Article

Easing ĐApp Interaction for Non-Blockchain Users from a Conceptual Modelling Approach

Miguel A. Teruel [1,2,*] and Juan Trujillo [1,2]

[1] University of Alicante, Department of Software and Computing Systems, 03690 San Vicente del Raspeig, Spain; materuel@dlsi.ua.es, jtrujillo@dlsi.ua.es
[2] Lucentia Lab,03450 Alicante, Spain; mteruel@lucentialab.es, jtrujillo@lucentialab.es
* Correspondence: materuel@dlsi.ua.es; Tel.: +34-96-5909581

Featured Application: Easing the interaction of non-Blockchain users with a decentralized application (ĐApp) through a middleware created by using conceptual modeling in UML

Abstract: Blockchain decentralized applications (ĐApps) are applications which run on Blockchains nodes. Thus, in order to interact directly with this sort of applications, users need to have a blockchain address, wallet and knowledge about how to make transactions in order to interact with ĐApps. Therefore, the knowledge required to use a ĐApp can easily make users to desist when trying to interact with them. In order to tackle this issue, we propose a software architecture that will be located in the middle of the user and the ĐApp, thus making users initially unaware that they are interacting with a ĐApp. This is achieved by analyzing the relationship between ĐApps and Apps by using UML modelling. Next, based in the previous analysis, we created a middleware for users to interact with a ĐApp in the same manner the do with a traditional web app, i.e. by using usernames, passwords and UI elements instead of addresses, private keys or transactions. Finally, in order to put the developed middleware into practice, we developed a ĐApp that makes use of it. This ĐApp registers the time control of workers from companies by using Blockchain to store the data in a secure and non-modifiable manner.

Keywords: Blockchain; ĐApp; UML; conceptual modelling; Ethereum; Smart Contract; Solidity; Quorum; middleware; Clockchain

1. Introduction

Back in 2008, a paper written by somebody under the pseudonym of Satoshi Nakamoto [1] revolutionized both the world economic system and the distributed management of data. Bitcoin was born as a distributed ledger that would enable the online payments without the intervention of a third party (i.e. bank, financial services, payment gateway, etc.). Indeed, instead of trusting a third party as the transactions' validator, this process is conducted by a network of peers, using cryptographic techniques. Needless to say, Bitcoin has been adapted widely, having a market capitalization of 160 trillion $ in May 2020.

Nevertheless, not only were cryptocurrencies [2] born but also the mechanism used to store the transfers of Bitcoins, namely the Blockchain [3]. A Blockchain is a data structure whose information is stored into blocks. Hence, each block is hashed and connected with the previous block. Therefore, in order to alter any block of the chain, the whole chain would have to be rehashed, fact that will require more than half the computing power of all the network's peers, which is known as a 51% attack [4,5]. Consequently, Blockchain arises as an immutable data structure to anonymize and secure data in a distributed manner, regardless of its application domain. As a matter of fact, Blockchain has already been applied to different domains such as healthcare [6] or education [7].

After the Bitcoin revolution, the next significant milestone arrived with Ethereum [8] and the smart contracts [9]. Hence, the Ethereum Blockchain not only stored transactions of its cryptocurrency, namely Ether. Indeed, this Blockchain can execute the code written in its smart contract, enabling us to manage users' digital assets (i.e. tokens) according to the rules stipulated in such contracts. Thanks to the capability of executing smart contracts, Ethereum enabled us to create decentralized applications that run in the Blockchain (ĐApps) [10,11] as well as decentralized autonomous organizations (DAO) [12], which are virtual organizations able to deal with digital assets without human intervention thanks to a set of rules codified in smart contracts.

We are talking about a world-wide revolution in the economic and distributed data storage paradigms that would enormously benefit user. However, such users are not entirely ready yet due to Blockchain's learning curve. Indeed, in order to interact with a ĐApp, users must have a Wallet per Blockchain, as well as the knowledge about how to use their public and private keys to sign and deploy transacts to the Blockchain that will interact with the smart contracts. Because of that reason, a ĐApp is prone to fail nowadays as far as the general public is concerned.

In order to tackle this issue, we propose a middleware which is located within the smart contracts and the user interface of the ĐApp, that will ease the interaction with it by making it similar to the classic user web interaction based on usernames, password and visual web controls (e.g. button, forms, and so on). In this sense, users will be able to use a ĐApp in a comfortable manner until they get used to directly interact with smart contracts. Moreover, in order to develop such middleware, we rely on the conceptual modeling analysis of the Ethereum Blockchain architecture.

This paper is structured as follows. After this introduction, Section 2 presents different works related to Blockchain modeling and ĐApps. Next, Section 3 widens the explanation of why users have trouble when interacting with ĐApps. Section 4 will explain the relationship between ĐApps and Apps from an UML modelling point of view. Section 5 will present out middleware proposal which will be put into practice in Section 6 by means of a running example. Finally, Section 7 will finish with our conclusions and future work.

2. Related Work

Despite being quite a recent field of research, we are able to find several works related to the development of ĐApps. For instance, the authors of [13] presented a ĐApp for sharing everyday objects. By using this ĐApp, which was deployed on the Ethereum Blockchain, users can register and rent objects by scanning QR codes. Besides, a different scenario was considered in [14], whose authors presented a ĐApp for real-time ride-sharing services. By means of this ĐApp, they propose a Blockchain version of the Uber platform, consisting in "decentralized, community-owned-and-managed transportation network without unsatisfactory centralized decision-making or risks". In a different vein, a ĐApp for the healthcare domain is presented in [15]. In this work, they presented FHIRChain, a ĐApp to store and share clinical data, thus benefiting from Blockchain's security. Therefore, they use digital health identities (Blockchain addresses) to authenticate patients in this ĐApp, thus maintaining data ownership and privacy.

As far as conceptual modeling and software engineering for Blockchain is concerned, we are able to find only a handful of works. Indeed, Rocha and Ducasse started the discussion regarding the need for applying software engineering to de development of smart contracts in [16]. They propose the usage of UML's class diagrams in order to model the structure of smart contracts since they are class-based artifacts. In this sense, the authors of [17] applied several UML diagrams to model an Ethereum-based ĐApp for managing workers contracts. In this work, they applied the Blockchain Oriented Software Engineering approach [18], which in turn, acknowledge the need for applying software engineering to Blockchain-based applications. Despite not been directly related to UML, in [19] it is proposed a fine-grained approach to identify which are the elements of an application architecture that could be implemented by using Blockchain. They illustrate such approach by means of a case study based on a system for the coordination and payment of craftsmen constructing buildings.

As far as we known, we cannot find any work aimed at easing the interaction of non-Blockchain expert users with a ĐApp. Therefore, this constitutes the main motivation for this work: to conceptually model and analyze the ĐApps structure, with the aim of creating a middleware. It would help users to interact with ĐApps without the need of being used to Blockchain concepts which could be cumbersome and overwhelming for non-expert users.

3. Interacting with ĐApps

When dealing with a ĐApp for the first time, it is quite usual that a user does not have the faintest idea about how to do it. That is because ĐApp interaction is based on making calls and transactions to smart contracts' methods. Such transactions, which typically involve the exchange of tokens, have to be made from the user's wallet, with is univocally identified along the Blockchain by means of its public addresses. Hence, in order to perform a transaction, the user would have to expend some tokens to pay for the transaction to be executed (known as gas on the Ethereum network). Afterwards, the user would have to sign the transaction by using his / her private key and to relay it through a Blockchain node for the transaction to be mined and executed. Finally, the user will have to follow the transaction execution through the transactions TX receipt, which can or cannot be successfully executed depending on several aspects such as the smart contract rules or the shortage of funds added to the transaction in order to pay for it.

Besides, the user's wallet is typically installed on just one device. Because of that reason, if the user needed to interact with the ĐApp by using a different device, the wallet will have to be restored by means of the private keys or 12-words mnemonic [20], a process most users are unaware of. Furthermore, for the time being, there are little support for mobile wallets supporting ĐApp transactions. Therefore, using a ĐApp with a mobile device could not be always possible.

Due to this level of complexity when executing a single Blockchain transaction, it is typical that a non-Blockchain user would not be able to interact with a ĐApp without previous training. Moreover, provided a user was able to finally interact with a ĐApp in a Blockchain does not means that he / she will be able to interact with another ĐApp deployed in a different Blockchain. That is because each Blockchain requires its own wallet and has unique interaction mechanisms. As an example, making transaction to a smart contract deployed on Ethereum [8] will significatively differ from a similar contract deployed on Ripple [21], Corda [22], HyperLedger [23] or the upcoming Libra [24] due to the differences among such Blockchains [25]. Therefore, users will have to learn the specifics of each Blockchain in order to interact with different ĐApps.

Because of the aforementioned reasons, if we need our ĐApps to be used by every user regardless of their Blockchain expertise, a mechanism that eases ĐApp interaction is necessary. For that reason, Section 4 will analyze the conceptual model of a Blockchain, and the Section 5 will present our proposed middleware to simplify the interaction with a ĐApp, thus partially camouflaging it into a common web app for the users to interact with no prior Blockchain knowledge.

4. From ĐApp to App: an UML Modeling Analysis

In order to identify how to ease the interaction with ĐApps, we opted for a Model Driven Development approach. Therefore, we will identify the relationship among ĐApps and Apps elements by modeling an Ethereum-based Blockchain architecture and a common App. Figure 1 shows such modeling. Hence, at the top of the figure we can see the UML class architecture of a ĐApp interacting with a Blockchain, as suggested in [16]. Besides, at the bottom it can be seen the equivalent App architecture where their classes have been related to each Blockchain element they will have to access in order to expose the Blockchain functionality to a non-expert user in an App. For the sake of the model comprehension, the relationship among ĐApp and App elements will be presented in the following through the model entities:

- *User*: The proposed App model features a collection of users (*UsersDB*). In this sense, The *User* entity will be associated with the Blockchain Address for the user to be able to interact with the Blockchain without needing to be aware of its address' public and private keys. Besides, the *User* entity will also be associated to the Blockchain tokens' *Balances*. Therefore, it will represent the number of tokens the user will have. Note that for the sake of the model understandability, we will consider only ERC20 (fungible) tokens [26]. Nevertheless, the model could be easy adapted to work with ERC721 [27] (non-fungible) tokens.

- *Login*: Because of we are modeling an App, it is common to have a login functionality. Therefore, our model will have an *Action* called *Login*. Hence, when a user logs in with a correct email / password combination, the Wallet containing its *Address* will be unlocked, thus enabling such user to make *Transactions* towards the Blockchain.
- *Recovery*: One of the issues of Blockchain it that if the user loses the private key of an address, it will never be able to access it since Blockchain does not provide any private key recovery mechanism. It is worth noting that users are used to count with password recovery systems in almost every App. Therefore, this functionality will be enclosed within the *Recovery* entity. Therefore, if requested, this entity can access the *UserDB* in order to provide password restoration mechanisms. Note that what will be recovered will be the User's password, not the private key. That is because, in order to facilitate the interaction, we would provide a non-custodial system, where the user is not directly responsible for its private key or wallet, being those custodied by the App database.
- *Movement*: This entity will represent the transfer of tokens among users in a straightforward manner. In that sense, if a user wanted to send funds to a different App user, the only required data would be the number of tokens to be transferred and the recipient's email. Therefore, the recipient address will never be required, thus making it more difficult to miss the recipient when making token transferences.
- *Interaction*: The emission of payable Blockchain *Transactions* (invocation to payable Methods) will be translated into *Interactions*. It is worth noting that we are referring to operations that alter the current Blockchain status. Therefore, such transactions require gas (they need to be paid for by using Ether in the case of Ethereum) in order for the miner to confirm the validity of such transaction. Nevertheless, in order to make the model easier to understand, we will consider a private Ethereum Blockchain whose transactions, despite modifying the Blockchain states, require no gas.
- *Query*: If an *Interaction* represented a payable *Method* call, *Query* will correspond to a view one (a read-only method). Therefor, View methods are methods that, despite reading data from the Blockchain, do no change its status. Therefore, since no modifications are involved, they do not need to be mined, not requiring any gas to be processed. Hence, we will represent all the Blockchain data requests into *Queries* that will invoke view methods in order to retrieve information (e.g. token balances, smart contact status, etc.).
- *App*: Needles to say that a *ĐApp* interact with one or more smart contract. Therefore, our *App* entity will also have an association with the smart contracts it has access to, Therefore, our users will never have to be aware of the addresses of the smart contracts they are interacting with as well as the address of smart contacts that define the tokens they treasure.

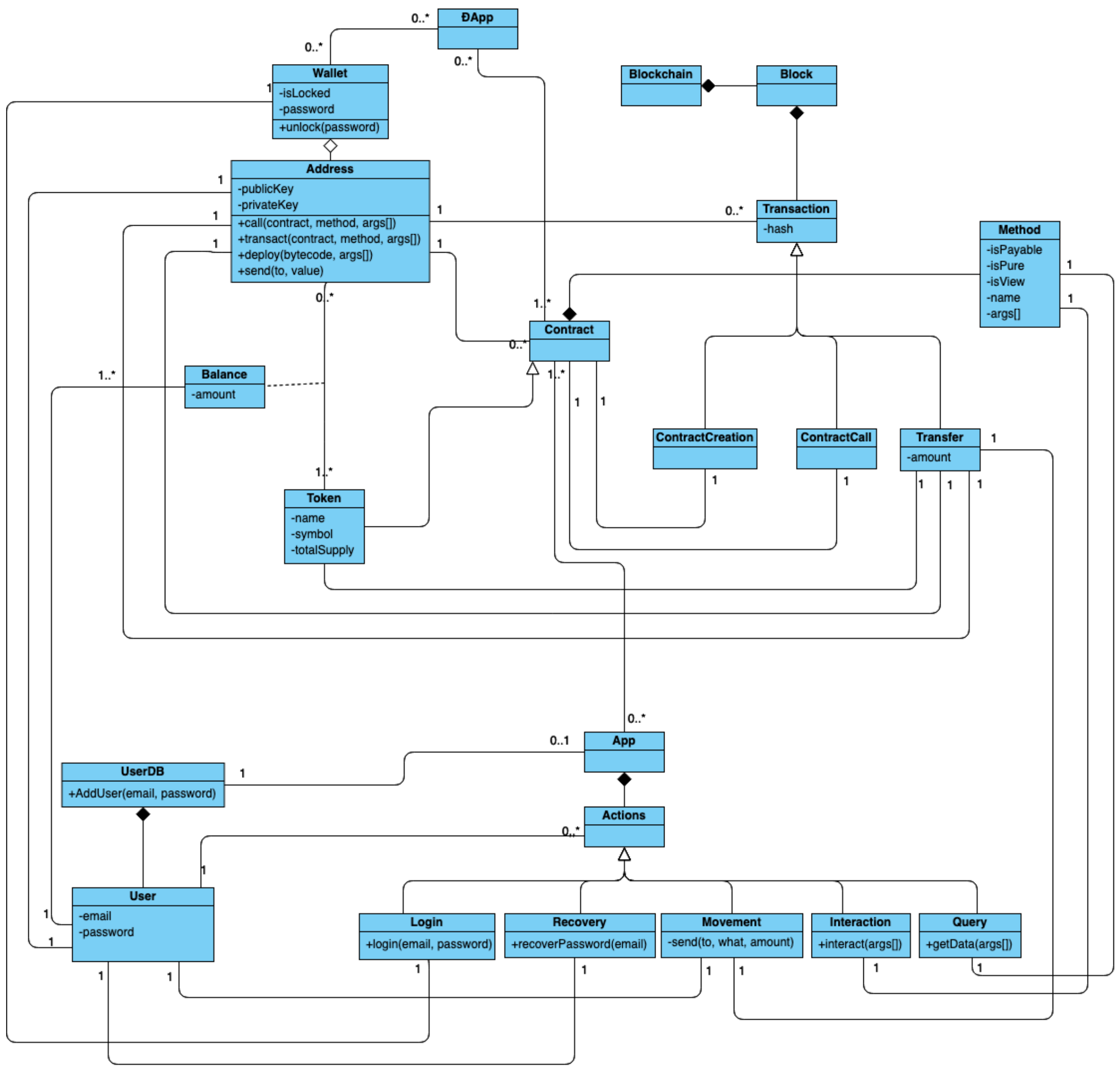


Figure 1. Relationship among ÐApps and App elements

Once the model elements have been explained, we will present in the next section how to implement it. Therefore, the relationship between models will be implemented by using a middleware that will transform user interaction thought an App to Blockchain calls and transactions.

5. Middleware to Ease ÐApp User Interaction

With the aim of tackling the ÐApp interaction problem explained in Section 错误!未找到引用源。 we propose a technique based on a middleware that will "translate" the Blockchain's mechanisms into concepts users are familiar with. In other word, the middleware will implement the required functionality to transform the ÐApp model shown on Figure 1 to the App-equivalent one. From now on, we will exemplify the proposed middleware by using Ethereum since it currently is the most used Blockchain to deploy smart contracts.

In a common Ethereum ÐApp, the software architecture is distributed between a JavaScript-based web application featuring the Web3 library and a wallet [28] as frontend and a set of Ethereum's smart contracts as backend (top of Figure 2). This architecture, for starters, make users unable to interact if the they do not have an Ethereum wallet (e.g. Metamask [29]) embedded into their browsers. Furthermore, provided the wallet is running and properly installed, they will be able to log in and interact with the ÐApp by signing transactions with their private keys. What is more, for those transactions involving the modification of the Blockchain, the user will have to put some gas into the transaction (i.e. pay for it) in order for it to be successfully committed into the Blockchain (Figure 3). Finally, users will also have to take another decision before releasing the transaction: deciding how much gas they are willing to put on in. Hence, putting too much gas will result into an unnecessarily expensive transaction. Opposingly, putting little could make the transaction unsuccessful, since no miner will mine the transaction into a block due to the low reward to be received. This process gets even more complicated if the user deal with a private Ethereum-based Blockchain, such as Quorum, since the transaction will have to be manually adjusted to use no gas.

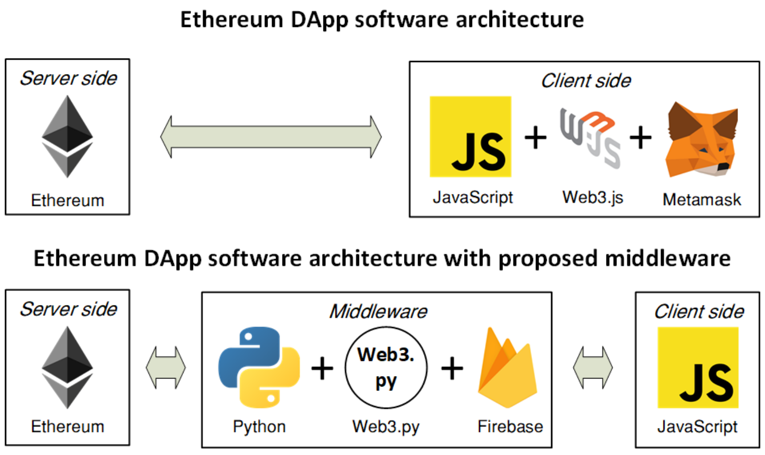
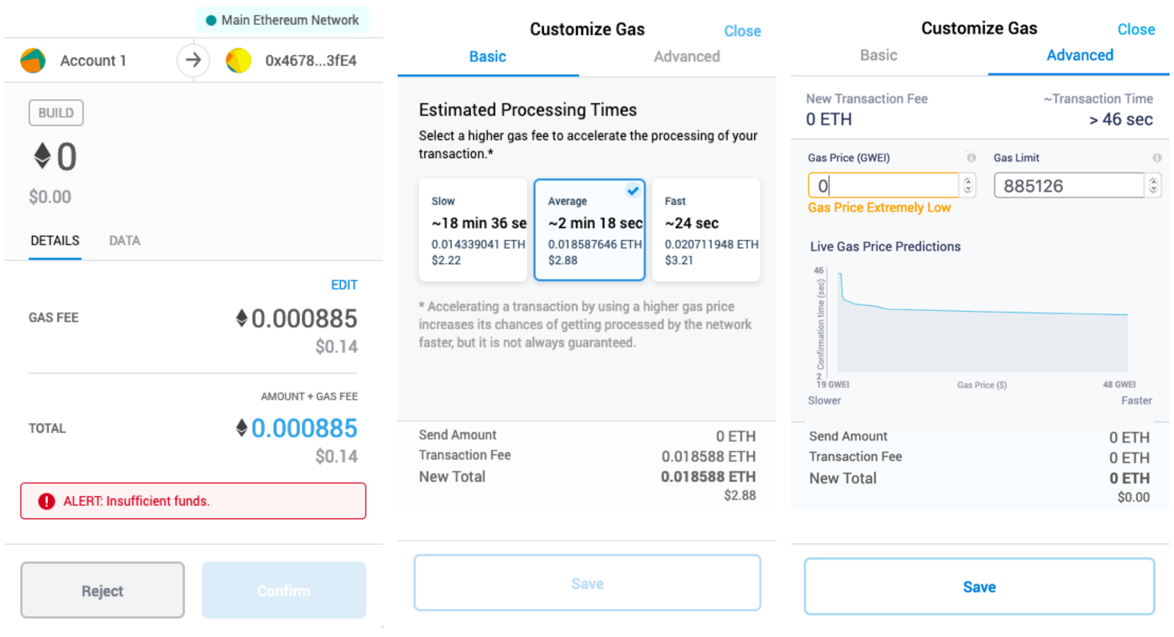


Figure 2. Ethereum software architecture comparison


Main Ethereum Network
Account 1 0x4678...3fE4
BUILD
0
$0.00
DETAILS DATA
EDIT
GAS FEE 0.000885
$0.14
AMOUNT + GAS FEE
TOTAL 0.000885
$0.14
ALERT: Insufficient funds.
Reject Confirm

Customize Gas Close
Basic Advanced
Estimated Processing Times
Select a higher gas fee to accelerate the processing of your transaction.*

Slow	Average	Fast
~18 min 36 se	~2 min 18 sec	~24 sec
0.014339041 ETH	0.018587646 ETH	0.020711948 ETH
$2.22	$2.88	$3.21

* Accelerating a transaction by using a higher gas price increases its chances of getting processed by the network faster, but it is not always guaranteed.

Send Amount	0 ETH
Transaction Fee	0.018588 ETH
New Total	0.018588 ETH
	$2.88

Save

Customize Gas Close
Basic Advanced
New Transaction Fee 0 ETH
~Transaction Time > 46 sec
Gas Price (GWEI) 0
Gas Limit 885126
Gas Price Extremely Low
Live Gas Price Predictions

Send Amount	0 ETH
Transaction Fee	0 ETH
New Total	0 ETH
	$0.00

Save

(a) **(b)** **(c)**

Figure 3. Sending a transaction to an Ethereum Smart Contract (**a**) and deciding how much gas to spend (**b**) and customizing it for a private Blockchain (**c**)

Putting everything together lead us to a straightforward conclusion: a non-expert user interaction with a ÐApp will probable fail due to the considerable number of concepts he / she needs to know. Therefore, we propose using a middleware (bottom of Figure 2) that will relieve the user of the need of mastering Blockchain concepts and technologies when interacting with a ÐApp for the first time. In order to demonstrate our proposal, we will use a Python implementation and a Firebase database. Nevertheless, any technology could be used provided it has the required libraries (e.g.

Web3 for the Ethereum) to interact with the Blockchain. In the following, the different components of our proposed middleware will be shown.

5.1. Login into the ĐApp

In a normal ĐApp, a user logs in by using its Blockchain public address, typically by linking it to the ĐApp through the wallet's address. Needless to say, that in the event the user has no wallet, this operation will not succeed. In order to tackle this first issue, we propose the usage of a mechanism every user is familiar with: emails and passwords. Therefore, in order to use this mechanism, we need to establish a correspondence between users' emails and their public addresses. With this aim, we used a Firebase database.

Thus, when a user signs up into the ĐApp, a register form will be shown requiring the email and password. In this very moment, our middleware will create an Ethereum wallet by using the Web3 package *web3-eth-personal*. Hence, such wallet will be encrypted by using the provided password. At this point the user will receive a confirmation email with their public and private address since he / she could also interact directly with the smart contract as soon as he / she mastered Ethereum procedures. In parallel, we will store the user email, public address and a hash of the password. This will enable the user to log in in a traditional manner without having to be aware of the Blockchain keys.

5.2. Sending Transactions to the Smart Contract

Once the problematic to logging in has been solved, the devious transaction mechanism will be eased. To do that, we will extremely simplify the transaction bust just making them transparent to the user. Hence, sending a transaction will be as easy as clicking a button in the user interface.

This procedure is possible by encapsulating the transaction invocation into a RESTful API POST call that will receive the parameters required by the transaction. Then, it will invoke the send transaction method from the Web3 *web-eth* package. Nevertheless, if we are dealing with Ethereum main network, gas is required to relay the transaction. For that reason, the middleware developer could provide the users' wallet with gas to run transactions and refuel it in the event it was low enough. Besides, the Ethereum calls (i.e. queries to the Blockchain which do not require gas since they are read-only) are encapsulated into RESTful GET calls accordingly. It is worth noting that this mechanism will also benefit the frontend developer, who will be able to develops the web interface without having to interact with Web3 directly.

5.3. Enabling Users to Change their Passwords

As said before, the authentication mechanisms of a Blockchain is the public and private key pair. This fact makes it impossible to change the private key of an address. Moreover, if the user wanted a different identification id, (i.e. a new address), the funds on the old wallet will have to be manually transferred to the new one. This action, that will have a cost in gas, will generate a new private key. Moreover, all the data in the smart contract related to the old address will remain assigned to it. Hence, this data could only be migrated to the new address if the smart contract implemented a transfer data mechanism, which will probably result expensive as far as gas is concerned.

However, thanks to our middleware, a user does not have to deal with this problem since an email can always be changed in the Firebase database, thus generating a new login id that is independent of the address and, consequently, independent of its funds and stored data.

The problem comes when dealing with password changes. The users' wallets are physically located in the Ethereum node when using our middleware, and encrypted by using the users' passwords. Nevertheless, the Web3 library does not provide the functionality to re-encrypt such wallets by using a new password. Because of that reason, we propose the direct interaction with Geth, the command line interface for Ethereum nodes implemented in Go [30]. Thus, then a user wants is password to be changed, he / she will ask the ĐApp for it, thus receiving an email with a link which will enable the user to do so. Therefore, when the user provides the new password, the ĐApp will invoke a middleware RESTful method which, in turn, will connect to Geth via SSH and change the password accordingly. Finally, the new password will be re-hashed and stored in the database to enable the user to log in again by using the new credentials.

5.4. Tokens Management

Needless to say, one of the main benefits of Blockchain is the tokenization of assets, thus providing value to physical or virtual entities. Consequently, our middleware must provide a double functionality. On the one hand, since the users are not required to control their wallets (despite they can do so by importing their private keys into a normal Ethereum wallet) they require a mechanism to transfer tokens among users. On the other hand, as the ĐApp can provide the users with tokens under certain conditions (DAO), its manager, who is not required to be Blockchain-savvy, need to have a way to create new tokens (a.k.a. minting tokens).

In this sense, the middleware provides functionality to both users and managers to query token balances, transfer tokens among users and mint tokens (only for managers). However, this operation will benefit from our middleware due to the email-address translation mechanism. Hence, a user could transfer tokens to a different one by only knowing his / her email. Besides, they will also be able to transfer tokens to the contract in order to exchange them for whatever asset the manager decide, without knowing the contract address. In this sense, a user will never send funds to the wrong contract or user.

5.5. Security Concerns

One of the key features of Blockchain is its security. Therefore, our ĐApp will have to keep up this by implementing several security procedures. Hence, we propose the usage of JWT [31] as credentials for the users as an extra layer over the Blockchain security.

When a user logs is successfully, a JWT will be created by the middleware and sent to the frontend for the user interface to be able to invoke the RESTful API. Moreover, the user's wallet will be unlocked for a number of seconds equal to the JWT expiration time. Hence, in a user logs out, the JWT will be blacklisted and the wallet locked. Nevertheless, if a user does not log out, the JWT will automatically expire and the wallet will be locked, thus requiring a new log in to keep on using the ĐApp.

Besides, we propose implementing an additional security layer regarding IP addresses whitelisting. With this aim, our Ethereum node will only be accessible to users whose IP are whitelisted. Therefore, if an unauthorized user tries to unlock a wallet without signing in properly, the node will automatically reject the operation. For that reason, when a user log in, his / her IP will be whitelisted. Finally, the log out of a user will proceed to remove the IP address from the whitelist. It is worth noting that using this procedure will disable users so interact directly with the Blockchain through our node. However, this issue will be only applicable to private Ethereum-based Blockchains where only a few (or even one) node are allowed to access the Blockchain.

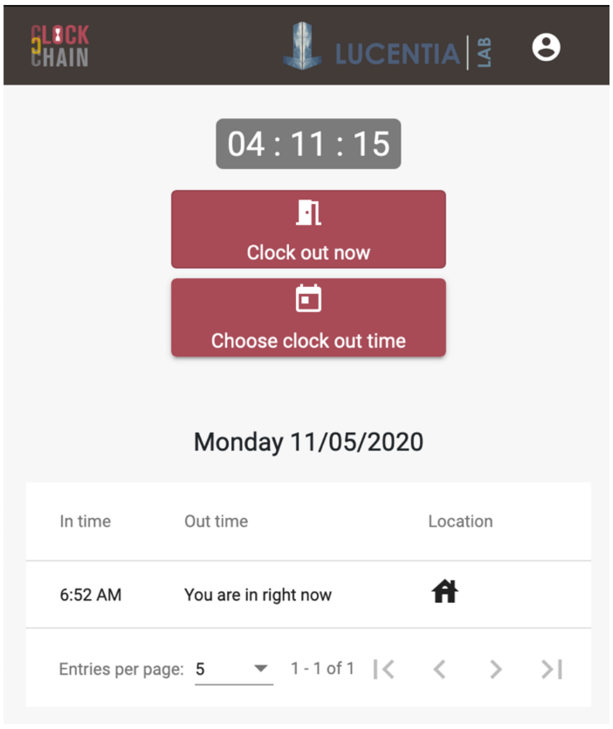
6. Running Example: Employee Schedule Tracker ĐApp over Quorum Blockchain

Once the proposed middleware has been explained, this section will put it into practice. To do so, we developed a ĐApp aiming at tracking employee schedule over an Ethereum-based Blockchain, namely Quorum. More specifically we deployed our smart contracts over the Quorum network of Alastria [32], a Spanish national Blockchain consortium consisting of the main companies in Spain (banks, telecommunication, public entities, transportation, IT, and so on). This semipublic permissioned Blockchain will enable us to run our ĐApp in a secure environment as the same time we maintain the application expenses constant since no gas is required to run transactions on the Blockchain.

Regarding the developed ĐApp, it is named Clockchain [33] after its functionality and underlying technology. This ĐApp arises due to a new Spanish law, the Royal Decree-Law 8/2019, of March 8th, on urgent social protection measures and the fight against job insecurity in the workday.

This law states that the workers' daily entries and exits must be safely stored during a minimum of 4 years in a secure and non-modifiable storage medium. Indeed, the security, linger and non-modifiability requirements for the data to be stored make this a perfect scenario to apply Blockchain. Nevertheless, the main issue there is that users are not used at all to interact with a ĐApp. Therefore, we applied the proposed middleware to this scenario in order to make Clockchain really easy to be used for anyone.

Figure 4 shows the main interface of Clockchain after logging in. It is worth noting that, for performing such log in, neither users need a wallet, neither they have to be aware they are working with a Blockchain-based system. On the contrary, they only use a combination of email / password to log in. Once the users are logged in, they will see all their current entries and exits during the current day, as well as the elapsed working time (Figure 4). Such information is retrieved from the Blockchain in a transparent manner, thus corresponding to a *Query Action* (see Section 4) of the middleware. Indeed, what these actions will perform in the middleware is a call to a view function inside the smart contract. **Figure 4.** Clockchain interface for a user when clocking out

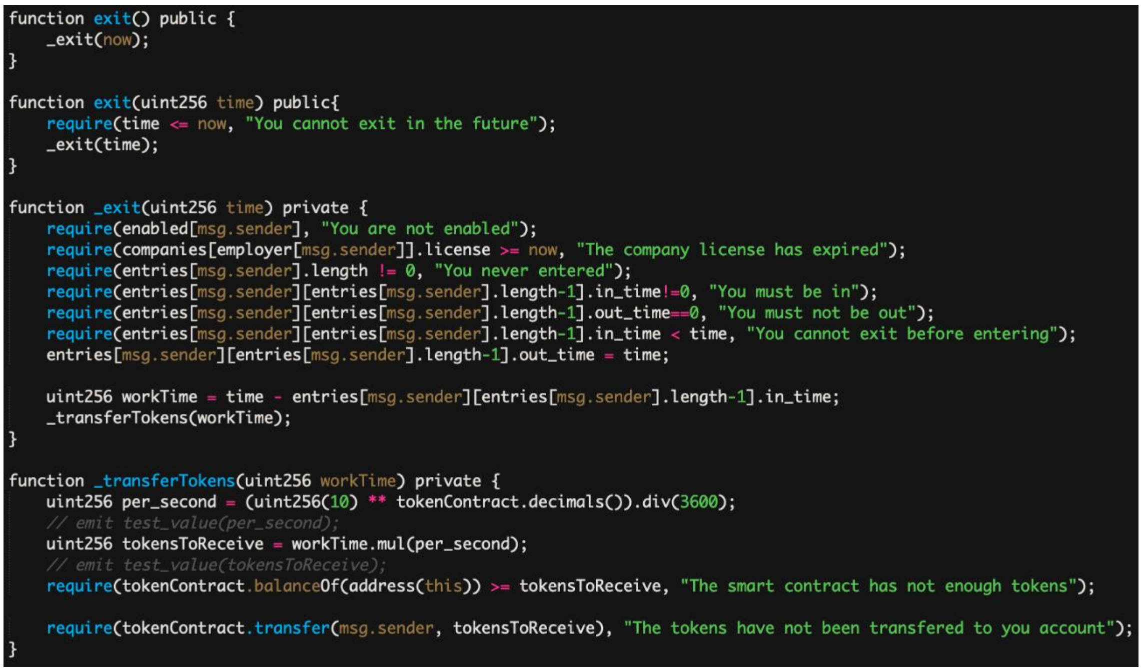


In time	Out time	Location
6:52 AM	You are in right now	🏠

Entries per page: 5 1 - 1 of 1

Figure 4. Clockchain interface for a user when clocking out

Besides, Figure 4 shows the two *Interaction Actions* the user can perform at that moment. More specifically, we can see the interface of a worker who has already clocked in around four hours ago from home. Therefore, the interactions with the Blockchain that could be performed at that moment are to clock out at that very moment, or to choose a different clock out time (in case the workers forgot to clock out at a time in the past). As an example, Figure 5 shows the corresponding smart contract methods such interactions will correspond to. For this example, the button *Clock out now* will correspond to the public function *exit().* Besides, the button *Choose clock out time* will correspond in turn to the function *exit(uint256 time)* since the user could choose the clocking out time. As it can be seen in Figure 5, both public functions will call the private *_exit(uint256 time)* function. This function, after checking the requirements for the Blockchain transaction to be performed, will store the clocking out time of the worker represented by its Blockchain public address (*msg.sender*). Moreover, our smart contract will also reward the worker's time with tokens, that will be transferred to its address by means of the *_transferTokens(uint256 workTime)* function. Indeed, each worker will receive one ERC20 token for each worked hour. Hence, the user will be able to query and transfer the received tokens by using the ĐApp's web interface, not requiring any external wallet such as Metamask.



```
function exit() public {
    _exit(now);
}

function exit(uint256 time) public{
    require(time <= now, "You cannot exit in the future");
    _exit(time);
}

function _exit(uint256 time) private {
    require(enabled[msg.sender], "You are not enabled");
    require(companies[employer[msg.sender]].license >= now, "The company license has expired");
    require(entries[msg.sender].length != 0, "You never entered");
    require(entries[msg.sender][entries[msg.sender].length-1].in_time!=0, "You must be in");
    require(entries[msg.sender][entries[msg.sender].length-1].out_time==0, "You must not be out");
    require(entries[msg.sender][entries[msg.sender].length-1].in_time < time, "You cannot exit before entering");
    entries[msg.sender][entries[msg.sender].length-1].out_time = time;

    uint256 workTime = time - entries[msg.sender][entries[msg.sender].length-1].in_time;
    _transferTokens(workTime);
}

function _transferTokens(uint256 workTime) private {
    uint256 per_second = (uint256(10) ** tokenContract.decimals()).div(3600);
    // emit test_value(per_second);
    uint256 tokensToReceive = workTime.mul(per_second);
    // emit test_value(tokensToReceive);
    require(tokenContract.balanceOf(address(this)) >= tokensToReceive, "The smart contract has not enough tokens");

    require(tokenContract.transfer(msg.sender, tokensToReceive), "The tokens have not been transfered to you account");
}
```

Figure 5. Smart contact methods for clocking out

Finally, in order to provide our ÐApp of the required scalability to deal with an unpredictable number of users, it was decided to deploy it in the Google Cloud Platform (GCP) as shown in Figure 6. In that sense, we will create the Blockchain node by using a GCP Compute Engine instance. In the event the node got overloaded, we will be able to scale it in order to improve its power. This can happen due to either a vast number of users using our ÐApp concurrently or a growth of the Blockchain. Therefore, we could increase the computational power or the disk size accordingly.

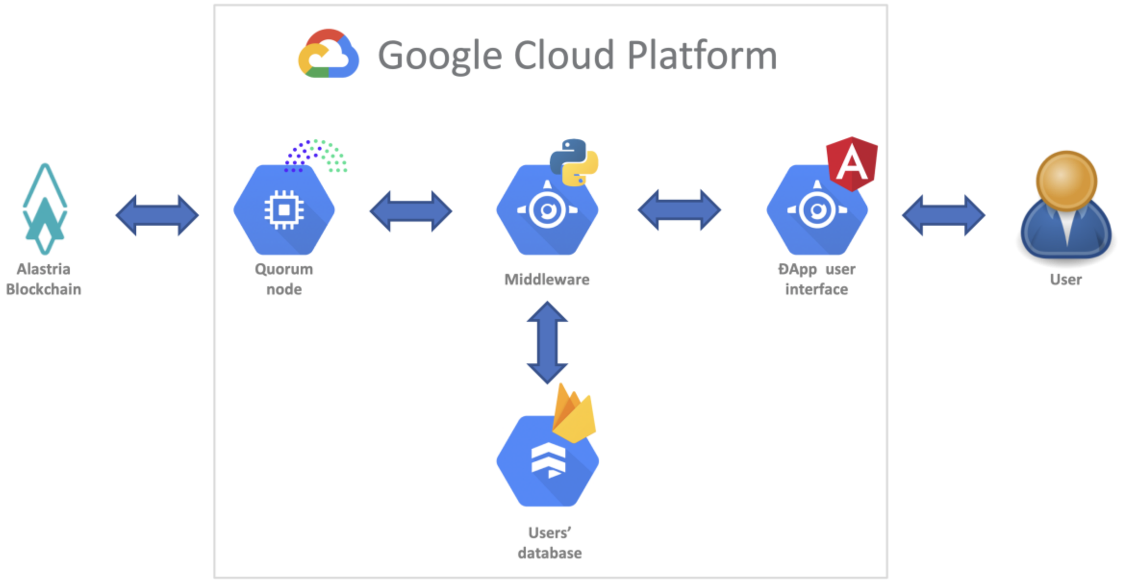


Figure 6. Clockchain ÐApp cloud architecture

Besides, the middleware and the frontend are deployed as GCP AppEngine instances. In this case, this will provide us with total flexibility regarding the number of simultaneous users. Hence, more AppEngine instances will be created dynamically to deal with the users' demand. Opposingly, if the ÐApp is not been used, all AppEngine will be shut down, thus reducing the ÐApp computation expenses to the cost of just maintaining the node active. Finally, the database will act in the same manner, thus generating expenses only when it is accessed.

7. Conclusions and Future Work

Blockchain supposes a new paradigm that not only has it revolutionized the world digital economic system but also the foundations of distributed data management. Thus, beyond the cryptocurrencies such as Bitcoin, Blockchain provides us with the required tools to create smart contracts that will run in a distributed environment. Thanks to these smart contracts, we are able to develop Blockchain distributed applications, or ĐApps. These are applications that will run in a distributed manner on the Blockchain by obeying the rules codified on the smart contracts.

The problem with a relatively new-flanged technology such as Blockchain is that users need to be aware of many concepts before interacting with an application of this sort (e.g. addresses, private keys, wallets, gas, gas price, transactions, signatures, etc.). Therefore, it is very common that a user confronting a ĐApp for the first time fails, due to the pronounced learning curve required to interact with Blockchains.

In order to tackle this issue, we analyzed and related the elements of both ĐApps and Apps by using conceptual modeling. Hence, this analysis leads us to the identification of how a user would be able to interact with a Blockchain through an App in an easy manner, thus not requiring any knowledge of Blockchain concepts.

After this analysis, we proposed the usage of a middleware that will ease the interaction with a ĐApp by making users feel like they were using a typical web application. Indeed, instead of using wallets, public addresses, private keys and signed transactions, they with interact by using concepts they are familiar with, namely emails, passwords and web interfaces. In other words, this middleware will make the users able to interact with a ĐApp without any Blockchain knowledge. Therefore, our middleware consists of five approaches that will facilitate the interaction with a ĐApp: enabling login with email and passwords, sending transactions, decoupling users' credentials from Blockchain data, tokens management and security concerns.

In order to demonstrate how to put the middleware into practice, we presented a ĐApp that tracks employees' schedules by using an Ethereum-based permissioned Blockchain. Hence, this ĐApp implements all the approaches of the proposed middleware, thus easing the user interaction without losing the security and reliability of Blockchains.

As a future work, we will work towards the model-based automatization of the middleware generation. In this sense, an initial version of the middleware would be automatically generated by taking the smart contracts behind it as an input. Finally, a scaffolding of the ĐApp could also be generated, thus proving web developers with a basic functionality to start building the ĐApp from.

Regarding Clockchain, it will be analyzed how to provide automatic scalability features to the node without shutting it down. Hence, by providing the node (or nodes) with the required scalability, we will achieve to have a fully scalable environment able to grow depending on the number of concurrent users.

Author Contributions: Conceptualization, methodology, investigation, software, writing – original draft and writing – review& editing, M.T.; supervision and fund acquisition, J.T.

Funding: This work has been funded by the ECLIPSE project (RTI2018-094283-B-C32) from the Spanish Ministry of Science, Innovation and Universities.

Acknowledgments: We would like to thank the development team of Lucentia Lab for developing the web interface of Clockchain.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System 2008.
2. Luther, W.J. Cryptocurrencies, network effects, and switching costs. Contemp. Econ. Policy 2016, 34, 553–571, doi:10.1111/coep.12151.

3. Swan, M. Blockchain: Blueprint for a new economy; 1st ed.; O'Reilly Media, Inc., 2015;
4. Bradbury, D. The problem with Bitcoin. Comput. Fraud Secur. 2013, 2013, 5–8, doi:10.1016/S1361-3723(13)70101-5.
5. Sayeed, S.; Marco-Gisbert, H. Assessing Blockchain Consensus and Security Mechanisms against the 51% Attack. Appl. Sci. 2019, 9, 1788, doi:10.3390/app9091788.
6. Khezr, S.; Moniruzzaman, M.; Yassine, A.; Benlamri, R. Blockchain Technology in Healthcare: A Comprehensive Review and Directions for Future Research. Appl. Sci. 2019, 9, 1736, doi:10.3390/app9091736.
7. Alammary, A.; Alhazmi, S.; Almasri, M.; Gillani, S. Blockchain-Based Applications in Education: A Systematic Review. Appl. Sci. 2019, 9, 2400, doi:10.3390/app9122400.
8. Wood, G. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Proj. yellow Pap. 2014, 151, 1–32.
9. Christidis, K.; Devetsikiotis, M. Blockchains and Smart Contracts for the Internet of Things. IEEE Access 2016, 4, 2292–2303, doi:10.1109/ACCESS.2016.2566339.
10. Cai, W.; Wang, Z.; Ernst, J.B.; Hong, Z.; Feng, C.; Leung, V.C.M. Decentralized Applications: The Blockchain-Empowered Software System. IEEE Access 2018, 6, 53019–53033, doi:10.1109/ACCESS.2018.2870644.
11. Raval, S. Decentralized applications: harnessing Bitcoin's blockchain technology; " O'Reilly Media, Inc.," 2016;
12. Mehar, M.I.; Shier, C.L.; Giambattista, A.; Gong, E.; Fletcher, G.; Sanayhie, R.; Kim, H.M.; Laskowski, M. Understanding a Revolutionary and Flawed Grand Experiment in Blockchain. J. Cases Inf. Technol. 2019, 21, 19–32, doi:10.4018/JCIT.2019010102.
13. Bogner, A.; Chanson, M.; Meeuw, A. A Decentralised Sharing App running a Smart Contract on the Ethereum Blockchain. In Proceedings of the 6th International Conference on the Internet of Things (IoT'16); ACM Press, 2016; pp. 177–178.
14. Yuan, Y.; Wang, F.-Y. Towards blockchain-based intelligent transportation systems. In Proceedings of the IEEE 19th International Conference on Intelligent Transportation Systems (ITSC'16); IEEE, 2016; pp. 2663–2668.
15. Zhang, P.; White, J.; Schmidt, D.C.; Lenz, G.; Rosenbloom, S.T. FHIRChain: Applying Blockchain to Securely and Scalably Share Clinical Data. Comput. Struct. Biotechnol. J. 2018, 16, 267–278, doi:10.1016/j.csbj.2018.07.004.
16. Rocha, H.; Ducasse, S. Preliminary steps towards modeling blockchain oriented software. In Proceedings of the 2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB); 2018; pp. 52–57.
17. Lallai, G.; Pinna, A.; Marchesi, M.; Tonelli, R. Software Engineering for DApp Smart Contracts managing workers Contracts. In Proceedings of the Distributed Ledger Technology (DLT'20); 2020.
18. Porru, S.; Pinna, A.; Marchesi, M.; Tonelli, R. Blockchain-Oriented Software Engineering: Challenges and New Directions. In Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C'17); IEEE, 2017; pp. 169–171.
19. Wessling, F.; Ehmke, C.; Hesenius, M.; Gruhn, V. How much blockchain do you need? In Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB '18); ACM Press, 2018; pp. 44–47.
20. Pillai, A.; Saraswat, V.; Arunkumar, V.R. Smart Wallets on Blockchain—Attacks and Their Costs. In Proceedings of the International Conference on Smart City and Informatization; 2019; pp. 649–660.
21. Armknecht, F.; Karame, G.O.; Mandal, A.; Youssef, F.; Zenner, E. Ripple: Overview and outlook. In Proceedings of the International Conference on Trust and Trustworthy Computing; 2015; pp. 163–180.
22. Brown, R.G.; Carlyle, J.; Grigg, I.; Hearn, M. Corda: an introduction. R3 CEV, August 2016, 1, 15.
23. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In Proceedings of the Proceedings of the Thirteenth EuroSys Conference; 2018; p. 30.
24. Taskinsoy, J. Facebook's Project Libra: Will Libra Sputter Out or Spur Central Banks to Introduce Their Own Unique Cryptocurrency Projects? SSRN Electron. J. 2019, doi:10.2139/ssrn.3423453.
25. Valenta, M.; Sandner, P. Comparison of ethereum, hyperledger fabric and corda. [ebook] Frankfurt Sch. Blockchain Cent. 2017.

26. Victor, F.; Lüders, B.K. Measuring Ethereum-Based ERC20 Token Networks. In Proceedings of the Financial Cryptography and Data Security; Goldberg, I., Moore, T., Eds.; Springer International Publishing: Cham, 2019; pp. 113–129.
27. Entriken, W.; Shirley, D.; Evans, J.; Sachs, N. Erc-721 non-fungible token standard. Ethereum Found. 2018.
28. Pustišek, M.; Kos, A. Approaches to front-end iot application development for the ethereum blockchain. Procedia Comput. Sci. 2018, 129, 410–419.
29. Lee, W.-M. Using the MetaMask Chrome Extension. In Beginning Ethereum Smart Contracts Programming; Springer, 2019; pp. 93–126.
30. Iyer, K.; Dannen, C. The ethereum development environment. In Building Games with Ethereum Smart Contracts; Springer, 2018; pp. 19–36.
31. Jones, M.; Tarjan, P.; Goland, Y.; Sakimura, N.; Bradley, J.; Panzer, J.; Balfanz, D. Json web token (jwt). 2012.
32. Alastria Alastria Spanish National Blockchain Consortium Available online: https://alastria.io/ (accessed on Nov 22, 2019).
33. Lucentia Lab Clockchain Available online: https://www.clockchain.es (accessed on May 11, 2020).