POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica (Software Engineering)

Tesi di Laurea Magistrale

Crosschain with Hyperledger Fabric and Ethereum

Dapp build using Hyperledger Fabric and Ethereum networks

Relatore
prof.ssa Valentina Gatteschi

Candidato
Stefano Franzoni

Tutor aziendale
dott. Alfredo Favenza

Anno accademico 2019 – 2020

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduction

Capitolo 2

State of the art

2.1 Current state of networks solution

There's three main solution about blockchain network[1]:

- **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 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.

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

The choose of the blockchain to use depends on the Use Case.

The main advantages of the **public blockchain** coul be fall into two major categories:

1. 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.

2. 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 work public blockchain fully eliminate intermediary.

Compared to the public bloclchain the advantages of a **private blockchain** are:

1. 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.

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

3. 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.

4. 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.

5. 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 aim[2], 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 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.

[2] **The 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

Blockchain Name	Network	Currency	Consensus	Smart Contract
Bitcoin	Public	Bitcoin	PoW	Possible but less extendible
Ethereum	Public	Ether	PoS	Multiple programming languages (Solidity, Vyper)
Hyperledger	Permissioned - Federal/Private	None	Pluggable or PBFT	Multiple programming languages (Go, Java, Javascript, Solidity)

2.1.1 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 malicous node that could compromise data and transactions performed over the network. The entire architecture and consesus are distributed in order to minimize the liability of data manipulation.
- **Consortium**: The basis idea is that the network is composed by a set of truster or semi-trusted node, that compose the governance of the network. The consensus mechanism is not so stricted beacause the starting ipothesis is different and usually it need to have a good performance and low latency of the transactions. On the other hand this features could lead to a data manipulation.

2.1.2 Limitation

Now that is clead 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 an asset transactions amoung users without limit and granted security and authentication, on the other hand consortium could

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

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 network must be synchronize and world state must be the same in each moment. It means that each transactions that involves both blockchains before to be archived the both networks must reach a strong synch amoung the ledgers.
- **Time Effort**: to make it usable it mean that the transactions and synchronization must be performed in a reasonable time.
- **Identity**: over the blockchains network the identity if the user or in other word the owner of the wallet is handled in several ways. For example Ethereum handle 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 an other problem is the mapping of that different mechanism that blockchains implements to allow authentications.

2.1.3 Current Solution

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

- **New Blockchain**: During the year was born several networks and frameworks that propose to allow the interconnection between public and private blockchains. Many of that solutions are based on new blokchchain networks that are stuctured in order to allow architectural level the interoperability, one of that reality is Ark[4]. 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 vicevers pass by this Sidechain.

2.1.4 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**[5] 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 amoung blockchains. Of course it still have many limit for the use, in particolar for the identity mapping from eth address to fabric identity and vice-versa.

[4] https://ark.io/

[5] https://github.com/hyperledger/fabric-chaincode-evm

2.2 Blockchain application in Fashion Environment

2.2.1 Provenance case Martine Jarlgaard

Thanks to the blckchain 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 Provenance[6] producing clothes with digital tag: The tag could be a QRCode od 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 an 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.

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 VeChain BabyGhost

2.3 Use Case Example to apply Blockchain

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.

[6] Provenance withepaper (2015) Blockchain: the solution for transparency in product supply chains. https://www.provenance.org/whitepaper

2.3.1 Sustainability Token

PlasticToken

It's an ERC20 chaincode that run over the hyperledger fabric network[7]. 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.

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.

[7] https://ptwist.eu/

Capitolo 3

Solution

3.1 Oveview

3.1.1 Actors

The overview and flow of the Dapp developed is shown in **Figure 3.1**. The **Actors** involved in the system are:

- **User**: It's the *end user*. It use the web-app to send old clothes and purchase from Reclothes store
- **Reclothes Admin**: It's the *system admin*, it perform the actions in order to handle the system
- **Producer**: It's part of the upcycling process. It receive the materials to perform the recycling process

Each of that access to the system wich different permission and priviledges. Once the user is logged in, It could access to several features. It's possible to split the overview flow shown in **Figure 3.1** into 2 subflow starting from Reclothes actor, the **User side** and the **Producer side**.

1. **User Side**
   (a) User send Box with old clothes and receive 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

3.1.2 Token echanged

The **Token** exchanged over the networks will be:

1. **Over Fabric Net**
   (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.
   (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 Net**
   (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.

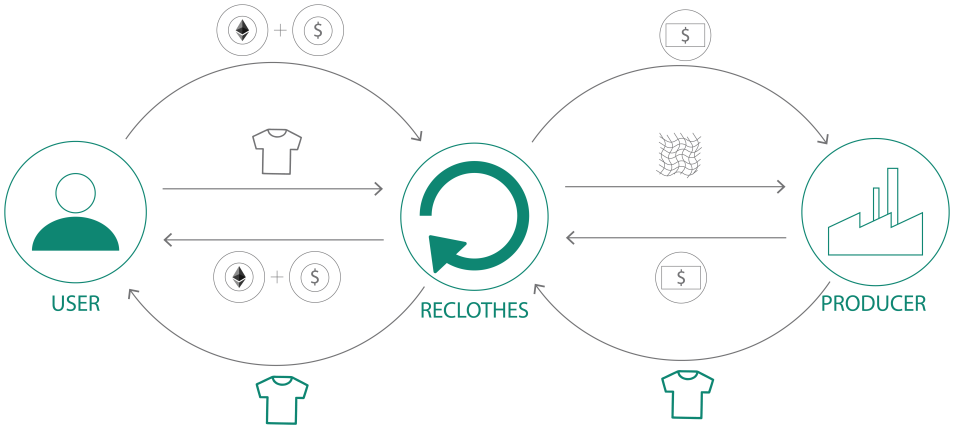

Figura 3.1. UseCase Overview

3.2 Crosschain interaction

Thanks to the introduction of Solidity bytecode over the Fabric network it's possible to adopt ethereum technologies(languages, API and all releted innovation) in Hyperledger Fabric world. The cross-chain solution that we adopt in the following Use Cases is Application Level. The main core idea of the solution is mapping application level the Eth Wallet with Hyperledger Fabric Identity and use one for transactions over eth network and the other one over fabric network. Exploiting Web3.js API we invoke eth or fabric smart contracts, with that solution all the invocation forward one other network start Dapp Side. The security is granted Fabric Side using certificate x.509, the mechanism of authentication doen't change. On the other hand ethereum side is handle in the following way, the own eth address is specified during registration phase and saved over fabric network, the security is granted by that at the moment of transactions the sign is performed Metamask side, in that way only the real owner of the eth account could sign and approve the transaction.

3.2.1 Technologies Used

- **Metamask**: It's used as ethereum wallet to perform and sign the transactions started by dapp
- **Web3**: It's the software library used to interact with smart contract
- **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.
- **Fabric Chaincode EVM**: It's the EVM chaincode that allow to run Solidity smart contract over the Fabric network
- **Expressjs**: Web Framework used to develop web-app and smart contract API
- **Infura**: allow to run a Ethereum node in order to set an endpoint used to interact with own contract.
- **Docker**: The fabric network components run inside Docker containers.

Figure 3.2 show The Architectural Flow and how the technologies is used and interact each other. Metamask is used as eth wallet to sign transaction over ethereum network, the actor is logged over the dapp client. Therefore 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. Fab3 allow to use `fabric-chaincode-evm` and run solidity code over fabric network. Moreover the dapp client talk with ethereum public blockchain network, used as network endpoint api supplied by Infura.

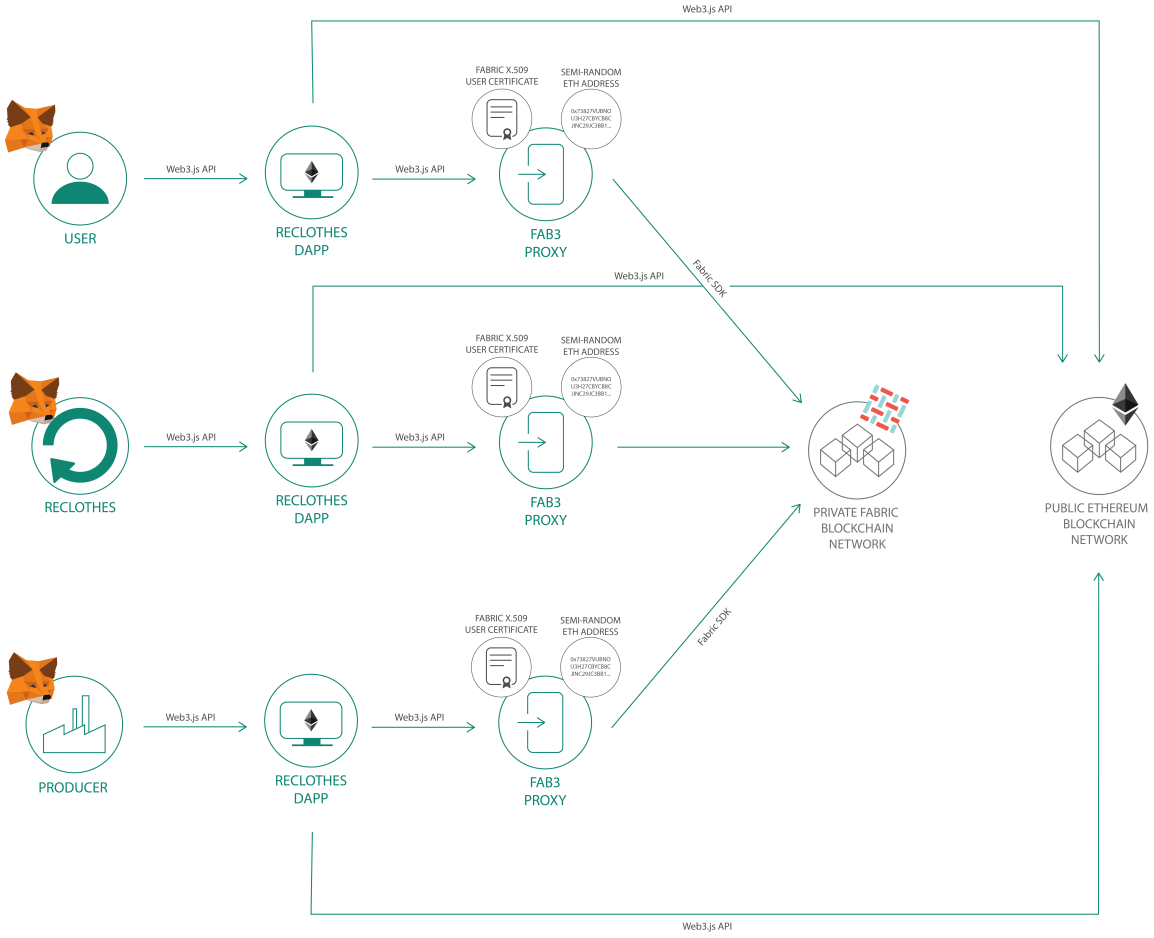

Figura 3.2. Architectural Flow

3.3 Use Cases

3.3.1 UseCase 1 - User Side

As shown in **Figure 3.3** 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.

The Internal Flow of the **Send Box** macro process 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

The two process that start the transactions are the **Evaluation** performed by The Reclothes Admin and the **Purchase Items** performed by the Users over the platform store.

1. The **Evaluation** process works in the following way:
   (a) Reclothes Admin visualize the next pending request to be evaluated
   (b) Reclothes Admin evaluate it and set an amount value of Fabric points and ERC20 Token to be send
      i. The points are sent over Fabric network invoking a chaincode function
      ii. The ERC20 Token are send over Ethereum network (Ropsten), sending the token to the ETH Address stored in the smart contract during the User Registration Phase.
   (c) Once both transactions succed, both tokens are accredited and User could spend it
2. The **Purchase Items** process in a similar way:
   (a) User choose the items to purchase over the web-app store
   (b) Reclothes Admin evaluate it and set a value amount of Fabric points and ERC20 Token to send
      i. The points are sent over Fabric network invoking a chaincode function
      ii. The ERC20 Token are send over Ethereum network (Ropsten), sending the token to the ETH Address stored in the smart contract during the User Registration Phase.
   (c) Once both transactions succed, both token are accredited and User could spend it

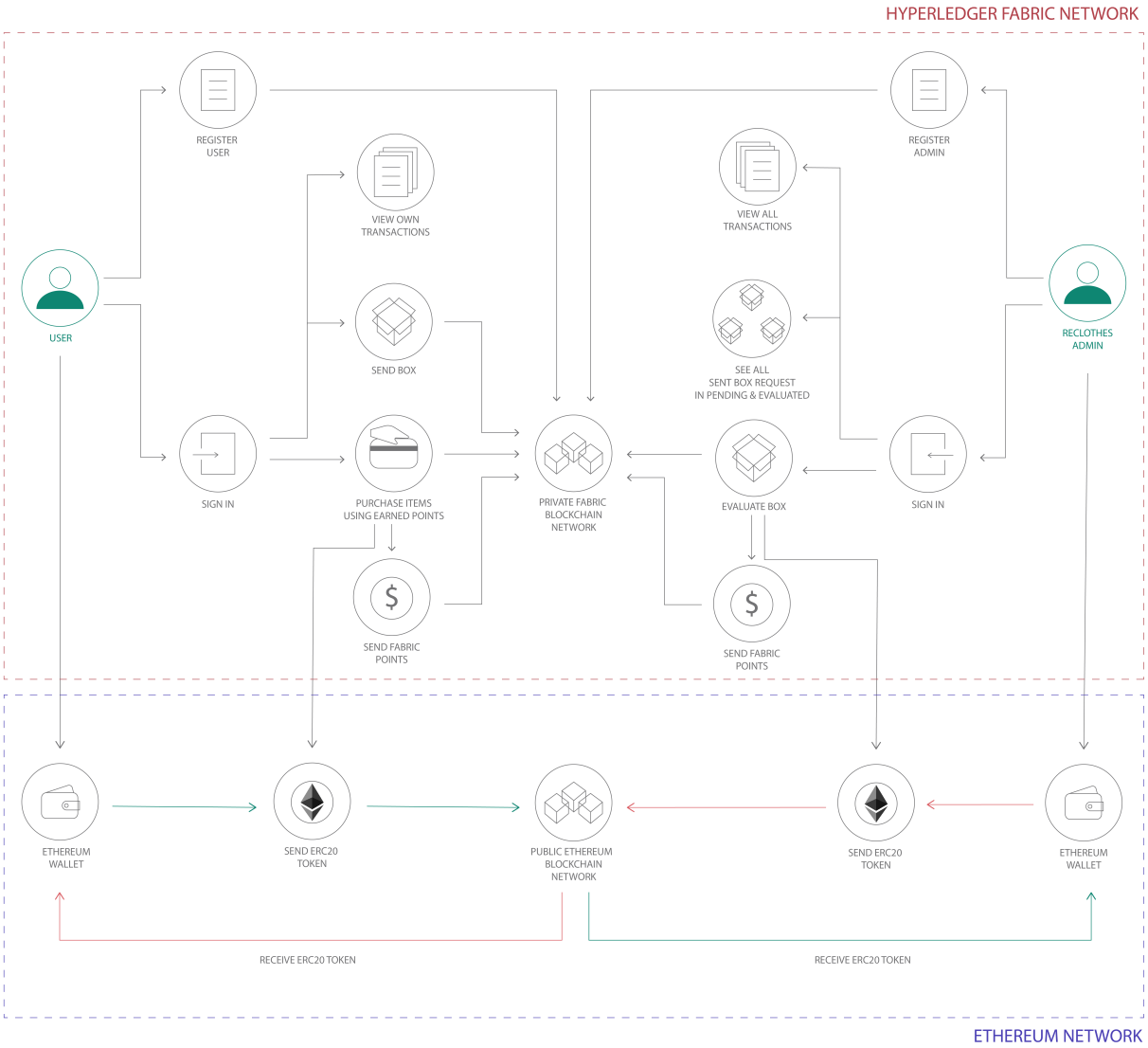

Figura 3.3. UseCase 1

3.3.2 UseCase 2 - Producer Side

The Use case diagram shown in **Figure 3.4** describe how Reclothes Admin and Producer interact each other. In this case all the features are performed over the Hyperledger Fabric network.

We could split the flow into two subflow, the first one from Reclothes to Producer, that we could identify with the two actions *Send Box* and *Purchase Box*, on the other hand the second subflow from Producer to Reclothes is summarize into *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.

1. **from Reclothes to Producer**

   (a) **Send Box**

      i. Send Box with old materials to be recycled by the Producer Company

      ii. Receive **Regeneration Credits** based on the old materials evaluation

   (b) **Purchase Box**

      i. Reclothes Admin purchase Box by Producer Company relized with recycled materials, spending the Regeneration Credits

2. **from Producer to Reclothes**

   (a) **Evaluate Material**

      i. Producer Admin perform the evaluation of the materials received by Reclothes, after the evaluation the transactions of Regeneration Credits is performed by the Fabric chaincode

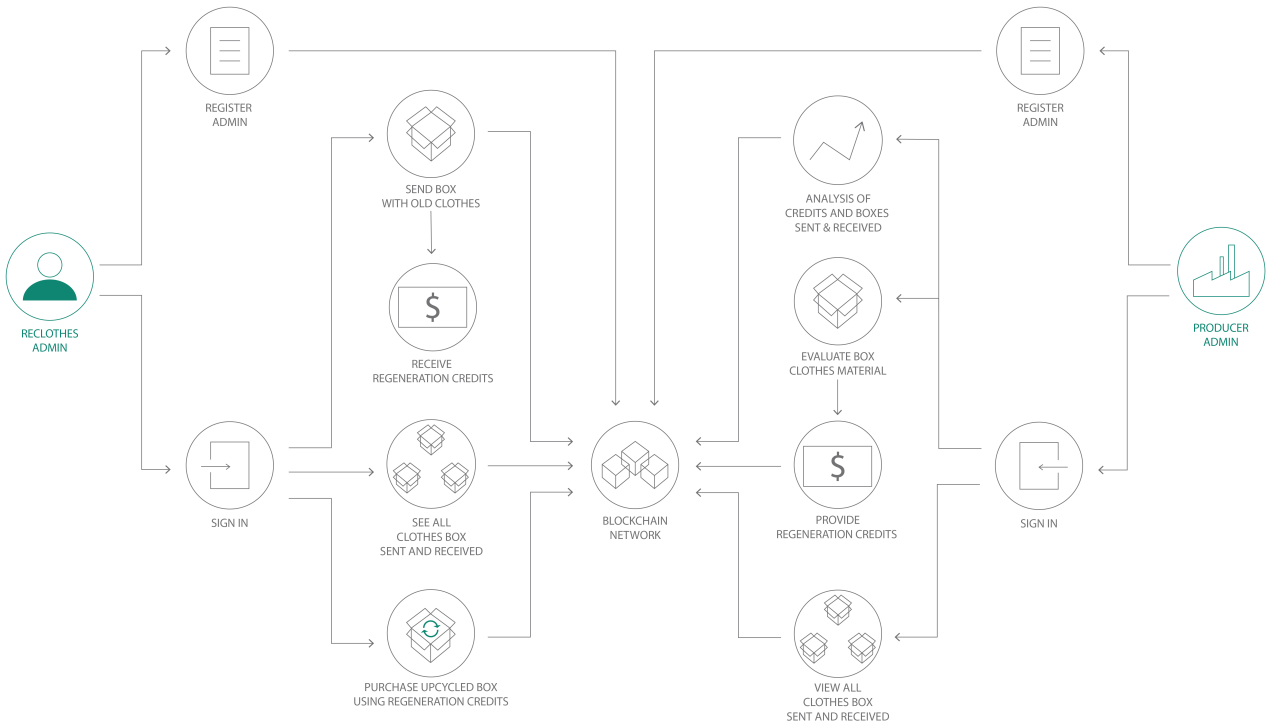

Figura 3.4. UseCase 2

3.4 Smart Contract

All the smart contract are developed in Solidity[1].

Both the Fabric contracts are use the `fabric-chaincode-evm` in order to allow Solidity code into fabric network. there's three smart contracts involved in the system, 2 run over the fabric private network and 1 over the ethereum public network.

1. **Hyperledger Fabric**
   (a) **User Contract**: handle the the User side, registration and interactions phase.
   (b) **Producer Contract**: handle the interaction from Reclothes to Producers.
2. **Ethereum**
   (a) **ERC20 Contract**: it's a standard smart contract with a Max Supply fixed to 100.000.000 . The Contract run over Ropsten network and we access to it through the Infura node.

3.4.1 User Contract

Data Structure 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 environment)
4             address publicAddress;  // external eth public address of User Admin
5             string firstName;
6             string lastName;
7             string email;
8             uint points;            // Fabric points amount
9             bool isRegistered;      // Flag for internal use
10            uint numTransaction;    // number of transactions performed
11            mapping(uint => PointsTransaction) userTransactions;
12            uint numBox;            // number of box transaction evaluated
13            mapping(uint => ClothesBox) box;
14        }
15
16        // model a admin
17        struct Admin {
18            address adminAddress;   // Admin address (inside fabric environment)
19            address publicAddress;  // external eth public address of Admin
20            string name;
```

[1] To 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

```
        bool isRegistered;      // Flag for internal use
    }

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

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

Transactions There's two functions that performs transaction from and to Reclothes:

1. **earnPoints**: It's an internal function called by `sendBox` function. It performs the fabric points transaction from Reclothes to User.
2. **usePoints**: performs the fabric points transaction when the User purchase item by Reclothes store.

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

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

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

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

    // add transction
    transactionsInfo.push(earnTx);

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

    usersTransactions[totTx] = earnTx;
    totTx++;

  }

```

```
28
29      //Update users with points used
30      function usePoints (uint _points) onlyUser(msg.sender) public {
31
32        // verify enough points for user
33        require(users[msg.sender].points >= _points, "Insufficient points");
34
35        // update user account
36        users[msg.sender].points = users[msg.sender].points - _points;
37
38        PointsTransaction memory spendTx = PointsTransaction({
39          points: _points,
40          transactionType: TransactionType.Redeemed,
41          userAddress: users[msg.sender].userAddress,
42          adminAddress: 0
43        });
44
45        // add transction
46        transactionsInfo.push(spendTx);
47
48        users[msg.sender].userTransactions[users[msg.sender].numTransaction] =
              spendTx;
49        users[msg.sender].numTransaction++;
50
51        usersTransactions[totTx] = spendTx;
52        totTx++;
53      }
```

3.4.2 Producer Contract

Data Structure 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 fabric
                environment)
4           address publicAddress;  // external eth public address of Producer Admin
5           string name;            // Producer admin name
6           bool isRegistered;      // Flag for internal use
7           uint numBox;            // number of box transactions evaluated
8           uint pointsProvided;    // amount of points provided by own evaluations
9           mapping(uint => ClothesBox) box;
10      }
11
12      // model a admin
13      struct Admin {
14          address adminAddress;   // Admin address (inside fabric environment)
15          address publicAddress;  // external eth public address of Admin
16          string name;            // Admin name
17          bool isRegistered;      // Flag for internal use
18          uint numBox;            // number of box transaction evaluated
19          uint creditSpent;       // amount of points provided by own evaluations
20          mapping(uint => ClothesBox) box;
21      }
```

```
22
23      struct ClothesBox {
24          address adminAddress; // reclothes-producer Admin
25          uint tshirt;          // Number of item
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      }
```

Transactions There's two functions that perform transactions from and to Reclothes:

1. **evaluateBox**: function called by Producer to evaluate box materials, sent by Reclothes, to send Regeneration Credits.
2. **buyUpcycledBox**: function called by Reclothes Admin that spend earned Regeneration Credits to purchase upcycled clothes by Producer.

```
1       // Evaluate Old Box
2       function evaluateBox(uint _points) onlyProducer() public {
3           //check correct pending request index
4           require(evaluatedIndex < pendingIndex, "No more pending request");
5
6           //check if evaluation is done
7           require(!pendingBox[evaluatedIndex].isEvaluated, "Request just evaluated")
                ;
8
9           //pop pending request
10          ClothesBox storage box = pendingBox[evaluatedIndex];
11
12          //update box transaction
13          box.isEvaluated = true;
14          box.points = _points;
15
16          //add evaluated box
17          evaluatedBox[evaluatedIndex] = box;
18          evaluatedIndex++;
19
20          debtPoints += _points;
21          totPointsProvided += _points;
22      }
23
24      function buyUpcycledBox(uint _tshirt, uint _pants, uint _jackets, uint _other,
             uint _points) onlyAdmin() public {
25          require(debtPoints >= _points, "Not enought credits accumulated in old
                material boxes");
26
27          ClothesBox memory box = ClothesBox({
28           adminAddress: msg.sender,
29           tshirt: _tshirt,
30           pants: _pants,
31           jacket: _jackets,
32           other: _other,
33           isEvaluated: true,
34           points: _points
35          });
36
```

```
37          admins[msg.sender].box[admins[msg.sender].numBox] = box;
38          admins[msg.sender].numBox++;
39          admins[msg.sender].creditSpent += _points;
40
41          //add upcycled box
42          upCycledBox[upCycledIndex] = box;
43          upCycledIndex++;
44
45          debtPoints -= _points;
46          totBoxNew++;
47      }
```

3.4.3 ERC20 Contract

It's a standard ERC20 token with the following features:

- **Symbol**: CO2
- **Name**: CarbonToken
- **Total supply**: 100000000
- **Decimals**: 18

The main features of the contract are describer by ERC20 interface

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

3.5 Network Architecture

The **Figure 3.5** show the main components of own network architecture build for the application. It includes:

1. **1 Orderer** organization with *1 ordered* running
2. **3 Organizations** each with 1 peer, Peer0, running
   (a) *Org1*: User Organization
   (b) *Org2*: Reclothes Admin Organization
   (c) *Org3*: Producer Organization
3. **2 Channels**
   (a) *Chanel12*: It's the cannel between Org1 and Org2 and allow the comunication between User and Reclothes
   (b) *Chanel23*: It's the cannel between Org2 and Org3 and allow the comunication between Reclothes and Producer

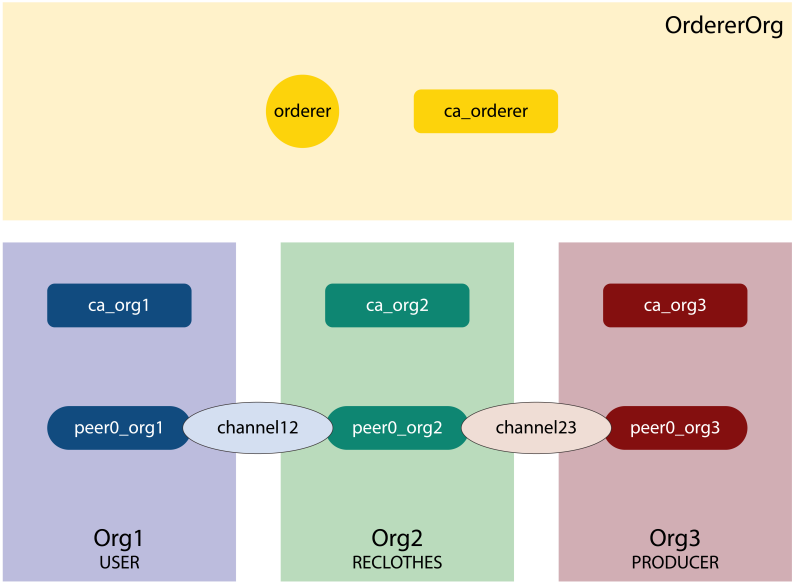

Figura 3.5. Fabric Network

3.5.1 Fabric Network

The **Figure 3.6** 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.
- **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.

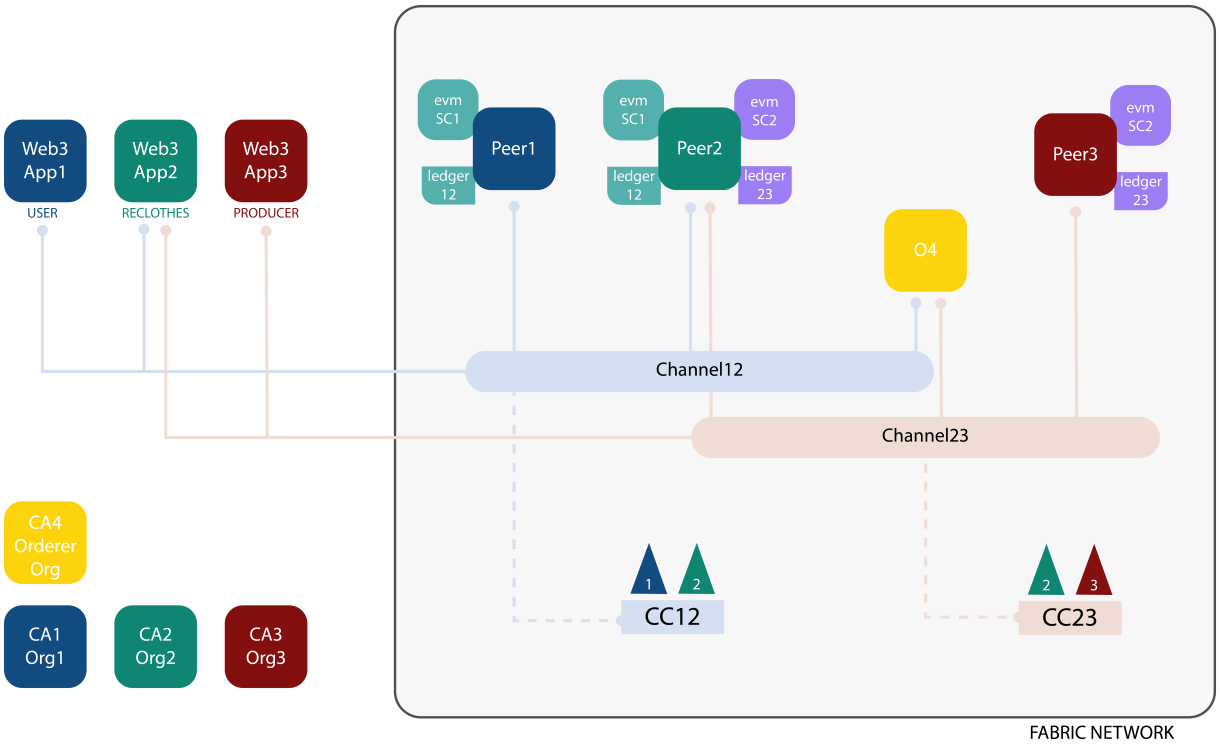

Figura 3.6. Fabric Network Components

The *chaincode* it's invoked calling the evm chaincode by the *App* Client, using the channel communication. Than 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. 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

To design and set up network components and rules, It's wrote into the `config.yaml` file. The network is structured in the following lines of code:

```
Organizations:
- &OrdererOrg
    Name: OrdererOrg
    ID: OrdererMSP
    MSPDir: crypto-config/ordererOrganizations/example.com/msp
    Policies:
        Readers:
            Type: Signature
            Rule: "OR('OrdererMSP.member')"
        Writers:
            Type: Signature
            Rule: "OR('OrdererMSP.member')"
        Admins:
            Type: Signature
            Rule: "OR('OrdererMSP.admin')"

- &Org1
    Name: Org1MSP
    ID: Org1MSP
    MSPDir: crypto-config/peerOrganizations/org1.example.com/msp
    Policies:
        Readers:
            Type: Signature
            Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
        Writers:
            Type: Signature
            Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
        Admins:
            Type: Signature
            Rule: "OR('Org1MSP.admin')"
    AnchorPeers:
        - Host: peer0.org1.example.com
          Port: 7051

- &Org2
    Name: Org2MSP
    ID: Org2MSP
    MSPDir: crypto-config/peerOrganizations/org2.example.com/msp
    Policies:
        Readers:
            Type: Signature
            Rule: "OR('Org2MSP.admin', 'Org2MSP.peer', 'Org2MSP.client')"
        Writers:
            Type: Signature
            Rule: "OR('Org2MSP.admin', 'Org2MSP.client')"
        Admins:
            Type: Signature
            Rule: "OR('Org2MSP.admin')"
    AnchorPeers:
        - Host: peer0.org2.example.com
          Port: 8051

- &Org3
    Name: Org3MSP
    ID: Org3MSP
    MSPDir: crypto-config/peerOrganizations/org3.example.com/msp
    Policies:
        Readers:
```

```
59                  Type: Signature
60                  Rule: "OR('Org3MSP.admin', 'Org3MSP.peer', 'Org3MSP.client')"
61              Writers:
62                  Type: Signature
63                  Rule: "OR('Org3MSP.admin', 'Org3MSP.client')"
64              Admins:
65                  Type: Signature
66                  Rule: "OR('Org3MSP.admin')"
67          AnchorPeers:
68              - Host: peer0.org3.example.com
69                Port: 9051
70
71
72                                          ...
73                                          ...
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     Channel23:
104         Consortium: SampleConsortium
105         <<: *ChannelDefaults
106         Application:
107             <<: *ApplicationDefaults
108             Organizations:
109                 - *Org2
110                 - *Org3
111             Capabilities:
112                 <<: *ApplicationCapabilities
```

Run of the network

The Hyperledger Fabric network run inside **Docker container**, to authomate the running of the network I create a script that execute the following step:

1. **Create Cryptographic Artifact**: 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.

2. **Run Docker Container**: Once the crypto materials is created than it must run the docker containers for the components of the networks. The **Figure 3.7** shows the related output.

3. **Orgs initializations and channels join**: 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 3.8** show the related output.

4. **Chaincode Installation**: Once all the network is run we are going to install tha chaincode over the peers. In own case we are going to install `fabric-chaincode-evm`. **Figure 3.9** show the related output.

```
################################################
##### 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
1c73cc686c1b        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
8ba3808d6a13        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

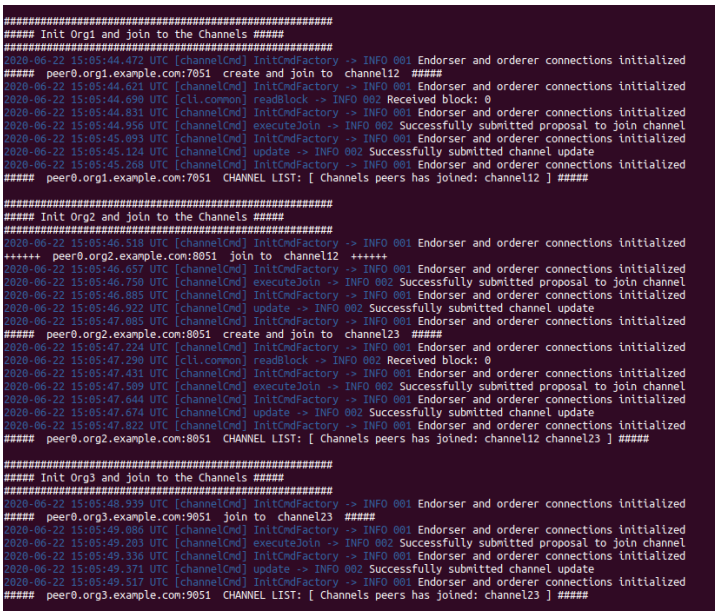

Figura 3.8. Orgs initializations and channels join

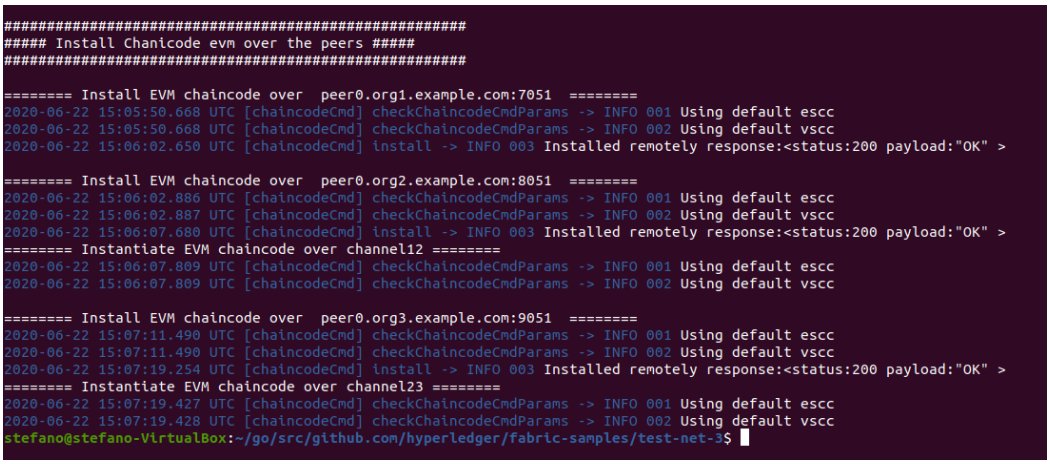

Figura 3.9. Chaincode Installation

3.5.2 Ethereum Network - Ropsten

To run the ERC20 token it's used the testnet Ropstan 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 3.10** 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.

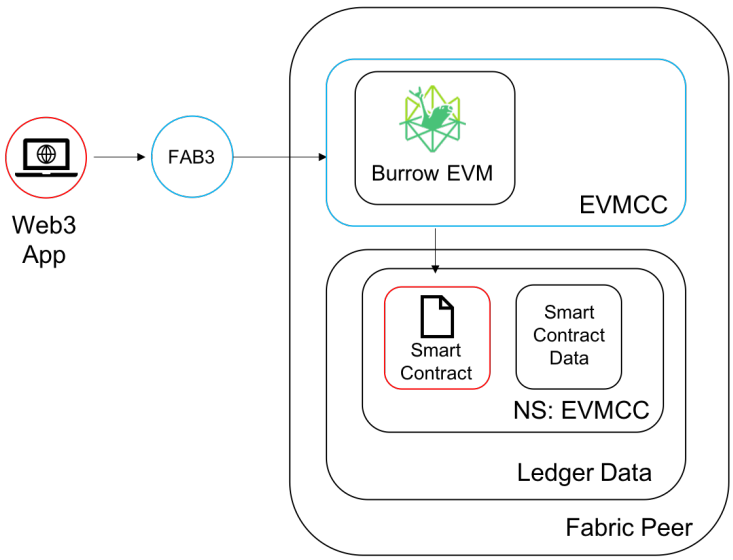

Figura 3.10. End To End

Chaincode Invocations

The **Figure 3.11** 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.

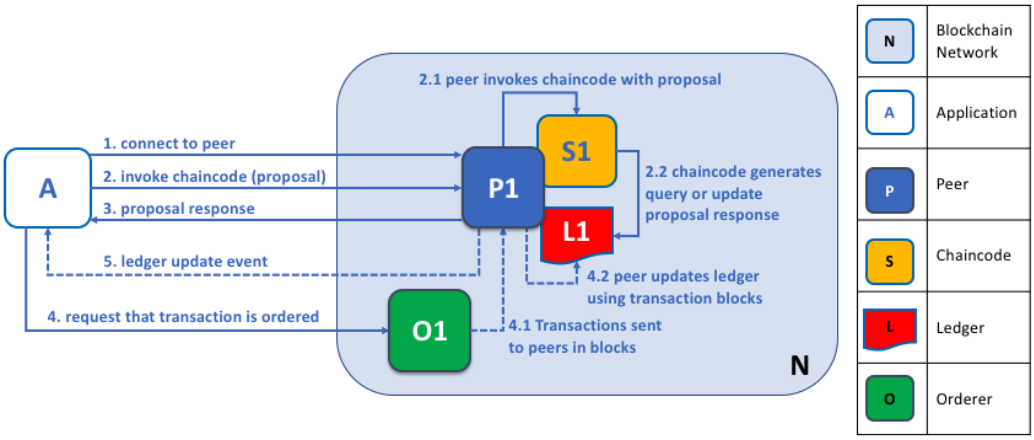

Figura 3.11. Smart Contract Invocation Process

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

```
# Environment variables required for setting up Fab3
export FAB3_CONFIG=${GOPATH}/src/github.com/hyperledger/fabric-chaincode-evm/
    examples/network-sdk-config.yaml
export FAB3_USER=User1
export FAB3_ORG=Org1
export FAB3_CHANNEL=channel12
export FAB3_CCID=evmcc
export FAB3_PORT=5000
```

Once the fab3 is set up and the instance is in running we could perform chaincode invocation using our Dapp. The **Figure 3.12** 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.

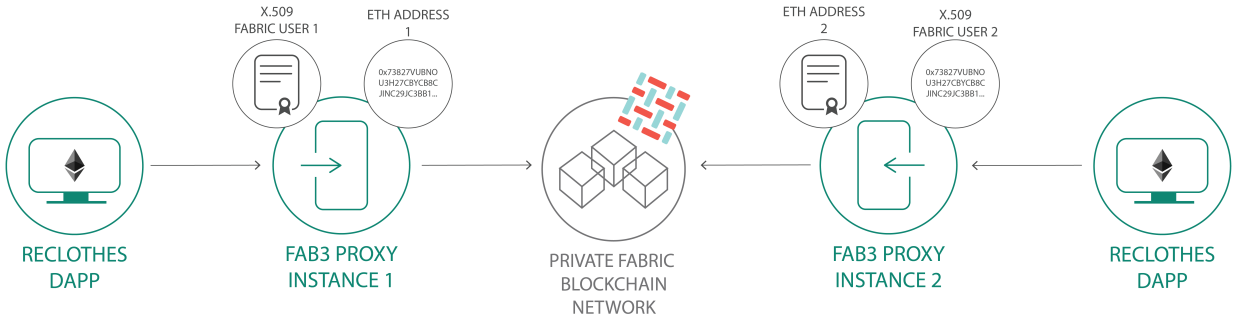

Figura 3.12. Fab3 Proxy Flow

The **Figure 3.13** 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 3.11.

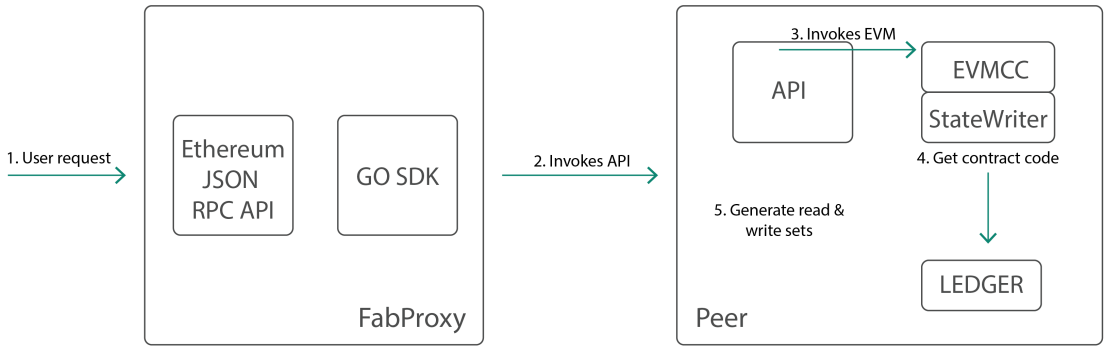

Figura 3.13. Fab3 Invocation Process

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 comunication 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**, **Reclothes 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 adresses to be reffered during the app running.
      i. **UserContract.js**: running the script using `node .js file`, it return the address of the deployed contract. The **Figure 3.14** 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 3.14.
2. **dapp.js**: there's the core file that handle the contracts invokations, set up the contract address referance, 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`.

Once is all set up we could run the web-app with the command `npm start`, there's an initializzation phase and that the app is ready to use and in running of over the specified PORT. The **Figure 3.15** shows the output.

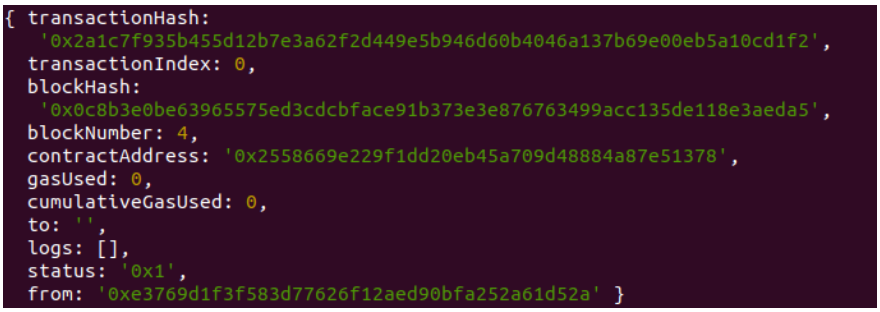

Figura 3.14. Deploy User Contract

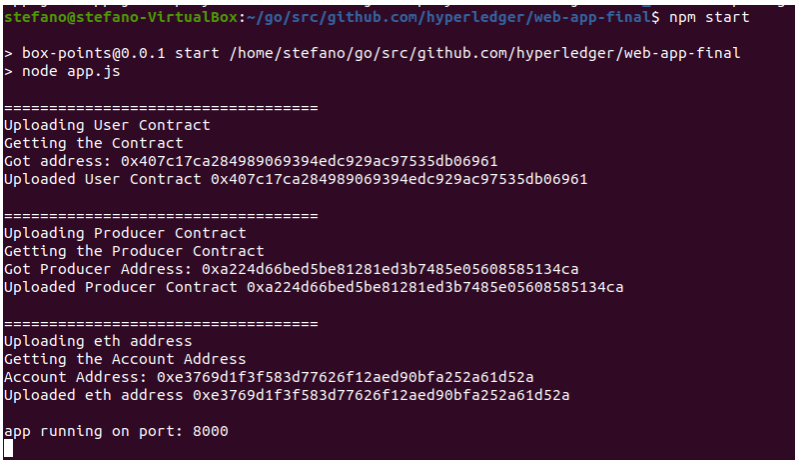

Figura 3.15. App Running

3.7.3 Views

Homepage

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

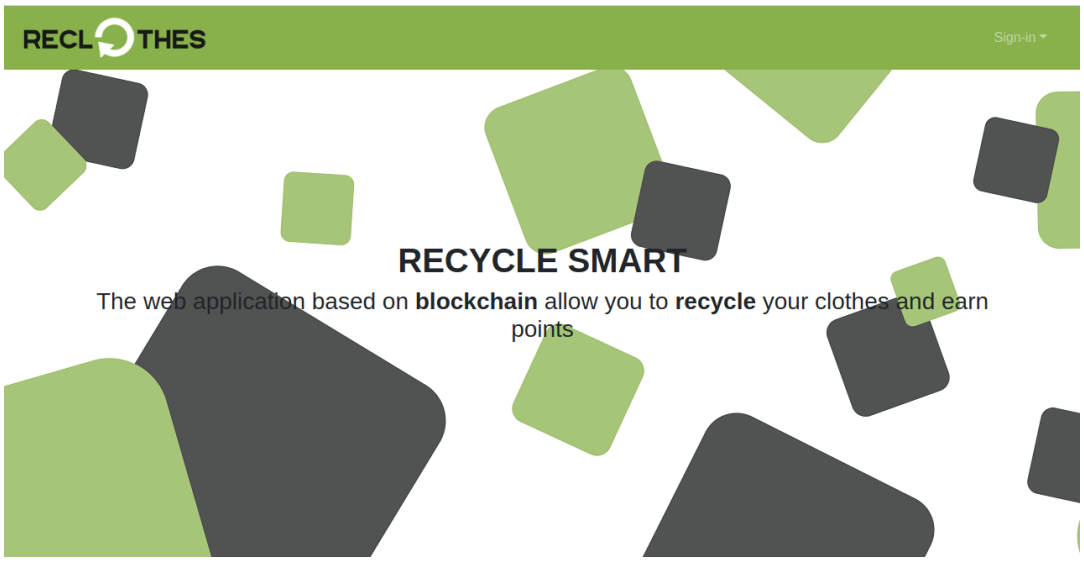

Figura 3.16. Home

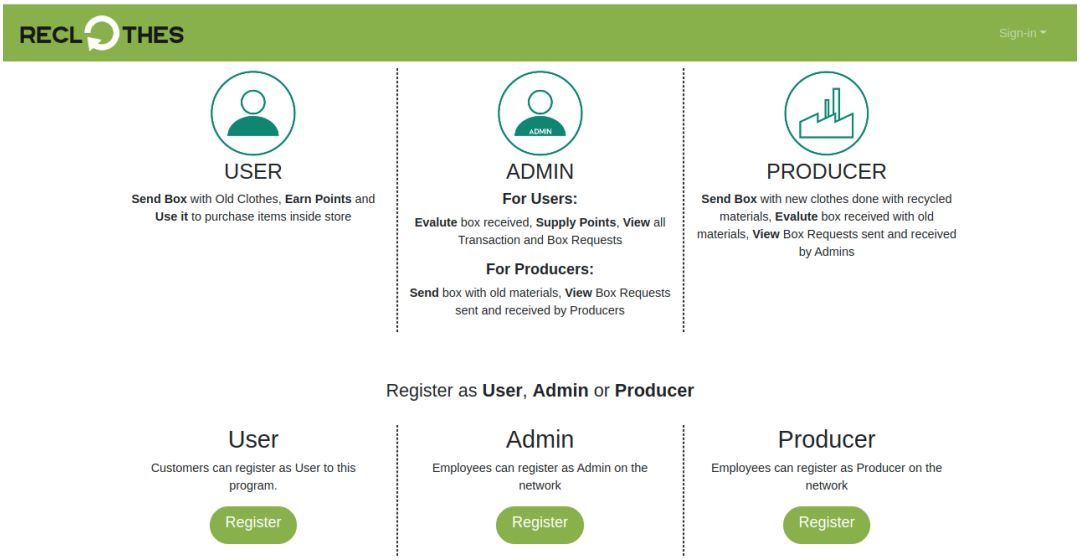

Figura 3.17. Registration Phase

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 3.19** 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 3.20** 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.

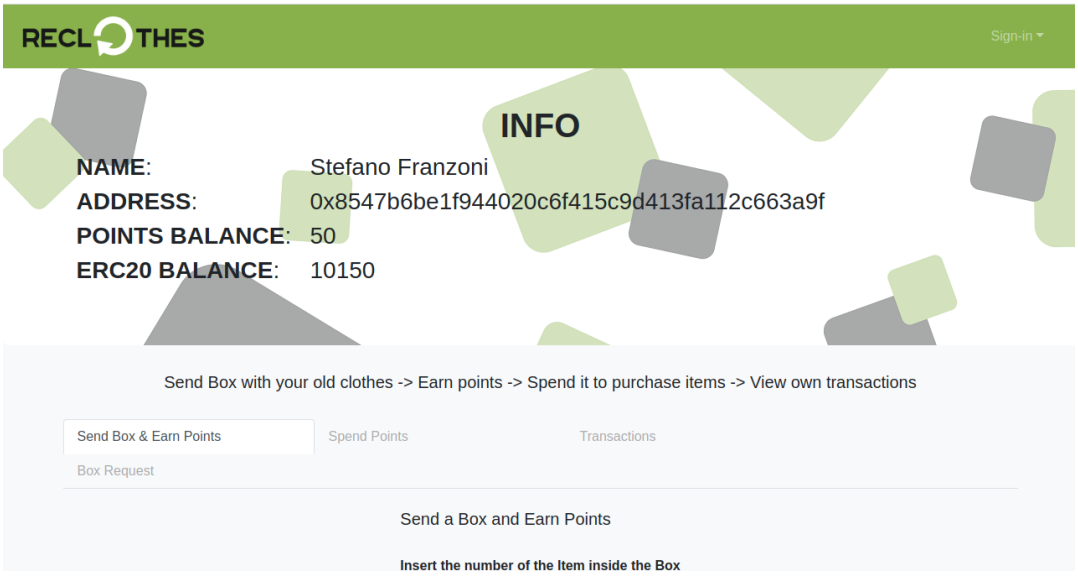

Figura 3.18. User Info

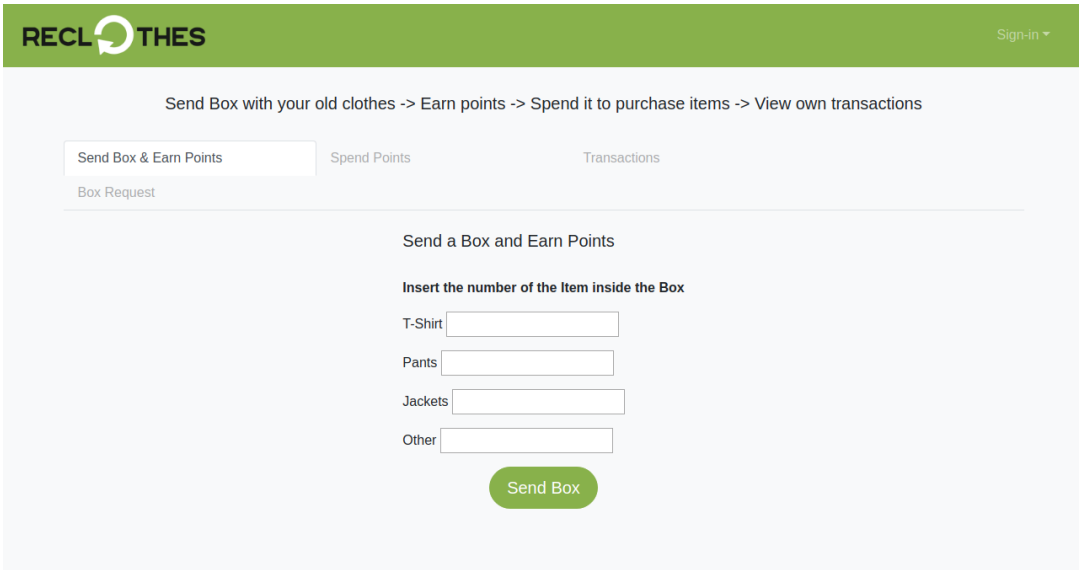

Figura 3.19. Send Box

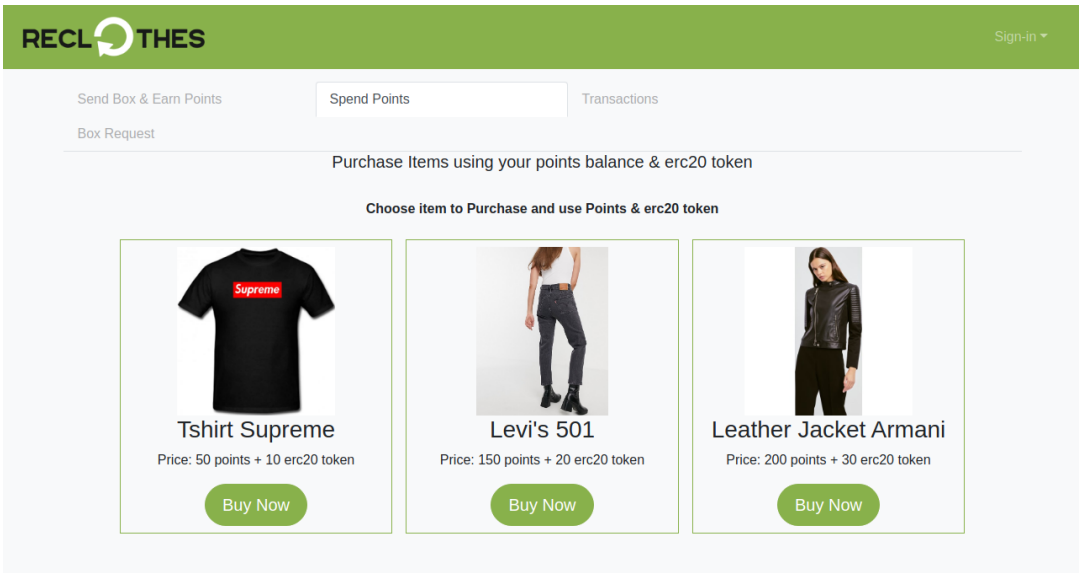

Figura 3.20. Purchase Clothes

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 releated to the User type to be handled and there's a streight distinctions about Users and Producers.

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

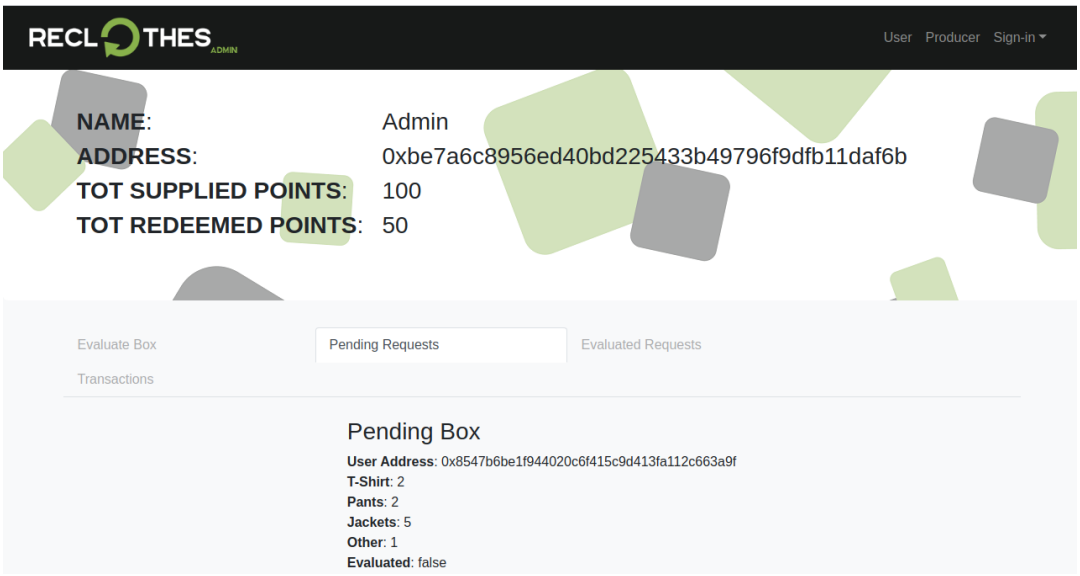

Figura 3.21. Admin Info

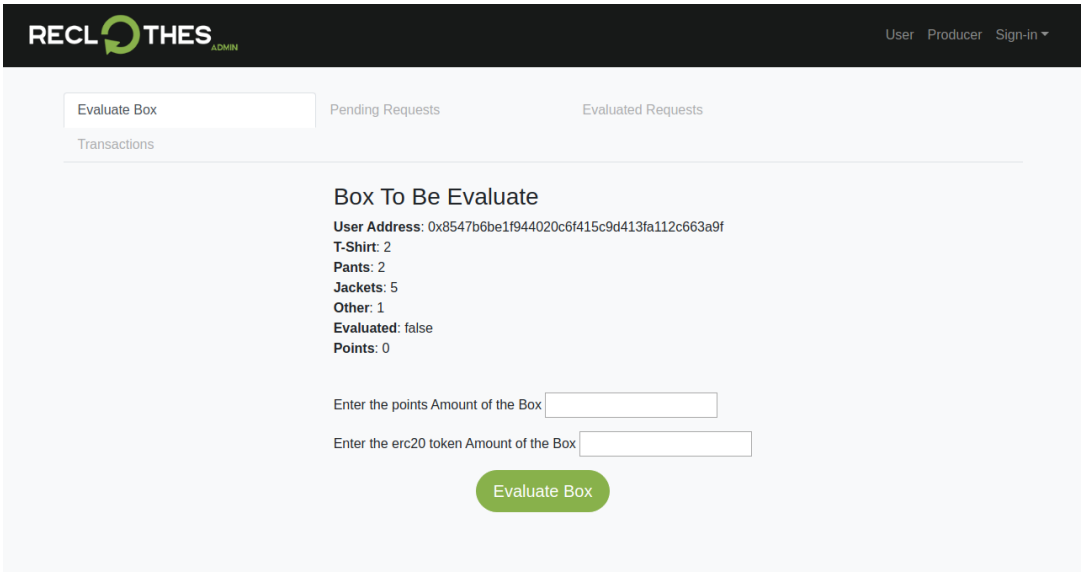

Figura 3.22. Evaluate Box

Admin For Producers This section show the view of the Admins that handle Producer side. The Figure 3.24 show all the Admin for Producers informations.

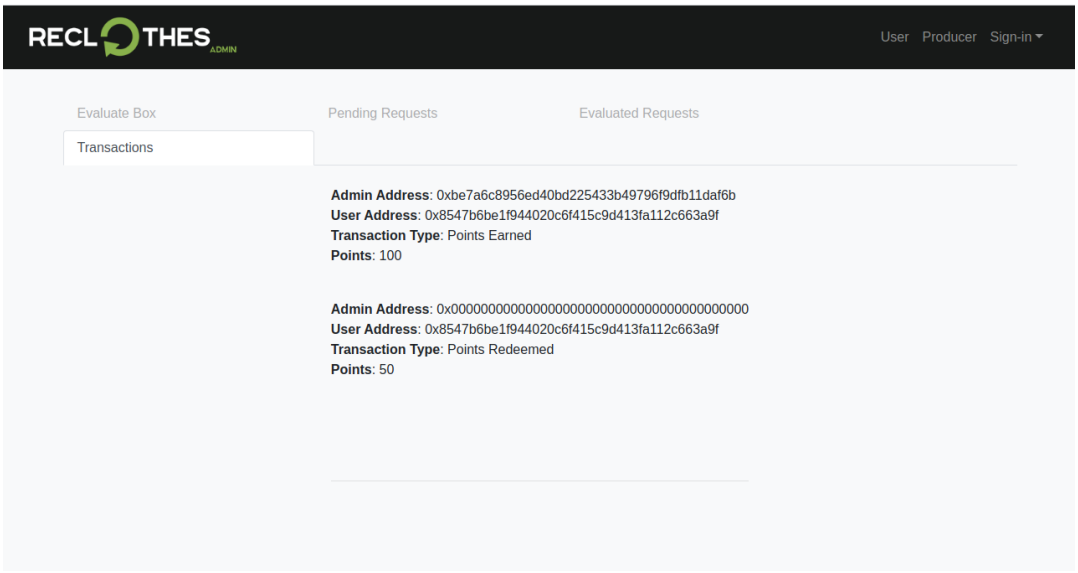

Figura 3.23. Transactions

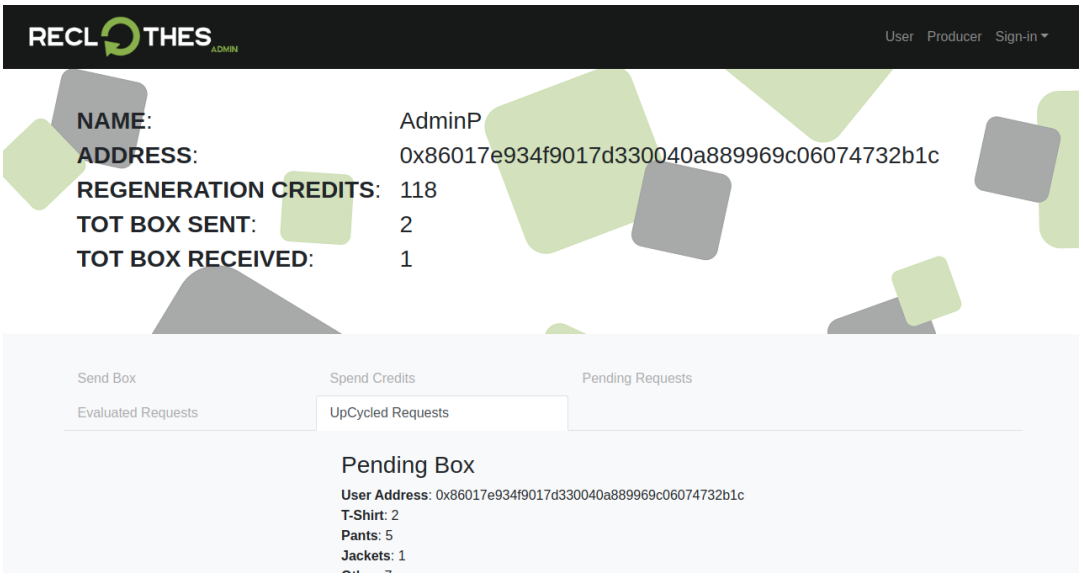

Figura 3.24. Admin for Producers Info

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.

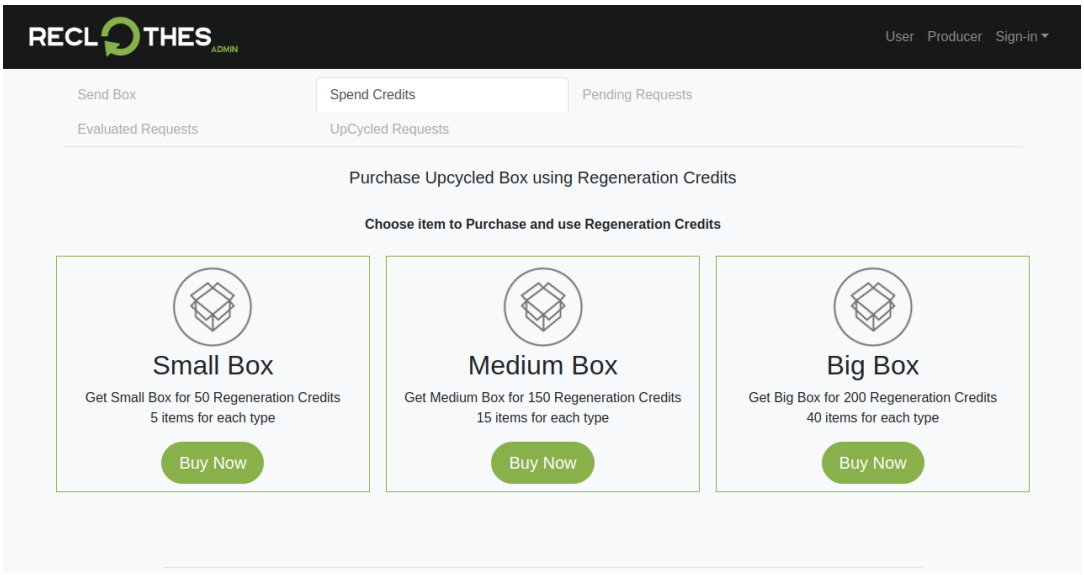

Figura 3.25. Admin Spend Regeneration Credits

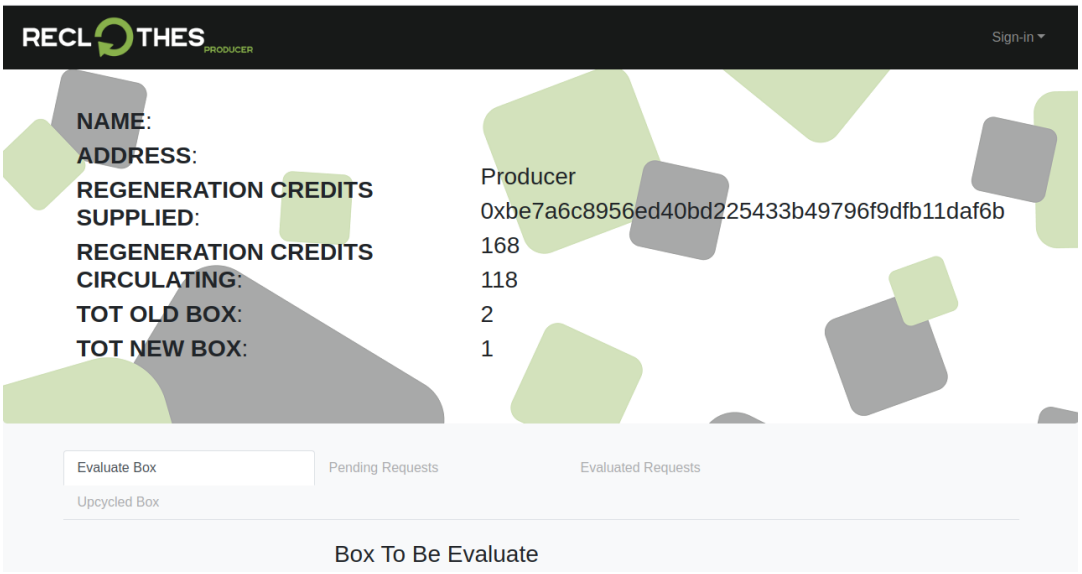

Figura 3.26. Producers Info

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.
  - **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.

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

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 4.1** show the log when User Perform the *Send Box* action.

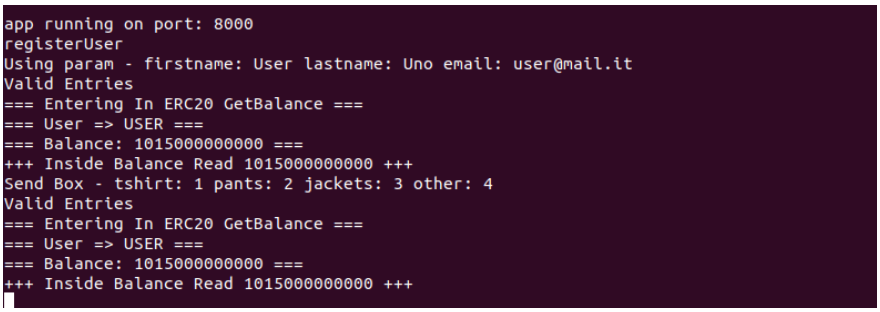

Figura 4.1. User Send Box

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 4.2** and **4.3** 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.

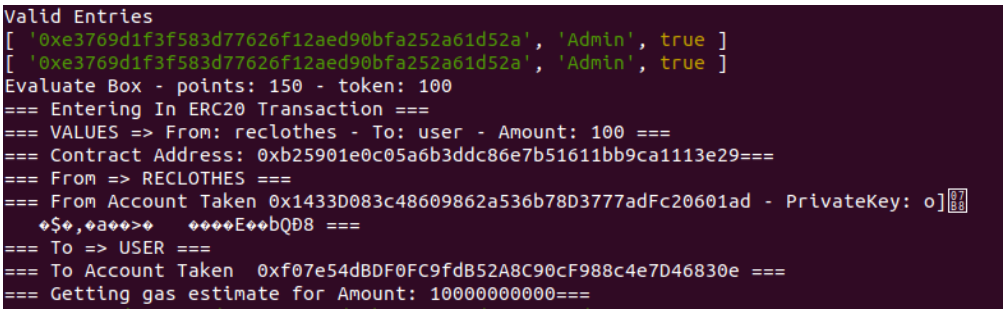

Figura 4.2. Init Transaction from Reclothes to User

The Figure 4.4 and Figure 4.5 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.

```
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

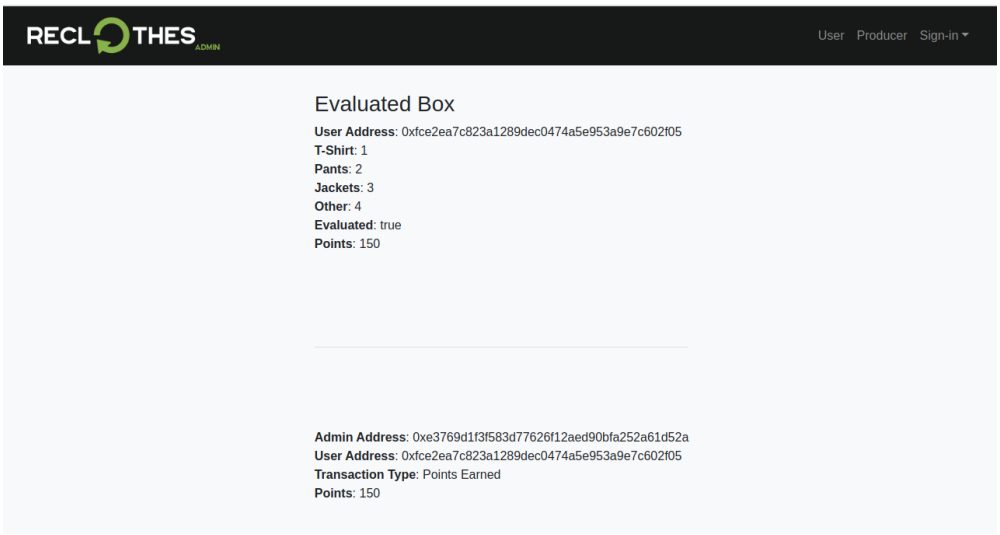

Figura 4.4. Fabric transaction history

4.2.2 Use Case 1 - Unit Test 2

Purchase Item

The **Figures 4.6** 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:

- **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.

Figura 4.5. Ethereum transaction over etherscan

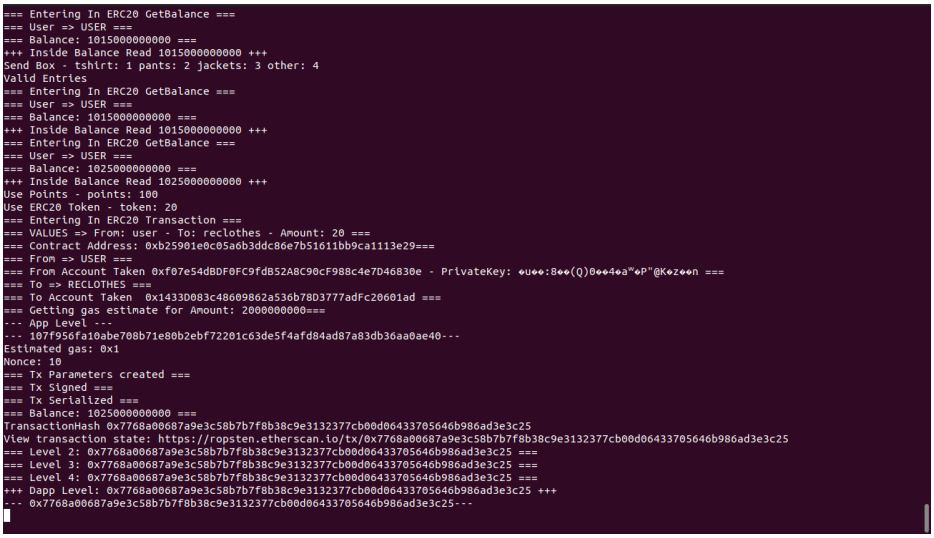

Figura 4.6. Transaction from User to Reclothes

- **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:
  - **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 4.7** 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

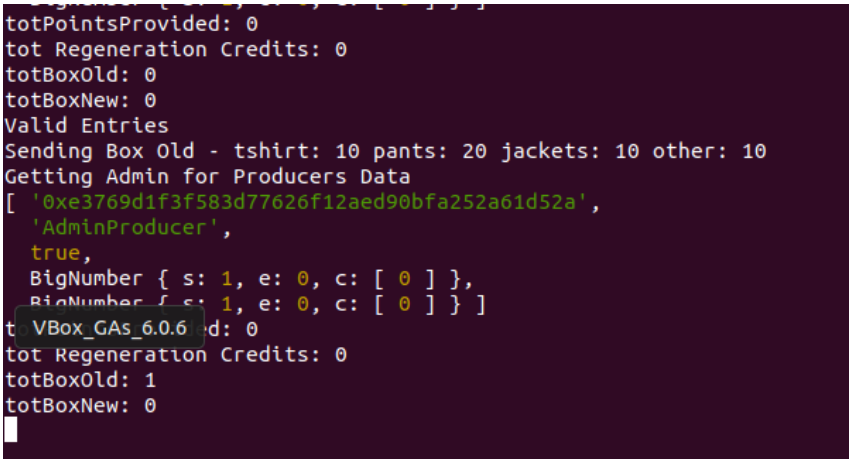

Figura 4.7. Admin Send old clothes

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 4.9** shows the output of the evaluation process.
Once the evaluation process is archived and the Regeneration Credits is section from Producer to Admin, **Figure 4.9** show the info update of the Admin for Producer.

Purchase Upcycled Material

Once the Admin sent box with old clothes and the evaluation process is archived, Admin has a Regeneration Credits amount to e 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 4.11** show the output of purchase process and the Tot Regeneration Credits update after purchase box process is performed.
The **Figure 4.12** show the the update of Producer Informations, the circulating Regeneration Credits amount is changed and the Tot Box New number too.

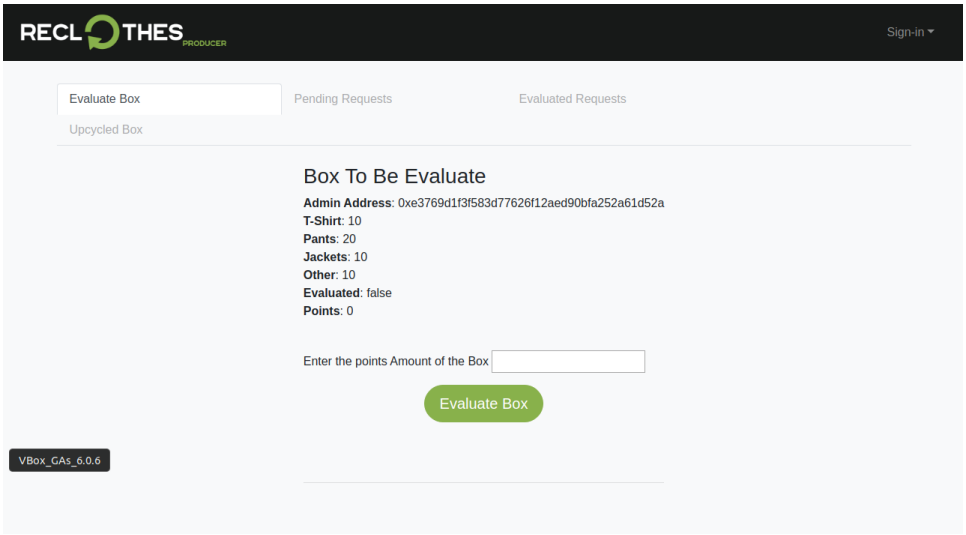

Figura 4.8. Box to be evaluated

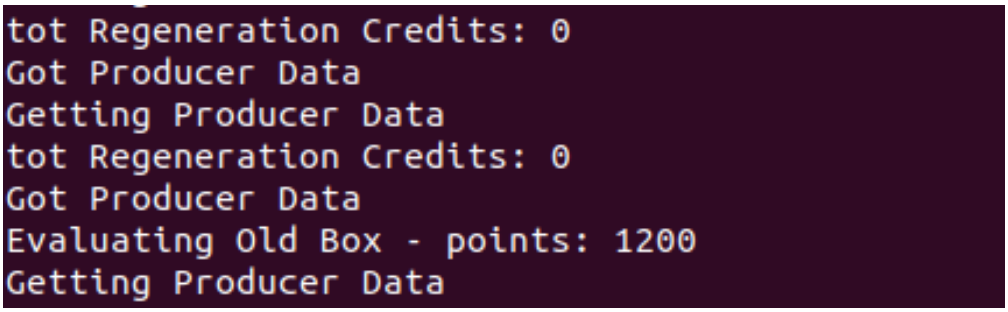

Figura 4.9. Producer Evaluate old materials

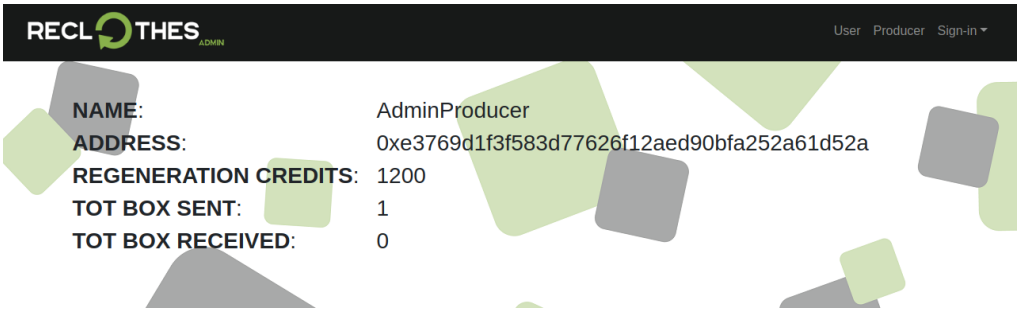

Figura 4.10. Admin Info update

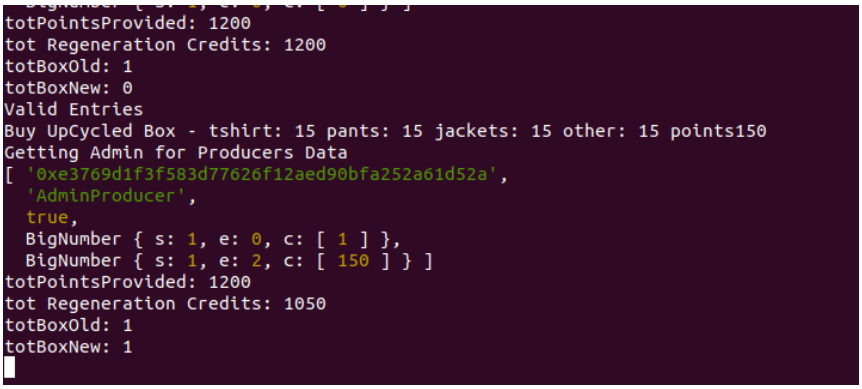

Figura 4.11. Purchase Recycled Clothes

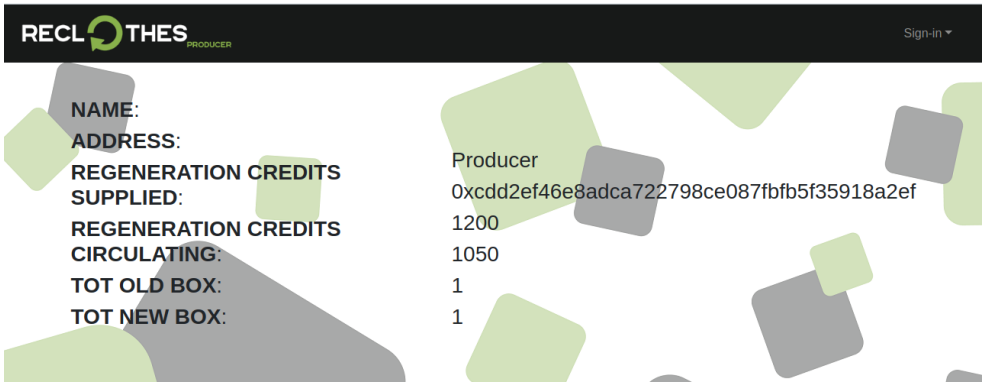

Figura 4.12. Producer Infos Update