POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Blockchain and smart contracts in the Fashion industry

A Decentralized Application built on Ethereum and Hyperledger Fabric

Relatori
prof. Fabrizio Lamberti
ing. Valentina Gatteschi

Candidato
Stefano Franzoni

Tutor aziendale
dott. Alfredo Favenza

Sessione di Luglio 2020

Acknowledgements

A mia madre. Alla curiosità, all'amore e alla tenacia che mi ha trasmesso in questi anni.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last few years, the Blockchain technology has been living an era of growth, many different applications for it being found in several different fields. The first application of blockchain was in finance with the Bitcoin, right after many environments such as logistics, law and administration found huge applications in it.

This kind of technology is based on a peer to peer network that allows to transfer assets among users involved in the network, without the need for a third party. The overall network and the transactions, performed by the joint nodes, are handled by a consensus algorithm that performs the validation of the transactions and updates the ledger by adding the transactions block to the chain. The ledger is a file containing a set of records, each record is a transaction processed by the network. Moreover, all the nodes involved in the network share the same ledger that contains the state of the overall network. The most significant use of the blockchain technology is in the finance environment, which founds a great instrument in cryptocurrencies, exploiting the decentralized architecture, the security advantages and the transparency. In fact, everyone has read access over the network and he can visualize the data of the transactions processed.

In the last few years, many companies have been getting interested in the blockchain applications. Since, in many cases, the data that they share are sensitive information and companies don't want to keep them public, the companies need for a different solution. Therefore, recently, this interest has resulted in the first permissioned blockchain solutions, maintained by a consortium system of nodes.

In fact, the blockchain world nowadays are split between permissioned and permissionless blockchains:

- **Permissionless Blockchain**: The permissionless blockchain is a fully decentralized network where anyone in the world can read or send transactions, as well as participate in the consensus process.

- **Permissioned Blockchain**: the permissioned solution is a blockchain where the consensus process is controlled by a pre-selected set of nodes, which could be defined as a "partially decentralized" network. Moreover, a membership mechanism could be implemented, in order to to handle the read and write access over the network.

Based on the above considerations, the two main objectives of this thesis work are:

- **Blockchain and supply chain management**: The goal is to create a blockchain solution for the management of the supply chain process of a Fashion Company. In particular, the thesis addresses the issue of transparent fashion upcycling, which is generally characterized by processes involving many different actors. The goal of the blockchain-based solution devised is to track the items over the flow more clearly and transparently as possible.

- **Cross-chain solution**: The goal is to implement a cross-chain interaction between Hyperledger Fabric and Ethereum network. Fabric manages all the aspects related to the supply chain (orders, production, part of sale process). Ethereum instead is only used to the end-user sell process of the items. The goal is to reach the interoperability between public and private blockchains.

The target reached at the end of the developed thesis work would conceivably be:

- **Simple management process**: It is to reach a simplified model of the overall management process of the transactions and users involved into the system.

- **Supply Chain increased transparency**: A simplified handling process of the supply chain. The goal is to make as clear as possible the tracking process of the clothes over the actors involved, from the producing of the items to the selling process.

- **Technology improvements and modular solutions**: A cross-chain technology means to generalize a solution that could be applied to other use cases. The integration among more blockchains networks, is a big challenge nowadays, in the following chapter it is described in details the current situation and the solution chosen.

The rest of the document is structured as follows:

- **Chapter 2 - State of the art**: focuses on previous studies in this field. It gives an overview of the current solutions to the issue faced in the thesis work.
- **Chapter 3 - Solution**: explains how the problem at hand was solved using the method and theory. It shows all the technologies used to implement the chosen solution and how they interact with each other describing the implementation details.
- **Chapter 4 - Results**: describes the outcome of the tests done.
- **Chapter 5 - Conclusion**: handles the conclusion and future works, I summarize the overall conclusions.

Chapter 2

State of the art

This chapter gives an overview of the current solutions and technologies used during the thesis work. It gives an overview about blockchain technology and its application in fashion environment.

2.1 What is Blockchain

In the previous chapter, I briefly described the Blockchain behaviors. Blockchain is a growing list of records, called blocks, that are linked. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data.[1]

By design, a blockchain is immutable and resistant to data modification. It runs over a peer to peer network and each node of the network maintains a copy of the *distributed ledger*. Therefore, exploiting the peer to peer network architecture and the cryptographic science, it is obtained a distributed data storage, immutable, secure, and continuously synchronized among network nodes. The other core part of the Blockchain is the *consensus* mechanism, it is the algorithm that handles the consensus process in charge of validate transactions and add it to the chain of blocks, performing the ledger update.

2.1.1 Consensus mechanism

The main consensus algorithms are:

- **Byzantine fault tolerance - BFT**: The concept of Byzantine Fault Tolerance in Blockchain is the feature of reaching an agreement or consensus about particular blocks based on the proof of work, even when some nodes are failing to respond or giving out malicious values to misguide the network. The main objective of BFT is to safeguard the system even when there are some faulty nodes. This will also help to reduce the influence of faulty nodes.[2]

- **Proof of work - PoW**: it deters *denial-of-service* attacks. A proof of work is a piece of data which is difficult to produce but easy for others to verify and which satisfies certain requirements. Producing a proof of work can be a random process with low probability so that a lot of trial and error is required on average before a valid proof of work is generated. In other words, it is like a problem to solve spending a lot of computing power to validate transactions and create new blocks.[3]
- **Proof of stake - PoS**: it uses a pseudo-random election process to select a node to be the validator of the next block, based on a combination of factors that could include the staking age, randomization, and the node's wealth. The size of the stake determines the chances for a node to be selected as the next validator to forge the next block - the bigger the stake, the bigger the chances. Where in Proof of Work-based systems more and more cryptocurrency is created as rewards for miners, the Proof-of-Stake system usually uses transaction fees as a reward.[4]

Usually, over the network, there are specific nodes, called *miners* with huge computational power, which handles the transactions in exchange for *transaction fees* and they have a reward for each block created.

Figure 2.1 shows an example of how the PoW algorithm works.

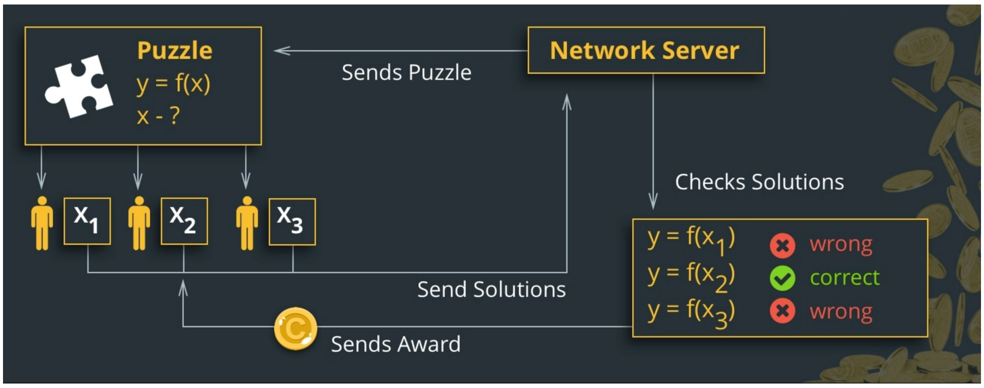


Figure 2.1. How PoW works

2.1.2 51% attack

It can be performed when a group of miners controlling more than 50% of the network's mining hash rate or computing power. The attackers would be able to prevent new transactions from gaining confirmations, allowing them to halt payments between some or all users. Figure 2.2 shows the attack situation, considering that in the blockchain is kept just the longer branch, meanwhile, the shorter one is discarded.[5]

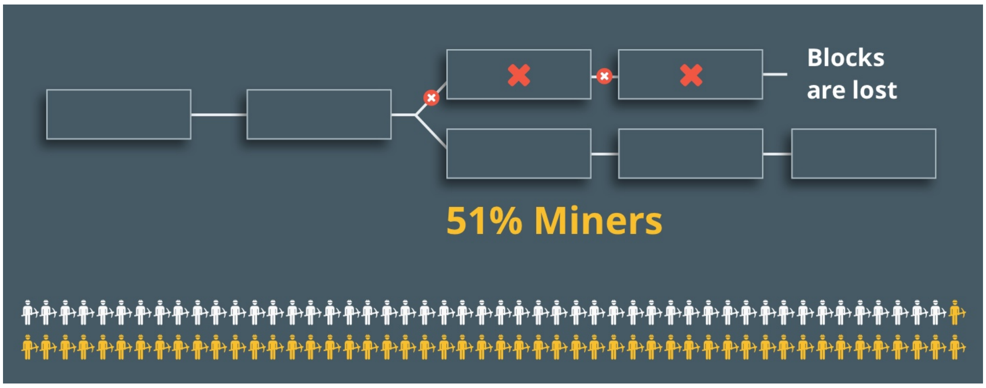


Figure 2.2. 51% attack

Nevertheless, the 51% attack does not pay, because it needs a huge computational power to be performed, and once the network is compromised, no one continues to use it and the token's price goes to zero. Therefore, it brings everybody to play by the rules.

2.1.3 Bitcoin

The Blockchain concept was born in 2008 when *Satoshi Nakamoto* (it is an alias behind which there are a person or a group of people) published the whitepaper *"Bitcoin: A Peer-to-Peer Electronic Cash System"*; a manifesto where Satoshi explained the implementation details of a digital currency that can be transferred over a peer-to-peer network without the need for a central bank that controls it. The first Blockchain implementation is based on the Bitcoin protocol. Nakamoto inside the whitepaper exposes a peer-to-peer network, where each node shares a distributed ledger containing all the transactions done. Each node of the network contains a public key, that is the public address, and a private key used to sign transactions. The asset exchanged over the network is the BTC, a cryptocurrency transferred among the nodes. In 2009 he releases the open-source software and Bitcoin become reality.

2.1.4 Smart Contract

The smart contract concept was introduced by computer scientist *Nick Szabo*. It is a computer program or a transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement.[6]. Since the integration of the smart contract inside Blockchain, it is possible to exploit the network to transfer each kind of asset described by the contract.

2.2 Current state of networks solution

Starting from the Introduction's consideration, below it is listed the details of the three main solutions for blockchain networks[7]:

- **Public blockchains**: they are peer to peer networks in which anyone in the world has read access to the network, anyone can send transactions over the network, and anyone in the world can participate in the consensus process. In a public blockchain every node is potentially untrusted, so the consensus mechanism is developed in order to prevent every malicious node that could compromise data and transactions performed over the network. The entire architecture and consensus are distributed in order to minimize the liability of data manipulation. The consensus process defines the blocks that get added to the chain determining the current state of the network. The security issue is solved by a mix of cryptographic algorithms and cryptoeconomics solution. The idea is to combine economic incentives proportional to the resources that the node can bring to bear. The resources targeted depend on the consensus algorithm used. For example proof of work(PoW) involves computational resources into the consensus mechanism. On the other hand, proof of stake(PoS) involves the token amount of the node involved in consensus. These blockchains are generally considered to be "fully decentralized".[7]

- **Consortium blockchains**: The basic idea of the consortium blockchain is that the network is composed by a set of trusted or semi-trusted nodes, that compose the governance of the network. The consensus mechanism is not as complex as that of public blockchains, because the starting hypothesis is different and usually it needs to have a good performance and low latency of the transactions. I provide an example to understand how governance's nodes are involved in the consensus process: One might imagine a consortium of 15 financial institutions, each of them operates a node and 10 of them must sign every block in order for the block to be valid. The right to read the blockchain may be public, or restricted to the participants, and there are also hybrid routes such as the root hashes of the blocks being public together with an API that allows members of the public to make a limited number of queries and get back cryptographic proofs of some parts of the blockchain state. These blockchains may be considered "partially decentralized".[7]

- **Fully private blockchains**: a fully private blockchain is a consortium blockchain where write permissions are kept centralized to just one organization. Read permissions may be public or restricted to an arbitrary extent. Likely applications include database management, auditing, etc internal to a single company. Therefore, public readability may not be necessary at all in many cases, although, in other cases public auditability is desired.[7]

2.2.1 Behind the Blockchains

With the blockchain technology, cryptographic science found the most applications. The concept behind the public blockchain is that all the transaction data are completely public; nevertheless, the identity of the user involved is kept secret. This idea has found a huge application in the cryptocurrencies environment. The *Fintech*[1] is the environment in which the public blockchains has found the most applications. The main rule of the overall system is **"keep it transparent , safe and anonymous"**, it means that all the transactions processed by the public blockchain networks are **transparent** and every node has read access. The **Safe** concept is granted by the combination of cryptographic science and economic incentives. **Anonymous** has granted thanks to the cryptographic science applications, for example, the bitcoin wallet is based on a key pair computed on elliptic curve algorithms. If no one shares the identity associated with the wallet public key, or keep the private key public, the user identity continues to be anonymous.

As explained above the blockchain world is divided between public and private blockchain. Figure 2.3 shows an overview of the main difference of the two kind of networks.

[1] It is the technology and innovation that aims to compete with traditional financial methods in the delivery of financial services.

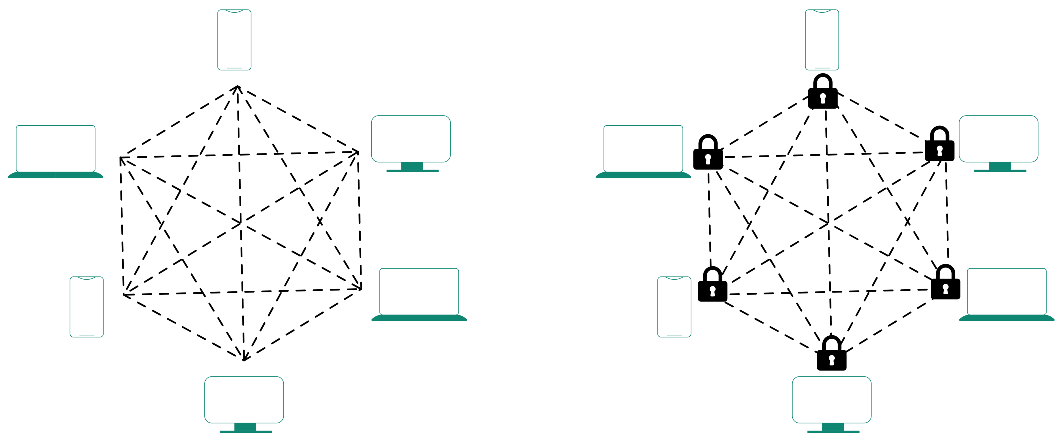


Figure 2.3. Public vs Private networks

To understand the behavior about public and private blockchains, we are going to list the features for both, in order to adapt the choice based on own needs:

1. The main advantages of the **Public blockchain** could fall into two major categories:

   (a) Public blockchains provide a way to protect the users of an application from the developers; the code is public and everyone can see how it works. This solution limits the authority of the developers over the application. Moreover, the user identity is always mapped into a wallet address.

   (b) Public blockchains are open, and therefore are likely to be used by many entities and gain some network effects. Besides, the public blockchain fully eliminates intermediaries. Here is an example of a transfer of ownership case. A wants to sell an item to B. Right now there is a standard risk problem of the involved counterparty: if A sends first, B may not send money, and if B sends first the money A might not send the item. All the problems related to these kinds of cases could be resolved using smart contracts, running over the public blockchain, moreover, the costs is close to zero. With the smart contract implementations, A can send the item, to be sold, to a program that immediately sends it to the first person that in the meanwhile sends money to the program.

2. Compared to the public blockchain, the advantages of a **Private blockchain** are:

   (a) The consortium or companies running a private blockchain can easily, if desired, change the rules of a blockchain, revert transactions, modify balances, etc. In some cases, eg. national land registries, this functionality is necessary.

(b) The validators are known, so any risk of a 51% attack[2], arising from some miner collusion in China, does not apply.

(c) Transactions are cheaper, since they only need to be verified by a few nodes that can be trusted to have very high processing power, and do not need to be verified by ten thousand laptops. This is a hugely important concern right now, as public blockchains tend to have transaction fees exceeding $0.01 per tx.

(d) Nodes can be trusted to be very well-connected, and faults can quickly be fixed by manual intervention, allowing the use of consensus algorithms which offer finality after much shorter block times.

(e) If read permissions are restricted, private blockchains can provide a greater level of, well, privacy.

From many analysis it gets out that the 75% of already implemented projects are designed specifically for private aim [8], which means that a need is growing to improve the Consortium Blockchains that allow a memberships mechanism build for company use cases, which maintains the transactions private to guarantee the privacy of the business process and data.
On the other hand, in some processes it is useful to implements public blockchain solutions, so there is a growing need to improve the interoperability about a consortium and public blockchains into a cross-chain solution.

Table 2.1 shows a comparing among three of the main blockchain networks. The comparison is based on the kind of the **network**, the **currency** that runs over each network, the **consensus** algorithm used and the possibility to develop and run **smart contract** over that.

2.2.2 CrossChain and interoperability

Michael Burgess, chief operating officer of Ren states that **"All interoperability solutions will likely have trade-offs; so it's a matter of designing systems that find a balance between security, governance, adaptability, and economic incentives that suit their target market."**

[2]It is a potential attack on a blockchain network, where a single entity or organization is able to control the majority of the hash rate, potentially causing a network disruption.https://academy.binance.com/security/what-is-a-51-percent-attack

Name	**Network**	**Currency**	**Consensus**	**Smart Contract**
Bitcoin	Public	Bitcoin	PoW	Possible but less extendible
Ethereum	Public	Ether	PoS	Solidity, Vyper
Hyperledger	Permissioned	None	PBFT	Go, Java, Javascript, Solidity

Table 2.1. Comparing among blockchains features

"Private chains operating without distributed consensus are more prone to data manipulation and the integrity of the data/assets being transferred from a private, permissioned and centralized chain to a more decentralized chain could be questioned. Overall, there is no one solution that fits all in terms of being public/private, centralized/decentralized — it is a broad spectrum with specific trade-offs."[9] , quoting the words of Agarwal, CEO of Persistence.

What is getting from the point of view of the industry experts; it is a trade-off solution to obtain cross-chain interoperability, between public and private.

Limitations

Considering the pros and cons of each network listed, interoperability could, in some cases, be the solution of many cases problem, for example, the public blockchain could allow an asset transaction among users without limit and granted security and authentication. On the other hand, consortium solution could allow companies to set up roles over their own network and to keep information data private. Nevertheless, the integration between the two blockchain solutions has several Achilles heels to be evaluated and managed:

- **Synchronization**: both networks must be synchronized and the world state must be the same in each moment. This means that each transaction that involves both blockchains, must reach strong synch among the ledgers, before being validated.
- **Time**: in order for it to be usable, the transactions and synchronization must be performed in a reasonable time.
- **Identity**: each blockchain implementation handles the identity mechanism in a specific way. This means that the user wallet is implemented using specific cryptographic algorithms and solutions. For example, Ethereum handles it as a Key pair, private and public, that allow authentication of the wallet

owner. On the other hand, Hyperledger Fabric implements the authentication mechanism for the user of the network using x.509 certificates. So the other problem is the mapping of these different mechanisms that blockchains implement to allow authentications.

Current solution

The new challenge of cross-chain was born a few years ago and it has brought many companies and research centers to design solutions to fix the problem and allow interoperability. **Figure 2.4** shows the theoretical solution to the interoperability problem between Bitcoin and Ethereum, at each layer of blockchain architecture[10].

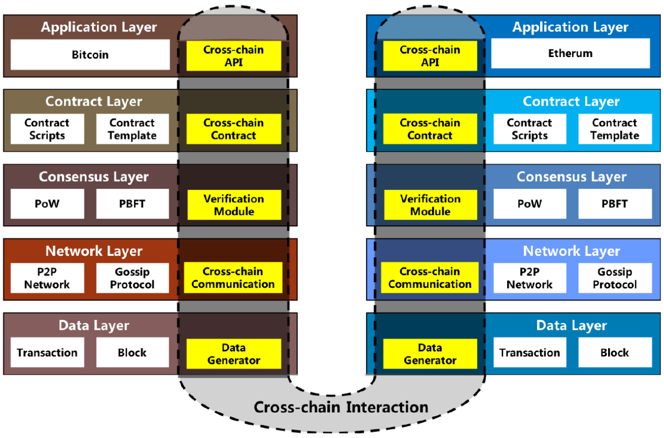


Figure 2.4. CrossChain Interactions

The main ideas to perform interoperability is:

- **New Blockchain**: Over the last few years several networks and frameworks have appeared that propose to allow the interconnection between public and private blockchains. Many of those solutions are based on new blockchain networks that are structured in order to allow, architectural level, the interoperability, for example, Ark[11] is a blockchain-based platform that allows anyone to customize their own blockchain. But the biggest challenges still remains to allow interconnection between the well-known blockchain networks.
- **Architectural Framework**: There are thousands of frameworks proposed over the last few years, but the cross-chain isn't still a consolidated reality. Nevertheless, most of the solutions share the same idea, a Sidechain[12] between the two blockchains. Introducing a new layer between the two mainnet that allows mapping, using ad-hoc API, the requests from one network to the

other one. In a nutshell all the requests from the one to the other blockchain and vice-versa passing by the sidechain.[13] [14]

- **Atomic Swaps**[15]: it allows users to trade one cryptocurrencies for another directly in a peer-to-peer transaction Hashed TimeLock Contracts (HTLCs)[16]. Atomic swaps are not a true form of cross-chain communication (as the two chains do not communicate), but a mechanism that allows two parties to co-ordinate transactions across chains. Atomic swaps can be effective if used correctly and are they are the mechanism that enables the Lightning Network[17].

- **Relay**: it allows a contract to verify block headers and events on another chain. Several approaches to relays exist, ranging from verifying the entire history of a chain to verifying specific headers on-demand. Each method has trade-offs between the cost of operation and the security of the relay. Relays are often quite expensive to operate, as we saw first-hand with BTCRelay[18].

- **Merged Consensus**: it allows for two-way interoperability between chains through the use of a relay chain. Merged consensus can be quite powerful, but generally must be built into the chain from the ground up. Projects like Cosmos[19] and ETH2.0[20] use merged consensus.

- **Federations**: it allows a selected group of trusted parties to confirm the events of one chain on another. While federations are powerful, their obvious limitation lies in the requirement to trust a third party.

- **Chaincode EVM**: In the last year, IBM technical ambassador developed an **EVM chaincode**[21] able to run bytecode of Solidity smart contract over the Hyperledger Fabric network. It is not a real cross-chain solution but it is a step forward interoperability among blockchains. It still has many limits, for example, there is not a real identity mapping mechanism from eth address to fabric identity and vice-versa.

In the thesis work, I focused my attention on Hyperledger and Ethereum, two of the main blockchain solutions used in the world, the former for permissioned cases, the latter one for public processes. For that reason I choose to exploit the *Chaincode EVM* to implement the thesis work solution.

2.3 Blockchain application in Fashion Environment

In the last few years, many Fashion companies have been getting interested in the blockchain for tracking and counterfeiting issues.

2.3.1 Provenance case Martine Jarlgaard

Thanks to the blockchain feature it is possible to store in an immutable way the record associated with each transaction performed over the supply chain. One of the first fashion houses that started to use the blockchain technology for its own company is Martine Jarlgaard that in 2017, the fashion company made a partnership with Provenance[22] producing clothes with digital tag: The tag could be a QRCode or an RFID reader using NFC technology. That tag provides the entire history of the related clothes, providing each step of the producing process.

The actors of the supply chain process are:

- **British Alpaca Fashion Farm**: It cares about alpacas livestock and shearing.
- **Two Rivers Mill**: It cares about wool spinning.
- **Knitster LDN**: It cares about the knitting process.
- **Martine Jarlgaard**: It cares about the design of the clothes and the final work.

Each actor of the supply chain is a blockchain node that takes part in the supply chain pipe through the transactions of the exchanged assets, such as wool, cloth, and so on. Each transaction is registered over the blockchain and visible at each node.

Customer side the user has a clear vision of the entire production process, from the material used to the item produced. It allows the company to gain credibility and transparency of the products sale.

2.3.2 Counterfeiting - VeChain BabyGhoast

BabyGhoast by combining blockchain technology with NFC chips, it creates a digital identity for each cloth produced. It improved the tracking process over the supply chain. Moreover, it allows protecting the brand and the users against counterfeit items. In order to implement the solution it is used VeChain technology, that includes inside BabyGhoast clothes an RFID/NFC chip or QRCode, that allows identification of the item thanks to a unique ID VeChain. Moreover, by scanning the chip or QRCode using the VeChain Pro application, it is possible to access the data related to the item and the production process.

2.4 ASIS model

Armadio Verde is an Italian community that was born to share children's clothes. Once it grew up, it allows adult clothes sharing too. The working model is based on the sharing principle. Every user, after is signing up to the platform, can book a pick up of their old clothes. The clothes must be in a good state, clean and put in a box. Once the box arrives at Armadio Verde, the clothes are going to be checked and evaluated. For each approved clothes, a dedicated form is created with all the related information. After the upcycling process, the clothes are shared over the platform store. The user that sent the clothes earns an amount of "star"(the money used over the platform). The star could be used to purchase other clothes adding a few euros for each item. The clothes that could not be shared on the platform for the reselling process, are sent to a certified Onlus.

2.5 Sustainability Token

PlasticToken

Plastic Token is an ERC20 chaincode that runs over the Hyperledger Fabric network[23]. It provides functionalities to read and write, with access and rights control, into the distributed ledger. The ERC20 chaincode is the software securely handling the PlasticTokens. These tokens are up to the ERC20 standard, meaning a fixed amount of tokens will be minted when the chaincode is deployed. This amount is called “TotalSupply” and will be assigned to a special user, called “central bank” in the current implementation. Once the original PlasticToken[24] supply is minted, users can interact with it via a “transfer” functionality. It allows the central bank to send tokens to any previously enrolled user, then each user can use this same function to transfer tokens between each other

It runs over the Plastic Twist project.

ECOCoin

The ECO coin[25] is a new cryptocurrency that is earned through sustainable action. The ECO coin aims to reward anyone, anywhere in the world carrying out sustainable actions. Eating meat-free meals, switching to a green energy provider or riding a bike to work can earn you ECOs which users could spend in ECO new sustainable marketplace to buy ecological experiences, services and goods.

It is based on consortium blockchain architecture and each marketplace that want to involve their business in ECO environment must be accepted as a governance member of the network.

2.5.1 Solution

As explained before, the Fashion environment has several advantages to exploit transparency and traceability of the blockchain solution. Nevertheless, right now the current solutions are based on consortium blockchains, such as Jarlgaard solution, that allow handling the internal process in a better way, but it is less useful if applied to the end-user. For that reason, in the thesis work, I developed the interoperability between the two networks. The internal processes are still managed by the consortium blockchain, exploiting all the membership's advantages and maintaining transparency. On the other hand parts of the end-user side, are handled by using a public blockchain, in order to detach as much as possible by own case, the token involved, considering the token used as a reference asset and not just for own use. Moreover, there is no concept for the interoperability implementation between Hyperledger Fabric and Ethereum blockchains, as specified there are many architectural differences between the two networks. Therefore, in the thesis work, I proposed an API based solution, at Application layer, that performs the cross-chain between the two networks.

Chapter 3

Solution

3.1 Overview

The goal of the thesis is to develop a software application, blockchain-based, for a Fashion company. The fashion company wants a blockchain solution to handle the internal processes with a tracking mechanism of the clothes over the supply chain. There are many actors involved in the system, therefore, the software has to implement a membership mechanism to handle access over the network. Moreover, for the sustainability issue, the company wants to introduce an asset token exchanged with the clothes, which represents the CO_2 emission saved. Therefore the token must be run over a public blockchain in order to be detached by own use.

Below there is the thesis work overview. It is listed the technologies used for the solution, the work produced and the main processes of the thesis.

3.1.1 Use Cases

Figure 3.1 shows the overview and the main flow of the overall application.

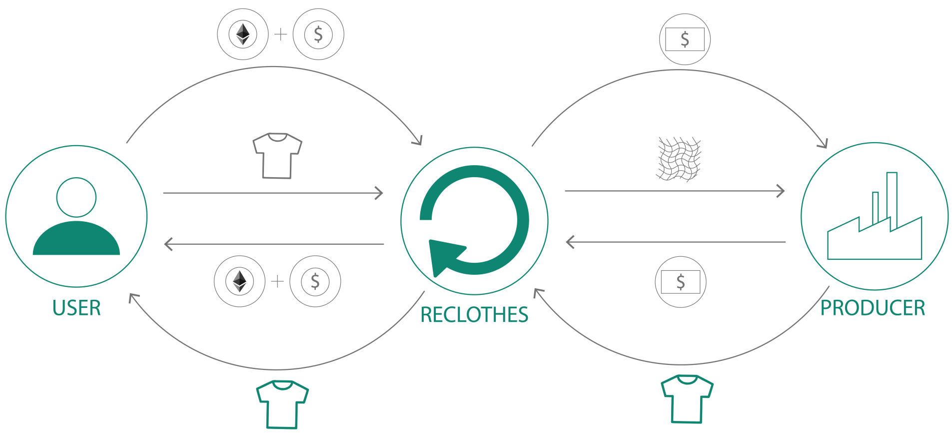


Figure 3.1. UseCase Overview

Actors

The main **actors** involved in the system are three:

- **User**: It is the *end-user*. It uses the web-app to *send old clothes* and *purchase items* from Reclothes store. The User is the actor that starts the entire process flow, sending the clothes. This action is mandatory for the whole process.
- **Reclothes Admin**: It is the *system admin*, it performs the actions in order to handle the system. The Reclothes Admin handles both parts, User Side and Producer Side. About User Side, it performs a set of actions in order to *handle* in the best way *the clothes arrived and the tokens provided*. On the other hand, Producer Side, the Admin cares about to *handling the recycling and upcycling process*, providing the old materials to the Producer and spend, when the platform needs, the token received to order recycled clothes.
- **Producer**: It is part of the upcycling process. It receives the materials to *perform the recycling process*. In the test case, I consider just one Producer, that receives the entire old materials to be recycled. However, the system is developed in order to allow a set of Producers registered. It allows the Reclothes Admin, during the Send Old Clothes process, to choose the Producer toward which ship the material.

Application Flow

Each actor accesses to the system with different permission and privileges. Once logged in, the user can access to several features and he can performs a set of actions over the system. For a better understanding, we are going to split the overview flow shown in **Figure 3.1** into 2 sub-flow starting from Reclothes actor, considered as the *System Admin*, the **User side** on the left side and the **Producer side** on the right side.

Each side has a set of main actions that are going to modify the world state of the blockchains. Based on that principle, the smart contract invocations are going to produce transactions that modify the ledgers in an immutable way, adding a new block to the chains. The main processes are the following:

1. **User Side**
   (a) User sends Box with old clothes and receive Fabric points and ERC20 Token.
   (b) User purchases items inside dapp store using Fabric points and ERC20 Token.
2. **Producer Side**
   (a) Reclothes send clothes box with old materials and receive Regeneration Credits.
   (b) Reclothes spend the Regeneration Credits to purchase upcycled clothes by Producer.

these processes are described in detail in the 3.3 Section, which analyzes deeper the transactions process.

Token exchanged

There are two **Token** categories exchanged over the networks.

Hyperledger Fabric side is exchanges two kinds of tokens, both are point-based, integrated with the smart contracts that handle User and Producer side both. The User points are handled Reclothes side, which means that the Reclothes Admin decides the amount to be sent to the User. In order to handle the amount of the transaction and establish a standard behavior, it needs a reference table that sets a fixed amount for each clothes received. The producer side points, *Regeneration Credits*, are handled by the Producer side, is the Producer that receives the old materials, and then choose the amount to be sent. As the User points, it needs a table to fix rules for the corresponding amounts for the received materials evaluation.

On the other side, there is the ERC20 token that runs over the Ethereum network. ERC20 is a standard protocol that allows everyone to implement its token following fixed rules. That standard includes a set of fixed operations: `totalSupply, balanceOf, transfer, transferFrom, approve and allowance`.

To clarify the tokens exchanged over the thesis work and their behavior, here is a list below:

1. **Over Fabric Network**
   (a) **User Token**: It is a token, points-based, used to handle part of the payment system related to clothes shipping from User to Reclothes and vice-versa.
   (b) **Regeneration Credits**: It is a token, points-based, used to handle the credit system related to clothes shipping from Reclothes to Producers and vice-versa.
2. **Over Ethereum Network**
   (a) **CO2 Token**: It is and ERC20 Token run over public network in charge to handle part of the payments related to clothes shipping from User to Reclothes and vice-versa.

3.1.2 Work overview

Figure 3.2 shows the overall system and the thesis work produced to create the application is composed of the following parts:

- **Networks**: They are the networks over which the blockchains run. The project involves two kinds of networks:
  - **Hyperledger Fabric Network**: the main network. I choose Hyperledger Fabric because it has a highly modular architecture and it implements a well-defined membership mechanism. Fabric is a solid network and provides at the developers a lot of instruments that simplify the implementation process.
  - **Ethereum Network**: the side network used for token exchanged for own use. The Ethereum is a strong reality and widely used to develop solutions applied to the public blockchain. It provides huge tools and instruments that allow the developers to develop contracts easily. In this thesis work is used the Ropsten testnet.

- **Shell Scripts**: to set up everything in the best way, it is produced a set of shell scripts that run network or shut it down, install chaincode, and run part of the system mandatory for the application use.
- **Smart Contracts**: The smart contracts perform project use case actions. Three smart contracts have been developed. Each contract performs one specific flow and includes just a set of the overall actors involved in the system.
- **Dapp**: It is the web application, it allows the actors to interact with the system. It is a decentralized application that communicates with the blockchain networks. Once the user is logged in, with related rights, he can perform smart contracts invocation using the web-app.

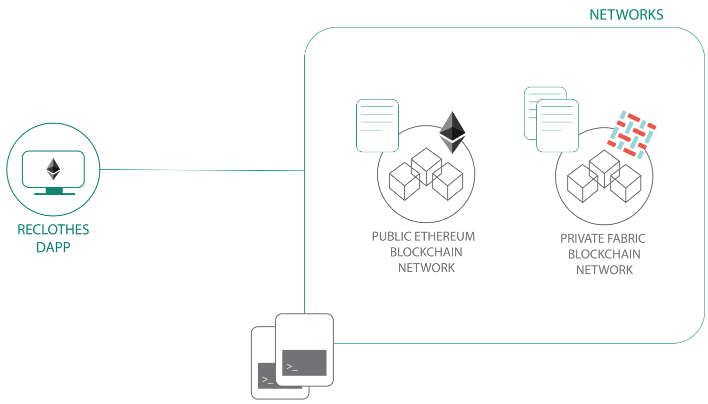


Figure 3.2. Interactions among implemented parts

3.2 CrossChain interaction

3.2.1 Why a cross-chain solution is needed

One of the goals of the thesis work is to implement a good integration between the two blockchain networks involved in the system. The need for a cross-chain solution applied to own cases is to keep the `CO`$_2$ `Token` exchanged public, so that, for the future use, it can be reused in other environments and applications. In that way, the token is not strictly correlated to own personal use, but it could become a standard token to be exchanged over Ethereum, and corresponding to an asset related to CO_2 emissions. The behavior of that token is better analyzed in 3.4.3 section.

The main requirement to obtain a good integration is to perform cross-chain process without compromise the security issue both sides, Fabric and Ethereum. Therefore we need to care about the technologies behaviors and what's the technical basis upon which blockchains works.

Before introducing the chosen solution, it is important to have a look at the technologies used for thesis development.

3.2.2 Technologies Used

Below there are all the main technologies used, involved in the application, and in the cross-chain process. **Figure 3.3** shows how these technologies and tools are used and interact with each other.

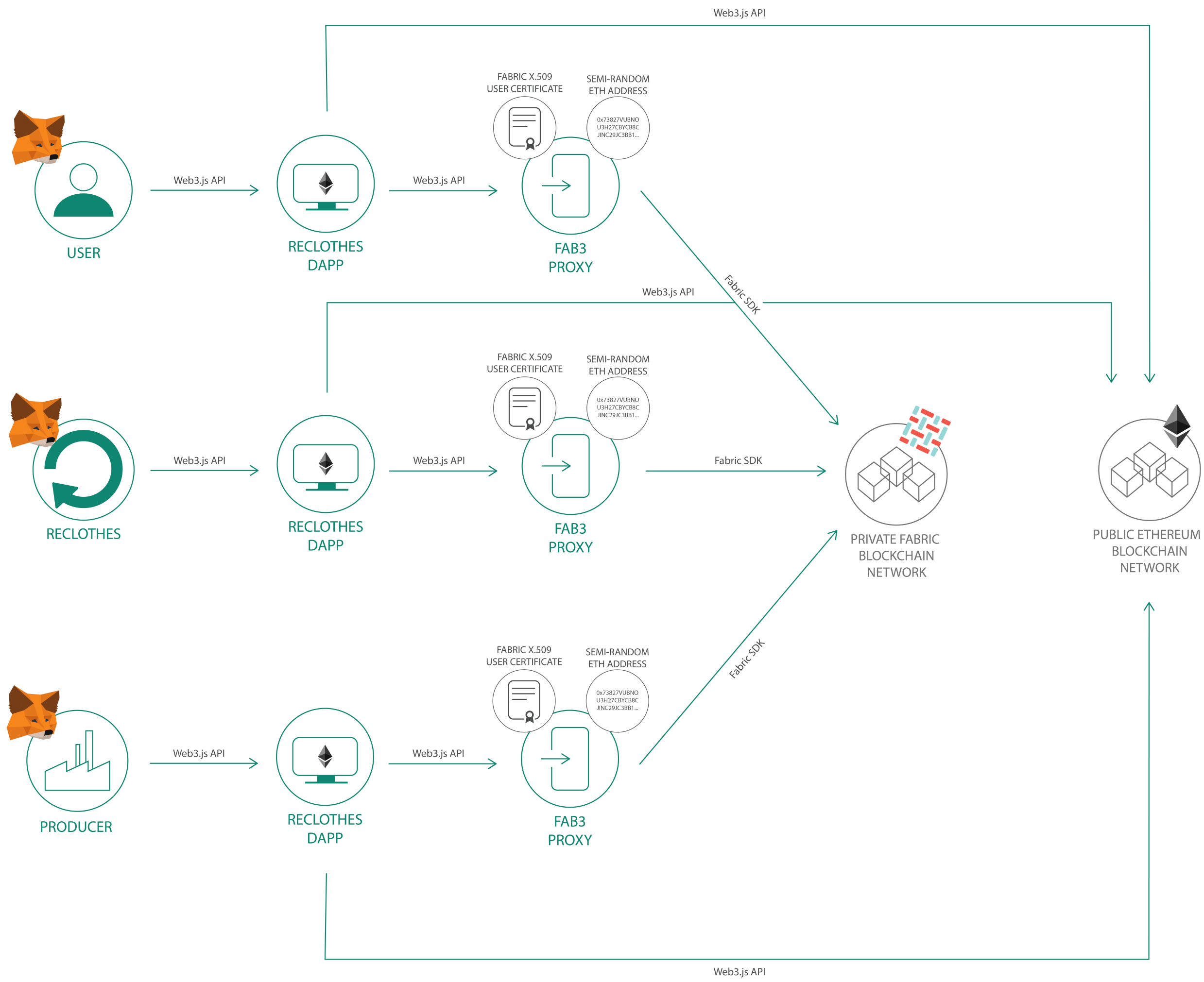


Figure 3.3. Architectural Flow

Below it is listed all the technologies involved in the cross-chain solution.

- **Hyperledger Fabric**: Hyperledger Fabric is a modular blockchain framework that acts as a foundation for developing blockchain-based products, solutions, and applications using plug-and-play components that are aimed for use within private enterprises.

- **Ethereum**: Ethereum is an open-source, blockchain-based, decentralized software platform. It enables to build and run smart contracts and distributed applications (DApps).

- **Metamask**: It is used as Ethereum wallet to perform and sign the transactions started by dapp. Exploiting Metamask API, a high level of security is granted to perform transactions over the Ethereum network. It is integrated into the thesis work Application side; for the right usage of the entire application is mandatory that the User is logged in Metamask over the wallet specified during the registration phase.

- **Web3**: It is the software library used to interact with smart contracts. The Web3.js API fulfills the developers' needs for the integration between website/client and Ethereum blockchain. It is a collection of libraries that allows developers to perform actions like sending Ether from one account to another, read and write data from smart contracts, create smart contracts, and much more.

- **Fab3 Proxy**: It maps the Web3 API with the Fabric SDK in order to interact with Fabric network. It performs a mapping between the Fabric Identity (X.509) with an Ethereum address, generated on the fly, used to perform dapp calls. In other words, it works like a bridge between Ethereum technologies and tools, used for dapp development, and Fabric chaincode, that run over the Fabric peers and use the GO SDK to allow the chaincode invocation.

- **Fabric Chaincode EVM**: It is the Ethereum Virtual Machine chaincode that allows running Solidity smart contracts over the Fabric network. It is a core part of the entire thesis work. Thanks to EVM chaincode it is possible to run solidity bytecode over Fabric peers.

- **Remix**: It is an online editor that allows developing well-structured Solidity smart contracts. Thanks to the plugins, that could be installed over the editor, it is possible to compile the written smart contracts code. Once the compiling process succeeds, it produces the corresponding smart contract's bytecode and the smart contract's ABI. Both bytecode and ABI are used to define smart contract behaviors. These parameters are passed as an argument during the deployment process.

- **Expressjs**: It is a web framework used to develop web-app and smart contract API. It's a light, easy, and fast framework that integrates several methods useful for HTTP and middleware API development.

- **Infura**: It allows running an Ethereum node, to set an endpoint used to interact with the contract. It allows in an easy way to set up a public endpoint for the deployed contract address. It provides personal API and key for the endpoint access. Moreover, it provides a well defined and detailed dashboard to analyze all the smart contract invocations, providing deeper analyses for the called method too.

- **Docker**: The Fabric network components run inside Docker containers. It is mandatory for Fabric network blockchains, each peer(node) of the network run inside a specific and dedicated container. It allows being monitored and analyzed independently.

Thanks to the introduction of the EVM chaincode developed by the IBM technical ambassadors, it is possible to run Solidity bytecode over the Fabric network. It allows the possibility to joint Ethereum technologies over Hyperledger Fabric Network. That innovation doesn't improve only the integration network side but developing side too. With the `fabric-chaincode-evm` a new communication way from dapp/client side is open to the network side. For example, that integration allows to use Web3.js library to invoke smart contracts running over Fabric. Moreover, most of the improvements done in the Ethereum environment can be used to interact with the Hyperledger Fabric world. It means languages, API, libraries, and tools that are finding a huge application in the Ethereum world, so far.

The cross-chain solution chosen involves the Application Layer. The core idea of the solution is to map, at the Application level, the Ethereum wallet with Hyperledger Fabric identity. Once there is a one-to-one association, is used the eth wallet for transactions over Ethereum network and the related Fabric identity over Fabric network. Exploiting Web3.js API we invoke Ethereum or Fabric smart contracts, by using this solution all the invocation processes are forward, to the corresponding network, starting from Dapp API.

The authentication mechanism doesn't change and the security continues to be ensured, Fabric side, using certificate x.509. Once the user is authenticated and recognized by the x.509 certificate, Fabric network logged the user into the platform and give him the access to the data information and all the related privileges based on the actor role.

On the other hand, Ethereum side, the user Ethereum public address is specified during the registration phase and saved over Fabric chaincode to the corresponding User data structure. When the user is involved inside transaction processes, all the transactions refer to the public Ethereum address reported during the registration phase.

Therefore when there is an incoming transaction the tokens will be sent to the public address reported in User Data info. When an outgoing transaction occurs, the security is granted thanks to the Metamask integration in the transaction process. The transaction's sign, that allows performing operation, is performed by Metamask side, in that way only the real owner of the Ethereum account could sign and approve the transaction. The private key is stored over Metamask wallet and just the real owner, that is logged in to the account, can perform the sign of the transaction.

Furthermore, the dapp client uses Fab3 Proxy to map the identity from Ethereum address to Fabric identity x.509 certificate and forward request to Fabric network. That process is independent by the Ethereum address specified during the registration phase and does not interact with that. Fab3 allows to use `fabric-chaincode-evm` and run solidity code over Fabric network, It performs a mapping process among the received requests dapp side. Fab3 receives the Web3 request and map it using the GO SDK to forward, in the right way, all the request to the Fabric peers.

Moreover, the Dapp client talks with Ethereum public blockchain network, using the network endpoint API supplied by Infura. For some kind of actions performed over the platform, part of the request is forward over Ethereum network.

Figure 3.4 shows where the private keys are stored. The Hyperledger wallet could be stored in many ways, into a FileSystem, in memory, using a HSM or a Database[26]. For the thesis work, I consider the device filesystem storage, containing the Hyperledger wallet with the certificate and both private and public keys. Always on the device used, it is stored the private key of the Ethereum wallet, in fact, Metamask store on the device the private key of the associated Ethereum account[27].

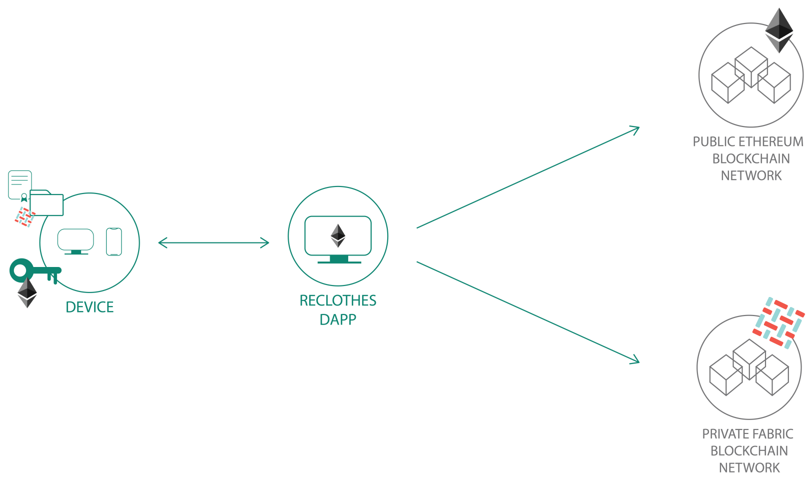


Figure 3.4. Where keys are stored

3.3 Use Cases

As explained in the previous Sections, for a better outline, the use cases are splint inside the **User Side** and the **Producer Side**. The main actor of the system is still Reclothes Admin, that is linked to both side and interacts with all the other actors in order to supply the management supports that allow the entire system works.

3.3.1 UseCase 1 - User Side

As shown in **Figure 3.5** both actors User and Reclothes Admin, once logged in, have access to a set of features. The use case diagram shows all the actions that both users can perform over the networks and the flows that each action follows. The features are split over the two networks, the Fabric one and the Ethereum one. All the flows start from one of the two actors involved and in the end it merges to one of the two blockchain networks. Each actor has a dedicated Ethereum wallet used for Ethereum token transactions.

Below is listed and analyzed all the actions that users could perform over the system:

- **Actions in common**
  - **Registration**: The registration phase involves the actor that fills a form with all the mandatory data. To proceed to a successful registration process it is mandatory that the actor owns the appropriate x.509 certificate, released by the Certification Authority related to the Role in which the user tries to sign up. For example, User has a specific Certification Authority that is different from Reclothes CA.
  - **Sign In**: The sign-in is automatic. Once the Fabric network recognizes the certificate, it proceeds to log the actor in, with the related rights. Once the user is logged in, the chaincode is invoked and going to read the Ethereum address(generated by Fab3) used for the registration phase and provide access to the methods. In other words, Fabric certificate provides access to the network (peers, channels, and ledger), instead, the Ethereum address(Fab3 side) is used to provide access to the smart contract.
- **User Operations**
  - **Read Operations**

    * **View own transactions**: once logged in, the User can view all the own transactions processed by the network, with a flag that shows transaction status. The transactions include token exchanged over the network and box requests sent to Reclothes. It gave the possibility to monitor and manage each process in which the User is involved.

  - **Write Operations**

    * **Send Box**: It is the starting point of the overall application flow. In the following Subsection, I'm going deeper in order to explain how that process works and what transactions depend on that.
    * **Purchase Items**: It is a write operation, belongs to that start a transaction process. Both networks are involved in that process. Even that is explained deeper in the following Subsection.

- **Reclothes Admin Operations**

  - **Read Operations**

    * **View all transactions**: once is logged in, the Reclothes Admin can view all the transactions processed by the network related to all the users involved, with a flag that shows transaction status. The transactions include a token exchanged over the network. It gave the possibility to monitor and manage each process in which there is a token transactions for analysis aim.
    * **View All Box Requests**: The Admin is allowed to analyze the process of the box shipping. The box data structure includes all the relevant data. Moreover, it includes a flag that specifies the status of the request, that flag could be `Pending, Evaluated`.

  - **Write Operations**

    * **Evaluate Box**: Even this process belongs to write operations because it starts a transaction process that writes the blockchain world state.

The main action of the overall system is the send box operation performed by the User towards Reclothes. It is the starting point of the overall flow. The Internal Flow of the **Send Box** macro process, and what that process belongs to, is the following one:

1. User send box with old clothes

2. Reclothes Admin receive box, evaluate it

3. The web app performs the payments from Reclothes Account to User Account

4. Once both transactions succeed, both tokens are accredited and User could spend it

Transactions

In the first use case both the blockchain networks are involved in. The main part of the flow and the most critical one is the transaction process. Considering always *Reclothes Admin* the main actor of the system, there are two kinds of transactions in which Admin is involved. The `outgoing` transaction, that starting by ***`Evaluation`*** process, it is performed by the Reclothes Admin once it receives the clothes box sent by the User. The other one is the `incoming` transaction, in that case, the token is exchanged from the User to Reclothes Admin. The action that starts the incoming transaction process is the ***`Purchase Item`***, performed by the Users over the platform store.

The outgoing and incoming transactions are strictly correlated due to the token flow. As explained in the previous Section the main and the first one action is the ***`Send Box`***, which involves the `Evaluation`. The Evaluation is the first outgoing transaction process performed over the system. Once the tokens are moved from the Reclothes Admin, the User is allowed to use applications and purchase items over it.

1. **Figure 3.6** shows the **Evaluation** process, which works in the following way:

   (a) Reclothes Admin visualizes the next pending request to be evaluated. The Admin visualizes all the related information associated to the box request: `userAddress` it's the Ethereum user address of the sender, `t-shirt, pants, jacket, other` with the related number of items associated to the request, and the status of the request, at this point still `In Pending`.

   (b) Reclothes Admin evaluates it. For a better evaluation process, it is proposed a solution based on a reference table with a fixed amount for each item, related to the clothes status. Then there is a filtering process. Each item inside the box is filtered based on platform criteria. Then the Admin decides the status of the clothes and its final destination, which may be the platform store or recycling materials. Once the overall clothes are evaluated and are set a total amount value of Fabric points and ERC20 Token is set, the transaction process can start.

i. The Fabric points are sent over Fabric network invoking the chaincode function `sendPoints(address toAddress)`. That function accredited the specified amount of Fabric points, updating the User balance.

ii. The ERC20 token is sent over the Ethereum network. During the thesis development, I have used the Ropsten testnet to exchange the token. There is a previous step before performing the transaction of the tokens. The Fabric chaincode is invoked to obtain the Ethereum wallet address related to the sender box User. Once that the Fabric chaincode returns the Ethereum account, stored in the smart contract during the User Registration Phase, the application performs the transfer of the ERC20 token from the Reclothes wallet to the User Ethereum wallet

(c) Once both transactions succeed, both the transaction return to the application and is performed an additional check in order to synchronize both transactions. Tokens are accredited and information about balances are updated. From that moment the User can spend the received tokens over the platform store, performing purchasing.

2. **Figure 3.7** shows the **Purchase Items** process, which works in a similar way, but inverting the previous flow:

   (a) The User chooses the items to purchase over the web-app store. The items (t-shirt, pants, jacket, or other) are represented with the related form, which shows all the relevant information. Over the chaincode the smart contract store a dedicated data structure for clothes data information. The related price is expressed through tokens, Fabric token and CO_2 token both.

   (b) Once the items are chosen, the purchase process starts. The User sends the Fabric tokens over Fabric network and the CO_2 token over the Ethereum network. First of all, a set of controls is executed in order to check both balances and evaluate whether the User could perform the purchase transaction. Once that all the check is passed correctly, both transactions start. Each one over the dedicated network. Once the transfer process is performed, the smart contracts return the operation results to the dapp, that communicate the results of the operations through a message .

   (c) If the transfers succeed, both token balances are updated and the User can continue to perform actions over the platform.

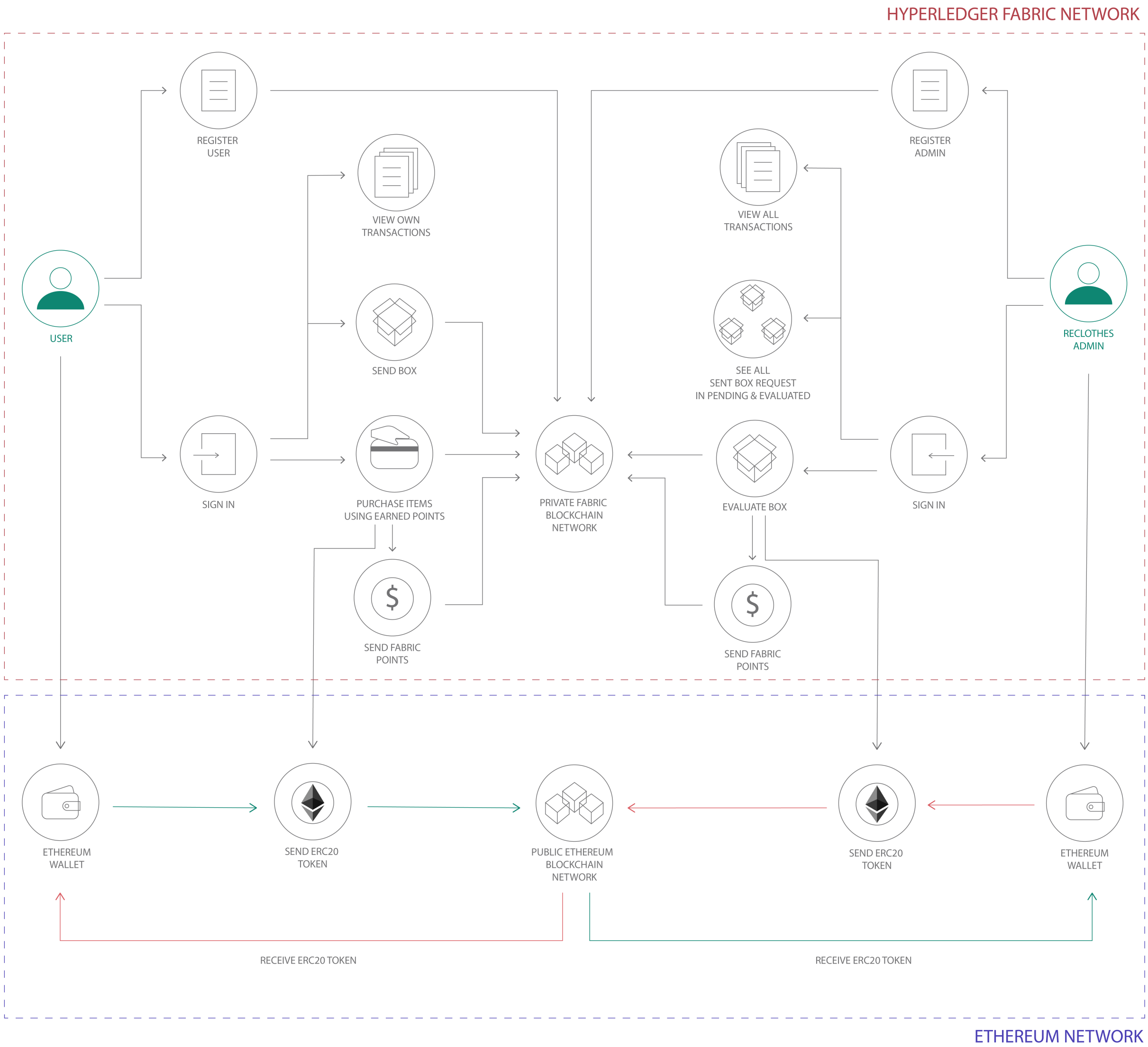


Figure 3.5. UseCase 1

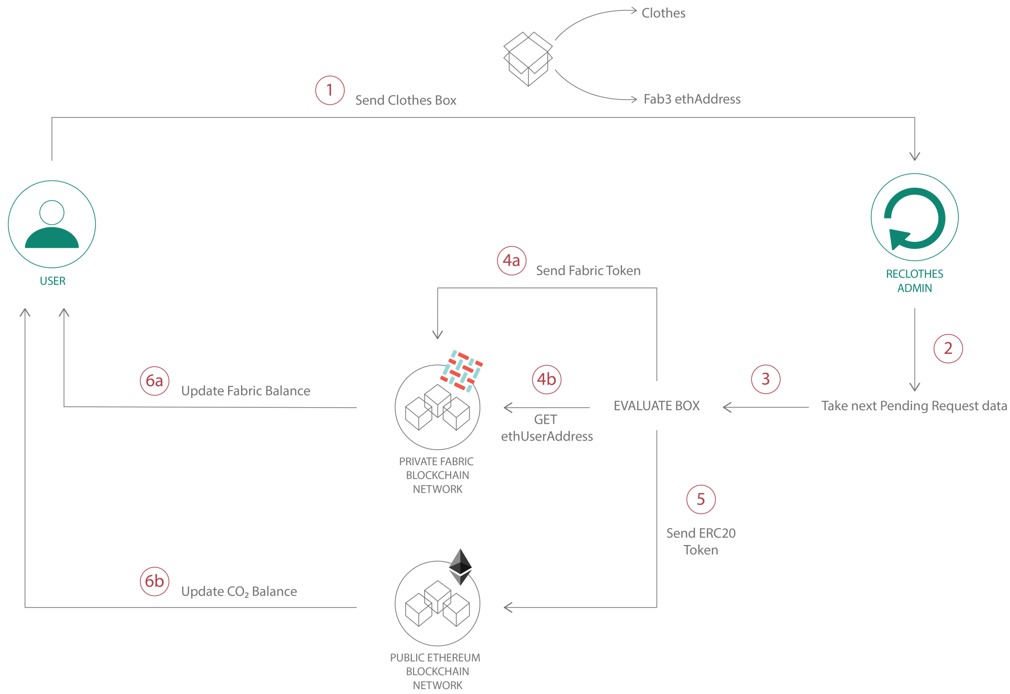


Figure 3.6. Evaluation Process

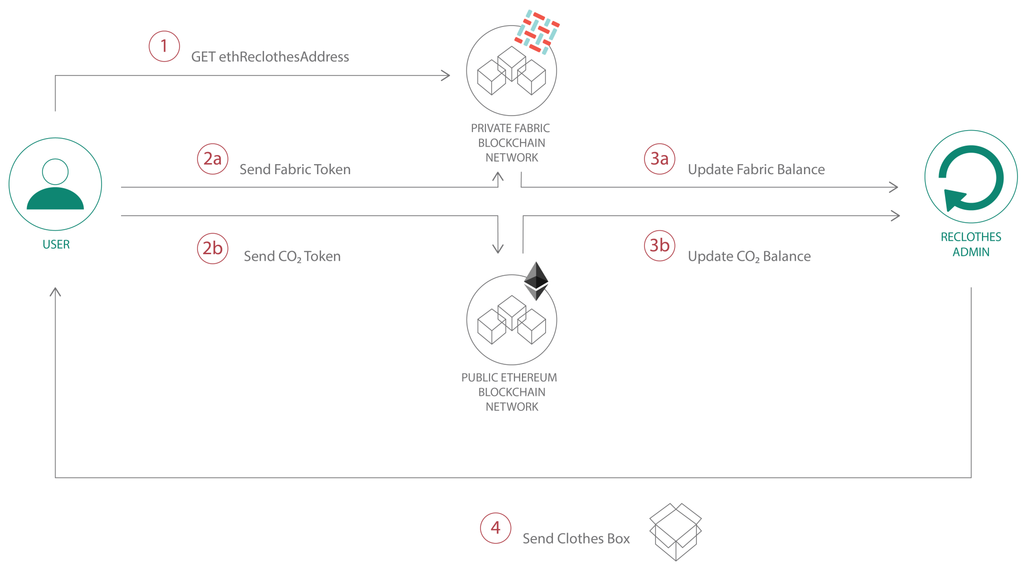


Figure 3.7. Purchase Process

3.3.2 UseCase 2 - Producer Side

Use case 2 is related to the right side of the overall flow schema shown in Figure 3.1. It shapes the interactions between Reclothes Admin and Producers. For a better understanding of the process involved in that interaction, the use case 2 diagram is shown in **Figure 3.8**. In that case, all the features are performed over the Hyperledger Fabric network, so there is not a cross-chain part. The token exchanged, `Regeneration Credit`, is based over Fabric smart contract and it is point-based, without the need to involve the Ethereum blockchain.

Analyzing Figure 3.8, even here, there is the main action that leads to a transaction process. Looking at the diagram we could split the flow into two sub-flow, the first one from Reclothes to Producer. In that sub-flow we could identify two main actions ***Send Box*** and ***Purchase Box***. As specified in the previous use case, these is the operation that performs a world state update of the blockchain ledger. On the other hand, in the second sub-flow, from Producer to Reclothes, just one action that produces an outgoing token transfer is involved, the ***Evaluate Material*** function.

All the assets exchanged are handled using **Regeneration Credits**. It is a Fabric token exchanged and handled by the Fabric chaincode, it runs over Fabric network. To test the use case I consider just one Producer that performs the overall recycling process, even if the smart contract is structured in order to allow the handling and management of more Producer actors involved in the system. In the case of many Producers involved in the recycling process, an ERC20 integration to handle the Regeneration Credits exchanged can be an improvement. Moreover, that change will not have a strict correlation between credits and Reclothes. It allows the Producers to use the token to handle the internal process with more clients.

1. **from Reclothes to Producer**

   (a) **Send Box**

      i. The Reclothes Admin after has performed the filtering process over the clothes box received by the Users. Then all the clothes in a bad status, that could not be resold inside the platform store, are sent to the Producer in order to recycle the material and produce upcycled clothes. The Admin performs the Send Box operation, like the Send Box performed in the use case 1, it contains the same data inside the request (`t-shirt, pants, jacket, other` with the related number of items). The box is sent to the Producer Company. In the case of more Producers, the send box request includes the selected Producer Company chosen.

ii. Once the Producer performed the `Evaluation` process over the sent clothes box, The Reclothes Account gain the corresponding amount of **Regeneration Credits** based on the old materials evaluation. Once that the balance is updated, the Reclothes Admin can spend that credits to purchase items.

(b) **Purchase Box**

i. Reclothes Admin can purchase boxes by the Producer Company with inside clothes realized with recycled materials. At the moment the application allows to purchase three boxes options:

A. *Small Box: 5 items for 50 Regeneration Credits.*

B. *Medium Box: 15 items for 150 Regeneration Credits.*

C. *Big Box: 40 items for 200 Regeneration Credits.*

ii. Once that the Reclothes Admin chooses the box size to purchase, the transaction process starts and the chaincode is invoked. Before is performed a previous check, to control if the Reclothes's wallet, containing the Regeneration Credits, is enough. Then is invoked the transfer method of the smart contract, the Regeneration Credits are redeemed to the Producer account and the shipping of the Box start, with inside the recycled clothes.

2. **from Producer to Reclothes**

(a) **Evaluate Material**

i. Once the box sent by the Reclothes Admin arrives, it must be evaluated. The Evaluation process consists to evaluate all the materials related to the clothes received. To obtain a standard evaluation there is a reference table listing a fixed amount of Regeneration Credits provided for each clothes based on *material* and *weight*. Once the Producer Admin performed the evaluation of the materials for each clothes, and the total amount of Regeneration Credit is fixed, the transaction process starts. The chaincode is invoked and the transaction is performed from Producer account to Reclothes account over Fabric network. Producer side, there are two parameters to analyze:

A. **Regeneration Credits Supplied**: It is the total amount of credits emitted over the time.

B. **Regeneration Credits Circulating**: It is the amount of credits that Reclothes Admin owns and could spend for purchasing.

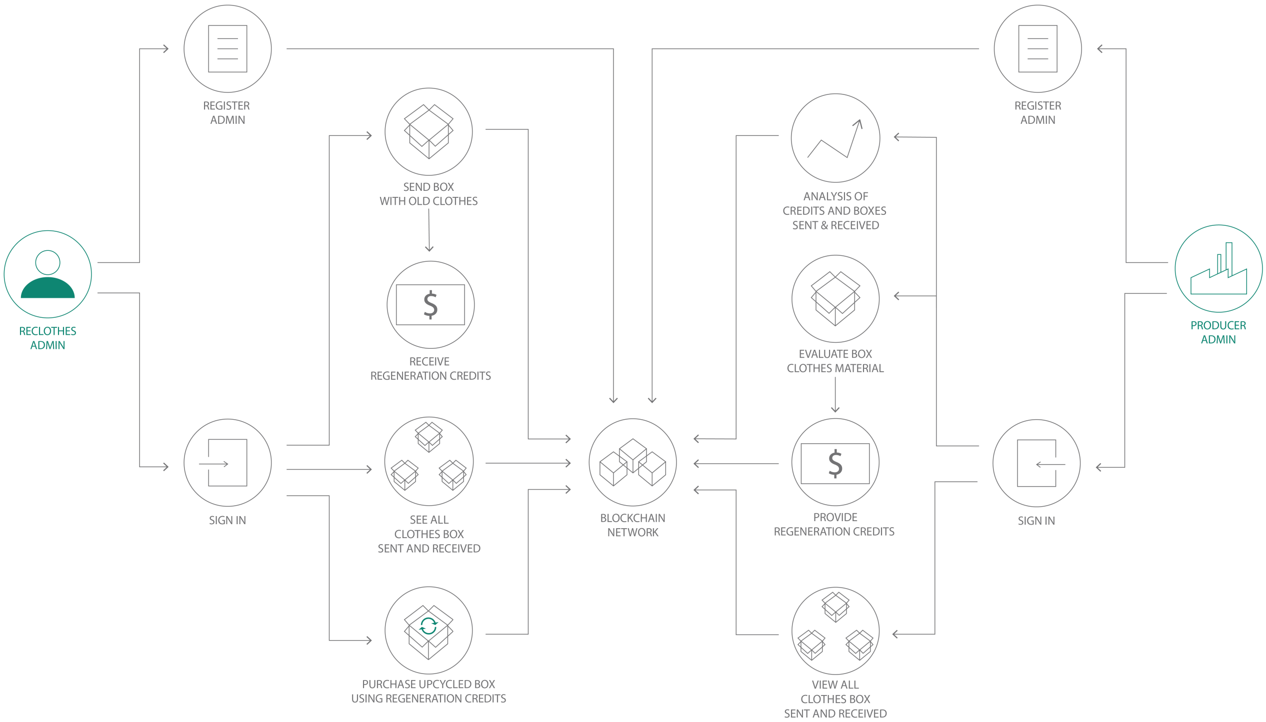


Figure 3.8. UseCase 2

3.4 Smart Contract

For the smart contract developments I exploited the `fabric-chaincode-evm`[1], it allows us to run Ethereum smart contract bytecode inside an Hyperledger Fabric peer. Therefore EVM chaincode bring us to the development of the smart contract in Solidity or Vyper programming languages.

For the development it is used **Remix**[28], it is an online editor that allows us to write and compile Solidity smart contracts code, providing all the Solidity version compiler. Once that the smart contract code is written and the `.sol` file is produced, the compiling process produces two files mandatory for the deployment and uses of the smart contract over Fabric network. The two file produced are:

- **ABI**: The *Application Binary Interface* is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and

[1] To run Solidity Contract over Fabric Network, it's used `fabric-chaincode-evm`[21], it is an Ethereum virtual machine chaincode developed by IBM developers. To allows the integrations there is the need for additional components such as Fab3 Proxy

for contract-to-contract interaction. The ABI is a .json file that describes the deployed contract and its functions. It allows us to contextualize the contract and call its functions. In other words, the ABI is the description of the contract interface. It contains no code and cannot be run by itself. It is mandatory for smart contract use because the bytecode is the executable EVM code, but by itself, it is without context.

- **Bytecode**: This is the code that is stored on-chain that describes a smart contract. This code does not include the constructor logic or constructor parameters of a contract. It is a hexadecimal representation of the final contract. It uses the ABI to find the context of the behind contract logic.

In order to handle the overall system, three Solidity smart contracts have been developed. Two of them run over the Hyperledger Fabric network, exploiting the membership mechanism to access of the chaincode. In other words, the permission mechanism behind the logic is performed in both networks and chaincode side. The network side filters, at the certificate layer, the access to the network. On the other hand, the registration mechanism implemented over the chaincode filter the user logged to the network.

The third smart contract has been developed to run over the Ethereum network. For my thesis work it is used the Ropsten testnet. The access to the contract, in that case, is provided by the contract address generated during the deployment phase.

For a better view, below are reported all the contracts involved in the system:

1. **Hyperledger Fabric**
   (a) **User Contract**: It handles the User side, registration, and interaction phase. That contract shapes the use case 1 functionalities. There are the dedicated data structures that care about storing data of actors involved(`User` and `Admin`). It provides a set of getter and setter methods and it has a couple of functions that lead to a transaction process.
   (b) **Producer Contract**: It handles the interaction from Reclothes to Producers. That contract shapes the use case 2 behaviors. There are the dedicated data structures that care about storing data of actors involved(`Admin` and `Producer`). It provides a set of getter and setter methods and it includes a couple of function that leads to a transaction process.
2. **Ethereum**

(a) **ERC20 Contract**: It is a standard smart contract with a max Supply fixed to 1,000,000,000. The contract is structured following the ERC20 standard. It is not strictly correlated to the thesis application and it is exchangeable among each user that owns an Ethereum wallet. The Contract is deployed over the Ropsten network and is accessible using the public contract address. The access to the contract is performed using an Infura node as Ethereum network endpoint.

3.4.1 User Contract

The User Contract contains all the features described in the Hyperledger Fabric side of use case 1.

Data Structure

In that contract all the transaction data, related to the points and clothes box transactions, are stored. Moreover, the information related to the actors involved in the system, are stored too. Thanks to the registration phase, the contract can perform an additional check over the actors that are logged in over the Fabric network. The address specified during the registration phase (`msg.sender`) is the Ethereum address generated on the fly by Fab3 Proxy. In the following Sections, there is a better and deeper explanation of the Fab proxy module and how it works.

The model of the data structures is divided into 4 structs:

1. **User**: model all users data.

```

        // model a user
            struct User {
                address userAddress;    // User address (inside fabric environment)
                address publicAddress;  // external eth public address of User Admin
                string firstName;
                string lastName;
                string email;
                uint points;            // Fabric points amount
                bool isRegistered;      // Flag for internal use
                uint numTransaction;    // number of transactions performed
```

```
            mapping(uint => PointsTransaction) userTransactions;
            uint numBox;            // number of box transaction evaluated
            mapping(uint => ClothesBox) box;
        }
```

2. **Admin**: model Reclothes Admin data.

```
        // model a admin
        struct Admin {
            address adminAddress;   // Admin address (inside fabric environment)
            address publicAddress;  // external eth public address of Admin
            string name;
            bool isRegistered;      // Flag for internal use
        }
```

3. **PointsTransaction**: Model transactions data and incorporate `TransactionType`, it is used to identify the flows direction.

```
        // model points transaction
        enum TransactionType { Earned, Redeemed }
        struct PointsTransaction {
            uint points;
            TransactionType transactionType;
            address userAddress;    // user address involved
            address adminAddress;   // admin address involved
        }
```

4. **ClothesBox**: The box sent with old clothes.

```
        // model clothes box to ship
        struct ClothesBox {
            address userAddress; // reclothes-producer Admin
            uint tshirt;         // Number of item
            uint pants;          // Number of item
            uint jacket;         // Number of item
            uint other;          // Number of item
            bool isEvaluated;    // Flag to check if box evaluation is performed
            uint points;         // fabric value amount of the box
        }
```

Getter

The User Contract allows access to a set of methods, to obtain information about the system status. Once the user is logged in as User or Admin, it could perform part of that getter invocation. Parts of the method are developed for internal usage, the other ones are dedicated to providing to the actor's information about the system, or it is useful to start other invocations dapp side. Access to some method is handled using the modifier method that performs a filtering process of the function caller.

The *Getter* methods involve the *User* data and the *Box requests*. Below it is explained just the main getters related to Box requests.

1. **Pending Box**: it returns the pending Box at index `_pendingIndex`. For example, the method `getNextPendingRequest` calls the `getPendingRequest`, passing as argument the `pendingIndex` variable stored in the contract.

```
        //Get PendingBox by index
        function getPendingRequest(uint _pendingIndex) public
           view returns(address, uint, uint, uint, uint, bool,
            uint) {
            // only admin can call
            require(admins[msg.sender].isRegistered, "Admin
               address not found");

            //check index
            require(_pendingIndex<pendingIndex, "Wrong index")
               ;

            return (pendingBox[_pendingIndex].userAddress,
               pendingBox[_pendingIndex].tshirt, pendingBox[
               _pendingIndex].pants, pendingBox[_pendingIndex
               ].jacket, pendingBox[_pendingIndex].other,
               pendingBox[_pendingIndex].isEvaluated,
               pendingBox[_pendingIndex].points);
        }
```

2. **Evaluated Box**: it returns the evaluated Box at index `_evaluatedIndex`. It is useful to get a list of all evaluated box. The `evaluatedIndex` variable stores the number of boxes evaluated.

```
        //Get EvaluatedBox by index
        function getEvaluatedRequest(uint _evaluatedIndex)
           public view returns(address, uint, uint, uint, uint
           , bool, uint) {
            // only admin can call
            require(admins[msg.sender].isRegistered, "Admin
               address not found");

            //check index
            require(_evaluatedIndex<evaluatedIndex, "Wrong
               index");
```

```

            return (evaluatedBox[_evaluatedIndex].userAddress,
                 evaluatedBox[_evaluatedIndex].tshirt,
                evaluatedBox[_evaluatedIndex].pants,
                evaluatedBox[_evaluatedIndex].jacket,
                evaluatedBox[_evaluatedIndex].other,
                evaluatedBox[_evaluatedIndex].isEvaluated,
                evaluatedBox[_evaluatedIndex].points);
        }
```

3. **Transaction Info**: it returns the Transaction info at index `_transactionIndex`. It is called just by the User. The `users[msg.sender].numTransaction` variable stores the number of the transactions done.

```
        function getTransactionInfo(uint _transactionIndex)
            onlyUser(msg.sender) public view returns(uint, uint
            , address, address) {
            //require index exists
            require(users[msg.sender].numTransaction >
                _transactionIndex && _transactionIndex >= 0, "
                Wrong transaction index");

            return (users[msg.sender].userTransactions[
                _transactionIndex].points, uint(users[msg.
                sender].userTransactions[_transactionIndex].
                transactionType), users[msg.sender].
                userTransactions[_transactionIndex].userAddress
                , users[msg.sender].userTransactions[
                _transactionIndex].adminAddress);
        }
```

Transactions

The **sendBox** method is a core part of the overall smart contract, below there is the function's code, it allows to perform the transaction processes.

```
    //handle box
    function sendBox(uint _tshirt, uint _pants, uint _jackets, uint
        _other) onlyUser(msg.sender) public {

        pendingBox[pendingIndex] = ClothesBox({
            userAddress: msg.sender,
            tshirt: _tshirt,
            pants: _pants,
            jacket: _jackets,
            other: _other,
            isEvaluated: false,
            points: 0
        });
```

```

            users[msg.sender].box[users[msg.sender].numBox] =
                pendingBox[pendingIndex];

            users[msg.sender].numBox++;
            pendingIndex++;
    }
```

The transactions process is the main process of the overall smart contract. This method performs a write access to the smart contract and modifies the world state of the ledger stored over the Fabric blockchains peers.

There are two functions that perform transactions between actors involved in the smart contracts, these are :

1. **earnPoints**: It is an internal function called by `EvaluateBox`. Once that user performed the `sendBox` process, Admin side, starts the evaluation process. Therefore the Admin evaluates the pending request and sets a total amount of points related to the box received. Then the `EvaluateBox` function call the internal function `earnPoints` passing as argument the amount to be transfer and the userAddress of the clothes box sender. Then the function performs the Fabric points transaction from Reclothes to User.

```
        //evaluate box
        function evaluateBox(uint _points) onlyAdmin(msg.
            sender) public {
            //check correct pending request index
            require(evaluatedIndex < pendingIndex, "No more
                pending request");

            //check if evaluation is done
            require(!pendingBox[evaluatedIndex].isEvaluated, "
                Request just evaluated");

            //pop pending request
            ClothesBox storage box = pendingBox[evaluatedIndex
                ];

            //update box transaction
            box.isEvaluated = true;
            box.points = _points;

            //send points to the userAddress
            earnPoints(_points, box.userAddress);

            //add evaluated box
            evaluatedBox[evaluatedIndex] = box;
            evaluatedIndex++;
```

```
        }

        //get user balance
        function getBalance() public view returns (uint) {
            return users[msg.sender].points;
        }

        //update users with points earned
        function earnPoints (uint _points, address
            _userAddress ) onlyAdmin(msg.sender) internal {

          // verify user address
          require(users[_userAddress].isRegistered, "User
              address not found");

          // update user account
          users[_userAddress].points = users[_userAddress].
              points + _points;

          PointsTransaction memory earnTx = PointsTransaction
              ({
            points: _points,
            transactionType: TransactionType.Earned,
            userAddress: _userAddress,
            adminAddress: admins[msg.sender].adminAddress
          });

          // add transaction
          transactionsInfo.push(earnTx);

          users[_userAddress].userTransactions[users[
              _userAddress].numTransaction] = earnTx;
          users[_userAddress].numTransaction++;

          usersTransactions[totTx] = earnTx;
          totTx++;

        }
```

2. **usePoints**: It is related to the purchase process. When the User performs a purchase over the platform store, there is the calculation of the overall amount related to the items purchased, and internally is invoked the `usePoints` function. That function after a set of previous checks, then decrease the User balance of the related amount passed to the function.

```

        //Update users with points used
        function usePoints (uint _points) onlyUser(msg.sender)
            public {

```

```
        // verify enough points for user
        require(users[msg.sender].points >= _points, "
            Insufficient points");

        // update user account
        users[msg.sender].points = users[msg.sender].points
            - _points;

        PointsTransaction memory spendTx = PointsTransaction
            ({
          points: _points,
          transactionType: TransactionType.Redeemed,
          userAddress: users[msg.sender].userAddress,
          adminAddress: 0
        });

        // add transaction
        transactionsInfo.push(spendTx);

        users[msg.sender].userTransactions[users[msg.sender
            ].numTransaction] = spendTx;
        users[msg.sender].numTransaction++;

        usersTransactions[totTx] = spendTx;
        totTx++;
    }
```

3.4.2 Producer Contract

The Producer Contract contains all the features described in the use case 2 diagram shown in the Figure 3.8.

Data Structure

In that contract, all the transaction data related to the points and clothes box transactions, are stored. Moreover, the information related to the actors involved in the system are stored; in this case, the actors involved are the `Admin` and the `Producer`. As the previous contract, the registration phase provides an additional check over the actors logged in over the Fabric network. The address specified during the registration phase (`msg.sender`) is always the Ethereum address generated on the fly by Fab3 Proxy.

Briefly explaining the behavior of the relationship among contracts. The Fab3 proxy has a 1 to 1 association instance/user. There is the possibility that the Admin logged and registered, over `UserContract`, associated with one Fab3 instance, set over the channel that communicates with `UserContract`, must perform another registration with a new Fab3 proxy instance, in order to set the communication with the channel dedicated for `ProducerContract`. It means that for each Fab3 instance there is a new Eth address generated and the Admin could have two Ethereum addresses, one associated with `UserContract` and the other one associated with `PrducerContract`.

The model of the data structures is divided into 3 structs:

1. **Producer**: model all Producers data.

```
        // model a producer
        struct Producer {
            address adminAddress;   // Producer Admin address (inside fabric environment)
            address publicAddress;  // external eth public address of Producer Admin
            string name;            // Producer admin name
            bool isRegistered;      // Flag for internal use
            uint numBox;            // number of box transactions evaluated
            uint pointsProvided;    // amount of points provided by own evaluations
            mapping(uint => ClothesBox) box;
        }
```

2. **Admin**: model Reclothes Admin data.

```
        // model a admin
        struct Admin {
            address adminAddress;   // Admin address (inside fabric environment)
            address publicAddress;  // external eth public address of Admin
            string name;            // Admin name
            bool isRegistered;      // Flag for internal use
            uint numBox;            // number of box transaction evaluated
            uint creditSpent;       // amount of points provided by own evaluations
            mapping(uint => ClothesBox) box;
        }
```

3. **ClothesBox**: The box sent with old clothes.

```
        struct ClothesBox {
            address adminAddress; // reclothes-producer Admin
            uint tshirt;          // Number of item
            uint pants;           // Number of item
```

```
            uint jacket;         // Number of item
            uint other;          // Number of item
            bool isEvaluated;    // Flag to check if box
                evaluation is performed
            uint points;         // fabric value amount of the
                box

            //mapping(uint => Clothes) clothes;
        }
```

Getter

The Producer Contract allows access to a set of method to get information about the system status. Once the user is logged in as Admin or Producer, he can performs part of that getter invocation. Parts of the method are developed for internal usage, the other ones are dedicated to provide to the actor's information about the system, or it is useful to start other invocations dapp side. Access to some method is handled using the modifier method that performs a filtering process of the function caller.

Below I reported only the main smart contract methods.

```

    /**************************************************/
    /*** All Box Requests -> Old, Evaluated, UpCycled ***/
    /**************************************************/

    function getPendingRequest(uint _pendingIndex) public view
        returns(address, uint, uint, uint, uint, bool, uint) {
         //check index
         require(_pendingIndex<pendingIndex && _pendingIndex>=0, "
             Wrong index");

         return (pendingBox[_pendingIndex].adminAddress, pendingBox[
             _pendingIndex].tshirt, pendingBox[_pendingIndex].pants,
              pendingBox[_pendingIndex].jacket, pendingBox[
             _pendingIndex].other, pendingBox[_pendingIndex].
             isEvaluated, pendingBox[_pendingIndex].points);
    }

                        ....

    /**************************************************/
    /***************** Data of Requests ****************/
    /**************************************************/

```

```
    function getRegenerationCredit() public view returns(uint) {
        return debtPoints;
    }

                          ....
```

Transactions

As explained above, transactions process are the main ones of the smart contracts, leading to a write operation. .

There are two functions that perform transactions between the actors involved in that contract:

1. **evaluateBox**: The evaluation process starts by the invocation of `SendBox` function. Once that there are pending box requests, the next one is evaluated following the price table in order to standardize clothes materials evaluation by `material type` and `weight`. It is set an overall amount of value corresponding to the clothes box request. The transfer process involves, in the thesis work, just one Producer. The points are handled with a `debtPoints` variable that is updated by these two functions. In that case `evaluateBox` add the amount value of the box to the `debtPoints` variable.

```
        // Evaluate Old Box
        function evaluateBox(uint _points) onlyProducer()
            public {
            //check correct pending request index
            require(evaluatedIndex < pendingIndex, "No more
                pending request");

            //check if evaluation is done
            require(!pendingBox[evaluatedIndex].isEvaluated, "
                Request just evaluated");

            //pop pending request
            ClothesBox storage box = pendingBox[evaluatedIndex
                ];

            //update box transaction
            box.isEvaluated = true;
            box.points = _points;

            //add evaluated box
            evaluatedBox[evaluatedIndex] = box;
            evaluatedIndex++;

```

```
            debtPoints += _points;
            totPointsProvided += _points;
        }
```

2. **buyUpcycledBox**: This process leads a purchase order performed by the Admin to the Producer. Admin chooses a kind of fixed box(`small, medium, large`) with a fixed Regeneration Credits price associated. Before performing the purchase process, there is a check of the `debtPoints` balance to allow or not the transaction of the box. If the amount of the Regeneration Credits is enough to buy upcycled clothes, then the `debtPoints` is updated and the value of the purchased box is subtracted to the overall balance.

```
        function buyUpcycledBox(uint _tshirt, uint _pants,
            uint _jackets, uint _other, uint _points) onlyAdmin
            () public {
            require(debtPoints >= _points, "Not enought
                credits accumulated in old material boxes");

            ClothesBox memory box = ClothesBox({
             adminAddress: msg.sender,
             tshirt: _tshirt,
             pants: _pants,
             jacket: _jackets,
             other: _other,
             isEvaluated: true,
             points: _points
            });

            admins[msg.sender].box[admins[msg.sender].numBox]
                = box;
            admins[msg.sender].numBox++;
            admins[msg.sender].creditSpent += _points;

            //add upcycled box
            upCycledBox[upCycledIndex] = box;
            upCycledIndex++;

            debtPoints -= _points;
            totBoxNew++;
        }
```

3.4.3 ERC20 Contract

ERC-20 is a technical standard used to issue and implement tokens over the Ethereum blockchain. The standard describes a common set of rules that should be followed for a token to function properly within the Ethereum ecosystem. Therefore, ERC-20 should not be considered as a piece of code or software. Instead, it may be described as a technical guideline or specification.

The choice to develop an ERC-20 token leads to relaxing the limitation related to the token usage. That contract is deployed over the Ethereum network and it is public, accessible to everyone that owns an Ethereum wallet. The decision, as well as a cross-chain interaction process, leads to open the doors to an external usage of the token, due to what the asset represents

The asset wants to represent the CO_2 emission saved. For example, as asset exchange to measure the emission saved recycling a t-shirt even to produce it starting from scratch.

The main information associated to the created token are:

- **Symbol**: CO2, it is used to identify a token, this is a three or four letter abbreviation of the token.
- **Name**: CarbonToken, it able to identify them.
- **Total supply**: 1,000,000,000, it is the max supply of the token.
- **Decimals**: 18, it is used to determine what decimal place the amount of the token will be calculated. The most common number of decimals to consider is 18.

The main features of the contract are describer by ERC20 interface

```
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

Below I list and explain in details the six mandatory functions that defines the erc20 tokens:

- **totalSupply()**: the supply could easily be fixed, as it happens with Bitcoin, this function allows an instance of the contract to calculate and return the total amount of the token that exists in circulation.
- **balanceOf()**: This function allows a smart contract to store and return the balance of the provided address. The function accepts an address as a parameter, so it should be known that the balance of any address is public.
- **approve()**: When calling this function, the owner of the contract authorizes, or approves, the given address to withdraw instances of the token from the owner's address.
- **transfer()**: This function lets the owner of the contract send a given amount of the token to another address just like a conventional cryptocurrencies transaction.
- **transferFrom()**: This function allows a smart contract to automate the transfer process and send a given amount of the token on behalf of the owner.
- **allowance()**: This functions allow the caller to check if the given balance's address has enough token to send the amount to an other address.

3.5 Network Architecture

3.5.1 Main Components

Before going deeper to explain my network architectural choice, it is important to have an overview of the main components involved in the Hyperledger Fabric Architecture:

1. **Peer**: It is the fabric node, there are different kinds of peers and each one can perform specific actions
    (a) **Anchor Peer**: this kind of peer is used for communications between organizations. It makes peers in different organizations aware each other.
    (b) **Committing Peer**: Every peer in the channel
    (c) **Endorsing Peer**: every peer that has the smart contract installed can be an endorsing peer.
    (d) **Peer Node**: each peer maintains a copy of the ledger for each channel it is a member of.
    (e) **Leader Peer**: an organization can have multiple peers in a channel. Only one peer from the organization needs to receive the transactions. The leader distributes transactions from orderers
2. **Certification Authority**: Everyone who wants to interact with the network needs an identity. The CA provides the means for each actor to have a verifiable digital identity. Exploiting CAs is implemented the membership mechanism providing a permissioned blockchain.
3. **MSP**: Membership Service Providers (MSP) is a Hyperledger Fabric component that offers an abstraction of membership operations. In particular, an MSP abstracts away all cryptographic mechanisms and protocols behind issuing certificates, validating certificates, and user authentication.
4. **Orderer**: It is like a network administration point. The ordering nodes support the application channels for ordering transactions, create blocks and add it to the chain.
5. **Organization**: Identify a category of users involved in the network. each user's certificate of the Organizations is released by the same CA. The organizations are used in permissioned mechanism allowing or not read and write access to specific data over the network.
6. **Consortium**: A group of organizations that share a need to transact. It could share a set of permission over the network.

7. **Channel**: A channel allows a consortium, group of participants, to create a separate ledger of transactions. The transactions, stored in the world state, are visible only to the members of the channel.

8. **Ledger**: It is stored over the peer and consists of the World State of the blockchains. All the transactions performed over the chain is merkled in the world state[29]

3.5.2 Own Architecture

The **Figure 3.9** shows the main components of the network architecture built for the thesis work. It includes:

1. **3 Peer**: One dedicated peer for each organization involved in the system.
2. **1 Orderer** organization with *1 ordered* node running
3. **3 Organizations** each with 1 peer, Peer0, running
   (a) *Org1*: User Organization
   (b) *Org2*: Reclothes Admin Organization
   (c) *Org3*: Producer Organization
4. **2 Channels**
   (a) *Chanel12*: It is the channel between Org1 and Org2, and allows the communication between User and Reclothes
   (b) *Chanel23*: It is the channel between Org2 and Org3, and allows the communication between Reclothes and Producer
5. **2 Consortiums**
6. **CC12**: related to channel 1, allows actors related to Org1 and Org2 to join the channel.
7. **CC23**: related to channel 2, allows actors related to Org2 and Org3 to join the channel.

This is a test network, light, to test the entire thesis project. Hyperledger Fabric allows us to implements in an easy way more components adding orderer or Peers. It makes Hyperledger architecture highly modular. For production the architecture needs some modification, adding more orderers and peers for each organization, in order to maximize the fault tolerance.

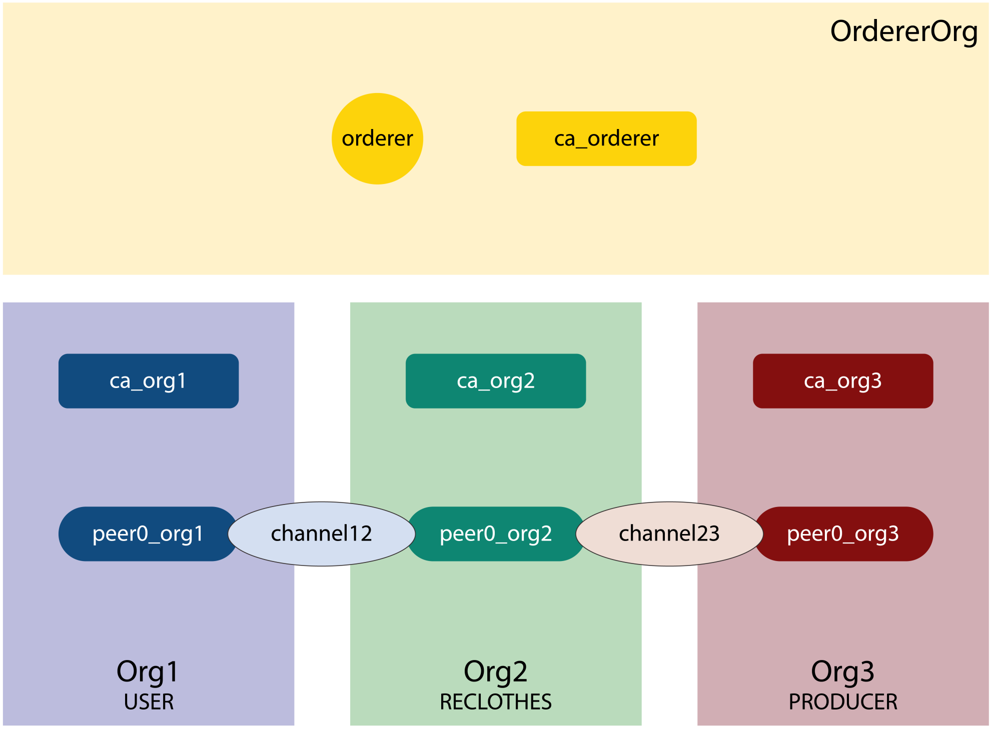


Figure 3.9. Fabric Network

3.5.3 Fabric Network

Figure 3.10 shows how components interact each other. It is possible to separate components into 2 categories, inside and outside Fabric Network. First of all we need to describe the components involved:

- **Web3 App**: It is the Dapp and the Client connection to the network.
- **Channel**: It is the channel above which transfers data.
- **CA**: It is the Certification Authority in charge of release certificates.
- **Peer**: It is "Fabric node", the endpoint of the internal network. It owns by specific CA with fixed rights, linked to the connected channels.
- **evm SC**: It is the Ethereum Virtual Machine Chaicode, used to run Solidity Smart Contract. The chaincode is installed over the peer.
- **ledger**: It is the ledger associated to the channel connected. There is a 1 to 1 association between ledger and channel.

- **CC**: It is the *Consortium*, It is associated to the channel, it manages ownerships and it includes a set of Organizations allowed.
- **Docker**: The network components run inside docker container.

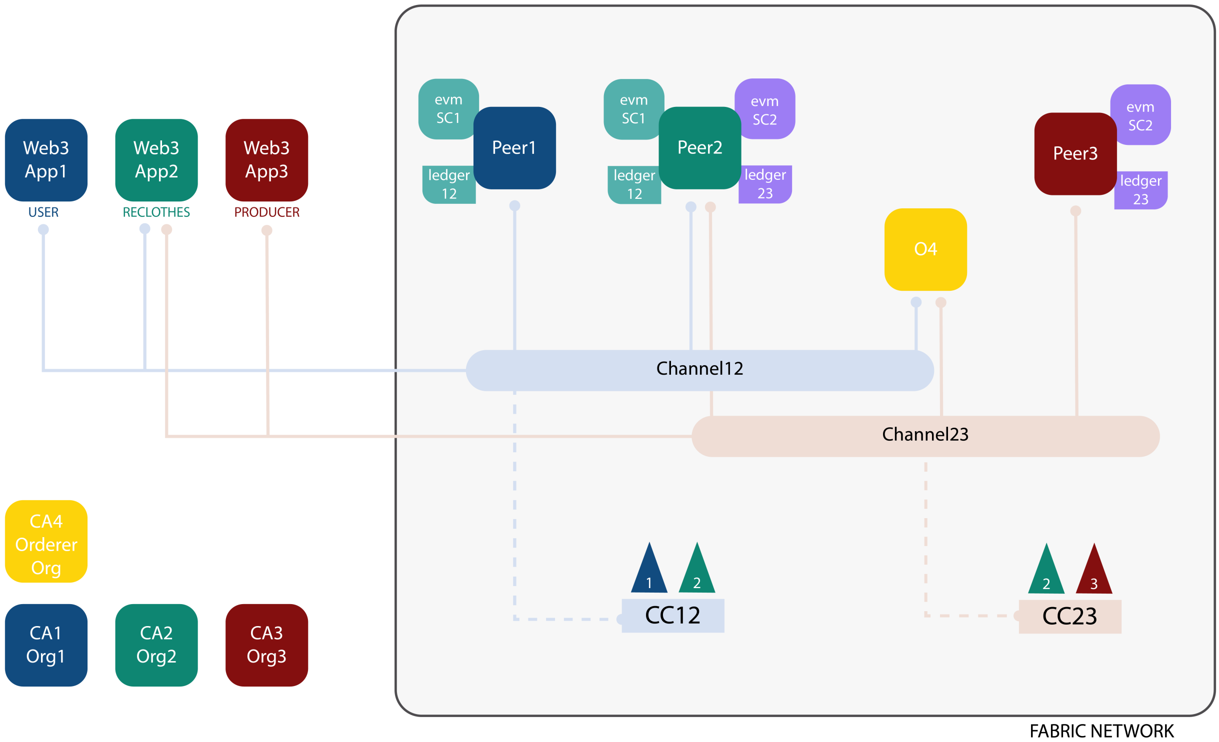


Figure 3.10. Fabric Network Components

By Figure 3.10 it is possible to see that in the architecture there are two ledgers, each of that associated with one channel that involves just part of organizations per channel. The ledger is associated with one or more smart contract deployed over the chaincode, in own case we use 1 smart contract deployed each chaincode.

The *chaincode* is invoked calling the EVM chaincode by the *App* Client, using channel communication. The channel is the only communication way between outside and inside the network. The external actor that invoke the chaincode must have the privileges to join the specified channel. The chaincode installed over the peer once is invoked agreed to the request and invoke the chaincode("smart contract") method.

Once the method returns, the chaincode forwards the reply to the App client. Then the Dapp forward the answer to the *Orderer* peer that validates the response, create a new block, add it to the chain, communicate it to the peer in order to synchronize the network and updating the Ledger World State.

Config File

The network architecture in Hyperledger Fabric is highly modular and scalable. Hyperledger provides to the developers a set of developed test network[30] for testing purpose and that help developers to better understand the structure and the steps of the creation. The `fabric-samples` contains a set of tools that allow releasing all the cryptographic materials required for the usage of the network, such as certificates related to the user belonging to a specific organization. In other words, all the MSP works are handled by fabric-samples tools.

The most useful thing about the Hyperledger network is that the components could be added or removed in an easy way. To design and set up network components and rules, it is has been written the `config.yaml` file.

The `config.yaml` is structured in the following way:

- **Organizations**: here must be listed all the organizations involved in the network, specifying the MSP and the policies of each Org.
- **Consortium**: once the organizations are specified, the file defines the consortiums involved in the network. The consortium will be joint to specific channels.
- **Channels**: in the last Section must be specified the channels of the networks, specifying the consortium and the related organizations allowed over the channel.

Run of the network

To set up the network in the best way, some scripts that performs the integration of several components in the best way, are created.

The macro process flow of the network running is:

1. **Check Prerequisite**: using the `fabric-samples`[30] there is some prerequisite to check, to run all the materials in the correct way. Check the prerequisite looking at fabric documentations[31].
2. **Run Network**: Once the `config` file is designed and developed, we need to include evm chaincode in the volumes of docker files. Then it is possible to run the network.
3. **Join all the components**: Once the network is in running, it is important to join all the components each other, for example join the peers to the dedicated channels.

4. **Chaincode**: Once that all components are set up in the right way and the network is in running, it is possible to install the chaincode over the peers that we want to use.

The Hyperledger Fabric network runs inside **docker containers**. To automatize the running of the network I created scripts that setup Fabric locally using docker containers, install the EVM chaincode (EVMCC) and instantiate the EVMCC over the Fabric peers. All that thesis steps use the Hyperledger `fabric-sample` repository to deploy Fabric locally and the `fabric-chaincode-evm` repo for the EVMCC.

The scripts developed are the following ones:

- `net_up.sh`:
  - **Generate crypto materials for organizations**: First of all, using the `fabric-sample` supplied by IBM, it is possible to run a tool in charge of creating all the cryptographic material for the actors used to operate over the network.
  - **Generate channel artifacts**: In the same way, the cryptographic materials generated for the Organizations must be generated for the channel involved in the network.
  - **Run docker containers**: Once the crypto materials are created then it must run the docker containers for the components of the networks. The **Figure 3.11** shows the related output.
  - **Initialize Orgs and join them to the channels**: Once all the containers are in running then there are the organization's initializations and each peer owner by an Org is joined to the channel following the `config file` specifications. **Figure 3.12** shows the related output.
  - **Instantiate and Install evm chaincode**: Once all the network is in running we are going to install the chaincode over the peers. In my case, we are going to install `fabric-chaincode-evm`. **Figure 3.13** shows the related output.
- `net_down.sh`:
  - **remove all docker containers in running**
  - **remove all docker volumes created**
- `fab1.sh`:
  - **network-sdk-config.yaml**: It is the mapping SDK file from web3js request to the fabric requests.

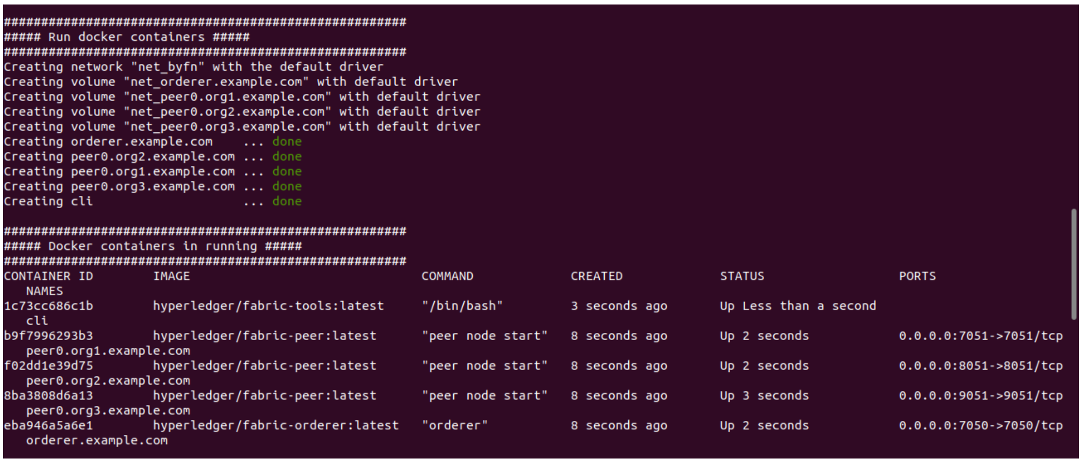


Figure 3.11. Run of the Docker Containers

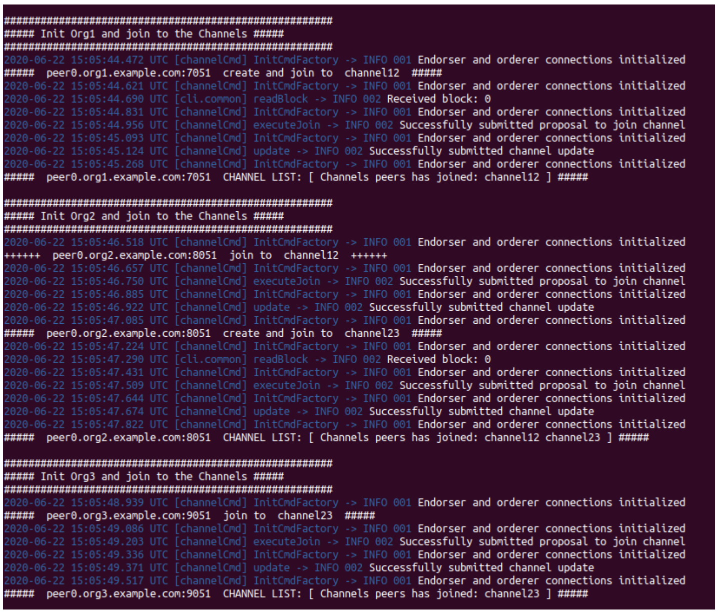


Figure 3.12. Orgs initializations and channels join

Scripts produced

For a better understanding of the setting up and running of the entire network, I reported below the two main scripts. The two files `net_up.sh` and `net_down.sh` show in the best way all the steps that allow the network running.

`net_up.sh`:

```
#!/bin/bash
```

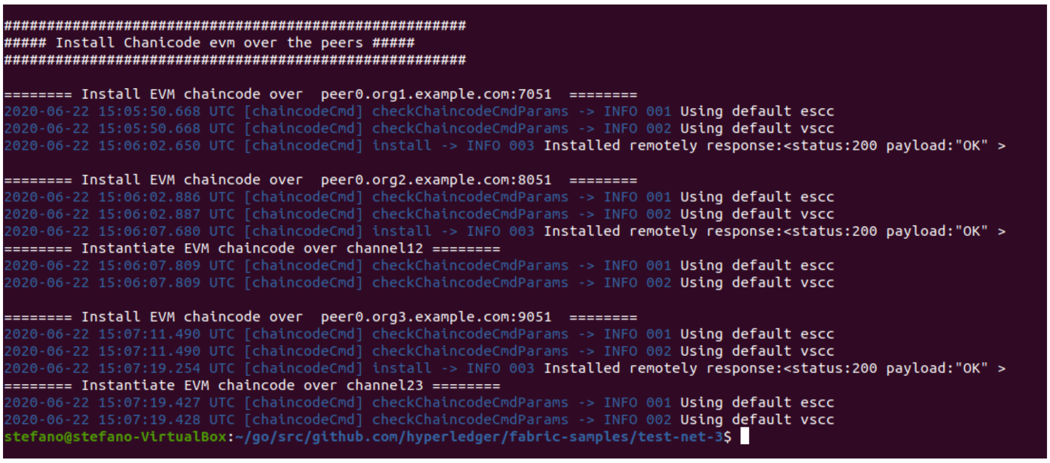


Figure 3.13. Chaincode Installation

```

##### Generate Crypto Material for Organizations #####
../bin/cryptogen generate --config=./crypto-config.yaml

##### Build Channel Artifact #####
./channel_artifact.sh

##### Run docker containers #####
docker-compose up -d

##### Docker containers in running #####
docker ps

##### Init Org1 and join to the Channels #####
docker exec cli scripts/org1_init.sh

##### Init Org2 and join to the Channels #####
docker exec cli scripts/org2_init.sh

##### Init Org3 and join to the Channels #####
docker exec cli scripts/org3_init.sh

##### Install Chanicode evm over the peers #####
docker exec cli scripts/install.sh
```

`net_down.sh`:

```
#!/bin/bash

# STOP AND DELETE THE DOCKER CONTAINERS
docker-compose down -v
```

```
    docker rm $(docker ps -aq)
    docker rmi $(docker images dev-* -q)

    # DELETE THE OLD DOCKER VOLUMES
    docker volume prune

    # DELETE OLD DOCKER NETWORKS (OPTIONAL: seems to restart fine
       without)
    docker-compose -f  down --volumes --remove-orphans
    docker network prune

    # DELETE SCRIPT-CREATED FILES
    rm -rf channel-artifacts/*.block channel-artifacts/*.tx crypto-
       config

    # Remove created folder
    rm -rf channel-artifacts

    # VERIFY RESULTS
    docker ps -a
    docker volume ls
    ls -l
```

3.5.4 Ethereum Network - Ropsten

To run the ERC20 token it is used the testnet Ropsten against the mainnet. To set up and upload the ERC20 Token over the Ethereum network it is used the following tools:

- **My Ether Wallet**: To upload ERC20 contract.
- **Etherscan.io**: To monitor and analyze transactions over the network.
- **Metamask**: To create user wallets and sign eth transactions.
- **Infura**: To set up a node in order to use it as endpoint and communicate with the Ropsten network, it is used as *Provider* in *Web3* library.

3.6 Fabric and EVM chaincode interaction

3.6.1 Chaincode invocation

To analyze how evm chaincode allows running Solidity bytecode inside Hyperledger Fabric network, first of all, we analyze the interaction process and chaincode invocation Process of Hyperledger Fabric blockchain.

End to End Interactions

Going deeper, the **Figure 3.14** shows the flow of the end to end communication. How all the components are boxed inside the Peer component. The Fab3 maps the web3 request and forwards it to fabric peer. the request arrives at the evmcc that invokes Solidity smart contract methods.

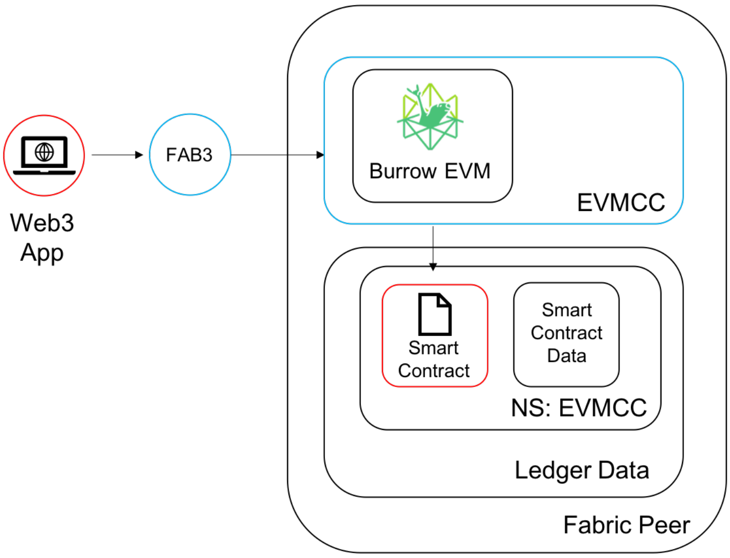


Figure 3.14. End To End

Chaincode Invocations

The **Figure 3.15** below describes the internal workflow of the chaincode invocation, where is involved the *Client*, the *Peer* and the *Orderer*. All the information are transferred over the set channel and in the thesis work, the client doesn't interact directly but using *Fab3 Proxy* as intermediary.

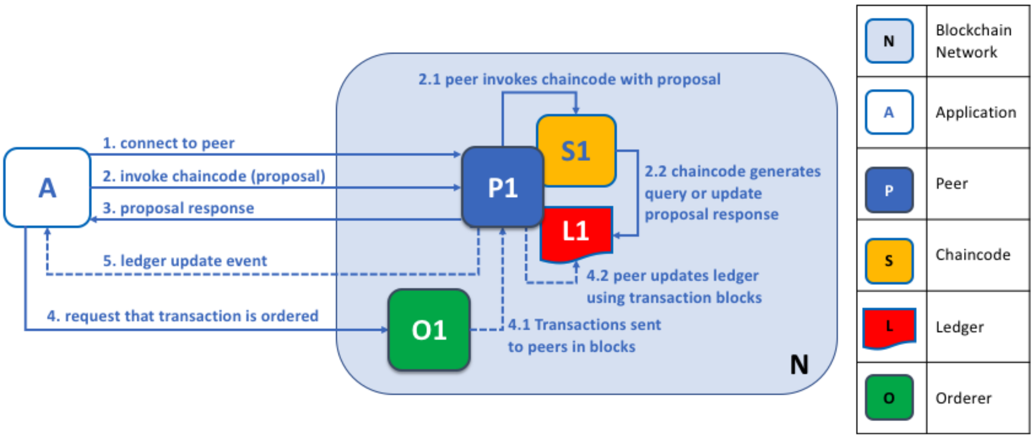


Figure 3.15. Smart Contract Invocation Process. Source

3.6.2 Fab3 Proxy

The Fab3 Proxy is a main component of the entire architecture. It works as a bridge between the Ethereum world and the Hyperledger Fabric one. Each instance of Fab3 map 1 Fabric user and generates a semi-random Eth address starting from the public key of the user's x.509 certificate related to the Fabric identity. The Following environment variable set the mandatory data to run a Fab3 proxy instance:

- **CONFIG**: It is the path to a compatible Fabric SDK Go config file, used to communicate and map the requests and forward it to the Fabric network.
- **USER**: User identity being used for the proxy. Matches the user's names in the crypto-config directory specified in the config.
- **ORG**: Organization of the specified user.
- **CHANNEL**: Channel to be used for the transactions.
- **CCID**: ID of the EVM chaincode deployed in your fabric network.
- **PORT**: Port the proxy will listen on. If not provided default is 5000

Below there is an example of environments variable setting up used to run Fab3 instance.

```
# Environment variables required for setting up Fab3
export FAB3_CONFIG=${GOPATH}/src/github.com/hyperledger/fabric-
    chaincode-evm/examples/network-sdk-config.yaml
export FAB3_USER=User1
```

```
export FAB3_ORG=Org1
export FAB3_CHANNEL=channel12
export FAB3_CCID=evmcc
export FAB3_PORT=5000
```

Once the Fab3 is set up and the instance is in running it is allowed to perform chaincode invocation using the thesis's Dapp. **Figure 3.16** shows the flow of the invocation process at upper level. It shows the main components that are involved in that process.

1. **Dapp** calls method that performs smart contract invocation.
2. **Fab3** maps the request and forwards it to Fabric network.
3. **Fabric network** processes the request and issues a response.

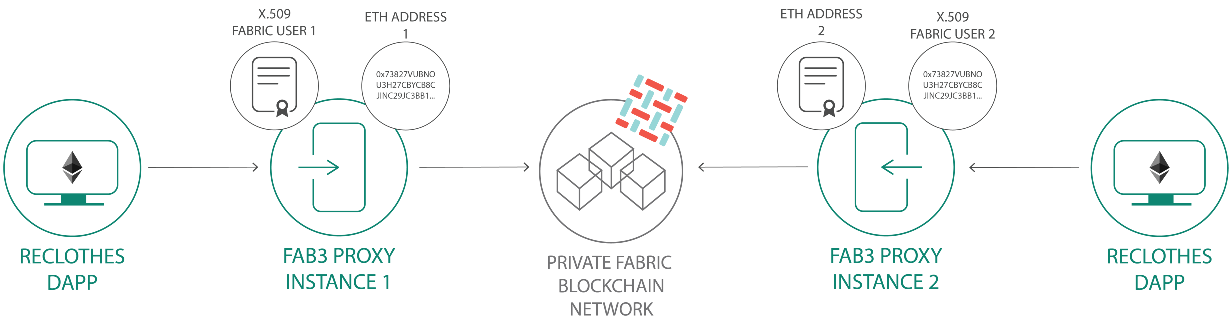


Figure 3.16. Fab3 Proxy Flow

The **Figure 3.17** shows the internal components that take part to the invocation process. Fab3 agreed to the request using *Ethereum JSON RPC API*, map it and forward it to the Fabric network using *GO SDK*. Once the request arrives at *EVMCC*, it invokes smart contract bytecode and then follows the standard process explained in Figure 3.15.

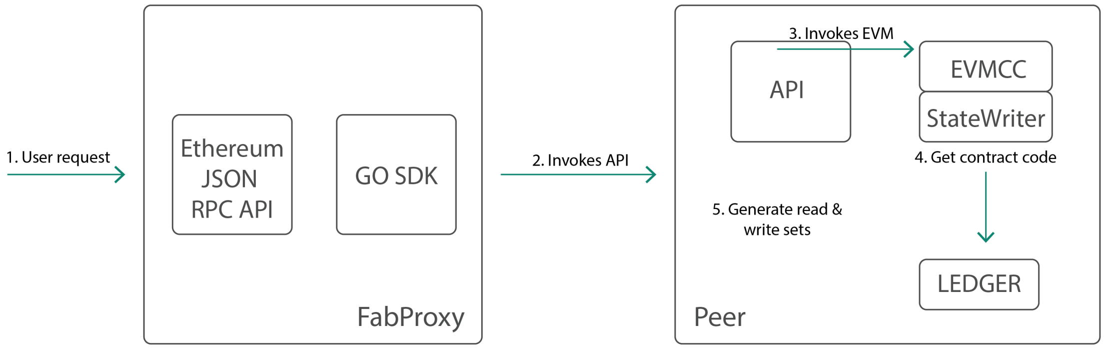


Figure 3.17. Fab3 Invocation Process

3.7 Dapp

3.7.1 Technologies used

The client application is a web app, composed by a front-end part and a mid-level with API that allows the communication with smart contracts of both Blockchains. The technologies used are the following one:

- **Expressjs**: It is a node.js framework that allows developing API for the application.
- **Bootstrap**: To build a user-friendly front-end in order to interact in the best way.
- **Web3**: Ethereum Javascript API, It is a collection of libraries that allow you to interact with a local or remote Ethereum node.
  - **web3 0.20.2**: used for dapp developments, Fabric side, It is a stable version and it is the version used in `fabric-chaincode-evm` development.
  - **web3 1.0.0**: used for Ethereum transactions, It is a version with more features.

Starting from the Homepage the User is allowed to register itself as **User**, **Reclothes Admin** or **Producer**.

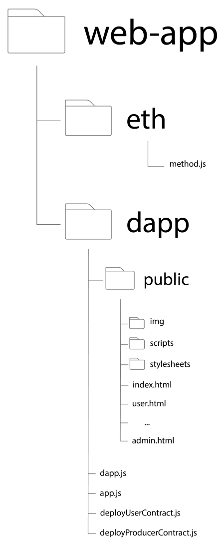


Figure 3.18. DApp directory tree

3.7.2 Core part of the web-app

Figure 3.18 the directory tree of the web app folder.

The technical files and flow that dapp follows to run up it is the following one.

1. **Contract Address Generation**:
   (a) This step is in charge to run a script that deploys the contract addresses to be refereed during the app running.
      i. **deployUserContract.js**: running the script using `node .js file`, it returns the address of the deployed contract. **Figure 3.19** shows the related output and the Contract Address printed.
      ii. **deployProducerContract.js**: running the script using `node .js file`, it returns the address of the deployed contract. The deployed process and output is similar to Figure 3.19.
2. **dapp.js**: It is the core file that handles the contracts invocations, it set up the contract address reference, and connect to a specific Fab3 instance.
3. **app.js**: It set up the API called by the web-app, it maps the request and forwards to `dapp.js`.
4. **method.js**: here there are all the Ethereum API endpoints. It is called by the web-app to perform calls over Ethereum network.

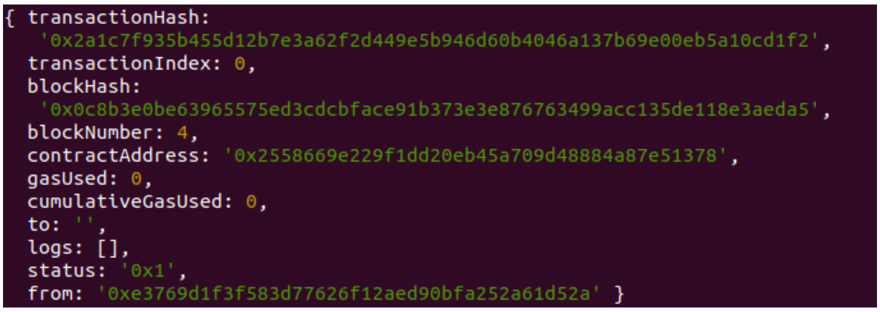


Figure 3.19. Deploy User Contract

Once everything is set up, it is possible to run the web-app with the command `npm start` and there is an initialization phase. After that, the app is ready to be used and in running over the specified PORT. The **Figure 3.20** shows the output.

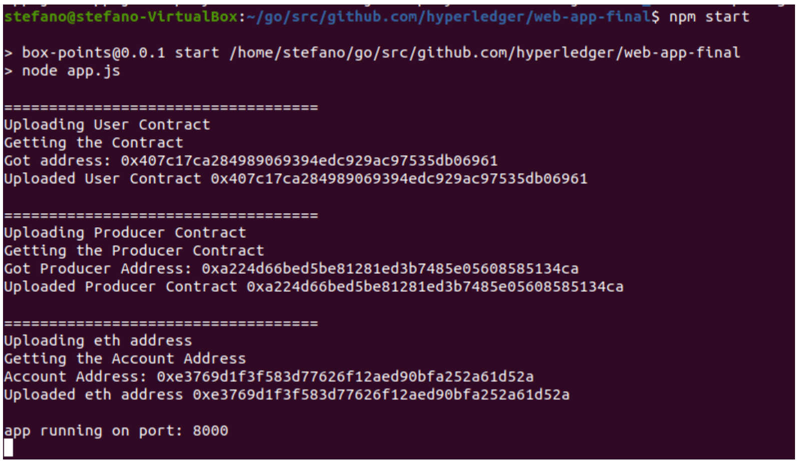


Figure 3.20. App Running

3.7.3 Views

Below I list all the views produced in the thesis work. For a better usability and understand of application behaviors, I care about the UI too.

Homepage

The homepage allows user to view the feature of each User type and to access to the registration page.

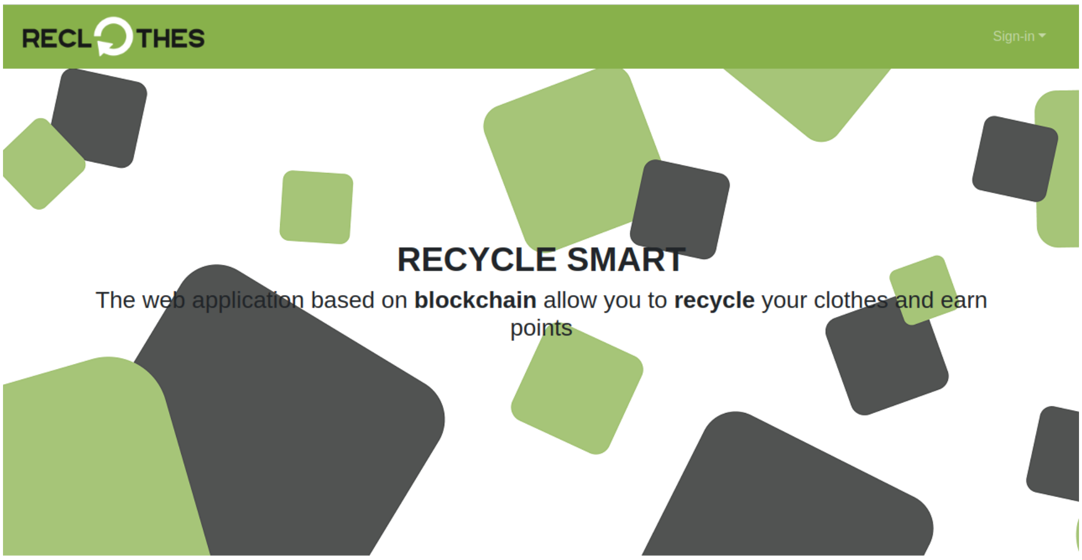


Figure 3.21. Home

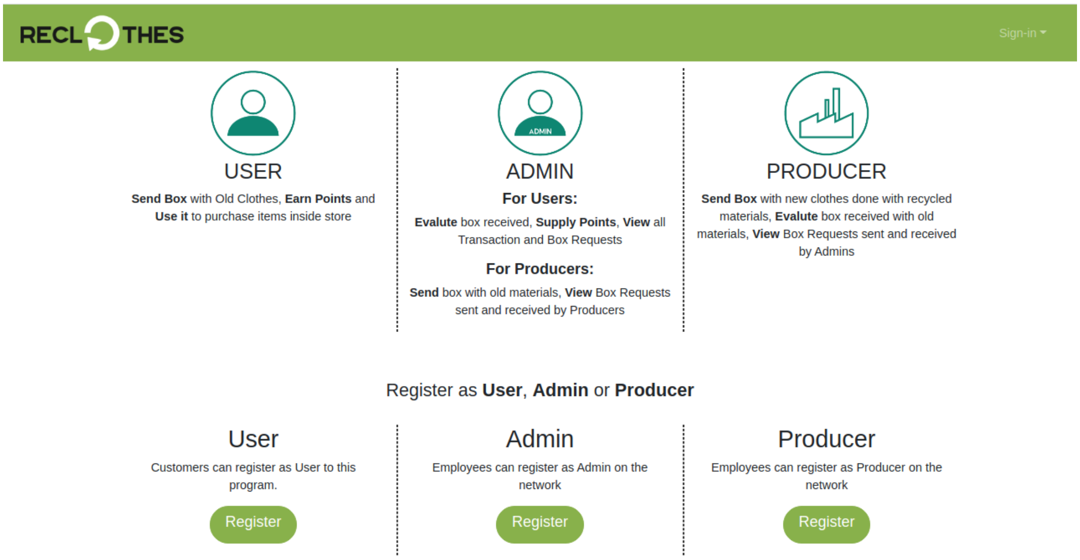


Figure 3.22. Registration Phase

User Page

The User page allows to view an overview infos once the user is logged in.

- **Address**: It is the public eth address setup during registration phase.
- **Points Balance**: It is the Fabric points balance earned by the user sending the boxes.
- **ERC20 Balance**: It is the eth balance of the public token running over eth network.

Figure 3.24 shows how to compile the form in order to send box with old clothes. It is a simulation of the real process of sending boxes, in the real case should be implemented using a QRCode or RFID placed over the boxes.

Figure 3.25 shows how the store should be. The purchasing of the items over the platform starts the transaction process.

There is another Section about info that the user is allowed to see. **Transactions** performed over the Fabric network and **Box Requests**, there is all the history about the box sent and received with all the related information's data.

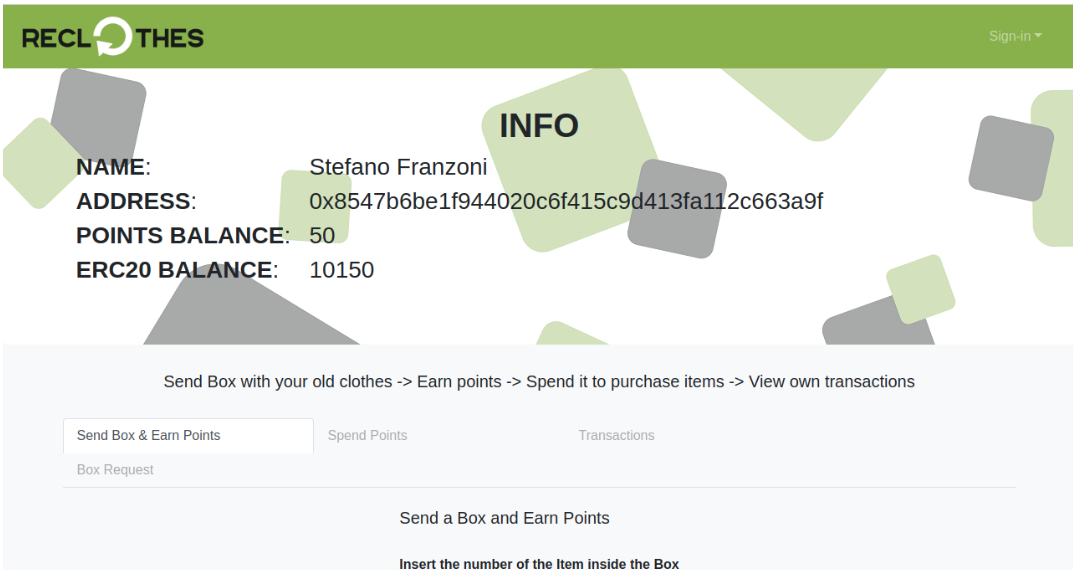


Figure 3.23. User Info

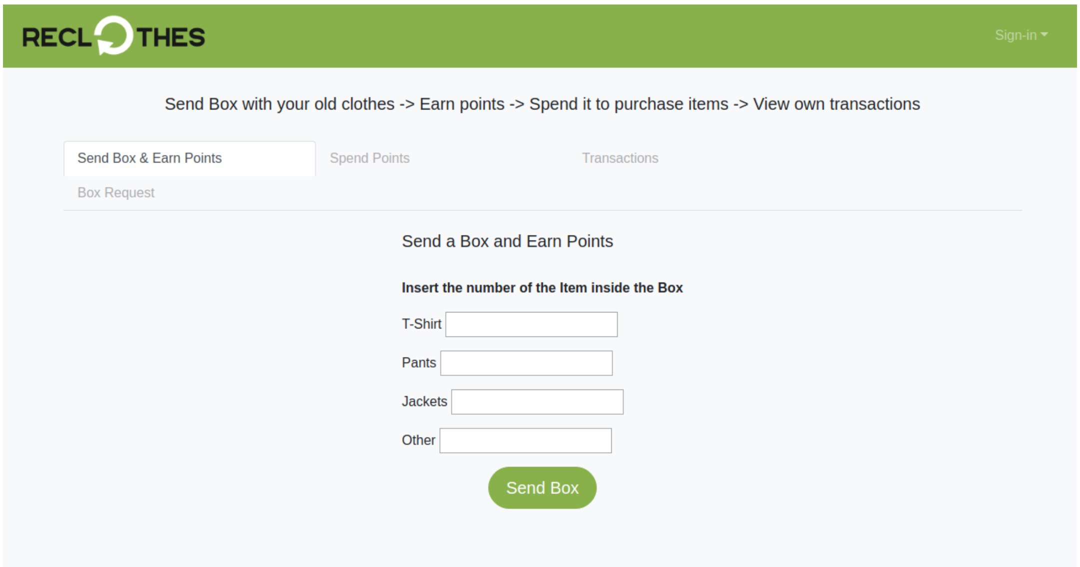


Figure 3.24. Send Box

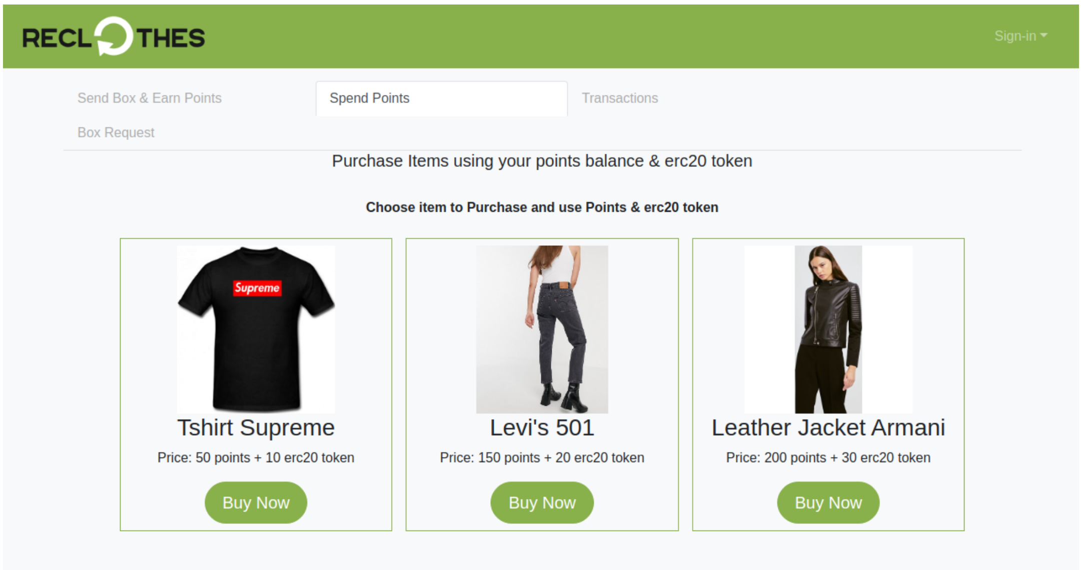


Figure 3.25. Purchase Clothes

Reclothes Admin Page

In the previous Sections, I explained the logical split about *Admin for User* and *Admin for Producers*. In the following views, I divide the features related to the user type handled and there is a strong distinction about Users and Producers.

Admin For Users This section shows the views of the Admins that handle User side. Figure 3.27 shows the field to compile in order to perform the evaluation, meanwhile, Figure 3.28 shows the transactions records. Here a link to a ropsten transaction.

Admin For Producers This Section shows the views of the Admins that handle Producer side. The Figure 3.29 shows all the Admin for Producers information's data.

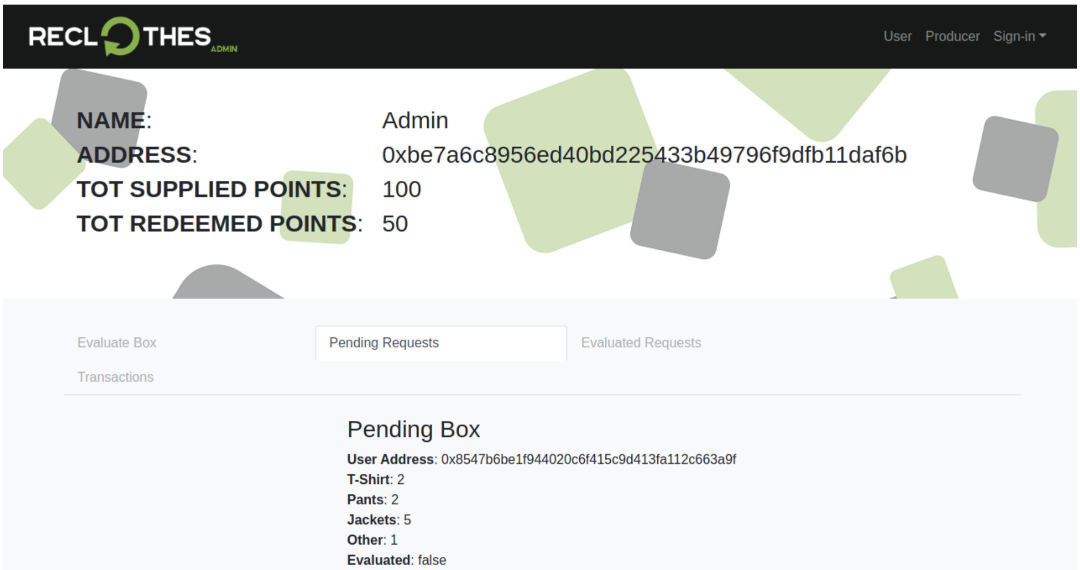


Figure 3.26. Admin Info

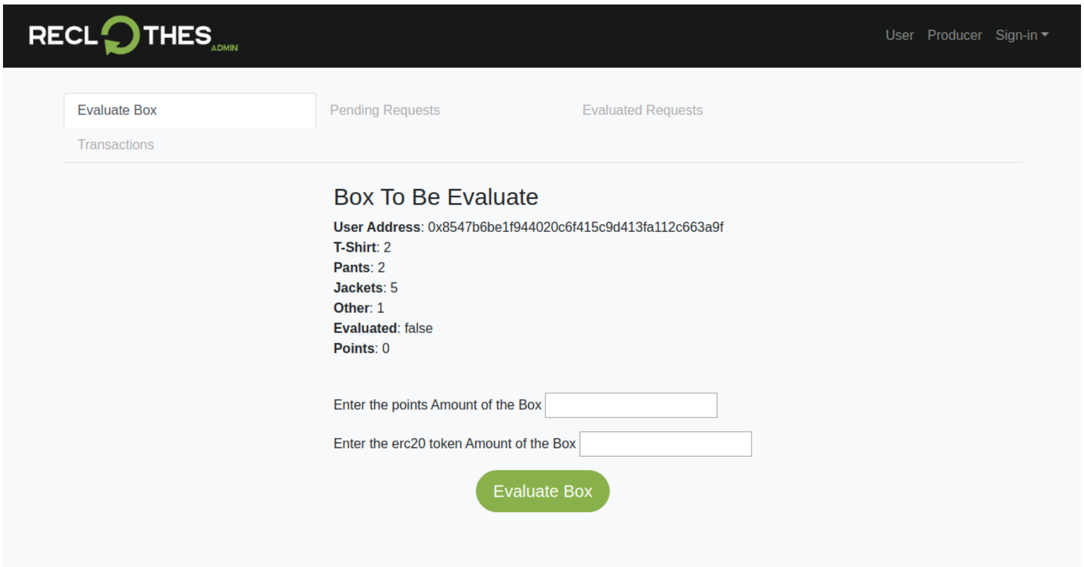


Figure 3.27. Evaluate Box

Producer

This Section shows the views of the Producer side. The evaluation process of the old materials received by Reclothes is the same of the previous one.

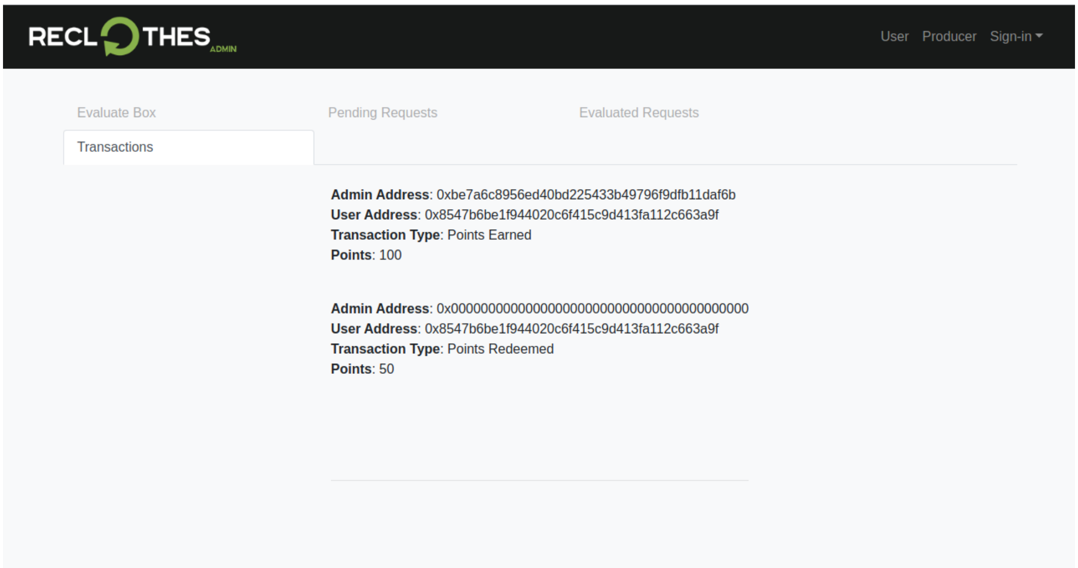


Figure 3.28. Transactions

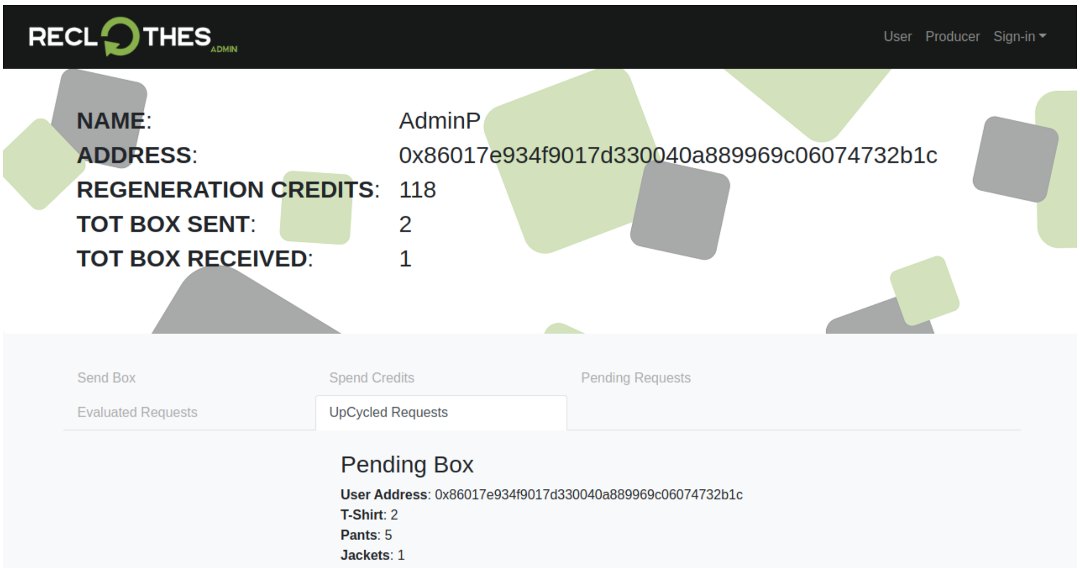


Figure 3.29. Admin for Producers Info

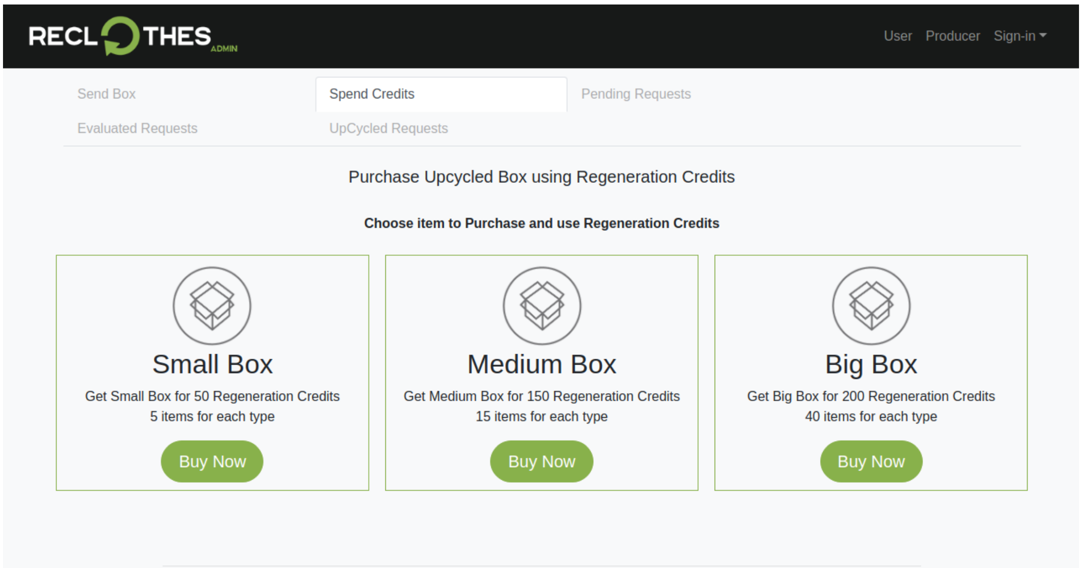


Figure 3.30. Admin Spend Regeneration Credits

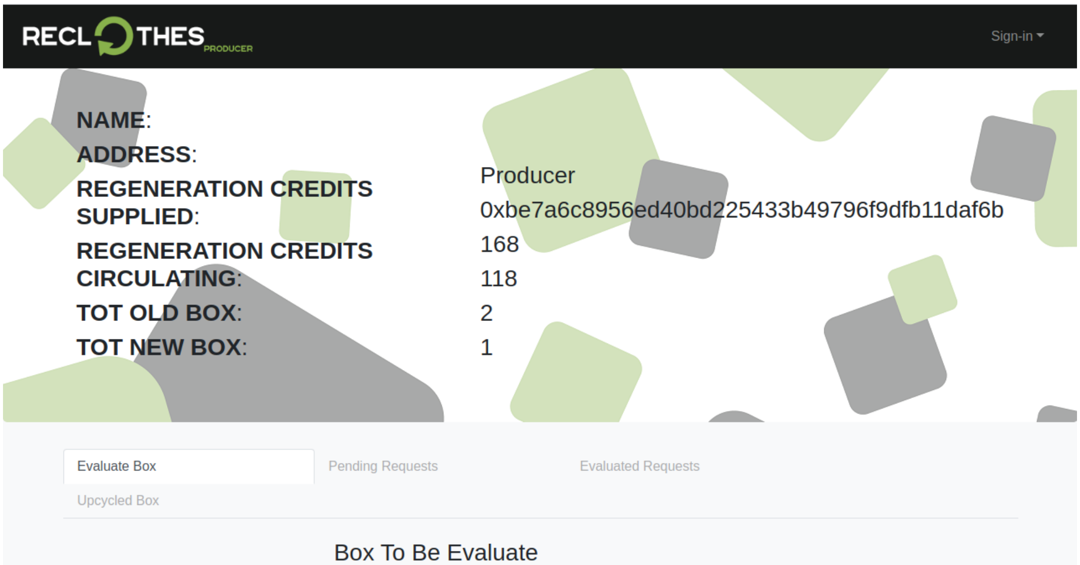


Figure 3.31. Producers Info

3.8 Cost Analysis

In the following Section, there are a cost analysis of the networks used for the thesis work.

3.8.1 Hyperledger Fabric

Hyperledger Fabric is a permissioned network and it doesn't have transaction fees. On the other hand the companies that run own project over Fabric have to maintain the costs related to the node of the networks created. There's several services and companies that sell hosting services related to Hyperledger network nodes.

To estimate the costs, our analysis is based on Amazon Managed Blockchain[32] test network owned by a single customer. This network has three Starter Edition members(1 each actors/organization) to simulate a multi-party transaction. Each member has a single **bc.t3.small** peer node with 20 GiB of storage and writes 9 MB to the network per hour.

- **The hourly cost for this network is**:
  - **Membership cost**: (3 Starter Edition members) x ($0.30 per hour) x (1 hour) = $0.90 per hour
  - **Peer node cost**: (3 members) x (1 bc.t3.small peer node per member) x (0.034 per hour) x (1 hour) = $0.102 per hour
  - **Peer node storage cost**: (3 members) x (1 peer node per member) x (20 GiB storage per peer node) x ($0.10 per GB-month) x (1 hour) = $0.009 per hour
  - **Data written cost**: (3 members) x (9 MB per hour) x (0.10 per GB) = $0.003 per hour
- **Total test network hourly cost**: $1.014
- **Total test network year cost**: $8882.64

3.8.2 Ethereum

The Ethereum Network prices is related to the transactions fees costs that depends on the data size of the transaction and on the gas price expressed in ether. The price for each gas defines the transaction time, it means that more gas price corresponds to a less time of transaction computed and vice-versa. There are several features that going to influence the transaction fees over Ethereum network, such as ether price, or network traffic.

The following price analysis is related to the current value of the ether that is $237.36.[33] [34]

Gas Price	Confirmation Time	Transfer Price
1 gwei	128 secs - 2 minutes	$0.005
34 gwei	85 secs - 1 minutes 25 secs	$0.17
66 gwei	13 secs	$0.33

Table 3.1. Comparing among Ethereum transactions price

Considering 100.000 active Users. Each User on average sends 2 boxes with old clothes per year. Therefore, Reclothes Side, the application performs 200.000 evaluation processes that belong to **200k** of incoming transactions per year. The incoming transaction number, multiplied for the minimum transaction cost, belongs to an overall cost of 200.000x$0,005, which is **$1000**.

Since there are just 70% of active users, and considering an average of 10 purchasing per User every year. The Ethereum network has to process 70.000x10 = 700.000 transactions. Therefore as the above hypothesis to get the less cost per transaction; the overall costs of the outcome transaction is of 700.000x0,005 = $3500.

In the end, the Ethereum network's overall cost is $4500 per year.

3.8.3 Overall costs

Based on the 3.8.1 and 3.8.2, supposing a community of 100k users. The overall cost of the two networks is of $13,382.62 per year.

Chapter 4

Results

4.1 Target archived

The goals to archive include both logical and technical target. The improvements reached by thesis development are the following ones:

The main target to reach is to improve Value Chain [1] value of the overall system. The goal could be split inside **Technical Goal** and **Logical Goal**.

- **Technical Goal**
  - **CrossChain Interaction**: Integrates into the same application both permissioned and permissionless Blockchain networks. The integration is done application side. Some API endpoints start transactions in both networks, one over Fabric network and the other one over Ethereum.
  - **Traceability**: This goal is archived implementing smart contracts, Hyperledger Fabric side, that tracks all the clothes box and store the entire transactions passed over the system.
- **Logical Goal**
  - **Supply Chain**: The target is to simplify the supply chain process, all the steps inside the chain are handled as transactions, stored over the ledger and updating world state and smart contract data.

[1] This process includes the following phases: design and development of the product,raw materials management, production, shipping, selling and final use

- **Sustainability**: The entire process aims to support sustainability. Thanks to traceability feature, it is possible to follow the lifetime of the clothes until they finish to Producer, that performs the material recycling in order to produce new upcycled clothes.
- **Counterfeiting**: Assign a UID to each clothes produced it is possible to fight the Counterfeiting implementing new features such as the clothes registrations. In that way it is possible to have a secure register containing all the clothes.

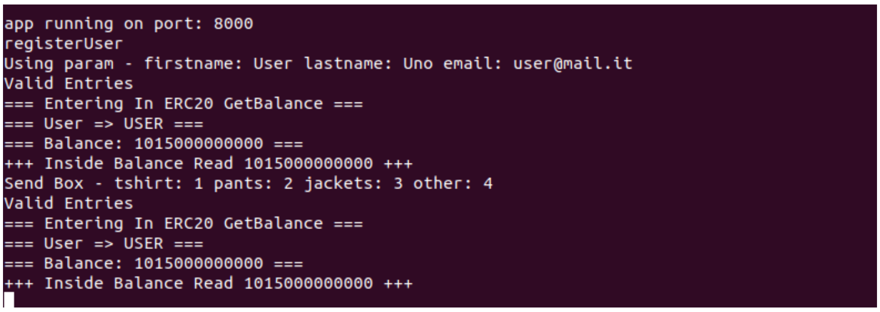
4.2 Use Case Test

4.2.1 Use Case 1 - Unit Test 1

Send Box and Evaluation

Performing the Test over the Use Case 1 about the send box and evaluation processes. The following figures show the results over the call of the related methods and how the application works.

Figure 4.1 shows the log when User performs the *Send Box* action.



```
app running on port: 8000
registerUser
Using param - firstname: User lastname: Uno email: user@mail.it
Valid Entries
=== Entering In ERC20 GetBalance ===
=== User => USER ===
=== Balance: 1015000000000 ===
+++ Inside Balance Read 1015000000000 +++
Send Box - tshirt: 1 pants: 2 jackets: 3 other: 4
Valid Entries
=== Entering In ERC20 GetBalance ===
=== User => USER ===
=== Balance: 1015000000000 ===
+++ Inside Balance Read 1015000000000 +++
```

Figure 4.1. User Send Box

Once the Box Request was successfully sent, the smart contract is invoked and the transaction is performed. Admin could visualize the pending box requests to be evaluated. Then the Reclothes Admin, UI side, insert the value amount of the tokens and start the evaluation process.
Figures 4.2 and **4.3** shows the Fabric Transaction performed then the initialization of the Ethereum transaction. In the end, once the eth transaction was performed, the etherscan link associated with the related TransactionHash, allow the user to see the transaction history and info.

Figure 4.2. Init Transaction from Reclothes to User

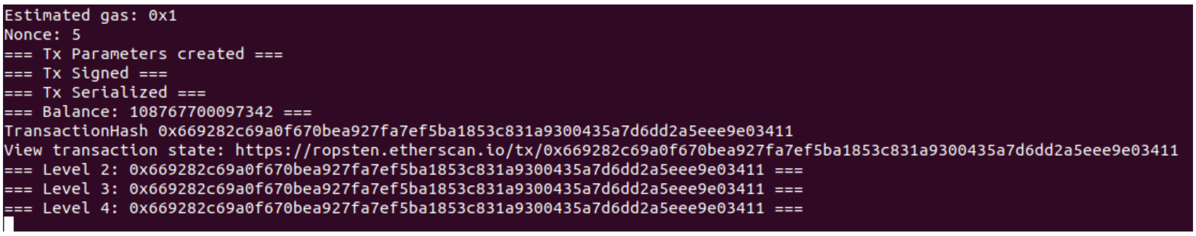


Figure 4.3. Init Transaction from Reclothes to User

Figure 4.4 and Figure 4.5 shows the proof of the transactions succeed. The first Figure shows the User page that allows visualizing the history of transactions done and all related requests. The second one shows the etherscan page with all the information about Ethereum's transaction, in this case from Reclothes eth Account to User Account.

4.2.2 Use Case 1 - Unit Test 2

Purchase Item

The **Figures 4.6** shows the Purchase process. As the figure shows there is, first of all, the Fabric transaction and then the Eth transaction. Once all the previous checks are performed, the transaction from User account to Reclothes account starts. Once the transaction is performed the method prints the etherscan link to monitor the transaction and all the related info.

4.2.3 Use Case 2 - Unit Test 1

To test the Use Case 2 I decided to track the behaviors of two processes:

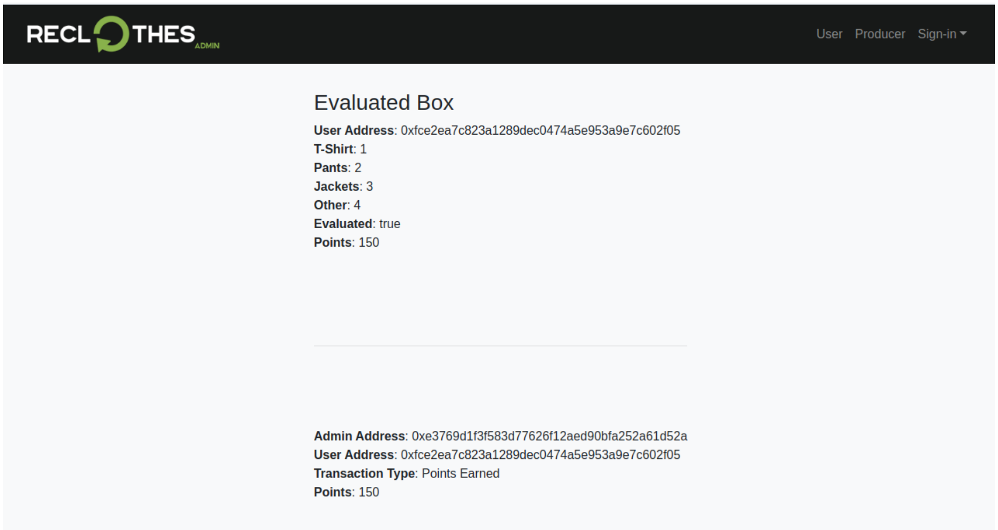


Figure 4.4. Fabric transaction history

Figure 4.5. Ethereum transaction over etherscan

- **Send Old Material and Evaluation**: The Admin for Producer sends a box with inside the old materials to be recycled. Once the box arrived at Producer, then it is going to be evaluated and starts a Regeneration Credits transaction from Producer to AdminP.
- **Purchase Upcycled Material**: The Admin for Producer spends the earned Regeneration Credits to purchase by Producer recycled materials. The purchase options right now are three:

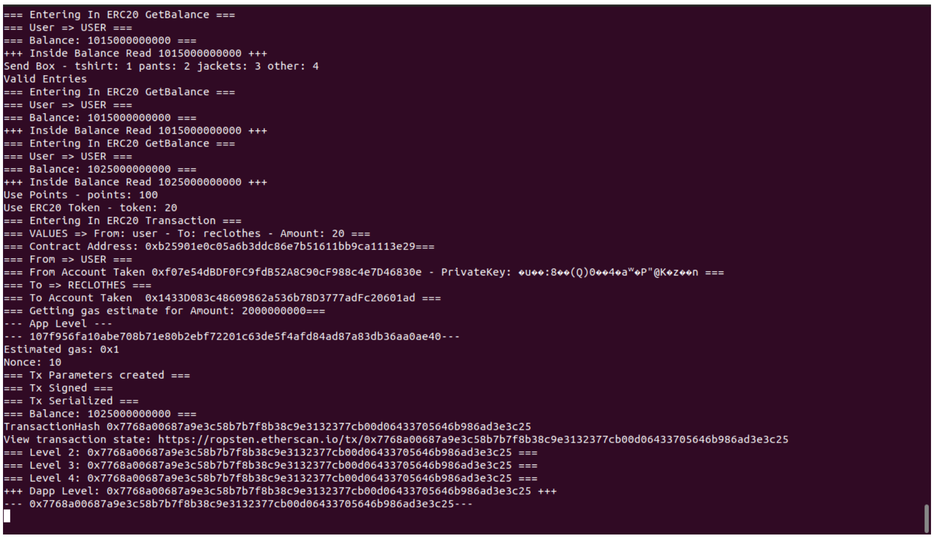


Figure 4.6. Transaction from User to Reclothes

- **Small Box**: 50 Regeneration Credits for 5 upcycled items.
- **Medium Box**: 150 Regeneration Credits for 15 upcycled items.
- **Big Box**: 200 Regeneration Credits for 40 upcycled items.

Send Old Material and Evaluation

The **Figures** 4.7 shows the log of the send old clothes process. In that case is sent a box with inside:

- **t-shirt**: 10
- **pants**: 20
- **jacket**: 10
- **other**: 10

Once the box is sent, the evaluation process starts. The Producer evaluates materials received and issue Regeneration Credits amount that Admins could spend when need, to purchase recycled items. The **Figure ??** shows the page used to perform evaluation Process by Producer. In that case I set a Regeneration Credits amount of 1200. The **Figure** 4.9 shows the output of the evaluation process.
Once the evaluation process is archived and the Regeneration Credits is sent from Producer to Admin. The **Figure** 4.9 shows the info update of the Admin for Producer.

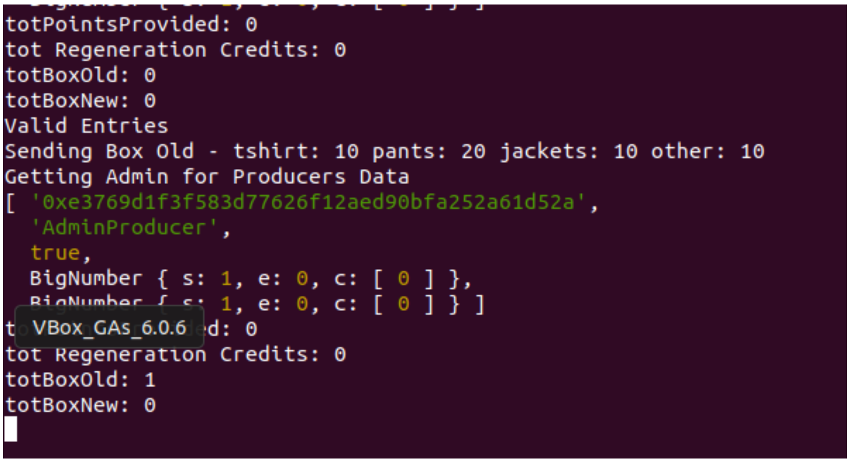


Figure 4.7. Admin Send old clothes

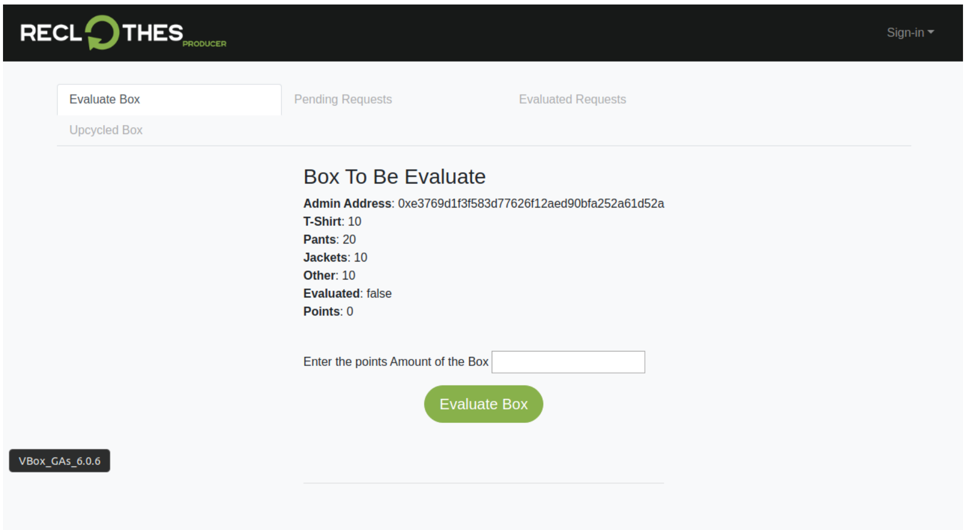


Figure 4.8. Box to be evaluated

Purchase Upcycled Material

Once the Admin sent the box with old clothes and the evaluation process is archived, Admin has a Regeneration Credits amount to spend purchasing upcycled clothes by Producer and then resell them inside the platform store. In our test case, we purchase a **Middle Box** spending an amount of 150 Regeneration Credits.
The **Figure 4.11** shows the output of the purchase process and the Tot Regeneration Credits update once the purchase box process is performed.

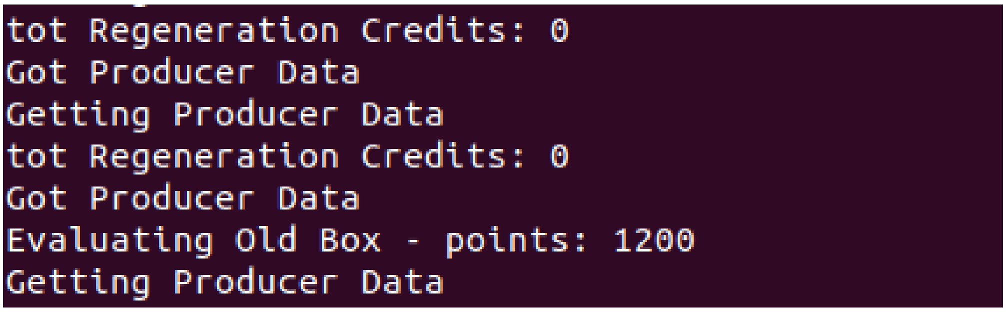


Figure 4.9. Producer Evaluate old materials

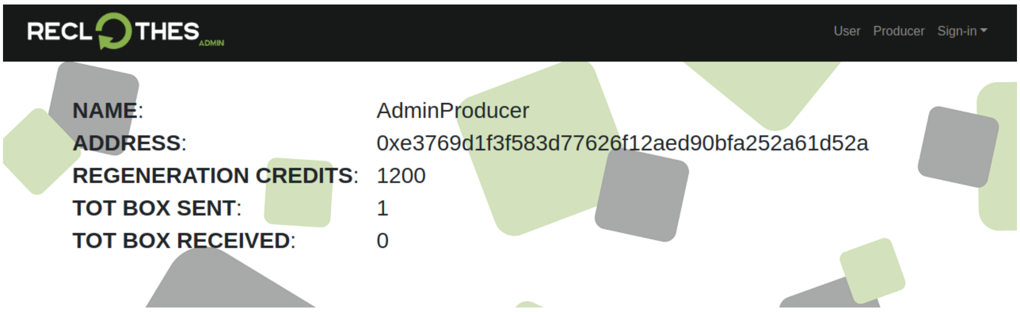


Figure 4.10. Admin Info update

The **Figure** 4.12 shows the update of Producer Information, the circulating Regeneration Credits amount is changed and the Tot Box New number is updated.

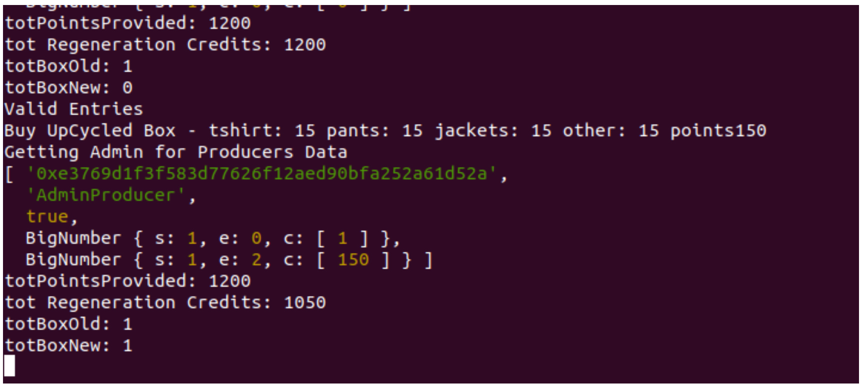


Figure 4.11. Purchase Recycled Clothes

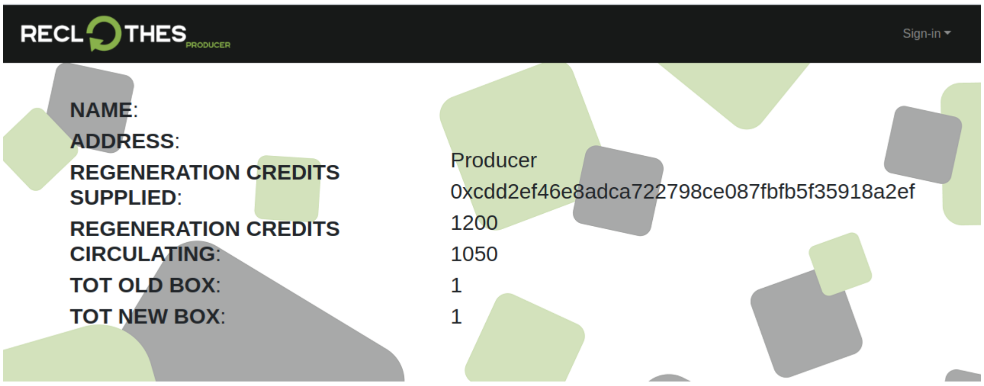


Figure 4.12. Producer Infos Update

Chapter 5

Conclusion

The objective of the thesis was to implement a blockchain-based solution to handle the supply chain and management processes, as lean and transparent as possible. Moreover, the thesis work has to include a cross-chain part between public and private blockchain networks. In particular, a decentralized application has been developed that allows the users to register over the network with specific rights. The cross-chain is implemented at the Application layer, developing specific API endpoints that allow the interoperability of the networks.

For a clearer understanding and overview of all the work produced, I listed each part developed below:

- **Hyperledger Network**: the network architecture is designed based on case needs. It is implemented and produced with a set of scripts that automatize all the setting up and running processes.
- **Smart Contracts**: tree smart contracts, that shape the use case deal, are produced.
- **ERC20 Token**: an ERC20 token in running and accessible over the Ropsten network, is produced.
- **Dapp**: a web-application that integrates both the blockchain used, is produced.

In the end, the thesis archived a **simple managements process** solution, the process is structured in a simplified way, clear and easy to use and manage. Each actor is associated with a specific organization inside the network, taking part in the network's governance with the specific right access. Thanks to the blockchain applications to the thesis case, the **supply chain** process are clearer and transaction-based. Moreover, the actors involved communicating in a good way, keeping a well defined permissioned access to the data. The management, admin side, is improved and it is more transparent. It allows the application to gain credibility by the end-user due to provide the proof of the worked materials in a sustainability way.

The **cross-chain** solution is implemented in the Application layer. To obtain a more modular solution is better to implement the cross-chain at a lower layer, such as at the chaincode side. In that way, the solution could be more adaptable and modular than the developed ones. In any case, to apply that kind of solution it is mandatory the development of other modules such as a storing and mapping mechanism between eth wallet and Fabric identity.

We can conclude that the thesis results archives all the fixed targets. However for **future works** the Hyperledger network architecture could be improved, adding more components, such as adding orderer peers, in order to maximize fault tolerance. The application could be improved in several parts. The integration and the synchronization between the two networks could be handle in a better way, developing additional checks, in order to minimize the fault cases. The API endpoints that handle the crosschain part could be improved adding checks about the wrong parameter passed and failure of just one transaction, Ethereum or Hyperledger, in order to synchronize in the best way both the networks. The cross-chain could be implemented at a lower layer, in order to automatize mechanism such as the accounts mapping between the two blockchains. It makes the solution more pluggable and minimizes developing errors.

Bibliography

[1] *Blockchain Definition*. URL: https://en.wikipedia.org/wiki/Blockchain.

[2] *Byzantine fault tolerance BFT*. URL: https://www.tutorialspoint.com/what-is-byzantine-fault-tolerance#:~:text=The%20concept%20of%20Byzantine%20Fault,values%20to%20misguide%20the%20network.

[3] *Proof of work*. URL: https://en.bitcoin.it/wiki/Proof_of_work.

[4] *Proof of stake*. URL: https://academy.binance.com/blockchain/proof-of-stake-explained.

[5] *51% attack*. URL: https://it.cointelegraph.com/explained/proof-of-work-explained.

[6] *Smart Contract Definition*. URL: https://en.wikipedia.org/wiki/Smart_contract.

[7] Vitalik Buterin. *On Public and Private Blockchains*. 2015. URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/.

[8] Chibuzor Udokwu et al. *The State of the Art for Blockchain-EnabledSmart-Contract Applications in the Organization*. 2018. URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/.

[9] Shiraz Jagati. *Blockchain Interoperability: The Holy Grail for Cross-Chain Deployment*. 2020. URL: https://cointelegraph.com/news/blockchain-interoperability-the-holy-grail-for-cross-chain-deployment.

[10] H. Jin, X. Dai, and J. Xiao. *Towards a Novel Architecture for Enabling Interoperability amongst Multiple Blockchains*. 2018.

[11] *Ark.io*. URL: https://ark.io/.

[12] *Sidechain*. URL: https://en.bitcoin.it/wiki/Sidechain.

[13] Peter Robinson et al. "Atomic Crosschain Transactions for Ethereum Private Sidechains". In: *ArXiv* abs/1904.12079 (2019).

[14] Y. Jiang et al. "A Cross-Chain Solution to Integration of IoT Tangle for Data Access Management". In: (2018).

[15] *Atomic swap*. URL: https://en.bitcoin.it/wiki/Atomic_swap.

[16] *Hash Time Locked Contracts*. URL: https://en.bitcoin.it/w/index.php?title=Hash_Time_Locked_Contracts&source=post_page.

[17] *Lightning Network*. URL: https://lightning.network/.

[18] *BTC Relay*. URL: http://btcrelay.org/.

[19] *Cosmos*. URL: https://cosmos.network/.

[20] *Github repository of ETH2.0*. URL: https://github.com/ethereum/eth2.0-specs.

[21] *Github repository of EVM chaincode for Hyperledger Fabric*. URL: https://github.com/hyperledger/fabric-chaincode-evm.

[22] *Provenance whitepaper Blockchain: the solution for transparency in product supply chains*. 2015. URL: https://www.provenance.org/whitepaper.

[23] Mirko Koscina, Mariusz Lombard-Platet, and Pierre Cluchet. *PlasticCoin: an ERC20 Implementation on Hyperledger Fabric for Circular Economy and Plastic Reuse*. 2019. URL: https://www.researchgate.net/publication/336626518_PlasticCoin_an_ERC20_Implementation_on_Hyperledger_Fabric_for_Circular_Economy_and_Plastic_Reuse.

[24] *PlasticTwist European Project*. URL: https://ptwist.eu/.

[25] Next Nature Network. *The ECO coin: a cryptocurrency backed by sustainable assets*. URL: https://uploads-ssl.webflow.com/5c1b58255c613376879c2558/5c4970105b4d237571564f43_ECOcoin_white_paper_v1.0.pdf.

[26] *Hyperledger Fabric Wallet storage*. URL: https://hyperledger-fabric.readthedocs.io/en/release-1.4/developapps/wallet.html?highlight=wallet.

[27] *Metamask informations*. URL: https://metamask.io/.

[28] *Remix, online editor and compiler*. URL: http://remix.ethereum.org/.

[29] *Merkle Tree*. URL: https://en.wikipedia.org/wiki/Merkle_tree.

[30] *Hyperledger Fabric samples*. URL: https://github.com/hyperledger/fabric-samples.

[31] *Hyperledger Fabric official documentation*. URL: https://hyperledger-fabric.readthedocs.io/en/master/.

[32] *Amazon Managed Blockchain*. URL: https://aws.amazon.com/it/managed-blockchain/.

[33] *Etherscan Gas Tracker*. URL: https://etherscan.io/gastracker.

[34] *EthGasStation: website to track ether and gas prices*. URL: https://ethgasstation.info/.