Covid-19 Supply Chain

Andrea Trianni - 1806198
Marco Adriani - 1530783

February 2022

Contents

1 Preface

The pandemic of the last two years is affecting the world, in changing our lives and our habits. One of the strongest weapons we actually have is the vaccine, which have saved and is still saving a lot of lives. But those vaccines, and their distribution, is centralized in the hands of few manufacturers and few distributors, possibly causing a loss of transparency and trust in those companies. For this main reason the COVID-19 Supply Chain DApp was developed by our team, in order to give more transparency to the process of distribution and administration of the vaccines.

Our team is composed by Andrea Trianni and Marco Adriani. While the back-end is written by both of us, the front-end was developed by Andrea. Marco, instead, was the one who draw the diagrams and mainly wrote this paper.

In the section called ”Background” we explore a little bit the history and the aim of the Blockchain Technology. In section ”Presentation of the context” instead we will talk more deeply about the DApp, its aim and why we used blockchain, while in ”Software Architecture” and ”Implementation” we will show our work and our results. In the last two sections, ”Known issues and limitations” and ”Conclusions”, we will discuss the limits of our work and the next steps we can explore in the future.

2 Background

2.1 Blockchain: history and principles

Blockchain is a technology based on a decentralized network, in which nodes can in an efficiently and securely register transactions in a permanent and verifiable way. Distributed systems were known since 1990s, and something like modern blockchain was initially mentioned in a paper during the same period. However it was just with the famous article of Satoshi Nakamoto (pseudonymum of the one who wrote the paper) ”Bitcoin: A Peer-to-Peer Electronic Cash System” that this technology became popular and started to be studied and developed, and the first blockchain was created. From that moment on a lot of other blockchains rose associated with their digital assets, the cryptocurrencies. Now we have a great number of them, with different features and consensus algorithms. For some of them is also possible to write code above them, in order to create decentralized applications: the so called DApps. In particular one of the most popular right now is the Ethereum one, which we used in this project, with its associated programming language

Solidity.

One of the most important concepts in blockchain world is **transaction**. Parties own accounts and every transaction is identified by a transaction ID. In this environment a transaction involves a certain amount of the cryptocurrency which is transferred from one or more accounts to one or more another accounts. What is important is that those transactions are stored into **blocks**, which are mined and added to a chain that collects all the blocks in the same structure (from here the name "blockchain"). When an user wants to submit a new transaction, he sends it to the network and wait until some mining node will eventually add it to the new block of the blockchain, stored temporarily in a so called **transaction pool**.

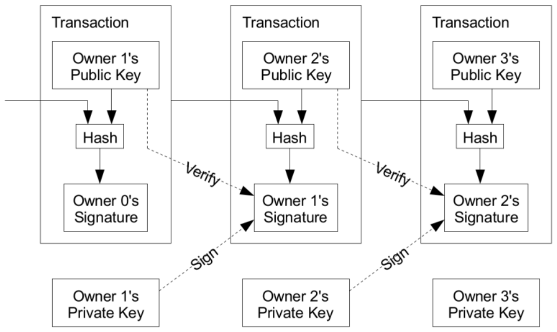


Figure 1: Bitcoin blockchain

The way in which those blocks are organized is perfect for a distributed network: every node has his own copy of the chain making cheating really hard and expensive. The **validity** of a transaction is checked considering if it is well formed, properly signed by the sender and checking if the sender has enough money to do it. If a transaction is not valid, the block will not be validated and so for miners it is not convenient to consider a not valid transaction at all.

How the next block is mined depends on the consensus algorithm we use. Most important are:

- **Proof of Work (POW):** in this kind of consensus algorithm nodes have to solve a really difficult (in terms of computational complexity) problem, and the first node who truly finds a solution is the one that can add the next block. It will be then rewarded with a certain amount of cryptocurrency. For example, This is the case of Bitcoin or

Ethereum. Blockchains based on PoW are generally more secure but they have to deal with pollution and wasting of energy.

- **Proof of Stake (POS):** in this kind of consensus algorithm, cryptocurrency owners validate blocks based on the number of coins a validator stakes. It was developed in order to minimize the use of energy of PoW, but it is generally more difficult to make it secure.

Once the consensus for one block is reached, all the nodes will upload their copy of the blockchain. Due to this fact a transaction in any block will be irreversible: the only way to delete it from the history is to reach consensus for another blockchain, which is really hard (in practice impossible) for just one node.

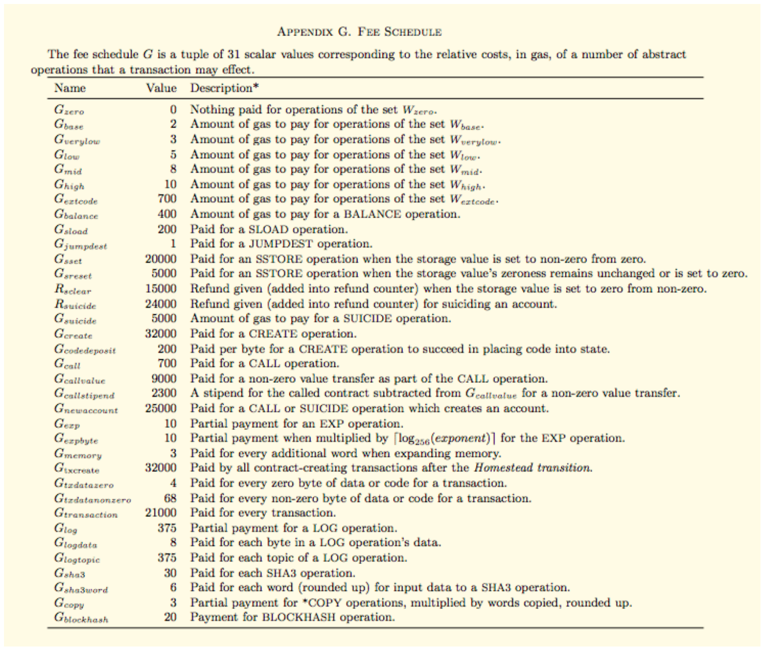
Another reason for which blockchains became so popular, especially for the economic point of view, is that they ensure a pseudononimity for the users (we can't know who is the person who made the transaction, but just his address). This fact ensures **transparency**, which is an important feature in order to guarantee trust.

2.2 Programming on the blockchain

As mentioned above, some blockchains permit to develop code and programs on it. This is the case of Ethereum, one of the most popular blockchains in the world, due also to the possibility to program the so called **smart contracts** on it.

In Ethereum we have two possible owners for an account: **Externally Owned Accounts**, the "classic" ones, and **Contract Accounts**. The latter can be triggered sending a transaction or by another contract with a message. This is a really odd account: it can be executed performing even complex operations, it creates other contracts, it contains a balance in ether and has a personal storage for its data. It also keeps as information the number of contract-creations made.

Everytime a contract account receives, it activates the code which is run on **EVM** (Ethereum Virtual Machine). It is a distributed virtual machine that acts like a single one, with its own code (the EVM bytecode) and high-level languages implemented in order to make easier to program on it. The most used is surely **Solidity**, and also this project is written in this language. Anyway, the execution of a contract has a cost, based on the computational and storage cost of its single operations: this cost is called **gas**. One of the most important things to keep in mind while programming in Solidity is indeed try to use less gas as possible, limiting the usage of


APPENDIX G. FEE SCHEDULE

The fee schedule G is a tuple of 31 scalar values corresponding to the relative costs, in gas, of a number of abstract operations that a transaction may effect.

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Figure 2: Gas cost of operations in Ethereum

high cost operations. Anyway the amount of gas we can spend in a single transaction is limited by a certain amount called **gasLimit**, and gas is purchased from sender's account balance. But it is fundamental to take in mind that gas is not Ether (the cryptocurrency of Ethereum): gas cost remained the same even if Ether price changes, and Ether can be traded by accounts. The reason above the idea of gas is making DoS attacks really expensive, defending the network from this sort of attack.

The way Ethereum permits the compiling and the execution of smart contracts brought a revolution into the blockchain world. Incentivized by the advantages of this kind of architecture, such as pseudononimity and transparency, a lot of DApps were born and started a new revolution in web programming, called **Web 3.0**. But another kind of entity arose: **digital tokens**, in particular the **NFT** ones. They are not currencies but something more like objects that can be traded or used in a certain environment. In Ethereum blockchain we can write smart contracts which define a specific token, and the most popular are the **Not-fungible** ones. They can represent a digital artwork, a token that can be spent to buy specific goods and so on.

But like we mentioned another important revolution is the development of the DApps. The main difference between them and the classical web applications is that we don't have anymore a client-server architecture, with a central entity controlling the interaction with the clients, but an application with the back-end wrote on a smart contract and a front-end based on it. In this way this kind of application has a decentralized architecture, avoiding a lot of possible issues such as trust on the central entity, more resistance in case of attacks (it's harder to attack a distributed network then a centralized one) and transparency.

3 Presentation of the context

3.1 Aim of the DApp

COVID-19 pandemic affected deeply our lives and habits. Vaccines are the most important weapon against it, but one of the main problem is the lack of confidence in the vaccines and pharmaceutical companies. This lack of confidence brought huge waves of protest all around the globe and a lot of people is actually refusing vaccine administration, causing more deaths and a slowdown in the health service all over the countries. This problem is not easy to solve: most of the countries, in Europe particularly, is making pressure to this part of population with restrictions and punishments forcing the administration of the vaccines, but in some cases rise just more rage and loss of trust in the institutions. Another very involved actor in this scenario are the pharmaceutical companies. They are often accused to manipulate data and to produce just for profit, causing a possible aversion from those companies. On the other hand is a matter of fact that the production is centralized over a few companies, and the distribution is totally controlled by the government, so the risk to have abuses is real.

Due to this possible scenario we propose our DApp "COVID-19 supply chain" is indeed a decentralized application with the goal to give more transparency and trust to the vaccine supply chain, from the pharmaceutical companies which deployed the dose to the hub in which it is administrated: in our DApp every movement is tracked down the blockchain, and the history of every batch is accessible to everyone.

So the aim of our DApp is to offer a system that can solve, or that at least helps to solve, the problem of the lack of trust in public health and pharmaceutical companies, offering a complete and transparent record everyone can access to. Also, every actor involved can update the status of a batch independently and autonomously, accordingly to their role in the chain.

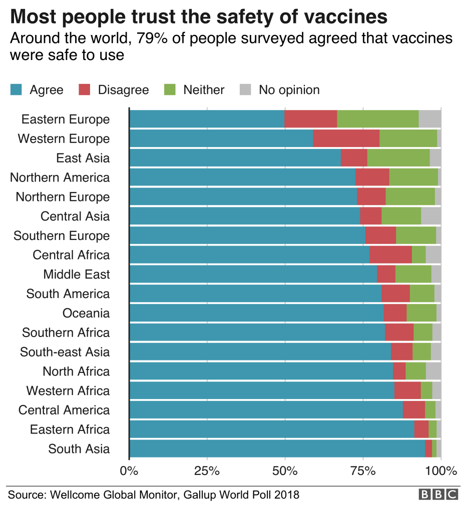


Figure 3: No-vax populations, image from BBC. Click here.

Also, it can be a good starting point for a different approach in the problem of ascertain if a citizen got the vaccine for real. Instead of having a big centralized system that collects the data of all the vaccination campaign, including the information about who is vaccinated, we can use our decentralized app to check it. In this way we could have a secure system to ensure that one citizen received his dose, without possibility of hacking in order to obtain fake certificates of vaccination.

3.2 Why the blockchain?

Blockchain can come to rescue to solve our issues. The main feature is that it ensures transparency and impossibility to modify the history of a batch. In this way, once a batch is inserted and its status is updated along its history, those informations can't be modified or deleted. Every vaccine dose is thus perfectly traceable and controlled, solving the problem of avoiding the manipulation of data by the main actors (political and economical) of the COVID-19 vaccines administration. Also, thanks to the blockchain, it will be impossible to fake a certificate of a vaccine administration.

Even if the contract is written in Solidity and it could run on a Ethereum blockchain, the fact that every writer is known combined with the necessity of public verifiability makes a **public permissioned blockchain** the best candidate to hosts our application.

4 Software Architecture

The software is based on a three tier architecture. There is a front-end side, available with a standard browser, a logic layer, represented by the smart contract, and a back-end side (the blockchain).
Starting from this last, the environment in which the back-end side is developed is **truffle**, a development environment based on **Ethereum** Blockchain. Truffle compiles and deploys the contract into the chain. We use it also to run Unit Test. The smart contract we have designed is written in **Solidity** language, stored in *CovidSupplyChain.sol*. We used **Ganache** in order to simulate on our machine the blockchain.
The webapp, instead, is based on a **NodeJS** project. The various pages are written as **ReactJS** Components, antoher library we used. We also exploit **Metamask** as a tool that connects the app with the smart contract, via RPC. In order to use Metamask we have used also the **web3** library.

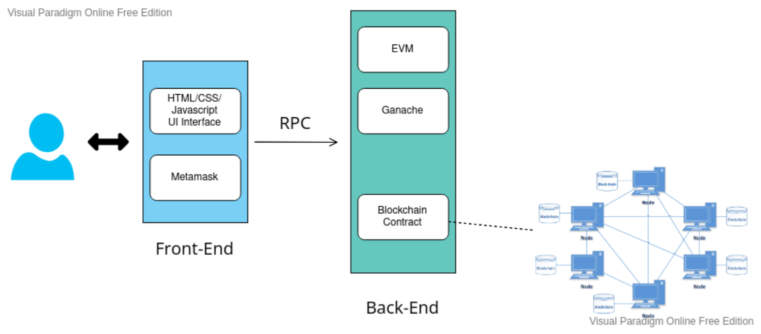

Visual Paradigm Online Free Edition
HTML/CSS/ Javascript UI Interface
Metamask
Front-End
RPC
EVM
Ganache
Blockchain Contract
Back-End
Node
Visual Paradigm Online Free Edition

Figure 4: Component diagram of the application

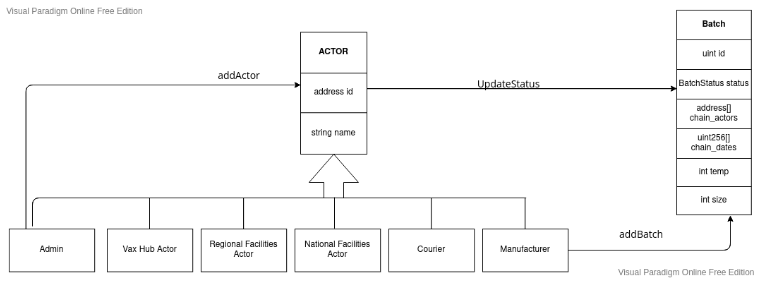


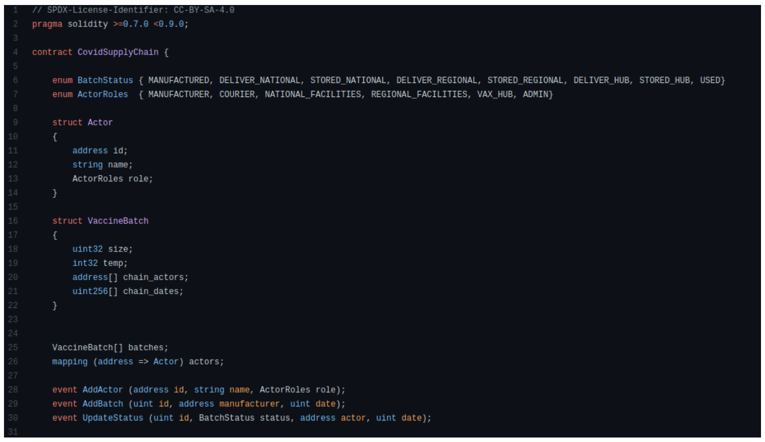
Figure 5: Concept diagram of the application

4.1 Back-end

The main core of the back-end, like as mentioned before, is the file ”Covid-SupplyChain.sol” in which the app contract is deployed. As we can see in Figure 4, main structures in the contract are **Actor**, representing an actor that use the system, and **VaccineBatch** which represents a specific vaccine dose. An Actor is represented by its id, its name and its role (manufacturer, courier, ecc.). A batch instead is composed by its size, its temperature and two vectors: the former is storing the sequence of actors which manipulated the batch during its history, the latter the corresponding dates in which their status changed.

Three are the events that can occur, all called by the corresponding function: event *AddActor* invoked by *addActor* function, event *AddBatch* invoked by function *addBtach* and event *UpdateStatus* invoked by *updateStatus* function.

Two another important functions are *getTimeline* and *getMyLastNBatch*. The former takes as input a *batch_id* and returns history info about the status the batch, while the latter returns information about the last n batch modified into the system by a given actor.



```
// SPDX-License-Identifier: CC-BY-SA-4.0
pragma solidity >=0.7.0 <0.9.0;

contract CovidSupplyChain {

    enum BatchStatus { MANUFACTURED, DELIVER_NATIONAL, STORED_NATIONAL, DELIVER_REGIONAL, STORED_REGIONAL, DELIVER_HUB, STORED_HUB, USED}
    enum ActorRoles  { MANUFACTURER, COURIER, NATIONAL_FACILITIES, REGIONAL_FACILITIES, VAX_HUB, ADMIN}

    struct Actor
    {
        address id;
        string name;
        ActorRoles role;
    }

    struct VaccineBatch
    {
        uint32 size;
        int32 temp;
        address[] chain_actors;
        uint256[] chain_dates;
    }


    VaccineBatch[] batches;
    mapping (address => Actor) actors;

    event AddActor (address id, string name, ActorRoles role);
    event AddBatch (uint id, address manufacturer, uint date);
    event UpdateStatus (uint id, BatchStatus status, address actor, uint date);

```

Figure 6: Part of the smart contract

The constructor of the contract add the first actor of the system, the owner, that is of course an admin. An admin is the only one who can enter into the Administration section (see below) and add other actors thanks to the modifier *onlyAdmin*.

4.2 Front-end

The WebApp is based on a NodeJS project. We wrote the front-end using the React Library, to exploit the Component concept, and lots of other amazing features. Every page of the website is based on a custom React component and has a CSS responsive layout in order to fit on various screen sizes.
We used the web3 library in order to connect the front-end to Metamask and therefore interact with the blockchain via RPC protocol.

The starting page of the app is the Home, from which an actor can access to the desired pages and use tools accordingly to their role and responsibilities. The only functionality that is accessible for everyone is the one in Scan Section in which it is possible to look at the history of the batches. We will explore pages and their functionalities in details in next section.

4.3 UML diagrams and use-cases

Now in this section we will present the logical structure behind the app, with the help of some UML diagrams. We start with the Use-Case diagram shown in figure 6: as we can see main actors here are the Admin, the only one who can actually add an actor. The Vaccine Actor who represents a courier or a worker in some facility or hub, who can update the status of a batch accordingly with his role. And the Manufacturer, which is a special Vaccine Actor that can also add a batch to the system. The User instead is an actor representing a general user who can access to the system and search information about batches.

Now let's see more in details, with the help of the sequence diagrams, in detail how the functions work.

We start with the addActor use-case. The sequence diagram is in Figure 8. In this case the actor wants to enter in the administration section and after that he wants to add a new actor. Metamask communicate with the contract and confirm if is okay or not, and if positive the actor is added to the system, and contract emits *AddActor* event (Figure 8).

Similar sequence is for the *updateBatch* use-case, the only thing that change is the actor involved (the check now is to control if it is the right actor to do the next move) and the kind of event emitted, in this case *UpdateBatch* (Figure 9).

Same thing for the *addBatch* use-case. In this case the only actor that is allowed is a manufacturer (also this time it is checked by Metamask that talks with the contract). The event emitted is *AddBatch* (Figure 10).

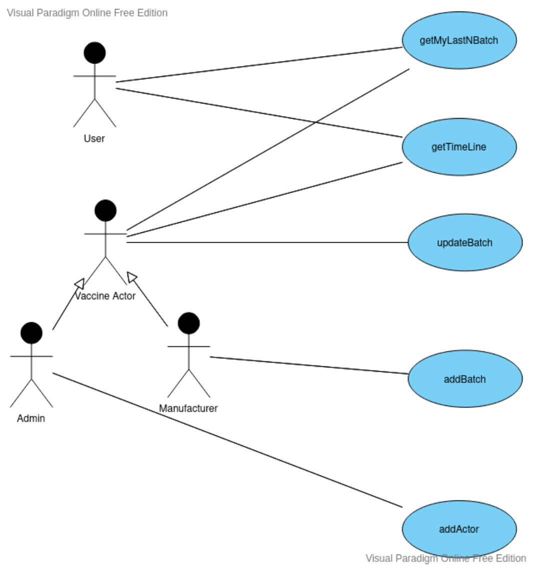


Figure 7: Use-case diagram of the DApp

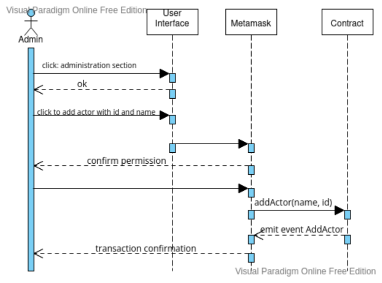

Visual Paradigm Online Free Edition
Admin
User Interface
Metamask
Contract
click: administration section
ok
click to add actor with id and name
confirm permission
addActor(name, id)
emit event AddActor
transaction confirmation
Visual Paradigm Online Free Edition

Figure 8: Sequence diagram of addActor use-case

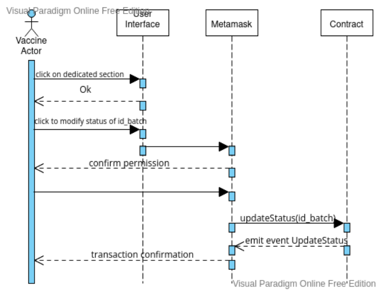


Figure 9: Sequence diagram of updateBatch use-case

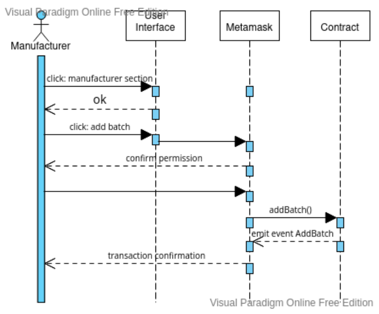


Figure 10: Sequence diagram of addBatch use-case

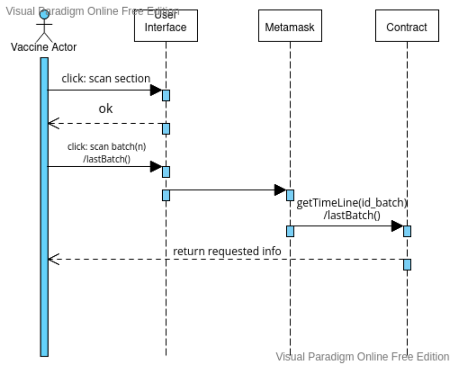

Visual Paradigm Online Free Edition
Vaccine Actor
User Interface
Metamask
Contract
click: scan section
ok
click: scan batch(n) /lastBatch()
getTimeLine(id_batch) /lastBatch()
return requested info
Visual Paradigm Online Free Edition

Figure 11: Sequence diagram of scan use-case

The last case we consider is the info requested by a general user. We put togheter in this diagram both the *getMyLastNBatch* and *getTimeline*. In this situation the functionality is public, so everyone can use it, then there is no check on the actor using it (Figure 11).

5 Implementation

We present now an example of implementation of the DApp. In Figure 11 we can see the home page. Here we have six possible sections to explore, each one with its own functionality (Figure 12):

- **Administration**: the admin panel in which the admin (i.e. the owner of the contract) can add a new actor.
- **Manufacturer Section**: section dedicated to add new batches, functionality that is only active for manufacturers.

- **Courier Section**: section dedicated to the couriers that can update the status of a batch once they move it.
- **Facilities Section**: In this section a facility worker can update the status of a batch when it is assigned to a courier.
- **Vaccine Hub Section**: in this section a vaccine hub worker can update the status of a batch as arrived to the destination hub.
- **Scan Section**: a public section in which everyone can have access and search for a specific batch.

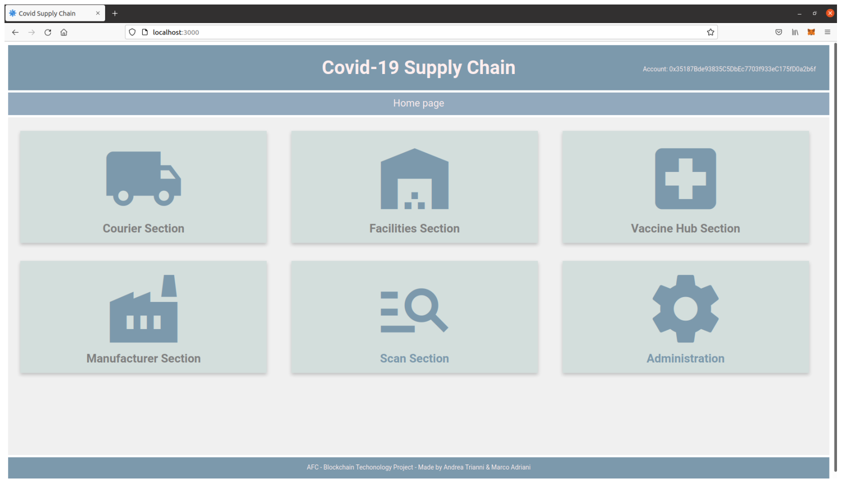
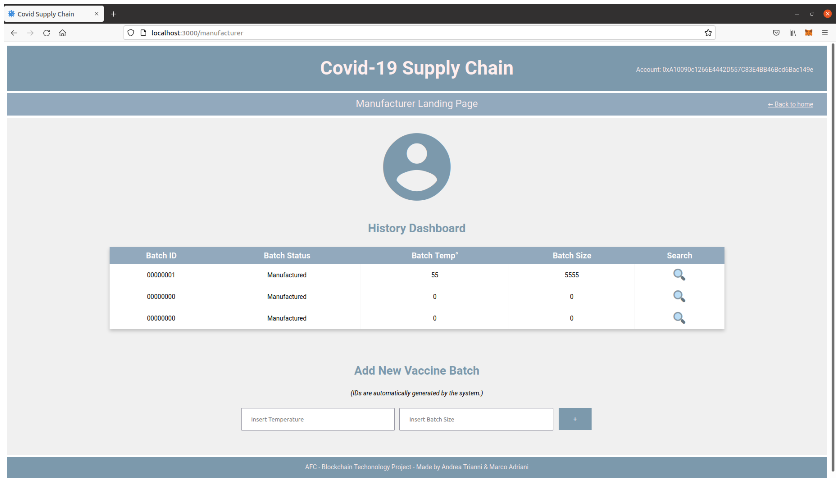


Figure 12: Home page of the DApp

Then suppose a batch is going through its life. A manufacturer add it to the blockchain, as the one with id 0000001 with a given temperature in the Manufacturer Section as shown in Figure 13. After some passages, we can see how the user, in Scan Section, can search the batch calling for example the functionality getTimeLine() and he can get in return something like Figure 14.



Covid-19 Supply Chain

Account: 0xA10090c1266E4442D557C83E4BB46Bcd6Bac149e

Manufacturer Landing Page

← Back to home

History Dashboard

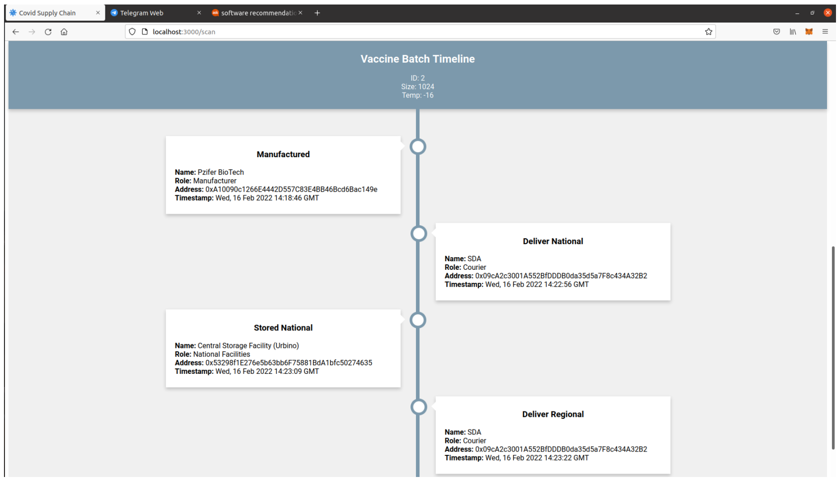
Batch ID	Batch Status	Batch Temp°	Batch Size	Search
00000001	Manufactured	55	5555	🔍
00000000	Manufactured	0	0	🔍
00000000	Manufactured	0	0	🔍

Add New Vaccine Batch

(IDs are automatically generated by the system.)

AFC - Blockchain Techonology Project - Made by Andrea Trianni & Marco Adriani

Figure 13: addBatch function example



Vaccine Batch Timeline

ID: 2
Size: 1024
Temp: -16

Manufactured

Name: Pzifer BioTech
Role: Manufacturer
Address: 0xA10090c1266E4442D557C83E4BB46Bcd6Bac149e
Timestamp: Wed, 16 Feb 2022 14:18:46 GMT

Deliver National

Name: SDA
Role: Courier
Address: 0x09cA2c3001A552BfDDDB0da35d5a7F8c434A32B2
Timestamp: Wed, 16 Feb 2022 14:22:56 GMT

Stored National

Name: Central Storage Facility (Urbino)
Role: National Facilities
Address: 0x53298f1E276e5b63bb6F75881BdA1bfc50274635
Timestamp: Wed, 16 Feb 2022 14:23:09 GMT

Deliver Regional

Name: SDA
Role: Courier
Address: 0x09cA2c3001A552BfDDDB0da35d5a7F8c434A32B2
Timestamp: Wed, 16 Feb 2022 14:23:22 GMT

Figure 14: Scan function example

6 Known issues and limitations

Our DApp is a good starting point, but it is not an arriving one. Other features can be implemented, such as generating vaccines certifications. Also we add actors just from the admin panel, and they can be added arbitrarily by the admin himself, who has the total control on who could be an actor of the system. This is a lack of security, giving all this centralized power to just one entity. Certificates for actors or another way to add them to the system can be a better solution in order to avoid possible attacks to the admin or to avoid the situation in which he can be corrupted.

7 Conclusions

"COVID-19 Supply Chain" is a DApp useful to track down the moving sequence of a COVID-19 vaccine batch. The main aim of the DApp is to guarantee a trusted way to show how vaccine batches are displaced and from whom.

In the future this DApp can be the basis, for example, of a trusted and secure system to insert and check vaccine certificates, or many other features useful to help the world to getting out at the best possible way from those years of pandemic. Even if the implementation to this work it may be necessary, we think that this could be a possible good starting point.

References

[1] "Bitcoin: A Peer-to-Peer Electronic Cash System" (2009), Satoshi Nakamoto

[2] "Ethereum: a secure decentralised generalised transaction ledger, Berlin version 6" (2022)