



POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica (Software Engineering)

Tesi di Laurea Magistrale

Blockchain and smart contracts in the Fashion industry

A Decentralized Application built on Ethereum and Hyperledger Fabric

Relatori

prof.ssa Valentina GATTESCHI
prof. Fabrizio LAMBERTI

Candidato

Stefano FRANZONI

Tutor aziendale

dott. Alfredo FAVENZA

ANNO ACCADEMICO 2019 – 2020

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduction

During this years the Blockchain technology are living a growing era, It found a huge applications and bring important innovations in many environments.

The first use of the blockchain technology in in finance environment, it found a great application in cryptocurrencies, exploiting the decentralized architecture. The overall networks and joint nodes are handled by a Consensus algorithm that perform the verification of execute the upgrade of the distributed ledger.

The two main challanges faced during the documents going to be:

- **Blockchain application in Fashion Environment:** The use cases taked on during the project are related the fashion environments, from resell process over a market place to the managements of the recycling process.
- **CrossChain Solution:** To obtain a good interaction level among the blockchain networks used.

The target to archive at the end of the developed project would like to be:

- **Simple managements process:** Fashion Company side, is required at the end of the work to obtain a simplified model of the overall management process of the transactions and users involved into the system.
- **Supply Chain:** A simplified handling process of the supply chain. The goal is to make more clear possible the transfers of the clothes over the actors involved, from the producing of the items to the sell process.

- **technology improvements and modular solutions:** A crosschain 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.

Capitolo 2

State of the art

2.1 Current state of networks solution

There's three main solution about blockchain network¹:

- **Public blockchains:** a public blockchain is a blockchain that anyone in the world can read, anyone in the world can send transactions to and expect to see them included if they are valid, and anyone in the world can participate in the consensus process - the process for determining what blocks get added to the chain and what the current state is. As a substitute for centralized or quasi-centralized trust, public blockchains are secured by cryptoeconomics - the combination of economic incentives and cryptographic verification using mechanisms such as proof of work or proof of stake, following a general principle that the degree to which someone can have an influence in the consensus process is proportional to the quantity of economic resources that they can bring to bear. These blockchains are generally considered to be "fully decentralized".
- **Consortium blockchains:** a consortium blockchain is a blockchain where the consensus process is controlled by a pre-selected set of nodes; for example, one might imagine a consortium of 15 financial institutions, each of which operates a node and of which 10 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

¹<https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>

of some parts of the blockchain state. These blockchains may be considered "partially decentralized".

- **Fully private blockchains:** a fully private blockchain is a blockchain where write permissions are kept centralized to 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, and so public readability may not be necessary in many cases at all, though in other cases public auditability is desired.

2.1.1 Behind Blockchains

The ideas behind the birth of blockchain technology is to keep informations public and exploit cryptographic algorithms to keep identity secret, maintaining authentication to perform the transaction process. That idea found a huge use cases application, in particular among the cryptocurrencies environment, that's the area in which the public blockchains found more applications. The main rule of the overall system is "**keep it transparent, safe and anonymous**", It mean that all the transactions processed by the public blockchain networks is **transparent** and everyone has read access. The **Safe** concept is the major application of the cryptographic science, for example the bitcoin wallet is based on a key pair based on elliptic curve algorithms. **Anonymous** is strictly correlated to the Safe concept, for the same reason that transactions are safe, the identity could be anonymous, if noone share the identity associated to wallet public key. There's only a public key that correspond to the owner of a wallet with a balance inside in.

On the other hand the majority of the companies, that would like to use blockchains solution for improvement on internal processes, don't want to share and keep own informations public. The goal of the main Companies use cases is to handle internal company processes, such as supply chain process, using a membership mechanism that allow some nodes of the governance to have read or write access to the ledger.

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

1. The main advantages of the **public blockchain** could be fall into two major categories:
 - (a) Public blockchains provide a way to protect the users of an application from the developers, establishing that there are certain things that even the developers of an application have no authority to do. The code is public and everyone could see how it works, on the other hand each identity of the user is protected by cryptographic algorithms.

- (b) Public blockchains are open, and therefore are likely to be used by very many entities and gain some network effects. To give a particular example, consider the case of domain name escrow. Currently, if A wants to sell a domain to B, there is the standard counterparty risk problem that needs to be resolved: if A sends first, B may not send the money, and if B sends first then A might not send the domain. However, if we have a domain name system on a blockchain, and a currency on the same blockchain, then we can cut costs to near-zero with a smart contract: A can send the domain to a program which immediately sends it to the first person to send the program money, and the program is trusted because it runs on a public blockchain. In other words public blockchain fully eliminate intermediary.
2. Compared to the public blockchain the advantages of a **private blockchain** are:
- (a) The consortium or company 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 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 several analysis it get out that the 75% of already implemented projects are designed specifically for private aim². It means that is growing up the need to improve of the Consortium Blockchains that allow a memberships mechanism build for company use cases, it maintain the transactions private in order to grant the

²The State of the Art for Blockchain-EnabledSmart-Contract Applications in the Organization - Chibuzor Udokwu, Aleksandr Kormiltsyn, Kondwani Thangalimodzi, Alex Norta - Department of Software Science Tallinn University of Technology, Tallinn, Estonia

privacy of the business process and data.

On the other hand in some process is usefull to use public blockchain, so is growing up the need to improve the interoperability about consortium and public blockchains into a crosschain solution.

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

Tabella 2.1. Comparing among blockchains features

2.1.2 Overview

Michael Burgess, chief operating officer of Ren said 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.**"

"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."³ , quoting the words of Agarwal, CEO of Persistence.

The main idea of the industry experts is the trade-off concept to obtain a cross-chain interoperability, between public and private. To understand it we are going to remind the strong differences in working foundamental of both blockchains:

- **Public:** It's a peer to peer network in which 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.

³<https://cointelegraph.com/news/blockchain-interoperability-the-holy-grail-for-cross-chain-deployment>

- **Consortium:** The basis idea 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 so strict because the starting hypothesis is different and usually it needs to have a good performance and low latency of the transactions. On the other hand these features could lead to a data manipulation.

2.1.3 Limitation

Now that is clear the pros and cons of each networks the interoperability in some cases it could be the solution of many use cases problem, the public could allow asset transactions among users without limit and granted security and authentication, on the other hand consortium could allow a set of features and informations that users and mostly companies don't want to keep it public. Nevertheless the Achilles heel is:

- **Synchronization:** The both networks must be synchronized and world state must be the same in each moment. It means that each transaction that involves both blockchains before to be archived the both networks must reach a strong sync among the ledgers.
- **Time Effort:** to make it usable it means that the transactions and synchronization must be performed in a reasonable time.
- **Identity:** over the blockchains network the identity of the user or in other word the owner of the wallet is handled in several ways. 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 the authentication mechanism for the user of the network is implemented with x.509 certificates. So another problem is the mapping of these different mechanisms that blockchains implement to allow authentications.

2.1.4 CrossChain Current Solution

This problem and new challenge of crosschain was born a few years ago and there were several solutions that try to fix the problem and allow interoperability. The main ideas to perform interoperability are:

- **New Blockchain:** During the year were born several networks and frameworks that propose to allow the interconnection between public and private blockchains. Many of these solutions are based on new blockchain networks

that are stuctured in order to allow architectural level the interoperability, one of that reality is Ark⁴. But we know that one of the biggest challenge ramain to allow interconnection between the well-known blockchain networks.

- **Architectural Framework:** There are thousand of frameworks proposed over the last years, but the cross-chain isn't still a reality. Nevertheless the main idea that shares the majority of proposed idea is the Sidechain. Introducing a new level between the major layer of the mainnet that allow the mapping, using ad-hoc API, the request from one network to the other one. In a nutshell all the requests from public to private blockchains and vicevere pass by this Sidechain.
- **Atomic Swaps:** allow users to trade one cryptocurrency for another directly in a peer-to-peer transaction Hashed TimeLock Contracts (HTLCs)⁵. 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 coordinate transactions across chains. Atomic swaps can be effective if used correctly and are the mechanism that enables the Lightning Network⁶.
- **Relay:** llow 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 BTCRelay⁷.
- **Merged Consensus:** allow 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 Cosmos⁸ and ETH2.0⁹ use merged consensus.
- **Federations:** allow 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.

⁴<https://ark.io/>

⁵https://en.bitcoin.it/w/index.php?title=Hash_Time_Locked_Contracts&source=post_page

⁶<https://lightning.network/>

⁷<http://btcrelay.org/>

⁸<https://cosmos.network/>

⁹<https://github.com/ethereum/eth2.0-specs>

2.1.5 Chaincode EVM

In our analysis we spend a great attention around Hyperledger and Ethereum, two of the main blockchain solutions used in the world, the first one for permissioned use cases and the second one for public processes. In the last year IBM technical ambassador developed an **EVM chaincode**¹⁰ able to run bytecode of Solidity smart contract over the Hyperledger Fabric network. It isn't a real cross-chain solution but it's a step forward interoperability among blockchains. Of course it still have many limit for the use, in particular for the identity mapping from eth address to fabric identity and vice-versa.

2.2 Blockchain application in Fashion Environment

2.2.1 Provenance case Martine Jarlgaard

Thanks to the blockchain behaviour that allow an immutable record registered for each transaction performed over the supply chain of the items productions. One of the first fashion house that start to use the blockchain technology for own company is Martine Jarlgaard that in 2017, made a partnership with Provenance¹¹ producing clothes with digital tag: The tag could be a QRCode or an RFID readed using NFC technology. That tag provide 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 design of the clothes and final work.

Each actors of the supply chain is a blockchain node that take part at 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.

¹⁰<https://github.com/hyperledger/fabric-chaincode-evm>

¹¹Provenance whitepaper (2015) Blockchain: the solution for transparency in product supply chains. <https://www.provenance.org/whitepaper>

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

2.2.2 Counterfeiting - VeChain BabyGhoast

2.3 Use Case asis

Armadio Verde is an Italian community was born with the aim to share childrens clothes and than once it growing up, it allow to share adult clothes too. The working model is based on the share principle. Every user after is sign up to the platform could book a pick up of own old clothes. The clothes must be in a good state, clean and put in a box. Once the box arrive to Armadio Verde, the clothes going to be checked and evaluated, for each approved clothes is created a dedicated form with all the related information. After the upcycling process the clothes going to be shared over the platform store. The user that send the clothes earn an amount of "star"(the money used over the platform). The star coul be used to purchase other clothes adding a few euro for each item. The clothes that couldn't share over the platform for reselling process, is sent to a certified Onlus.

2.3.1 Sustainability Token

PlasticToken

It's an ERC20 chaincode that run over the hyperledger fabric network¹². It provides functionalities to read and write, with access and rights control, into the distributed ledger. The ERC20 chaincode is the software handling the PlasticTokens in a secure manner. 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 "centralbank" in the current implementation. Once the original PlasticToken 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 run over the Plastic Twist project.

¹²<https://ptwist.eu/>

ECOCoin

The ECO coin 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 user could spend in ECO new sustainable marketplace to buy ecological experiences, services and goods.

It's based on consortium blockchain architecture anche each marketplace that want to involve their business in ECO environment it must be accepted as governance member of the network.

Capitolo 3

Solution

3.1 Oveview

3.1.1 Work Produced

The **Figure ??** show the overview and the main flow of the overall application. The Overall System and the work produced to create the application is composed by the following parts:

- **Networks:** It's the networks over that the blockchains runs. The project involves two kind of networks:
 - **Hyperledger Fabric Network:** It's the main network used for own project.
 - **Ethereum Network:** It's the side network used for token exchanged for own use. In that project is used the testnet Ropsten.
- **Shell Scripts:** to allow to setting up everything in the best way, It's produced a set of shell scripts that run network or shut down it, install chaincode, and run part of the system mandatory for the application use.
- **Smart Contracts:** The smart contracts perform project use cases actions. There's three smart contracts and each of that perform one specific flow and involves just a set of overall actors of the system.
- **Dapp:** It's the web application used to allow actors to interact with the system. It's a decentralized application that communicate with the blockchain networks. Once the user is logged in with related privileges, could perform smart contracts invocation using the web-app.

3.1.2 Actors

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

- **User:** It's the *end user*. It uses the web-app to send old clothes and purchase from Reclothes store. The User is the Actor that starts the entire process flow, *sending the clothes*, mandatory for the entire process.
- **Reclothes Admin:** It's 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, it cares about to handle the recycling and upcycling process, providing to the Producer old materials and spend, when the platform needs, the token received to order recycled clothes.
- **Producer:** It's part of the upcycling process. It receives the materials to perform the recycling process. In our use case test we 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 and the Reclothes Admin during the *Send Old Clothes* process could choose the Producer that prefers.

3.1.3 Application Flow

Each Actor has access to the system with different permissions and privileges. Once the user is logged in, it could access to several features and is allowed to perform a set of actions over the system. For a better understanding we are going to split the overview flow shown in **Figure ??** into 2 subflows 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 that are going to produce transactions modifying the ledgers in an immutable way, adding a new block to the chains, are the following ones:

1. User Side

- (a) User sends Box with old clothes and receives Fabric points and ERC20 Token

- (b) User purchase items inside dapp store using Fabric points and ERC20 Token

2. Producer Side

- (a) Reclothes send clothes box with old matherials and receive Regeneration Credits
- (b) Reclothes spend the Regeneration Credits to purchase upcycled clothes by Producer

That process going to be described in details in the ?? section that analyze deeper the transactions process.

3.1.4 Token exchanged

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

Over Hyperledger Fabric side going to be exchanged two kind of tokens, both are points based integrated with the smart contracts that handle User and Producer side both. The user points are going to be handle totally Reclothes side, it means that it's the Reclothes Admin to decide the amount to be send to the User. To handle the transactions amount and establish a standard behaviour it's need a reference table that set a fixed amount for each clothes received. The producer side points, *Regeneration Credits*, are totally handled by Producer side, there's the producer that receive old materials and than choose the amount to be send. As the user points it needs a table to fix rules for the corrisponding amounts for received materials evaluation.

On the other side, there's the ERC20 token that run over the ethereum network. ERC20 is a standard protocol that allow to implement own token following fixed rules. That standard token includes a set of fixed operations allowed over that, `totalSupply`, `balanceOf`, `transfer`, `transferFrom`, `approve` and `allowance`.

For a better understanding of the exchanged token over own project, It's listed below:

1. Over Fabric Network

- (a) **User Token:** It's a token, points based, used to handle part of payment system related to clothes shipping from User to Reclothes and viceversa.

Solution

- (b) **Regeneration Credits:** It's a token, points based, used to handle part of credit system related to clothes shipping from Reclothes to Producers and viceversa.

2. Over Ethereum Network

- (a) **CO2 Token:** It's and ERC20 Token run over public network in charge to handle part of payments related to clothes shipping from User to Reclothes and viceversa.

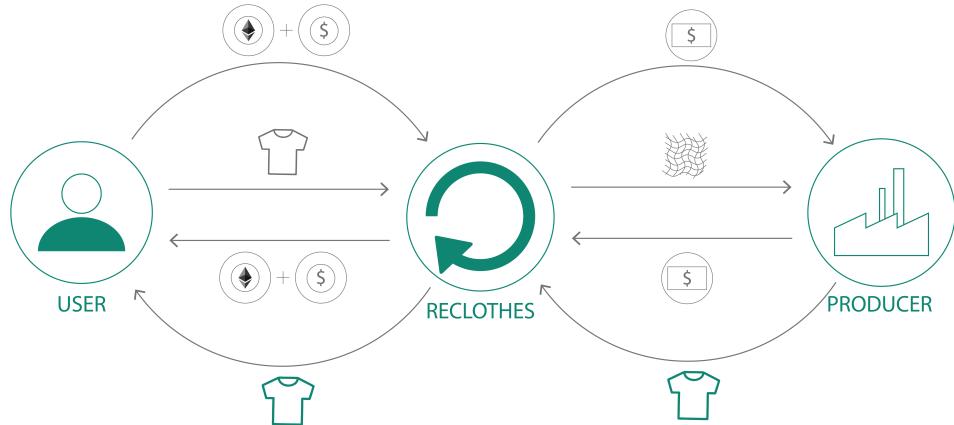


Figura 3.1. UseCase Overview

3.2 Crosschain interaction

3.2.1 Why needs crosschain solution

One of the goal of the project is to build a good integration between the two blockchains networks involved in the system. The needs of a cross-chain solution applied to own use cases is to keep the CO₂ Token exchanged public, in order that for future use could be reused in other environment and applications. In that way the token is not strictly correlated to own personal use, but It could became a standard token to be exchanged over ethereum, and corresponding to an asset related to CO₂ emissions. The behaviour of that token going to be analized better in ?? section.

The requirements to obtain a good integration is to perform cross-chain process without compromise the security issue both side, fabric and ethereum. Therefore we need to care about the technologies behaviour both and what's the technical basis upon which blockchains works.

To understand the solution choosed, to perform the cross-chain interaction, it's need to understand the technologies that we used to develop the project.

3.2.2 Technologies Used

Below there's all the main technologies used, involved in application and crosschain process. The **Figure ??** shows how that technologies and tools is used and interact each other.

- **Metamask:** It's used as ethereum wallet to perform and sign the transactions started by dapp. It grant an high level of security to perform and sign transactions over the ethereum networks. It is integrated in own project, application side, and for the right usage of the entire application, It's mandatory that the user is logged in Metamask over the wallet specified during registration phase.
- **Web3:** It's the software library used to interact with smart contract. Web3.js it's the API that enables the developers to fulfill the integration between website/client and ethereum blockchain. It is a collection of libraries that allow developers to perform actions like send Ether from one account to another, read and write data from smart contracts, create smart contracts, and much more.

- **Fab3 Proxy:** It map the Web3 API with the Fabric SDK in order to interact with Fabric network. It perform a mapping between the Fabric Identity (X.509) with an eth address, generated on the fly, used to perform dapp call. In other word it works like a bridge between ethereum technologies and tools, used for dapp development, and fabric chaincode, that run over the fabric peer and use the GO SDK to allow the chaincode invocation.
- **Fabric Chaincode EVM:** It's the ethereum virtual machine chaincode that allow to run Solidity smart contract over the Fabric network. It's a core part of the entire project. Thanks to that chaincode is allowed to run solidity bytecode over fabric peers.
- **Remix:** is an online editor that allow to develop well structured solidity smart contracts. Thanks to the plugins, that could be installed over the editor, it's possible to compile the wrote smart contracts code. Once the compiling process succed, it produce the corresponding smart contract's bytecode and the smart contract's ABI. Bytecode and ABI both are used to define the smart contract behaviours. That parameters are passed as argument during the deployment process.
- **Expressjs:** Web Framework used to develop web-app and smart contract API. It's a light, easy and fast framework that integrates several methods usefull for HTTP and middleware API development.
- **Infura:** allow to run a Ethereum node in order to set an endpoint used to interact with own contract. It allow in an easy way to set up a public endpoint for own deployed contract address. It deliver personal API and key for the endpoint access. Moreover it provide a really well defined and detailed dashboard to analyze all the smart contract invocations, providing deeper analisys for the called method too.
- **Docker:** The fabric network components run inside Docker containers. It's mandatory for fabric network blockchains, each peer(node) of the network run inside a specific and dedicated container. It allow to be monitored and analized in an indipendent way.

Thanks to the introduction of the EVM chaincode developed by the IBM technical ambassador, it's possible to run Solidity bytecode over the Fabric network. It allow the possibility to adopt ethereum technologies over Hyperledger Fabric Network. That innovation doesn't improve only by the integration network side but client side too, because with the `fabric-chaincode-evm` it's opened a new communication way from dapp/client side to the network side. The Web3 libraries is allowed to handle the smart contract invocation and all the most of the improvements done

in the Ethereum environment, could be used to interact with Hyperledger Fabric world. It means languages, API, libraries and tools that by now founds a huge application in ethereum reality.

The CrossChain solution that I choose to implement, in the following Use Cases, involves the Application Layer. The main core idea of the solution is to mapping, at application level, the ethereum Wallet with Hyperledger Fabric Identity and use one for transactions over ethereum network and the other one over fabric network. Exploiting Web3.js API we invoke ethereum or fabric smart contracts, using that solution all the invocation processes going to be forward, to the corresponding network, starting Dapp Side.

The security is granted Fabric Side using certificate x.509, the authentication mechanism doesn't change. Once the user is authenticated and recognized by own x.509 certificate, fabric network logged the user in to the platform and give him the access to the data informations and all the related privedges based on the actor Role.

On the other hand ethereum side is handled in the following way, the own ethereum public address is specified during registration phase and saved over fabric chaincode to the corrisponding User data structure. When the user is involved inside transaction processes, all the transactions reffer to the public ethereum address reported during registration phase.

Therefore when there's an incoming transaction the tokens will be send to the public address reported in User Data infos.

When there's an outgoing transaction, the security is granted thanks to the Metamask integration during the transaction process. At the moment of transactions the sign, that allow to perform transaction, is performed 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 own account could perform the sign of the transaction.

Figure ?? shows The Architectural Flow and how the technologies is used and interact each other.

Metamask is used as ethereum wallet to sign transaction over ethereum network, In the entire project we suppose that the Actor is logged in over Metamask account reported during registration phase. It is collocated at browser layer of the architectural flow.

Furthermore, fabric side the actor is logged over fabric network using standard fabric authentication process, spending own x.509 certificate. Therefore the Dapp client show the access to the actor page. The dapp client use Fab3 Proxy to map the identity from eth address to fabric identity x.509 certificate and forward request to fabric network, that process is independent by the ethereum address specified during registration phase and doesn't interact with that. Fab3 allow to use `fabric-chaincode-evm` and run solidity code over fabric network, It perform a mapping process among the received requests dapp side. Fab3 receive the Web3 request and map it using the GO SDK in order to forward in the right way all the request to the Fabric peer.

Moreover the Dapp client talk 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 are going to be forward over ethereum network.

Solution

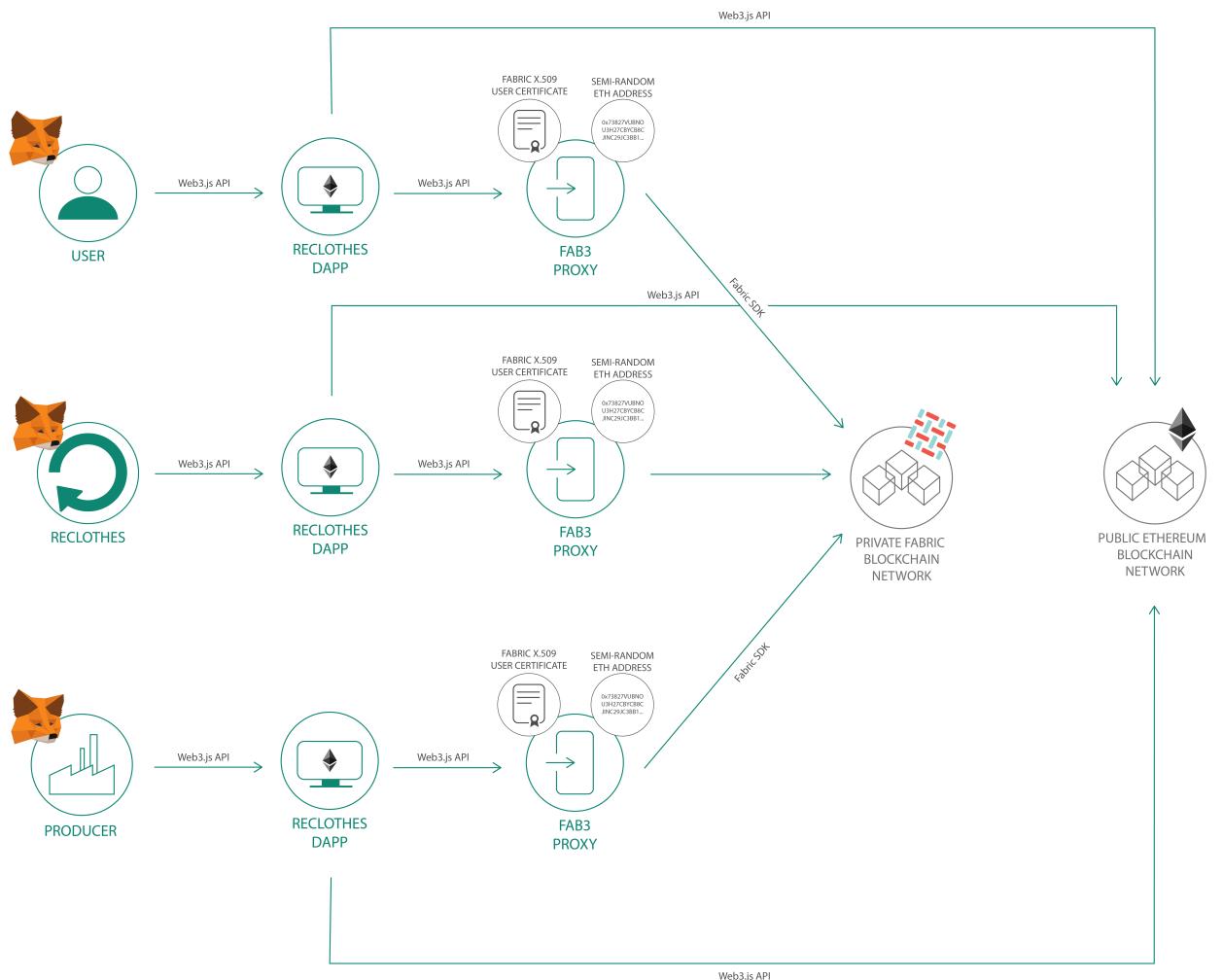


Figura 3.2. Architectural Flow

3.3 Use Cases

As specified in the previous sections, for a better outline we are going to split the use cases inside the **User Side** and the **Producer Side**. The main actor of the system remain Reclothes Admin that is linked to the both side and interacts with all the other actors in order to supply the management support that allow the entire system works.

3.3.1 UseCase 1 - User Side

As shown in **Figure ??** both Actors User and Reclothes Admin, once is logged in, access to a set of features. The Use case diagram show all the action that both users could perform over the networks and the flows that each actions follow. The features are split over the two network, the Fabric one and the Ethereum one. All the flows start from one of the two actors involved and at the end merge to the one of the two blockchain networks. Each actor has a dedicated Ethereum wallet used for ethereum token transaction.

We are going to analyze all the actions that users coul perform over the system:

- **Actions in common**

- **Registration:** The registration phase involves the actor that fill a form with all the mandatory data. To proceed to a successfull registration process it's mandatory that the actor owns the appropriate x.509 certificate, released by the certification authority related to the Role in which the user try to sign up. For example User has a specific Certification Authority that is different by the Reclothes CA.
- **Sign In:** The sign in is authomatic once the fabric network recognize the certificate and logged in as the correspondic actor, once is logged in, the chaincode is invoked and going to read the ethereum address used for the registration phase and provide the access to the methods. In other world fabric certificate provide the access to the network (peers, channels and ledger), instead the ethereum address used provide the access to the smart contract.

- **User Operations**

- **Read Operations**

- * **View own transactions:** The User, once is logged in, could view all the own transactions processed by the network, with a flag that

show transaction status. The transactions includes token exchanged over the network and box request sent to Reclothes. It give the possibility to monitor and manage each process in which user is involved.

- **Write Operations**

- * **Send Box:** It's the starting point of the overall application flow. In the following subsection we going deeper in order to explain how that process works and what transactions depends by that.
- * **Purchase Items:** It's a write operations, belong of that start a transaction process. in that process is involved both the networks. Even that is explained deeper in the followinf subsection.

- **Reclothes Admin Operations**

- **Read Operations**

- * **View all transactions:** The Reclothes Admin, once is logged in, could view all the transactions processed by the network related to all the users involved, with a flag that show transaction status. The trasactions includes token exchanged over the network. It give the possibility to monitor and manage each process in which there's 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 include all the relevant data, moreover it includes a flag that specify the status of the request, that flag could be **Pending**, **Evaluated**.

- **Write Operations**

- * **Evaluate Box:** Even this process belong to write operation, beacuse it start a transaction process that going to be write the blockchain world state.

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

1. User send box with old clothes
2. Reclothes Admin receive box, evaluate it
3. The web app perform the payments from Reclothes Account to User Account
4. Once both transactions succed, both token are accredited and User could spend it

Transactions

In that first Use Case are involved both the blockchain networks. The main part and the most critical one is the transaction action. Considering always *Reclothes Admin* the main actor of the system, there's two kind of transaction in which admin is involved. The **outgoing** transaction that starting by **Evaluation** process performed by the Reclothes Admin once it receive the clothes box sent by the user. The other one is the **incoming** transaction, in that case the token are exchanged from the User to the Reclothes Admin. The action that start 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 we told in the previous section the main and first one action is the **Send Box** that involve the **Evaluation**. The Evaluation is the first outgoing transaction process over the system. Once the token are moved from the Reclothes Admin, the User is allowed to use application and purchase items over it.

1. The **Evaluation** process works in the following way:
 - (a) Reclothes Admin visualize the next pending request to be evaluated. The Admin visualize all the related informations associated to the box request: **userAddress** it's the ethereum user address of the sender, **tshirt**, **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 evaluate it. For a better evaluation process is proposed a solution based on a reference table with a fixed amount for each items, related to the clothes status. At that points there's a filtering process, each items inside the box, is filtered based on platform criteria. Than the Admin decide the status of the clothes and its final desination (platform store or recycling material). Once that the overall clothes was been evaluated and there's been set a total amount value of Fabric points and ERC20 Token, the transaction process could 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 are send over Ethereum network, during the project development we are going to use the Ropsten testnet to exchange the token. There's a previous step before performing token transaction, the fabric chaincode is invoked in order to obtain the ethereum address related to the sender box user. Once that the fabric chaincode return the ethereum account, stored in the smart contract during

the User Registration Phase, the application perform the transfer of the ERC20 token from the Reclothes wallet to the User ethereum wallet

- (c) Once both transactions succed, both the transaction return to the application and it's performed an additional check in order to synchronize both transactions. Tokens are accredited and informations about balances are going to be update. From that moment the User could spend the received tokens over the platform store, performing purchasing.
2. The **Purchase Items** process works in a similar way but inverting the previous flow:
- (a) User choose the items to purchase over the web-app store. The items (tshirt, pants, jacket or other) is represented with the related form reporting all the relevant informations. over the chaincode the smart contract store a dedicated data structure for storing clothes data informations. The related price is expressed through tokens, fabric tokens and CO₂ tokens both.
 - (b) Once the items is choosed, start the purchase process. The User send the fabric tokens over fabric network and the CO₂ token over ethereum network. First of all there's been executed a set of controls in order to check both the balances and evaluate if the User could perform the purchase transaction. Once that all the check is passed correctly the both transaction starts each over own network. Once the transfer process is performed the smart contracts return the operation results to the dapp, that communicate with a message the results of the operations.
 - (c) If the transfers succed, both the tokens balance going to be update and the User could continue to perform actions over the platform.

3.3.2 UseCase 2 - Producer Side

The Use Case 2 is related to the right side of the overall flow schema shown in Figure ???. It shape 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 ???. In this case all the features are performed over the Hyperledger Fabric network, so there's no a crosschain part. The token exchanged, **Regeneration Credit**, is based over fabric smart contract and it's point based, without the needs to involve the ethereum blockchain.

Going to analyze the Figure ???, even here there's the main action that leads to a transaction process. Looking at the diagram we could split the flow into two

Solution

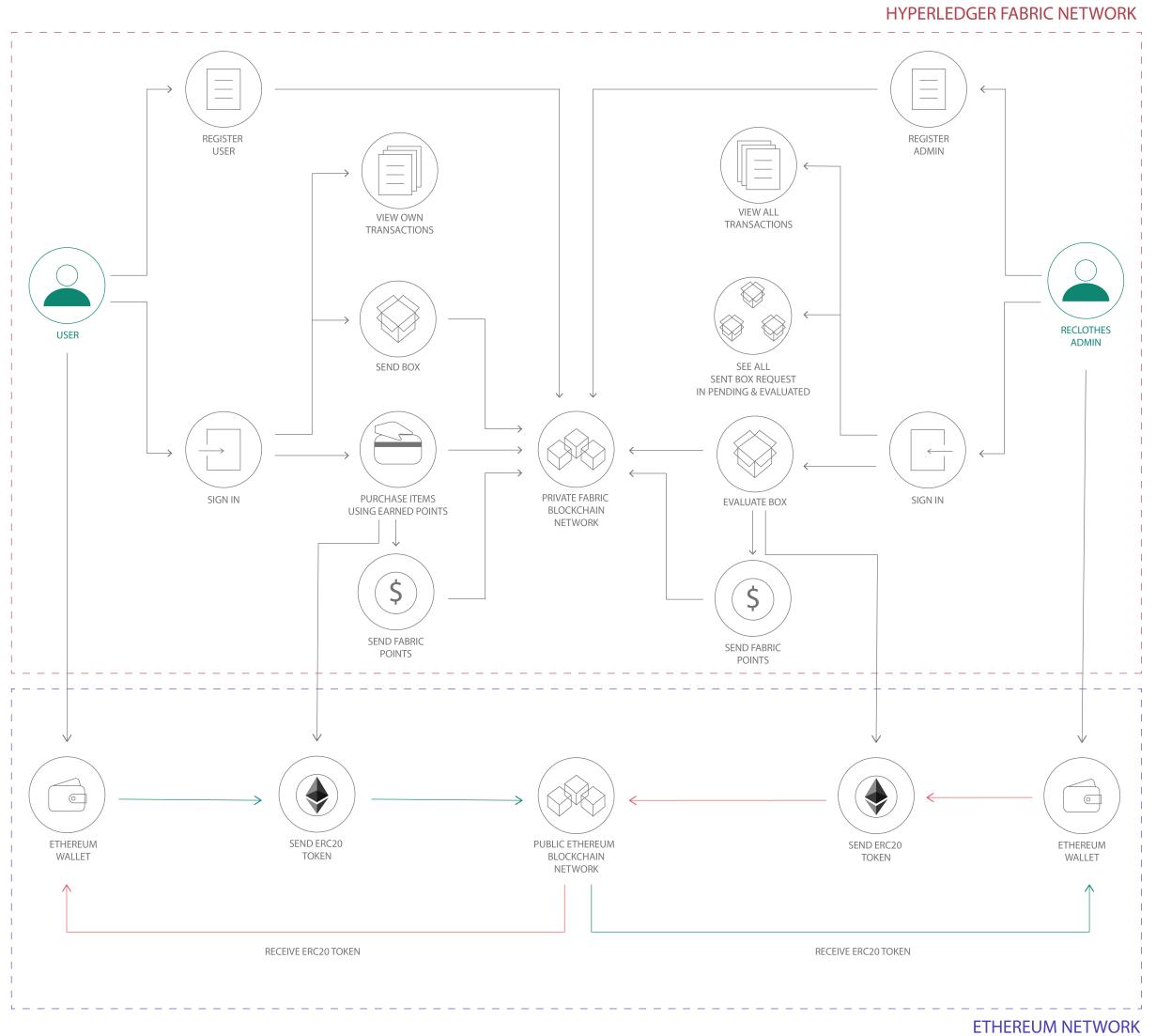


Figura 3.3. UseCase 1

subflow, the first one from Recclothes to Producer. In that subflow we could identify two main actions *Send Box* and *Purchase Box*, as specified in the previous use case, these are the operation that perform a world state update of the blockchain ledger. on the other hand, in the second subflow, from Producer to Recclothes, is involved just one action that produce al outgoing token transfer, the *Evaluate Material* function.

All the asset exchange is handle using **Regeneration Credits** a Fabric token exchanged and handle by the Fabric chaincode and running over Fabric network.

To test our use case we consider just one Producer that perform 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 case of many Producers us involved in the recycling process, an ERC20 integration to handle the Regeneration Credits exchanged could be an improvement, moreover it could not strictly correlated that credits with Reclothes, but the Producers could use it to handle this process with more clients.

1. from Reclothes to Producer

(a) Send Box

- i. The Reclothes Admin after the filtering process performed over the clothes box received by the Users, all the clothes in a bad status that couldn't be resell inside the platform store, are send to the Producer in order to recycle the material and produce upcycled clothes. The Admin performs the Send Box operation, as the Send Box performed in the Use Case 1, it contains the same data inside the request (**tshirt**, **pants**, **jacket**, **other** with related number of items). the box are going to be send to the Producer Company. In 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 could spend that credits to purchase items.

(b) Purchase Box

- i. Reclothes Admin could purchase boxes by Producer Company with inside clothes relized with recycled materials. At the moment there's 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.*

Once that Reclothes Admin choose the box size to order, start the transaction process, the chaincode is invoked. After previous check to control the balance related to the Regeneration Credits owns by Reclothes account, than is invoked the transfer method of the smart contract, the Regeneration Credits are going to redeemed to the Producer account and start the shipping of the Box containing the recycled clothes.

2. from Producer to Reclothes

(a) Evaluate Material

i. Once the box sent by the Reclothes Admin arrive, must be evaluated. The Evaluation process consists in evaluate all the materials related to the clothes received. To obtain an evaluation standart there's a reference table handle a fixed amount of regeneration credits provided for each clothes based one *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, start the transaction process. The chaincode is invoked and the transaction is performed from Producer account to Reclothes account over Fabric network. Producer side there's two parameters that could be analyzed:

- A. **Regeneration Credits Supplied:** It's the total amount of credits emitted over the time.
- B. **Regeneration Credits Circulating:** It's the amount of credits that Reclothes Admin owns and could spend for purchasing.

Solution

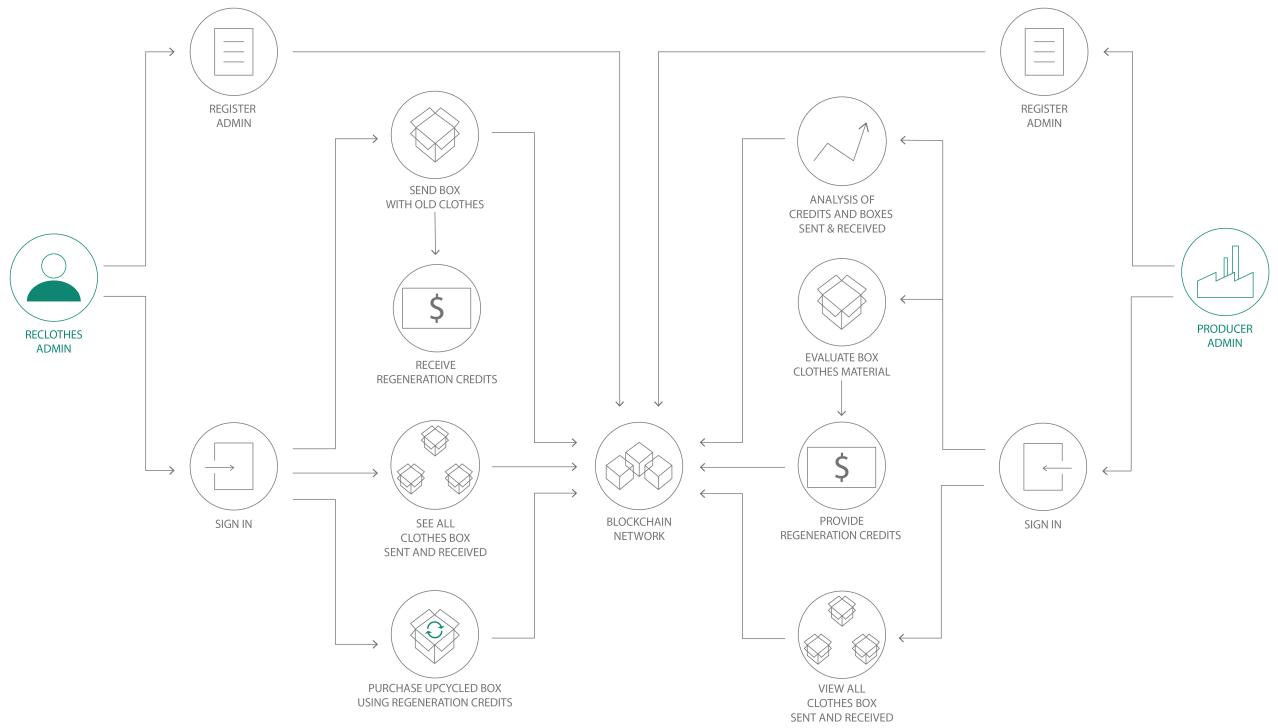


Figura 3.4. UseCase 2

3.4 Smart Contract

For the smart contract developments we exploit the `fabric-chaincode-evm`¹, it allow to run ethereum smart contract bytecode inside an Hyperledger Fabric peer. Therefore evm chaincode allow us the development of the smart contract in Solidity or Vyper programming languages.

For the development it's user **Remix**², It's an online editor that allow to write and compile Solidity smart contracts code, providing all the Solidity version compiler. Once that the smart contract code is wrote and the `.sol` file is produced, the compiling process produce two files mandatory for the deployment and use of the smart contract over fabric network. The two file produced are:

¹To run Solidity Contract over Fabric Network, It's used `fabric-chaincode-evm`, an ethereum virtual machine chaincode developed by IBM developers, to allow the integrations there's the need of additional components such as Fab3 Proxy

²<http://remix.ethereum.org/>

- **ABI:** The *Application Binary Interface* is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and 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 word The ABI is the description of the contract interface. It contains no code and cannot be run by itself. It's 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's an hexadecimal representation of the final contract. It use the ABI to find the context of the behind contract logic.

To handle the overall system was produced three smart contract in Solidity. Two of that run over the Hyperledger Fabric network, exploiting the membership mechanism to access of the chaincode. In other word the permissioned mechanism behind the logic is performed both networks side and chaincode side. The network side filter at 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 developed run over the Ethereum network, for that project It's used the Ropsten testnet. The access to the contract, in this case, is provided by the contract address generated during the deployment phase.

For a better view below we list all the contract involved in the system:

1. Hyperledger Fabric

- (a) **User Contract:** handle the User side, registration and interactions phase. That contract shapes the Use Case 1 functionalities. There's the dedicated data structures that cares about storing of actors involved(**User** and **Admin**). It provide a set of getter and setter methods and that a couple of function that leads to a transaction process.
- (b) **Producer Contract:** handle the interaction from Reclothes to Producers. That contract shapes the Use Case 2 functionalities. There's the dedicated data structures that cares about storing of actors involved(**Admin** and **Producer**). It provide a set of getter and setter methods and that a couple of function that leads to a transaction process.

2. Ethereum

- (a) **ERC20 Contract:** it's a standard smart contract with a Max Supply fixed to 1.000.000. The contract is stuctured following the ERC20 standard. It's not correlated to this project and it's exchangeable amoung user that owns an ethereum wallet. The Contract is deployed over Ropsten network and is accesible using the public contract address. We access to it 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 part of Use Case 1.

Data Structure

In that contract we are going to store all the transactions informations related to the points and clothes box transactions. Moreover we store the informations of the actors involved in the system, that registration phase perform an additional control 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's going to be a better explanation of Fab proxy module and its use.

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

1. **User:** model all users data
2. **Admin:** model Reclothes Admin data
3. **PointsTransaction:** Model transactions data and incorporate `TransactionType` used to identify the flows
4. **ClothesBox:** The box sent with old clothes

```
1 // model a user
2 struct User {
3     address userAddress;      // User address (inside fabric
4                           environment)
5     address publicAddress;   // external eth public address
6                           of User Admin
7     string firstName;
8     string lastName;
9     string email;
```

Solution

```
9         uint points;           // Fabric points amount
10        bool isRegistered;    // Flag for internal use
11        uint numTransaction;  // number of transactions
12        performed
13        mapping(uint => PointsTransaction) userTransactions;
14        uint numBox;           // number of box transaction
15        evaluated
16        mapping(uint => ClothesBox) box;
17
18    // model a admin
19    struct Admin {
20        address adminAddress;   // Admin address (inside fabric
21        environment)
22        address publicAddress; // external eth public address
23        of Admin
24        string name;
25        bool isRegistered;     // Flag for internal use
26
27    }
28
29    // model points transaction
30    enum TransactionType { Earned, Redeemed }
31    struct PointsTransaction {
32        uint points;
33        TransactionType transactionType;
34        address userAddress;   // user address involved
35        address adminAddress; // admin address involved
36
37    }
38
39    // model clothes box to ship
40    struct ClothesBox {
41        address userAddress; // reclothes-producer Admin
42        uint tshirt;          // Number of item
43        uint pants;           // Number of item
44        uint jacket;          // Number of item
45        uint other;           // Number of item
46        bool isEvaluated;     // Flag to check if box evaluation
47        is performed
48        uint points;           // fabric value amount of the box
49    }
50
```

Getter

The User Contract allow to access to a set of method to obtain informations of the system status. Once the user is logged in as User or Admin could perform part of that getter invocation. Parts of the method are developed for an internal usage, the other ones are dedicated to provide to the actors information about the system, or is usefull to start other invokations dapp side. The access to some

Solution

method are handled using modifier method that perform a filtering process of the function caller.

```
1  /************************************************************************/
2  /* **** Users Data ****/
3  /* **** *****/
4  //get User eth public address
5  function getUserEthAddress() onlyUser() public view returns(
6      address ethAddress){
7      return users[msg.sender].publicAddress;
8  }
9
10 //get Reclothes Admin eth public address
11 function getAdminEthAddress() onlyAdmin() public view returns(
12     address ethAddress){
13     return admins[msg.sender].publicAddress;
14 }
15 /* **** All Box Requests -> Old , Evaluated , UpCycled ***/
16 /* **** *****/
17
18 //Get PendingBox by index
19 function getPendingRequest(uint _pendingIndex) public view
20     returns(address, uint, uint, uint, uint, bool, uint) {
21     // only admin can call
22     require(admins[msg.sender].isRegistered, "Admin address not
23         found");
24
25     //check index
26     require(_pendingIndex<pendingIndex, "Wrong index");
27
28     return (pendingBox[_pendingIndex].userAddress, pendingBox[
29         _pendingIndex].tshirt, pendingBox[_pendingIndex].pants,
30         pendingBox[_pendingIndex].jacket, pendingBox[
31         _pendingIndex].other, pendingBox[_pendingIndex].
32             isEvaluated, pendingBox[_pendingIndex].points);
33 }
34
35 //Get Next Pending Request
36 function getNextPendingRequest() onlyAdmin(msg.sender) public
37     view returns(address, uint, uint, uint, uint, bool, uint) {
38     //check index
39     require(evaluatedIndex<pendingIndex, "No More Pending
40         Request");
```

Solution

```
35         return (pendingBox[evaluatedIndex].userAddress, pendingBox[  
36             evaluatedIndex].tshirt, pendingBox[evaluatedIndex].  
37             pants, pendingBox[evaluatedIndex].jacket, pendingBox[  
38             evaluatedIndex].other, pendingBox[evaluatedIndex].  
39             isEvaluated, pendingBox[evaluatedIndex].points);  
40     }  
41  
42     //Get EvaluatedBox by index  
43     function getEvaluatedRequest(uint _evaluatedIndex) public view  
44         returns(address, uint, uint, uint, uint, bool, uint) {  
45         // only admin can call  
46         require(admins[msg.sender].isRegistered, "Admin address not  
47             found");  
48  
49         //check index  
50         require(_evaluatedIndex<evaluatedIndex, "Wrong index");  
51  
52         return (evaluatedBox[_evaluatedIndex].userAddress,  
53             evaluatedBox[_evaluatedIndex].tshirt, evaluatedBox[  
54             _evaluatedIndex].pants, evaluatedBox[_evaluatedIndex].  
55             jacket, evaluatedBox[_evaluatedIndex].other,  
56             evaluatedBox[_evaluatedIndex].isEvaluated, evaluatedBox[  
57             _evaluatedIndex].points);  
58     }  
59  
60     function getTransactionInfo(uint _transactionIndex) onlyUser(  
61         msg.sender) public view returns(uint, uint, address,  
62         address) {  
63         //require index exists  
64         require(users[msg.sender].numTransaction >  
65             _transactionIndex && _transactionIndex >= 0, "Wrong  
66             transaction index");  
67  
68         return (users[msg.sender].userTransactions[  
69             _transactionIndex].points, uint(users[msg.sender].  
70             userTransactions[_transactionIndex].transactionType),  
71             users[msg.sender].userTransactions[_transactionIndex].  
72             userAddress, users[msg.sender].userTransactions[  
73             _transactionIndex].adminAddress);  
74     }  
75  
76     //return box requests number  
77     function getUserBoxNum() onlyUser(msg.sender) public view  
78         returns(uint) {  
79         return users[msg.sender].numBox;  
80     }  
81  
82     //Get UserBox by index
```

```
62     function getUserRequest(uint _index) onlyUser(msg.sender)
63         public view returns(address, uint, uint, uint, uint, bool,
64         uint) {
65             //check index
66             require(_index<users[msg.sender].numBox, "Wrong index");
67
68             ClothesBox memory box = users[msg.sender].box[_index];
69
70             return (box.userAddress, box.tshirt, box.pants, box.jacket,
71                     box.other, box.isEvaluated, box.points);
72         }
73
74     function getAdminTransactionInfo(uint _transactionIndex)
75         onlyAdmin(msg.sender) public view returns(uint, uint,
76         address, address) {
77         //require index exists
78         require(totTx > _transactionIndex && _transactionIndex >=
79                 0, "Wrong transaction index");
80
81         return (usersTransactions[_transactionIndex].points, uint(
82             usersTransactions[_transactionIndex].transactionType),
83             usersTransactions[_transactionIndex].userAddress,
84             usersTransactions[_transactionIndex].adminAddress);
85     }
86
87     //return tot transaction number
88     function getTotTransactionNum() onlyAdmin(msg.sender) public
89         view returns(uint) {
90             return totTx;
91         }
```

Transactions

The transactions process are the main process of the overall smart contract. That method perform a write access to the smart contract and going to be to modify the world state of the ledger stored over the Fabric blockchains peers.

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

1. **earnPoints**: It's an internal function called by EvaluateBox function. Once that user performed the sendBox process, start the evaluation process, Admin side. Therefore the admin evaluate the pending request and set a total amount of points related to the box received. That the EvaluateBox function call the internal function **earnPoints** passing as argument the amount to be transfer and the userAddress of the clothes box sender. Than the function performs the fabric points transaction from Reclothes to User.

2. **usePoints**: It's related to the purchase process. When the User perform a purchase over the platform store, there's be calculated the overall amount related to the items purchased and internally is invoked the **usePoints** function. That function after a set of previous checks than going to decrease the user balance of the related amount passed to the function.

```
1  /************************************************************************/
2  /* ***** Transactions Operations *****/
3
4
5
6  //update users with points earned
7  function earnPoints (uint _points, address _userAddress )
8    onlyAdmin(msg.sender) internal {
9
10    // verify user address
11    require(users[_userAddress].isRegistered, "User address not
12      found");
13
14    // update user account
15    users[_userAddress].points = users[_userAddress].points +
16      _points;
17
18    PointsTransaction memory earnTx = PointsTransaction({
19      points: _points,
20      transactionType: TransactionType.Earned,
21      userAddress: _userAddress,
22      adminAddress: admins[msg.sender].adminAddress
23    });
24
25    // add transction
26    transactionsInfo.push(earnTx);
27
28    users[_userAddress].userTransactions[users[_userAddress].
29      numTransaction] = earnTx;
30    users[_userAddress].numTransaction++;
31
32
33    //Update users with points used
34    function usePoints (uint _points) onlyUser(msg.sender) public {
35
36      // verify enough points for user
37      require(users[msg.sender].points >= _points, "Insufficient
38        points");
```

Solution

```
39     // update user account
40     users[msg.sender].points = users[msg.sender].points - _points
41     ;
42
43     PointsTransaction memory spendTx = PointsTransaction({
44         points: _points,
45         transactionType: TransactionType.Redeemed,
46         userAddress: users[msg.sender].userAddress,
47         adminAddress: 0
48     });
49
50     // add transction
51     transactionsInfo.push(spendTx);
52
53     users[msg.sender].userTransactions[users[msg.sender].
54         numTransaction] = spendTx;
55     users[msg.sender].numTransaction++;
56
57     usersTransactions[totTx] = spendTx;
58     totTx++;
59
60     /****** Clothes Box Operations ******/
61
62     //handle box
63     function sendBox(uint _tshirt, uint _pants, uint _jackets, uint
64         _other) onlyUser(msg.sender) public {
65
66         pendingBox[pendingIndex] = ClothesBox({
67             userAddress: msg.sender,
68             tshirt: _tshirt,
69             pants: _pants,
70             jacket: _jackets,
71             other: _other,
72             isEvaluated: false,
73             points: 0
74         });
75
76         users[msg.sender].box[users[msg.sender].numBox] =
77             pendingBox[pendingIndex];
78
79         users[msg.sender].numBox++;
80         pendingIndex++;
81     }
82
83     //evaluate box
84     function evaluateBox(uint _points) onlyAdmin(msg.sender) public
85     {
```

```
84         //check correct pending request index
85         require(evaluatedIndex < pendingIndex, "No more pending
86             request");
87
88         //check if evaluation is done
89         require(!pendingBox[evaluatedIndex].isEvaluated, "Request
90             just evaluated");
91
92         //pop pending request
93         ClothesBox storage box = pendingBox[evaluatedIndex];
94
95         //update box transaction
96         box.isEvaluated = true;
97         box.points = _points;
98
99         //send points to the userAddress
100        earnPoints(_points, box.userAddress);
101
102        //add evaluated box
103        evaluatedBox[evaluatedIndex] = box;
104        evaluatedIndex++;
105    }
106
107    function getBalance() public view returns (uint) {
108        return users[msg.sender].points;
109    }
```

3.4.2 Producer Contract

The Producer Contract contains all the features described in the Use Case 2 diagram shown in the Figure ??.

Data Structure

In that contract we are going to store all the transactions informations related to the points and clothes box transactions. Moreover we store the informations of the actors involved in the system, in this case the actors involved going to be **Admin** and **Producer**. As the previous contract the registration phase provide an additional control over the actors 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.

Briefly explaining the behaviour of relationship among contracts, the fab proxy has a 1 to 1 association instance/user. There's the possibility that the Admin

logged and registered, over `UserContract`, associated to one fab3 instance, setted over the channel that communicate with `UserContract`, must to perform another registration with a new fab3 proxy instance setted to communicate with the channel dedicated for `ProducerContract`. It means that for each fab3 instance there's be a new eth address generated and the Admin could has two ethereum address, one associated to `UserContract` and the other one associated to `PrducerContract`.

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

1. **Producer**: model all producers data
2. **Admin**: model Reclothes Admin data
3. **ClothesBox**: The box sent with old clothes

```
1 // model a producer
2 struct Producer {
3     address adminAddress;    // Producer Admin address (inside
4         fabric environment)
5     address publicAddress;   // external eth public address of
6         Producer Admin
7     string name;             // Producer admin name
8     bool isRegistered;      // Flag for internal use
9     uint numBox;            // number of box transactions
10    evaluated
11    uint pointsProvided;   // amount of points provided by own
12        evaluations
13    mapping(uint => ClothesBox) box;
14 }
15
16 // model a admin
17 struct Admin {
18     address adminAddress;    // Admin address (inside fabric
19         environment)
20     address publicAddress;   // external eth public address of
21         Admin
22     string name;             // Admin name
23     bool isRegistered;      // Flag for internal use
24     uint numBox;            // number of box transaction
25     evaluated
26     uint creditSpent;       // amount of points provided by own
27         evaluations
28     mapping(uint => ClothesBox) box;
29 }
30
31 struct ClothesBox {
32     address adminAddress; // reclothes-producer Admin
33     uint tshirt;          // Number of item
```

Solution

```
26     uint pants;           // Number of item
27     uint jacket;          // Number of item
28     uint other;           // Number of item
29     bool isEvaluated;     // Flag to check if box evaluation is
                           performed
30     uint points;          // fabric value amount of the box
31
32     //mapping(uint => Clothes) clothes;
33 }
```

Getter

The Producer Contract allow to access to a set of method to obtain informations of the system status. Once the user is logged in as Admin or Producer could perform part of that getter invocation. Parts of the method are developed for an internal usage, the other ones are dedicated to provide to the actors information about the system, or is usefull to start other invokations dapp side. The access to some method are handled using modifier method that perform a filtering process of the function caller.

Below I reported only the main smart contract methods.

```
1   ****
2   *** All Box Requests -> Old, Evaluated, UpCycled ***
3   ****
4
5
6   function getPendingRequest(uint _pendingIndex) public view
7       returns(address, uint, uint, uint, uint, bool, uint) {
8       //check index
9       require(_pendingIndex<pendingIndex && _pendingIndex>=0, "Wrong index");
10
11      return (pendingBox[_pendingIndex].adminAddress, pendingBox[
12          _pendingIndex].tshirt, pendingBox[_pendingIndex].pants,
13          pendingBox[_pendingIndex].jacket, pendingBox[
14          _pendingIndex].other, pendingBox[_pendingIndex].
15          isEvaluated, pendingBox[_pendingIndex].points);
16 }
```

```
13  function getNextPendingRequest() public view returns(address,
14      uint, uint, uint, uint, bool, uint) {
15      //check index
16      require(evaluatedIndex<pendingIndex, "No More Pending
17          Request");
```

Solution

```
17     return (pendingBox[evaluatedIndex].adminAddress, pendingBox
18         [evaluatedIndex].tshirt, pendingBox[evaluatedIndex].
19         pants, pendingBox[evaluatedIndex].jacket, pendingBox[evaluatedIndex].
20         other, pendingBox[evaluatedIndex].
21         isEvaluated, pendingBox[evaluatedIndex].points);
22 }
23
24 /**
25  * ***** Evaluated Request -> Box with Old Clothes evaluated
26  * *****/
27 function getEvaluatedRequest(uint _evaluatedIndex) public view
28     returns(address, uint, uint, uint, uint, bool, uint) {
29     //check index
30     require(_evaluatedIndex<evaluatedIndex && upCycledIndex>=0,
31             "Wrong index");
32
33     return (evaluatedBox[_evaluatedIndex].adminAddress,
34             evaluatedBox[_evaluatedIndex].tshirt, evaluatedBox[
35                 _evaluatedIndex].pants, evaluatedBox[_evaluatedIndex].
36                 jacket, evaluatedBox[_evaluatedIndex].other,
37                 evaluatedBox[_evaluatedIndex].isEvaluated, evaluatedBox
38                 [_evaluatedIndex].points);
39 }
40
41 /**
42  * ***** UpCycled Request -> Box with New Clothes *****/
43 function getUpCycledRequest(uint _upCycledIndex) public view
44     returns(address, uint, uint, uint, uint, bool, uint) {
45     //check index
46     require(_upCycledIndex<upCycledIndex && upCycledIndex>=0,
47             "Wrong index");
48
49     return (upCycledBox[_upCycledIndex].adminAddress,
50             upCycledBox[_upCycledIndex].tshirt, upCycledBox[
51                 _upCycledIndex].pants, upCycledBox[_upCycledIndex].
52                 jacket, upCycledBox[_upCycledIndex].other, upCycledBox[
53                 _upCycledIndex].isEvaluated, upCycledBox[_upCycledIndex]
54                 .points);
55 }
56
57 /**
58  * ***** Data of Requests *****/
59
60 function getTotPointsProvided() public view returns(uint) {
61     return totPointsProvided;
62 }
63
64 function getRegenerationCredit() public view returns(uint) {
65     return debtPoints;
```

```
48     }
49
50     function getTotBoxOld() public view returns(uint) {
51         return totBoxOld;
52     }
53
54     function getTotBoxNew() public view returns(uint) {
55         return totBoxNew;
56     }
```

Transactions

As always be the transactions process are the main ones of the smart contrancts, leadind to a write operation. .

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

1. **evaluateBox**: The evaluation process start by the invocation of `SendBox` function. Once that there's pending box request, the next one going to be evaluated and following the price table, to evaluate clothes materials by `material type` and `weight`, there's setted an overall amount value corresponding to the clothes box request. The transfer process, considering just one Producer involved in own project, so the points is handled with a `debtPoints` variable that is updated by these two functions. In that case `evaluateBox` going to add the amount value of the box to that `debtPoints` variable.
2. **buyUpcycledBox**: That process leads a purchase order performed by the Admin to the Producer. Admin choose a kind of fixed box(`small`, `medium`, `large`) with a fixed Regeneration Credits price associated. Before performing the purchase process, It's checked the `debtPoints` balance to allow or not the transaction of the box. Once that the amount of regeneration credits is enough to buy upcycled clothes, than the `debtPoints` is updated and the value of the purchased box is subtract to the overall balance.

```
1 // Evaluate Old Box
2
3     function evaluateBox(uint _points) onlyProducer() public {
4         //check correct pending request index
5         require(evaluatedIndex < pendingIndex, "No more pending
6             request");
7
8         //check if evaluation is done
```

Solution

```
8     require(!pendingBox[evaluatedIndex].isEvaluated, "Request
9         just evaluated");
10    //pop pending request
11    ClothesBox storage box = pendingBox[evaluatedIndex];
12
13    //update box transaction
14    box.isEvaluated = true;
15    box.points = _points;
16
17    //add evaluated box
18    evaluatedBox[evaluatedIndex] = box;
19    evaluatedIndex++;
20
21    debtPoints += _points;
22    totPointsProvided += _points;
23 }
24
25 function buyUpcycledBox(uint _tshirt, uint _pants, uint
26     _jackets, uint _other, uint _points) onlyAdmin() public {
27     require(debtPoints >= _points, "Not enough credits
28         accumulated in old material boxes");
29
30     ClothesBox memory box = ClothesBox({
31         adminAddress: msg.sender,
32         tshirt: _tshirt,
33         pants: _pants,
34         jacket: _jackets,
35         other: _other,
36         isEvaluated: true,
37         points: _points
38     });
39
40     admins[msg.sender].box[admins[msg.sender].numBox] = box;
41     admins[msg.sender].numBox++;
42     admins[msg.sender].creditSpent += _points;
43
44     //add upcycled box
45     upCycledBox[upCycledIndex] = box;
46     upCycledIndex++;
47
48 }
```

3.4.3 ERC20 Contract

ERC-20 is a technical standard used to issue and implement tokens on 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's public, accessible to everyone that own an ethereum wallet. The decision as well as a cross-chain interaction process, lead to open the doors to an external usage of the token, due to what the asset represents.

The asset want to represent the CO₂ emission saved. For example as asset exchange to measure the emission saved recycling a tshirt even to produce it starting from scratch.

The main information associated to the created token are:

- **Symbol:** CO2, It's used to identify a token, this is a three or four letter abbreviation of the token.
- **Name:** CarbonToken, are able to identify them.
- **Total supply:** 100000000, It's the max supply of the token.
- **Decimals:** 18, It's used to determine to 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 described by ERC20 interface

```
1  contract ERC20Interface {  
2      function totalSupply() public constant returns (uint);  
3      function balanceOf(address tokenOwner) public constant  
4          returns (uint balance);  
5      function allowance(address tokenOwner, address spender)  
6          public constant returns (uint remaining);  
7      function transfer(address to, uint tokens) public returns (  
8          bool success);  
9      function approve(address spender, uint tokens) public  
10         returns (bool success);  
11     function transferFrom(address from, address to, uint tokens  
12         ) public returns (bool success);
```

```
8     event Transfer(address indexed from, address indexed to,
9         uint tokens);
10    event Approval(address indexed tokenOwner, address indexed
11        spender, uint tokens);
12 }
```

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 is 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 cryptocurrency 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 is important to have an overview of the principal components involved in the Hyperledger Fabric Architecture:

1. **Peer:** It's the fabric node, there's different types of peers and each of them could perform specific actions
 - (a) **Anchor Peer:** this kind of peer is used for communications between organizations. It makes peers in different organizations aware of 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, thanks to that it implements 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:** Is like a network administration point. The ordering nodes support the application channels for ordering transactions, create blocks and add them to the chain.
5. **Organization:** Identify a category of users involved in the network. Each user certificate of the Organizations are 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 allow 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 consiste to the Worls State of the blockchains. All the transactions performed over the chain is merkled in the world state³

3.5.2 Own Architecure

The **Figure ??** show the main components of own network architecture build for the application. 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) *Channel12*: It's the cannel between Org1 and Org2 and allow the comunication between User and Recclothes
 - (b) *Channel23*: It's the cannel between Org2 and Org3 and allow the comunication between Recclothes and Producer
5. **2 Consortiums**
6. **CC12**: related to the channel 1, allow the use of actors owned by Org1 and Org2.
7. **CC23**: related to the channel 2, allow the use of actors owned by Org2 and Org3.

This is a test network, light for test the entire project. Hyperledger Fabric allow to implements in an easy way more components adding orderer or Peers, it make hyperledger architecture highly modular. For production the architecture needs some modification. Adding more orderers and peers for each organizations, in order to maximize the fault tolerance.

³https://en.wikipedia.org/wiki/Merkle_tree

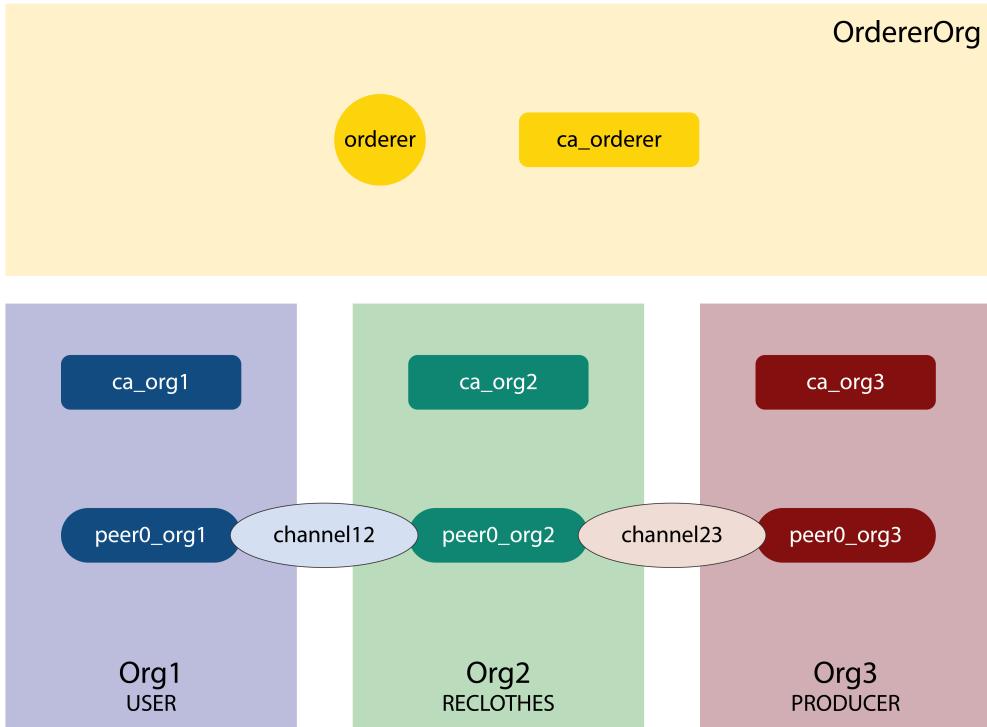


Figura 3.5. Fabric Network

3.5.3 Fabric Network

The **Figure ??** show how components interact each other. We could separate components into 2 categories, inside and outside Fabric Network. First of all we need to describe the components involved :

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

Solution

- **CC:** It's the *Consortium*, It's associated to the channel, manage ownerships and It include a set of Organizations allowed.
- **Docker:** The network components run inside docker containers.

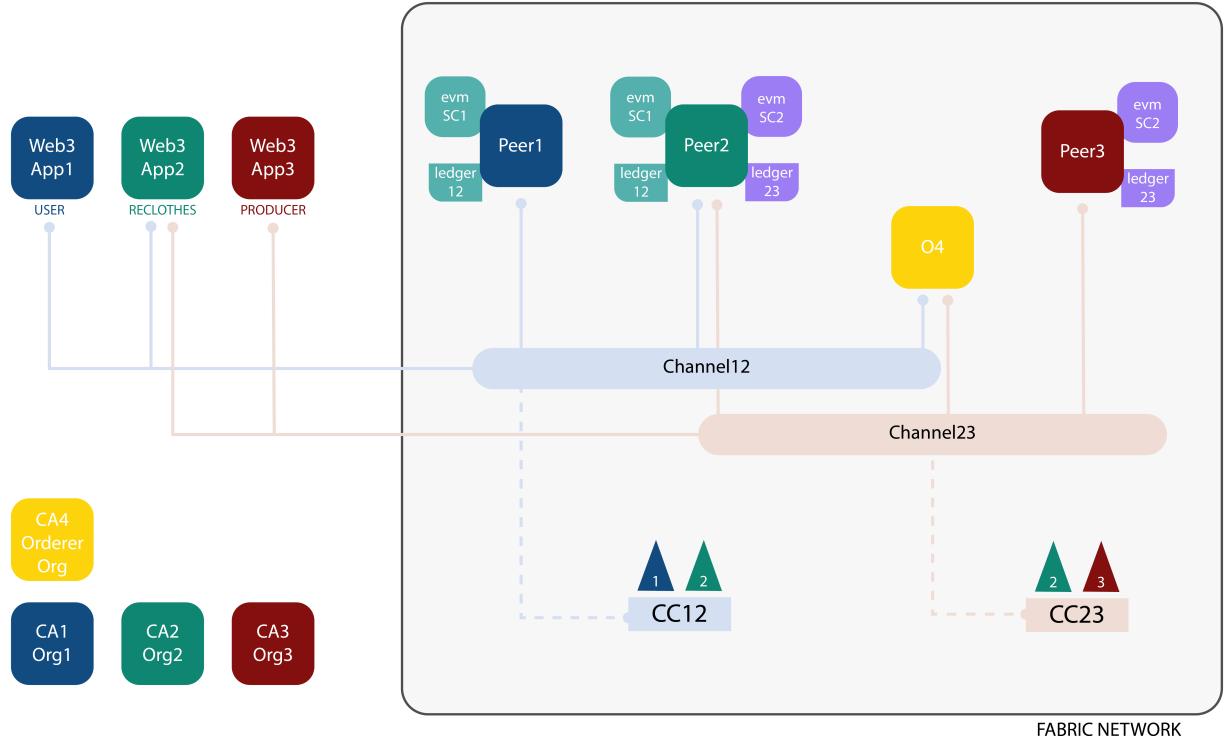


Figura 3.6. Fabric Network Components

By the Figure ?? it's possible to see that in own architecture ther's two ledger, each of that associated to one channel that involves just part of organizations per channel. The ledger is associated to 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 the channel communication. The channel is the only communication way between external and internal network. The external actor that invoke the chaincode must to have the priviledges for join to 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 forward the reply to the App client. Than the Dapp forward the answer to the *Orderer* peer that validate the response, create a new block, add it to the chain, communicate it to the peer in order to syncronize the network and updating the Ledger World State.

Config File

Network Architecture in Hyperledger Fabric is highly modular and scalable. Hyperledger provide to the developers a set of developed test network⁴ for testing purpose and that help developers to understand better the structure and the creations steps. The `fabric-samples` contains a set of tools that allow to release all the cryptographic materials required for the networks usage, such as certificates related to the user belonging to a specific organization. In other words all the MSP works is handled by fabric-samples tools.

The most usefull 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's wrote the `config.yaml` file.

The network is structured in the following lines of code:

```
1    Organizations:
2      - &OrdererOrg
3        Name: OrdererOrg
4        ID: OrdererMSP
5        MSPDir: crypto-config/ordererOrganizations/example.com/msp
6        Policies:
7          Readers:
8            Type: Signature
9            Rule: "OR('OrdererMSP.member')"
10         Writers:
11           Type: Signature
12           Rule: "OR('OrdererMSP.member')"
13         Admins:
14           Type: Signature
15           Rule: "OR('OrdererMSP.admin')"
16
17      - &Org1
18        Name: Org1MSP
19        ID: Org1MSP
20        MSPDir: crypto-config/peerOrganizations/org1.example.com/
21          msp
22        Policies:
23          Readers:
24            Type: Signature
25            Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP
26              .client')"
27          Writers:
28            Type: Signature
29            Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
```

⁴<https://github.com/hyperledger/fabric-samples>

```
28         Admins:
29             Type: Signature
30             Rule: "OR('Org1MSP.admin')"
31     AnchorPeers:
32         - Host: peer0.org1.example.com
33             Port: 7051
34
35     - &Org2
36         Name: Org2MSP
37         ID: Org2MSP
38         MSPDir: crypto-config/peerOrganizations/org2.example.com/
39             msp
40     Policies:
41         Readers:
42             Type: Signature
43             Rule: "OR('Org2MSP.admin', 'Org2MSP.peer', 'Org2MSP
44                 .client')"
45         Writers:
46             Type: Signature
47             Rule: "OR('Org2MSP.admin', 'Org2MSP.client')"
48     Admins:
49         Type: Signature
50         Rule: "OR('Org2MSP.admin')"
51     AnchorPeers:
52         - Host: peer0.org2.example.com
53             Port: 8051
54
55     - &Org3
56         Name: Org3MSP
57         ID: Org3MSP
58         MSPDir: crypto-config/peerOrganizations/org3.example.com/
59             msp
60     Policies:
61         Readers:
62             Type: Signature
63             Rule: "OR('Org3MSP.admin', 'Org3MSP.peer', 'Org3MSP
64                 .client')"
65         Writers:
66             Type: Signature
67             Rule: "OR('Org3MSP.admin', 'Org3MSP.client')"
68     Admins:
69         Type: Signature
70         Rule: "OR('Org3MSP.admin')"
71     AnchorPeers:
72         - Host: peer0.org3.example.com
73             Port: 9051
74
75
76     ...
77
78     ...
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1990
1991
1992
1993
1994
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
22
```

```
74     ...
75
76 Profiles:
77
78     OrdererGenesis:
79         <<: *ChannelDefaults
80         Orderer:
81             <<: *OrdererDefaults
82             Organizations:
83                 - *OrdererOrg
84             Capabilities:
85                 <<: *OrdererCapabilities
86             Consortiums:
87                 SampleConsortium:
88                     Organizations:
89                         - *Org1
90                         - *Org2
91                         - *Org3
92
93     Channel12:
94         Consortium: SampleConsortium
95         <<: *ChannelDefaults
96         Application:
97             <<: *ApplicationDefaults
98             Organizations:
99                 - *Org1
100                - *Org2
101                Capabilities:
102                    <<: *ApplicationCapabilities
103    Channel123:
104        Consortium: SampleConsortium
105        <<: *ChannelDefaults
106        Application:
107            <<: *ApplicationDefaults
108            Organizations:
109                - *Org2
110                - *Org3
111            Capabilities:
112                <<: *ApplicationCapabilities
```

Run of the network

For set up the network in the best way is created some scripts that perform the integration of the several components in the best way.

The macro process flows of the network running is:

1. **Check Prerequisite:** using the `fabric-samples`⁵ there's some prerequisite to check to run all the materials in the correct way. Check the prerequisite looking at fabric documentations⁶.
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's possible to run the network.
3. **Join all the components:** Once the network is in running 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 setted up in the right way and the network is in running, it's possible to install the chaincode over the peers that we want to use.

The Hyperledger Fabric network run inside **Docker containers**, to authomatize the running of the network I create scripts that setup fabric locally using docker containers, install the EVM chaincode (EVMCC) and instantiate the EVMCC on our fabric peers. This step uses 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 materiale for organizations:** First of all using the `fabric-sample` supplied by IBM, It's possible to run a tool in charge to create 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 to be generted for the channel involed to the network.
 - **Run docker containers:** Once the crypto materials is created than it must run the docker containers for the components of the networks. The **Figure ??** shows the related output.
 - **Inizialize Orgs and join to the channels:** Once all the containers is in running then there's the organizations initializations and each peer owner by an Org is joined to the channel following the `config file` specifications. **Figure ??** show the related output.

⁵<https://github.com/hyperledger/fabric-samples>

⁶<https://hyperledger-fabric.readthedocs.io/en/master/>

- **Instantiate and Install evmm chaincode:** Once all the network is run we are going to install the chaincode over the peers. In own case we are going to install `fabric-chaincode-evm`. **Figure ??** show the related output.

- `net_down.sh`:
 - remove all docker containers in running
 - remove all docker volumes created
- `fab1.sh`:
 - **network-sdk-config.yaml**: Its the mapping SDK file from web3js request to the fabric requests.

```
#####
##### Run docker containers #####
#####
Creating network "net_byfn" with the default driver
Creating volume "net_orderer.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer0.org3.example.com" with default driver
Creating orderer.example.com ... done
Creating peer0.org2.example.com ... done
Creating peer0.org1.example.com ... done
Creating peer0.org3.example.com ... done
Creating cli ... done

#####
##### Docker containers in running #####
#####
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
 NAMES
1c73cc086c1b      hyperledger/fabric-tools:latest   "/bin/bash"        3 seconds ago     Up Less than a second
cli
b9f7996293b3      hyperledger/fabric-peer:latest    "peer node start"  8 seconds ago     Up 2 seconds          0.0.0.0:7051->7051/tcp
peer0.org1.example.com
f02dd1e39d75      hyperledger/fabric-peer:latest    "peer node start"  8 seconds ago     Up 2 seconds          0.0.0.0:8051->8051/tcp
peer0.org2.example.com
8ba3808deaa3      hyperledger/fabric-peer:latest    "peer node start"  8 seconds ago     Up 3 seconds          0.0.0.0:9051->9051/tcp
peer0.org3.example.com
eba946a5a6e1      hyperledger/fabric-orderer:latest  "orderer"          8 seconds ago     Up 2 seconds          0.0.0.0:7050->7050/tcp
orderer.example.com
```

Figura 3.7. Run of the Docker Containers

Solution

```
#####
##### Init Org1 and join to the Channels #####
#####
2020-06-22 15:05:44.472 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
##### peer0.org1.example.com:7051 create and join to channel12 #####
2020-06-22 15:05:44.621 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:44.691 UTC [cli.common] readBlock -> INFO 002 Received block: 0
2020-06-22 15:05:44.831 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:44.951 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-06-22 15:05:44.991 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:45.124 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
2020-06-22 15:05:45.208 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
#####
##### peer0.org1.example.com:7051 CHANNEL LIST: [ channels peers has joined: channel12 ] #####
#####
##### Init Org2 and join to the Channels #####
#####
2020-06-22 15:05:46.518 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
##### peer0.org2.example.com:8051 join to channel12 #####
2020-06-22 15:05:46.657 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:46.750 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-06-22 15:05:46.881 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:46.922 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
2020-06-22 15:05:47.065 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
#####
##### peer0.org2.example.com:8051 create and join to channel23 #####
2020-06-22 15:05:47.290 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:47.290 UTC [cli.common] readBlock -> INFO 002 Received block: 0
2020-06-22 15:05:47.431 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:47.501 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-06-22 15:05:47.644 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:47.674 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
2020-06-22 15:05:47.822 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
#####
##### peer0.org2.example.com:8051 CHANNEL LIST: [ channels peers has joined: channel12 channel23 ] #####
#####
##### Init Org3 and join to the Channels #####
#####
2020-06-22 15:05:48.939 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
#####
##### peer0.org3.example.com:9051 join to channel23 #####
2020-06-22 15:05:49.203 UTC [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
2020-06-22 15:05:49.336 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-06-22 15:05:49.371 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
2020-06-22 15:05:49.517 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
#####
##### peer0.org3.example.com:9051 CHANNEL LIST: [ channels peers has joined: channel23 ] #####
#####
```

Figura 3.8. Orgs initializations and channels join

```
#####
##### Install Chaincode evn over the peers #####
#####
===== Install EVM chaincode over peer0.org1.example.com:7051 =====
2020-06-22 15:05:50.668 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-06-22 15:05:50.668 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2020-06-22 15:06:02.650 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
===== Install EVM chaincode over peer0.org2.example.com:8051 =====
2020-06-22 15:06:02.886 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-06-22 15:06:02.887 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2020-06-22 15:06:07.680 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
===== Instantiate EVM chaincode over channel12 =====
2020-06-22 15:06:07.809 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-06-22 15:06:07.809 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
===== Install EVM chaincode over peer0.org3.example.com:9051 =====
2020-06-22 15:06:07.11.490 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-06-22 15:06:07.11.490 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
2020-06-22 15:06:07.19.254 UTC [chaincodeCmd] install -> INFO 003 Installed remotely response:<status:200 payload:"OK" >
===== Instantiate EVM chaincode over channel23 =====
2020-06-22 15:06:07.19.427 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc
2020-06-22 15:06:07.19.428 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc
stefano@stefano-VirtualBox:~/go/src/github.com/hyperledger/fabric-samples/test-net-3\$
```

Figura 3.9. Chaincode Installation

Scripts produced

For a better understanding of the setting up and running of the entire network i reported below the two main script. 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`:

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

Solution

```
2      #Generate crypto material for organizations
3      echo "#####
4      echo "##### Generate Crypto Material for Organizations #####
5      echo "#####
6      echo "#####
7      ./bin/cryptogen generate --config=./crypto-config.yaml
8
9      # Build channel artifact
10     echo "#####
11     echo "##### Build Channel Artifact #####
12     echo "#####
13     ./channel_artifact.sh
14
15
16      # Run up docker containers
17      echo
18      echo "#####
19      echo "##### Run docker containers #####
20      echo "#####
21      docker-compose up -d
22
23      echo "#####
24      echo "##### Docker containers in running #####
25      echo "#####
26      docker ps
27
28      # Org1 Initialization and join channels
29      echo "#####
30      echo "##### Init Org1 and join to the Channels #####
31      echo "#####
32      docker exec cli scripts/org1_init.sh
33
34      # Org2 Initialization and join channels
35      echo "#####
36      echo "##### Init Org2 and join to the Channels #####
37      echo "#####
38      docker exec cli scripts/org2_init.sh
39
40      # Org3 Initialization and join channels
41      echo "#####
42      echo "##### Init Org3 and join to the Channels #####
43      echo "#####
44      docker exec cli scripts/org3_init.sh
45
46      # Install and Instantiate evmcc
47      echo
48      echo "#####
49      echo "##### Install Chanicode evm over the peers #####
50      echo "#####
51      docker exec cli scripts/install.sh
```

net_down.sh:

```
1  #!/bin/bash
2
3  # STOP AND DELETE THE DOCKER CONTAINERS
4  docker-compose down -v
5  docker rm $(docker ps -aq)
6  docker rmi $(docker images dev-* -q)
7
8  # DELETE THE OLD DOCKER VOLUMES
9  docker volume prune
10
11 # DELETE OLD DOCKER NETWORKS (OPTIONAL: seems to restart fine
12 # without)
12 docker-compose -f down --volumes --remove-orphans
13 docker network prune
14
15 # DELETE SCRIPT-CREATED FILES
16 rm -rf channel-artifacts/*.block channel-artifacts/*.tx crypto-
17 config
18
19 # Remove created folder
20 rm -rf channel-artifacts
21
22 # VERIFY RESULTS
22 docker ps -a
23 docker volume ls
24 ls -l
```

3.5.4 Ethereum Network - Ropsten

To run the ERC20 token it's used the testnet Ropsten against the mainnet. To set up and upload own ERC20 Token over the ethereum network it's used:

- **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 allow to run Solidity bytecode inside Hyperledger Fabric network, first of all we're going to analyze the interaction process and chaincode invocation Process of Hyperledger Fabric Blockchain.

End to End Interactions

Going deeper, the **Figure ??** show the flow of the end to end communication. How all the components are boxed inside the Peer component. The Fab3 map the web3 request and forward it to fabric peer. the request arrive to the evmcc that invoke Solidity smart contract methods.

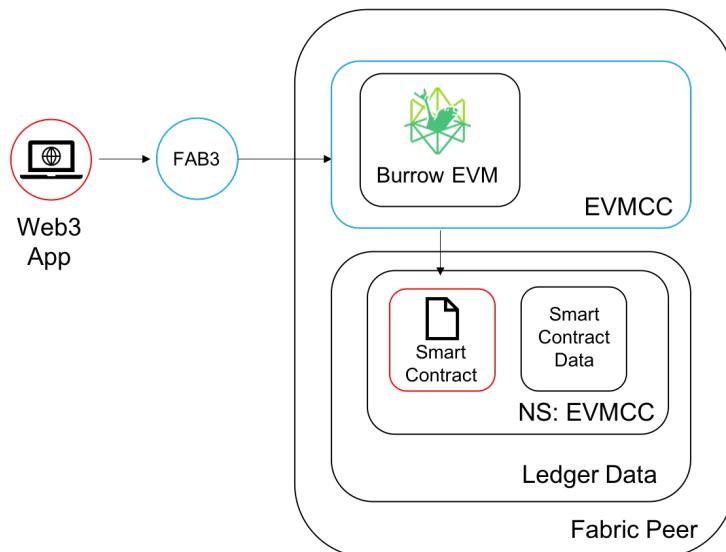


Figura 3.10. End To End

Chaincode Invocations

The **Figure ??** below describe the internal workflow of the chaincode invocation, where's involved the *Client* the *Peer* and the *Orderer*. All the information are transfer over the setted channel and in our study case, the client doesn't interact directly, but using *Fab3 Proxy* as intermediary.

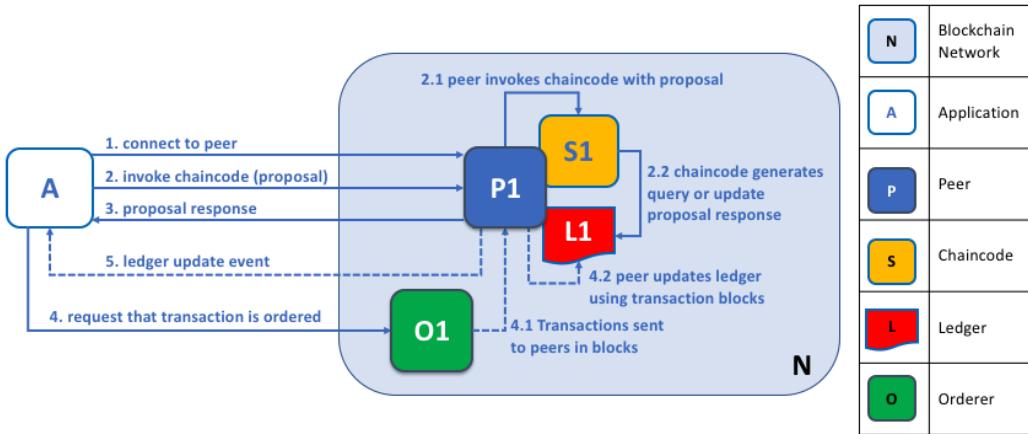


Figura 3.11. Smart Contract Invocation Process

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 going to map 1 fabric user and going to generate a semi random eth address starting from the public key of own x.509 certificate related to Fabric identity. The Following environment variable going to set the mandatory data to run a Fab3 proxy instance:

- **CONFIG:** It's the path to a compatible Fabric SDK Go config file, used to communicate and map the requests and forard it to fabric network.
- **USER:** User identity being used for the proxy. Matches the users names in the crypto-config directory specified in the config.
- **ORG:** Organization of the specified user.
- **CHANNEL:** Channel to be used fo 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

```

1 # Environment variables required for setting up Fab3
2 export FAB3_CONFIG=${GOPATH}/src/github.com/hyperledger/fabric-
   chaincode-evm/examples/network-sdk-config.yaml
3 export FAB3_USER=User1
4 export FAB3_ORG=Org1
5 export FAB3_CHANNEL=channel12

```

Solution

```
6 export FAB3_CCID=evmcc  
7 export FAB3_PORT=5000
```

Once the fab3 is set up and the instance is in running we could perform chaincode invocation using our Dapp. The **Figure ??** shows the flow of the invocation process at upper level, It shows the main components that are involved in that process,

1. **Dapp** call method that perform smart contract invocation.
2. **Fab3** map the request and forward it to Fabric network.
3. **Fabric network** process the request and issue a response.

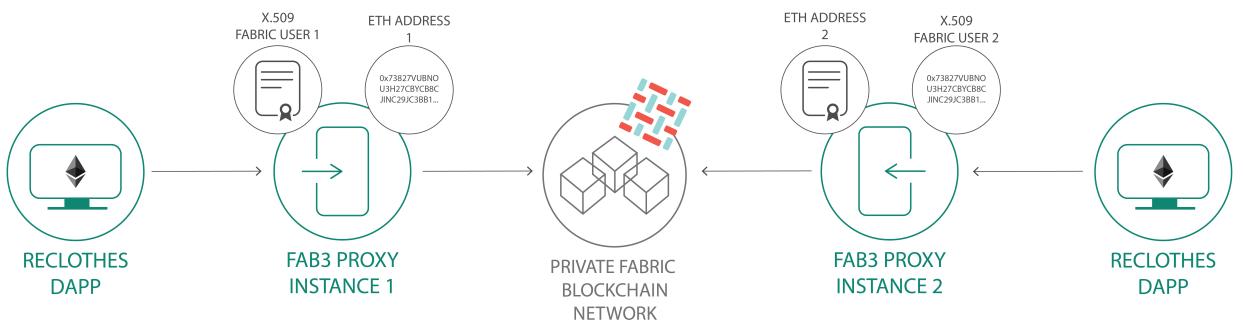


Figura 3.12. Fab3 Proxy Flow

The **Figure ??** show the internal components that take part to the invocation process. Fab3 agreed the request using *Ethereum JSON RPC API* map it and forward to the fabric network using *GO SDK*. Once the request arrive to *EVMCC* it invoke smart contract bytecode and than follow the standard process explained in Figure ??.

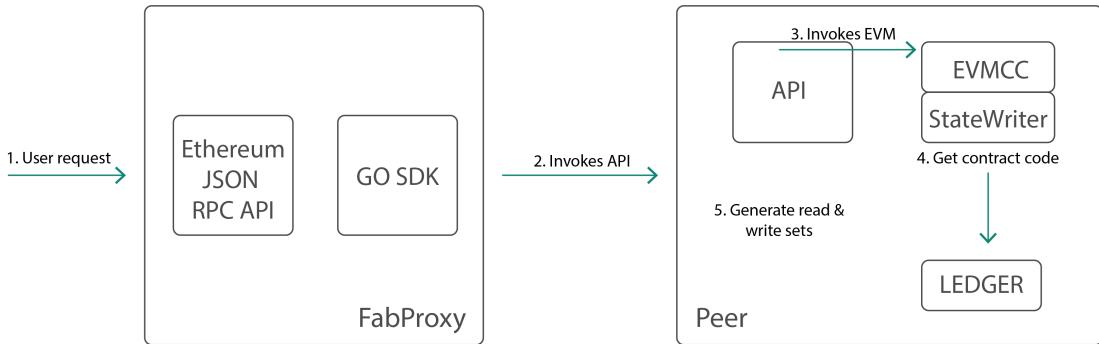


Figura 3.13. Fab3 Invocation Process

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 allow the communication with smart contracts of both Blockchains. The technologies used are the following one:

- **Expressjs:** It's a node.js framework that allow to develop api for own application
- **Bootstrap:** To build a user friendly front-end in order to interact in the best way
- **Web3:** Ethereum Javascript API, It's 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's a stable version and it's the version used in `fabric-chaincode-evm` development
 - **web3 1.0.0:** used for ethereum transactions, It's a version with more functionalities but less stable.

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

3.7.2 Core part of the web-app

The technical files and flow that dapp follow to run up it's the following one.

1. Contract Address Generation:

- (a) This step is in charge to run a script that deploy the contract addresses to be referred during the app running.
 - i. **UserContract.js**: running the script using `node .js` file, it return the address of the deployed contract. The **Figure ??** show the related output and the Contract Address printed.
 - ii. **ProducerContract.js**: running the script using `node .js` file, it return the address of the deployed contract. The deployed process and output is similar to Figure ??.
2. **dapp.js**: there's the core file that handle the contracts invokations, 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, map the request and forward to `dapp.js`.

```
{ transactionHash:  
  '0x2a1c7f935b455d12b7e3a62f2d449e5b946d60b4046a137b69e00eb5a10cd1f2',  
  transactionIndex: 0,  
  blockHash:  
  '0x0c8b3e0be63965575ed3cdcbface91b373e3e876763499acc135de118e3aeda5',  
  blockNumber: 4,  
  contractAddress: '0x2558669e229f1dd20eb45a709d48884a87e51378',  
  gasUsed: 0,  
  cumulativeGasUsed: 0,  
  to: '',  
  logs: [],  
  status: '0x1',  
  from: '0xe3769d1f3f583d77626f12aed90bfa252a61d52a' }
```

Figura 3.14. Deploy User Contract

Once is all set up we could run the web-app with the command `npm start`, there's an initialization phase and that the app is ready to use and in running of over the specified PORT. The **Figure ??** shows the output.

Solution

```
stefano@stefano-VirtualBox:~/go/src/github.com/hyperledger/web-app-final$ npm start
> box-points@0.0.1 start /home/stefano/go/src/github.com/hyperledger/web-app-final
> node app.js

=====
Uploading User Contract
Getting the Contract
Got address: 0x407c17ca284989069394edc929ac97535db06961
Uploaded User Contract 0x407c17ca284989069394edc929ac97535db06961

=====
Uploading Producer Contract
Getting the Producer Contract
Got Producer Address: 0xa224d66bed5be81281ed3b7485e05608585134ca
Uploaded Producer Contract 0xa224d66bed5be81281ed3b7485e05608585134ca

=====
Uploading eth address
Getting the Account Address
Account Address: 0xe3769d1f3f583d77626f12aed90bfa252a61d52a
Uploaded eth address 0xe3769d1f3f583d77626f12aed90bfa252a61d52a

app running on port: 8000
```

Figura 3.15. App Running

3.7.3 Views

Homepage

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



Figura 3.16. Home

Solution

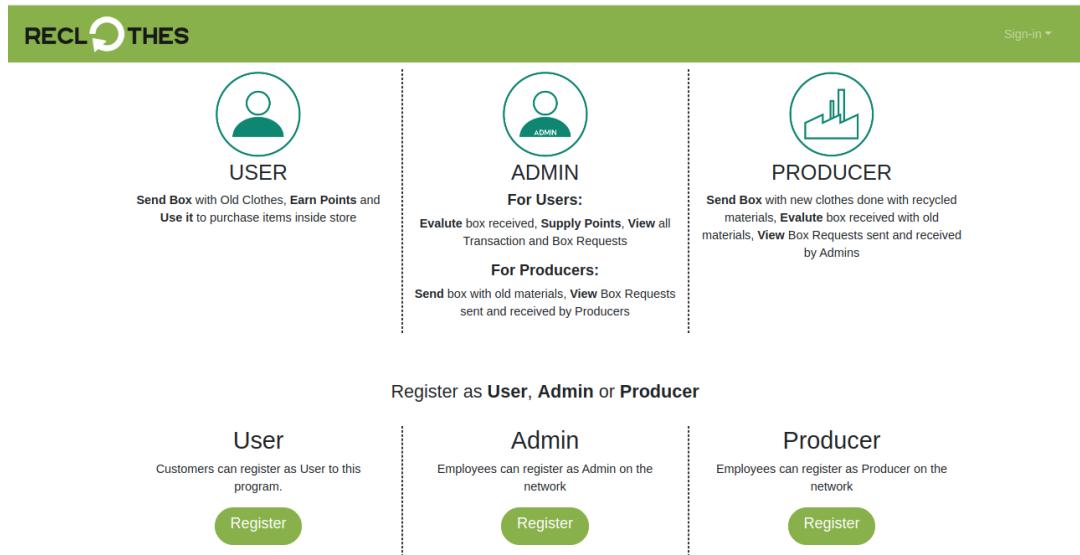


Figura 3.17. Registration Phase

User Page

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

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

The **Figure ??** show how to compile the form in order to send box with old clothes. It's a simulation of the real process to sending box, that in the real case could be implemented thow a QRCode or RFID placed over the boxes.

The **Figure ??** show how should be the store, purchase items over the platform start the transaction process.

There's other section about infos that user is allowed to see. **Transactions** performed over the fabric network and **Box Requests** that's all the history about the box sent and received with all the related informations.

Solution

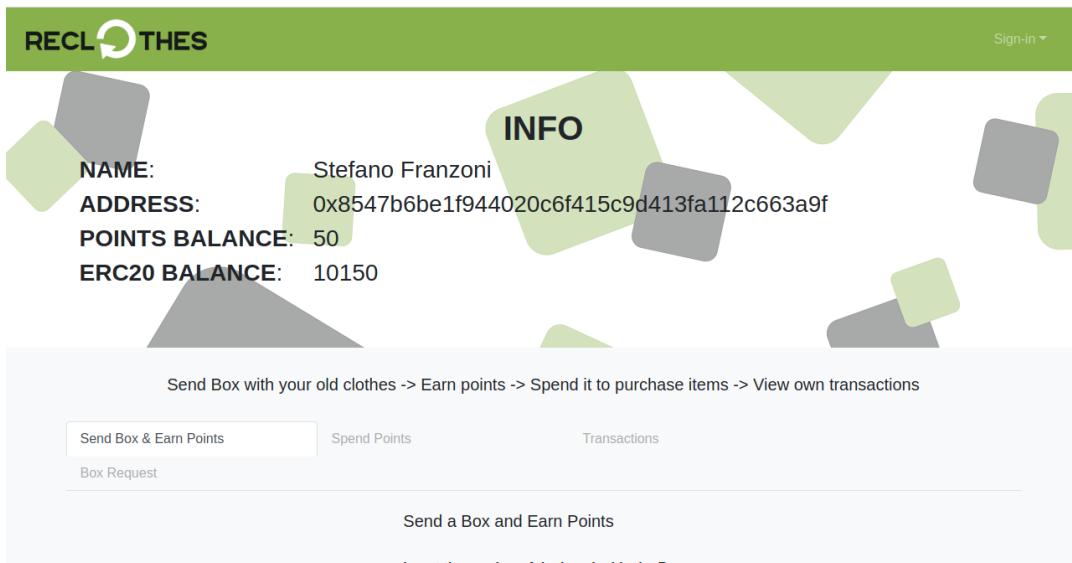


Figura 3.18. User Info

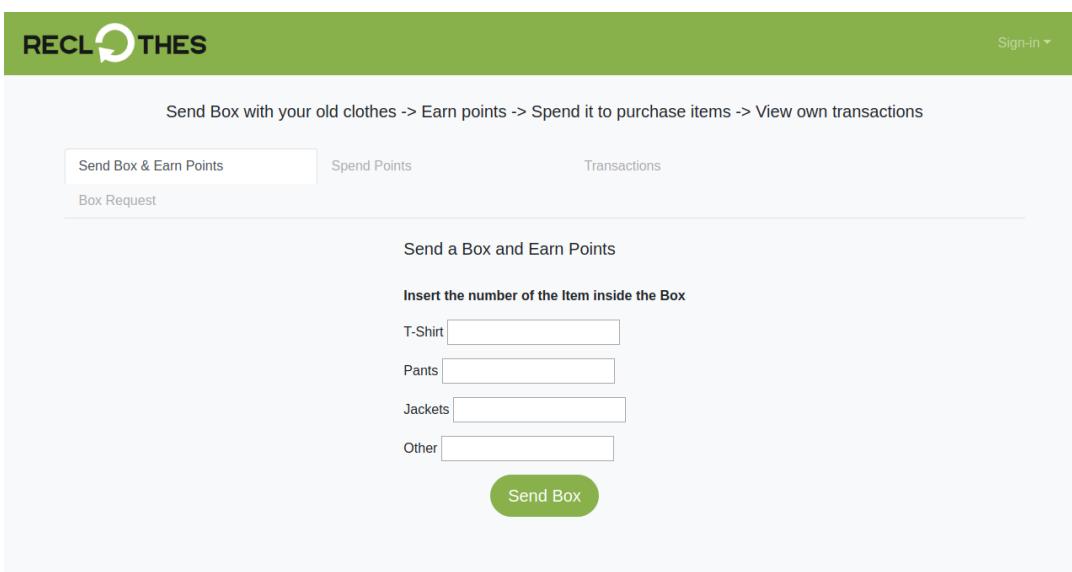


Figura 3.19. Send Box

Reclothes Admin Page

In the previous sections we talk about a logical split about Admin for User and Admin for Producers. In the following views we divide the feature related to the User type to be handled and there's a straight distinctions about Users and

Solution

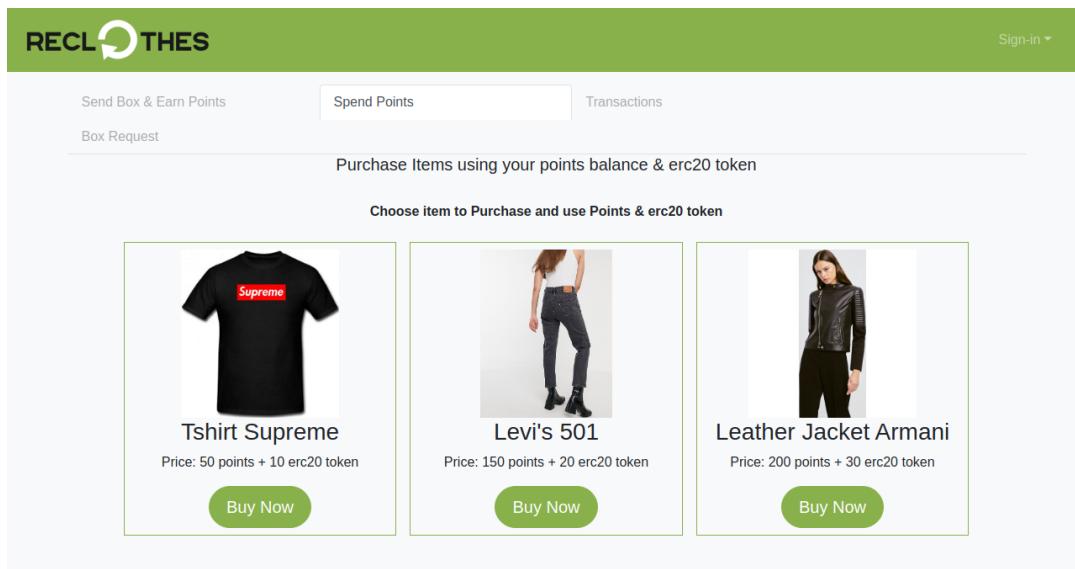


Figura 3.20. Purchase Clothes

Producers.

Admin For Users This section show the view of the Admins that handle User side.

The screenshot shows the RECLOTHES admin interface. At the top, there is a black header bar with the logo 'RECLOTHES ADMIN' and a 'Sign-in ▾' button. Below the header, there are three tabs: 'User', 'Producer' (which is selected), and 'Sign-in ▾'. On the left, there is a summary of a user's information:

NAME:	Admin
ADDRESS:	0xbe7a6c8956ed40bd225433b49796f9dfb11daf6b
TOT SUPPLIED POINTS:	100
TOT REDEEMED POINTS:	50

Below this, there are three more tabs: 'Evaluate Box', 'Pending Requests' (which is selected), and 'Evaluated Requests'. Under 'Pending Requests', there is a section titled 'Pending Box' with the following details:

User Address: 0x8547b6be1f944020c6f415c9d413fa112c663a9f
T-Shirt: 2
Pants: 2
Jackets: 5
Other: 1
Evaluated: false

Figura 3.21. Admin Info

Solution

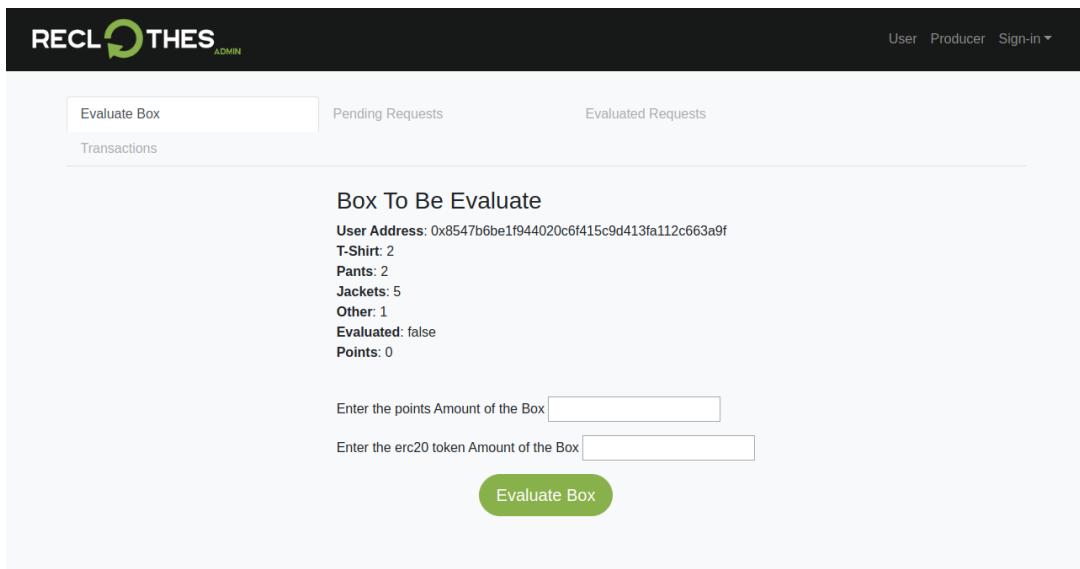


Figura 3.22. Evaluate Box

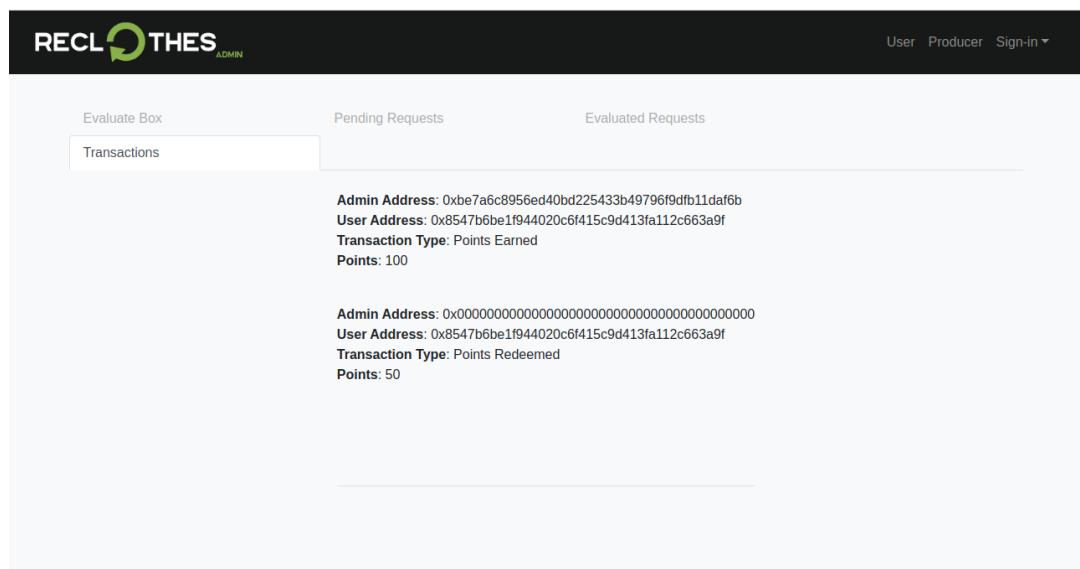


Figura 3.23. Transactions

Admin For Producers This section show the view of the Admins that handle Producer side. The Figure ?? show all the Admin for Producers informations.

Solution

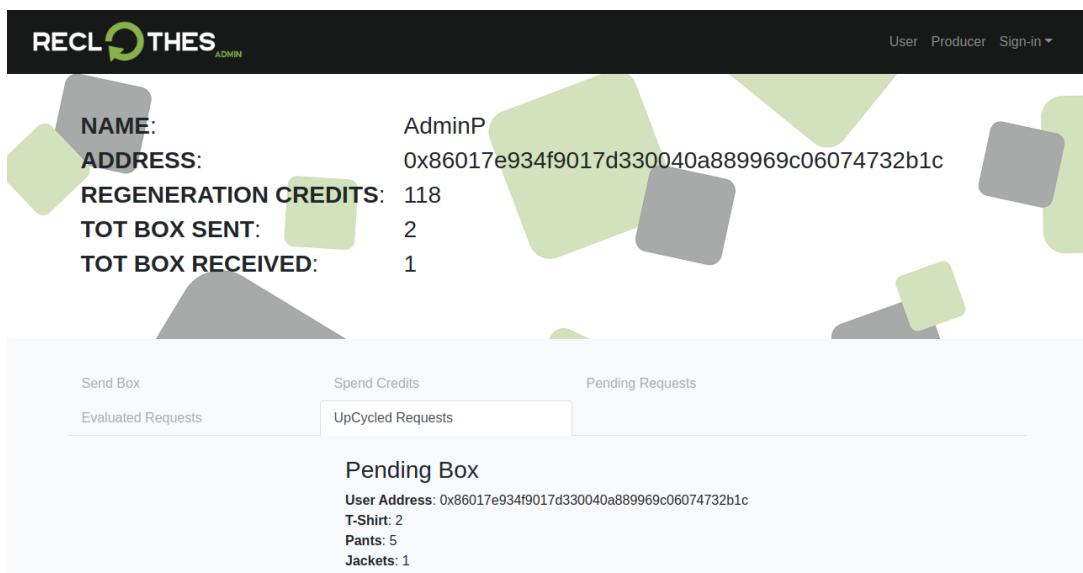


Figura 3.24. Admin for Producers Info

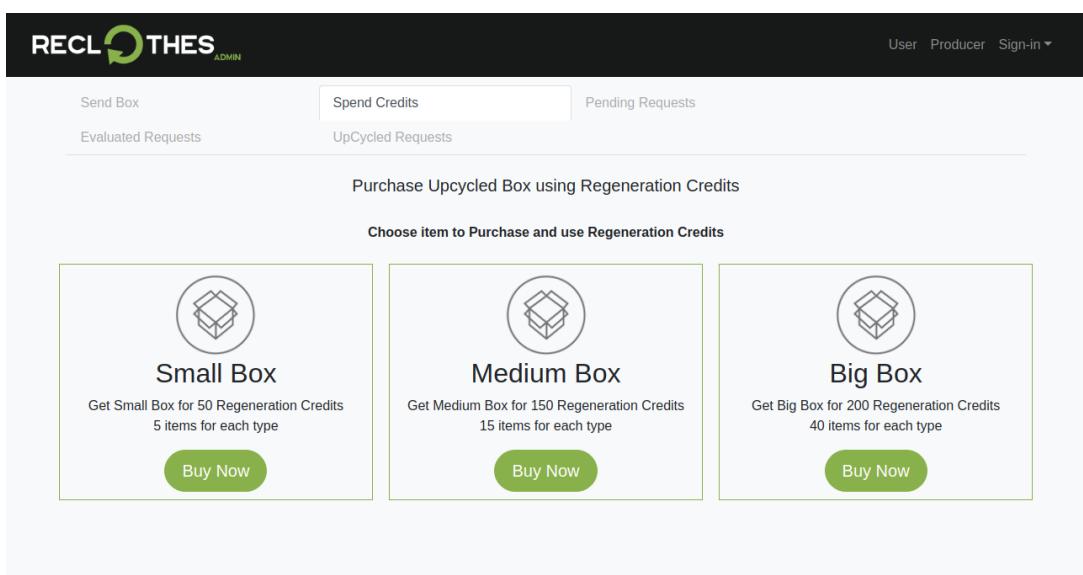


Figura 3.25. Admin Spend Regeneration Credits

Producer

This section show the view of the Producer side. The evaluation process of the old materials received buy Reclothes it's the same of the previous one.

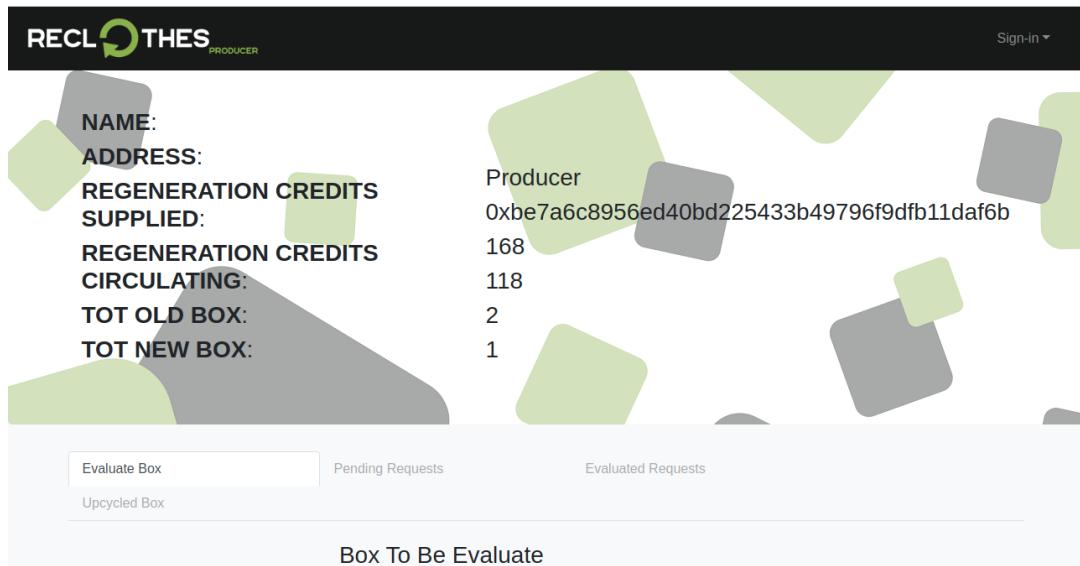


Figura 3.26. Producers Info

3.8 Cost Analysis

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 mantain 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 Blockchain⁷ 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

⁷<https://aws.amazon.com/it/managed-blockchain/>

- **Peer node cost:** $(3 \text{ members}) \times (1 \text{ bc.t3.small peer node per member}) \times (0.034 \text{ per hour}) \times (1 \text{ hour}) = \0.102 per hour
 - **Peer node storage cost:** $(3 \text{ members}) \times (1 \text{ peer node per member}) \times (20 \text{ GiB storage per peer node}) \times (\$0.10 \text{ per GB-month}) \times (1 \text{ hour}) = \0.009 per hour
 - **Data written cost:** $(3 \text{ members}) \times (9 \text{ MB per hour}) \times (0.10 \text{ 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 going to define the transaction time, it means that more gas price corrisponde to a less time of transaction computed and viceversa. There's several features that going to influence the transaction fees over ethereum network, such as ether price, or network traffic.

Capitolo 4

Results

4.1 Target archived

The goal to archive includes logical and technical targets both. The improvements reached by thesis development are the following one

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

- **Technical Goal**

- **Crosschain Interaction:** Integrate in the same application both permissioned and permissionless Blockchain networks. The integration was done application side, there's some API endpoints that 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 track 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 is handled as transactions, stored over the ledger and updating world state and smart contract data.

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

- **Susteinability:** The entire process is aims to support susteinability. Using treacebility feature it's possible to follow the lifetime of the clothes until they finish to Producer, that perform the material recycling in order to produce new upcycled clothes.
- **Counterfeiting:** Assign a UID to each clothes produced it's possible to fight the Counterfeiting implementing new feature such us the clothes registrations, we're going to have a secure register contining 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 application works.

The **Figure ??** show the log when User Perform 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 +++
||
```

Figura 4.1. User Send Box

Once the Box Request was successfully send, the smart contract is invoked and the transaction is performed. Admin could see the pending box request to be evaluated. Then the Reclothes Admin UI side insert the value amount of the tokens and start the evaluation process.

The **Figures ??** and **??** show the Fabric Transaction performed then the inizialization of the Ethereum Transaction and at the end once the eth transaction was performed the etherscan link, associated to the related TransactionHash, that allow to see the transaction history and infos.

Results

```
Valid Entries
[ '0xe3769d1f3f583d77626f12aed90bfa252a61d52a', 'Admin', true ]
[ '0xe3769d1f3f583d77626f12aed90bfa252a61d52a', 'Admin', true ]
Evaluate Box - points: 150 - token: 100
== Entering In ERC20 Transaction ==
== VALUES => From: reclothes - To: user - Amount: 100 ==
== Contract Address: 0xb25901e0c05a6b3ddc86e7b5161bb9ca1113e29 ==
== From => RECLTHES ==
== From Account Taken 0x1433D083c48609862a536b78D3777adFc20601ad - PrivateKey: o] []
  *$*,*a**>*  ***E***bQ08 ==
== To => USER ==
== To Account Taken 0xf07e54dBDF0FC9fdB52A8C90cF988c4e7D46830e ==
== Getting gas estimate for Amount: 10000000000 ==
```

Figura 4.2. Init Transaction from Reclothes to User

```
Estimated gas: 0x1
Nonce: 5
== Tx Parameters created ==
== Tx Signed ==
== Tx Serialized ==
== Balance: 108767700097342 ==
TransactionHash 0x669282c69a0f670bea927fa7ef5ba1853c831a9300435a7d6dd2a5eee9e03411
View transaction state: https://ropsten.etherscan.io/tx/0x669282c69a0f670bea927fa7ef5ba1853c831a9300435a7d6dd2a5eee9e03411
== Level 2: 0x669282c69a0f670bea927fa7ef5ba1853c831a9300435a7d6dd2a5eee9e03411 ==
== Level 3: 0x669282c69a0f670bea927fa7ef5ba1853c831a9300435a7d6dd2a5eee9e03411 ==
== Level 4: 0x669282c69a0f670bea927fa7ef5ba1853c831a9300435a7d6dd2a5eee9e03411 ==
```

Figura 4.3. Init Transaction from Reclothes to User

The Figure ?? and Figure ?? shows the proof of the transacactions succed. The first Figure show the User page that could visualize the history of transactions done and all related requests. The second one show the etherscan page with all the informations about ethereum transaction, in this case from Reclothes eth Account to User Account.

4.2.2 Use Case 1 - Unit Test 2

Purchase Item

The Figures ?? shows the Purchase process. As figure shows there's first of all the fabric transaction and then the eth transactions, after performed all the previous check start the transaction from User account to Reclothes account. Once the transaction is performed the method print the etherscan link to monitor the transaction and all related infos.

4.2.3 Use Case 2 - Unit Test 1

To test the Use Case 2 we're going to track the behaviour of two process:

Results

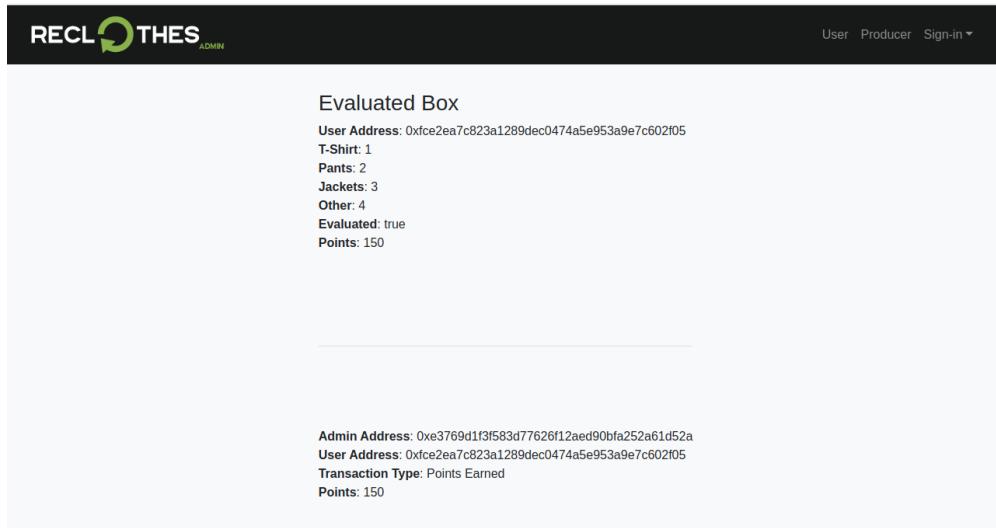


Figura 4.4. Fabric transaction history

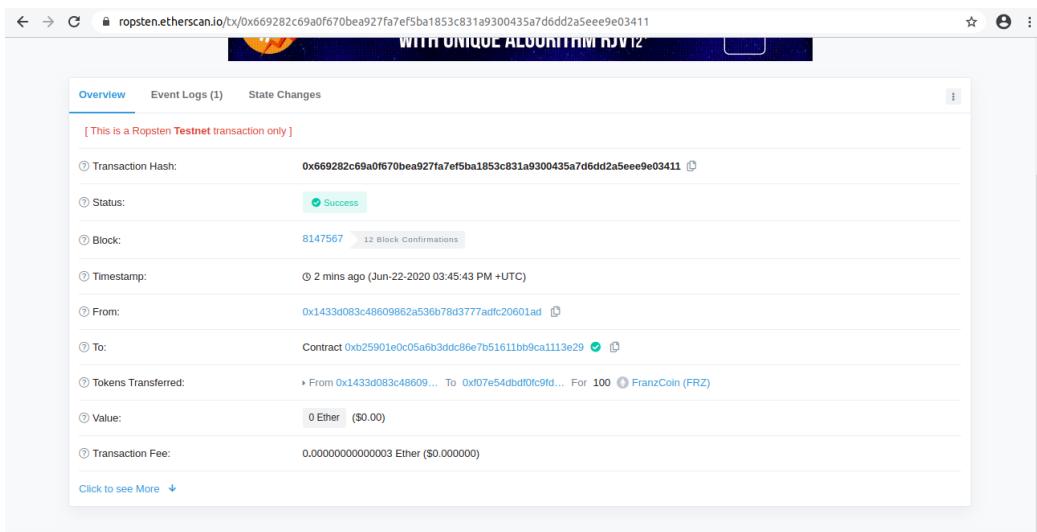


Figura 4.5. Ethereum transaction over etherscan

- **Send Old Material and Evaluation:** The Admin for Producer send a box within old materials to be recycled. Once the box arrived to Producer, than it's going to be evaluated and there's a Regeneration Credits transaction from Producer to AdminP.
- **Purchase Upcycled Material:** The Admin for Producer spend the earned Regeneration Credits to purchase by Producer recycled materials. The purchase options right now are three:

Results

Figura 4.6. Transaction from User to Reclothes

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

Send Old Material and Evaluation

The **Figures ??** show the log of the send old clothes process. In that case we're going to send a box with inside:

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

Once the box is sent than start the evaluation process. The Producer evaluate 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 we set a Regeneration Credits amount of 1200, **Figure ??** shows the output of the evaluation process.

Once the evaluation process is archived and the Regeneration Credits section from Producer to Admin, **Figure ??** show the info update of the Admin for Producer.

Results

```
totPointsProvided: 0
tot Regeneration Credits: 0
totBoxOld: 0
totBoxNew: 0
Valid Entries
Sending Box Old - tshirt: 10 pants: 20 jackets: 10 other: 10
Getting Admin for Producers Data
[ '0xe3769d1f3f583d77626f12aed90bfa252a61d52a',
  'AdminProducer',
  true,
  BigNumber { s: 1, e: 0, c: [ 0 ] },
  BigNumber { s: 1, e: 0, c: [ 0 ] } ]
tot Regeneration Credits: 0
totBoxOld: 1
totBoxNew: 0
VBox_GAs_6.0.6
```

Figura 4.7. Admin Send old clothes

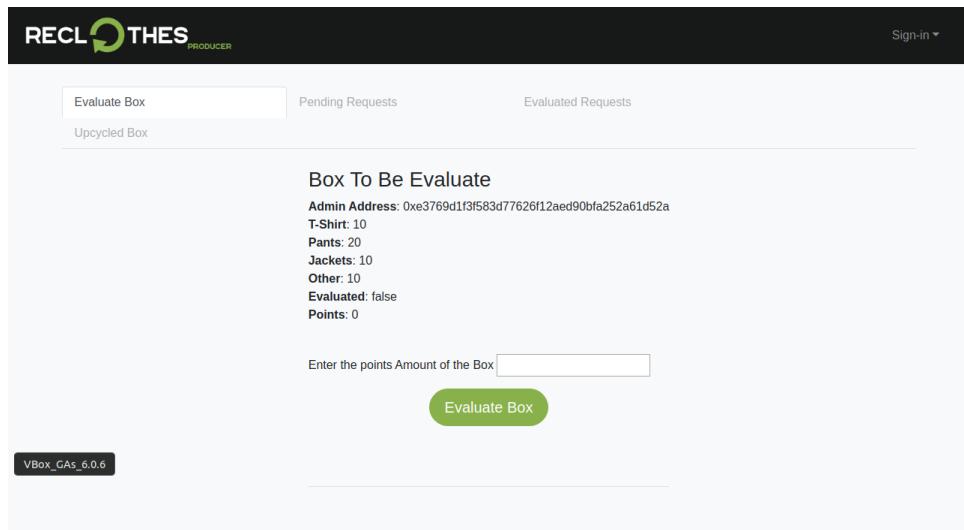


Figura 4.8. Box to be evaluated

Purchase Upcycled Material

Once the Admin sent box with old clothes and the evaluation process is archived, Admin has a Regeneration Credits amount to spend to purchase upcycled clothes to send inside platform store. In our test case we purchase a **Middle Box** going to spend 150 Regeneration Credits.

The **Figure ??** show the output of purchase process and the Tot Regeneration Credits update after purchase box process is performed.

The **Figure ??** show the the update of Producer Informations, the circulating

Results

```
tot Regeneration Credits: 0
Got Producer Data
Getting Producer Data
tot Regeneration Credits: 0
Got Producer Data
Evaluating Old Box - points: 1200
Getting Producer Data
```

Figura 4.9. Producer Evaluate old materials

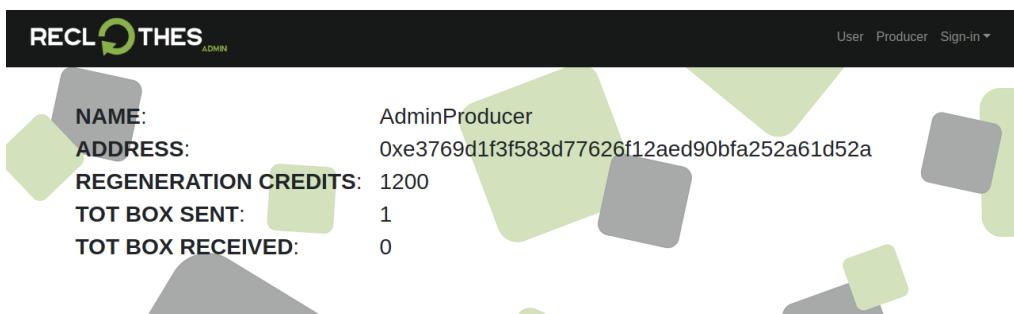


Figura 4.10. Admin Info update

Regeneration Credits amount is changed and the Tot Box New number too.

```
BigNumber { s: 1, e: 0, c: [ 1 ] }
totPointsProvided: 1200
tot Regeneration Credits: 1200
totBoxOld: 1
totBoxNew: 0
Valid Entries
Buy UpCycled Box - tshirt: 15 pants: 15 jackets: 15 other: 15 points150
Getting Admin for Producers Data
[ '0xe3769d1f3f583d77626f12aed90bfa252a61d52a',
  'AdminProducer',
  true,
  BigNumber { s: 1, e: 0, c: [ 1 ] },
  BigNumber { s: 1, e: 2, c: [ 150 ] } ]
totPointsProvided: 1200
tot Regeneration Credits: 1050
totBoxOld: 1
totBoxNew: 1
```

Figura 4.11. Purchase Recycled Clothes

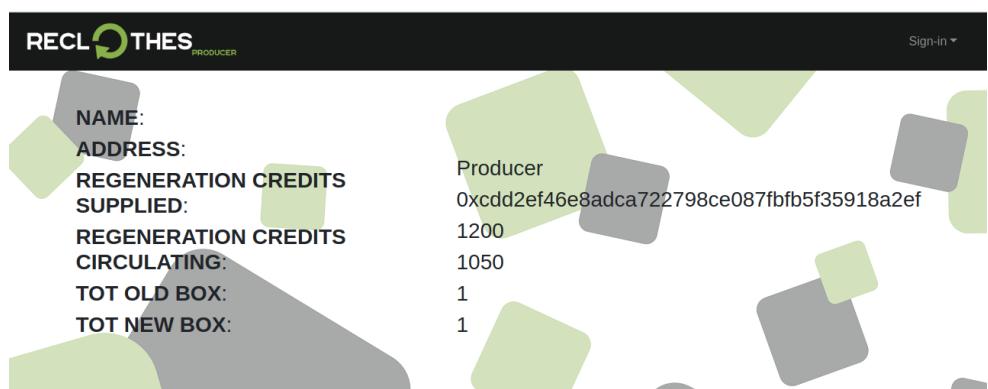


Figura 4.12. Producer Infos Update

Capitolo 5

Conclusion

For a clearer understand and overview about all the work produced, that are listed below:

- **Hyperledger Network:** It's designed a network architecture based on own needs. It's implemented and produced with a set of scripts that authomatize all the setting up and running process.
- **Smart Contracts:** Are Produced two smart contract that shapes the use case deal.
- **ERC20 Token:** It's produced an ERC20 token in running and accessible over the Ropsten network.
- **Dapp:** There's developed a web-application that integrates perfectly both the blockchains.

Than we are going to analyze all the target archived:

- **Simple managements process:** the process are structured in a simplified way, clear and easy to the usage and understanding.
- **Supply Chain:** thanks to the blockchain applications to the use case, the Supply Chain is clear and transaction based. The managements admin side is improved with great results.
- **technology improvements and modular solutions:** the crosschain solution is implemented at Application Layer, it could be performed in other use case taking the principles used during dapp implementation. In other development it's possible to think to a solution at lower layer, such as chaincode

Conclusion

side, that solution could be more adaptable and modular than the developed one. In any case to apply that kind of solution it's mandatory the development of other modules such as a storing and mapping mechanism between eth wallet and Fabric Identity.