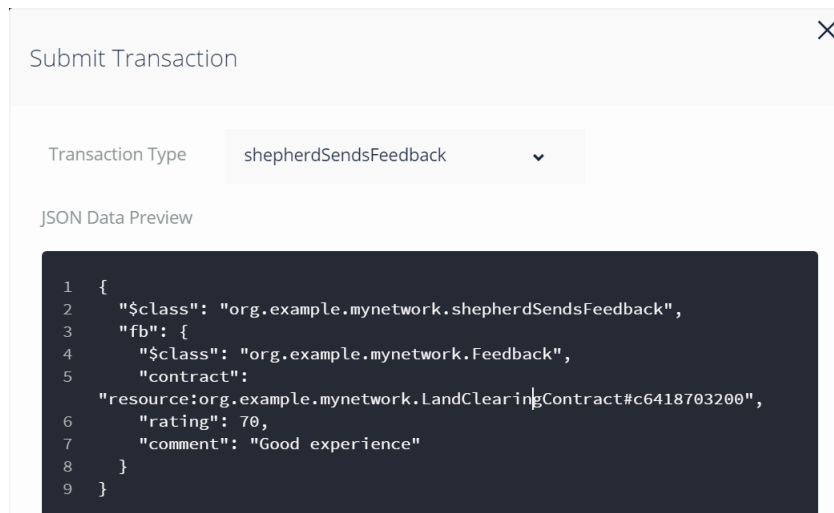


Figure 5.40: John sends Rick his feedback about the finished contract

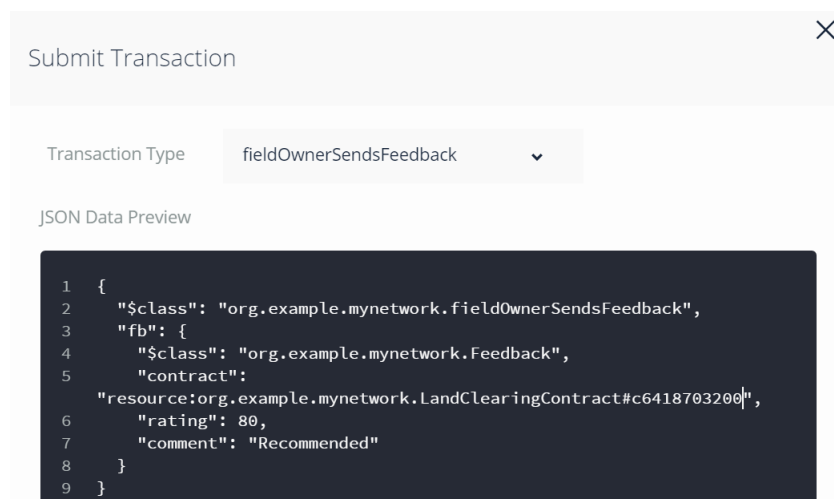
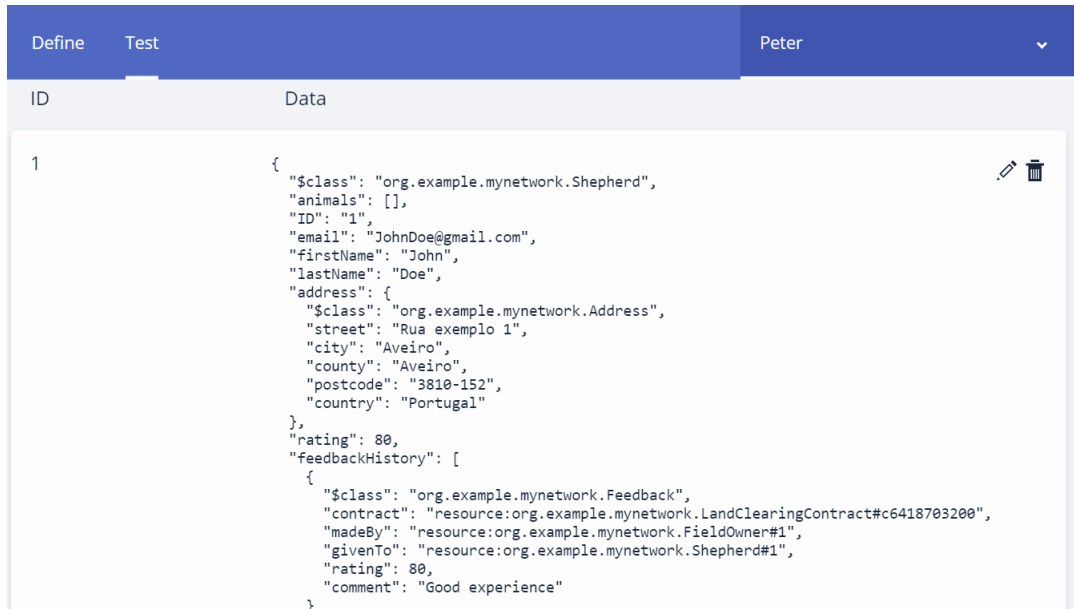


Figure 5.41: Rick sends John his feedback about the finished contract

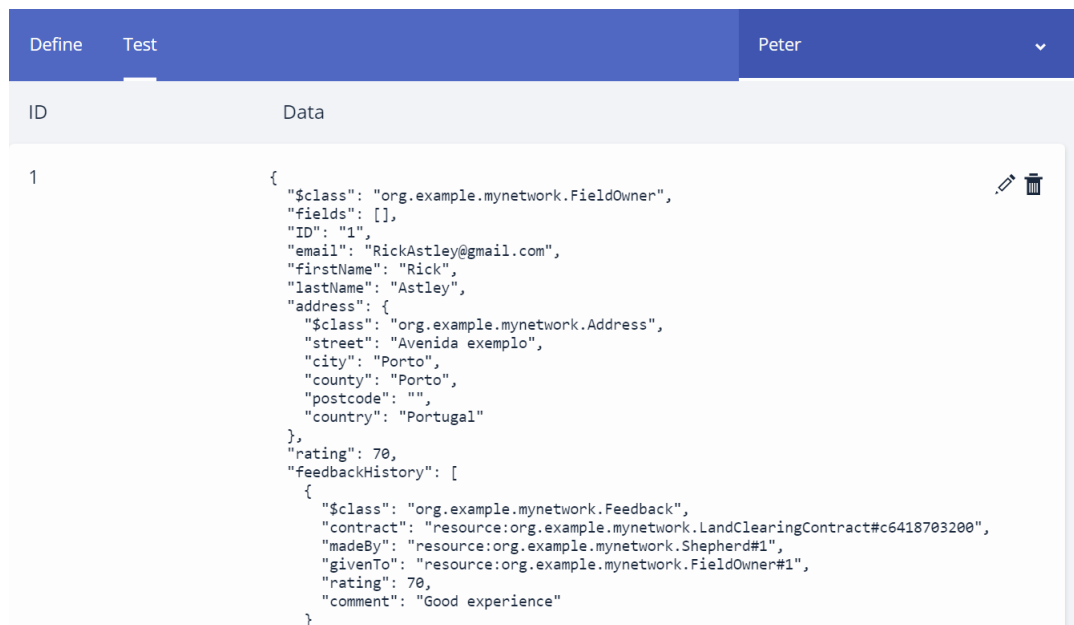
In these transactions, John and Rick send each other their comments and rating relative to their satisfaction regarding the service. In the transaction they must input the ID of the contract, the rating score from 0 to 100, and an optional comment. This will in turn add an entry in the feedbackHistory field of each user with the specified parameters, sets the madeBy field to the transaction sender's ID, and the givenTo field to the other participant in the contract. This is visible to any user in the platform, as can be seen in the following images, where we created another user - Peter, who is able to visualize the feedback history of both users.

Here we can see that Peter can visualize the other participant's profiles, where he can find the feedback history, and the rating field, which is an average of all of the ratings received.



```
Define Test Peter
ID Data
1 {
  "$class": "org.example.mynetwork.Shepherd",
  "animals": [],
  "ID": "1",
  "email": "JohnDoe@gmail.com",
  "firstName": "John",
  "lastName": "Doe",
  "address": {
    "$class": "org.example.mynetwork.Address",
    "street": "Rua exemplo 1",
    "city": "Aveiro",
    "county": "Aveiro",
    "postcode": "3810-152",
    "country": "Portugal"
  },
  "rating": 80,
  "feedbackHistory": [
    {
      "$class": "org.example.mynetwork.Feedback",
      "contract": "resource:org.example.mynetwork.LandClearingContract#c6418703200",
      "madeBy": "resource:org.example.mynetwork.FieldOwner#1",
      "givenTo": "resource:org.example.mynetwork.Shepherd#1",
      "rating": 80,
      "comment": "Good experience"
    }
  ]
}
```

Figure 5.42: John's profile updated with the feedback entry given by Rick



```
Define Test Peter
ID Data
1 {
  "$class": "org.example.mynetwork.FieldOwner",
  "fields": [],
  "ID": "1",
  "email": "RickAstley@gmail.com",
  "firstName": "Rick",
  "lastName": "Astley",
  "address": {
    "$class": "org.example.mynetwork.Address",
    "street": "Avenida exemplo",
    "city": "Porto",
    "county": "Porto",
    "postcode": "",
    "country": "Portugal"
  },
  "rating": 70,
  "feedbackHistory": [
    {
      "$class": "org.example.mynetwork.Feedback",
      "contract": "resource:org.example.mynetwork.LandClearingContract#c6418703200",
      "madeBy": "resource:org.example.mynetwork.Shepherd#1",
      "givenTo": "resource:org.example.mynetwork.FieldOwner#1",
      "rating": 70,
      "comment": "Good experience"
    }
  ]
}
```

Figure 5.43: Rick's profile updated with the feedback entry given by Rick

To finalize, here we can see some more examples of the access control restrictions we created being applied to the participants. These restrictions are in accordance with the ones we defined in table 3.1. In this case John, when trying to act maliciously for his own benefit, first by sending an `AnimalEntersField` transaction, and then by trying to alter the feedback he received from Rick.

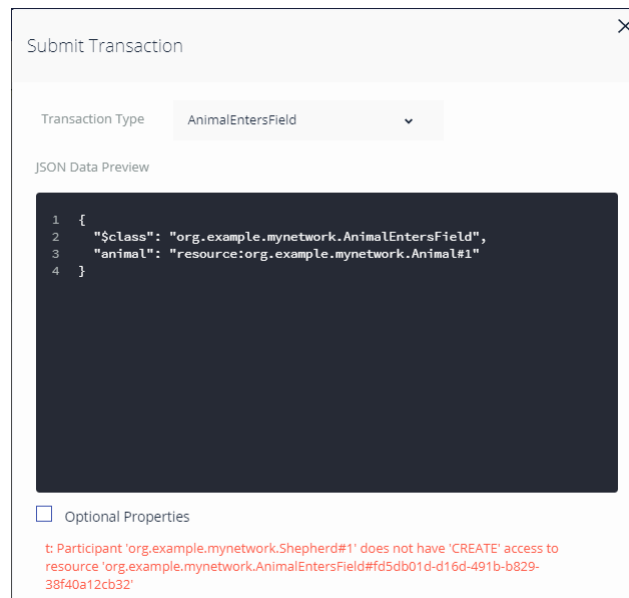


Figure 5.44: John trying to send an `AnimalEnterField` transaction

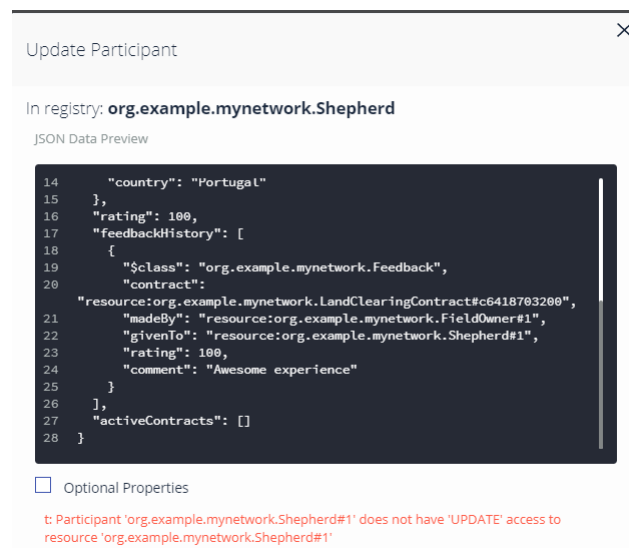


Figure 5.45: John trying to send an `UpdateParticipant` transaction



## CHAPTER 6

---

### Conclusions

---

The system created allows participants to easily register in the platform along with their assets. Users can then engage in contracts for land clearing services through a process where sheep owners publicize their listings on our marketplace and then accept or refuse solicitations made by field owners on the respective offer.

The system is prepared to receive the input from the SheepIT hardware, and feed it onto our blockchain network, where it is visible for both parties to verify and follow the progress of each pasture.

A rating system was built, where both parties can rate each other when the contract reaches its finishing condition, accordingly to how satisfied with the overall process, therefore shaping the rating associated with the other user, which is visible to users when looking for land clearing service listings.

The management of the health of the animals through the platform was not implemented, as we had some difficulties conceptualizing how we would manage appointments, reports, and confidentiality issues, but also how to find an efficient way to guarantee to a field owner that all of the animals clearing his terrain are in fact problem-free.

Access control logic was implemented successfully to ensure that data confidentiality requirements were taken into consideration, presenting each user with only the data and functionalities that concern

them.

A REST API was created, which makes all of our functions available through easy-to-use endpoints, ready to be consumed by a proper user-friendly web interface. We built a web interface with the use of Yeoman, but we feel that it still needed a lot of work to become close to what an end-user would like to use to interact with the network.

Data integrity is taken care of by the underlying consensus mechanism of HLF, preventing a single entity from adding invalid data to the ledger, instead, transactions need to be validated by one peer from each organization before being endorsed, and therefore added to the chain.

When it comes to the configuration and architecture of the network, we did not achieve the ideal solution that we devised, as we had trouble setting up multiple orderer nodes in kafka mode, and instead are running a single one, failing to guarantee crash-tolerance. The network was also configured using docker containers, with both organizations' fabric networks running on the same machine. The ideal setup would have been to set these up across multiple machines hosted in separate IP networks, domains, or secure cloud environments, which would allow us to test our solution in a state which would be more similar to a production environment.

In summary, we believe that the results of the work are very close to the requirements and that the goal of the dissertation was reached. We think that the system could be used in the real world if all of the functionalities were implemented and the system built on top of a resilient architecture. Having achieved that, this platform could become an easy way for field and sheep owners to engage in business with each other, as it would be the first one of its kind.

However, having reflected on the utility of the blockchain for this problem in particular, we believe that this was not an ideal use case for blockchain technology, as we believe that its use doesn't bring much advantage to the end user when compared to a potential centralized solution. We state this because the population we're targeting with this platform already belongs to a small niche (sheep owners and field owners looking to engage in land clearing services). Out of this population, it is unlikely that a large group would be interested in the advantages our blockchain platform brings (decentralization, data immutability, integrity, provenance). However, we believe that different aspects of this work could bring more value to our end users, had they been developed further. The main example would be the tracking of the health of animals since birth.



## 6.1 Future work

In the duration of this work, the Hyperledger Composer tool has become deprecated, which means that even though it is usable, it is no longer in active development from Hyperledger.

With this in mind, the first step for any future work to be made on our project should be to re-implement the business logic of our system in another Hyperledger framework, or even a different platform. The next step should be to implement the network across multiple machines, in order to test the feasibility of a scaled-up version of this system by monitoring how the network would handle the expected transaction throughput. With a convincing result, a good-looking and well-functioning web interface or app for users to interact with the network should be built, making the whole system ready to go live in order to be used by sheep and field owners in order to engage in land clearing contracts with each other.



---

## Bibliography

---

- [1] ICNF - a limpeza de terrenos junto de habitações é obrigatória? <http://www2.icnf.pt/portal/icnf/faqs/dfci/obrigat-limp-habit>. Accessed: 2018-11-18.
- [2] Salomão Rodrigues. Cabras anãs limpam quatro mil metros de terreno em três meses. <https://www.jn.pt/local/noticias/aveiro/santa-maria-da-feira/cabras-anas-limpam-quatro-mil-metros-de-terreno-em-tres-meses-9519159.html>. Accessed: 2018-11-18.
- [3] SmartAKIS - what is smart farming? <https://www.smart-akis.com/index.php/network/what-is-smart-farming/>. Accessed: 2018-09-14.
- [4] SheepIT Project - motivation. <http://www.av.it.pt/sheepit/Motivation.html>. Accessed: 2018-07-22.
- [5] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report.
- [6] CoinMarketCap - total market capitalization. <https://coinmarketcap.com/charts/>. Accessed: 2018-09-03.
- [7] Preethi Kasireddy. Medium - what do we mean by “blockchains are trustless”? <https://medium.com/@preethikasireddy/eli5-what-do-we-mean-by-blockchains-are-trustless-aa420635d5f6>. Accessed: 2019-09-29.

- [8] A. M. Magadlela et. al. Brush clearing on hill land pasture with sheep and goats. <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1439-037X.1995.tb00188.x>. Accessed: 2019-10-11.
- [9] Elm goats - environmentally conscious brush abatement providing fire breaks and defensible space. <http://www.elmgoats.com/why.html>. Accessed: 2018-11-18.
- [10] Luís Nóbrega, Paulo Pedreiras, and P. Gonçalves. *SheepIT, an IoT-Based Weed Control System: 8th International Conference, HAICTA 2017, Chania, Crete, Greece, September 21–24, 2017, Revised Selected Papers*, pages 131–147. 01 2019.
- [11] Antonio Ferreira Cardoso. *Advanced Sheep Posture Controller Controlador Avancado de Postura de Gado Ovino*. PhD thesis.
- [12] Luís Nóbrega, Paulo Pedreiras, and Pedro Gonçalves. SheepIT-An Electronic Shepherd for the Vineyards. Technical report.
- [13] José Paulo and Gomes Pereira. *Sistema de localização de baixo consumo para ovelhas*. PhD thesis, 2018.
- [14] G J Bishop-Hurley, D L Swain, D M Anderson, P Sikka, C Crossman, and P Corke. Virtual fencing applications: Implementing and testing an automated cattle control system. *Computers and Electronics in Agriculture*, 56:14–22, 2007.
- [15] AngiesList - rent a ruminant llc. <https://www.angieslist.com/companylist/us/wa/vashon/rent-a-ruminant-llc-reviews-2329648.htm>. Accessed: 2018-11-18.
- [16] The goat lady. <http://www.thegoatlady.com/index.html>. Accessed: 2018-11-18.
- [17] Joshua Palmer. AngiesList - rent a goat to clear brush and maintain your lawn. <https://www.angieslist.com/articles/rent-goat-clear-brush-and-maintain-your-lawn.htm>. Accessed: 2018-11-18.
- [18] Airbnb - instant book. <https://www.airbnb.com/host/instant-book>. Accessed: 2019-10-11.
- [19] Airbnb - when wil i get my payout? <https://www.airbnb.com/help/article/425/when-will-i-get-my-payout>. Accessed: 2019-10-12.
- [20] Airbnb - what is the airbnb service fee? <https://www.airbnb.com/help/article/1857/what-is-the-airbnb-service-fee>. Accessed: 2019-10-12.
- [21] The Economist - the great chain of being sure about things. <https://www.economist.com/briefing/2015/10/31/the-great-chain-of-being-sure-about-things>. Accessed: 2018-07-26.

- [22] Demiro Massessi. Medium - public vs private blockchain in a nutshell. <https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f>. Accessed: 2018-07-26.
- [23] Binance - what is a blockchain consensus algorithm? <https://www.binance.vision/blockchain/what-is-a-blockchain-consensus-algorithm>. Accessed: 2019-10-02.
- [24] Binance - byzantine fault tolerance explained. <https://www.binance.vision/blockchain/byzantine-fault-tolerance-explained>. Accessed: 2019-10-02.
- [25] Luke Fortney. Investopedia - bitcoin mining, explained. <https://www.investopedia.com/terms/b/bitcoin-mining.asp>.
- [26] Frank Jacobs. BigThink - how bitcoin consumes more energy than 159 individual countries. <https://bigthink.com/strange-maps/bitcoin-consumes-more-energy-than-159-individual-countries>. Accessed: 2018-08-27.
- [27] Julia Magas. Cointelegraph - casper: What will the upgrade bring to the ethereum's network? <https://cointelegraph.com/news/casper-what-is-known-about-the-new-ethereums-network-upgrade>. Published: May 16, 2018.
- [28] Janno Siim. Proof-of-Stake Research Seminar in Cryptography. Technical report.
- [29] Nick Szabo. Formalizing and securing relationships on public networks by nick szabo. <https://ojphi.org/ojs/index.php/fm/article/view/548/469>. Accessed: 2018-12-19.
- [30] Alyssa Hertig. Coindesk - what is a dao? <https://www.coindesk.com/information/what-is-a-dao-ethereum>. Accessed: 2018-12-20.
- [31] Cryptocompare - the dao, the hack, the soft fork and the hard fork. <https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/>. Accessed: 2018-12-20.
- [32] Niklas Luhmann. The Future of Democracy. Technical Report 1, 2016.
- [33] Ved Raj. readwrite - a comprehensive guide to top blockchain platforms. <https://readwrite.com/2019/11/01/a-comprehensive-guide-to-top-blockchain-platforms/>. Accessed: 2018-11-15.
- [34] Tamas Blummer et. al. An Introduction to Hyperledger. Technical report. Published: July 2018.
- [35] Christian Cachin. Architecture of the Hyperledger Blockchain Fabric \*. Technical report, 2016.

- [36] Muntasir Mamun. Medium - how does hyperledger fabric work? <https://medium.com/coinmonks/how-does-hyperledger-fabric-works-cdb68e6066f5>. Accessed: 2019-03-11.
- [37] Hyperledger - introduction to hyperledger composer. <https://hyperledger.github.io/composer/latest/introduction/introduction.html>. Accessed: 2018-09-02.
- [38] Hyperledger - connection profiles. <https://hyperledger.github.io/composer/latest/reference/connectionprofile>. Accessed: 2019-03-12.
- [39] Hyperledger - private data. <https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html>. Accessed: 2019-03-12.
- [40] Vitalik Buterin. A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM. Technical report, 2014.
- [41] Ropsten ethereum faucet. <https://faucet.ropsten.be/>. Accessed: 2018-11-15.
- [42] Craig Williams. Kauri - reducing mainnet transactions with a sidechain. <https://kauri.io/article/a66d50655f8746edb7df90de4b8eb416/reducing-mainnet-transactions-with-a-sidechain>. Accessed: 2019-09-13.
- [43] Kieran Smith. Bravenewcoin - ethereum en route to a million transactions per second. <https://bravenewcoin.com/insights/vitalik-ethereum-en-route-to-a-million-transactions-per-second>. Accessed: 2019-11-20.
- [44] Ethgasstation - how long does an ethereum transaction really take? <https://ethgasstation.info/blog/ethereum-transaction-how-long/>. Accessed: 2019-11-20.
- [45] Philipp Sandner Martin Valenta. Comparison of Ethereum, Hyperledger Fabric and Corda. Technical report. Published: June 2017.
- [46] Thomas Simms. Cointelegraph - upgraded hyperledger fabric sees 7-fold increase in transaction speed. <https://cointelegraph.com/news/upgraded-hyperledger-fabric-sees-7-fold-increase-in-transaction-speed>. Accessed: 2019-11-20.
- [47] Elli Androulaki et. al. Hyperledger fabric: A distributed operating system for permissioned blockchains. <https://arxiv.org/abs/1801.10228v1/>. Accessed: 2019-11-20.
- [48] MINISTERIO DA AGRICULTURA DO DESENVOLVIMENTO RURAL E DAS PESCAS. Decreto-Lei n° 276/2001 de 17 de Outubro. Technical report. Published: 2001-10-17.

- [49] Severiano Rocha e Silva et. al. Manual boas práticas ovinos. <https://www.confagri.pt/content/uploads/2018/11/Manual-Boas-Pr%C3%A1ticas-UCADESA-OVINOS.pdf>. Accessed: 2019-10-10.
- [50] MIT - i. basic principles of information protection. <http://web.mit.edu/Saltzer/www/publications/protection/Basic.html>. Accessed: 2018-12-29.
- [51] Bitinfocharts - ethereum transaction fee. <https://bitinfocharts.com/comparison/ethereum-transactionfees.html>. Accessed: 2019-02-26.
- [52] Hyperledger - membership. <https://hyperledger-fabric.readthedocs.io/en/latest/membership/membership.html#msp-structure>. Accessed: 2019-03-13.
- [53] About enterprise plan. [https://console.bluemix.net/docs/services/blockchain/enterprise\\_plan.html#enterprise-plan-about](https://console.bluemix.net/docs/services/blockchain/enterprise_plan.html#enterprise-plan-about). Accessed: 2019-03-14.
- [54] IBM Cloud - about starter plan. [https://console.bluemix.net/docs/services/blockchain/starter\\_plan.html#starter-plan-about](https://console.bluemix.net/docs/services/blockchain/starter_plan.html#starter-plan-about). Accessed: 2019-03-14.
- [55] IBM Cloud - about enterprise plan. <https://console.bluemix.net/docs/services/blockchain/howto/pricing.html#ibp-pricing-starter-pricing>. Accessed: 2019-03-14.
- [56] Hyperledger - build your first network. [https://hyperledger-fabric.readthedocs.io/en/latest/build\\_network.html](https://hyperledger-fabric.readthedocs.io/en/latest/build_network.html). Accessed: 2019-11-22.
- [57] Hyperledger - deploying a hyperledger composer blockchain business network to hyperledger fabric (multiple organizations). <https://hyperledger.github.io/composer/v0.19/tutorials/deploy-to-fabric-multi-org>. Accessed: 2019-11-22.
- [58] Hyperledger - connection profiles. <https://hyperledger.github.io/composer/v0.19/reference/connectionprofile>. Accessed: 2018-11-29.
- [59] Akarsh Agarwal. Medium - understanding the node-sdk of hyperledger and running the first example. <https://medium.com/mlg-blockchain-consulting/understanding-the-node-sdk-of-hyperledger-and-running-the-first-example-5f6753393ec8>. Accessed: 2019-11-27.

