Diogo Duarte de Abreu Farinha

Bachelor in Electrical and Computer Engineering Sciences

Blockchain based architecture to increase trustability and transparency in manufacturing

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in
Electrical and Computer Engineering

Adviser:	José Barata de Oliveira, Professor, Universidade Nova de Lisboa
Co-adviser:	André Dionísio Rocha, Professor, Universidade Nova de Lisboa

Examination Committee

Chairperson:	Doctor André Damas Mora
Raporteur:	Doctor João Paulo Pimentão
Member:	Doctor André Dionísio Rocha

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2020

Blockchain based architecture to increase trustability and transparency in manufacturing

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Dedico este trabalho à minha mãe Luísa, ao meu pai Acácio e ao meu irmão Guilherme.

Acknowledgements

Neste espaço, gostaria de prestar os meus mais sinceros agradecimentos a todos os que contribuiram e continuam a contribuir para este meu percurso.

Gostaria de agradecer ao Professor José Barata e ao Professor André Rocha por me terem acolhido e guiado nesta árdua tarefa que é realizar uma tese.

Aos meus amigos; nomeadamente Pepi, Gonçalos, Bernardo e malta do MP que sempre estiveram presentes e genuinamente me tornam numa melhor pessoa. Um grande obrigado.

Ao Tó, toda a disponibilidade e orientação ao longo do curso não tem preço e estarei eternamente agradecido.

Aos meus pais, ao meu irmão e aos meus avós, a quem devo tudo.

Quando olho para o lado e vejo as pessoas que tenho a meu lado, só posso estar grato.

Por fim, gostaria de deixar uma palavra para o Professor Tiago Cardoso. Foi um privilégio ter sido seu aluno. Descanse em paz.

Abstract

As the fourth industrial revolution is now more of a reality than a mere futuristic vision, necessary changes need to be addressed in the manufacturing environment. The general inability of companies and customers to distinguish between truthful and untruthful information has opened a dangerous void; one that is often filled with fraud and greed. This, allied with the general population's concerns over privacy and data mismanagement, has led to the birth of a new technology: blockchain.

A universally accessible ledger that generates immutable data allows systems to be based on cryptography. Decentralized by nature, users no longer need to trust a centralized node with their data. The user becomes empowered as they are rewarded for participating, as opposed to a central entity reaping all of the wealth generated by the network.

For companies in the manufacturing business, blockchain technology is also specially beneficial. Removing the trust factor among the actors in the horizontal integration of industry 4.0 translates to more transparency in the supply chain, which leads to a better overall vision and understanding of the product.

The aim of this thesis is to implement an architecture connecting suppliers, retail and customers based on blockchain technology. Complete and reliable traceability of the product to the customer is the goal. By providing truthful information to all the actors involved in a product's life cycle, value can be added, thus increasing transparency and trust in the whole network.

Keywords: Blockchain, Industry 4.0, Manufacturing, Transparency, Traceability

Resumo

À medida que a quarta revolução industrial se vai tornando cada vez mais presente, certas questões necessitam de ser repensadas. A incapacidade geral de empresas e do consumidor de distinguir informações verídicas de não verídicas abre um vazio que muitas vezes dá azo a ganância e fraude. A juntar a isto, uma acrescida preocupação da população geral em matérias como privacidade e uso indevido de dados levou a que uma nova tecnologia emergisse: blockchain.

Uma lista universalmente acessível que gera dados imutáveis permite que sistemas tenham como base a criptografia. Descentralizada por natureza, os utilizadores deixam de ser forçados a confiar num nó central a sua informação. O utilizador ganha poder, passa a ser visto como um membro valioso e é recompensado por participar, ao invés de uma entidade central colher toda a riqueza gerada na rede.

Para empresas no mundo da manufatura, a tecnologia de blockchain é especialmente atrativa. Remover a necessidade de confiança entre os atores na integração horizontal na indústria 4.0 traduz-se em maior transparência na supply chain, o que leva a uma melhor visão e compreensão do produto no geral.

A finalidade desta tese é implementar uma arquitetura que conecta fornecedores, retail e consumidores baseada na tecnologia de blockchain. Rastreabilidade completa e fiável do produto é o objetivo. É possível acrescentar valor ao partilhar informação verídica a todos os atores envolvidos no ciclo de vida de um produto, o que leva a um aumento de transparência e confiança em toda a rede.

Palavras-chave: Blockchain, Indústria 4.0, Manufatura, Transparência, Rastreabilidade

Contents

List of Figures

List of Tables

Acronyms

ASIC	Application-specific integrated circuits.
B2B	Business-to-business.
CPS	Cyber-physical Systems.
GUI	Graphical User Interface.
HTTP	Hypertext Transfer Protocol.
IoE	Internet of Everything.
IoT	Internet of Things.
IT	Information Technology.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
P2P	Peer-to-peer.
PBFT	Practical Byzantine Fault Tolerance.
PK	Public Key.
PoS	Proof of Stake.
PoW	Proof of Work.
SHA	Secure Hash Algorithm.
SK	Secret Key.
UI	User Interface.
URL	Uniform Resource Locator.

VE	Virtual Enterprise.
WHO	World Health Organization.

Chapter 1

Introduction

Industry 4.0 has helped the phenomenon of widespread digitalization to take off. While this has been a key contributor to shaping the world as we know it, it isn't so without some serious downsides.

In this chapter, some of these problems and resulting consequences are explored. A contextualization of blockchain technology is also presented, pinpointing some of its use cases and how its implementation can be beneficial to the various honest parties involved in transactions.

1.1 Motivation and background

More and more resources are allocated to the research and development of smart devices, connected between themselves forming a complex network with the intent of gathering information about the user. This relatively new business model allowed some companies to emerge as technological giants. Instead of selling a product, the user becomes the product. With huge amounts of data, including user's behaviours, interactions and preferences, an accurate image of the person is painted, which is then used to make deals with third parties. Users make use of these technologies, hoping the actors are honest and well intentioned. However, many examples of data mismanagement and cyberattacks do exist that invariably end up with sensitive, private data falling into the wrong hands. The fact that these centralized architectures are commonly used means that information regarding transactions is solely managed in a central, private server where it can be easily tampered.

This can lead to a commonly overlooked problem: lack of traceability. Since there isn't a mechanism that allows customers to verify with certainty the origin of the product being purchased, fraud can happen in multiple levels. A notorious industry specially vulnerable is the wine industry. Retail fraud can affect millions worldwide because from the customers' perspective, there isn't a feasible way to truthfully verify that a specific bottle indeed came from the producer indicated in the label. As the market for old and rare wine significantly increased in the past few decades, collectors are also in serious jeopardy. These wines, commonly costing thousands of euros, are often bought with the intent of not being drunk but rather kept indefinitely. Because these are so rare and expensive, very few can claim to be an expert on how they taste and collectors are often deterred from using methods that can compromise the wine. The conclusions are that tampering wine bottles is a lucrative business and fraud is hard to detect, creating a dangerous combination [1].

Another field particularly suffering from fraud is the pharmaceutical industry. A problem severely aggravated by the internet, as it is a platform that allows a trafficker or a seller to reach millions while maintaining anonymity. It is a market estimated to be worth 75 billion dollars per year, as the World Health Organization (WHO) calculates that up to 50% of online sold pharmaceuticals are counterfeit [2]. It's not just harmless ripoffs like water and random minerals contained in a pill. Traffickers have been upping their game, building labs to produce these drugs. Chemicals are used and the active substance is present, making it even more scary. The product is dangerous; the doses are wrong, the substances used have no quality and the know-how just isn't there to manufacture these sensible products, where errors can mean the difference between life and death [3]. As expected, pharmaceutical fraud has catastrophic consequences, specially in developing countries. Although pinpointing a number of victims is nearly impossible, estimates range from 100,000 to 700,000 deaths per year caused by counterfeits [2].

1.2 Contextualization

Industry 4.0 is changing the manufacturing environment. However, as this revolution continues to happen, some issues still pose a difficult challenge to overcome.

After some initial bad reputation gained by being associated with purchases in the Darknet, blockchain has now gathered the attention of giants such as Microsoft, Facebook, Google, IBM, among others who are actively researching and developing the technology. Due to the interest of these companies directly leading to a maturation of the technology, media coverage, money involved in cryptocurrency trading and peoples' concerns over transparency, the perspective is now changing.

By being an open environment that generates immutable data and isn't based on a trust model, blockchain technology promises to address some existing problems such as transparency and traceability across the value chain of a product's life cycle.

Users no longer need to trust a central authority since the architecture is now based on cryptography and said to be trustless, meaning participants don't need to trust other participants or mediators. Participants, also called peers since the network is Peer-to-peer (P2P), can check in real time the transactions that happened in the network with confidence that these are honest and indeed happened, adding a layer of transparency. The technology also enables complete, reliable traceability of transacted goods since it is immutable, meaning once a contract is consumed and added to the blockchain, it takes an amount of computational power that simply does not exist to alter it. And even in the remote possibility that it will exist somewhere in the future, widely used security protocols based on the same cryptographic methods will also be compromised, sending shock waves throughout Information Technology (IT) and deeply changing it.

Below, use cases for the technology are presented in Table 1.1 (adapted from [4]):

Category	**Usage of blockchain**
Data storage management	Management of access policies and users' data
	Management of data storage contracts
	Management of document storage contracts
	Immutable, tamper-proof log of events
Trade of goods and data	Purchases of assets
Identity management	Management of identity verification
	Public Key Infrastructure (PKI)
Other	Timestamping service
	Implementation of a lottery
	Banking applications (automated, distributed ledgers)
	Implementation of a social cryptocurrency

Table 1.1: Use cases of the blockchain

Multiple blockchains have emerged with intricate design and implementation specificities, aiming at actuating in different sectors. As the technology matures, more and more enterprises will be able to reap the benefits of using blockchain technology in their areas, hopefully meaning a step in the right direction.

1.3 Objectives

The work on this thesis aims at studying and subsequently implementing a platform using blockchain technology in a manufacturing environment. The desired outcome of this platform is to replace the traditional trust based model with a trustless one across all actors in the value chain of a product's life cycle.

Since it is inserted in the manufacturing environment, a few characteristics must be taken into account. Broadly speaking, the developed platform should try to achieve the following:

- **Transparency** Between the actors in a supply chain

- **Immutability** Stored information shouldn't be adulterated
- **Value to the consumer** The consumer should also benefit from the platform

1.4 Document overview

This document is divided into six chapters:

1. **Introduction** - Motivation and background sub chapter exemplifies and addresses the problems which leads to the existence of this thesis while Contextualization tries to give an insight into what blockchain technology can do to solve the previously mentioned challenges.
2. **State of the art** - The focus is on industry 4.0, the blockchain technology and how it can be helpful in the manufacturing environment.
3. **System architecture** - An overview of the concepts that support the architecture.
4. **Implementation** - A description of the actual developed work.
5. **Tests and validation** - The tests conducted to assess if the platform is working as intended.
6. **Conclusions and future work** - A brief look at what has been done and considerations regarding future versions.

Chapter 2

State of the art

This chapter will start by presenting an overview of Industry 4.0 and its related concept of integration in manufacturing. Followed by this, blockchain technology is introduced, underlining key concepts such as decentralization, cryptography and consensus. Finally, an attempt is made to understand how the manufacturing industry can benefit with the use of blockchain.

2.1 Industry 4.0

Technological leaps drastically change the way we produce goods and the term "industrial revolution" is used to denote these major paradigm changes [5].

The first industrial revolution, occurring in the XVIII century, was characterized by the introduction of mechanized systems, such as coal/water/steam powered machines in an effort to replace human labour [6]. The second industrial revolution was defined by the widespread use of electricity and its impact on industrial processes in the XIX century [6] [7]. The third industrial revolution, taking place in the XX century, is rather different from its predecessors, having consisted in digitalization and the use of IT to automate industrial processes [6].

Forwarding to the XXI century and a new paradigm is observable. Industry 4.0 (or fourth industrial revolution) is the term used to denote this shift. It promises to address necessary changes in industrial systems such as shorter development periods, individualization on demand, decentralization and resource efficiency [5].

Integration of the Internet of Things (IoT) into manufacturing is considered key in enabling the fourth industrial revolution [8]. The IoT allows objects to gather and communicate information between themselves in order to reach common goals [9]. There is an effort to make the IoT presence felt in everyday objects, even in our home's basic appliances [10]. The other major player in the scene is Cyber-physical Systems (CPS) [6]. CPS are the fusion between computation and physical systems, in the sense that there is a loop between computation and physical process where one affects the behaviour of the other and vice-versa [11]. By integrating CPS and the IoT, production becomes even better suited to meet the demand, hence creating an industrial revolution [12]. With more and more objects making use of these technologies, an intelligent network is formed characterized by a cycle of feedback, thus allowing them to have better decision-making processes. This also creates a huge amount of information (big data) that can be further used in analysis, to improve and optimize the existing processes [7]. Smart factories, i.e. the ones making use of these technologies, have a more flexible approach to manufacturing - producing custom products (even in batch sizes of 1) to meet the customer's requirements and to take advantage of opportunities, discovered by the analysis of data, through innovative services, while having a better productivity and resource efficiency [12].

As this revolution is happening, the IoT continues to morph into Internet of Everything (IoE), where not only machines are included in these smart networks, but a much more complex system emerges, encompassing people, machines, objects and data [8].

2.1.1 Integration

There are three key concepts that underpin the notion of industry 4.0: **(1) vertical integration** and networked manufacturing systems, **(2) horizontal integration** through value networks and **(3) end-to-end engineering across the product life cycle** [13]. A fourth characteristic; **(4) acceleration through exponential technologies**, might be considered to complete the loop. Figure 2.1 illustrates how they are connected

In more detail, we have:

1. **Vertical integration** refers to the cross-linking of several IT systems operating at different hierarchical levels in order to create more flexible and reconfigurable manufacturing systems [12]. As a result, these systems become much more adapt to not only rapidly react to fluctuations in demand or stock levels but also to faults that naturally occur, thus becoming better suited to the ever-changing environment. Through vertical integration, more data is generated and available to be collected. As we make use of the data, an emphasis is laid on resource (energy, human and materials) efficiency in an attempt to optimize existing processes [13].

2. **Horizontal integration** stretches beyond to encompass the cross-linking of several IT systems operating at different stages (logistics, warehousing, sales, production, marketing, etc) in the manufacturing and business processes, both intra-company and between different enterprises, creating local and global value networks [12]. Local value networks allow a greater flexibility and transparency within a company. By having this cross-linking of departments, the consumer is now able to actively

customize not only the manufacturing but also the ordering, planning and distribution of the product. A product's history is logged and available, making it traceable at all times [13].

Inter-cooperation, particularly helpful for small and medium sized enterprises, allows them to thrive in an otherwise harsh environment, typically dominated by the big, resourceful players. An example of a global value network is an Virtual Enterprise (VE). In this case, enterprises come together to share skills, competencies and information to better respond to business opportunities [14].

3. **End-to-end engineering** across the product life cycle describes the cross-linking and digitalization throughout the various stages of the product's life cycle since the acquisition of the materials until the product end of life while also including its conception and use [15].

4. **Acceleration** through exponential technologies alludes to how these technologies are acting as a catalyst to a transformation happening in every sector. The individualised solutions, flexibility and efficiency are allowing new disruptive business models to emerge, challenging the more rigid, traditional supply chains [13].

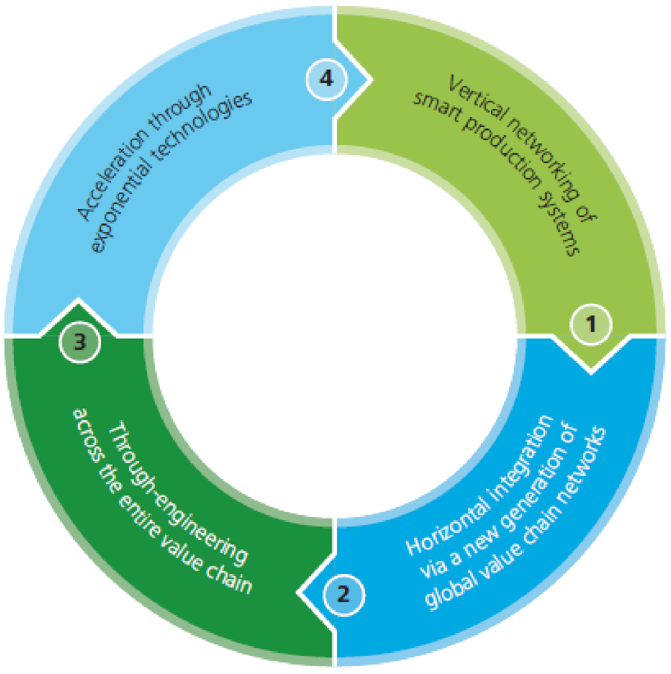

4
Acceleration through exponential technologies
Vertical networking of smart production systems
1
Horizontal integration via a new generation of global value chain networks
2
Through-engineering across the entire value chain
3

Figure 2.1: Industry 4.0 characteristics [13]

2.2 Blockchain

In 2008, the concept of blockchain was introduced by a person (or persons) by the name of Satoshi Nakamoto in his now famous paper [16]. In this paper, some inherent flaws of a trust based model are outlined: having a centralized institution as mediators increases the cost of the transaction as there is always a fee to be paid and the clients are forced to blindly put their trust upon these centralized companies, empowering them, when their data can easily be manipulated, tampered, shared or sold without consent to third parties.

So a new solution was proposed; a purely peer-to-peer electronic payment system, hence completely removing the need of a mediator, based on cryptographic proof of work instead of trust [16]. Multiple blockchain implementations which drastically differ from the original one exist. An emphasis is going to be given to Nakamoto's implementation due to being the pioneer that set in motion the whole phenomenon.

A blockchain is a collection of blocks linked together in a decentralized, universally accessible ledger [17]. As illustrated in figure 2.2, each block contains a cryptographic hash of the previous block, a timestamp and transactions validated through digital signatures. The information of each block is broadcast to every node, updated by miners, available to everyone and controlled by no one [18].

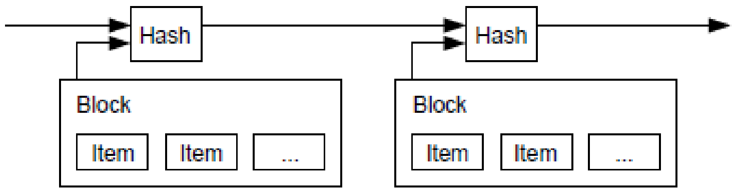


Figure 2.2: Each block includes the previous block's hash, forming a chain [16]

There are three types of blockchains (more later on hybrid implementations): Permissioned or private blockchains, permissionless or public blockchains and consortium. If everyone has access to the blockchain and is able to actively participate in the consensus mechanism the blockchain is said to be permissionless. In a consortium blockchain, only selected nodes have the ability to participate in the validation of transactions, therefore it isn't fully permissionless. Finally, in a private blockchain, the system behaves somehow similarly to a centralized system since a single entity controls which transactions are validated in the network [19].

2.2.1 Decentralization

In theory, a centralized system is a type of architecture where all or most of the processing is performed in a central node or central server. Solutions, resources and instructions are then passed by this central node to the attached client nodes. A decentralized system, on the other hand, is a system in which peers are equally privileged, where nodes operate on local information to accomplish goals, instead of operating based on a central influence. A simplified example of such systems is shown in figure 2.3.

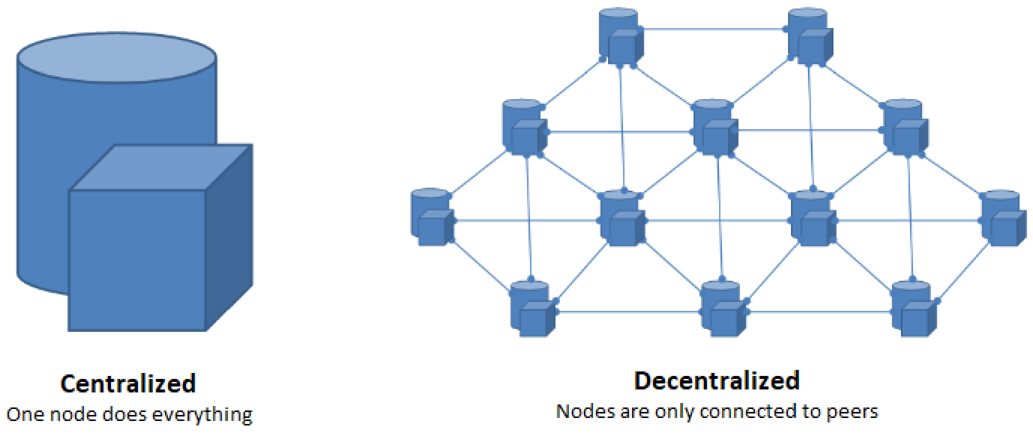


Figure 2.3: Centralized, at left, and decentralized systems, at right. Adapted from [20]

Decentralization has a number of benefits over its counterpart - it's considerably harder and more expensive to manipulate or compromise a decentralized system, since there isn't a central weak point of access. Decentralized systems are much less prone to failure, since there are multiple autonomous components working autonomously in the network. The fact that there is no central authority results in less censorship. The idea that participants can be rewarded ownership or economic stake (through tokens, or cryptocurrency) for producing work or value in the network, whereas in centralized systems only the ones controlling the central node reap the benefits of the network.

A blockchain, unlike ledgers controlled by traditional institutions, is replicated in every node of the network and accessible by everyone. Therefore, it's often hailed for being a decentralized system [17].

However, it's worth noting that almost no system is purely decentralized or purely centralized. It makes more sense to subdivide these broader terms and characterize systems in each category accordingly. This results in the creation of a spectrum (centralization in one hand and decentralization in the other) in which systems fall into, giving a better idea of how they are designed and dismissing the notion of a binary state [21].

To give a better idea where a blockchain is situated, three subcategories are considered [22]:

- **Architectural (de)centralization** - takes into account how many nodes make up the core of the network and how many of these can be compromised without the network failing completely

- **Political (de)centralization** takes into consideration how many individuals or entities ultimately control the network

- **Logical (de)centralization** relates to the behaviour of the network as a whole and characterizes whether nodes are differentiated between themselves or rather function as an amorphous swarm

A blockchain is architectural decentralized since there isn't a core exerting influence on the other existing nodes. Also, there isn't a central weak point to be exploited. While nodes can be attacked and exploited, the integrity of the whole network isn't at risk. It's also political decentralized because no individual or enterprise has special control over the network, every participating node is equally privileged. And finally, it is logically centralized since every node is running on the same protocol, behaving as a single computer.

2.2.2 Digital Signature

A digital signature is the digital counterpart of a physical signature. It's unique, meaning two persons shouldn't be able to produce the same signature. This allows to verify its authenticity, since it's possible to correspond a digital signature to its author. It's also desirable to tie it to the document it was originally signed on, so it cannot be replicated in another context without the person's consent.

To make it possible, each agent is assigned both a private (or Secret Key (SK)) and a Public Key (PK). The PK is used to validate the signature and the SK used for producing signatures [23]. A message M has a value that depends on both M and the signer's SK. The *PK* can be used to easily verify the authenticity of the signature PK [24][25].

A digital signature scheme:

1. **(SK,PK) <- generateKeys(keysize)** This method generates in polynomial time a matching key pair given an input size. The SK is kept private to the user, while PK is the key that everyone has access in order to verify the user's signature.

2. **sig <- sign(SK,M)** This method produces a signature that both depends on the SK and M.

3. **isValid <- verify(PK,M,sig)** This method returns the boolean value TRUE if SK is a valid signature for M using the PK and returns FALSE otherwise.

It is important to note that while **generateKeys** and **sign** should be randomized, **verify** should be a deterministic algorithm [21].

This methodology is what enables transactions in a blockchain context to happen. When two parties initiate a transaction, one uses its SK to sign the transaction data while the recipient can verify the authenticity of the signature using the other party's PK as shown in figure 2.4 [26].

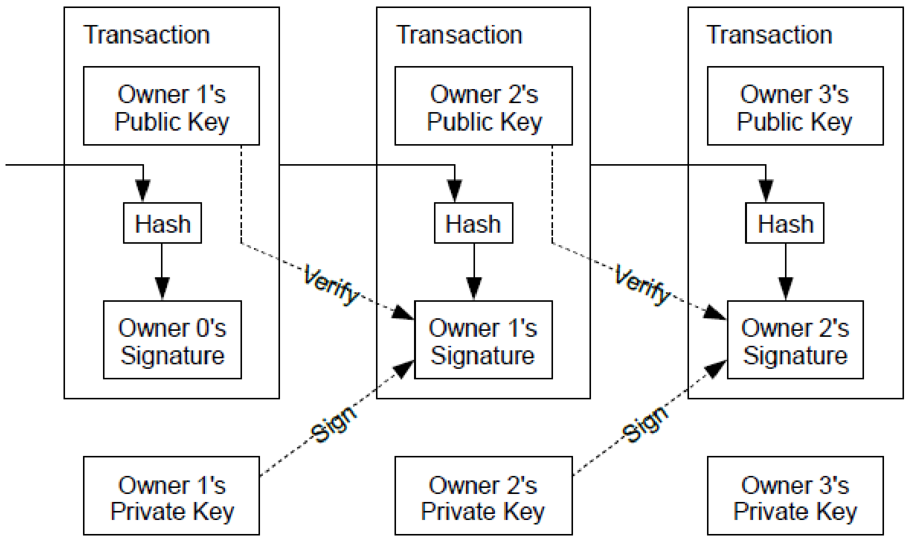


Figure 2.4: The usage of private and public keys in a blockchain. Adapted from [16]

2.2.3 Consensus

In a blockchain, for its attributed definition of being a single and global ledger to hold, its participants must agree on a number of aspects such as which transactions took place and their respective order. In order words, a consensus between all the nodes of the network must take place to make the system viable [27].

In systems possessing a central node controlling the network, this doesn't pose a challenge at all since most of the times this node has the authority to dictate what is added and what is rejected. However, in a blockchain, the solution to this problem isn't so trivial since there is no central authority and some nodes might be malicious. Therefore, a consensus validation protocol with the following properties is needed: it must terminate with all nodes agreeing on the value and that same value must have been produced by a honest node [21].

Transactions that happen are broadcast to the P2P network. The transaction data is then verified by the peers and periodically assembled in a block. However, a consensus hasn't been reached regarding this block and therefore it isn't automatically added to the blockchain [28].

In most implementations, the peers are encouraged to work on creating valid blocks that contain the transactions broadcast to them so it can be added to the blockchain. If they indeed manage to create a valid block, accepted by other peers, they are rewarded with tokens, or cryptocurrency specific to that blockchain [17]. These tokens can either be generated on the fly, i.e. the number of tokens in the network increases proportionally to the number of blocks in the blockchain, or rather be a fee of the transaction value. By using rewards to recognize those who produced honest work, people are encouraged to participate and more value is generated. If a peer so wishes, he/she can trade the tokens for physical currency. How much people are willing to pay for these tokens is entirely dictated by supply and demand of the market. In January 2017, Bitcoin broke the \$1000 mark, while on December of the same year it was almost worth \$20000, making it outrageously profitable for those who got into the train on time. Being a super volatile market, similar crashes have occurred as well [29].

2.2.3.1 Consensus mechanisms

Before continuing, it's important to understand the notion of a hash function. A hash function is a mathematical function used to map an input of arbitrary length into a value with a fixed length, normally called hash. For the same input, the function always returns the same hash value. Two almost identical inputs have completely different hashes. These functions are particularly useful in cryptography since it's easily verifiable that a given input maps to a given hash value. However, if the input is unknown, it's nearly impossible to reconstruct it only knowing its hash value, being trial and error the best known method. Secure Hash Algorithm (SHA)-256 is a cryptographic function whose many security protocols rely on. Since it produces a 256 bits output, it would take on average 2^{256} guesses to reach the desired input, an unfathomable large number [30].

Previously, it was stated that broadcast transactions don't automatically make it to the blockchain, but rather put on standby until a consensus is reached between the peers. In some implementations, in order for a block to generate consensus among the peers and be considered valid it must contain a proof (often, a cryptographic one) to go along with it.

A block containing transaction data (*tx*) and the previous block's hash, outputs an arbitrary string with a fixed length when ran through a hash function. To make the block valid, a hexadecimal string has to be attached, referred to as **nonce**. The condition is that a block, now containing the concatenation of the previous block's hash, *tx* and the nonce is considered valid only if its hash outputs a string that has a lower value than a current target T. So the condition $H(B.N) < T$ has to be met, where B is the block data, N is the nonce and H is the hash function. This T value fluctuates to ensure that a new block is generated after a given amount of time, different in each blockchain solution proposal [31].

As it was already noted, due to the fact that hash functions are completely unpredictable and pseudorandom, a peer can't just manipulate the hash value to quickly obtain the desired input through reverse engineering to quickly go around this challenge. The best known method to obtain the nonce is to try any random number, attach it to (*tx*) and the previous block's hash, run the hash function and check if the target is met. If it isn't, then repeat the process with another random number [17]. This method of trial and error is called mining and those who participate in it are called miners, since these participants are also performing a repetitive, arduous task hoping to "strike gold"[31].

As the number of miners increase and faster, specialized hardware is being deployed to the network, T becomes smaller to adjust to the increase of computational power and to ensure that a new block is formed in a regular fixed interval of time. Therefore, the process of mining greatly varies in difficulty. Nowadays, mining is often done professionally with specialized hardware such as Application-specific integrated circuits (ASIC). These chips are designed and built with sole purpose of mining. And even though everyone can participate and try to solve these hash puzzles, T can become small enough so that finding a solution is just not a feasible possibility for a common laptop or personal computer since, on average, it would take years to be the one finding a valid nonce. Therefore, not everyone in the network bothers with mining [21].

After the nonce for a given block is discovered, others can confirm that indeed the nonce is correct by simply hashing the block containing the nonce, *tx* and the previous block's signature. In other words, it's easily verifiable that the nonce is correct without having to go through all the work the miner had.

Miners compete between themselves to win the lottery of finding the nonce. The probability of a miner being the first to solve the challenge is directly proportional to the resources it can allocate to the task as more computational power means more possible tries in an amount of time. The first to find the nonce either gets a percentage of the transactions as a fee or a special *tx* is added in the block stating that he/she is awarded with a number of tokens (dependant on the blockchain and usually dynamic over time) as a reward for having solved the challenge [31]. This miner then broadcasts the valid block, with nonce included, which can finally be added to the blockchain as displayed in figure 2.5.

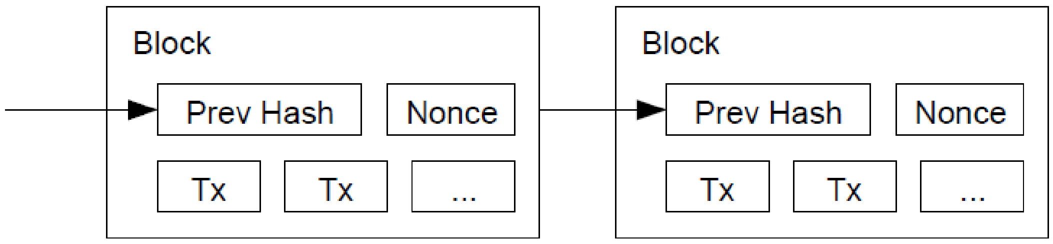


Figure 2.5: The nonce is aggregated in the block in order to reach part of the desired hash [16]

A valid block can be added to the chain by a miner. However, there is the possibility that multiple miners create different valid blocks based on the same preceding block. In a scenario where a compromised miner sets out to broadcast blocks containing fraudulent transactions, by chance he too can be the first to find the nonce. However, after a given amount of time, another miner will also find the nonce to a group of honest *tx* using the same preceding block the compromised miner used. In this situation, the blockchain becomes **forked** [31]. A forked blockchain is illustrated in figure 2.6.

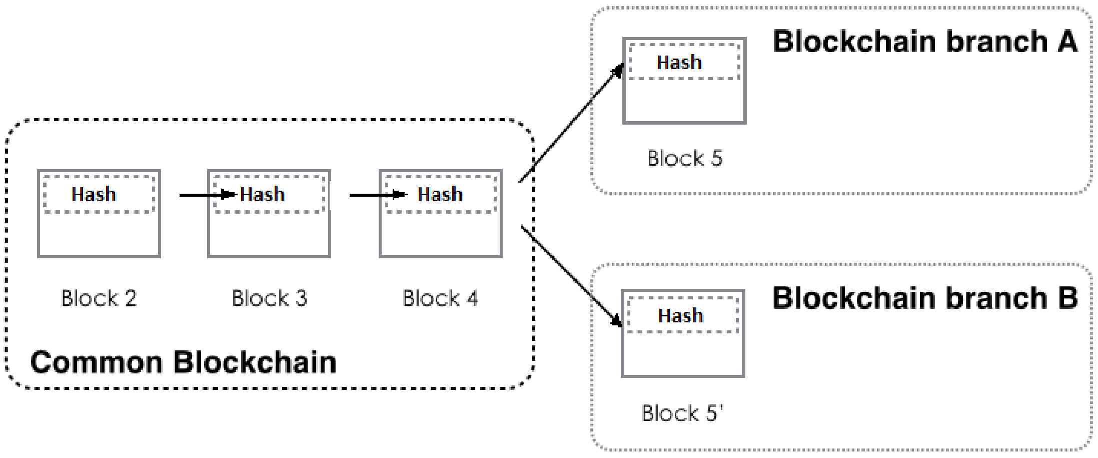


Figure 2.6: A blockchain fork. Adapted from [32]

In a forked blockchain, miners can then add subsequent blocks to any of the branches. In the previous scenario where the compromised miner that successfully found the nonce to the corrupted *tx*, that block is valid and is now in a branch. However, the pool of honest miners are not going to use the fraudulent block for their subsequent work because the previous corrupted *tx* was never used by them in the first place (if the compromised miner was to send the corrupted *tx*, honest miners would just discard it since the keys wouldn't verify). This leads to having the honest miners mining on the honest branch and the compromised miner or miners producing work on the corrupted branch. The protocol indicates that the valid branch is the one with more computational work put into it. So after a given time, the blockchain will discard the shortest branch and permanently add the branch that required the most mining power to generate to the blockchain [31].

Therefore, the integrity of the blockchain is assured if more than the majority of miners, weighted by computational power, is honest since it means that there is more than 50% chance that a block will come from a honest miner and fraudulent ones are discarded. This makes collusion between a group of individuals to their benefit at the expense of others extremely hard to happen since possessing the majority of computational power between themselves in a given blockchain is rather unmanageable [21]. The idea behind this extremely costly (in both electric energy and expensive hardware) consensus mechanism is to select nodes based on a resource that in theory nobody can monopolize such as computational power.

As every block depends on the previous block and so on, it is extremely difficult to alter any past transaction history since altering one block would require producing a new proof of work for every single subsequent block making it an infeasible task [17]. For this reason, a blockchain is commonly referred to as immutable.

This system is called Proof of Work (PoW). It is the original consensus mechanism proposed in the Satoshi Nakamoto's paper [16]. It is still widely used in platforms like

Bitcoin and Ethereum [33]. The major downside is that it is extremely costly energy wise. Estimates show that bitcoin mining consumes more electricity than a country like Ireland [34].

Due to this, other consensus mechanisms have emerged. One particularly gaining relevance is Proof of Stake (PoS). In this system, peers invest tokens resulting in a stake on a given block. The chance of becoming the validator for that block is proportional to the to the size of the block. Thus, if an individual owns 1% of all of the tokens in the network, he/she can expect to be the validator of 1% of all the blocks [19]. The same principle of the 51% attack still applies but no longer with with computational power. As long as no individual controls more than 50% of all existing tokens in the network, its integrity is, in theory, ensured.

2.2.4 Hybrid Implementations

The previously discussed PoW and PoS used in many permissionless blockchains result in a no privacy environment in the network due to the fact that the transactions rely on the approval by other nodes. This lack of confidentiality is often not particularly useful for a business. As an example, a company might want to give preferential treatment to a costumer for a number of reasons, such as incentivizing additional spending or as part of a loyalty program [35].

Public blockchains will always be more transparent and safe but if blockchain technology is to become relevant in the enterprise world, it has to take into consideration not only the customer's wishes but also the company's interests.

This is the gap hybrid implementations try to fill. They greatly vary between themselves implementation wise. It's assumed that because they are to be used in a business case, there is an environment of partial trust between the participants which allows for flexibility in the whole design process. Now, the participants' identity isn't unknown. All of them are registered and verified. These hybrid implementations often try to combine the benefits of both a private and a public blockchain: allow some sort of privacy on Business-to-business (B2B) transactions while also sharing new information on an open ledger with the consumer regarding their product [36].

Multiple hybrid approaches are emerging to meet the enterprise world's needs and desires. Hyperledger Fabric, the most notable framework of the Hyperledger project, developed by Linux Foundation and backed by big names not only of the tech world but also financial and software companies and even academic institutions allows users to set up channels between the participants that form a network [35]. Libra, a blockchain network with its own cryptocurrency, developed by Facebook and expected to launch in 2020, also takes a hybrid approach in an attempt to become the dominant currency used. Although it is decentralized on paper, the occurring transactions will not be verified by all the nodes, but rather a collection of validators, selected partners that will form the membership of the Libra Association. Also, to combat the volatility of traditional cryptocurrencies' value, thus adding stability and some intrinsic value which facilitates widespread adoption, Libra's value will be tied to other reserve assets such as other currency (U.S. Dollar, Euro, Japanese Yen, among others) and government securities from central banks [37].

In many hybrid implementations, often there isn't a predefined consensus mechanism. Hyperledger Fabric, a modular network, allows users the flexibility to select the most fitting consensus mechanism [38].

2.2.5 Smart Contracts

As blockchain technology advances, its applications have gone far beyond simple token transactions. Recent blockchain technology allows the carrying out of contracts where payment is dependant on the state of a number of variables agreed by peers making the contract. These type of contracts are referred to as smart contracts. In other words, automated, self-executing scripts whose execution doesn't rely on any human interference [27]. Since these reside in the blockchain, they possess a unique address. When a transaction is addressed to it, they will trigger and execute based on the data contained in the transaction. It's a trustless model, since there isn't the need to trust the peer we are trading with nor a third party that usually mediates transactions.

To illustrate the concept, a simple practical case is considered. Peer A is a collector of the asset X and wants to acquire as many as possible. He deploys a smart contract in a blockchain containing the following methods: a "deposit"method allowing him to store currency in the contract, a "trade"method that sends back currency for everyone that deposited 1 unit of X and a "withdraw"method allowing peer A to withdraw all of the assets X in the contract. Peer B, also participating in the same network of peer A, can now access the smart contract. He can now send to the contract 1 unit of X that he owns, which triggers the "trade"method that ends up with him getting currency. Peer A can now make use of the "withdraw"method to withdraw the 1 unit of X that belonged to peer B. The contract checks the signature of peer A to validate that it is indeed the contract owner withdrawing and the request goes through. All of the transactions are recorded and eventually added to the blockchain. Peer A will always be able to tell with certainty where that unit X came from and for how much it was acquired. This record is now theoretically immutable (unless it was designed to self-destroy) and was made without any third party in between.

A number of characteristics are worth mentioning; a smart contract has its own state and is able to take custody over both goods being traded (previously, transactions in a ledger were a matter of deleting and adding rows); a decent smart contract should contemplate every possible scenario and act accordingly while being deterministic, making its behaviour completely predictable [39].

2.2.6 Blockchain in Manufacturing

A supply chain is a set of organizations, parties and people directly involved in the various stages of a product's life cycle until it is delivered to the customer. Before getting to the consumer, products often travel through an immense network of retailers, distributors, transporters, storage facilities and suppliers, comprising the phases of design, production, delivery and sales [40].

Supply chains have played a significant role in globalization since they have been one of the catalysts for the growth of both manufacturing and trade worldwide [5]. However, there is more than meets the eye. Today's global supply chains have become rather complex and are generally slow and inefficient due to having to coordinate many parties in multiple countries, subject to several different laws and regulamentation.

A representation of the tedious and complicated administrative procedures involved in an import is shown in figure 2.7:

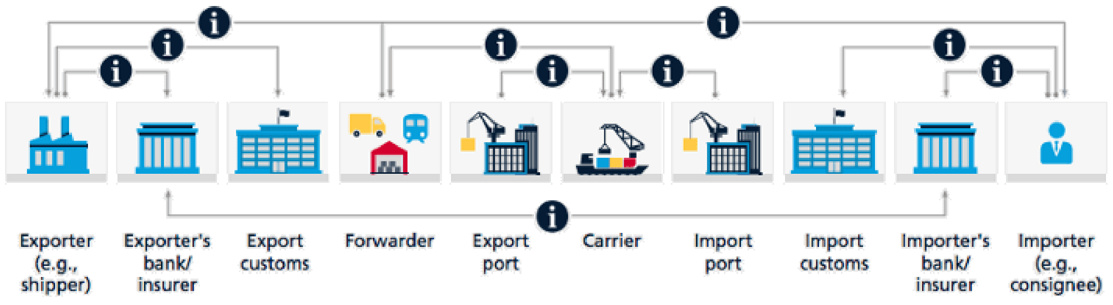


Figure 2.7: The information flow in international trade is both complex and documentation heavy (adapted from [41])

While supply chains are the backbone for any business in the manufacturing industry, generally not much information is shared to the outside. The consumers are often oblivious as to where are their products coming from and who is producing them, who's supplying what in the long supply chain and the several stages the product went through. Some certifications are welcomed but at the end of the day, most of the times consumers have no choice but to accept the claims regarding their product's information, as verifying certificates can prove to be both impractical and costly. Furthermore, the credibility of such certificates is oftentimes undermined by misconducts, usually fueled either by corruption or greed [42]. The Volkswagen emissions scandal is an example of such abuse [43].

Another challenge companies have to face is related to transparency in the supply chain, as truthful information regarding suppliers in deeper tiers (i.e. suppliers for the suppliers of a company) can be scarce [44]. Transparency and visibility in a supply chain is considered key for any business. It enables new analytics which leads to better decision making, optimizing processes and smoothing the information flow between all parties involved. Nevertheless, this can prove to be a hard task. It often involves putting a great deal of trust in every actor in the supply chain, trusting they are both well intentioned and capable from a technical standpoint [45].

Chapter 3

System architecture

In this chapter, a macroscopic view of the work is presented without discussing implementation choices. A brief overview, followed by a more in depth description of its underlying concepts is given.

3.1 Brief overview

As explained in the previous chapter, a purely permissionless, decentralized blockchain platform isn't exactly suited to a business environment due to a number of reasons, confidentiality being the one that stands out the most.

We'll start by defining retailers and suppliers in our platform. Retailers are the companies that sell goods directly to the public (customers). Suppliers are companies that provide something to the retailers. As seen in the previous chapter, these networks can become quite large rather quickly (retailers have a lot of suppliers, these suppliers then can have a lot of suppliers and so on) which can lead to wrong information.

Therefore, the goal is to develop a platform that connects suppliers, retailers and customers, each having their own interface. Companies get connected with their supply chain partners. All transactions between business partners get registered in several ledgers in the form of blocks, which form a blockchain. At the same time, a costumer that purchases a product from a retailer has access to information regarding transactions that are behind the making of said product. Hopefully, this can bring two major benefits: 1) companies can have a better insight on their supply chain partners and 2) costumers can better know the suppliers behind their final product.

This is possible with the creation of several ledgers that have precious information regarding previous transactions. The most important aspect is to ensure that these ledgers remain truly immutable. If otherwise, then the whole platform becomes pointless. In a decentralized system, this is somewhat achievable since each peer holds a copy of the ledger and to compromise it, one would need to hack into each of those copies to get the desired outcome. With enough nodes, this proves a practically unfeasible task. However, it's worth mentioning that there are no flawless architectures security wise. Due to the impossibility of having hundreds, even dozens peers to simulate a decentralized network, a centralized solution was developed instead. The underlying concepts of the platform as well as the steps taken to try to ensure the integrity of the ledgers will be explained in this chapter.

What businesses is the platform aiming to serve? It's a blockchain hybrid implementation with the intent of storing information in an immutable way and increase transparency between companies and between company-customer. Therefore, the platform is intended for every business and respective customers.

An abstract overview of the platform is displayed in figure 3.1. In this image, we have suppliers, a retailer and a customer. The retailer can see the past transactions of its three suppliers while the customer can check the transactions that were involved in product.

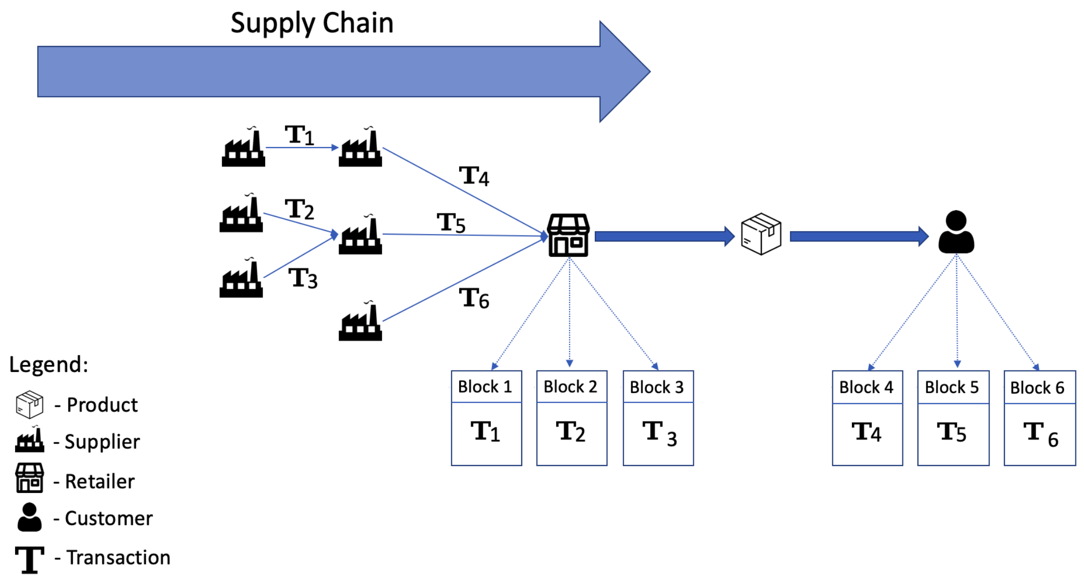


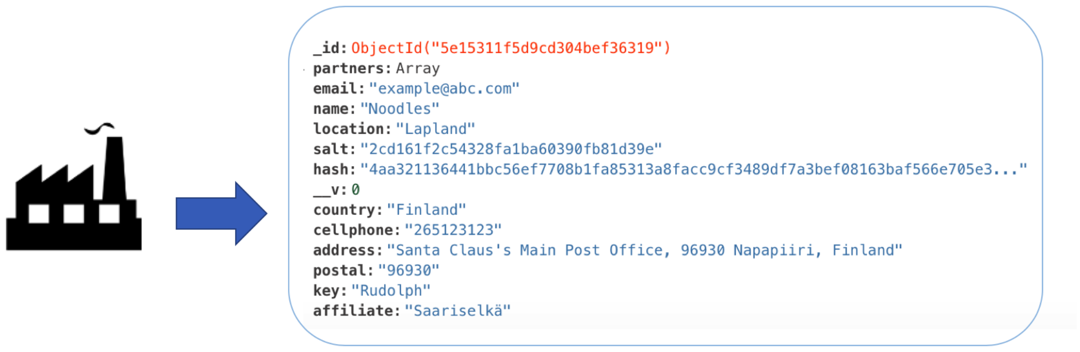
Figure 3.1: An overview of the platform

3.2 Identity

Unlike the original bitcoin's blockchain implementation, the anonymity of the different actors isn't an option. Their identity is a hard requirement since the idea is to increase the transparency in a manufacturing environment. The goal is not to enable transactions but rather to know who made them.

Whenever a company or a company's branch joins, a number of fields are registered to identify the new member. These fields can vary but the idea is to make it clearly identifiable to its business partners.

An example of the information registered is displayed in figure 3.2.



```
_id: ObjectId("5e15311f5d9cd304bef36319")
partners: Array
email: "example@abc.com"
name: "Noodles"
location: "Lapland"
salt: "2cd161f2c54328fa1ba60390fb81d39e"
hash: "4aa321136441bbc56ef7708b1fa85313a8facc9cf3489df7a3bef08163baf566e705e3..."
__v: 0
country: "Finland"
cellphone: "265123123"
address: "Santa Claus's Main Post Office, 96930 Napapiiri, Finland"
postal: "96930"
key: "Rudolph"
affiliate: "Saariselkä"
```

Figure 3.2: The company's information that is registered

Customers, on the other hand, have a much less active role in the network. They never get registered since they aren't participating in the transactions, meaning their identity is irrelevant. However, they also greatly benefit from the platform. As outsiders, they come into play whenever they acquire a product in its final form that belongs to a participating company. A ledger was created specifically for their product, containing relevant information they can inspect carefully. Later on, this will be explained in greater detail. This essentially tries to demolish the traditional trust based system in place. As long as the ledgers aren't compromised, a customer can have much more insight regarding his product

3.3 Peers

The peers are the active participants in the network. In this architecture, there isn't a single common ledger to everyone. It doesn't make sense for two companies that have nothing to do with each other, to be able to check each other's past transactions. In a scenario where a give company is testing out new products, it would be catastrophic to have that information out in the public. Therefore, transactions should only be visible to other relevant companies. The solution is the creation of several ledgers with different permissions. This way, a company can only certain ledgers which results in a degree of confidentiality.

The existence of multiple ledgers allow peers to join multiple conglomerates in accordance to what makes sense business wise. When a peer registers in the platform, some sort of unique identifier must be created as well. This way, a partner can be added or removed from networks. Apart from this, every time a peer registers, a new ledger is created specifically for that partner. As its business partners also join the platform, the original peer invites them to join his subset of the network. This brings many benefits as will be explained later.

Ideally, if every member of a supply chain of a given company is present and active in the platform, great insights can be generated which results in much more transparency.

Because peer A (PA) might have peer B (PB) and peer C (PC) as business partners but PC then has peer D (PD) and peer E (PE) as its own partners, the possibility to be in multiple networks within the platform must be supported.

A representation of several subset of networks within the whole network is displayed in figure 3.3. P stands for peer and the next letter indicates which peer. So ten different example peers are represented (A to H).

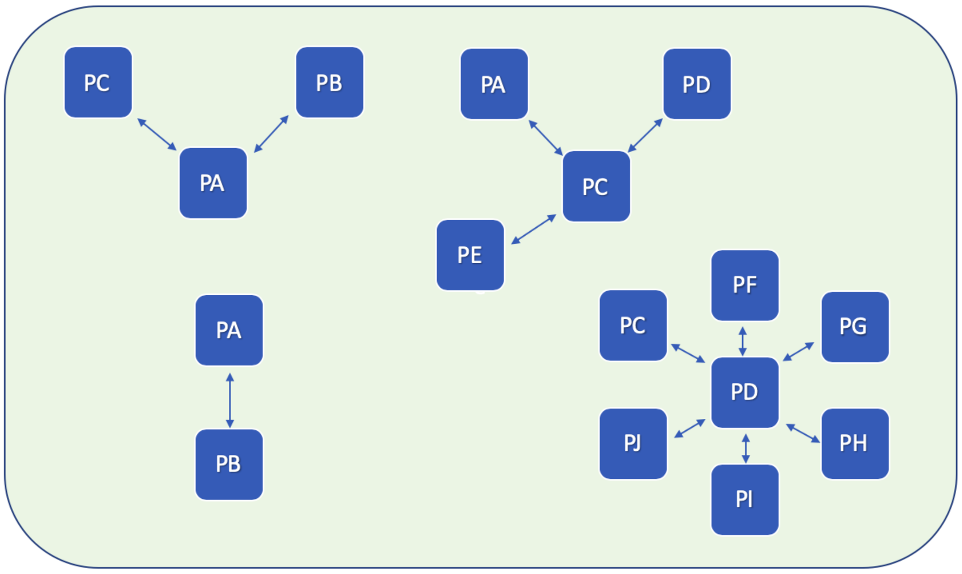
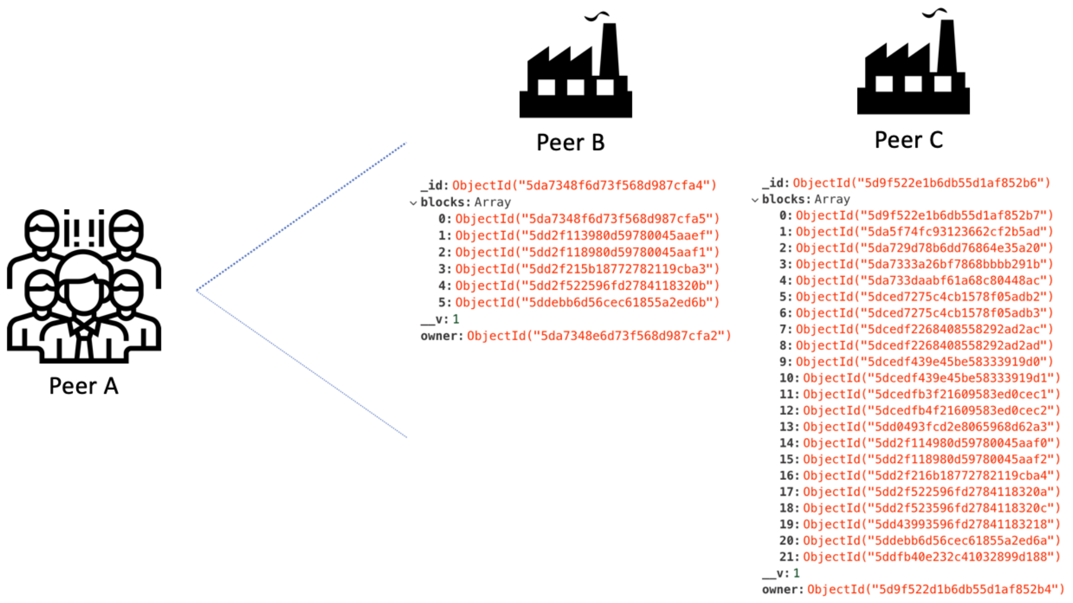


Figure 3.3: Several conglomerates in the network, ensuring confidentiality

It is established that peers should be in a conglomerate with their respective business partners. This brings a number of benefits to the participants. Firstly, it allows the inspection of your business partner's ledgers. This is of particular importance. This is a key point since this is what enables an increase of transparency between the several actors that comprise any given supply chain. By being allowed to so, a peer can inspect its partners past transaction and be certain that it isn't being wronged by its suppliers. This is vital on certain businesses. Even giant companies have many suppliers and therefore, there are a lot of raw materials handling and manufacturing processes that go beyond their scope. An electronic device company might be supplied from capacitors, to memory modules to processors. However, being in the final step of the supply chain just before the customer, these companies are required to make sure everything is regulation compliant and has quality. The ability to inspect its suppliers transactions empowers the company. Being able to truthfully check to whom and from where are these suppliers buying from can give great insight in making sure it is not being wronged.

It's important to note a relevant question: How far should a company be able to inspect his suppliers? As in, should the company be able to inspect their suppliers' suppliers by themselves? In our example, peer A is supplied by B and C. Naturally, peer A is able to inspect the latter two. But C is supplied by D. Should peer A be able to inspect D and its suppliers? A line must be drawn, otherwise due to the interconnectivity and globalization, company A could quite possibly be able to inspect every peer present in the platform. In this implementation, a decision was taken to only allow a peer to directly inspect its business partners.

An illustration is displayed in figure 3.4 (the objects are then further inspected).



```
_id: ObjectId("5da7348f6d73f568d987cfa4")
⌄ blocks: Array
   0: ObjectId("5da7348f6d73f568d987cfa5")
   1: ObjectId("5dd2f113980d59780045aaef")
   2: ObjectId("5dd2f118980d59780045aaf1")
   3: ObjectId("5dd2f215b18772782119cba3")
   4: ObjectId("5dd2f522596fd2784118320b")
   5: ObjectId("5ddebb6d56cec61855a2ed6b")
__v: 1
owner: ObjectId("5da7348e6d73f568d987cfa2")
```

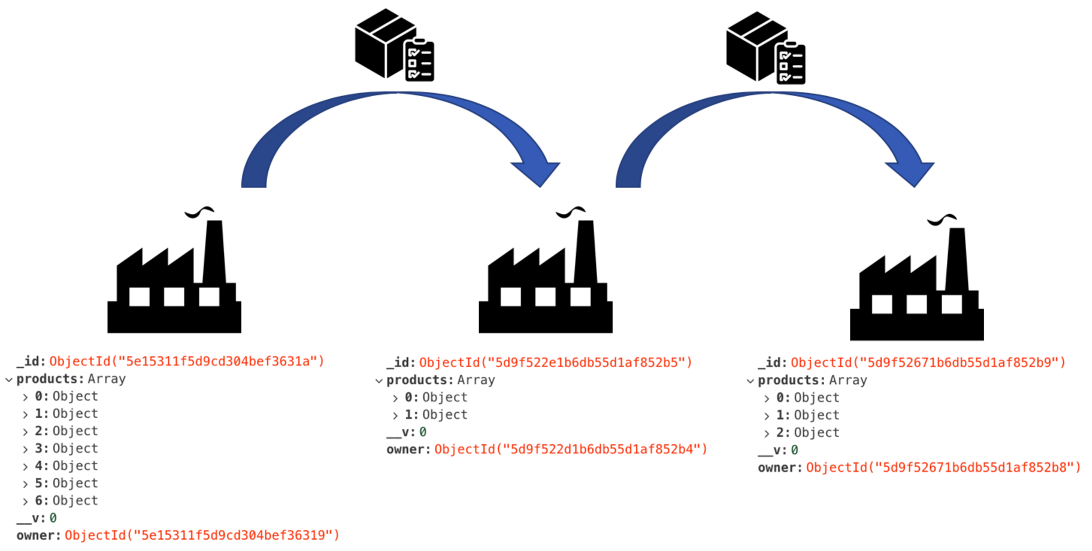
```
_id: ObjectId("5d9f522e1b6db55d1af852b6")
⌄ blocks: Array
   0: ObjectId("5d9f522e1b6db55d1af852b7")
   1: ObjectId("5da5f74fc93123662cf2b5ad")
   2: ObjectId("5da729d78b6dd76864e35a20")
   3: ObjectId("5da7333a26bf7868bbbb291b")
   4: ObjectId("5da733daabf61a68c80448ac")
   5: ObjectId("5dced7275c4cb1578f05adb2")
   6: ObjectId("5dced7275c4cb1578f05adb3")
   7: ObjectId("5dcedf2268408558292ad2ac")
   8: ObjectId("5dcedf2268408558292ad2ad")
   9: ObjectId("5dcedf439e45be58333919d0")
   10: ObjectId("5dcedf439e45be58333919d1")
   11: ObjectId("5dcedfb3f21609583ed0cec1")
   12: ObjectId("5dcedfb4f21609583ed0cec2")
   13: ObjectId("5dd0493fcd2e8065968d62a3")
   14: ObjectId("5dd2f114980d59780045aaf0")
   15: ObjectId("5dd2f118980d59780045aaf2")
   16: ObjectId("5dd2f216b18772782119cba4")
   17: ObjectId("5dd2f522596fd2784118320a")
   18: ObjectId("5dd2f523596fd2784118320c")
   19: ObjectId("5dd43993596fd27841183218")
   20: ObjectId("5ddebb6d56cec61855a2ed6a")
   21: ObjectId("5ddfb40e232c41032899d188")
__v: 1
owner: ObjectId("5d9f522d1b6db55d1af852b4")
```

Figure 3.4: Peer A can inspect its businesses partners' chains

It's also important to notice a key issue. This doesn't stop fraud per se. Every system will have its intrinsic flaws since it is man made. A company might still record false information in the platform. For example, a company might register that it gathered some certain raw materials from place B while they were expected to get it from place A. Being a private platform, it doesn't hold any sort of legal power that compels companies to be honest and truthful. It's up to certain entities such as regulators and other partners to ensure everything is done in accordance with what's stipulated for their own benefit.

Another major benefit of peers being grouped with their partners relates to transactions. A sort of inventory is associated with every peer in the network. This allows the listing of products with fields containing relevant information like the name of the product, the manufacturer, the amount, the value, an unique identifier (different manufacturers might produce two seemingly identical but different products like bolts so it's important to make the distinction) among others. By being in a conglomerate with other partners, a peer has access to the inventories of each of them. With this in place, a peer can then make purchases in the network that automatically get compiled in the form of immutable blocks (more on 3.4) that are then added to a ledger.

An illustration of peers inspecting inventories and making purchases is displayed in figure 3.5.



```
_id: ObjectId("5e15311f5d9cd304bef3631a")
⌄ products: Array
  › 0: Object
  › 1: Object
  › 2: Object
  › 3: Object
  › 4: Object
  › 5: Object
  › 6: Object
__v: 0
owner: ObjectId("5e15311f5d9cd304bef36319")
```

```
_id: ObjectId("5d9f522e1b6db55d1af852b5")
⌄ products: Array
  › 0: Object
  › 1: Object
__v: 0
owner: ObjectId("5d9f522d1b6db55d1af852b4")
```

```
_id: ObjectId("5d9f52671b6db55d1af852b9")
⌄ products: Array
  › 0: Object
  › 1: Object
  › 2: Object
__v: 0
owner: ObjectId("5d9f52671b6db55d1af852b8")
```

Figure 3.5: Peers in the same network can inspect each other's inventory and make purchases amongst themselves

Two important questions arise: how is the consensus mechanism processed and how do the assets and the monetary compensation switch hands? To tackle the first question, its important to remember that the platform is to be used in an environment of partial trust and theoretically, peer A isn't too skeptical about transactions between peer B and C. Because transactions only directly concern two parties and the network itself isn't decentralized, other peers don't get a vote on if the transaction is to be deemed valid and subsequently added to the network. Therefore, the consensus mechanism itself is made between two parties (buyer and seller) and it simply involves them both agreeing with each other on the several aspects that make up a transaction. This was the implemented approach. If the system was to be decentralized or was to be used in an environment where it is important to give parties this voting power (for example, in a network where users have anonymity) then a consensus mechanism would have to be devised. To answer the second question: just like they normally do. Just like many other blockchains implementations, there is no cryptocurrency associated with this specific platform. This would be possible, but would require additional resources to develop . Therefore, the platform doesn't support the notion of digital wallet. The idea isn't to replace traditional banks, but to update an otherwise dangerous way of storing information through the usage of an immutable registry. So, peers are responsible with moving goods, both material and currency, among themselves just like they normally do. Once a transaction is completed, they are encouraged to register the details in the network. The problems associated with damaged or stolen goods have existed for as long as supply chains exist. Blockchain can't solve those problems since it doesn't have legal power and it isn't a way of transporting physical goods either. It is simply a registry of transactions between parties.

Peers also have another important feature as pointed out in 3.2 that directly relates to customers. Companies that directly sell to customers can do so with increased transparency.

A final product is always the result of several transactions that happened between the several actors in a supply chain. In light to this, a company has an option within the platform to assemble a product with the intent of selling it to a customer by selecting all the transactions (that by now is already in the form of blocks, as will be explained in 3.4) of supplies that make up that specific final product. Then, a new ledger is created containing all this information. This ledger is then inspectable by the customer, through a serial key. This naturally can give a great insight to the customer

It increases transparency between the customer and the company. Now, the customer knows exactly what is the seller buying off suppliers (as well as a timestamp of purchases) and who these suppliers are. As long as the ledger's integrity isn't compromised, this information is reliable.

It's important to point out that this newly created ledger, that will be made available to the customer, only contains transactions that happened in the network. This essentially means two things: 1) there won't be any information regarding products that the company itself manufactures that are later integrated in the final product (constituents) and 2) there also won't be information regarding the manufacturing processes a product went through. 2) This doesn't seem of particular relevance, since the ordinary customer isn't too keen on understanding the manufacturing processes as long as the quality of the product is sound. And although solving 1) would certainly be interesting, oftentimes the fact that those constituents are associated to the well known company is enough to please the customers.

3.4 Blocks

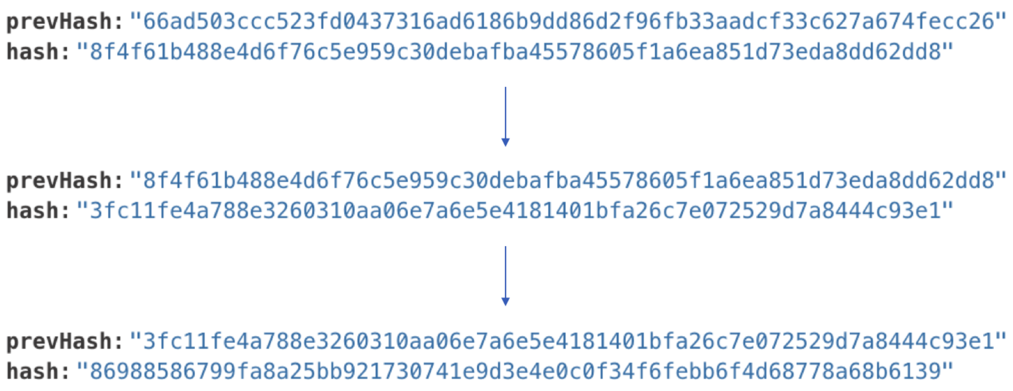
Blocks are the essence of any blockchain. The ability to store immutable information in the form of blocks remains the core of any blockchain network. And this is quite telling since different implementations can vary a great deal between themselves.

A block consists of compiled information about one, or more, transactions that took place in the network. In this specific platform, a block is created for a single transaction since the network won't have to withstand much traffic.

Before diving into more details, let's talk about integrity since it's vital to ensure blocks remain logical throughout time. If it was possible to change its fields after they are set in the network, then it would be just like a normal regular system and transactions could be adulterated. To tackle this problem, the solution used is the solution many other blockchain implementations use; through the usage of hash functions. Blocks possess several fields of information and two of the most important are related to its hash and to the previous block's hash which is related to its location.

As it was explained, hash functions are like digital signatures since every input has its own. Therefore, whenever a block is compiled, it's ran through a hash function to obtain the corresponding hash. Any number of the block's fields can be ran through the hash function as long as the process is consistent. However it's important to get differentiating fields through to ensure different transactions won't output the same hash (it's likely that a transaction of a specific product between two companies keeps repeating and thus it's important to feed the function more than the buyer/seller/product fields). A way to ensure this is to run through the hash function every single field, including the timestamp and the previous block's hash.

An illustration of three partial blocks in the same ledger is shown in figure 3.6.



```
prevHash: "66ad503ccc523fd0437316ad6186b9dd86d2f96fb33aadcf33c627a674fecc26"
hash: "8f4f61b488e4d6f76c5e959c30debafba45578605f1a6ea851d73eda8dd62dd8"

prevHash: "8f4f61b488e4d6f76c5e959c30debafba45578605f1a6ea851d73eda8dd62dd8"
hash: "3fc11fe4a788e3260310aa06e7a6e5e4181401bfa26c7e072529d7a8444c93e1"

prevHash: "3fc11fe4a788e3260310aa06e7a6e5e4181401bfa26c7e072529d7a8444c93e1"
hash: "86988586799fa8a25bb921730741e9d3e4e0c0f34f6febb6f4d68778a68b6139"
```

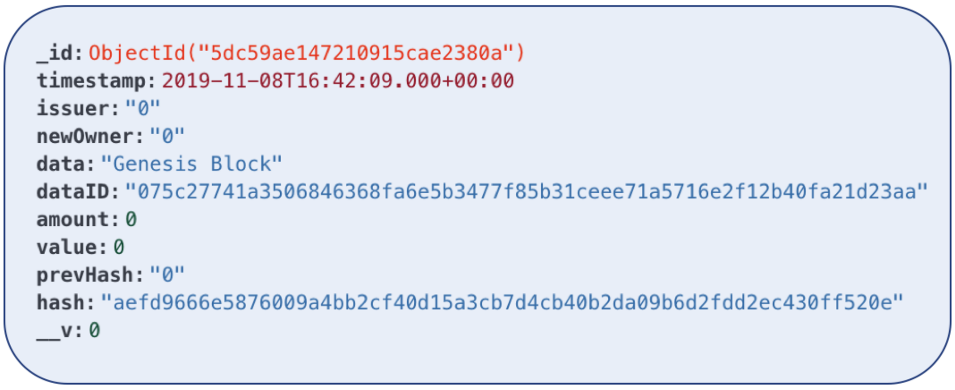
Figure 3.6: Blocks always have an unique hash as well as the previous block's hash

Obviously, in order to feed the hash function the previous block's hash field, we need to already know it first. This essentially means to know to which ledger does the block in the making belongs to. This isn't too hard since given the circumstances of the transactions, it's always known who is buying and selling and therefore where to place it. The listed item by the seller already contains most of the information needed such as the identity of the seller and the product description. To know who is buying is also trivial. Since a peer has to be logged into the platform to purchase, we can make use of the browser's token that is making the request to the network to get the identity of the buyer.

As a peer joins the network, a new block is always created. This block is called the *Genesis Block* or *Block zero* and it signals the beginning of that peer's ledger (hence the name Genesis) on which additional blocks are sequentially added. In some other blockchain implementations, namely those with a common ledger to everyone, this index is particularly useful since it serves as a good metric to judge the network's traffic due to blocks being generated every X minutes. Also, since cryptocurrencies are limited, indexes allow to make predictions about when are these going to cease. In this platform, neither problem is valid and therefore indexes aren't needed.

Even though the Genesis block mostly contains zeros, its hash is still likely unique unless two peers registered on the same date at the same exact time, since a timestamp service is used to generate the hash.

An example of a Genesis Block is shown in figure 3.7.



```
_id: ObjectId("5dc59ae147210915cae2380a")
timestamp: 2019-11-08T16:42:09.000+00:00
issuer: "0"
newOwner: "0"
data: "Genesis Block"
dataID: "075c27741a3506846368fa6e5b3477f85b31ceee71a5716e2f12b40fa21d23aa"
amount: 0
value: 0
prevHash: "0"
hash: "aefd9666e5876009a4bb2cf40d15a3cb7d4cb40b2da09b6d2fdd2ec430ff520e"
__v: 0
```

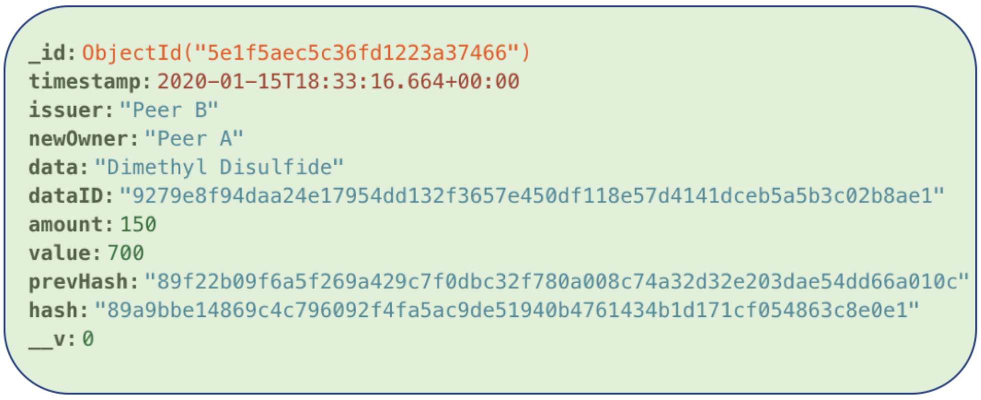
Figure 3.7: An example of a Genesis Block

Whenever a transaction takes place in the network between peers, it gets compiled into a block in the relation of one transaction per one block. As a result, the blocks must contain fields of information relevant to the proceedings. In this case, the following categories make it to the block:

- **timestamp** - a digital record of the time of occurrence of the block's creation
- **issuer** - The entity selling the product
- **newOwner** - The entity buying the product
- **data** - Product information, particularly its name
- **dataID** - The output of *hash(issuer + data)* to avoid ambiguity of same product with different manufacturers
- **amount** - The quantity of products sold in one transaction
- **value** - The monetary worth of the transaction
- **prevHash** - The previous block's hash
- **hash** - The output of *hash(timestamp + issuer + newOwner + data + dataID + amount + value + prevHash)*

As these fields get completed with the relevant information, transactions are easily identifiable. These are the blocks that are made available for further inspection by both the peers and the customers.

An example of a block in the network is shown in figure 3.8.



```
_id: ObjectId("5e1f5aec5c36fd1223a37466")
timestamp: 2020-01-15T18:33:16.664+00:00
issuer: "Peer B"
newOwner: "Peer A"
data: "Dimethyl Disulfide"
dataID: "9279e8f94daa24e17954dd132f3657e450df118e57d4141dceb5a5b3c02b8ae1"
amount: 150
value: 700
prevHash: "89f22b09f6a5f269a429c7f0dbc32f780a008c74a32d32e203dae54dd66a010c"
hash: "89a9bbe14869c4c796092f4fa5ac9de51940b4761434b1d171cf054863c8e0e1"
__v: 0
```

Figure 3.8: A regular block of the network

3.5 Ledgers

As mentioned before, ledgers are collections of blocks appended together and it can never be retroactively changed. By now, the ledgers that get generated along the navigation in the network have been approached. Whenever a peer registers, a new ledger is created containing only the genesis block initially. As a peer participates in the network, its corresponding chain gets longer in terms of number of blocks registered. It's worth noticing that whenever a transaction is made, two blocks are generated. Two because, that transaction concerns two parties and therefore there are two ledgers that need to be updated. The blocks are nearly identical with the exception of *prevHash* since that field relates to the ledger they will be appended to.

The longer a ledger is, the more past transactions are safe. It's not blocks themselves that grant an increased layer of security to the network but rather the way they are linked together forming a chain.

Because there isn't a consensus mechanism, there is no need to deal with the famous 51% attack nor the Byzantine fault condition. However, transactions can still be subject to undesired to change. Adulterating a transaction (as in, modifying an existent block) results in a completely different block's hash. This means that the subsequent block will have a *prevHash* that doesn't correspond to the previous block. Therefore, there are two major ways to check if the system is compromised: 1) Check if the visible hash codes indeed correspond to the other information fields present in the block and 2) Check if the sequence of *Hash* and *prevHash* of every consecutive blocks remains correct. Tampering one block means having to tamper every single subsequent block which proves itself an arduous task to do.

A new ledger is formed whenever a peer assembles a final product by selecting the relevant transactions. This new ledger is just a collection of loose blocks, since these in all likelihood aren't linked in any way. The corresponding transactions likely were done with other unrelated arrangements being done in between. This results in the blocks that make it to the new ledger, don't have corresponding *Hash* and *prevHash* pairs. This is the ledger that peers have access to with the corresponding serial key.

Chapter 4

Implementation

This chapter contains an overview of the implementation detailed in chapter 3. It starts with brief look at the most relevant frameworks and tools employed, to put in context the jargon and the implementation decisions that were taken. Next, relevant developed work in the database, the back end and the front end will also be explained. A general navigation through the website with the several levels of content is also provided. The entirety of the code produced is available on `https://github.com/dfarinha?tab=repositories`.

4.1 Frameworks and tools

Several frameworks and tools were used for the development of this platform, the most notable being:

- **Node.js** Node.js is an open-source, asynchronous event-driven JavaScript runtime environment that executes JavaScript code outside of a browser, allowing the development of web servers and modules with multiple functionalities [46].

- **Angular** Angular 2+ is an open-source, TypeScript-based web framework that allows the creation of single-page applications (upon startup, data is sent to the browser as a JavaScript Object Notation (JSON) payload and the HTML transformation process is shifted from server to client side. Interactions are done with re-writings of the current page within the browser rather than requesting and loading from the server), developed by Google [47].

- **MongoDB** MongoDB is an open-source NoSQL document database that uses JSON documents with schema. It has a number of features like aggregation, replication, support of JavaScript queries, etc [48]. MongoDB Compass was used, which is a Graphical User Interface (GUI) for MongoDB.
- **Express** Express.js is an open-source, web framework for Node.js that provides robust tooling for HTTP servers [49].
- **Passport** Passport is authentication middleware for Node.js that authenticates requests. It provides an array of multiple authentication strategies [50].

A free to use User Interface (UI) kit by Creative-Tim as a template for the landing page was also used.

4.2 Data Models

Data modelling is always going to be a flexible process that needs to take into consideration the purpose of the application, the intrinsic characteristics of the data and what will its manipulation patterns be. For this application, five schemas were created with Mongoose, an object modelling tool for MongoDB:

- **Users**
  - The schema Users describes a peer. Apart from the usual identification fields like *email* or *location*, fields like *key* (which indicates key people within the company) or *affiliate* (which indicates affiliate companies) help to paint a clearer picture of the company's operations. Whenever a peer joins the platform, these should be filled. Along with these, an unique ID is also assigned automatically. The field *partners* is an array of documents that naturally stores other users. In order to avoid saving whole documents in the array, only the reference to the unique ID is stored as a reference. When needed, MongoDB's *Populate()* allows the possibility to fetch all the desired data related to the user with the ID in question.

    Passwords are never stored because, should the database be breached, every password would be leaked. A modern day catastrophe in most cases. Hashing it and storing its output value is common practice even though that also isn't entirely safe due to the deterministic nature of said functions, where two identical inputs result in the same hash. Multiple strategies (like the usage of rainbow tables) have surfaced to try to decode hashed passwords and it has been shown that common "weak"passwords can easily be compromised. A way to go around this is to add a *salt* which is random value that is appended to the input of hash functions, creating unique outputs. The result is that even the same two passwords will yield different hashes, making it look like a non-deterministic function.

The several fields as well as its types that comprise the user's schema are shown in table 4.1.

Table 4.1: The user's schema

Field	**Type**
email	String
name	String
partners	Schema.Types.ObjectId, ref: 'Users'
location	String
address	String
country	String
cellphone	String
postal	String
key	String
affiliate	String
hash	String
salt	String

The model Users also has several methods, most notably:

* **setPassword** generates the salt and the subsequent hash
* **validatePassword** checks whether a log in attempt is valid
* **generateJWT** generates a JSON Web Token (JWT) for the user upon a successful login, allowing it to access protected routes

- **Block**
  - The schema Block describes a single block. In subsection 3.4, the anatomy of a block has already been examined. It is represented again in table 4.2.

Table 4.2: The block's schema

Field	**Type**
timestamp	Date
issuer	String
newOwner	String
data	String
dataID	String
amount	Number
value	Number
prevHash	String
hash	String

Because filling these fields isn't straightforward, a class block was also created. Notably, the constructor receives all of the arguments (the proper block's creation date as well, courtesy of a timestamp service) and makes use of a JavaScript hashing component to hash the necessary fields. Only then, is a document created in the block model.

- **Chain**
  - The chain model is relatively simple since it is merely an ordered list of blocks. It could be an additional field of the model Users in the form of an array but to avoid clogging that model too much, which affects visualization while testing, a new one was created. Because it's just a collection of existing entries in the database, it only has two fields both of which reference other documents.

    The chain model is represented in table 4.3.

    Table 4.3: The chain's schema

Field	**Type**
owner	Schema.Types.ObjectId, ref: 'Users'
blocks	Schema.Types.ObjectId, ref: 'Block'

- **Inventory**
  - The inventory model represents a peer's inventory. It is comprised of two fields. One of them is a reference to a document of type User since each inventory belongs to a user and the other field is a products array where each object is defined by a name, an amount and a value. These are the values that are later compiled into a block.

    The inventory model is displayed in table 4.4.

    Table 4.4: The inventory's schema

Field	**Type**
owner	Schema.Types.ObjectId, ref: 'Users'
products	[{ name: String, amount: Number, value: Number}]

    An inventory class was also created that is responsible for assembling an object with the *name*, *amount* and *value* values.

- **Final**
  - The final model represents a final product, after a peer has assembled it with the relevant blocks. The *name* field contains the name or a description of the assembled good while the *manufacturer* is a reference to an existent User, the peer behind the production and selling of the product. The *serial* field is needed to attach an unique key to the product so the customer can look it

up. Finally, it also has an array of objects of the type Block which shows every block behind the product.

The final model is displayed in table 4.5.

Table 4.5: The final's schema

Field	Type
name	String
serial	String
manufacturer	Schema.Types.ObjectId, ref: 'Users'
blocks	[{ type: Schema.Types.ObjectId, ref: 'Block'}]

The serial is generated through a simple JavaScript library that generates a random number in a selected interval.

4.3 Protected routes

Routing describe the response of the application when a request is received, through a path and a specific Hypertext Transfer Protocol (HTTP) request method (GET, POST, HEAD, PUT, etc) to a specific endpoint.

Before detailing the most relevant endpoints developed, an overview of route protection is needed.

Naturally, most of the routes are not available to users that are not signed in. This is due to: 1) A user has its own peer area with relevant information displayed and 2) most requests require an identity to know who the the peer behind the request is.

To ensure this condition within the platform, a control is made both server side and client side:

- **Server side** - Whenever a user signs in, a JWT is created and associated with that peer. While the session is active, every request by that user will have the JWT attached in the payload. When a request is received, the token is decrypted and analyzed to check its validity. If it isn't, the server won't respond to the request. When a peer logs out, the JWT is destroyed.

- **Client side** - An authentication guard and an authentication service ensure this control in the front end. Whenever a peer successfully logs in, its JWT will be sent by the back end. The token is then retrieved and associated to the user in the authentication service. An authentication guard communicates with the said service whenever a request is received to access a protected route. It then analyzes the validity of the token and answers accordingly. Only then does the guard let the user's request to a protected route go through to the back end.

4.4 Relevant routes

Several endpoints were created server-side, the most relevant ones being explained here. The corresponding flowcharts are presented at the end of the bullet list.

- **/api/users/**
  - This endpoint handles registration requests. It creates an object user through the body of the request and it analyses if properties *user.email* and *user.password* are present since these are required to perform a registration (an additional form validation is performed on the front end as well). If any of these two fields isn't present, error 422 (Unprocessable Entity) is thrown and registration fails. If both were inputted, the user is created and a JWT is generated to allow navigation.

    Afterwards, a new inventory for that user is automatically created in which *inventory.owner = user.id* (the reference to the user is stored in the field *owner* of the inventory) and the array *products* is empty. Likewise, a new chain is also created, belonging to the same newly created user. A genesis block is then created with the timestamp service and the fields as seen in subsection 3.4. This new genesis block is a normal block and is saved normally. As seen in 4.2, block objects aren't saved in the array *blocks* of the chain, only its references. Therefore, the reference of the genesis block is then stored in the array *blocks*. Finally, the process is complete and the newly created user is returned as a JSON object.

    A diagram that illustrates the registration process in the platform is shown in figure 4.1.

- **/api/users/addblocktochain**
  - This endpoint is responsible for adding blocks to the corresponding chains. In other words, it contains all the logic behind a transaction. When one happens, a couple things need to be done. Both parties' identity must be known, the transaction data needs to be analyzed, two different blocks need to be generated (the *prevHash* field will be different) and subsequently added to both parties' chains and the seller's inventory needs to be updated, since the product is no longer in his possession.

    Transactions are actions that are activated only by the buyer. A peer is browsing its partners' inventories and clicks on a product to buy it, triggering the transaction. With this in mind and with the fact that it is a protected route, it is known who is making the purchase through the request's payload. Because a product is clicked, it is possible to send the transaction data and the seller identity in the request's body. Therefore, through the request, it is possible to know: 1) the identity of the buyer (*newOwner*) and 2) the identity of the seller (*issuer*) and the transaction data.

After both identities and transaction information is known, the only thing left to do is to take a logical step by step approach to make the necessary alterations. Almost all information needed to compile a block is known by know, expect what will the *prevHash* be. Through the buyer's ID, its chain is fetched. An object chain, as seen before, has the field *owner* and *blocks* which is an array of blocks. Since it is now fetched, getting the latest block's hash (which is the new block's *prevHash*) is done with *chain.blocks[chain.blocks.length - 1].hash*. After this is also known, the new block for the buyer's chain can now be generated and pushed into its chain.

An identical process is repeated for the seller. After its chain is dealt with, the inventory also needs updating. Its inventory is then fetched and the transaction product is pulled from the array *products*. The whole process is now complete and the now updated inventory is returned as JSON.

A diagram that illustrates this process is displayed in figure 4.2.

- **/api/users/showpartnerchains**

  - This endpoint returns all the peer's partners in detail as well as all of the partners' blocks. Because objects aren't stored, only its reference, MongoDB's *populate()* fills in the rest of the data, as mentioned before. This allows the complete object to be sent to the front end and thus showing it all of its fields since showing only references to the user would be no good. When the request is received, again through the payload it is known who the peer behind the request is. With that, a comparison between the content of its field *partners* with each chain owner returns every chain of every peer's partner. Now that all the relevant chains are known, its fields *blocks* and *owner* are populated to return the wanted object.

    An illustration of the process to show the peer's partners and its chains is displayed in figure 4.3.

- **/api/chains/:id**

  - The previous endpoint returns all of the partners' chains, which can be a bit too broad. In this endpoint on the other hand, only one chain and its blocks is returned. The ID of the desired chain is sent through the Uniform Resource Locator (URL). This is done, for example, when a peer is browsing a list of its partners and clicks in one of them, triggering the request.

    When the request is received, the ID of the chain is retrieved from the URL, the corresponding chain is fetched and its blocks are populated. Then, the chain is now returned as a JSON.

- **/api/invent/addpartner**
  - This endpoint allows the peer to add partners to its list. Through the request's payload, the identity of the peer is known. In order to add a partner, some sort of unique identifier that identifies another peer. In this implementation, the email field was used. With that input, the corresponding peer is fetched.

    If no peer exists with that identifier, error 400 is thrown (Bad Request). If it does exist, its reference is then stored in the *partners* field of the original peer, which is then returned as a JSON object. This is a one way process. Peer A adds peer B, but peer A doesn't get automatically added to peer B. This could easily be implemented but in some use cases it might not be the desired behaviour. So, with flexibility in mind, in order for both peers to be added in each other's *partners* field, two requests need to be made.

    An illustration of the process is depicted in figure 4.4.

- **/api/invent/addproduct**
  - This endpoint allows the peer to add a product to its inventory. The user's identity is once again known through the payload.

    The peer inputs the product's *name*, *value* and *amount* values. These fields constitute the request's body and are used to create a product, using the *inventory* class. With the product created, the peer's inventory needs updating. Through its identity, its inventory is fetched. Following this, the newly created product can be pushed into the *products* array of the peer's inventory. After this is completed, the updated inventory is returned as JSON.

    The handling of a add product request is shown in figure 4.5.

- **/api/invent/addfinal**
  - This endpoint allows the peer to assemble a final product. This is the product that is later sold to the customer. As explained before, this product is created by selecting all of the blocks that contain relevant transactions. Because several blocks need to be selected, this action is done in the area where a peer is inspecting its own main ledger. While visualizing every block in the front end, a peer clicks on the desired ones to select them. A name, or a description should also be written before assembling the product. When all of this is concluded, the request to add a product takes place. The payload is also analyzed once again because the identity needs to be known. Following this, a new final product is created in the *final* class which contains *name*, *manufacturer* and *blocks* along with field *serial* that is generated in the process. This product is then stored and returned as a JSON.

    An illustration of this process is displayed in figure 4.6.

- **/api/invent/getfinal/:serial**

  – This endpoint handles requests by costumers. Because customers don't get registered, the route isn't protected.

    The costumer inputs the serial key given to him in the moment of purchase in the front page of the platform. The serial is then passed in the URL and the corresponding product is fetched. The *blocks* and *manufacturer* fields get populated and that object is sent as a JSON.

    An illustration of this sequence is displayed in figure 4.7.

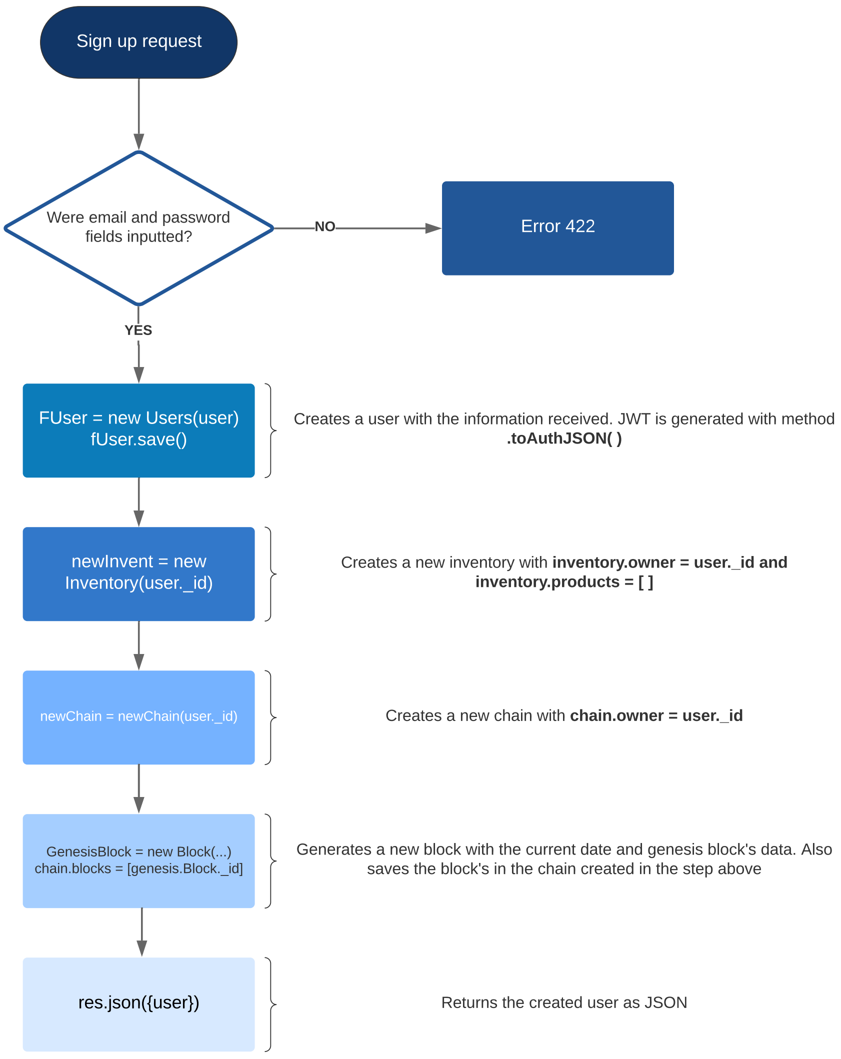

Sign up request
Were email and password fields inputted?
NO
Error 422
YES
FUser = new Users(user) fUser.save()
Creates a user with the information received. JWT is generated with method **.toAuthJSON( )**
newInvent = new Inventory(user._id)
Creates a new inventory with **inventory.owner = user._id and inventory.products = [ ]**
newChain = newChain(user._id)
Creates a new chain with **chain.owner = user._id**
GenesisBlock = new Block(...) chain.blocks = [genesis.Block._id]
Generates a new block with the current date and genesis block's data. Also saves the block's in the chain created in the step above
res.json({user})
Returns the created user as JSON

Figure 4.1: The process of registering a user

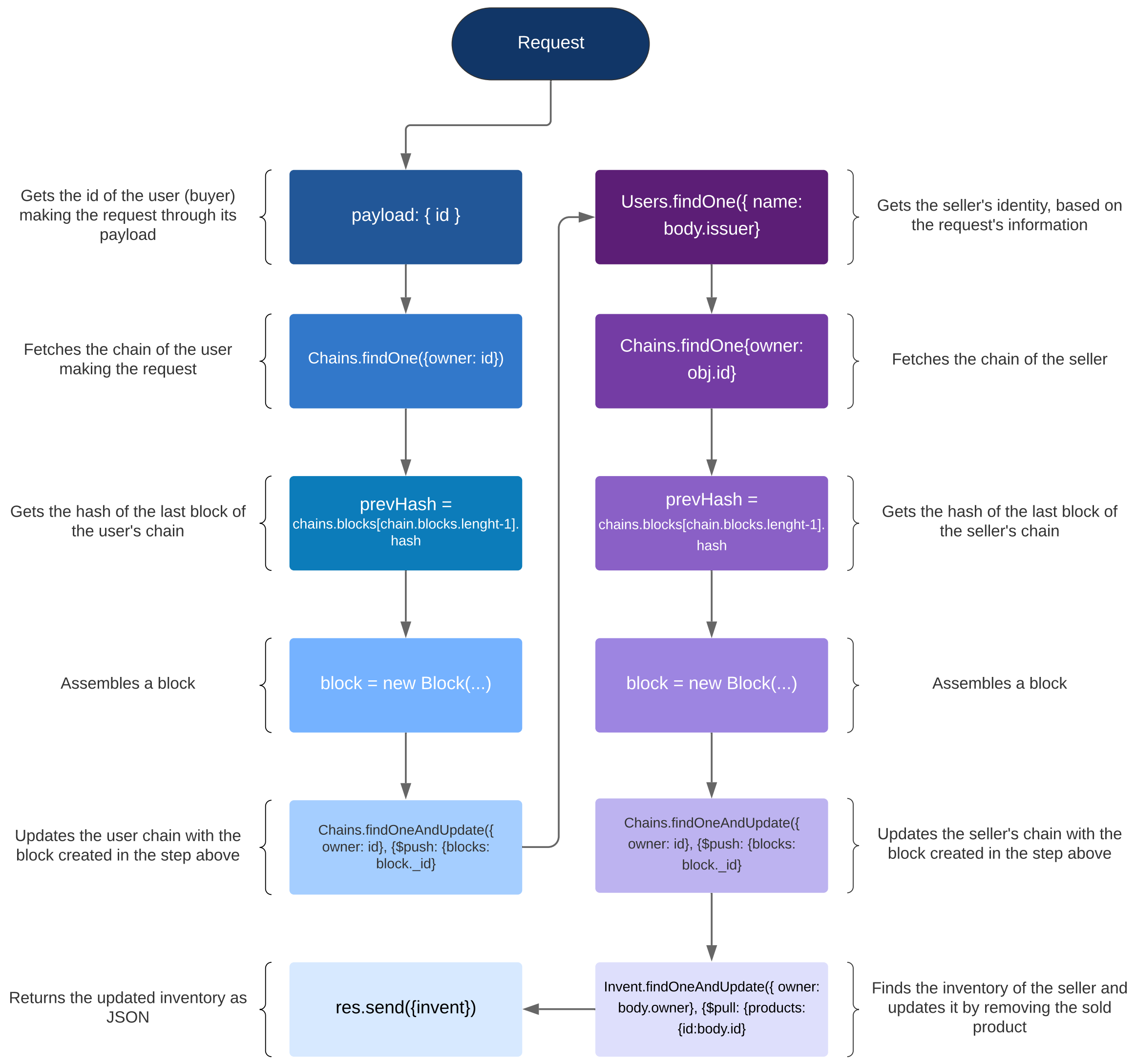

Request
Gets the id of the user (buyer) making the request through its payload
payload: { id }
Users.findOne({ name: body.issuer}
Gets the seller's identity, based on the request's information
Fetches the chain of the user making the request
Chains.findOne({owner: id})
Chains.findOne{owner: obj.id}
Fetches the chain of the seller
Gets the hash of the last block of the user's chain
prevHash = chains.blocks[chain.blocks.lenght-1].hash
prevHash = chains.blocks[chain.blocks.lenght-1].hash
Gets the hash of the last block of the seller's chain
Assembles a block
block = new Block(...)
block = new Block(...)
Assembles a block
Updates the user chain with the block created in the step above
Chains.findOneAndUpdate({ owner: id}, {$push: {blocks: block._id}
Chains.findOneAndUpdate({ owner: id}, {$push: {blocks: block._id}
Updates the seller's chain with the block created in the step above
Returns the updated inventory as JSON
res.send({invent})
Invent.findOneAndUpdate({ owner: body.owner}, {$pull: {products: {id:body.id}
Finds the inventory of the seller and updates it by removing the sold product

Figure 4.2: The process behind a transaction

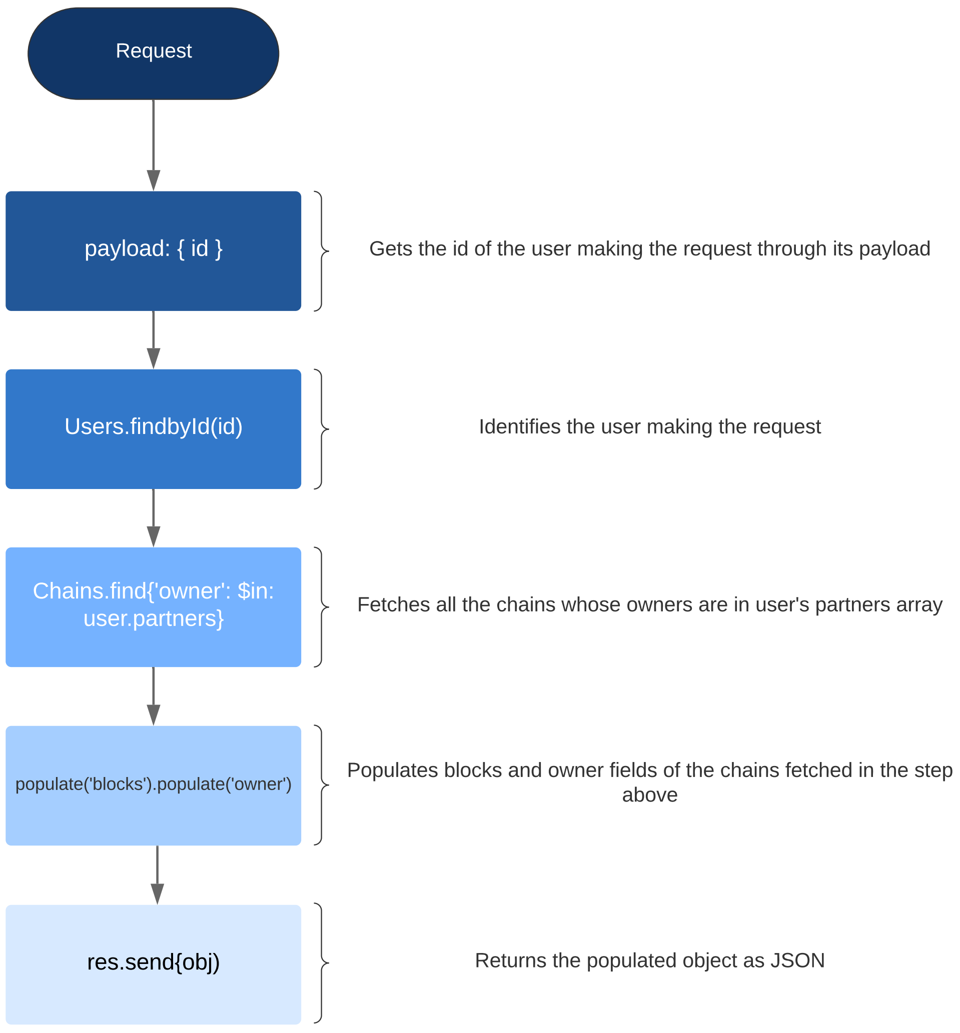

Request
payload: { id }
Gets the id of the user making the request through its payload
Users.findbyId(id)
Identifies the user making the request
Chains.find{'owner': $in: user.partners}
Fetches all the chains whose owners are in user's partners array
populate('blocks').populate('owner')
Populates blocks and owner fields of the chains fetched in the step above
res.send{obj)
Returns the populated object as JSON

Figure 4.3: The process behind showing the peer's partners and its chains

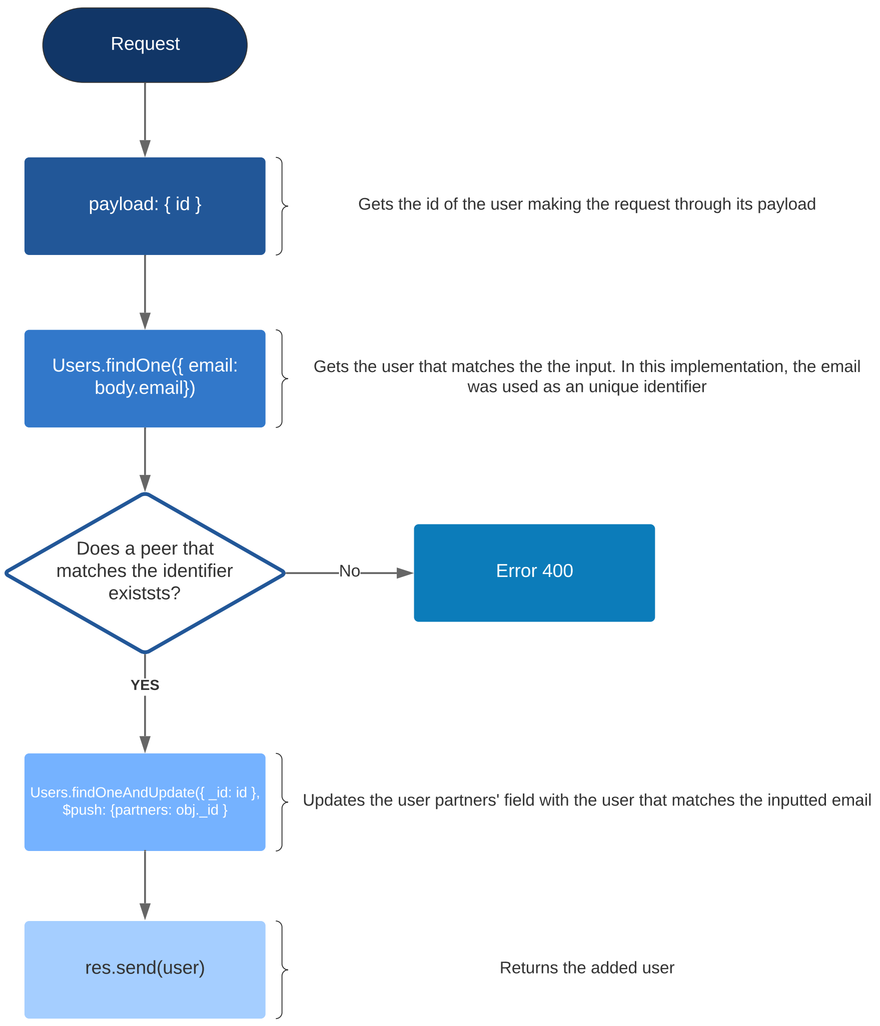

Request
payload: { id }
Gets the id of the user making the request through its payload
Users.findOne({ email: body.email})
Gets the user that matches the the input. In this implementation, the email was used as an unique identifier
Does a peer that matches the identifier existsts?
No
Error 400
YES
Users.findOneAndUpdate({ _id: id }, $push: {partners: obj._id }
Updates the user partners' field with the user that matches the inputted email
res.send(user)
Returns the added user

Figure 4.4: The process behind adding a partner

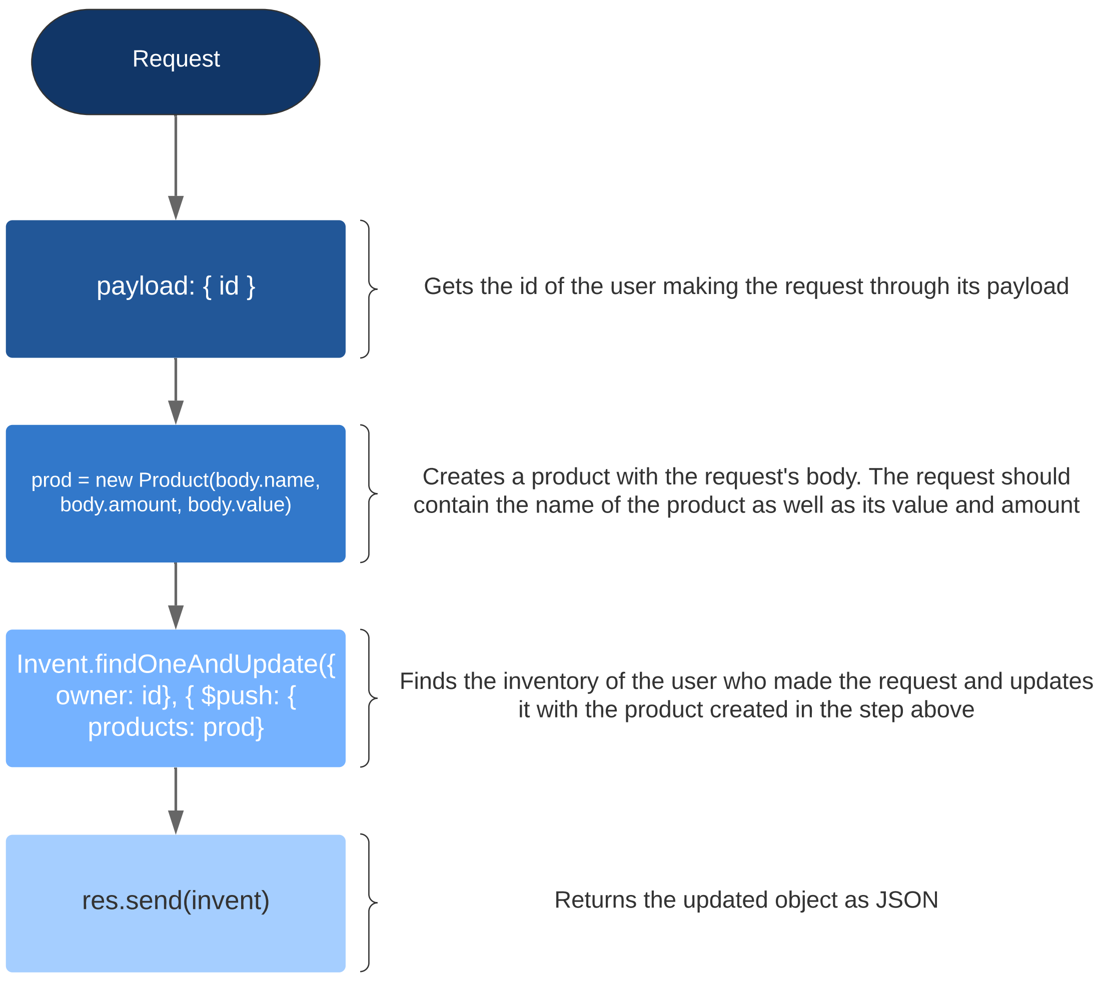

Request
payload: { id }
Gets the id of the user making the request through its payload
prod = new Product(body.name, body.amount, body.value)
Creates a product with the request's body. The request should contain the name of the product as well as its value and amount
Invent.findOneAndUpdate({ owner: id}, { $push: { products: prod}
Finds the inventory of the user who made the request and updates it with the product created in the step above
res.send(invent)
Returns the updated object as JSON

Figure 4.5: The process behind adding a product

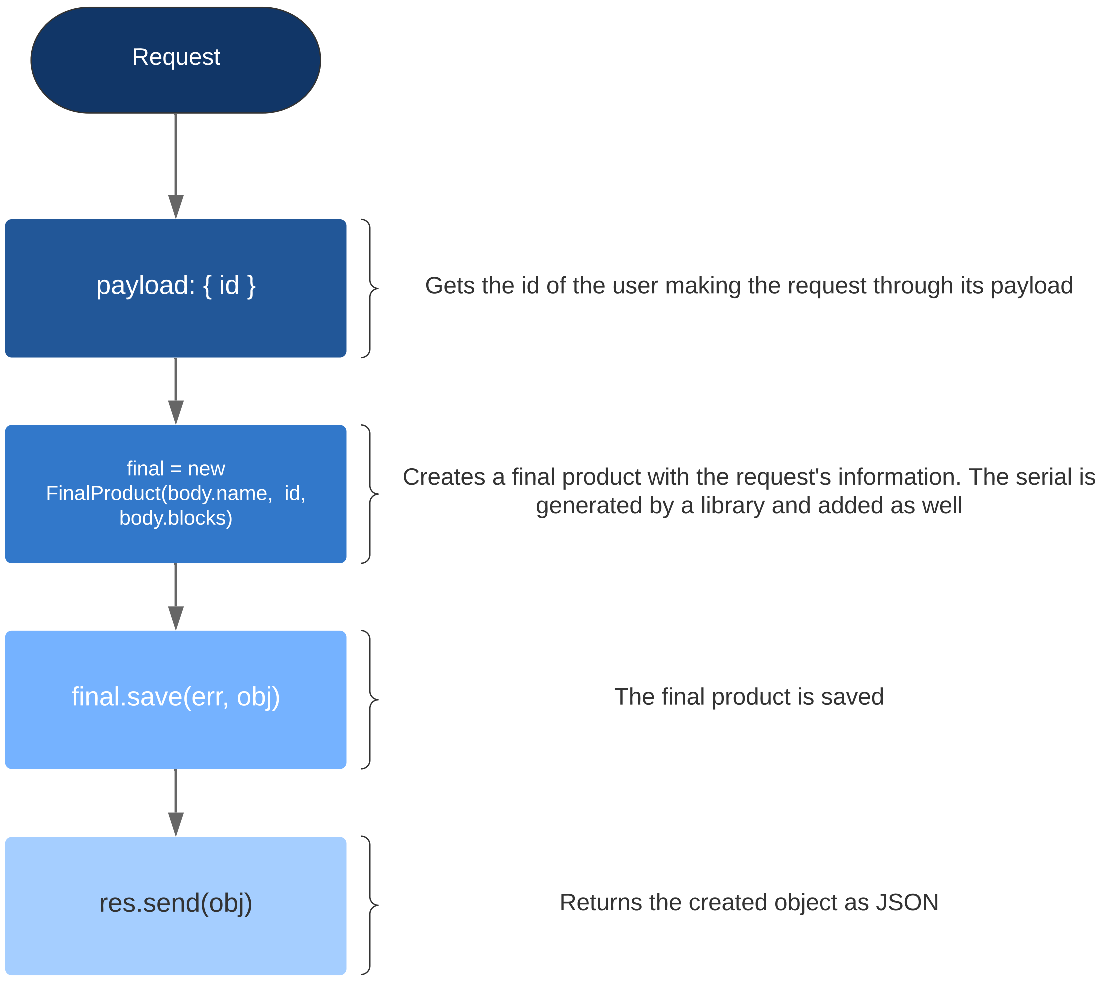

Request
payload: { id }
Gets the id of the user making the request through its payload
final = new FinalProduct(body.name, id, body.blocks)
Creates a final product with the request's information. The serial is generated by a library and added as well
final.save(err, obj)
The final product is saved
res.send(obj)
Returns the created object as JSON

Figure 4.6: The process behind creating a final product

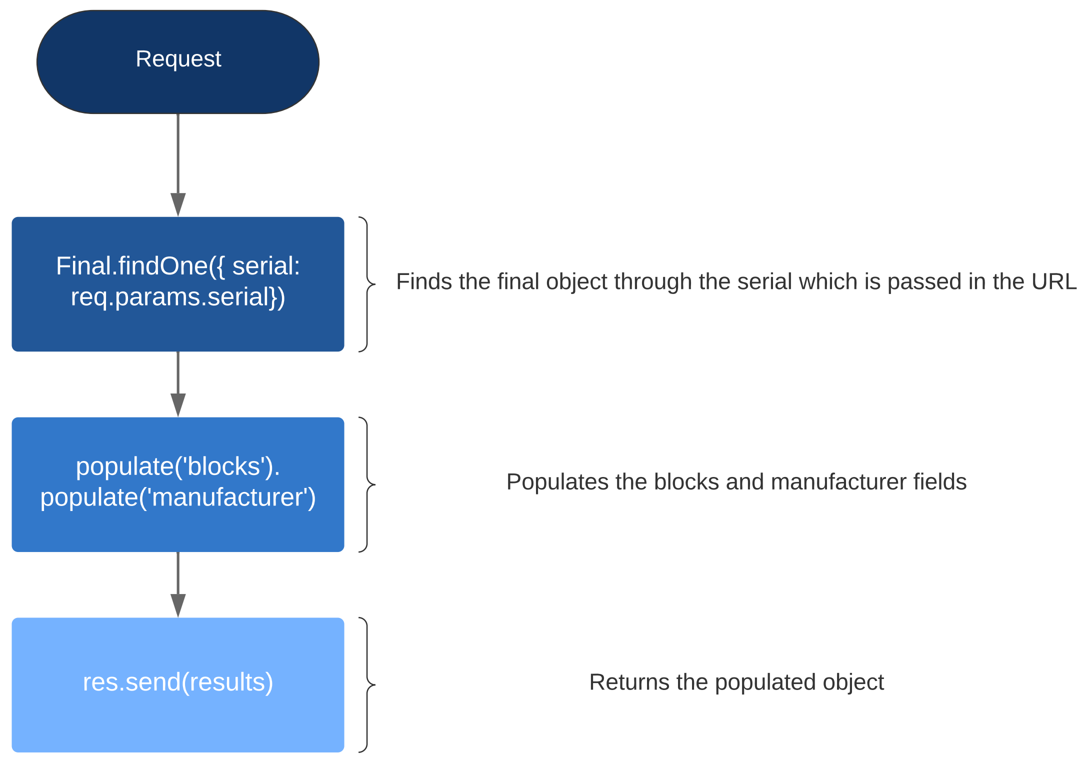

Request
Final.findOne({ serial: req.params.serial})
Finds the final object through the serial which is passed in the URL
populate('blocks').populate('manufacturer')
Populates the blocks and manufacturer fields
res.send(results)
Returns the populated object

Figure 4.7: The process behind returning a final product

4.5 General navigation

The top level navigation (or first level content) corresponds to the front page in which a brief description of the product is shown as well as the team behind the project (advisor, co-adviser and advisee). It's through the front page that a customer can make use of the serial key provided in a purchase. This leads to a deeper navigation level where the customer can inspect the relevant blocks. Also in the front page, authentication/registration is available to the peers. If valid, navigation is redirected to the peer's area. The authenticated peer is now referred simply as "peer". The peer area is comprised of six categories:

- **Partners' chains** - A list with the peer's partners. Each can be individually clicked, leading to the inspection of every single past transaction, in the form of blocks, related to that partner
- **Your own chain** - The category in which the peer is able to inspect the blocks that comprise its own chain (the same data that is displayed when the peer's partners inspect it). Through this category, the peer can also assemble a product to be sold

by selecting the relevant transactions. Through this operation, an unique serial key is generated which should be passed along to the customer

- **Partners' inventories** - The list of products currently being sold by the peer's partners, allowing purchases to be made which are added in the relevant ledgers
- **Manage inventory** - Where the peer can add and delete products which are then displayed to its partners
- **Partners' profiles** - A list with the peer's partners. Upon further clicking, additional information is displayed about the clicked partner such as contact information, key people within the organization, affiliate companies. The peer can also add a new partner to its network in this category, expanding business opportunities
- **Profile** - Where the peer can update its own profile with relevant information. This information is later displayed to the peer's partners

An illustration of the platform's levels of content is displayed in figure 4.8.

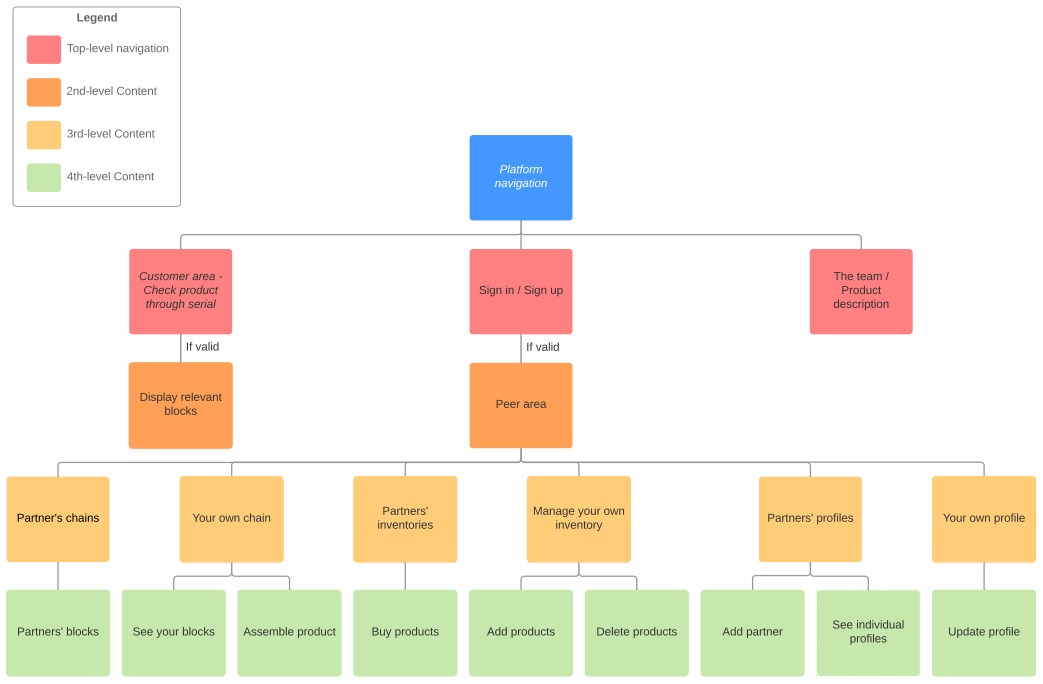


Figure 4.8: Navigation through the platform

Chapter 5

Tests and validation

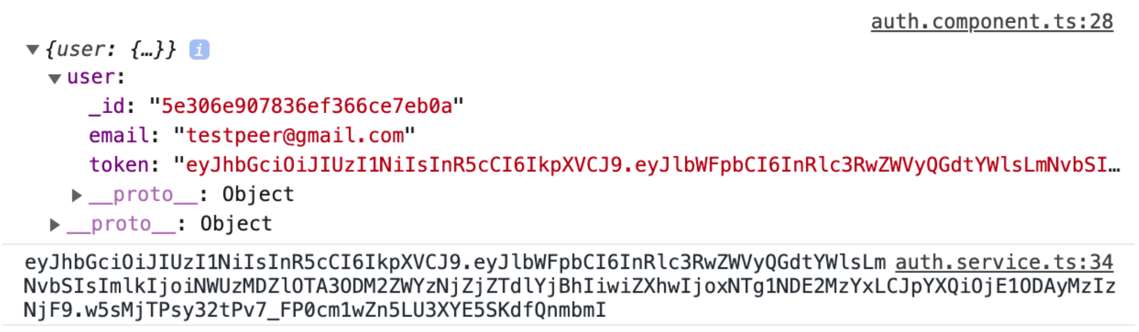
In the previous chapter, the most relevant routes were presented. As it was seen, every route has a response that involves the server returning an object. The tests are done by analyzing the several returned JSON objects sent by the server in order to check if the desired updates took place. Images in this section are taken from Google Chrome devTools' console upon valid requests to the server. Afterwards, screenshots taken in the browser, showcasing the frontend work are presented. Some additional screenshots of the frontend are presented in Annex I.

If the endpoints work correctly then it means the platform is working as intended and creating the front end becomes only a design challenge.

To test the platform, a tour with the user (*testPeer*), a coffee manufacturer, is taken as it explores the capabilities of the platform.

As the newly created *testPeer* inputs its crediantials, the sign in is successful and a valid JWT is generated to allow navigation. The token, while valid, is associated with its unique ID, allowing the server to authenticate requests.

The server response upon signing in is shown in figure 5.1.



```
                                                              auth.component.ts:28
▼{user: {…}} i
  ▼user:
     _id: "5e306e907836ef366ce7eb0a"
     email: "testpeer@gmail.com"
     token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InRlc3RwZWVyQGdtYWlsLmNvbSI…
   ▶__proto__: Object
  ▶__proto__: Object
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InRlc3RwZWVyQGdtYWlsLm auth.service.ts:34
NvbSIsImlkIjoiNWUzMDZlOTA3ODM2ZWYzNjZjZTdlYjBhIiwiZXhwIjoxNTg1NDE2MzYxLCJpYXQiOjE1ODAyMzIz
NjF9.w5sMjTPsy32tPv7_FP0cm1wZn5LU3XYE5SKdfQnmbmI
```

Figure 5.1: The server response upon sign in

As *testPeer* performs a request to inspect its chain, the response is shown in figure 5.2.

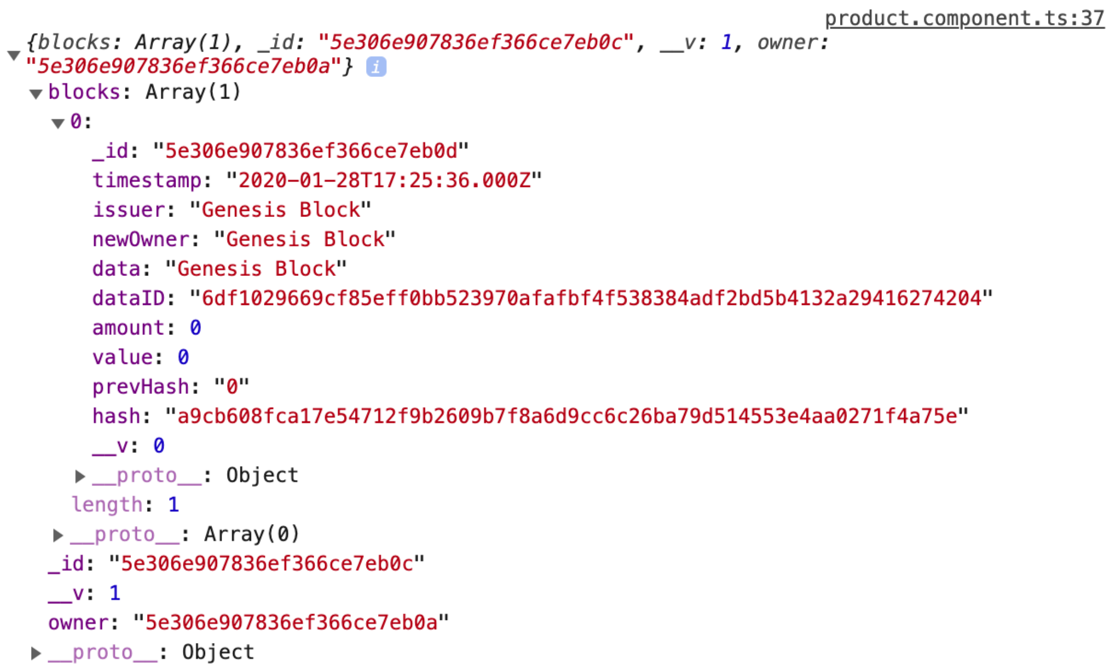

```
                                                                   product.component.ts:37
▼{blocks: Array(1), _id: "5e306e907836ef366ce7eb0c", __v: 1, owner:
 "5e306e907836ef366ce7eb0a"} i
  ▼blocks: Array(1)
    ▼0:
       _id: "5e306e907836ef366ce7eb0d"
       timestamp: "2020-01-28T17:25:36.000Z"
       issuer: "Genesis Block"
       newOwner: "Genesis Block"
       data: "Genesis Block"
       dataID: "6df1029669cf85eff0bb523970afafbf4f538384adf2bd5b4132a29416274204"
       amount: 0
       value: 0
       prevHash: "0"
       hash: "a9cb608fca17e54712f9b2609b7f8a6d9cc6c26ba79d514553e4aa0271f4a75e"
       __v: 0
      ▶__proto__: Object
     length: 1
    ▶__proto__: Array(0)
   _id: "5e306e907836ef366ce7eb0c"
   __v: 1
   owner: "5e306e907836ef366ce7eb0a"
  ▶__proto__: Object
```

Figure 5.2: Inspecting the newly created chain

It's noticeable that the chain's *owner* has the *testPeer*'s ID and it only contains one block, the Genesis Block.

Normally, the chain only holds the blocks' references. In this case, *populate()* was used to fill the multiple block's fields.

Next, *testPeer* adds *Peer X* and *Peer Y* to its partners list.

The response of the request by *testPeer* to inspect its partners list is shown in figure 5.3.

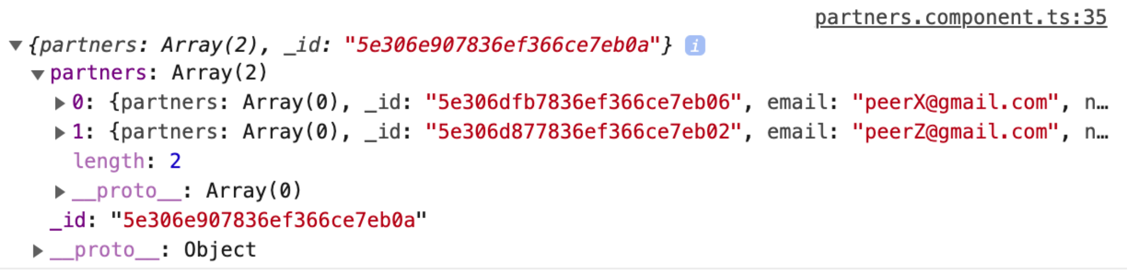
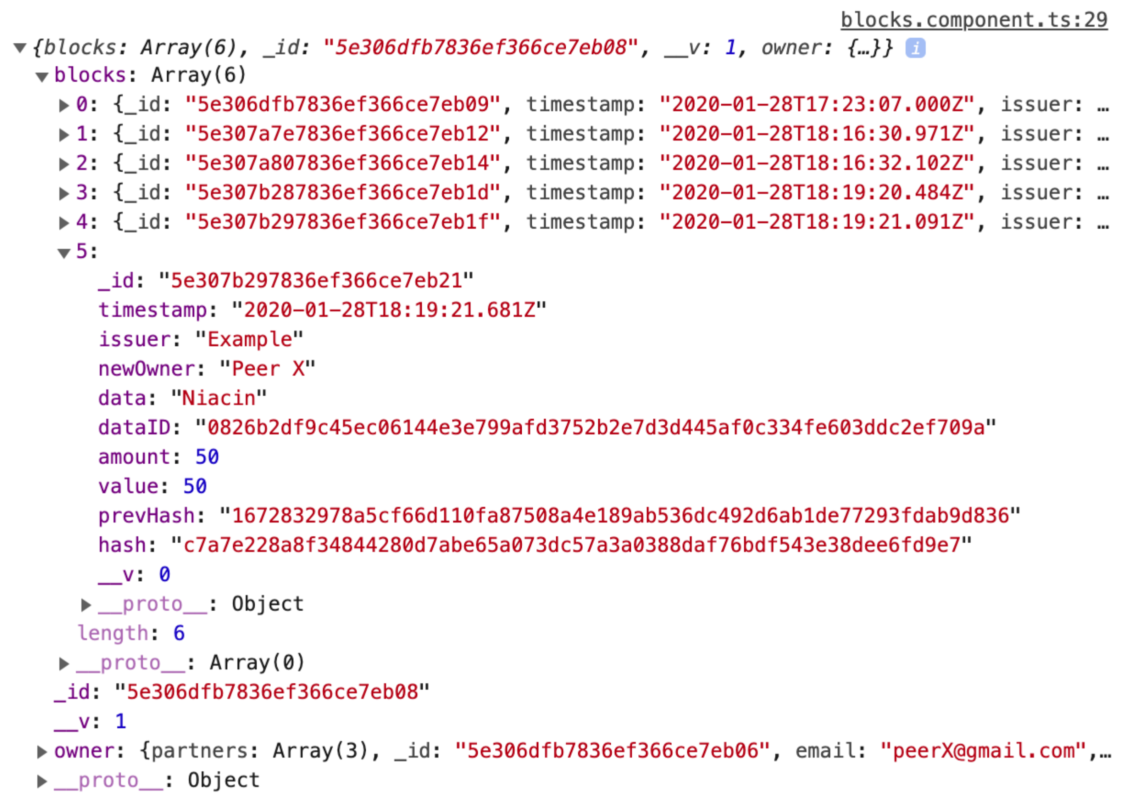

```
                                                                 partners.component.ts:35
▼{partners: Array(2), _id: "5e306e907836ef366ce7eb0a"} i
  ▼partners: Array(2)
    ▶0: {partners: Array(0), _id: "5e306dfb7836ef366ce7eb06", email: "peerX@gmail.com", n…
    ▶1: {partners: Array(0), _id: "5e306d877836ef366ce7eb02", email: "peerZ@gmail.com", n…
     length: 2
    ▶__proto__: Array(0)
   _id: "5e306e907836ef366ce7eb0a"
  ▶__proto__: Object
```

Figure 5.3: Inspecting the partners list

testPeer can inspect its partners' chains to get more information about previous transactions. Upon inspection of *Peer X*'s chain, the response received is shown in figure 5.4.

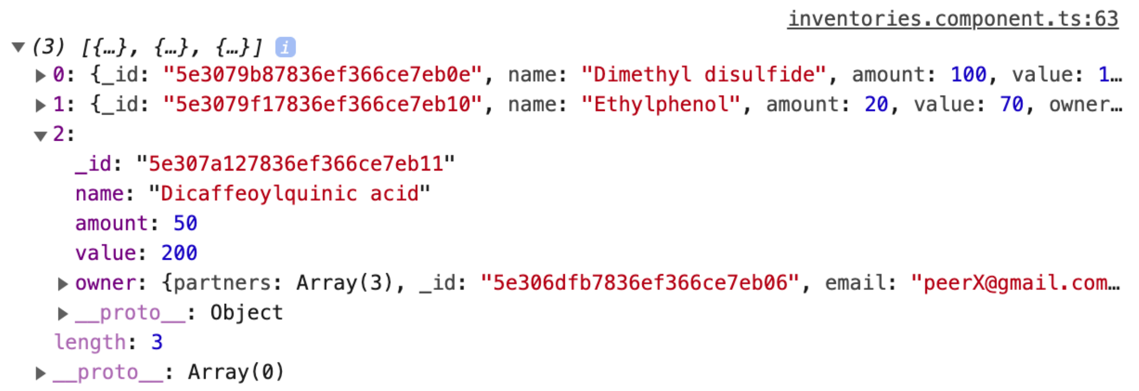


```
                                                                  blocks.component.ts:29
▼{blocks: Array(6), _id: "5e306dfb7836ef366ce7eb08", __v: 1, owner: {…}} ⓘ
  ▼blocks: Array(6)
    ▶0: {_id: "5e306dfb7836ef366ce7eb09", timestamp: "2020-01-28T17:23:07.000Z", issuer: …
    ▶1: {_id: "5e307a7e7836ef366ce7eb12", timestamp: "2020-01-28T18:16:30.971Z", issuer: …
    ▶2: {_id: "5e307a807836ef366ce7eb14", timestamp: "2020-01-28T18:16:32.102Z", issuer: …
    ▶3: {_id: "5e307b287836ef366ce7eb1d", timestamp: "2020-01-28T18:19:20.484Z", issuer: …
    ▶4: {_id: "5e307b297836ef366ce7eb1f", timestamp: "2020-01-28T18:19:21.091Z", issuer: …
    ▼5:
        _id: "5e307b297836ef366ce7eb21"
        timestamp: "2020-01-28T18:19:21.681Z"
        issuer: "Example"
        newOwner: "Peer X"
        data: "Niacin"
        dataID: "0826b2df9c45ec06144e3e799afd3752b2e7d3d445af0c334fe603ddc2ef709a"
        amount: 50
        value: 50
        prevHash: "1672832978a5cf66d110fa87508a4e189ab536dc492d6ab1de77293fdab9d836"
        hash: "c7a7e228a8f34844280d7abe65a073dc57a3a0388daf76bdf543e38dee6fd9e7"
        __v: 0
      ▶__proto__: Object
      length: 6
    ▶__proto__: Array(0)
    _id: "5e306dfb7836ef366ce7eb08"
    __v: 1
  ▶owner: {partners: Array(3), _id: "5e306dfb7836ef366ce7eb06", email: "peerX@gmail.com",…
  ▶__proto__: Object
```

Figure 5.4: Inspecting Peer X's chain. 5th block is expanded for visualization purposes

In this case, *peer X* has five transactions registered in the platform (the first block is the Genesis Block) which are can be inspected by its partners.

Inspecting its partners' inventories, allows *testPeer* to buy products. The received response is shown in figure 5.5.

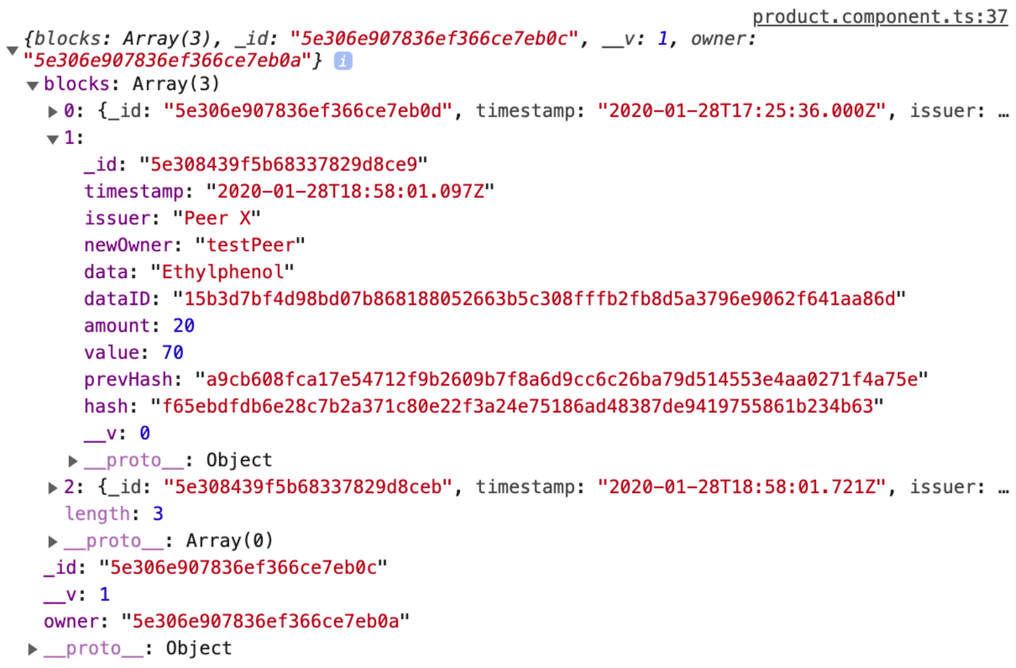


```
                                                           inventories.component.ts:63
▼(3) [{…}, {…}, {…}] ⓘ
  ▶0: {_id: "5e3079b87836ef366ce7eb0e", name: "Dimethyl disulfide", amount: 100, value: 1…
  ▶1: {_id: "5e3079f17836ef366ce7eb10", name: "Ethylphenol", amount: 20, value: 70, owner…
  ▼2:
      _id: "5e307a127836ef366ce7eb11"
      name: "Dicaffeoylquinic acid"
      amount: 50
      value: 200
    ▶owner: {partners: Array(3), _id: "5e306dfb7836ef366ce7eb06", email: "peerX@gmail.com…
    ▶__proto__: Object
    length: 3
  ▶__proto__: Array(0)
```

Figure 5.5: Inspecting partner's inventories. 3rd item is expanded for visualization purposes

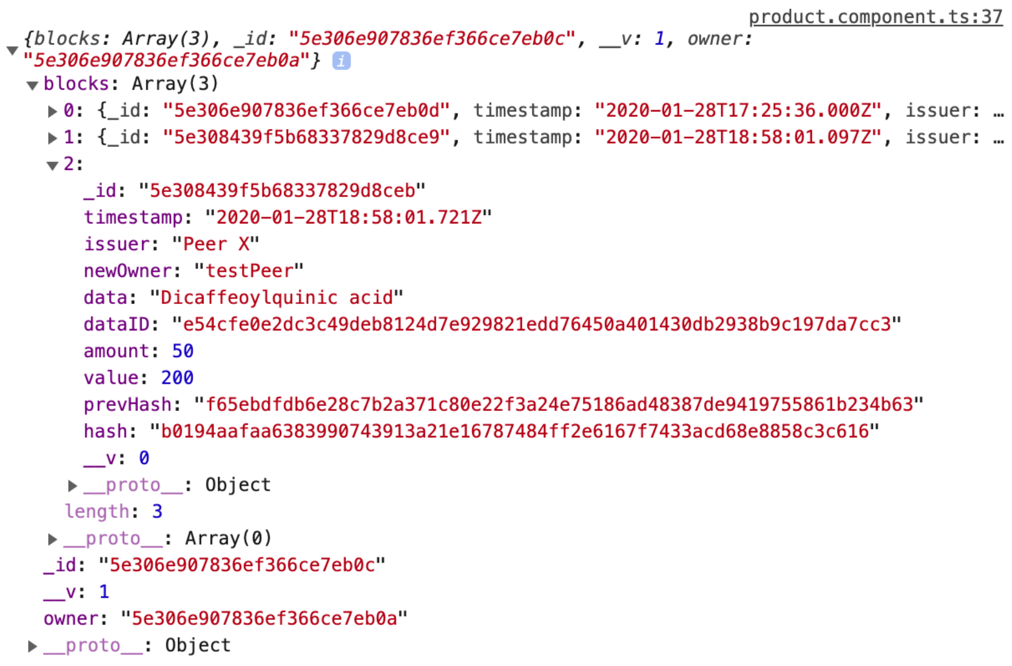
testPeer can currently buy three products, all belonging to *peer X*. In this case, two products get bought, Ethylphenol and Dicaffeoylquinic acid which are two chemical compounds present in coffee.

Upon a new request to inspect its own chain, *testPeer* gets the response by the server shown in figure 5.6 and 5.7.



```
                                                                  product.component.ts:37
▼ {blocks: Array(3), _id: "5e306e907836ef366ce7eb0c", __v: 1, owner:
  "5e306e907836ef366ce7eb0a"}
  ▼ blocks: Array(3)
    ▶ 0: {_id: "5e306e907836ef366ce7eb0d", timestamp: "2020-01-28T17:25:36.000Z", issuer: …
    ▼ 1:
        _id: "5e308439f5b68337829d8ce9"
        timestamp: "2020-01-28T18:58:01.097Z"
        issuer: "Peer X"
        newOwner: "testPeer"
        data: "Ethylphenol"
        dataID: "15b3d7bf4d98bd07b868188052663b5c308fffb2fb8d5a3796e9062f641aa86d"
        amount: 20
        value: 70
        prevHash: "a9cb608fca17e54712f9b2609b7f8a6d9cc6c26ba79d514553e4aa0271f4a75e"
        hash: "f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63"
        __v: 0
      ▶ __proto__: Object
    ▶ 2: {_id: "5e308439f5b68337829d8ceb", timestamp: "2020-01-28T18:58:01.721Z", issuer: …
      length: 3
    ▶ __proto__: Array(0)
    _id: "5e306e907836ef366ce7eb0c"
    __v: 1
    owner: "5e306e907836ef366ce7eb0a"
  ▶ __proto__: Object
```

Figure 5.6: The updated testPeer's chain. 2nd block is expanded for visualization purposes

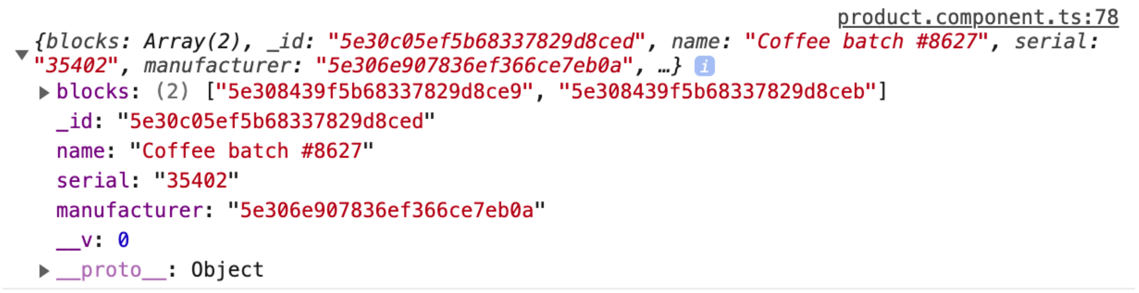


```
                                                                  product.component.ts:37
▼ {blocks: Array(3), _id: "5e306e907836ef366ce7eb0c", __v: 1, owner:
  "5e306e907836ef366ce7eb0a"}
  ▼ blocks: Array(3)
    ▶ 0: {_id: "5e306e907836ef366ce7eb0d", timestamp: "2020-01-28T17:25:36.000Z", issuer: …
    ▶ 1: {_id: "5e308439f5b68337829d8ce9", timestamp: "2020-01-28T18:58:01.097Z", issuer: …
    ▼ 2:
        _id: "5e308439f5b68337829d8ceb"
        timestamp: "2020-01-28T18:58:01.721Z"
        issuer: "Peer X"
        newOwner: "testPeer"
        data: "Dicaffeoylquinic acid"
        dataID: "e54cfe0e2dc3c49deb8124d7e929821edd76450a401430db2938b9c197da7cc3"
        amount: 50
        value: 200
        prevHash: "f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63"
        hash: "b0194aafaa6383990743913a21e16787484ff2e6167f7433acd68e8858c3c616"
        __v: 0
      ▶ __proto__: Object
      length: 3
    ▶ __proto__: Array(0)
    _id: "5e306e907836ef366ce7eb0c"
    __v: 1
    owner: "5e306e907836ef366ce7eb0a"
  ▶ __proto__: Object
```

Figure 5.7: The updated testPeer's chain. 3rd block is expanded for visualization purposes

The first block of the chain is the Genesis Block, as exhibited before. The following two are the purchases *testPeer* just did. The blocks are being appended correctly, as the sequence of hashes show.

testPeer now produces a batch of coffee with these newly acquired goods to sell to consumers. The response received after selecting these two new blocks and producing the batch of coffee is shown in figure 5.8.

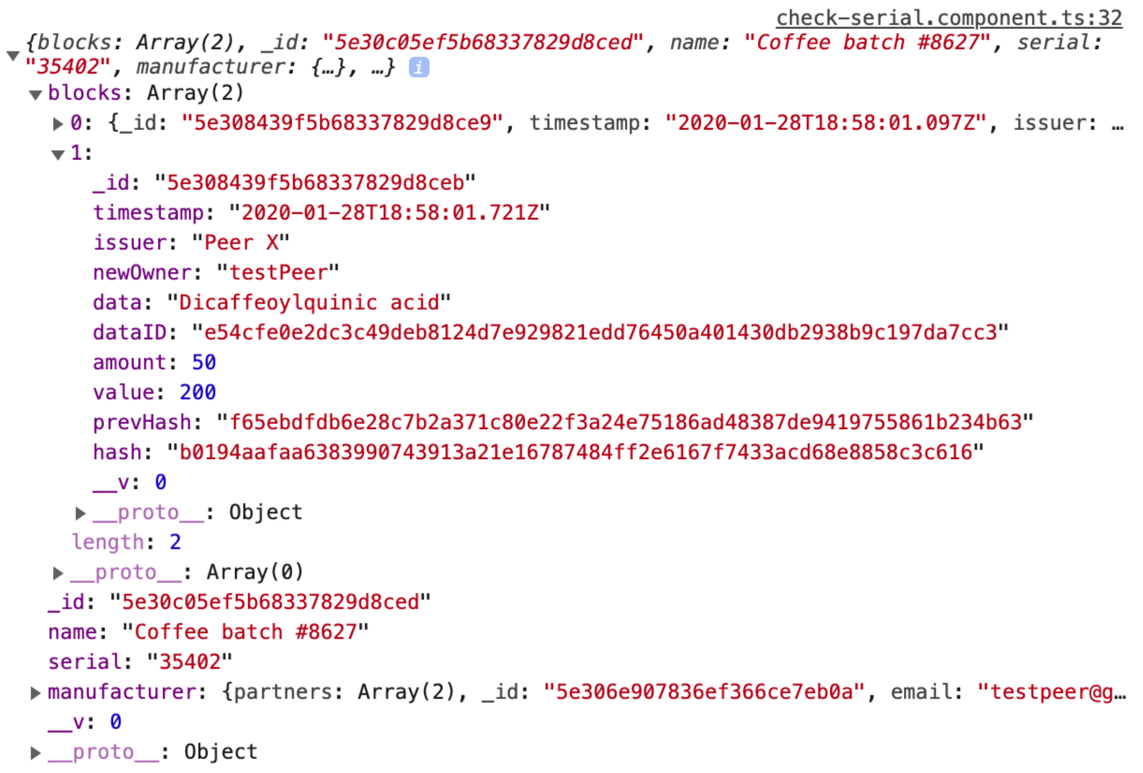


```
                                                      product.component.ts:78
▼{blocks: Array(2), _id: "5e30c05ef5b68337829d8ced", name: "Coffee batch #8627", serial:
 "35402", manufacturer: "5e306e907836ef366ce7eb0a", …} i
  ▶blocks: (2) ["5e308439f5b68337829d8ce9", "5e308439f5b68337829d8ceb"]
   _id: "5e30c05ef5b68337829d8ced"
   name: "Coffee batch #8627"
   serial: "35402"
   manufacturer: "5e306e907836ef366ce7eb0a"
   __v: 0
  ▶__proto__: Object
```

Figure 5.8: The newly created product

A serial key was generated for this new batch as the figure above shows.

Finally, as a user acquires the product, it can make use of the platform by inputting the serial key associated with the purchase. The corresponding request with the serial *"35402"* yields the response shown in figure 5.9.

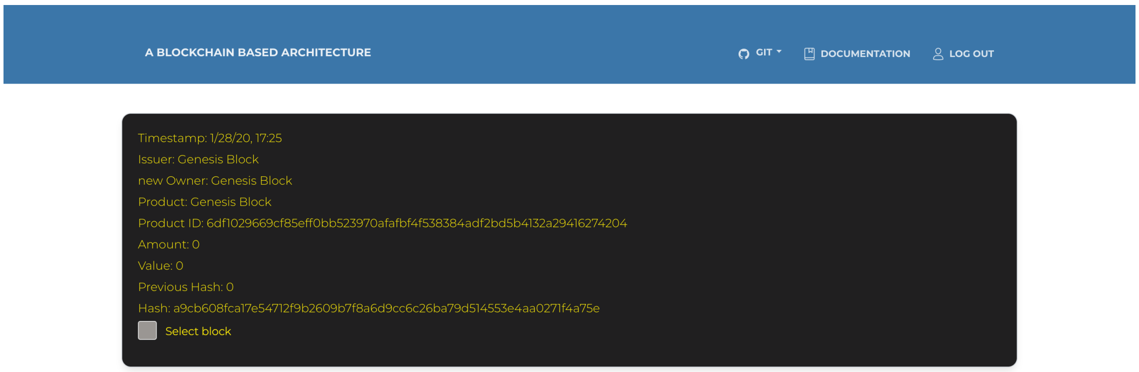


```
                                                   check-serial.component.ts:32
▼{blocks: Array(2), _id: "5e30c05ef5b68337829d8ced", name: "Coffee batch #8627", serial:
 "35402", manufacturer: {…}, …} i
  ▼blocks: Array(2)
    ▶0: {_id: "5e308439f5b68337829d8ce9", timestamp: "2020-01-28T18:58:01.097Z", issuer: …
    ▼1:
       _id: "5e308439f5b68337829d8ceb"
       timestamp: "2020-01-28T18:58:01.721Z"
       issuer: "Peer X"
       newOwner: "testPeer"
       data: "Dicaffeoylquinic acid"
       dataID: "e54cfe0e2dc3c49deb8124d7e929821edd76450a401430db2938b9c197da7cc3"
       amount: 50
       value: 200
       prevHash: "f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63"
       hash: "b0194aafaa6383990743913a21e16787484ff2e6167f7433acd68e8858c3c616"
       __v: 0
      ▶__proto__: Object
     length: 2
    ▶__proto__: Array(0)
   _id: "5e30c05ef5b68337829d8ced"
   name: "Coffee batch #8627"
   serial: "35402"
  ▶manufacturer: {partners: Array(2), _id: "5e306e907836ef366ce7eb0a", email: "testpeer@g…
   __v: 0
  ▶__proto__: Object
```

Figure 5.9: The inspection by a customer. 2nd block is expanded for visualization purposes

Now that it was shown the server works and the responses are what is desired, the frontend becomes just a design challenge. The following images are screenshots taken directly from the browser and present a graphical visualization of what was shown through Google Chrome devTools' console.

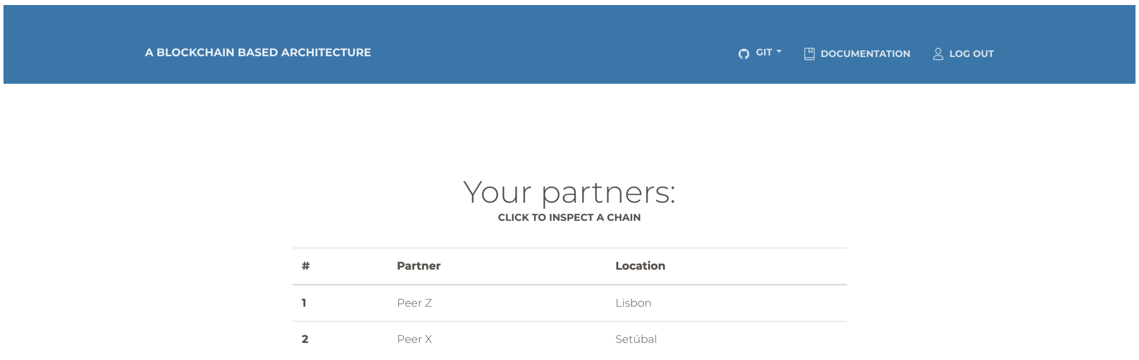
testPeer's inspection of its own newly created chain is shown in figure 5.10.



```
Timestamp: 1/28/20, 17:25
Issuer: Genesis Block
new Owner: Genesis Block
Product: Genesis Block
Product ID: 6df1029669cf85eff0bb523970afafbf4f538384adf2bd5b4132a29416274204
Amount: 0
Value: 0
Previous Hash: 0
Hash: a9cb608fca17e54712f9b2609b7f8a6d9cc6c26ba79d514553e4aa0271f4a75e
```

Figure 5.10: Inspecting the new chain in the frontend

The inspection of its partners list after adding two new members is shown in figure 5.11.

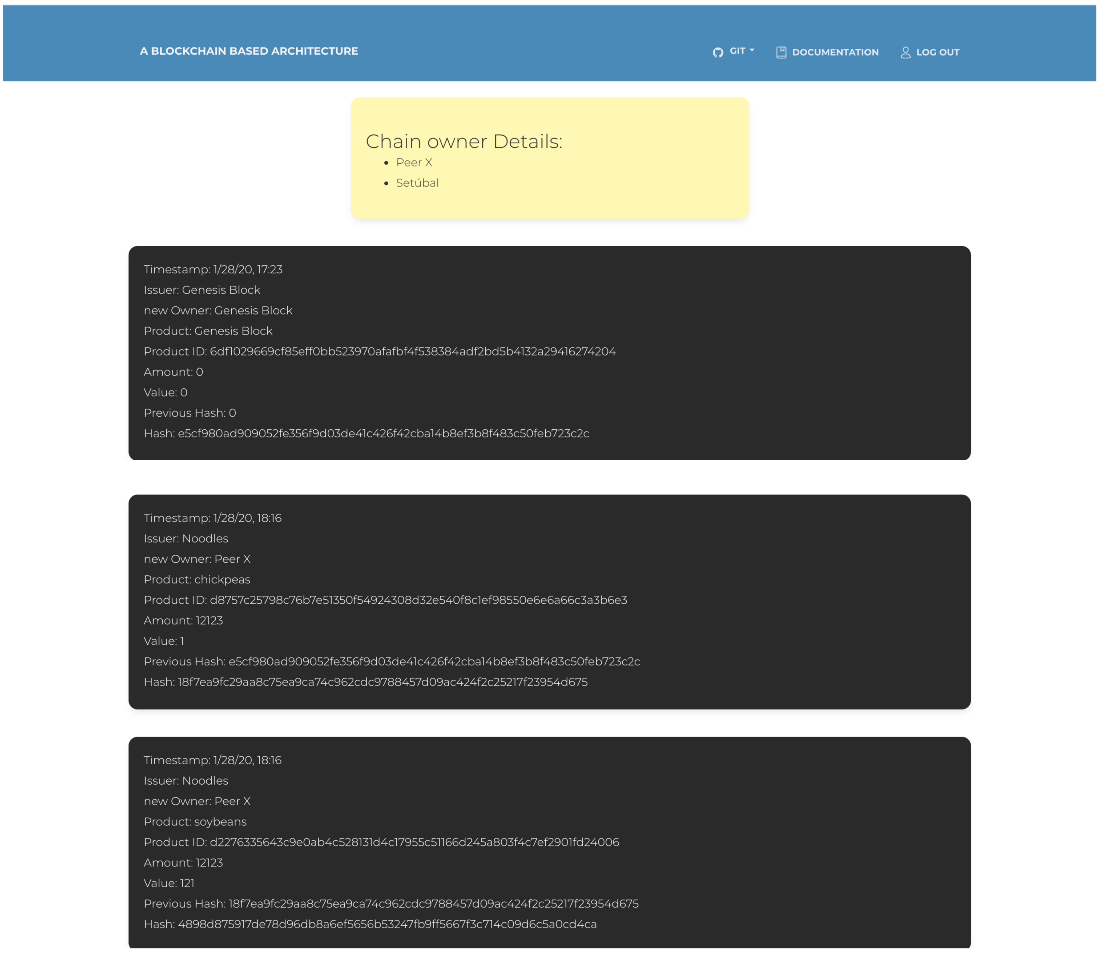


Your partners:
CLICK TO INSPECT A CHAIN

#	Partner	Location
1	Peer Z	Lisbon
2	Peer X	Setúbal

Figure 5.11: Inspecting the partners' list in the frontend

testPeer inspection of *peer X*'s chain is shown in figure 5.12.



Chain owner Details:

- Peer X
- Setúbal

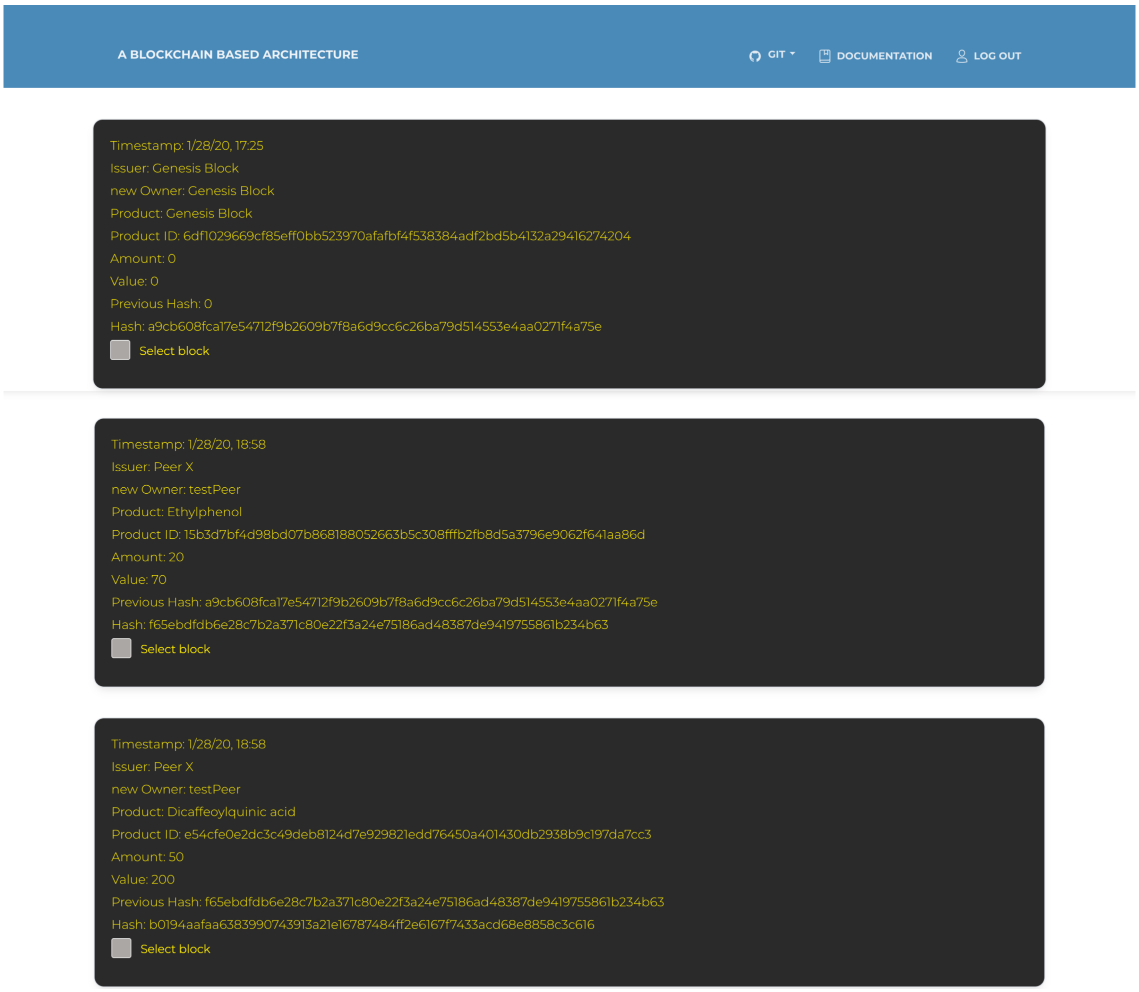
```
Timestamp: 1/28/20, 17:23
Issuer: Genesis Block
new Owner: Genesis Block
Product: Genesis Block
Product ID: 6df1029669cf85eff0bb523970afafbf4f538384adf2bd5b4132a29416274204
Amount: 0
Value: 0
Previous Hash: 0
Hash: e5cf980ad909052fe356f9d03de41c426f42cba14b8ef3b8f483c50feb723c2c
```

```
Timestamp: 1/28/20, 18:16
Issuer: Noodles
new Owner: Peer X
Product: chickpeas
Product ID: d8757c25798c76b7e51350f54924308d32e540f8c1ef98550e6e6a66c3a3b6e3
Amount: 12123
Value: 1
Previous Hash: e5cf980ad909052fe356f9d03de41c426f42cba14b8ef3b8f483c50feb723c2c
Hash: 18f7ea9fc29aa8c75ea9ca74c962cdc9788457d09ac424f2c25217f23954d675
```

```
Timestamp: 1/28/20, 18:16
Issuer: Noodles
new Owner: Peer X
Product: soybeans
Product ID: d2276335643c9e0ab4c528131d4c17955c51166d245a803f4c7ef2901fd24006
Amount: 12123
Value: 121
Previous Hash: 18f7ea9fc29aa8c75ea9ca74c962cdc9788457d09ac424f2c25217f23954d675
Hash: 4898d875917de78d96db8a6ef5656b53247fb9ff5667f3c714c09d6c5a0cd4ca
```

Figure 5.12: Inspecting a partners' chain in the frontend

The updated *testPeer*'s chain after the two purchases is shown in figure 5.13.



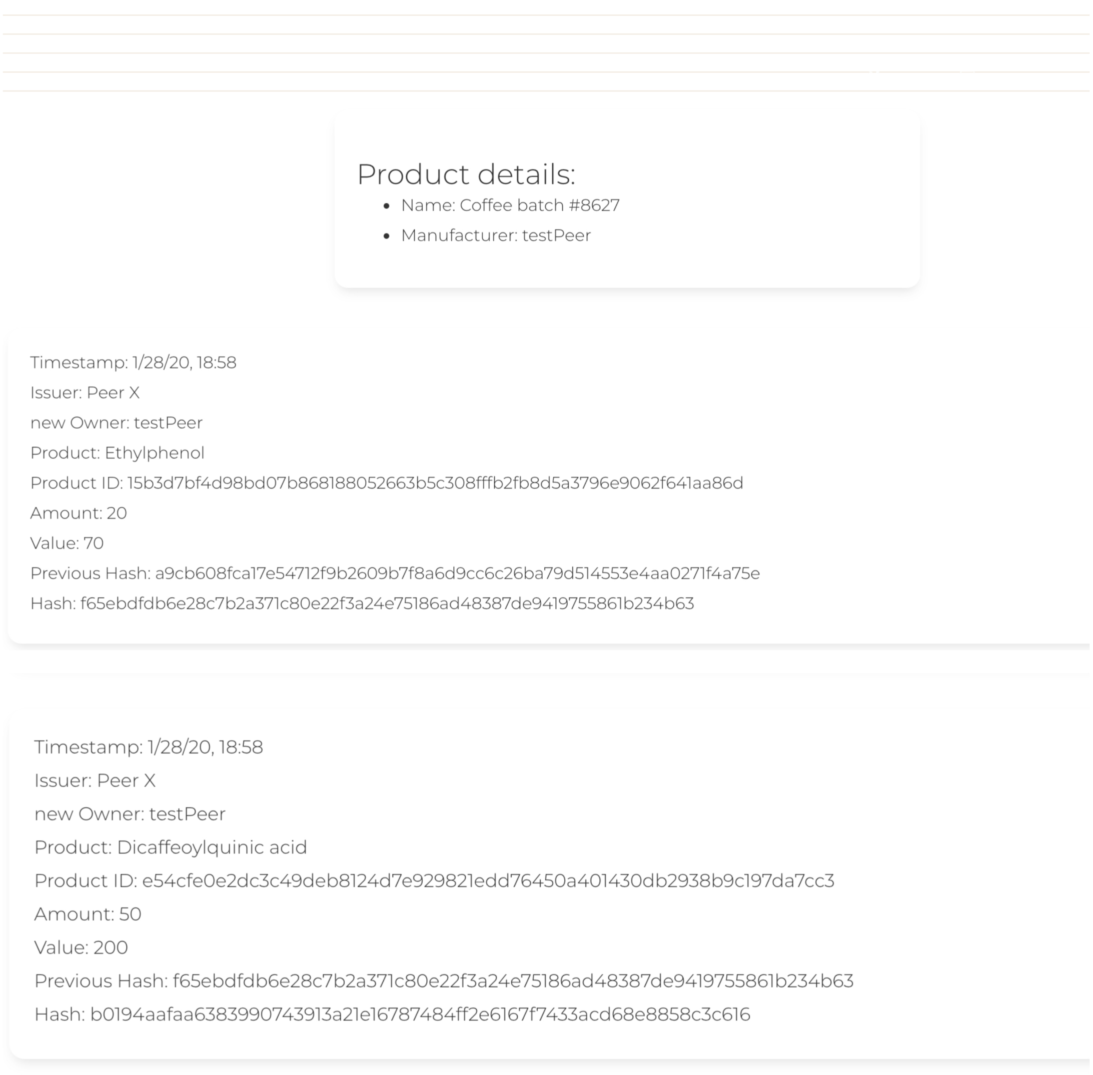
```
Timestamp: 1/28/20, 17:25
Issuer: Genesis Block
new Owner: Genesis Block
Product: Genesis Block
Product ID: 6df1029669cf85eff0bb523970afafbf4f538384adf2bd5b4132a29416274204
Amount: 0
Value: 0
Previous Hash: 0
Hash: a9cb608fca17e54712f9b2609b7f8a6d9cc6c26ba79d514553e4aa0271f4a75e
Select block

Timestamp: 1/28/20, 18:58
Issuer: Peer X
new Owner: testPeer
Product: Ethylphenol
Product ID: 15b3d7bf4d98bd07b868188052663b5c308fffb2fb8d5a3796e9062f641aa86d
Amount: 20
Value: 70
Previous Hash: a9cb608fca17e54712f9b2609b7f8a6d9cc6c26ba79d514553e4aa0271f4a75e
Hash: f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63
Select block

Timestamp: 1/28/20, 18:58
Issuer: Peer X
new Owner: testPeer
Product: Dicaffeoylquinic acid
Product ID: e54cfe0e2dc3c49deb8124d7e929821edd76450a401430db2938b9c197da7cc3
Amount: 50
Value: 200
Previous Hash: f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63
Hash: b0194aafaa6383990743913a21e16787484ff2e6167f7433acd68e8858c3c616
Select block
```

Figure 5.13: Inspecting own chain in the frontend

Finally, *testPeer* assembles a product (Coffee batch). The customer, upon inputting its serial key, is shown the blocks behind the product. This visualization is presented in figure 5.14.


Product details:

- Name: Coffee batch #8627
- Manufacturer: testPeer

```
Timestamp: 1/28/20, 18:58
Issuer: Peer X
new Owner: testPeer
Product: Ethylphenol
Product ID: 15b3d7bf4d98bd07b868188052663b5c308fffb2fb8d5a3796e9062f641aa86d
Amount: 20
Value: 70
Previous Hash: a9cb608fca17e54712f9b2609b7f8a6d9cc6c26ba79d514553e4aa0271f4a75e
Hash: f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63
```

```
Timestamp: 1/28/20, 18:58
Issuer: Peer X
new Owner: testPeer
Product: Dicaffeoylquinic acid
Product ID: e54cfe0e2dc3c49deb8124d7e929821edd76450a401430db2938b9c197da7cc3
Amount: 50
Value: 200
Previous Hash: f65ebdfdb6e28c7b2a371c80e22f3a24e75186ad48387de9419755861b234b63
Hash: b0194aafaa63839907439l3a21e16787484ff2e6167f7433acd68e8858c3c616
```

Figure 5.14: Customer inspection of final product in the frontend

As the figures show, the data that is generated remains consistent as it is stored in several places, which is the most important aspect. With these requests made to the server, the following capabilities of the platform were successfully tested:

- Signing in the platform
- Inspecting own chain
- Inspecting the partners list
- Inspecting a partner's chain
- Inspecting the partners' inventories
- Buying a product from a partner

- Assembling a new product
- Customer inspection of the acquired product

Chapter 6

Conclusions and future work

6.1 Conclusions

With increased demand for transparency, there has been somewhat of a rise of interest in blockchain technology.

This thesis aim is to better understand this technology and how it can be employed in a manufacturing environment through a platform that connects actors in a supply chain.

In the first chapter, some industries, particularly susceptible to fraud, were singled out in an attempt to explain the need for the technology. Also, some blockchain use cases were also presented to give a general overview of the technology. In the second chapter, a look into the state of the art was presented. Topics like industry 4.0, blockchain technology itself and supply chains were approached. Hopefully, this put the work into context by explaining how the technology works and how it fits in the evermore technological world around us. In the third chapter, an overview of the developed platform is presented, followed by descriptions of the implementation itself in the fourth chapter. Finally, in the fifth chapter, tests were conducted in the platform to determine whether it was working as intended.

This platform was developed with the objective of providing the groundwork for a more transparent system between all parties involved in a traditional supply chain. It connects suppliers, retailers and customers, generating truthful information that the parties can rely on. The hybrid implementation allowed flexibility in the design process. Both the traditional model and a blockchain one have its strengths and shortcomings. In this implementation, a compromise between privacy and transparency was reached. By being able to choose the convenient aspects of both, a common ground was attained.

The tests conducted in chapter five consisted in a tour through the platform, as a made up retailer added its business partners in the network, purchased products from them and assembled a final product to be sold to the public. All of the tests showed that the platform is storing the relevant information in the right places in an immutable way, thus showing that it is indeed working as intended. The final test, a customer wanting to track its products' origins, also returned the correct information.

While it still has a few rough edges, as previously discussed on this chapter, hopefully it was shown that it is possible for an alternative, more transparent architectures to be used. Companies can reap benefits in knowing with greater depth their business partners and costumers also gain by getting better insights on their product.

6.2 Future work

The main focus of future work would be to decentralize the network. Some difficulties arise mostly because there aren't truly common ledgers and therefore not too many nodes to use as storages. Because it doesn't make sense to save a copy of a peer's ledger in a stranger node, copies of that same ledger would only be saved in the owner and potentially its business partners. A ledger stored on a few select nodes could be at risk of adulteration since only a small number of machines would need to be compromised. Big conglomerates with up to hundreds of partners would probably be safe but small ventures that operate within a small circle could potentially be at risk. A strategy would need to be developed to tackle these problems. Another future feature would be to check whether the platform is compromised by analysing every block's hash automatically and check if they indeed match instead of having to do it manually.

Aside from that, future versions should also focus on modularity. Businesses vary a great deal between themselves, therefore it is always going to be hard to design something to suit each one. What kind of information should be stored about a product? And about a partner? How to deal with the asymmetry of business relations (should a small supplier be able to inspect a big company's ledger)? These and other, are questions that can have different answers depending on the businesses in question. With a modularity approach, each conglomerate could set up the network to suit their specific needs. Some quality of life updates, having an automatic buy action in place to buy product X every Y days from a partner for example, would also be increase the platform's value.

Regardless of this work in specific, it should be important for the people behind the development of future system implementations and even business models to think about the customer and how to reduce the possibility of fraud and corruption. These maledict practices, specially when occurring in the public sector, actively hinder the progress and development of entire countries and populations. It's up to everyone to keep pushing the transparency agenda.

Bibliography

[1] L. Holmberg. "Wine fraud." In: *International Journal of Wine Research* 2 (2010).

[2] A. Lavorgna. "The online trade in counterfeit pharmaceuticals: New criminal opportunities, trends and challenges." In: *European Journal of Criminology* 12.2 (2015).

[3] P. Aldhous. "Murder by medicine." In: *Nature - International Journal of Science* 434 (2005).

[4] M. Conoscenti, A. Vetrò, and J. C. D. Martin. "Blockchain for the Internet of Things: A systematic literature review." In: *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. 2016.

[5] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann. "Industry 4.0." In: *Business & Information Systems Engineering* 6.4 (Aug. 2014), pp. 239–242.

[6] C. Klingenberg. "Industry 4.0: what makes it a revolution?" In: July 2017.

[7] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, and A. G. Frank. "The expected contribution of Industry 4.0 technologies for industrial performance." In: *International Journal of Production Economics* 204.C (2018), pp. 383–394.

[8] M. Hermann, T. Pentek, and B. Otto. "Design Principles for Industrie 4.0 Scenarios." In: *2016 49th Hawaii International Conference on System Sciences (HICSS)*. Jan. 2016, pp. 3928–3937.

[9] E. sayed ali ahmed and Z. Kamal Aldein Mohammed. "Internet of Things Applications, Challenges and Related Future Technologies." In: *world scientific news* (Jan. 2017).

[10] C. Saidu, A. Usman, and P. Ogedebe. "Internet of Things: Impact on Economy." In: *British Journal of Mathematics Computer Science* 7 (Jan. 2015), pp. 241–251.

[11] E. A. Lee. "Cyber Physical Systems: Design Challenges." In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. May 2008, pp. 363–369.

[12] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry ; Final Report of the Industrie 4.0 Working Group*. Forschungsunion.

[13] R. Schlaepfer, M. Koch, and P. Merkofer. *Industry 4.0 - Challenges and solutions for the digital transformation and use of exponential technologies*. Tech. rep. Deloitte AG, 2015.

[14] L. M. Camarinha-Matos, H. Afsarmanesh, N. Galeano, and A. Molina. “Collaborative networked organizations – Concepts and practice in manufacturing enterprises.” In: *Computers Industrial Engineering* 57.1 (2009). Collaborative e-Work Networks in Industrial Engineering, pp. 46–60.

[15] T. Stock and G. Seliger. “Opportunities of Sustainable Manufacturing in Industry 4.0.” In: *Procedia CIRP* 40 (2016). 13th Global Conference on Sustainable Manufacturing – Decoupling Growth from Resource Use, pp. 536–541.

[16] S. Nakamoto. “Bitcoin: a peer-to-peer electronic cash system, Oct. 2008.” In: *URL http://www. bitcoin. org/bitcoin. pdf* (2008).

[17] M. E. Peck. “Blockchains: How they work and why they’ll change the world.” In: *IEEE spectrum* 54.10 (2017), pp. 26–35.

[18] M. Swan. *Blockchain: Blueprint for a new economy*. "O’Reilly Media, Inc.", 2015.

[19] J. Michael, A. Cohn, and J. R. Butcher. “BlockChain technology.” In: *The Journal* (2018).

[20] E. Grange. *Mesh World P2P Simulation Hypothesis*. 2016. URL: `https://www.delphitools.info/DWSH/` (visited on 01/30/2019).

[21] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. “Bitcoin and Cryptocurrency Technologies.” In: *Network Security* 2016.8 (2016).

[22] V. Buterin. “The Meaning of Decentralization.” In: *Medium* (Feb. 2017).

[23] W. Diffie and M. Hellman. “New directions in cryptography.” In: *IEEE Transactions on Information Theory* 22.6 (Nov. 1976).

[24] S. Goldwasser, S. Micali, and R. Rivest. “A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks.” In: *SIAM Journal on Computing* 17.2 (1988).

[25] L. Zhu and L. Zhu. “Electronic signature based on digital signature and digital watermarking.” In: *2012 5th International Congress on Image and Signal Processing*. 2012.

[26] M. Pilkington. “Blockchain technology: principles and applications.” In: *Research handbook on digital transformations* (2016).

[27] G. W. Peters and E. Panayi. “Understanding Modern Banking Ledgers Through Blockchain Technologies: Future of Transaction Processing and Smart Contracts on the Internet of Money.” In: *Banking Beyond Banks and Money: A Guide to Banking Services in the Twenty-First Century*. 2016.

[28] A. Hari and T. V. Lakshman. "The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet." In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. HotNets '16. 2016.

[29] URL: https://coinmarketcap.com/currencies/bitcoin/#charts (visited on 02/01/2019).

[30] H. Gilbert and H. Handschuh. "Security Analysis of SHA-256 and Sisters." In: *Selected Areas in Cryptography*. Springer Berlin Heidelberg, 2004.

[31] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse. "Bitcoin-NG: A Scalable Blockchain Protocol." In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016.

[32] R. Blasetti. *Brace Yourself For The Bitcoin Hard Fork*. 2017. URL: https://decentralize.today/brace-yourself-for-the-bitcoin-hard-fork-5c42e61e596c (visited on 02/02/2019).

[33] V. Buterin et al. "A next-generation smart contract and decentralized application platform." In: *Ethereum White Paper* (2013).

[34] K. J. O'Dwyer and D. Malone. "Bitcoin mining and its energy footprint." In: *Institution of Engineering and Technology* (2014).

[35] L. Foundation. "A Blockchain Platform for the Enterprise." In: *https://hyperledger-fabric.readthedocs.io/en/release-1.4/* (2019).

[36] L. Mearian. *Why hybrid blockchains will dominate ecommerce*. 2019. URL: https://www.computerworld.com/article/3435770/why-hybrid-blockchains-will-dominate-ecommerce.html (visited on 12/25/2019).

[37] I. Facebook. "The Libra Blockchain." In: *Libra White Paper* (2019).

[38] A. Baliga. "Understanding blockchain consensus models." In: *Persistent* (2017).

[39] K. Christidis and M. Devetsikiotis. "Blockchains and Smart Contracts for the Internet of Things." In: *IEEE Access* 4 (2016).

[40] J. T. Mentzer, W. DeWitt, J. S. Keebler, S. Min, N. W. Nix, C. D. Smith, and Z. G. Zacharia. "Defining supply chain management." In: *Journal of Business logistics* 22 (2001).

[41] *Blockchain: The technology reshaping supply chain management*. 2018. URL: https://kodiakrating.com/2018/07/30/blockchain-the-technology-reshaping-supply-chain-management/ (visited on 02/10/2019).

[42] S. A. Abeyratne and R. P. Monfared. "Blockchain ready manufacturing supply chain using distributed ledger." In: *International journal of research in engineering and technology* 05 (2016).

[43] "A mucky business." In: *The Economist* (2015).

[44] M. Frentrup, L. Theuvsen, et al. “Transparency in supply chains: Is trust a limiting factor.” In: *Trust and Risk in Business Networks, ILB-Press, Bonn* (2006).

[45] K. Kim and T. Kang. *Does Technology Against Corruption Always Lead to Benefit? The Potential Risks and Challenges of the Blockchain Technology*. 2017.

[46] Various. *Node.js*. `https://github.com/nodejs/node`.

[47] Google. *Angular*. `https://github.com/angular/angular`.

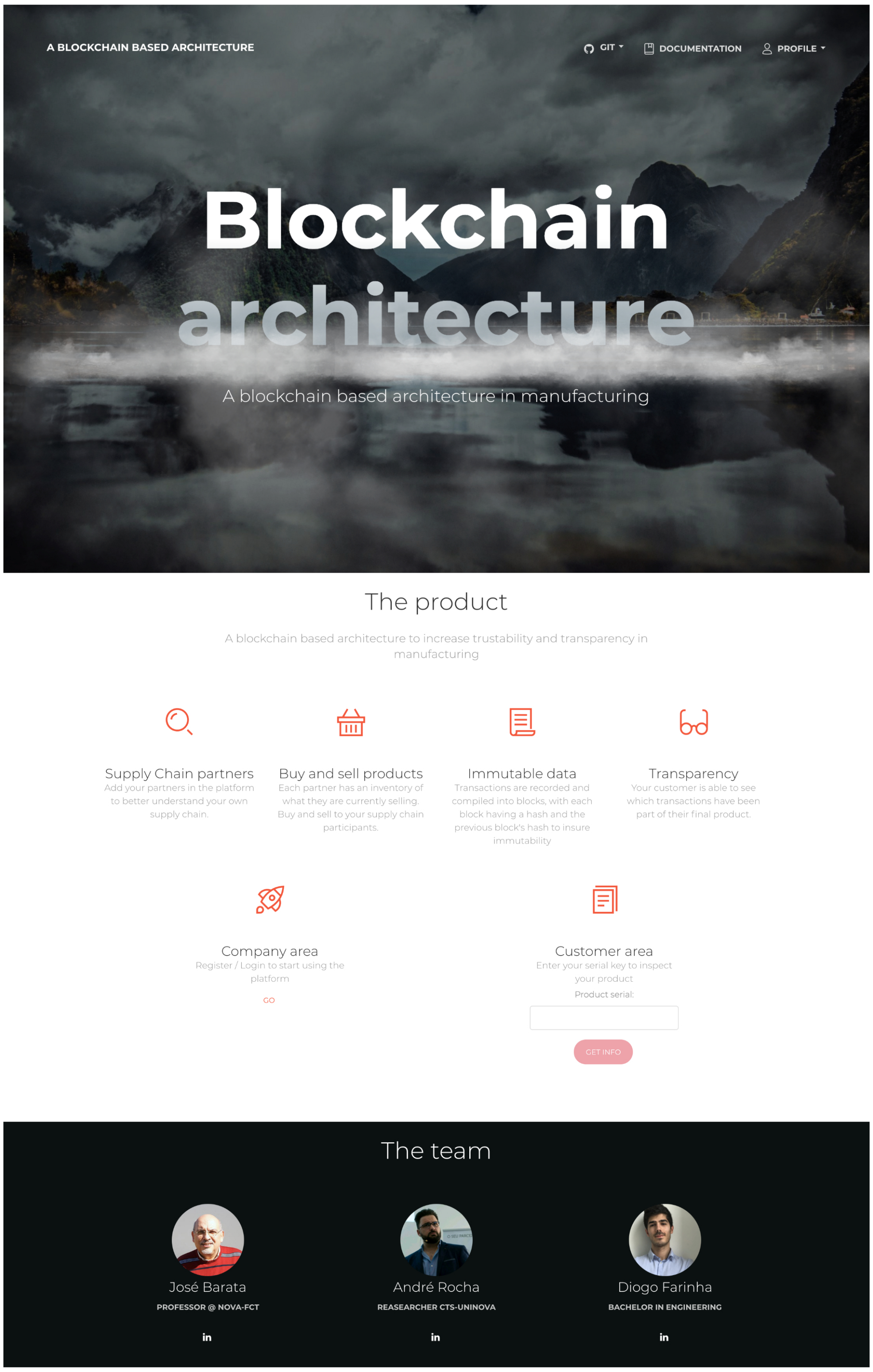
[48] Various. *MongoDB*. `https://docs.mongodb.com/manual/introduction/`.

[49] Various. *Express.js*. `https://github.com/expressjs/express`.

[50] Various. *Passport*. `https://www.npmjs.com/package/passport`.

Annex I

Annex 1



A BLOCKCHAIN BASED ARCHITECTURE

GIT DOCUMENTATION PROFILE

Blockchain architecture

A blockchain based architecture in manufacturing

The product

A blockchain based architecture to increase trustability and transparency in manufacturing

Supply Chain partners
Add your partners in the platform to better understand your own supply chain.

Buy and sell products
Each partner has an inventory of what they are currently selling. Buy and sell to your supply chain participants.

Immutable data
Transactions are recorded and compiled into blocks, with each block having a hash and the previous block's hash to insure immutability

Transparency
Your customer is able to see which transactions have been part of their final product.

Company area
Register / Login to start using the platform

GO

Customer area
Enter your serial key to inspect your product

Product serial:

GET INFO

The team

José Barata
PROFESSOR @ NOVA-FCT

André Rocha
REASEARCHER CTS-UNINOVA

Diogo Farinha
BACHELOR IN ENGINEERING

Figure I.1: The homepage

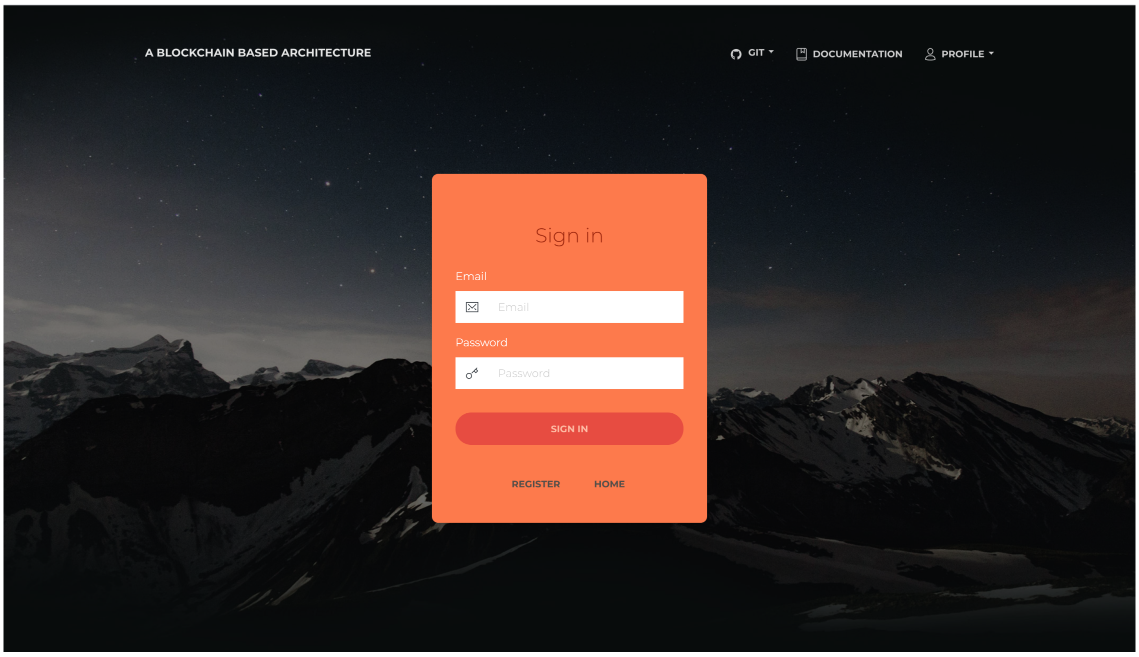


Figure I.2: Login

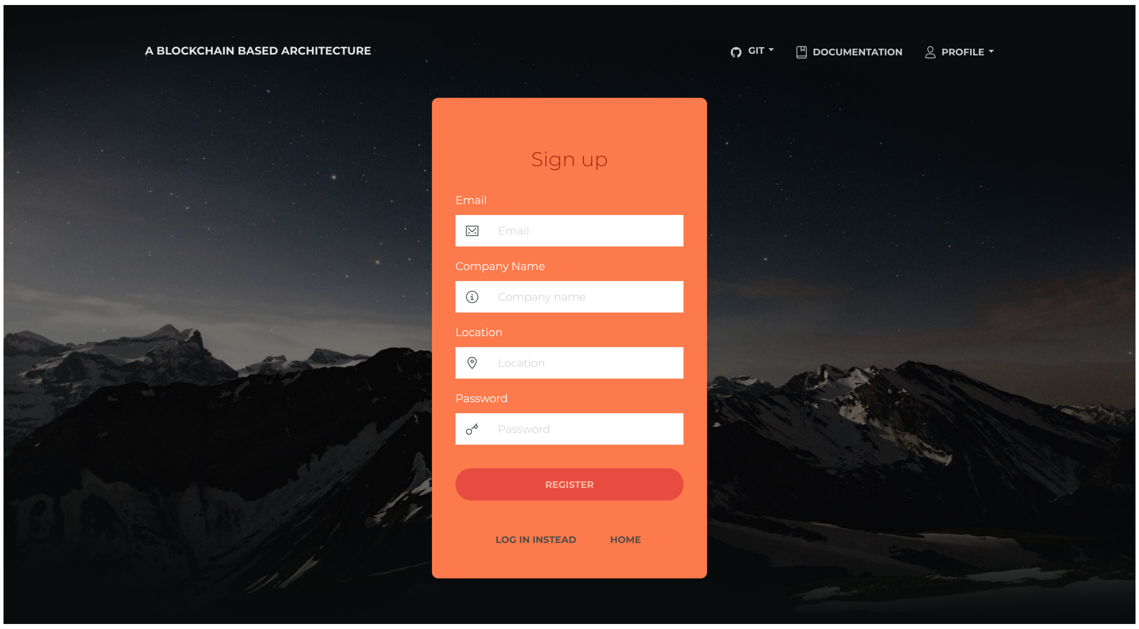
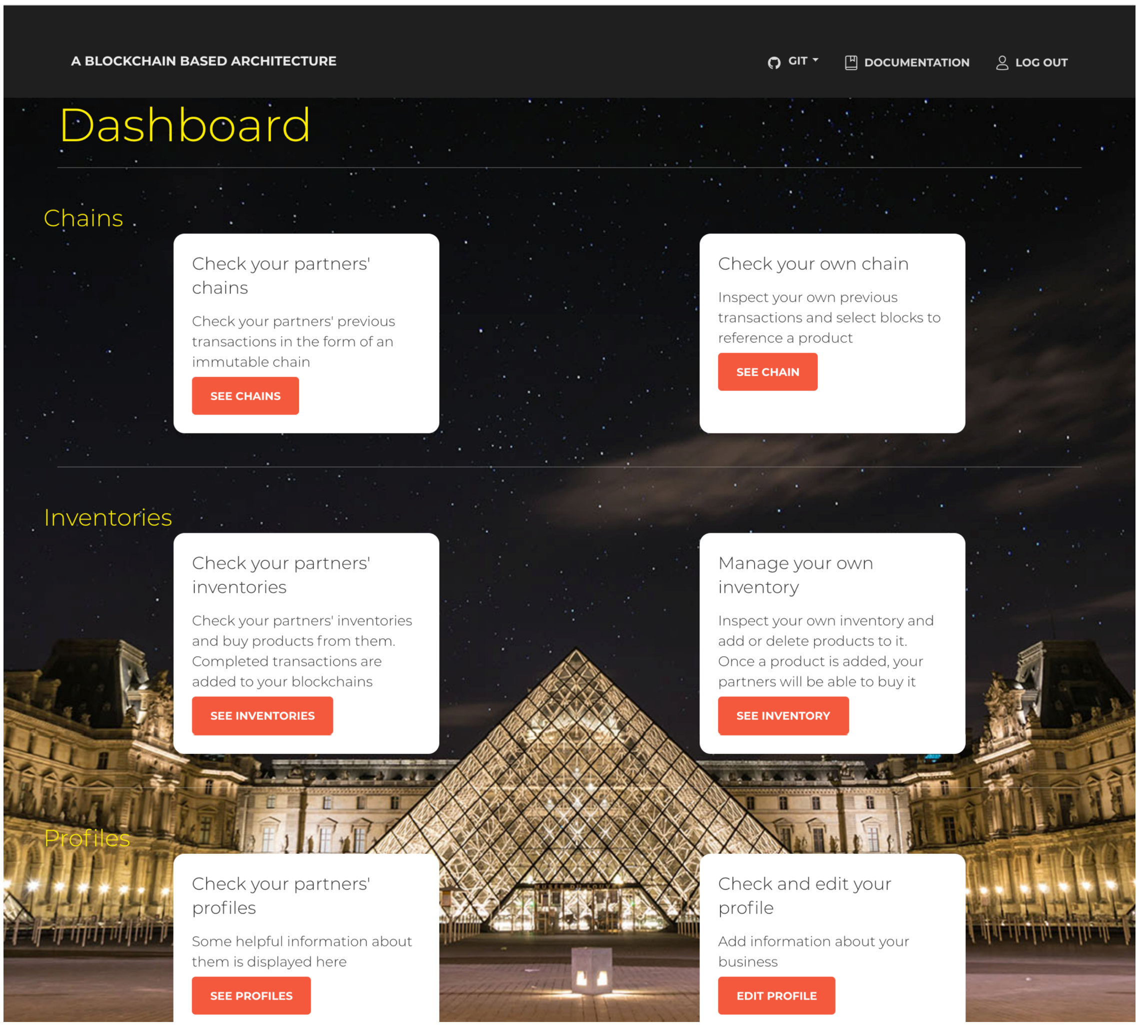


Figure I.3: Registration



Dashboard

Chains

Check your partners' chains

Check your partners' previous transactions in the form of an immutable chain

SEE CHAINS

Check your own chain

Inspect your own previous transactions and select blocks to reference a product

SEE CHAIN

Inventories

Check your partners' inventories

Check your partners' inventories and buy products from them. Completed transactions are added to your blockchains

SEE INVENTORIES

Manage your own inventory

Inspect your own inventory and add or delete products to it. Once a product is added, your partners will be able to buy it

SEE INVENTORY

Profiles

Check your partners' profiles

Some helpful information about them is displayed here

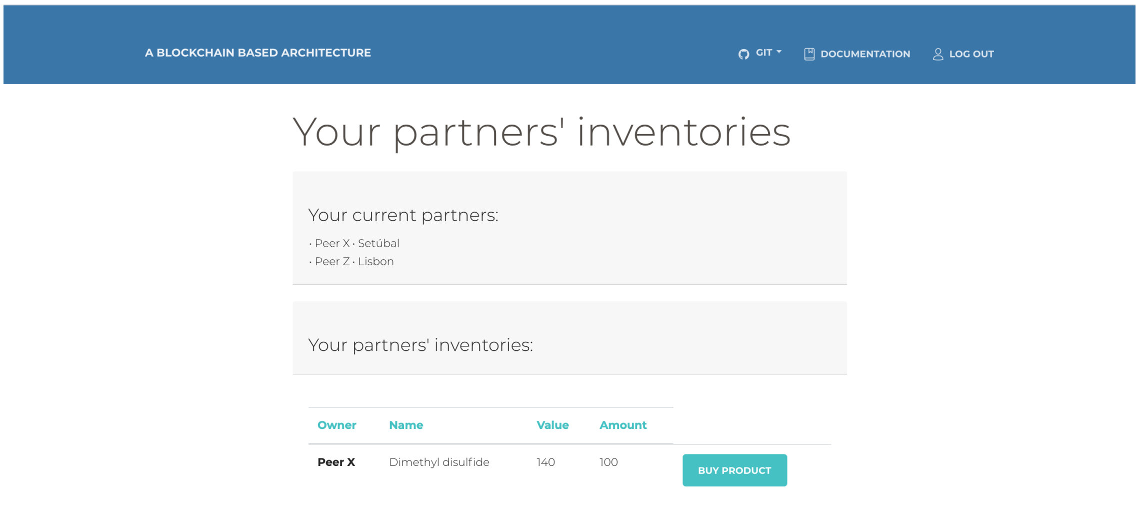
SEE PROFILES

Check and edit your profile

Add information about your business

EDIT PROFILE

Figure I.4: The peer's area


A BLOCKCHAIN BASED ARCHITECTURE

Your partners' inventories

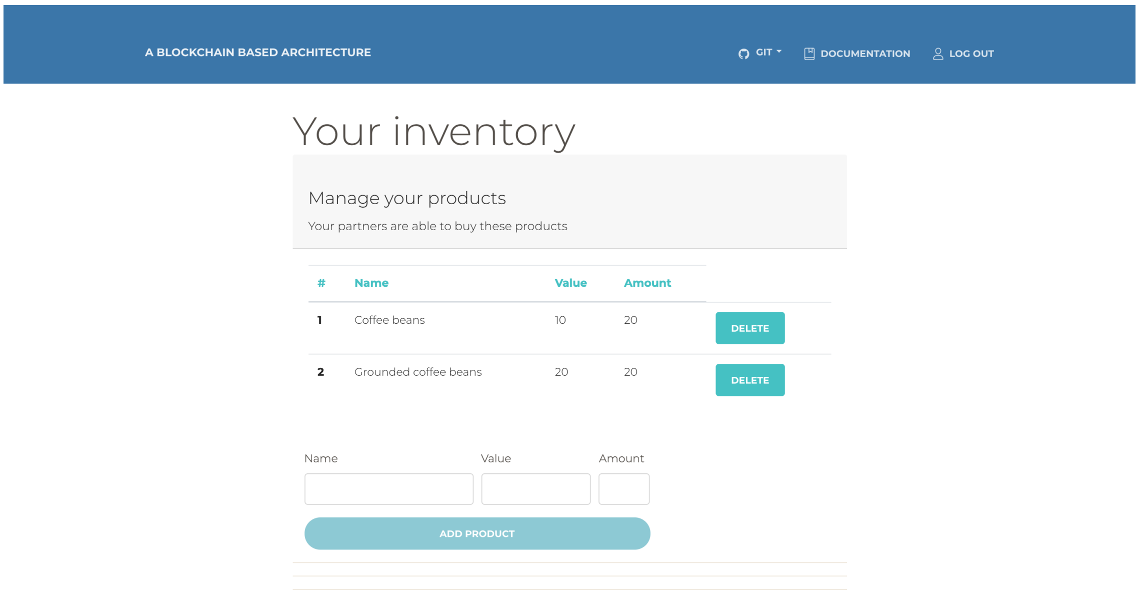
Your current partners:

· Peer X · Setúbal
· Peer Z · Lisbon

Your partners' inventories:

Owner	Name	Value	Amount	
Peer X	Dimethyl disulfide	140	100	BUY PRODUCT

Figure I.5: Inspecting partners' inventories


A BLOCKCHAIN BASED ARCHITECTURE

Your inventory

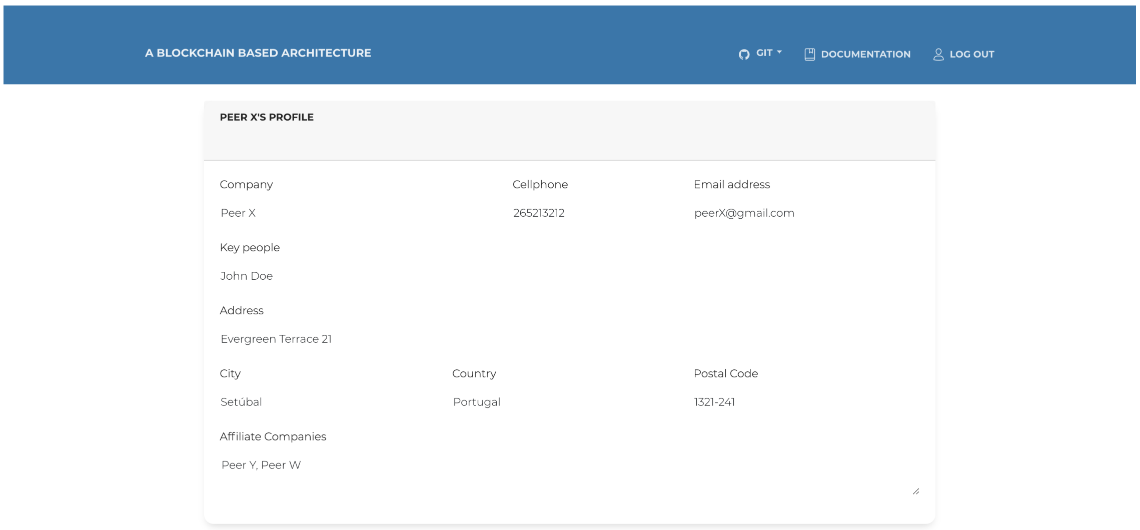
Manage your products

Your partners are able to buy these products

#	Name	Value	Amount	
1	Coffee beans	10	20	DELETE
2	Grounded coffee beans	20	20	DELETE

Name	Value	Amount

Figure I.6: Inspecting own inventory



PEER X'S PROFILE

Company	Cellphone	Email address
Peer X	265213212	peerX@gmail.com

Key people

John Doe

Address

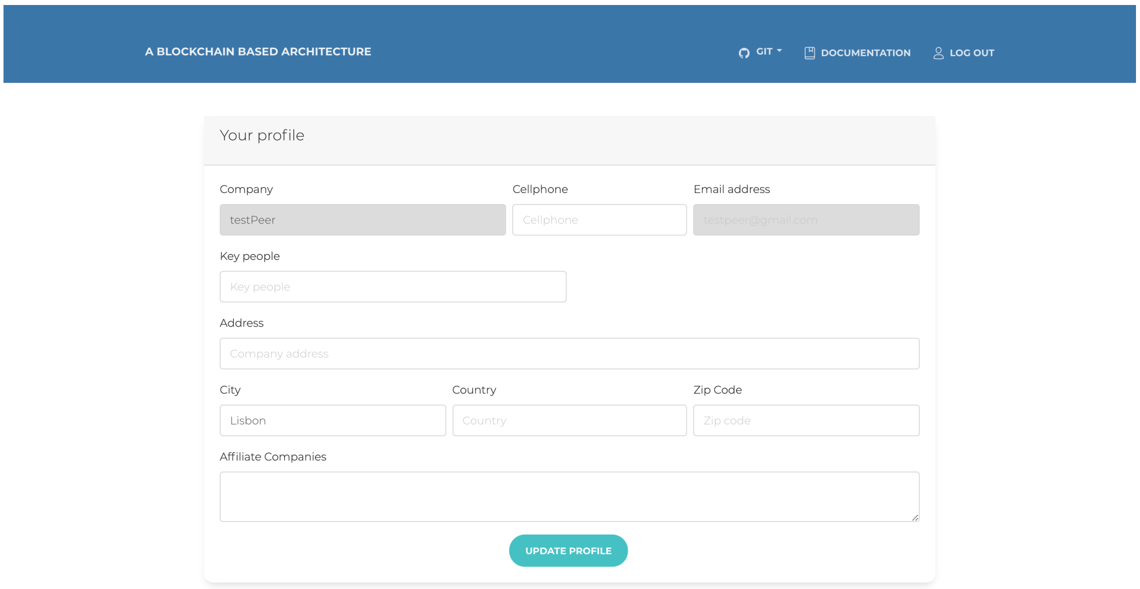
Evergreen Terrace 21

City	Country	Postal Code
Setúbal	Portugal	1321-241

Affiliate Companies

Peer Y, Peer W

Figure I.7: Inspecting partners' profile



Your profile

Company	Cellphone	Email address
testPeer	Cellphone	testpeer@gmail.com

Key people: Key people

Address: Company address

City	Country	Zip Code
Lisbon	Country	Zip code

Affiliate Companies

Figure I.8: Editing own profile