Ministère de l'enseignement supérieur et de la recherche scientifique

Université Saad Dahleb Blida

Faculté des Sciences

Département d'informatique

PROJET DE FIN D'ETUDE POUR L'OBTENTION DU DIPLOME DE MASTER EN SECURITE DES SYSTEMES D'INFORMATION

Mémoire réalisée par : MEKKI Rachid

Réalisation d'un système de vote en ligne avec blockchain

Promotrice : Mlle N. BOUSTIA

Présidente : Mlle M. ARKAM

Examinatrice : Mme M. MEZZI

2020/2021

ملخص

العقود والمعاملات هي في صميم مجتمعنا الحديث. إنها تحكم التفاعلات بين الدول والمنظمات العامة والخاصة والمجتمعات والأفراد في جميع أنحاء العالم. هذا هو السبب في أنهم يتعرضون في كثير من الأحيان للإساءة الرقمية والاحتيال ، ولهذا السبب أصبح الأمن السيبراني أكثر أهمية من أي وقت مضى في العصر الرقمي. تعد تقنية بلوك تشين بلعب دور حاسم في هذا السياق.

تجعل تقنية بلوك تشين من الممكن عدم الاعتماد على سلطة مركزية للمصادقة على سلامة المعلومات وملكيتها ، مع السماح بالمعاملات الآمنة والمجهولة المصدر وكذلك الاتفاقات المباشرة بين الأطراف المتفاعلة.

في هذا العمل ، سوف نقدم تقنية بلوك تشين والعقود الذكية وفائدتها في تحسين أمن المعلومات. الدافع الرئيسي في هذا المشروع هو توفير بيئة تصويت آمنة وإظهار أن نظام التصويت الإلكتروني الموثوق به ممكن باستخدام بلوك تشين.
للتحقق من صحة دراستنا ، قمنا بتطوير واجهة ويب لمحاكاة سيناريو التصويت باستخدام عقد ذكي تم نشره في شبكة محلية إثيريوم.

Abstract

Contracts and transactions are at the heart of our modern society. They govern interactions between nations, public and private organizations, communities and individuals around the world. That's why they are frequently subject to digital abuse and fraud, which is why cybersecurity is more important than ever in the digital age. Blockchain technology promises to play a crucial role in this context.

Blockchain technology removes the dependence on a centralized authority to certify the integrity and ownership of information, while enabling secure and pseudo-anonymous transactions and agreements directly between interacting parties.

In this work, we will present the Blockchain technology, smart contracts and their usefulness to improve information security. Our main motivation in this project is to provide a secure voting environment and to show that a reliable electronic voting scheme is possible using blockchain. To validate our study, we have developed a web interface to simulate the voting scenario using a smart contract deployed in an Ethereum local network.

Key words: Blockchain, Ethereum, Smart contract, Decentralized, peer to peer.

Remerciements

Je remercie dieu tout puissant de m’avoir guidée et donné le courage, la volanté et la patience pour réaliser ce travail.

A mes parents qui m’ont soutenue tout au long de mes études, je suis reconnaissant pour votre support, vos sacrifices et votre confiance pour m’avoir apporté un grand soutien moral.

Ma profonde gratitude à ma promotrice Mlle Narhimene Boustia que je remercie pour votre aide au cours de ces dernières années d'études universitaires.

Enfin, mes sincères remerciements à toutes personnes qui ont contribués à l’élaboration de ce travail de près ou de loin.

Contents

Table of figures

Chapter I: Blockchain

1 Introduction:

A blockchain is a distributed software network that functions both as a digital ledger and a mechanism enabling the secure transfer of assets without an intermediary. Just as the internet is a technology that facilitates the digital flow of information, blockchain is a technology that facilitates the digital exchange of units of value. Anything from currencies to land titles to votes can be tokenized, stored, and exchanged on a blockchain network.

1.1. Definition 1 :

Blockchain technology is a simple and creative way of securely and automatically transmitting data from one part of the world to another. It all begins with the creation of a transaction block. A consensus mechanism must validate a block by the majority of computers (nodes) spread around the internet. After verification, the block is attached to the existing chain, and the chain is stored in the digital ledger of every node. In this way, it will create a unique and immutable record. [1]

Since its creation, Blockchain serves as the public transaction ledger. Managed by a large group of computers that are not owned by anyone, Blockchain serves as a transaction ledger, its a timestamped series of unchangable and unalterable records, witchstores a large amount of data. Every blockof data is linked and secured using cryptographyc principles. The information in the network is open for every node in the network, since this network is not owned or controlled by a central authority. Some key features of this technology are decentralization, transparency, and immutability[1].

Blockchain is built on the top of peer-to-peer topology. It is a distributed ledger technology (DLT) that allows data to be stored globally, on thousands of servers. With this technology, each page in a ledger of transactions forms a block. That block has an impact on the next block or page through cryptographic hashing. [1]

The figure I.1 describes how Blockchain mainly consists of a chain of blocks, each one is a record of data that has been encrypted and given a unique identifier called a hash

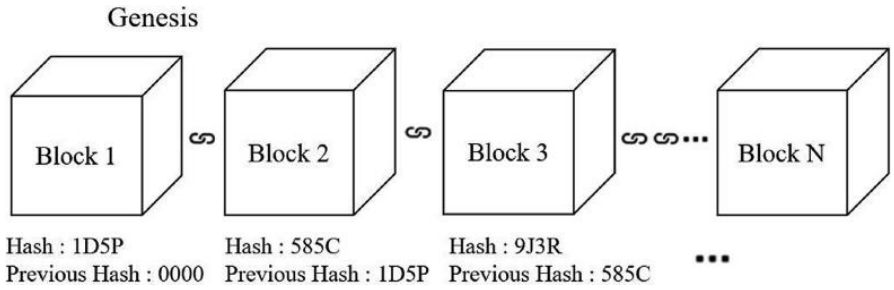

Figure I.1: Chain of block in blockchain[2]

1.2. Definition 2 :

Blockchain is a method of simultaneously recording information on many devices via the internet. The basic blockchain process is as follows. [3]

Assume someone makes a transaction, which must then be registered on the blockchain. A transaction can include the exchange of cryptocurrency, contracts, records, or any other type of information transfer. The requested transaction is then broadcast to a global peer-to-peer network made up of computers called nodes. These nodes have to verify the transaction as legitimate as well as the user's status by means of known algorithms. All the nodes perform the same activities and they all store a copy of the ledger. Once consensus between the nodes is reached, it becomes part of a block of data which contains other transactions on the ledger. If the block is complete, it competes with other blocks to become the next block of the existing blockchain. Once the block is attached and secured using cryptography, it is permanent and unalterable[3]. A blockchain is a spreadsheet that has been replicated thousands of times through a global network of computers. This spreadsheet is updated on a daily basis, allowing new transactions to be added. The decentralized part of blockchain is a consequence of the fact that the system constitutes of a worldwide peer-to-peer network and therefore multiple locations store the same information. Thus, it ensures that the information is easily verifiable and public. As a result, the information on the blockchain is hard to corrupt. It would need a massive amount of computer power to alter the recordings. The name blockchain actually comes from the way data is stored, namely blocks hold information and are thoroughly linked by chains.[3]

Each block in the blockchain contains transactions, a time stamp, a digital signature to identify the account who did the recording and a unique identifying link. This link, usually created by hashing, will point to the previous block in the chain and ensures that information is

unalterable. Therefore, blocks further down the chain are more secured than more recent blocks because many other blocks point to it.[3]

1.3. Definition 3 :

A blockchain is essentially a chain of chronological blocks that are maintained by nodes in a distributed network that do not trust one another. A node is a participant in the blockchain network, which is similar to a peer. Every node in the blockchain network stores the replicated data as blockchains.[4]

Blockchain is considered to be very secure. Every node in a blockchain network is equal-valued and can respond to clients' requests individually, which ensures scalability. Replication of the same data on distributed nodes and consensus ensures fault tolerance. Moreover, since blocks are linked together chronologically through hash values, the manipulation of data is even more challenging. Because as long as one node edit transactions, the hash value of the block changes. Once the nodes append a block into the blockchain, it is almost impossible to manipulate or erase this entry, since one needs to manipulate data in all nodes. By signing a digital signature in each transaction, the identity of the transaction executor can be traced later. Thus, by using cryptography, distributed database, consensus algorithms, and digital signature, blockchain provides "Trust" in a distributed network without the need for a certified central authority.[4]

2. Background :

The Blockchain technique is derived from the Bitcoin which is first proposed by Satoshi Nakamoto in 2008. Bitcoin's main goal is to propose a secure payment system that does not require a trusted third party, allowing online payments to be sent directly from one consumer to another without going through a financial institution. This is achieved by a distributed ledger which accounts for the ownership of coins. The key challenge is to resist double spending attacks, where an adversary could issue two transactions in parallel so as to transfer the same coin to different recipients. Essentially, enabling the distributed ledger to be secure against double spending attacks boils down to the Byzantine Generals problem (BGP). Existing BGP solutions primarily concentrate on the permission model, where the participants in the system are predetermined (or participants know each other). However, the participants who maintain the ledger in a secure digital payment system are under a permissionless model, where anyone can join or leave the system without permitted by a centralized or distributed authority.[5]

Bitcoin reached consensus on the distributed ledger in the permissionless model, in which it cleverly blends current primitives and solutions from decades of study into one framework to address the fundamental problems in digital currencies in a realistic way. Bitcoin is also the first cryptocurrency to be commonly used on a broad scale. Because of Bitcoin's popularity, the underlying technology known as "blockchain" has been extracted, which is of independent interest. Intuitively, the blockchain is a technique that maintains a public, immutable, and ordered ledger of records among all participants. It enables the participants to securely and periodically add new records into the ledger without trusting each other. It also ensures that the records attached to the ledger cannot be deleted or changed, ensuring that all truthful participants have the same view of the ledger. More recently, the blockchain technique has been further investigated and one of the most prominent manifestations is to enable smart contracts. As a disruptive technique with profound implications, Blockchain is changing the way users, including individuals and businesses, conduct transactions and implement smart contracts.[5]

3. Basic dictionary of blockchain :

In Blockchain, setup and network operations are built upon the core components described below:

3.1. Decentralized Applications :

dApps, shorthand for “decentralized applications,” include all applications that are based on distributed ledger technology. In the broadest sense, every cryptocurrency is a dApp, so all dApps should have certain properties[6] :

- All data is stored and cryptographically secured in a blockchain.
- It is free, open-source software, preferably developed by an open community.
- Decisions are reached by community consensus, ideally on the blockchain itself.
- Existing or newly created tokens are used to regulate access and rewards.

3.2. Peer to Peer :

A peer-to-peer (P2P) system is one in which participant peers or nodes rely on each other for service, rather than solely relying on dedicated and often centralized infrastructure. Each peer connects to a relatively small set of neighbors, which in turn also connects to other peers. Each peer in this network is a server, as well as a client at the same time. In a P2P system, users can submit queries to any random peer. The receiving peer first evaluates the query in

its resource base. If the peer can serve resources to the query, it returns the user the answer. Otherwise, the peer forwards the query to its neighbors randomly. To enable peers forward queries more specifically, destination indices can be used. [4]

3.3. Transactions:

A Blockchain transaction can be defined as a small unit of a task that is stored in public records. These records are also known as blocks. These blocks are executed, implemented and stored in blockchain for validation by all miners involved in the blockchain network. Each previous transaction can be reviewed at any time but cannot be updated.[7]

3.4. Block:

Blocks are files in which data about the Bitcoin network is permanently recorded; each block has two major sections. The "header" part of the chain connects to the previous block. This means that any block header contains the hash of the previous block, ensuring that no transaction in the previous block can be changed. The "body content" part of a block contains a validated list of transactions, their amounts, the addresses of the parties involved, and some additional information. So, provided the most recent block, all previous blocks in a blockchain can be accessed.[8]

3.5. Node:

Nodes can be any kind of device (mostly computers, laptops or even servers). All nodes on a blockchain are connected to each other and they constantly exchange the latest blockchain data with each other so all nodes stay up to date. They store, spread and preserve the blockchain data.[9]

When a miner attempts to add a new block of transactions to the blockchain, it broadcasts the block to all the nodes on the network. Based on the block's legitimacy (validity of signature and transactions), nodes can accept or reject the block. When a node accepts a new block of transactions, it saves and stores it on top of the rest of the blocks it already has stored.[9]

Nodes can have different roles in the network. Full nodes, or miners, are permanently connected and store the entire blockchain, they verify and propagate the activities and blocks in the network. Simple payment verification (SPV) nodes, on the other hand, do not store the

entire blockchain. Therefore, they rely on full nodes to receive and propagate transactions throughout the network. SPV nodes basically download the transactions which are significant for them. All nodes are considered equal although some have different tasks. The basic tasks of a node are[3] :

- Check the validity of transactions and add them to existing blocks or simply reject them. [3]
- Save and store blocks of transactions. [3]
- Broadcast and spread the transaction history to other nodes to secure synchronicity.[3]

The figure I.2 shows us the difference in connections between nodes in a decentralized, centralized and distributed networks.

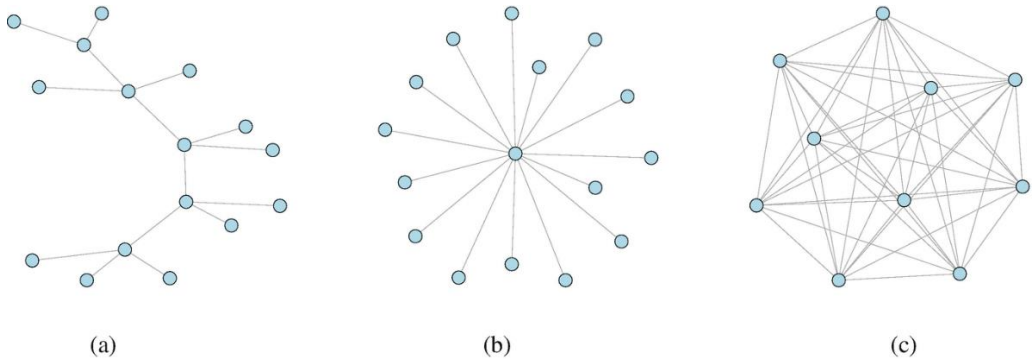

Figure I.2: Different types of graphs. (a) Decentralized. (b) Centralized. (c) Distributed ledger[3]

Nodes are able to come and go to the network in order to allow for a flexible and dynamic system. An active (online) node which goes offline, is forgotten after a predetermined amount of time to ensure a smooth working system. On the other hand, when an offline node comes back, it needs to get back up to speed. The node has to download all the blocks that were added to the blockchain while the node was offline. Theoretically, a single node can keep the blockchain afloat, however, the network would then be highly vulnerable to corruption.[3]

A new transaction is propagated through the network by moving over all the connections between the nodes. Eventually every node is connected to all the nodes in the network by using its peers. Once a node receives a transaction, it checks the senders righteousness and if the money has not yet been spent. Afterwards, the node sends the information to its peers until the whole network knows about the transaction. The time it takes for a whole block of transaction to go through the whole network can be quite long, this algorithm is called the flooding algorithm.[3]

For example, for Bitcoin the probability density function is shown in Figure I.3. Double spending is avoided because only the transaction which links first to a certain unspent transaction, is processed. A node will also reject an unusual script. Moreover, nodes, who act dishonest, will be punished. An example of a punishment is reclaiming the award for blocks received in the past.[3]

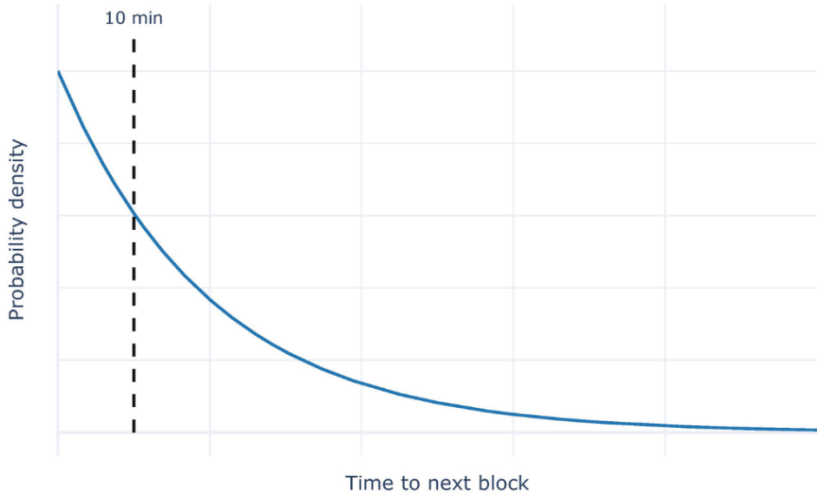

Figure I.3: Bitcon probability density function graph[3]

3.6. Miners:

In the Bitcoin jargon, mining refers to the process of attaching a new block to the Bitcoin ledger. Anybody participating into this process is called a miner. Miners are typically commercial enterprises that operate for profit, as they are rewarded with newly minted bitcoins as well. They are essential to ensure that all bitcoins are produced. As mining has been deliberately designed to be resource-intensive, it shows similarity with the extraction process of commodities such as gold or diamonds. There was a time when it was possible for individuals to mine bitcoins with their own computing equipment.[10]

The goal of the miner is to attach the newly formed block to the ledger. Bitcoin protocol makes it difficult for a new block to be attached. To conform to the rules, new blocks must contain the answer to a cryptographic puzzle.[10]

3.7. Cryptography :

Cryptography is the method of disguising and revealing information by using mathematics. In blockchain technology, cryptography is used with a dual purpose. First, it ensures that the

identity of the practitioners is hidden and second it assures that information is secured as well as the transactions.

The mostly used cryptography method, in case of cryptocurrencies and blockchain, is public-key cryptography. This method allows information to be passed alongside with the public key while the private key stays with the sender. Two different keys encrypt and decrypt the information, the sender's public and private key encrypt the information, while the private key of the receiver together with the sender his public key are used for decryption. Every transaction initiates the key generation algorithm which creates a new public and private key for the sender. Moreover, the public-private-key encryption algorithm places a digital signature on a hashed digest to ensure its integrity, the hashed digest is a product of the hashing algorithm performed on the data. The document itself, together with the private key of the user, is used to construct the signature. Hence, the signature will not match anymore if the data is altered. Moreover, the receiver of the data is able to verify the authenticity of the document by analyzing the digital signature and the data. The receiver uses the public key to decrypt the digital signature and the hash algorithm to scan the document. If both outcomes match, then the receiver knows that the content of the message is not corrupted in transit.[3]

As shown in Figure I.4 when we click on the send button the wallet creates a transaction containing information about the sender the recipient and the amount being sent, then the wallet will produce a unique digital signature by mathematically mixing the message with the sender's private key. After signing the transaction the wallet will group the signature along with the transaction message into a single file.

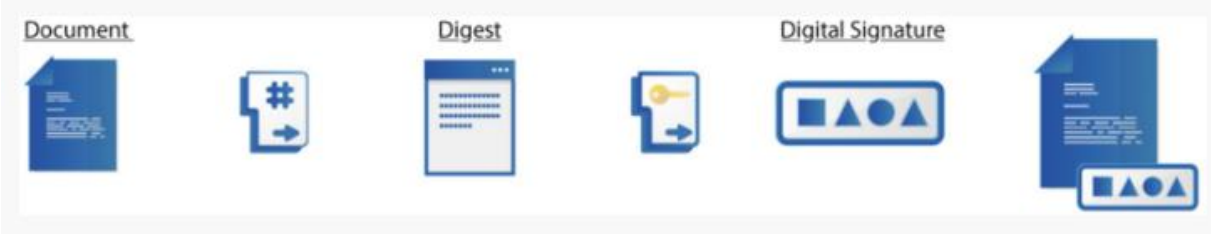

Figure I.4 : Cryptography steps in blockchain[3]

3.8. Hash Algorithm :

The hashing algorithm takes an input of arbitrary length and transforms this by using mathematical transformations to an output of a fixed length. Bitcoin, in particular, uses SHA-256 hashing algorithm. This particular algorithm translates every transaction to an output of seemingly random numbers of 256 bits. The input, in general, can be a transaction or a

document but it can even be a block of transactions. The algorithm is in some sense deterministic, because the same input will always generate the same output. It is almost impossible to determine the actual length of the input due to the fixed length of the hashed data. A good hashing function has the following five main properties[3] :

- **Deterministic**: Changing one single detail in the input completely changes the hashed output. Hence, the hashing function provides some kind of security.
- **Computationally efficient** : The time elapsed to produce the output should be small.
- **Collision resistance** : A collision happens when a hash function maps two different inputs to the same outcome. This is extremely hard theoretically and in practice almost impossible to find two different inputs in the hash algorithm to give the same output.
- **Hiding feature**: It is impossible to find the input value based on the hash value. Theoretically it is possible, in practice however this is computationally infeasible.
- **Random looking output** : It is hard to predict the output of the algorithm.

In the particular case of cryptocurrencies, cryptographic hash functions are used. One of the cryptographic hashing algorithms main objectives is to let the receiver check the authenticity of the information. In other words, this algorithm is implemented for information security, authenticity control and other security measures. The user checks the information by running it through a hashing algorithm. If the product of this algorithm matches the hash output send by the sender, the receiver is sure that the information has not been tampered with throughout the process.

The system does not hash an entire block of the chain at once, it rather hashes each transaction on its own. Afterwards each hashed transactions is linked together, by using hash pointers, with the remaining hashed transactions in the block.[3]

3.9. Merkle tree :

Blockchain most often uses a Merkle tree, also called a hash tree, to represent the hashed and recorded data on the Blockchain. This concept was introduced by Ralph Merkle in 1979. Merkle trees allow the user to validate individual transactions without having to download the entire blockchain.[3]

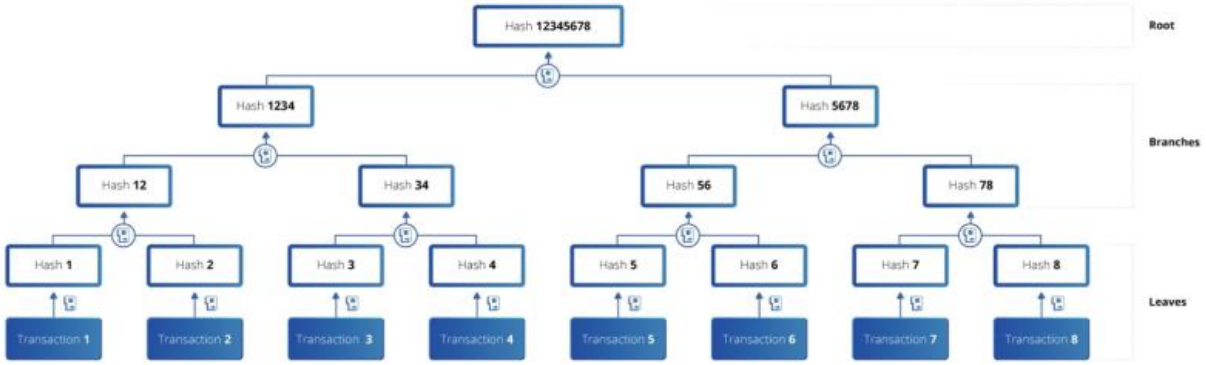

Figure I.5 : Merkel tree[3]

A Merkle tree is a type of binary tree, composed of a set of nodes with a large number of leaf nodes at the bottom of the tree containing the underlying data, a set of intermediate nodes where each node is the hash of its two children, and finally a single root node, also formed from the hash of its two children, representing the "top" of the tree. The purpose of the Merkle tree is to allow the data in a block to be delivered piecemeal: a node can download only the header of a block from one source, the small part of the tree relevant to them from another source, and still be assured that all of the data is correct. The reason why this works is that hashes propagate upward: if a malicious user attempts to swap in a fake transaction into the bottom of a Merkle tree, this change will cause a change in the node above, and then a change in the node above that, finally changing the root of the tree and therefore the hash of the block, causing the protocol to register it as a completely different block (almost certainly with an invalid proof of work).[11]

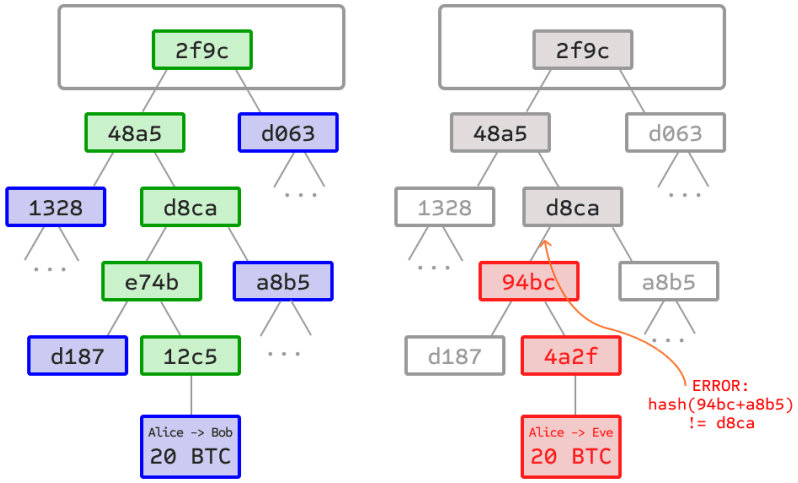

Figure I.6: Blockchain immutability in Merkel tree [11]

The Merkle tree protocol is arguably essential to long-term sustainability. A "full node" in the Bitcoin network, one that stores and processes the entirety of every block, takes up about

15 GB of disk space in the Bitcoin network as of April 2014, and is growing by over a gigabyte per month. Currently, this is viable for some desktop computers and not phones, and later on in the future only businesses and hobbyists will be able to participate. A protocol known as "simplified payment verification" (SPV) allows for another class of nodes to exist, called "light nodes", which download the block headers, verify the proof of work on the block headers, and then download only the "branches" associated with transactions that are relevant to them. This allows light nodes to determine with a strong guarantee of security what the status of any Bitcoin transaction, and their current balance, is while downloading only a very small portion of the entire blockchain.[11]

4. The types of Blockchain:

All blockchain structures fall into three categories or in orther words three types :

4.1. Public Blockchain :

These blockchains are intended to dispose of the requirement for a middleman and designed to be accessible by anyone with a computer and Internet access. Public blockchains are most applicable where a pure decentralized transaction is needed. Examples are Bitcoin blockchain and others like Ethereum, IOTA, and more. Some public blockchains limit the access to just reading or writing. Bitcoin, for example, uses an approach where anyone can write. These public blockchains are slow and resources intensive due to the computational capacity and power that’s required to carry these redundant transactions, however, they are very secure. Open public blockchains are most applicable where a pure decentralized transaction needs to occur.[12]

4.2. Private Blockchain :

In a private blockchain, a company sets up a permission network where all the participants are known and trusted. Access to the networking system is restricted. An invitation or permission needs to be obtained by the participants in order to join. The verification process may be allowed to be performed by only certain nodes through administrators. This is useful when the blockchain is used between companies that belong to the same legal entity. Private blockchain offers several benefits, including faster transaction speeds, privacy of the data/content, and a centralized control over providing access to the blockchain. Private blockchains are appropriate for more traditional business and governance models. For example, governments could use a private blockchain for voting polls and save billions at the

same time because the voting becomes fully resistive against corruption and truly secure. Much of the private blockchain-based application is taking place in the financial services. Many companies are all testing private blockchain technology as a replacement for paper-based and manual transaction processing.[12]

4.3. Consortium Blockchain :

This type of blockchain offers the benefits of both public and private blockchains. Consortium blockchain consists of the public blockchain (that all participants are a part of) and a private network (permission or invitation-based) that restricts participation. the consensus process is controlled by a preselected set of nodes[12]. Specifically, part of the network has the characteristic of being public (i.e. accessible to any participant), but only a privileged group is responsible for the consensus process. Thus, they are partially decentralised as the blockchain is only distributed among entitled participants[10]. Highly regulated enterprises and governments can benefit from a consortium blockchain. The technology enables flexibility and control over what data is kept private versus shared on a public ledger. There are several real-life examples of hybrid blockchain. For example, XinFin is a hybrid blockchain built on both Ethereum (a public blockchain), and Quorum (a private blockchain).[12]

5. Blockchain consensus :

The consensus mechanism including distributed computing, load balancing, and transaction validation in blockchains is the core support technology for stable operation and orderly derivation of the blockchain, which is used to solve the consistency problem of distributed systems. It guarantees the persistence of the ledger data. The goal of a blockchain consensus protocol is to ensure that all participating nodes agree on a common network transaction history, which is serialized in the form of a blockchain. In recent years, people have continuously researched on consensus algorithms and made certain progress. From incremental modifications of Nakamoto consensus protocol to innovative alternative consensus mechanisms, many consensus protocols have been proposed to improve the

performance of the blockchain network itself or to accommodate other specific application needs.[13]

5.1. Proof of Work :

Proof-of-Work (PoW) is the consensus algorithm implemented in Bitcoin. It was introduced by Bitcoin.com. Today, this algorithm is considered by some problematic due to its enormous energy drainage[3]. In proof-of-work blockchain, miners compete in brute-force search to successfully solve a hard cryptographic puzzle and win reward for block generation. Practically the puzzle is to construct a block whose hash is less than a certain value, which also known as difficulty of mining. Only block that follows the latest block and has a hash value less than difficulty of mining will be accepted by network. In the view of long term, the received reward for mining is proportional to rate of computational power nodes contribute in network[14]. Many recent cryptocurrencies do not use this consensus algorithm anymore; Ethereum is moving towards another algorithm. Proof-of-Work chooses the liveness and fault tolerance characteristics[3].

PoW has its issues. The most prominent problem is that PoW is not efficient, it needs lots of energy and computer power. It has been reported that a single Bitcoin transaction can cost as must energy as a Dutch household consumes in 2 weeks. Due to the high energy need, participants want a substantial amount of coins as reward to keep participating. If the energy consumption would go down, the reward can also go down. Another problem is that some companies and people mine blocks easier and faster because they have better equipment. Therefore, some users have a clear advantage over others, in other words, the blockchain becomes more centralized. [3]

5.2. Proof of Stake :

A main consideration regarding the operation of blockchain protocols which are based on PoW such as bitcoin is the energy required for their execution. For example, generating a single block on the bitcoin blockchain requires a number of hashing operations exceeding 2^{60}, which results in striking energy demands. [15]

The Proof-of-Stake algorithm was first used in 2011 in a coin called Peercoin[3]. PoS attempts to solve the energy expenditure problem created by PoW. To do so, PoS replaces PoW's competition by randomly selecting stake-holders to append to the blockchain. So, rather than

miners investing computational resources in order to participate in the leader election process, they instead run a process that randomly selects one of them proportionally to the stake that each possesses according to the current blockchain ledger. [15]

The main differences with the PoW algorithm are that miners are replaced with validators and the next validator of a block is chosen in a deterministic way. The system is called Proof-of-Stake because the more currency a user owns, the higher the amount at stake if the network is under attack[3].

In order to become a validator in the PoS algorithm, the user needs to have a certain amount of the currency at stake. Afterwards, the validators start to check all the presented blocks and try to find which block needs to be added to the chain by placing bets on the blocks. The probability of receiving the reward is proportionate to the amount of currency the validators put at stake. For example, if you have three validators, where the first validator has 3 coins, the second validator inserts 2 coins and the third has 5. Then, the probability that the first validator is chosen to validate the block is 30%, the second has 20% chance and the third 50%. In other words, some individuals have an higher chance to be chosen than others. However, everyone has a chance at the reward which is basically a part of all of the transaction costs and due to this randomization the PoS algorithm prevents centralization.[3]

If an individual wants to perform a 51% attack in this system, he needs to own the majority of all the coin in existence and this would be very costly to achieve. Moreover, the other individuals would either exit the currency, if they notice, or will drive up the price to prevent it. In general, the user with the highest amount of currency, has the highest incentive to keep the network secure.[3]

The PoS consensus protocol has one major flaw, namely the “nothing at stake” problem. This event occurs when the consensus fails. The block generators in this case are indifferent between supporting various blockchain histories. In other words, many validators build on every fork if forks occur.[3]

5.3. Delegated Proof of Stake :

A DPoS can be seen as a democratic form of committee-based PoS as the committee (consensus group) is chosen via public stake delegation. DPoS was designed to control the size of the consensus group so that the messaging overhead of the consensus protocol remains

manageable. The members of the consensus group are also called delegates. The election of delegates is called the delegation process. In the actual case, the delegation process and the soliciting of votes may involve outside incentives. Generally, an aspirating delegate needs to attract enough votes from normal token holders. This is often accomplished by offering a popular application and building up the reputation through propaganda campaigns. By casting a vote to a delegate via a blockchain transaction, a token holder entrusts the delegate his own stake. As a result, the delegate harvests the stake voting power from his voters.[15]

Delegated Proof-of-Stake combines real-time voting and a system of reputation to reach consensus. Every token owner in this protocol has a certain amount of power in the block producing process. The voting power of a token owner is proportionate to the number of tokens he or she possesses. Moreover, voters can give a proxy to another voter, to vote on their behalf. Block producers, also called witnesses, are selected by the token owners and are responsible for creating and signing new blocks. The number of block producers is limited, as a consequence this system is not fully decentralized. Furthermore, witnesses have to prove they have the best interest of the network at heart. Block producers can be voted in or out at any time. DPoS uses the threat of losses and reputation damage as incentives to keep the system honest.[3]

DPoS is not the perfect system, it exchanges decentralisation for scalability. Therefore, the system is easier to corrupt than Proof-of-Work. For example, if many users of the system are bribed, than the majority of block producers can be dishonest. Moreover, there can be an unlimited amount of forks. However, this behaviour will not last long, since these corrupt block producers will be voted out.[3]

5.4. Proof-of-Authority :

PoA is designed to be used alongside nonpublic blockchain systems. The key idea is to designate a set of nodes as the authority. These nodes are entrusted with the task of creating new blocks and validating the transactions. PoA marks a block as part of the blockchain if it is signed by majority of the authorized nodes. The incentive model in PoA highlights that it is in the interest of an authority node to maintain its reputation, to receive periodic incentives. Hence, PoA does not select nodes based on their claimed stakes.[16]

6. Layers of blockchain :

Blockchain is being used in a variety of fields, each with its own set of applications. In contrary to the internet, the structure of Blockchain cannot be generalized since different usage cases would necessitate different structures.[1]

6.1. Hardware or infrastructure layer :

The first layer of the Blockchain is the infrastructure layer or hardware layer. The Blockchain contents are maintained on nodes, which are servers or high-performance computers that are located remotely. Each node is in charge of validating transactions, organizing transactions inside blocks, broadcasting transactions to the Blockchain network, and so on. These nodes are also in charge of connecting a block to the network. Following consensus, these nodes are also in charge of adding a block to the chain.[1]

6.2. Data layer :

An individual block in the Blockchain is a container data structure that can hold far more than a list of transactions. This layer is in charge of the block structure in the Blockchain. This layer is responsible for the structure of the block in the Blockchain. A single block's data items are split into two sections, block header and block body. The block body contains the list of transactions, while the block header contains metadata about the block in question.[1]

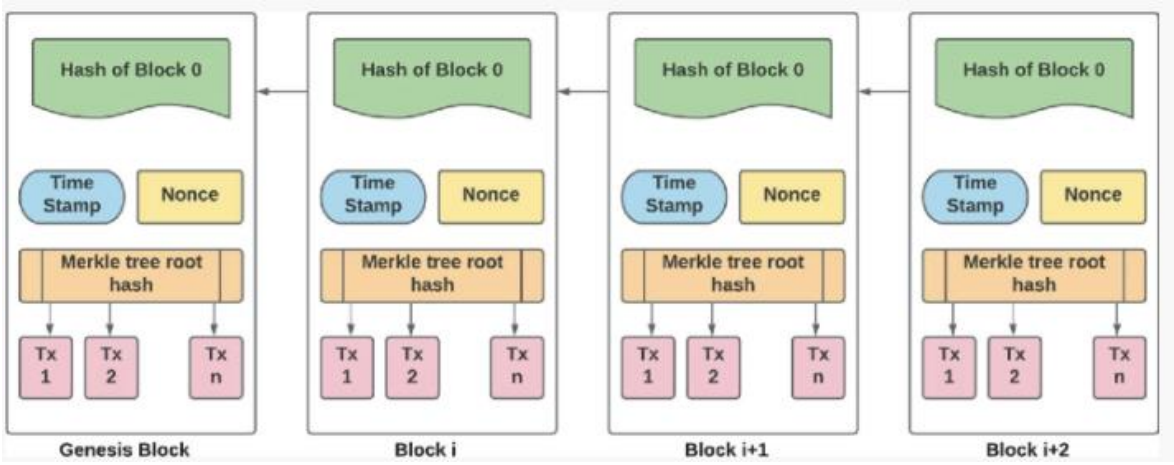

Figure I.7: The different attributes of block in a Blockchain[1]

Block Header :

The Block header section of the block contains all the data about the data in the particular block. The block header of the bitcoin block is further subdivided into six subsections, such as[1] :

- **Version :** The version number of the software.
- **Last Block :** The hash value of the previous block.
- **Merkle Root :** The root of the Merkle tree.
- **Time :** The time in second since 1970-01-01 T00:00UTC.
- **Miner :** Information about the miner.
- **Target :** The hash size in bits.
- **The Nonce :** A variable incremented by the proof of work.

Block Body :

The block body contains all of the block's confirmed transactions. When a transaction needs to be checked, the network's nodes check the transactions contained in the block body.[1]

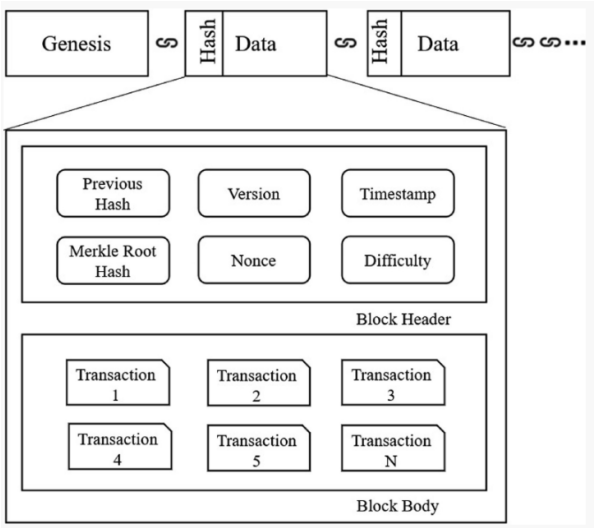

Figure I.8 : Structure of a block[2]

6.3. Network Layer :

In the Blockchain, the network layer is also known as the P2P or peer-to-peer layer. The full node and the light node are two different types of nodes. Full nodes are in charge of transaction verification and confirmation, mining, and consensus. The light nodes, on the

other hand, just hold the Blockchain headers and submit them as needed. The network layer is responsible for communication between nodes.[1]

6.4. Consensus Layer :

This is the most important and vital layer of any Blockchain. This layer is in charge of validating and organizing the blocks. The consensus mechanism is the reason that Blockchains are more secure and trusted than any other conventional method.[1]

6.5. Application Layer :

The application layer or the presentation layer is the bottom-most layer. This layer is comprised of smart contracts, chain code, and apps. The application layer is further divided into two sub-layers, such as the application sub-layer and execution sub-layer. The application sub-layer is a collection of software packages that are used by the users to interact with the Blockchain. The execution sub-layer constitutes smart contracts, chain code, and underlying rules. This sub-layer has the actual executable code and rules to be followed. [1]

7. Evolution of blockchain :

The functionalities of blockchain technology have evolved over time. Today, we can distinguish three stages :

7.1. Blockchain 1.0 :

Blockchain 1.0, an early-stage model of applying blockchain, which was used exclusively for the exchange as an electronic transaction system for the purpose of exchanging money is a mechanism in which all nodes manage each other's latest copies in a decentralized manner, did go live in 2009 as the technology behind Bitcoin. This innovative technology, which can prove the existence or non-existence of data and whether they have been tampered with by assuring the authenticity of the data by all network participants, has made it possible for network participants to trust each other's exchange of "who sent what information to whom" without relying on a centralized method, thereby enabling value transfer on a decentralized network[17].

7.2. Blockchain 2.0 :

The wasteful mining and poor scalability of the first generation Blockchain prompted Buterin to extend the concept of Blockchain beyond currency. This led to the advent of second

generation of Blockchain i.e. Ethereum which is based on new concepts of smart contracts along with Proof of Work consensus mechanisms.[2]

Smart Contracts are autonomous self-managing computer programs that execute automatically on the basis of predefined clauses between two parties. These contracts are impossible to be hacked or tampered with. So Smart Contracts largely reduce the cost of verification, execution, and fraud prevention and enable transparent contract definition.[2]

7.3. Blockchain 3.0 :

The major setback of Blockchains 1.0 and 2.0 are that they are not scalable at all, mostly based on Proof of Work and take hours to confirm transactions. All this let to the birth of the current generation of Blockchain called Blockchain 3.0 that aims to make cryptocurrencies globally viable. Apart from smart contracts, the third generation of Blockchain mainly involves Decentralized Apps (dApps). They are digital programs that run on a Blockchain network of computers instead of a single computer and thus are beyond the purview of any central authority. [2]

8. Conclusion :

In this chapter we have presented Blockchain technology, its background, the mecanism behind it and its hack proof capability, this thechnology has potential to change the traditional industry with the help of its evolotuion and the major spotlight the world is giving it.

Blockchain technology is expected to bring in a huge technological revolution. As technology advances, it is projected to gain broader acceptance in different domains. This technology's main strength is its ability to resist hacking. At the moment, society is most likely attempting to decipher both the corporate rationale and the technological rationale of this technology.

Blockchain technology is game-changing. It will make life easier and safer by changing how personal information is stored and how goods and services are purchased. Every transaction is recorded in a permanent and unchangeable record thanks to blockchain technology. Fraud, hacking, data theft, and information loss are all impossible with this unbreakable digital ledger.

In the next chapter we will cover the ethereum platform, the concept behind it and more information and terms related to this platform.

Chapter II: Ethereum:

1. Ethereum:

In 2013, Vitalik buterin, who thought of the blockchain as more than just a ledger envisioned the blockchain as a computational platform that could execute well-defined functions using contracts and arguments. This is made possible by the Ethereum Virtual Machine (EVM). The EVM allows for complete isolation of the executable code and safe execution of the applications built on the blockchain.[18]

Ethereum is a blockchain-based software platform that is primarily used to support the world's second-largest cryptocurrency by market capitalization after Bitcoin. Like other cryptocurrencies, Ethereum can be used for sending and receiving value globally and without a third party watching or stepping in unexpectedly. [19]
Value exchange is the main use case of the Ethereum blockchain today, often via the blockchain's native token, ether. But many of the developers are working on the cryptocurrency because of its long-term potential and the ambitious vision of its developers to use Ethereum to give users more control of their finances and online data. [19]

Ethereum is open access to digital money and data-friendly services for everyone no matter the background or location. It's a community-built technology behind the cryptocurrency ether (ETH) and thousands of applications you can use today.[20]

The intent of Ethereum is to create an alternative protocol for building decentralized applications, providing a different set of tradeoffs that we believe will be very useful for a large class of decentralized applications, with particular emphasis on situations where rapid development time, security for small and rarely used applications, and the ability of different applications to very efficiently interact, are important. Ethereum does this by building what is essentially the ultimate abstract foundational layer : a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions. A bare-bones version of Namecoin can be written in two lines of code, and other protocols like currencies and reputation systems can be built in under twenty. Smart contracts, cryptographic "boxes" that contain value and only unlock it if certain conditions are met, can also be built on top of the platform, with vastly more power than that offered by Bitcoin scripting because of the added powers of Turing-completeness, value-awareness, blockchain-awareness and state.[21]

The design behind Ethereum is intended to follow the following principles :

Simplicity :

The Ethereum protocol should be as simple as possible, even at the cost of some data storage or time inefficiency. An average programmer should ideally be able to follow and implement the entire specification.[11]

Universality :

A fundamental part of Ethereum's design philosophy is that Ethereum does not have "features". Instead, Ethereum provides an internal Turing-complete scripting language, which a programmer can use to construct any smart contract or transaction type that can be mathematically defined.[11]

Modularity :

The parts of the Ethereum protocol should be designed to be as modular and separable as possible. Over the course of development, our goal is to create a program where if one was to make a small protocol modification in one place, the application stack would continue to function without any further modification.[11]

Agility :

Details of the Ethereum protocol are not set in stone. Although we will be extremely judicious about making modifications to high-level constructs, for instance with the sharding roadmap, abstracting execution, with only data availability enshrined in consensus. Computational tests later on in the development process may lead us to discover that certain modifications, e.g. to the protocol architecture or to the Ethereum Virtual Machine (EVM), will substantially improve scalability or security. If any such opportunities are found, we will exploit them.[11]

Non-discrimination and non-censorship :

The protocol should not attempt to actively restrict or prevent specific categories of usage. All regulatory mechanisms in the protocol should be designed to directly regulate the harm and not attempt to oppose specific undesirable applications. A programmer can even run an infinite loop script on top of Ethereum for as long as they are willing to keep paying the per-computational-step transaction fee.[11]

2. Ethereum world state trie :

World state trie is a mapping between addresses and account states. It can be seen as a global state that is constantly updated by transaction executions. The Ethereum network is a decentralized computer and state trie is considered hard drive. All the information about accounts are stored in world state trie and you can retrieve information by querying it. World state trie is closely related to account storage trie because it has "storageRoot" field that points the root node in account storage trie.[22]

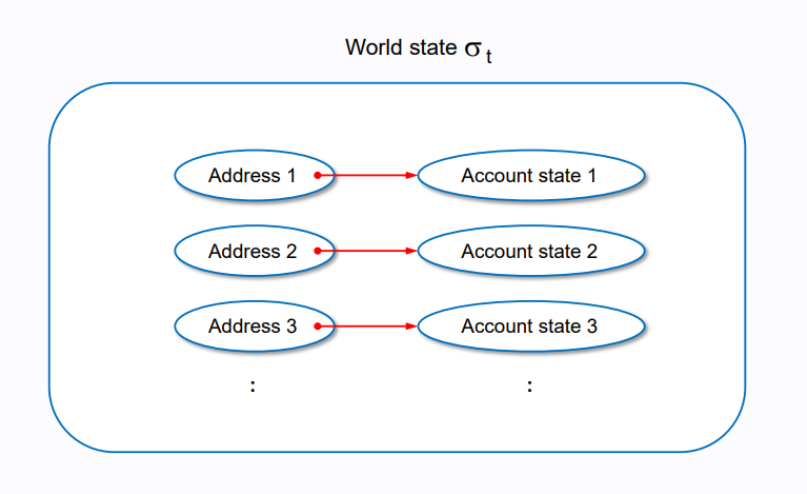

Figure II.1: Representation of the Ethereum world sate[23]

3. Web3 :

The web3.js is a collection of libraries that allow you to interact with a local or remote Ethereum node, using HTTP, WebSocket, or IPC. Through the web3.js APIs, the front end can then interact with the Smart Contracts. The web3.js APIs contain the following modules[24]:

- web3-eth – For the Ethereum blockchain and smart contracts
- web3-shh – For the whisper protocol to communicate p2p and broadcast
- web3-bzz – For the swarm protocol, the decentralized file storage
- web3-utils – Contains useful helper functions for DApp developers.

Web 3.0 is an internet where core services like DNS and digital identity are decentralized, and where individuals can engage in economic interactions with each other.[25]

4. Ethereum wallets :

Ethereum wallets are used by the owners of Ethereum accounts who save the private key and the public key of their accounts in a keystore file. The hot wallets store the key information online which can be accessible virtually from wherever the account holder has an internet connection. The cold wallets keep the keys offline such as on a hardware device or in a paper wallet in the form of a QR code. The cold wallets are less vulnerable to attackers.[26]

Ethereum wallets connect to the blockchain network fully or partially according to their configurations as either a full node or not. There are different types of Ethereum wallets which work on multiple platforms including desktop, mobile, web, and hardware devices.[26]

4.1. Full Node Wallets :

Full Node Wallets/Desktop Wallet download the entire blockchain data and connect locally to operate their functions. Full node means one of the peers in the whole distributed Ethereum network. Since the blockchain data are very big (gigabytes in size), it is possible for a desktop wallet to run as a full node.[26]

4.2. Web Wallets :

Web Wallets are online wallets that can be accessed from anywhere since wallet data are stored in a cloud environment. Since web wallets are light weight clients, they perform faster than other wallets. These wallets are most susceptible to malicious hacks and cipher attacks.[26]

5. Gas:

Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network.

Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas refers to the fee required to successfully conduct a transaction on Ethereum.[27]

The total execution cost for a contract consists of two components, namely the gas cost in units and gas price per unit. The gas cost is split into a fixed base cost of 21 000 gas and an execution cost dependent on the instructions executed while running the contract. [28]

Gas comes with the following keywords :

- **Gas limit :** This is the maximum amount of gas you are willing to pay to the miner for validating the transaction. The higher the price, the greater the chance that your transaction will be executed faster as that will attract more miners to prioritize your transaction over others. Also, insufficient gas in the gas limit will result in a failed transaction.[29]

  Due to the Turing-completeness of the EVM, the exact computational cost of a transaction cannot be predetermined. Hence, the sender is required to specify a *gas limit*, or the maximum amount of gas that may be consumed. As the computational steps of a transaction are executed, the required gas is subtracted from the paid gas. Once a transaction is completed, any unused gas will be refunded to the sender.[28]
- **Gas price :** This is the amount of ether or fraction of ether you are willing to spend on every unit of gas. The gas price is usually some amount of gwei, which is a fraction of a wei. Wei is the smallest unit of ether, and gwei is equivalent to 1000000000 wei.[29]

  Miners set a cut-off gas price to choose which transactions to include in their memory pool. When constructing a new block, they then choose the transactions with the most lucrative gas prices from their memory pool. A higher gas price will increase the fee which miners receive from a transaction, thereby motivating a miner to include a transaction in a block. [28]
- **Gas cost :** This is a static value for a particular operation.[29]

6. Turing Completeness :

Turing completeness is a property that describes the ability of a programming language to emulate a Turing machine. In fact, Turing completeness may be found in the great majority of computer languages. The Ethereum Virtual Machine is the most well-known example of a Turing complete smart contract machine (EVM).[30]

Ethereum was designed to be a Turing Complete blockchain. This is significant because it is required to understand the agreements that comprise smart contracts. Because Ethereum is Turing Complete, it has the potential to interpret and implement any future agreement, even those that have not yet been thought of. In other words, Ethereum's Turing Completeness means that it can use its code base to execute nearly any work as long as it has the right instructions, enough time, and enough computing power.[31]

7. Ethereum Virtual Machine:

7.1. Definition:

The Ethereum Virtual Machine or EVM is the runtime environment for smart contracts in Ethereum. It is not only sandboxed but actually completely isolated, which means that code running inside the EVM has no access to network, filesystem or other processes.[32]

7.2. Ether :

The purpose of Ether, the cryptocurrency, is to allow for the existence of a market for computation. Such a market provides an economic incentive for participants to verify/execute transaction requests and to provide computational resources to the network. [33]

Any participant who broadcasts a transaction request must also offer some amount of ether to the network, as a bounty to be awarded to whoever eventually does the work of verifying the transaction, executing it, committing it to the blockchain, and broadcasting it to the network. [33]

The amount of ether paid is a function of the length of the computation. This also prevents malicious participants from intentionally clogging the network by requesting the execution of infinite loops or resource-intensive scripts, as these actors will be continually charged.[33]

7.3. Accounts :

In Ethereum, the state is made up of objects called "accounts", with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields:

- The nonce, a counter used to make sure each transaction can only be processed once.
- The account's current ether balance.
- The account's contract code, if present.
- The account's storage (empty by default).

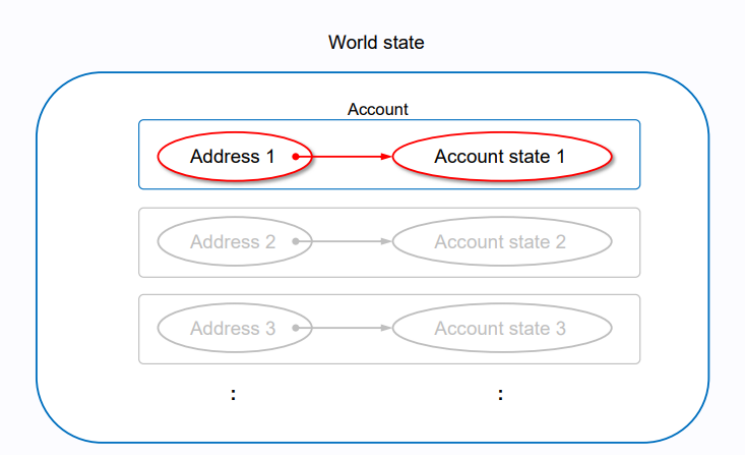

Figure II.2: Representation of an account[23]

There are two kinds of accounts in Ethereum which share the same address space: External accounts that are controlled by public-private key pairs (i.e. humans) and contract accounts which are controlled by the code stored together with the account.[32]

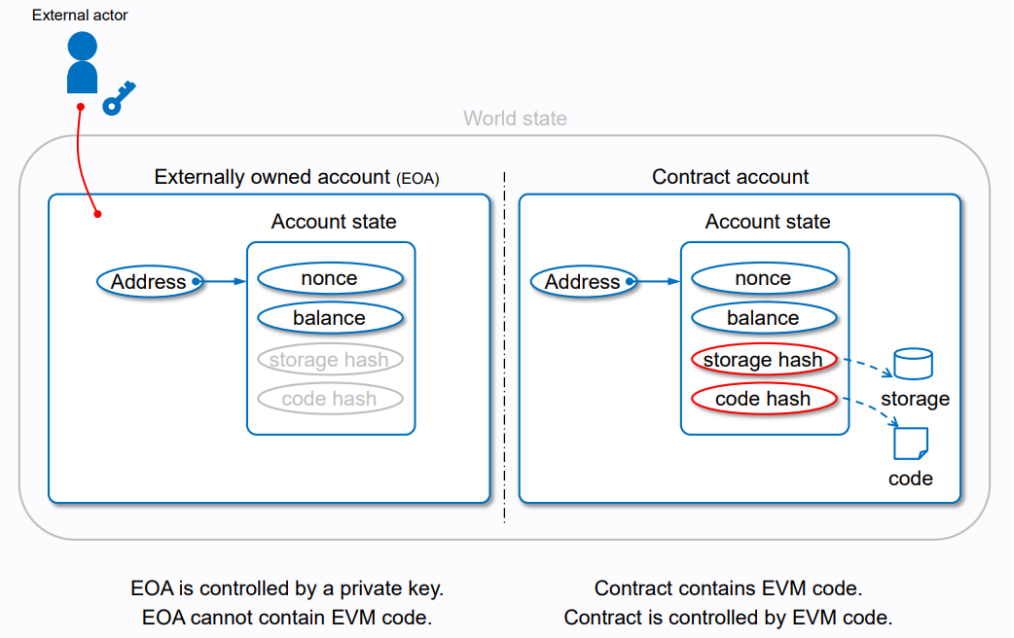

Figure II.3: External owned account and contract account[23]

The address of an external account is determined from the public key while the address of a contract is determined at the time the contract is created (it is derived from the creator address and the number of transactions sent from that address, the so-called “nonce”).[32]

Regardless of whether or not the account stores code, the two types are treated equally by the EVM.[32]

Furthermore, every account has a balance in Ether (in "Wei" to be exact, 1 ether is 10^{18} wei) which can be modified by sending transactions that include Ether.[32]

7.4. Storage Memory and Stack :

The Ethereum Virtual Machine has three areas where it can store data- storage, memory and the stack, which are explained in the following paragraphs[32]

Each account has a data area called storage, which is persistent between function calls and transactions. Storage is a key-value store that maps 256-bit words to 256-bit words. It is not possible to enumerate storage from within a contract, it is comparatively costly to read, and even more to initialise and modify storage. Because of this cost, you should minimize what you store in persistent storage to what the contract needs to run. Store data like derived calculations, caching, and aggregates outside of the contract. A contract can neither read nor write to any storage apart from its own.[32]

The second data area is called memory, of which a contract obtains a freshly cleared instance for each message call. Memory is linear and can be addressed at byte level, but reads are limited to a width of 256 bits, while writes can be either 8 bits or 256 bits wide. Memory is expanded by a word (256-bit), when accessing (either reading or writing) a previously untouched memory word (i.e. any offset within a word). At the time of expansion, the cost in gas must be paid. Memory is more costly the larger it grows (it scales quadratically).[32]

The EVM is not a register machine but a stack machine, so all computations are performed on a data area called the stack. It has a maximum size of 1024 elements and contains words of 256 bits. Access to the stack is limited to the top end in the following way: It is possible to copy one of the topmost 16 elements to the top of the stack or swap the topmost element with one of the 16 elements below it. All other operations take the topmost two (or one, or more, depending on the operation) elements from the stack and push the result onto the stack. Of course it is possible to move stack elements to storage or memory in order to get deeper access to the stack, but it is not possible to just access arbitrary elements deeper in the stack without first removing the top of the stack.[32]

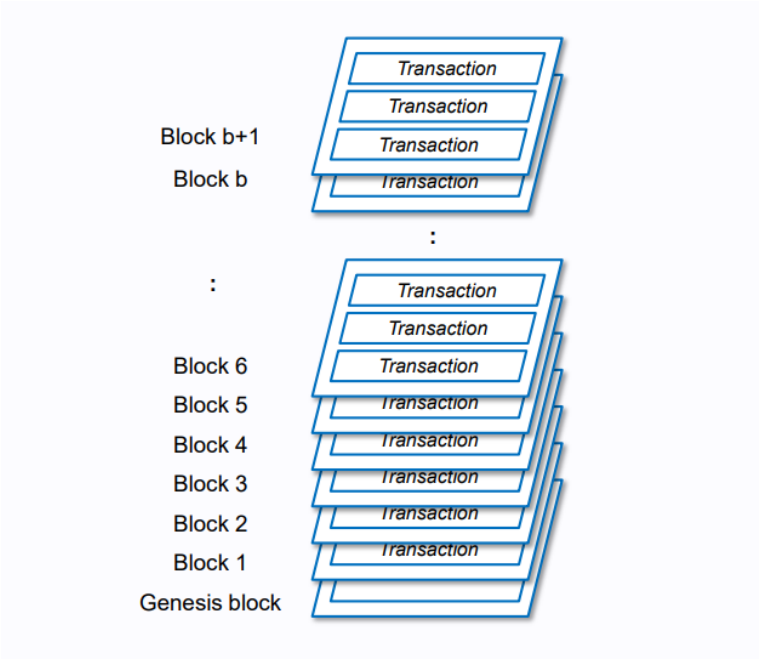

Figure II.4: Ledger as a stack of transaction[23]

7.5. Transactions:

The term "transaction" is used in Ethereum to refer to the signed data package that stores a message to be sent from an externally owned account. Transactions contain:[21]

- The recipient of the message
- A signature identifying the sender
- The amount of ether to transfer from the sender to the recipient
- An optional data field
- A STARTGAS value, representing the maximum number of computational steps the transaction execution is allowed to take
- A GASPRICE value, representing the fee the sender pays per computational step

The first three are standard fields expected in any cryptocurrency. The data field has no function by default, but the virtual machine has an opcode which a contract can use to access the data; as an example use case, if a contract is functioning as an on-blockchain domain registration service, then it may wish to interpret the data being passed to it as containing two "fields", the first field being a domain to register and the second field being the IP address to register it to. The contract would read these values from the message data and appropriately place them in storage.[21]

The STARTGAS and GASPRICE fields are crucial for Ethereum's anti-denial of service model. In order to prevent accidental or hostile infinite loops or other computational wastage in code, each transaction is required to set a limit to how many computational steps of code execution it can use. The fundamental unit of computation is "gas"; usually, a computational step costs 1 gas, but some operations cost higher amounts of gas because they are more computationally expensive, or increase the amount of data that must be stored as part of the state. There is also a fee of 5 gas for every byte in the transaction data. The intent of the fee system is to require an attacker to pay proportionately for every resource that they consume, including computation, bandwidth and storage; hence, any transaction that leads to the network consuming a greater amount of any of these resources must have a gas fee roughly proportional to the increment.[21]

7.6. Message calls:

Contracts can call other contracts or send Ether to non-contract accounts by the means of message calls. Message calls are similar to transactions, in that they have a source, a target, data payload, Ether, gas and return data. In fact, every transaction consists of a top-level message call which in turn can create further message calls.

Essentially, a message is like a transaction, except it is produced by a contract and not an external actor. A message is produced when a contract currently executing code executes the CALL opcode, which produces and executes a message. A message contains:[21]

- The sender of the message (implicit)
- The recipient of the message
- The amount of ether to transfer alongside the message
- An optional data field
- A STARTGAS value

A contract can decide how much of its remaining gas should be sent with the inner message call and how much it wants to retain. If an out-of-gas exception happens in the inner call (or any other exception), this will be signaled by an error value put onto the stack. In this case, only the gas sent together with the call is used up. In Solidity, the calling contract causes a manual exception by default in such situations, so that exceptions “bubble up” the call stack.[32]

The called contract will receive a freshly cleared instance of memory and has access to the call payload - which will be provided in a separate area called the calldata. After it has finished execution, it can return data which will be stored at a location in the caller's memory preallocated by the caller. All such calls are fully synchronous.[32]

Calls are limited to a depth of 1024, which means that for more complex operations, loops should be preferred over recursive calls.[32]

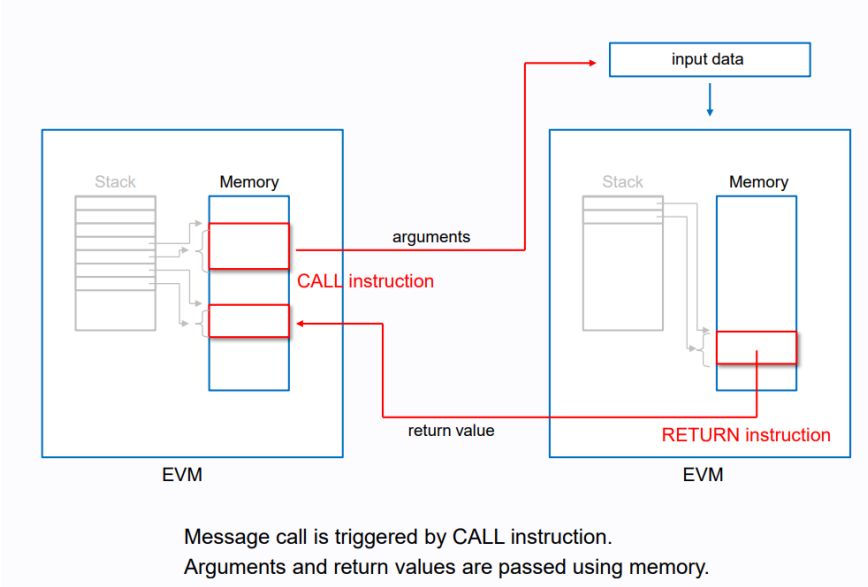

Figure II.5 : Instructions for Message call[23]

7.7. Ethereum networks:

Since Ethereum is a protocol, this means there can be multiple independent "networks" conforming to this protocol that do not interact with each other.[34]

Networks are different Ethereum environments we can access for development, testing, or production use cases. The Ethereum account will work across the different networks but the account balance and transaction history won't carry over from the main Ethereum network. For testing purposes, it's useful to know which networks are available and how to get testnet ETH.[34]

Public networks :

Public networks are accessible to anyone in the world with an internet connection. Anyone can read or create transactions on a public blockchain and validate the transactions being executed. Agreement on transactions and the state of the network is decided by a consensus of peers.[34]

- **Mainnet:** Mainnet is the primary public Ethereum production blockchain, where actual-value transactions occur on the distributed ledger.
- **Testnets:** These are networks used by protocol developers or smart contract developers to test both protocol upgrades as well as potential smart contracts in a production-like environment before deployment to mainnet.
- **Testnet Faucets:** ETH on testnets has no real value; therefore, there are no markets for testnet ETH. Since you need ETH to actually interact with Ethereum, most people get testnet ETH from faucets. Most faucets are webapps where you can input an address which you request ETH to be sent to.

Private networks :

An Ethereum network is a private network if its nodes are not connected to a public network (i.e. mainnet or a testnet). In this context, private only means reserved or isolated, rather than protected or secure.[34]

- **Development networks:** To develop an Ethereum application, you'll want to run it on a private network to see how it works before deploying it. Similar to how you create a local server on your computer for web development, you can create a local blockchain instance to test your dapp. This allows for much faster iteration than a public testnet.[34]
- **Consortium networks:** The consensus process is controlled by a pre-defined set of nodes that are trusted. For example, a private network of known academic institutions that each govern a single node, and blocks are validated by a threshold of signatories within the network.[34]

8. Conclusion:

We covered the fundamental concepts of Ethereum in this chapter. Of course, Ethereum is capable for a lot more than just the creation and transfer of the ETH money. The smart contract, which we'll look at in the following chapter, is at the heart of Ethereum. In the next chapter we will study smart contracts and the fundamental concept behind this thechnology.

Chapter III: Smart contracts

1. Introduction :

Bitcoin was invented to replace banks, but blockchain proved that it could replace almost any intermediary. It did not stop there, now that we had digital money, we could do something paper cash never could; program the money. Suddenly, we could replace lawyers and contracts in financial transactions. One of the most exciting topics to emerge in the blockchain ecosystem are smart contracts.

In recent years, the word ‘smart’ has increasingly been used to describe digitally enhanced objects.‘Smartphone’ is the most obvious example, but there are many others including ‘smart city’, ‘smart bulb’, ‘smart refrigerator’, ‘smart camera’ and so on. What links all of these developments is the technological advancements they deliver. With this in mind, it would be logical to assume that a smart contract is simply a digitally enhanced form of a conventional contract. While that is partly correct, it is only part of the story.[35]

1.1. Definition 1 :

Smart contracts are small, event-triggered programs that run in a trustless environment on a decentralized P2P network. They may be self-sufficient in offering a service or may be part of a decentralized application. In the latter scenario, they implement trust-related parts of the application logic like the exchange of assets, while the off-chain frontend interacts with users and other applications. Frequently, Smart contracts extend the concept of cryptocurrencies by implementing tokens as a special purpose currency.[36]

1.2. Definition 2 :

One of the most exciting topics to emerge in the blockchain ecosystem is the smart contract. A smart contract is a computer program whose code is stored in a distributed blockchain structure and that directly controls digital assets without relying on a third-party intermediary.[6]

Smart contracts can facilitate or even fully automate insurance processes, financial instruments, and legal processes that are currently heavily paper-based, long-winded, and expensive.[6]

2. Background :

The concept called ‘smart contracts’ is not entirely new. It was first given its name in the 1990s by computer scientist and legal theorist Nick Szabo. Szabo explained that the humble vending machine is a *"canonical real-life example, which we might consider to be the primitive ancestor of smart contracts"*. The mechanism of a vending machine is based on *if-then* logic, in which payment triggers irrevocable actions where money is retained and an item is supplied. Once this set of actions is triggered by the insertion of the coin, it cannot be stopped or reversed. Performance occurs automatically, without human intervention. This sequence of events is pre-programmed and embedded into a code of a vending machine.

In Szabo’s own words *"A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises."*

Due to technological limitations, smart contracts have been out of the spotlight for some time. The emergence of blockchain technology brought them back from obscurity and into the mainstream of technological advancement. Blockchain has enabled the progress of smart contracts from simple automated contracts to fully autonomous self-executing and self-enforced contracts built on decentralised platforms and supported by a blockchain ecosystem.

Bitcoin supports a set of scripts that enable the auto-enforcement of some special financial affairs other than direct electronic cash exchange. This procedure can be considered as the prototype of smart contracts. The Bitcoin scripts, on the other hand, are only usable in a few cases. Furthermore, because of its Forth-like, "old-style" nature, the script language is relatively difficult to learn for young programmers. The potential security issues (e.g., miscellaneous attacks) make this situation even worse. Despite the difficulty of creating smart contracts in Bitcoin, the scripts have been used to launch a number of decentralized applications, the most popular ones are data storage (evidence keeping), voting, gambling, and online poker games. Ethereum introduces a new virtual machine structure and supports Turing-complete programming languages, which greatly enrich smart contracts' functionalities. Specifically, Ethereum supports the execution of arbitrary deterministic computer programs in theory. The underlying Ethereum Virtual Machine (EVM) recognizes a low-level language called EVM bytecode. To reduce the learning cost and improve development efficiency, several high-level programming languages have been proposed, e.g., Solidity and Serpent, whose grammar is similar to mainstream programming languages.[37]

3. Smart contract devlopping platforms :

Smart contracts can be developed and deployed in different blockchain platforms. Each platform provides unique features for developing smart contracts, such as contract programming languages, contract code execution, and security levels.[38]

3.1. Bitcoin :

Bitcoin is a public blockchain platform that can be used to process cryptocurrency transactions, but with a very limited computing capability. Bitcoin uses a stack-based bytecode scripting language. The ability to create a smart contract with rich logic using the Bitcoin scripting language is very limited. Major changes would need to be made to both the mining functions and the mining incentivization schemes to enable smart contracts proper on Bitcoin's blockchain.[38]

3.2. NXT :

NXT is an open-source blockchain platform that relies entirely on a proof-of-stake consensus protocol. It includes a selection of smart contracts that are currently living. However, it is not Turing-complete, meaning only the existing templates can be used and no personalized smart contract can be deployed.[38]

3.3. Ethereum :

Ethereum is the first blockchain platform for developing smart contracts. It supports advanced and customized smart contracts with the help of a Turing-complete virtual machine, called the Ethereum virtual machine (EVM). EVM is the runtime environment for smart contracts, and every node in the Ethereum network runs an EVM implementation and executes the same instructions. Solidity, as a high-level programming language, is used to write smart contracts, and the contract code is compiled down to EVM bytecode and deployed on the blockchain for execution. Ethereum is currently the most popular development platform for smart contracts and can be used to design various kinds of decentralized applications (DApps) in several domains. Solidity is the well-known programming language used to write Ethereum smart contracts. For contract code execution, the contract code in Ethereum is included in a transaction, which is propagated in the peer-to-peer network, and any miner that receives this transaction can execute it in its local virtual machine.[38]

3.4. Hyperledger Fabric :

Rather than the public blockchain, such as Bitcoin and Ethereum that any party can participate in the network, Hyperledger Fabric is permissioned with only a collection of business-related organizations can join in through a membership service provider, and its network is built up from the peers whose are owned and contributed by those organizations. Hyperledger Fabric is an open-source enterprise-grade distributed ledger technology platform, proposed by IBM and supports smart contracts. It offers modularity and versatility for a broad set of industry use cases. The modular architecture for Hyperledger Fabric accommodates the diversity of enterprise use cases through plug and play components.[38]

Hyperledger Fabric supports multi-language smart contracts, such as Go, Java, and Javascript. In Hyperledger Fabric, when a transaction is created by the application, the transaction is only executed and signed by specified peers (endorsing peers). After receiving the application's transaction proposal, each of these endorsing peers independently executes it by invoking the chain-code to which the transaction refers.[38]

4. Caracteristic of smart contracts :

4.1. Automation and elimination of intermediaries

Smart contracts automate digital transactions. The blockchain ensures the necessary integrity and reliability of the underlying data and the proper execution of the smart contract, so that in the best case the parties neither know nor trust each other.[39]
This means that counter-performance risks and transaction costs can be reduced significantly, since transactions must be carried out directly and not monitored by third parties (such as lawyers, banks).[39]

4.2. Transparency and legal certainty :

Since the transactions to and from smart contracts are permanently logged in the blockchain, subsequent manipulation of account balances and register entries is impossible.[39]
An additional gain in transparency is possible on public blockchains, since all transactions can be traced and checked here. In addition, there can be a gain in legal certainty if smart contracts are created unchangeably and users can rely on the unaltered execution of transactions.[39]

4.3. Restriction to services in the blockchain :

Smart contracts can only provide services that can be digitally mapped in the blockchain. First and foremost, these are transactions and registry entries. Smart contracts seem to be particularly suitable for simple legal enforcement, such as the collection of payment and the corresponding release or blocking of the thing.[39]

Because of their digital nature, they can be easily standardized and duplicated. In particular, areas in which many contracts with similar or even identical content are used can be designed more efficiently and cost-effectively.[39]

4.4. Restriction to digitally verifiable events :

Smart contracts can only map digitally verifiable events, i.e. those that move within the binary logic of success. They regularly do not have a “smart” element in the sense of artificial intelligence or machine learning. Complex test tasks that go beyond pure true / false consideration can therefore currently not be mapped by smart contracts. Although this software is called "smart", it is no smarter than other software in this regard.[39]

4.5. Anonymity :

The use of smart contracts is basically anonymous. Since smart contracts have their own public key, users communicate directly with them in order to obtain the desired service. Because this is in the decentralized network of the blockchain and not on the server of the creator, only the anonymous transaction data (public key, date, amount) are saved when using the smart contract. However, some services now offer to identify users based on the transaction data.[39]

In principle, the parties have to disclose their identities “manually” if they are certainly interested in knowing with whom they are exchanging services. If they do not do this, any further legal prosecution becomes practically impossible, as there is hardly any possibility of assigning the encrypted public key to a specific person due to the lack of a central network instance. Smart contracts are therefore more suitable where there is either a low risk of poor performance and the identity of the parties is therefore not important, or where the contracting parties already know each other.[39]

5. Operational process of smart contracts:

Smart contracts enable automated execution of decisions based on the data purely. This means that based on the data flowing through the business logic defined in the smart contract, various clauses/actions are triggered. Thereby reducing manual biases of over the data & functions on the pre-defined set of conditions. To be clearer, a delay in payments must cause the addition of interest, which is usually accounted for in large-scale financial transactions or financial institutions. However, there are millions of freelancers who receive delayed payments with no interest. Now, instead of a paper contract, if a freelancer were to engage in business via a smart contract that was programmed to cover the addition of interest for delayed payments from the commissioning company, the effort and delay would be covered aptly. Conversely, if the smart contract were pre-programmed and agreed upon for a timely delivery of the project and the freelancer failed to deliver on time, penalties would be auto-calculated on the ledger, making it a fair programmatic agreement.[40]

For instance, the smart contract can define the constructor function that enables the smart contract creation. Invoking the constructor function via a transaction, whose sender becomes the smart contract owner, allows a new smart contract to be hosted on the blockchain. A self-destruct function is another example of the functions that can be defined in a smart contract. Usually, only the smart contract owner can destruct the contract by invoking this function.[38]

A smart contract is likely to be a class that includes state variables, functions, function modifiers, events, and structures which is intended to execute and control relevant events and actions according to the contract terms. Besides, it can even call other smart contracts. Each smart contract includes states and functions. The former are variables that hold some data or the owner's wallet address (i.e., the address in which the smart contract is deployed). We can distinguish between two state types, namely *constant states*, which can never be changed, and *writable states*, which save states in the blockchain. The latter are pieces of code that can read or modify states. We can distinguish between two function types, namely *read-only functions*, which do not require *gas* to run and *write functions* that require *gas* because the state transitions must be encoded in a new block of the blockchain. Furthermore, paying currency is required to avoid infinitely smart contract runs.[38]

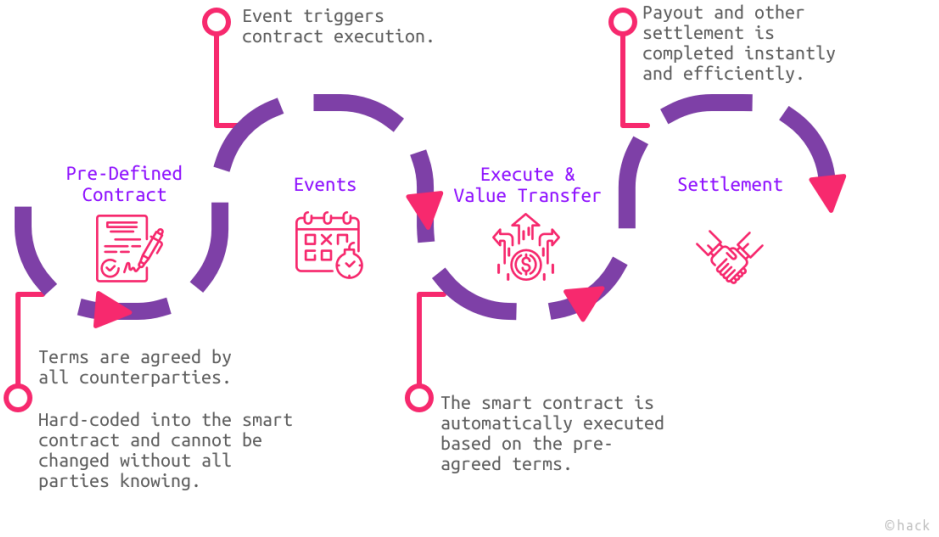

Figure III.1 : Operational process of smart contracts[41]

6. Benefits of smart contracts :

The main benefit of smart contracts is the simplification of certain processes through removing the human component from them and potentially one or several third-parties facilitating the operation. Although not as well-suited for complex contracts at the current stage of development, the codification of some relatively simple contractual relations as smart contracts has the potential to eliminate many errors caused by the human component.

Smart contracts also results in the added benefit of transparency, as automation of business processes eliminates many avenues of corruption, and the transparency of a public blockchain and smart contract platform such as Ethereum enables any interested parties to audit any transactions on the part of a smart contract.

7. Smart contract vulnerabilities :

The most serious threat to smart contracts is their software flaws. Nicola et al. conducted a systematic investigation on Ethereum smart contract attacks. Loi et al. created Oyente, a symbolic execution program that found four types of security flaws.[42] :

- The reentrancy vulnerability results from the fact that when an Ethereum smart contract invokes another one, the current execution will wait until the invocation finishes. If the callee is malicious, it could exploit the intermediate state of the caller to launch attacks. Such vulnerability has led to 60 million USD worth of Ether theft.
- The mishandled exceptions may happen when one smart contract calls another one. In particular, if the exception in the callee is not propagated to the caller or the caller

does not take care of the return value from the callee, the execution logic of caller will be affected.

- The transaction-ordering dependence refers to the potential attacks resulting from the unexpected order of transactions. Note that the miner that mines the block can determine the order of the transactions.
- The timestamp dependence refers to the potential vulnerability in smart contracts that use the block timestamp to control the execution of some important operations. A malicious miner may change the block timestamp to affect the execution of those important operations.

Smart contracts can also be used by attackers to undertake nefarious operations and even to attack the Ethereum infrastructure. According to Juels et al., criminal smart contracts can ease the leakage of personal data, the theft of secret keys, and a variety of "calling-card" crimes such as murder, terrorism, and so on. Malicious smart contracts have also been used to launch denial-of-service (DoS) attacks on the Ethereum platform.[42]

8. Oracles :

The network participants (nodes) validate and execute operations performed on the blockchain, such as smart contracts, but it is not uncommon for a smart contract to require data from external third parties. Given that blockchains cannot access data outside their networks, how is external data incorporated into the workflow ? Here is where Oracles come in.[6]

8.1. Definition :

Oracles are agents on the blockchain that can verify events from the real world and then provide the corresponding information as data to the smart contracts. Running on the blockchain, Oracles act as intermediaries between external data and smart contracts that also run on the blockchain. Smart contracts usually specify only sequences like "If condition *C* has occurred, operation *O* will be executed." To check whether the condition has been met, smart contracts often rely on external data. One example is sensors that measure real-world phenomena. For example, if you want to implement insurance through a smart contract that pays direct compensation when certain temperatures are reached, an Oracle must provide the temperature information.[6]

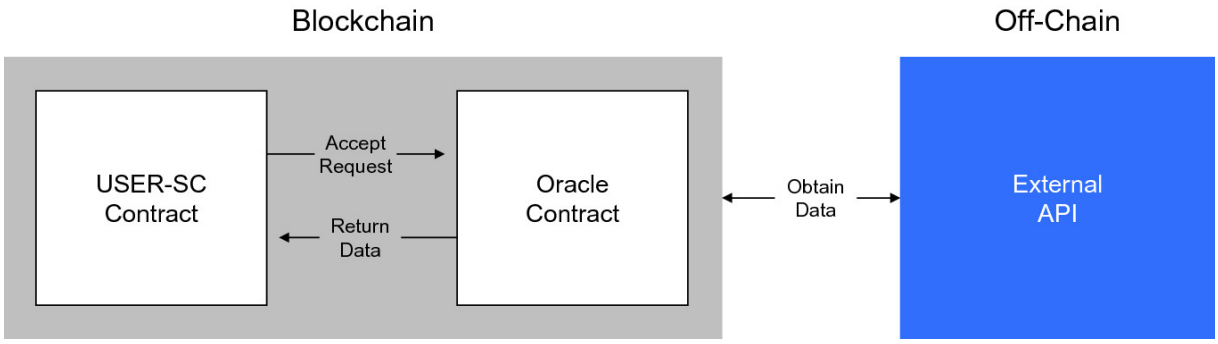

Figure III.2 : Communication between smart contracts and oracles[6]

The addition of an Oracle occurs through multi-signature (MultiSig) contracts, which require that an invocation of a smart contract method be signed by multiple parties. To improve trust in the external data a smart contract that an Oracle provides and to prevent the Oracle from being compromised by one party, data can be signed by multiple parties.

The basic principle of blockchain is that a consensus is reached through the involvement of multiple parties, eliminating the need to trust a single party. However, if the data is provided by a central authority like a bank, the data-providing authority must be trusted.[6]

8.2. Types of Oracles :

Oracles are divided into five categories, as well as by their functional setup, which is based on data flow and whether or not they are consensus-based.[6]

- **Software Oracle :** Data is available online and is, ideally, provided from multiple independent sources, has a public record history (e.g., air traffic information, meteorological data), and is signed by multiple parties.
- **Hardware Oracle** : Data is from real-world measurements like RFID sensors in a supply chain site. Challenges with this type of data source include that there is usually only one source, there is no public record history, it is signed by only one party, and transmitting the information in a secure and provably immutable way is difficult.
- **Inbound Oracles** : These Oracles provide a smart contract with information from the outside world. For example, a buy order is set to be executed as soon as the EUR-USD exchange rate falls below a certain limit.
- **Outbound Oracles**: These Oracles enable smart contracts to send data to the outside instead of just receiving it. For example, access to an area may be granted when a payment has been made on the blockchain (e.g., via a smart lock).

- **Consensus-based Oracles** : Several Oracles are combined so one does not have to rely on a single external source. These Oracles then form a consensus to make decisions. An example is a consensus Oracle that specifies that three out of five Oracles must agree before an operation is executed. Of course, it is also possible to apply scores to individual Oracles such that a source carries more weight if it appears to be more reliable than other sources.

9. Gas :

The developers of the EVM implemented a Turing-complete instruction set-up, which is possible because every instruction that the virtual machine executes has a price. In the Ethereum ecosystem, the cost of running a smart contract code is measured in units called gas.[6]

Value	Mnemonic	Gas Used	Subset	Removed from stack	Added to stack	Notes
0x00	STOP	0	zero	0	0	Halts execution.
0x01	ADD	3	very low	2	1	Addition operation
0x02	MUL	5	low	2	1	Multiplication operation
0x03	SUB	3	very low	2	1	Subtraction operation
0x04	DIV	5	low	2	1	Integer division operation
0x05	SDIV	5	low	2	1	Signed integer division operation (truncated)
0x06	MOD	5	low	2	1	Modulo remainder operation
0x07	SMOD	5	low	2	1	Signed modulo remainder operation
0x08	ADDMOD	8	mid	3	1	Modulo addition operation

Figure III.3 : Ethereum instructions and gas costs[6]

The EVM's individual programming instructions each have their own gas cost, so when an originator launches a smart contract, she must always set the gas limit to indicate the maximum gas that the smart contract may consume during its execution.[6]

Ideally, the program terminates before the gas limit is reached, and the originator pays exactly the gas the smart contract consumes. However, if all the gas is used up, either because the program gets stuck in a loop or the gas limit is too low, the smart contract terminates. In such cases, a correctly programmed smart contract will terminate without making any changes, while a poorly programmed smart contract may yield unexpected results, such as becoming permanently inaccessible or allowing fraudulent use.[6]

Ethereum-based smart contracts can be thought of as transactions with extensive rules and conditions. Gas consumption is higher for a complex smart contract than it is for a simpler one, so the gas limit should be higher to prevent the contract from running out of gas before it fully executes all the instructions contained in the smart contract code. Gas is awarded to the miner who successfully packs the transaction into a block and executes its code.[6]

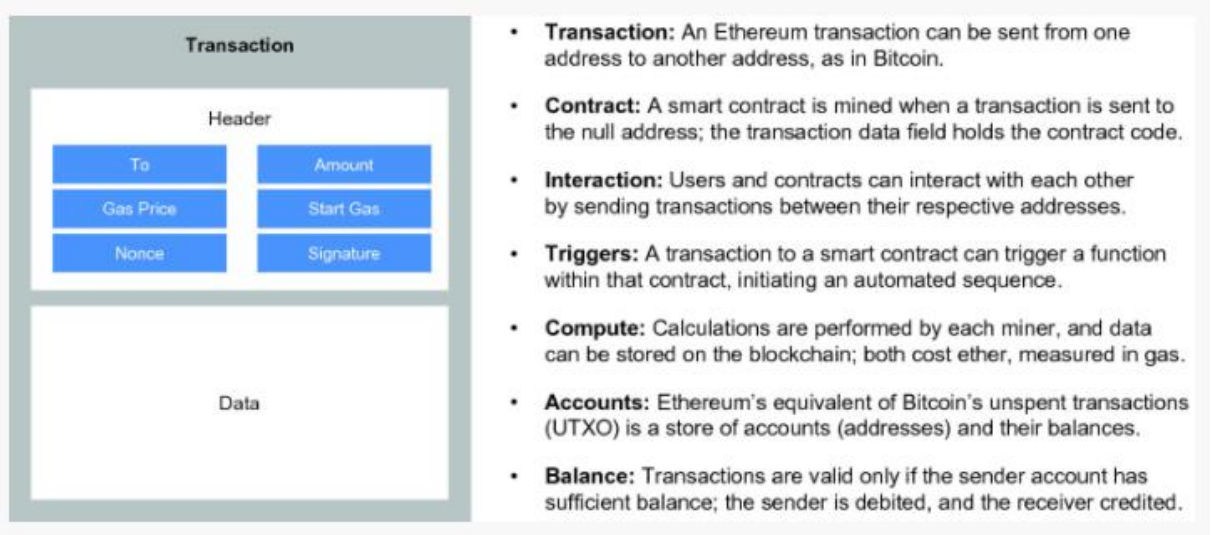

Figure III.4 : Representation of a smart contract as a transaction [6]

In addition to the gas limit, the originator specifies the gas price for the transaction, which sets how much Ether she is willing to pay per unit of gas. Thus, a market similar to transaction costs with Bitcoin emerges, where greater payments lead to faster transaction times. If the network is busy, the average price of gas rises as miners select transactions that bring in more Ether.[6]

The gas price is usually in the range of a billionth of an ether. A billionth (10^{-9}) of an ether is also called a gigawei (GW) or simply gwei, where 1 wei is 10^{-18} ethers.[6]

Consider the following example: We carry out a transaction that consumes 1475 gas and set the gas price at 17 GW. Thus, if we choose a gas limit of 1475 or more, we will pay $17 \times 10^{-9} \times 1475 = 0.000025075$ ethers for the transaction. If we set the gas limit to less than 1475, the transaction will not be completed, but we will still have to pay the miner for trying to execute the transaction before running out of gas.[6]

10. Conclusion :

In this chapter we presented smart contracts and covered the fundamental concepts behind it, as well as how to create and deploy our own smart contract. This technology has a large and flexible application framework that will expand with the rapid advances in technology.

In the next chapter we will concept and implement an online voting platform using blockchain technology, the ethereum platform and a smart contract.

Chapter IV: Conception and Implementation

1. Introduction :

We discussed the numerous elements for the realization of our voting platform in the previous chapters; in this chapter, we will explain the technical aspects of the realization of our web application, as well as demonstrate the functionalities through our web application.

2. Problematic :

The first voting system that comes to mind is the traditional ballot box, where each voter casts his or her ballot in a ballot box. Both the voting and the envelope are designed to ensure the confidentiality of the vote. The transparent ballot box and the public counting of votes allow everyone to check the count.

Despite the fact that traditional elections require a significant investment of human, material, and financial resources, the procedures are difficult and unpleasant for citizens in terms of time and effort. Furthermore, because this approach imposes intermediary authority that can fake the outcome, neither the traceability, integrity, nor transparency of the vote calculation can be guaranteed, and this is where the electronic voting comes to the scene.

Electronic voting is the use of computer and telecommunications technology in the voting process. A device (computer, telephone) is used to cast one's vote and to automatically count all the votes. The benefits are to facilitate voting and thereby increase participation in elections, while making the results immediate. Voting everywhere, as long as you vote, would therefore be the key to success, to participation in elections.

On the contrary, critics of electronic voting point out that it is unverifiable and the cure could be worse than the disease, improving the security of the ballot would require voters to be identified by name to ensure that their vote corresponds to their wishes, thus making the citizen's choice traceable and undermining the secrecy of the vote.

Others point to the complexity of organising elections with electronic voting. Securing the remote ballot, for example, requires personalised codes or passwords to be sent by post before using the platform.

Above all, the difficulty in establishing that there has been fraud or manipulation has the main consequence of making the concrete control of the election very delicate.

There are two main security flaws in electronic voting: the possibility of error or breakdown coming from the system itself or a malicious attack. Thus, the risks of hacking, particularly from abroad, regularly come up in the public debate.

3. Solution :

The electronic voting still isn't a reliable solution to give up on traditional voting, because of the risks of hacking and fraud or manipulation, blockchain technology can be used to eliminate such problems.

Blockchain, in its most basic form, is a digital ledger. To verify, process, and record all transactions across the system, the technology requires power from the peers or nodes on its network. This ledger is never saved; instead, it resides on the "chain," which is supported by millions of nodes at the same time. Blockchain's transaction database is incorruptible due to encryption and decentralization. and each record is easily verifiable. The network cannot be taken down or influenced by a single party because it doesn't exist in one place.[43]

Our solution requires us to develop a decentralized application using a smart in a contract and will achieve these goals:

- Checks and verifications at the time of registration.
- Limit the number of registrations to 1 for each user.
- Limit the number of votes to one for each voter.
- Calculate votes in real time.
- Limit the voting process with a time interval.

For this project we deployed our smart contract in a private blockchain network using Ganache to main purpose was to deploy a local blockchain network, another functionality of ganache is to provide us with multiple Ethereum wallets with an existing balance in them, we also used Truffle frame work to deploy our smart contract and Metamask that's will act as a bridge between our browser and the blockchain network and to also control our wallets

4. The development process used (Agile) :

Any agile methodology provides for the splitting of software development stages. Contrary to the traditional method which foresees the total planning of the project even before its development, the Agile Manifesto advocates instead the setting of short-term objectives. The project is thus fragmented into several sub-parts that the team in charge of it must achieve

progressively, readjusting the objectives if necessary to meet the client's expectations as much as possible.

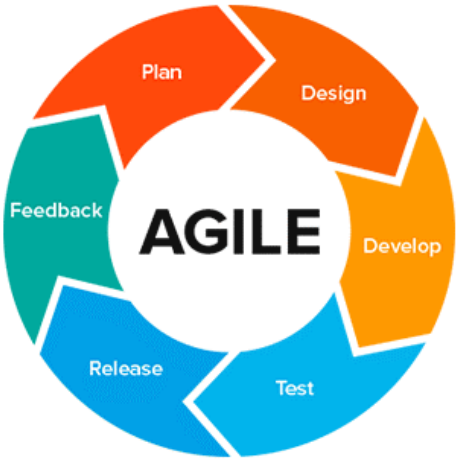

Figure IV.1 : Agile development methodology cycle

5. Conception :

5.1. Use case diagram :

General use case diagram :

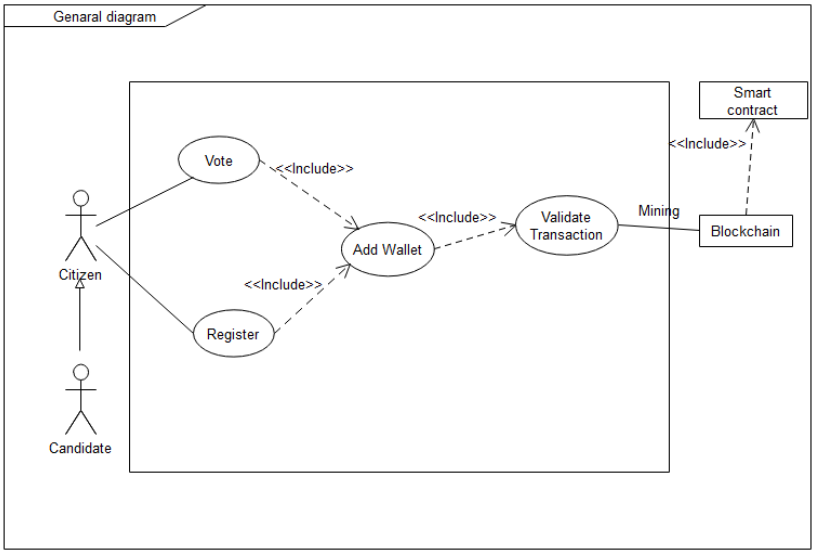

Figure IV.2: General use case diagram

It is required for all users (candidates and citizens) to login to their respective Ethereum accounts via Metamask to be able to connect to the blockchain network.

Candidates are alredy determided after deploying the smart contract, this is done to prevent adding candidates in the middle of the election, in the beginning Citizens (including candidates)

have to sign up by filling a form, we will be able to distinguish users from each other using their unique national ID number. After registration, each ID number will be linked to the ethereum wallet that the user has in their MetaMask account, the user will not be able to add any other wallet so that each user can only have one account for each ID.

Once on the voting page, the user can choose one of the candidate and vote, once that the user has voted, the voting option will not be available anymore for this user, the acces to the voting page will always be availabe for all the users since its also considered as a voting result consulting page, the difference will be that the users that voted will not have a voting option anymore

Inscription use case diagram :

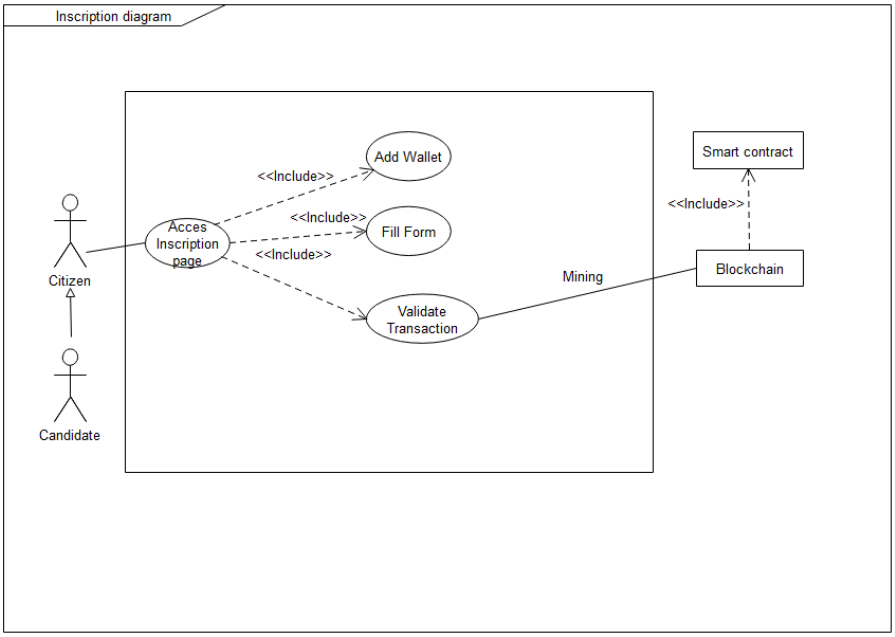

Figure IV.3: Inscription use case diagram

To sign up, citizens must login to their MetaMask accounts and fill the form provided on the web page, the inscription will fail if the user already has an account or if any information is invalid. Once form is filled correctly, the user has to validate the transaction to complete his registration.

Vote use case diagram :

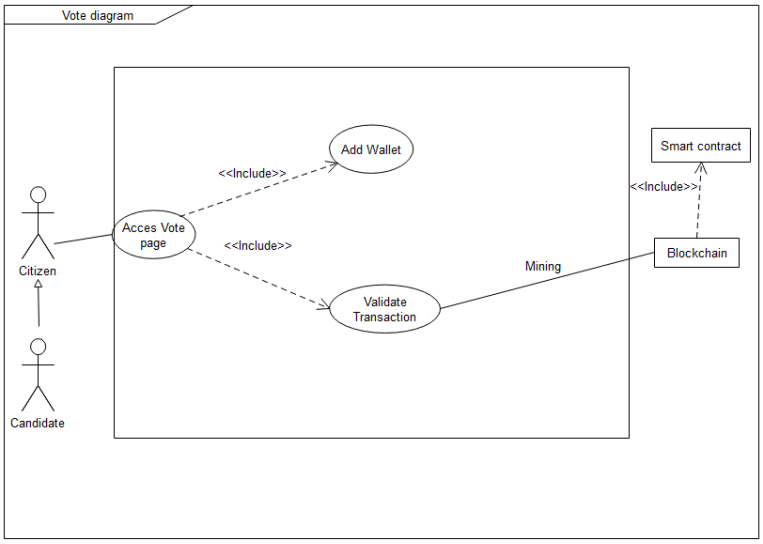

Figure IV.4 : Vote use case diagram

To vote, citizens must have already signed up and logged in to their Metamask account, once on the voting page the user can choose his favorite candidate and vote for him then the user must confirm the transaction, once that the user has voted,the voting option will not be available anymore but he will still be able to acces the voting page to consult the election results.

5.2. Class diagram:

Candidates are added in the smart contract its self, each candidate has a votecount attribute to calculate the number of votes that this candidate managed to receive, adding candidates from outside the smart contract is impossible but candidates are also considered users,when registering they will register exactly as any other user.

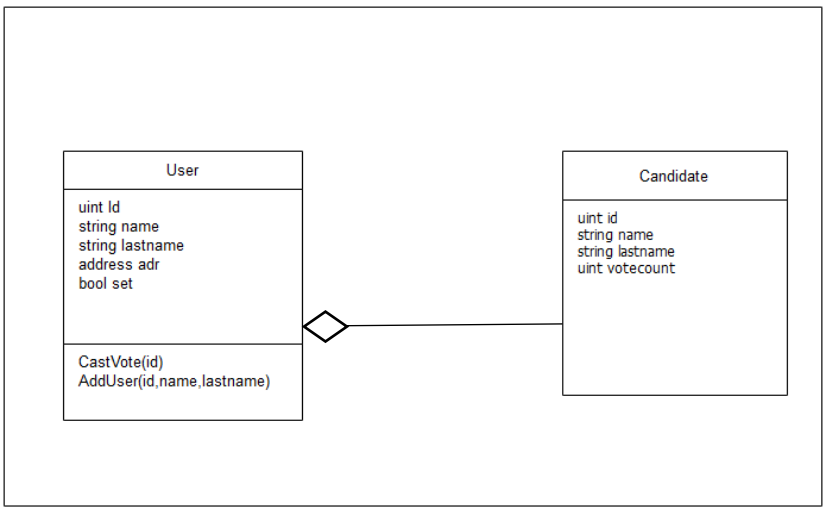

Figure IV.5: Class diagram

In our project we used the national ID number and the wallet addresses as our main information for authentidifaction, we used multiple mappings as a verification methode, the mapping of the users using the ID and the mapping for the addresses that are already registered, using mappings is the fastest and most secured way to store data on a smart contract.

Solidity mapping might look similar to an associative array at first glance but its not. It doesn't have indices, making it hard to loop through all addresses.

To track users that have voted, we set a boolean attribute in the user structure we also track the addresses that have voted by using an address mapping.

5.3. Sequance diagrams :

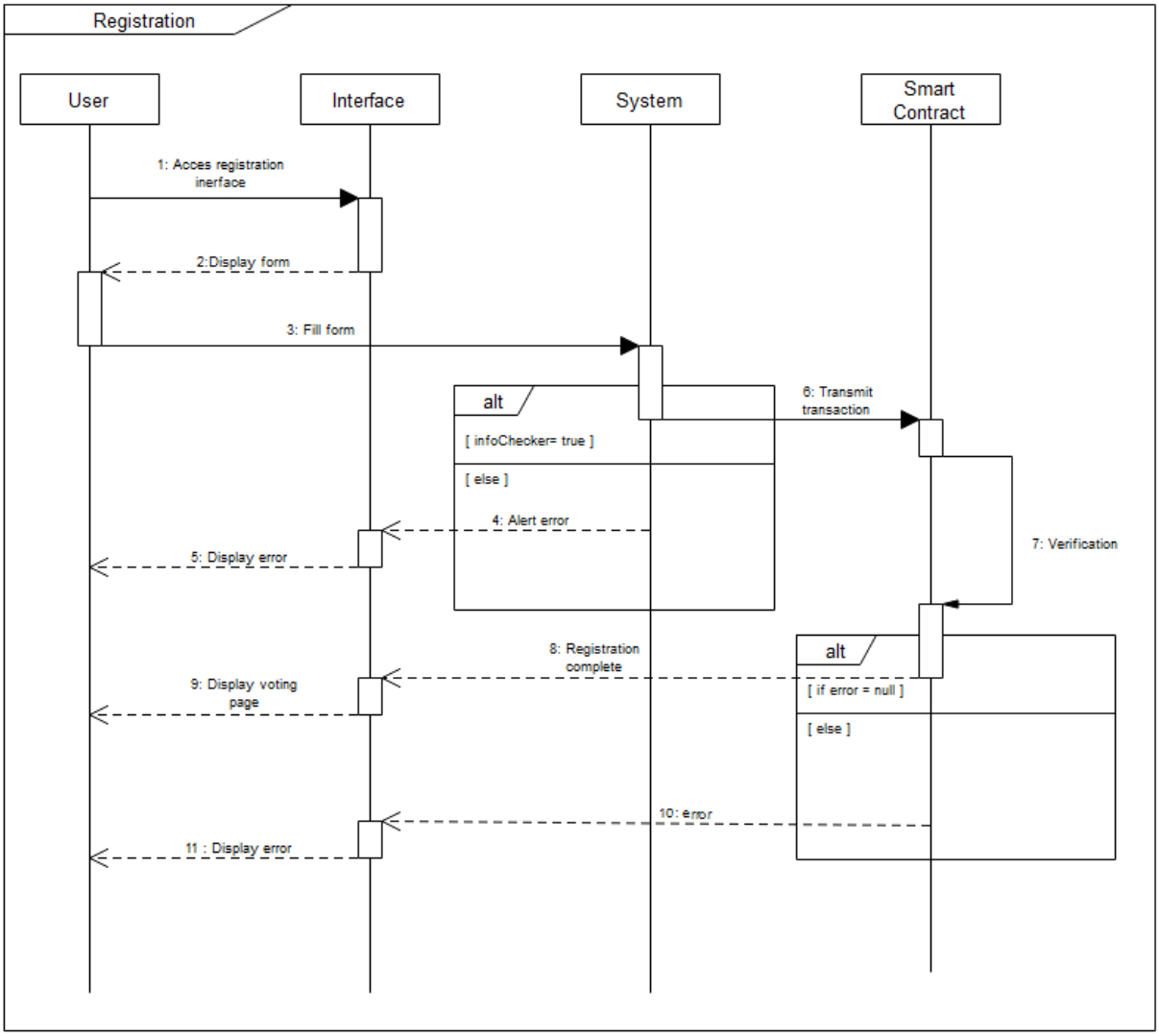

Figure IV.6 : Registration sequance diagram

The registration sequence diagram shows the sequence of interaction between the user and our smart contract, in this diagram the user accesses the registration interface and must connect to metamask and fill in a form, after validating the data and confirming the transaction, the system will check if the data entered is correct, if there is no problem the system will transmit the transaction to our smart contract and then the smart contract will also check if this information already exists. An alert message will be displayed in case of success or failure.

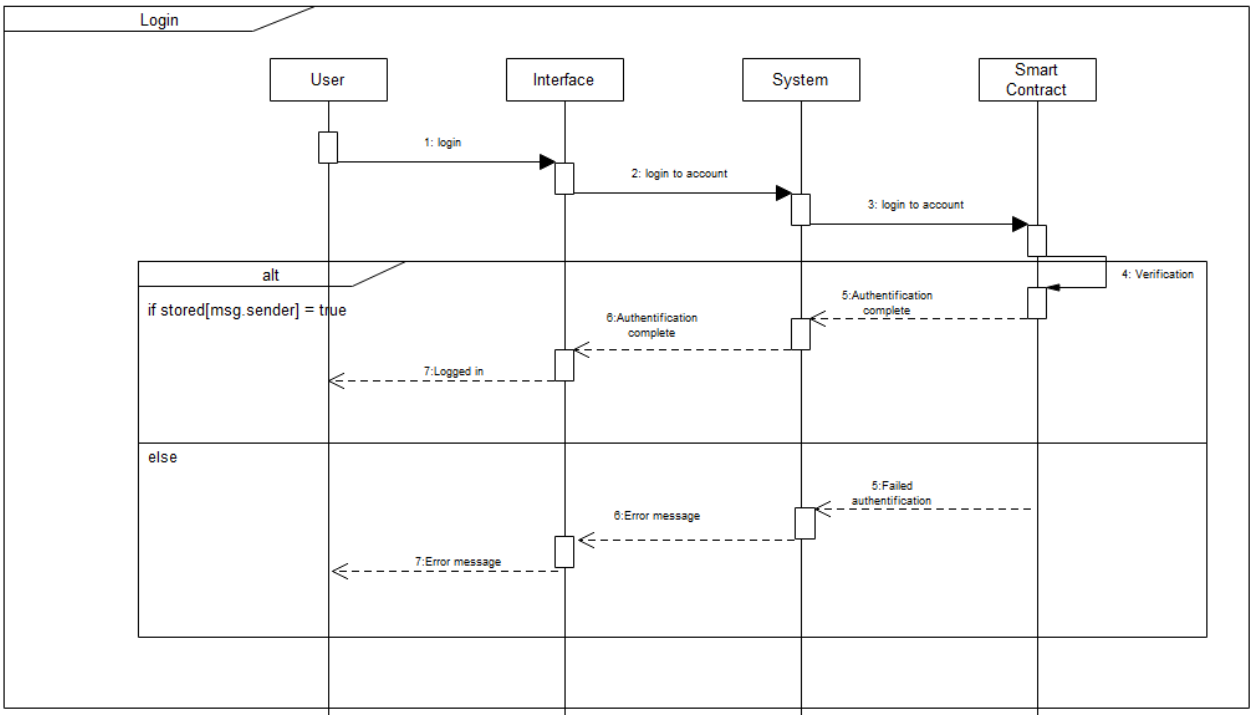

Figure IV.7: Login sequance diagram

The registration sequence diagram shows the sequence of interaction between the user and our smart contract, in this diagram the user accesses the welcome interface and must connect to metamask, once logged in metamask the user will have to login to his account, if the account has already been registered the user will be redirected to the voting page else an error message will appear.

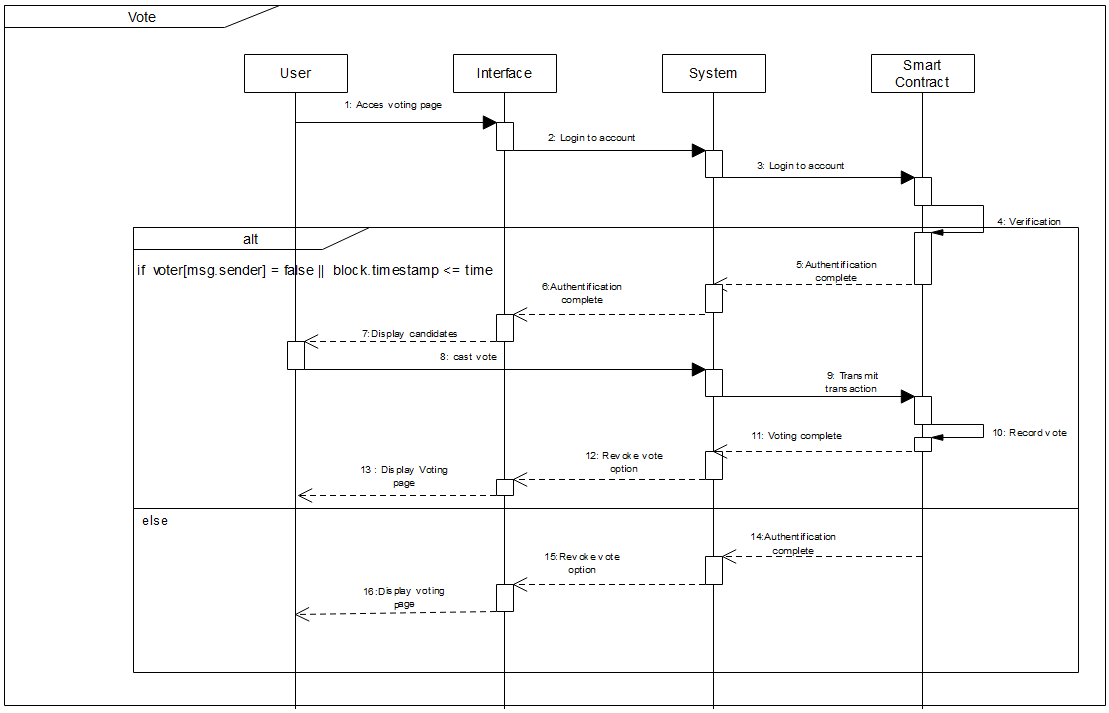

Figure IV.8 : Vote sequance diagram

The voting sequence diagram shows the sequence of interaction between the user and our smart contract, in this diagram the user will be able to see the candidates and also vote on one of these candidates, if the user hasn't registered before he will not have acces to this page. If the user has already voted before, the voting option will be revoked and only the election result well be displayed.

If the user hasn't voted before, he will have to choose a candidate and vote for him, once the user presses the vote button and confirmed the transaction the system will transmit the transaction to our smart contract,record the vote and revoke the voting option from the user.

6. Implementation :

The development of a system is a key step in the development of any project. In this section we will present the development tools used and the functionalities and interferences of our program.

6.1. Development tools :

NodeJs :

Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript, and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Node.js also provides a rich library of various JavaScript modules which simplifies the development of web applications using Node.js to a great extent [44]. To install nodeJs we need to download and install the nodeJs installer.

Truffle:

Truffle is a development environment, testing framework, and asset pipeline all rolled into one. It is based on Ethereum Blockchain and is designed to facilitate the smooth and seamless development of Distributed Applications. With Truffle, we can compile and deploy Smart Contracts, inject them into web apps, and also develop front-end for DApps.[45]

To install truffle we will the following command :

```
npm install truffle -g
```

Figure IV.9 : Truffle instalation command

This command will install truffle globaly in our computer.

Truffle Ganache:

Ganache is a personal blockchain for rapid Ethereum and Corda distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.[46]

Ganache allows you to establish a private Ethereum blockchain on which you can run tests, execute commands, and observe state while maintaining control over how the chain runs. It allows you to conduct any actions that you would on the main chain but at a lower cost. Many

developers use this to test their smart contracts as they are being developed. It provides convenient tools such as advanced mining controls and a built-in block explorer. Ganache can be installed by downloading and installing the ganache installer.

Web3.js:

The web3.js is a collection of libraries that allow you to interact with a local or remote Ethereum node, using HTTP, WebSocket, or IPC. Through the web3.js APIs, the front end can then interact with the Smart Contracts. [24]

6.2. Development approach:

Before explaining the functions used on our smart contract, we will present how this smart contract was deployed in the first place.

First, we need to run a blockchain server on our computer using Ganache, in our example we used the UI version of Ganache:

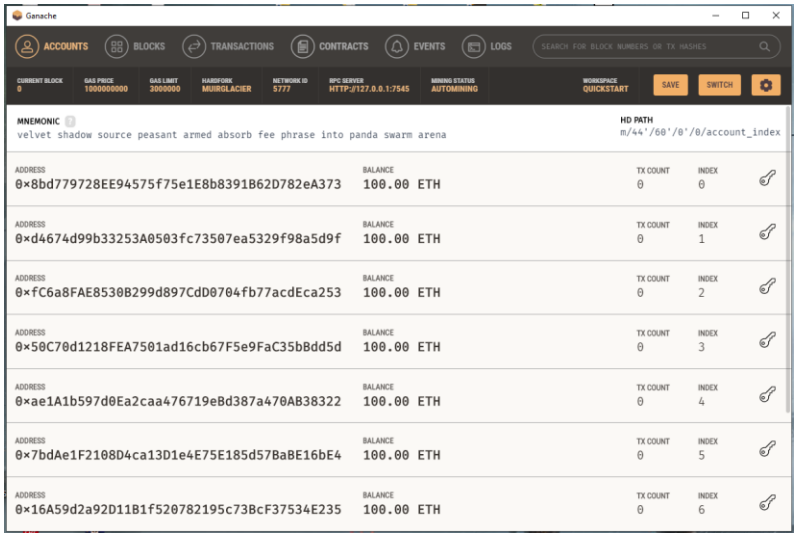

Figure IV.10 : Ethereum account created by ganache

Ganache will provide us with 10 Ethereum accounts by default, each account has 100 ETH on it.

Now, we will have to deploy our smart contract using the command *truffle migrate.* Once the smart contract deployed, we will have this output:

```
Replacing 'Election'
--------------------
> transaction hash:    0x7bc0d58d28aacb13f6f292063ef1c049c4b18f9767de21e3fe24f261917fd15e
> Blocks: 0            Seconds: 0
> contract address:    0xa20f5Fe56eFe18AadA5aa5814e1564DB744D7A0F
> block number:        3
> block timestamp:     1623495708
> account:             0x8bd779728EE94575f75e1E8b8391B62D782eA373
> balance:             99.98234941
> gas used:            1530778 (0x175b9a)
> gas price:           10 gwei
> value sent:          0 ETH
> total cost:          0.01530778 ETH


> Saving migration to chain.
> Saving artifacts
-------------------------------------
> Total cost:          0.01530778 ETH
```

Figure IV.11 : Smart contract migration result

Our smart contract will by default use the first account provided by ganache as our smart contract address and to pay for the transaction for the deployment of our smart contract.

ADDRESS	BALANCE	TX COUNT	INDEX
0×8bd779728EE94575f75e1E8b8391B62D782eA373	99.98 ETH	4	0

Figure IV.12 : Ethereum address used to create the smart contract

In this project we chose to use the Ethereum technology to develop a smart contract. Smart contracts are essentially programs that run when certain criteria are satisfied and are maintained on a blockchain. They're often used to automate the execution of an agreement so that all parties can be certain of the conclusion right away, without the need for any intermediaries or time waste. They can also automate a workflow, starting the following step when certain conditions are satisfied.[47]

Election.sol:

In our code we have defined two structures, one for candidates and one for users:

```
struct Candidate{
    uint id;
    string name;
    string lastname;
    uint votecount;

}
```

Figure IV.13 : Candidate Structure

```
struct Users{
    uint Id;
    string name;
    string lastname;
    address adr;
    bool set;
}
```

Figure IV.14 : User structure

To add the candidates, we have defined a private function that will only be accessible inside the smart contract where the call to this function will be made:

```
function AddCandidate(string memory _nom,string memory _prenom) private{
    candidateCount++;
    candidates[candidateCount]=Candidate(candidateCount,_nom,_prenom,0);

}
```

Figure IV.15 : AddCandidate function

The function responsible for adding users:

```
function AddUser(
  uint _Id,
  string memory _nom,
  string memory _prenom) public{
  require(IDCheck(_Id),"Id does not exist.");
  require(!stored[msg.sender], "Sender already stored.");


  require(!userslist[_Id].set,  "User already exists. ");
  stored[msg.sender] = true;
  userslist[_Id]=Users(_Id,_nom,_prenom,msg.sender,true);

  users[msg.sender]=Users(_Id,_nom,_prenom,msg.sender,true);
  emit NewUser(_Id);
}
```

Figure IV.16 : AddUser function

This function allows you to:

- Add users
- Check that the ID provided by the user exists
- Check if the user's address already exists.

- Check if the user has already been registered

When called, the function will first check if the ID number provided vas legitimate by searching for it in a database stored on our smart contract with the *IDCheck* function, then it will check if the wallet address is already stored, it will check if the user is already registered, lastly if all conditions are met, the function will create a new user and add him to the corresponding mappings.

The function checkUser is used when login in, when the user tries to access the voting page, the smart contract will check if the account exists in the stored addresses mapping, if the address exists the access will be granted.

```
function checkUser()public view returns (bool){
if(stored[msg.sender] == true){
  return  true;
}else{
  return false;}
}
```

Figure IV.17 : CheckUser function

The voting function also uses mappings, the function will first check if the voting period still hasn't passed, if the user's wallet address is stored, check if this address has already voted, if the time limit for the election has been passed and as an addition it will check if the candidate ID provided from the web page is also correct. Once the conditions are met, the function will mark that the address has voted and add to the vote count to the corresponding candidate.

This function is immune to the reentrancy attack known in solidity, since we are recording the vote only after we mark the voting account as an already voted account.

```
function vote(uint _id) public{
    require(block.timestamp <= time);
    require(stored[msg.sender]);
    require(!voter[msg.sender],"You already have voted ! ");
    require(_id>0 && _id <= candidateCount);

    voter[msg.sender]=true;

    candidates[_id].votecount++;

    emit votedEvent(_id);

}
```

Figure IV.18 : Vote function

The function ElectionEnd is the one responsible to monitor the end of the voting process to display the voting result, at the start of our contract we set a global variable *time* that receives the time at the moment of the deployment of our smart contract in seconds.

```
function ElectionEnd(uint256 _time)public view returns(bool){
        if (_time > time){
    return true;
  }
return false;
}
```

Figure IV.19 : ElectionEnd function

App.js:

Commancant par la function d'authentifications :

```
login: async function() {
  var err;

  if (window.ethereum) {
    await ethereum.enable();
  }
  // Load account data
  web3.eth.getCoinbase(function(err, account) {
    if (err === null) {
      App.account = account;
    }
  });
  var ch;
  ch = await App.contracts.Election.deployed().then(function(instance) {
    return instance.checkUser({
      from: App.account
    });
  });
  console.log(ch);
  if (ch == true) {

    window.location.href = "./index.html";
  } else {
    alert("This account does  not exist ! Please sign up or login to the correct account");
  }
},
```

Figure IV.20: Login function

The function will check if the browser is connected to the Ethereum network, get the MetaMask account address and check if the account exists by calling the checkUser function in our smart contract, if the conditions are met the user will be logged in.

The next function if the registration function:

```
signUp: async function() {
  if (window.ethereum) {
    await ethereum.enable();
  }
  // Load account data
  web3.eth.getCoinbase(function(err, account) {
    if (err === null) {
      App.account = account;
    }
  });
  if (App.infoChecker() == true) {
    App.contracts.Election.deployed().then(function(instance) {
      return instance.AddUser($('#SignId').val(), $('#SignName').val(), $('#SignLastName').val(), {
        from: App.account
      });
    }).then(function(result) {

    }).catch(function(err) {

    });

    App.ListenForSignup();

  } else {
    App.infoChecker();

  }
}
```

Figure IV.21 : SignUp function

The function will check if the browser is connected to the Ethereum network and get the MetaMask account address, the function will also check the validity of the form, the inputs of the form also have their own input control functions:

```
LettersOnly: function(input) {
  var regex = /[^A-Za-z]/gi;
  input.value = input.value.replace(regex, "");
},
NumbersOnly: function(input) {
  var regex = /[^0-9]/gi;
  input.value = input.value.replace(regex, "");
},
```

Figure IV.22 : Registration input control functions

If everything checks out, the registration function will call the AddUser function in our smart contract and add a new user.

Now the voting function, the function will send the selected candidate ID to our smart contract and will update the page with the new data.

```
castVote: function() {
  var candidateId = $('#candidatesSelect').val();
  App.contracts.Election.deployed().then(function(instance) {
    return instance.vote(candidateId, {
      from: App.account
    });
  }).then(function(result) {
    // Wait for votes to update
    $("#content").hide();
    $("#loader").show();
  }).catch(function(err) {
    console.error(err);
  });
},
```

Figure IV.23 : Castvote function

User interface:

On the welcome page, the user will be able to make a choice between logging in or signing up.

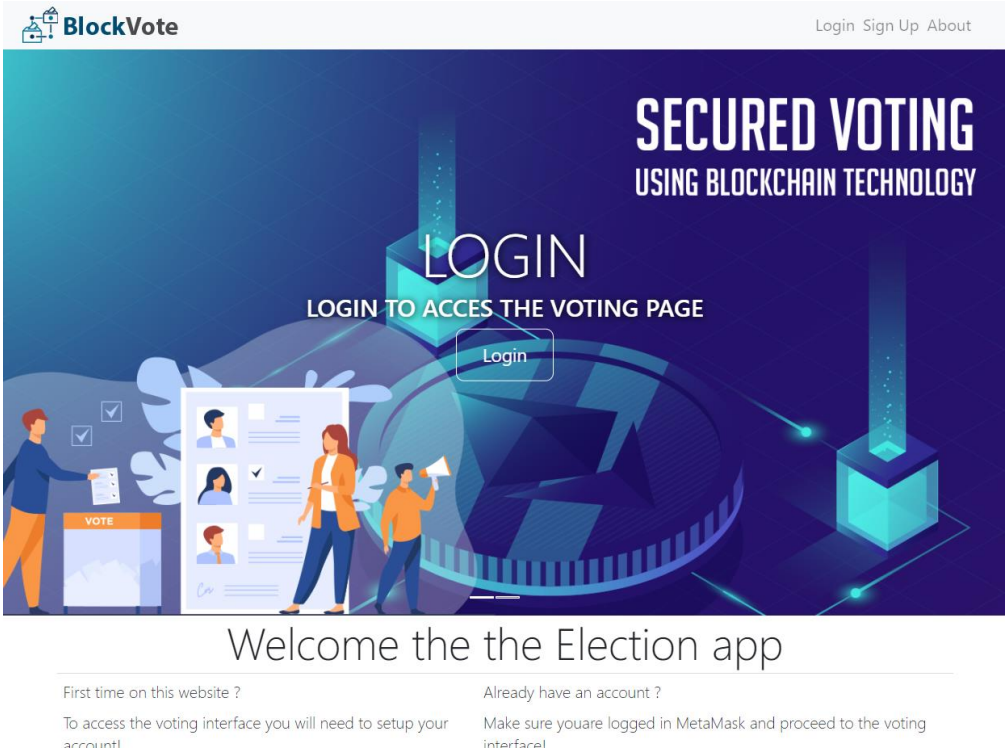

Figure IV.24 : Welcome page with the login slide active

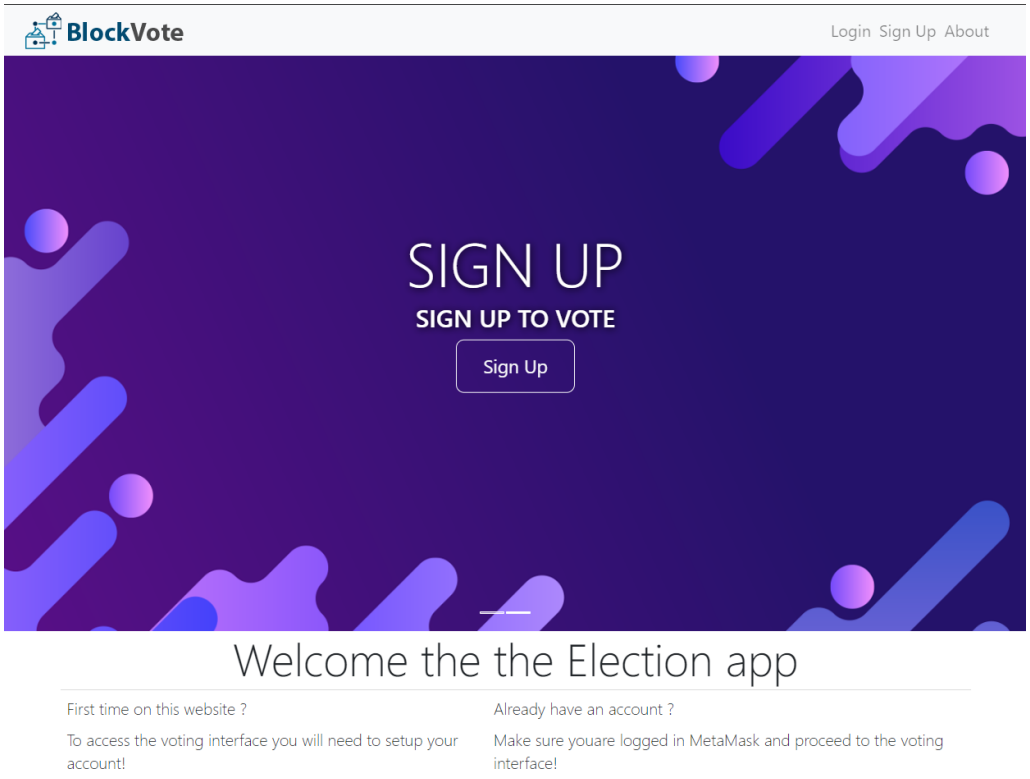

Figure IV.25 : Welcome page with the sign up slide active

On the registration page, the user will have to fill the required information and complete the registration by validating the transaction in MetaMask.

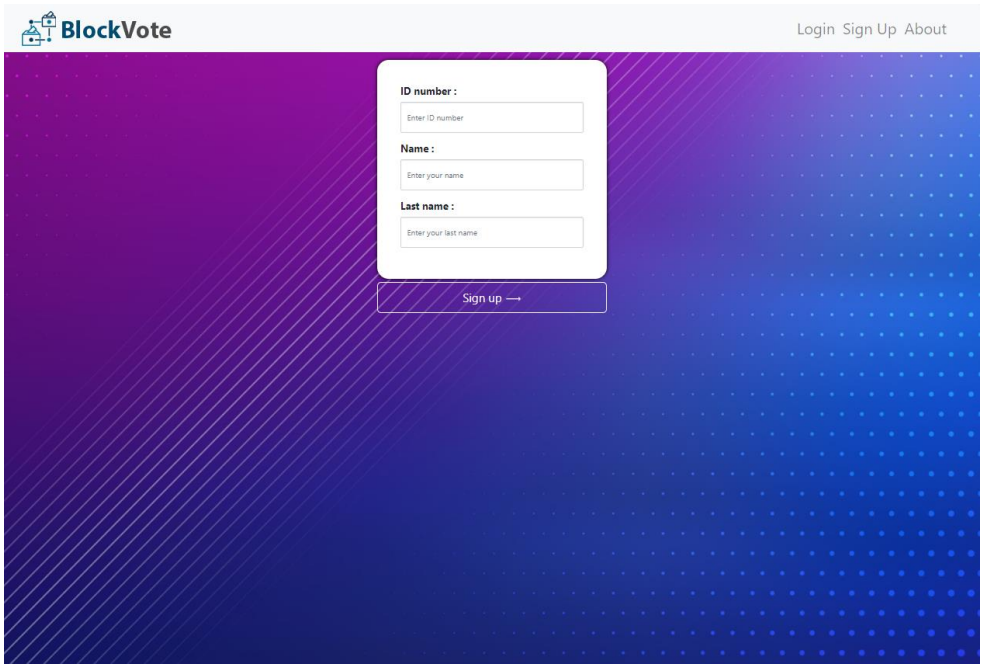

Figure IV.26 : Registration page

In case of an already existing account or any incorrect information, the page will display error messages and alerts to the user:

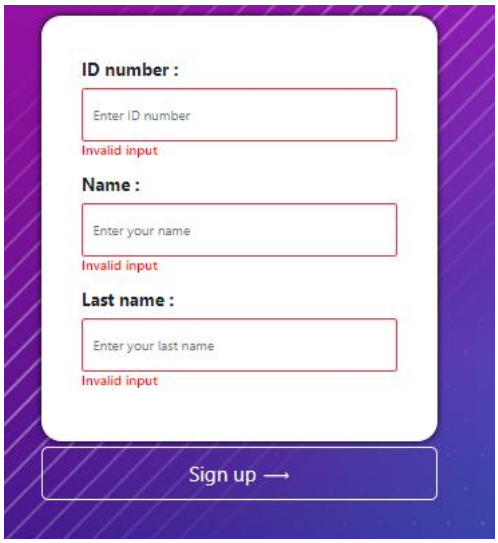

Figure IV.27 : Error handling in registration page

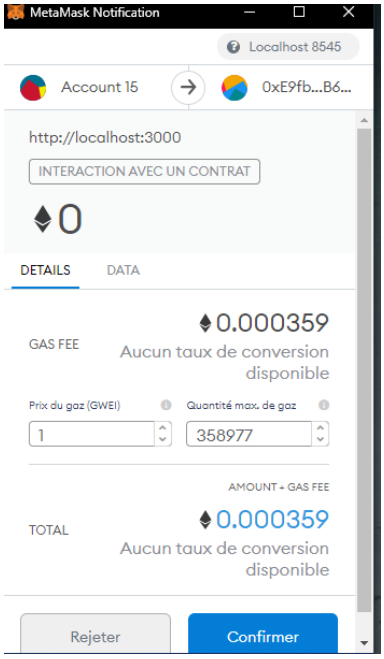

Figure IV.28: MetaMask registration transaction details

If all the informations are correct, the user will have to confirm the transaction, as the figure above is showing, the transaction will cost 0.000359 ether which is equivalent to 102.76 Dzd (30/06/2021).

Once Logged in, the user can now vote to his favorite candidate:

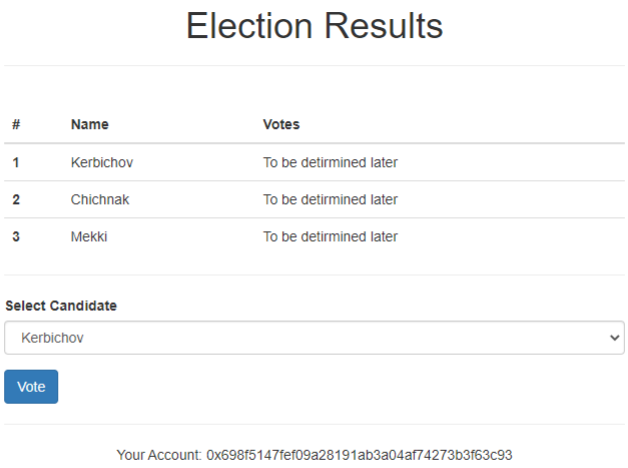

Figure IV.29 : Voting page

After choosing the candidate and pressing the vote button, the user must now confirm the transaction:

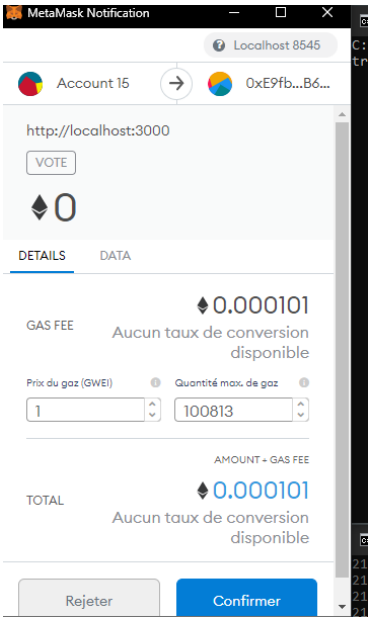

Figure IV.30: MetaMask voting transaction details

The voting transaction will cost the user 0.000101 ether which is equivalent to 28.91 Dzd (30/06/2021).

After confirming the transaction, our smart contract will store the vote, add the new transaction to the blockchain and the user will be redirected the updated web page and the voting option will be revoked from the user:

Figure IV.31 : Voting page after casting the vote

The election result will be displayed only after the voting period has passed:

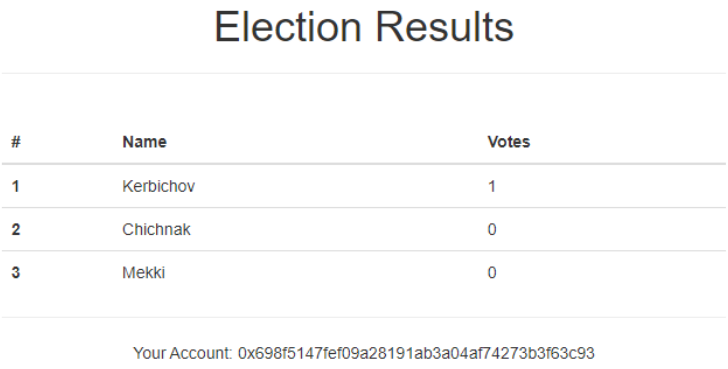

Figure IV.32 : Voting page after the end of the elections

Figure IV.33 : Vote transaction details

7. Conclusion :

In this chapter, we present the different tools and technologies that have been used to build our voting platform. Indeed, we have completed the implementation of this system and we have shown the interfaces realized to clarify the steps of use of this information system. We have suggested a method that provides maximal transparency, and immutability of data without the need for an intermediary authority.

General conclusion :

Blockchain has progressed far beyond its original use case of electronic money free of a central authority. New concepts have emerged as a result of this technology, ensuring immutability and increasing security. These characteristics make blockchain technology useful for a variety of applications, such as voting systems.

In this thesis, we have explored blockchain and proposed solutions based on this technology. Our proposal concerns an application for electronic voting using this technology in order to maintain a transparent and immutable elections.

In this thesis we have studied blockchain and Ethereum technologies as well as smart contracts, in our work we implemented and deployed a smart contract in a local Ethereum network and developed a graphical user interface to easily communicate with our smart contract. This project was extremely beneficial to us as it allowed us to expand our understanding of blockchain on both academic and practical levels. It also enabled us to uncover and learn new programming and development skills in the field of decentralized applications.

The potential of the Blockchain moving forward is huge, it is a fast growing technology all thanks to the rising demand for faster and more secure transactions along with full transparency. To complete this project, we propose that future work should focus on reducing transaction costs and using Oracles to enrich our smart contract and complete what is missing in the solidity language in order to build a much more optimal and ideal solution that satisfies our needs.

Bibliography

1. Samanta, S., et al., *Introduction to Blockchain Evolution, Architecture and Application with Use Cases*, in *Blockchain Technology and Innovations in Business Processes*, S. Patnaik, et al., Editors. 2021, Springer Singapore: Singapore. p. 1-16.
2. Mukherjee, P. and C. Pradhan, *Blockchain 1.0 to Blockchain 4.0—The Evolutionary Transformation of Blockchain Technology*, in *Blockchain Technology: Applications and Challenges*, S.K. Panda, et al., Editors. 2021, Springer International Publishing: Cham. p. 29-49.
3. Van der Auwera, E., et al., *Blockchain*, in *Financial Risk Management for Cryptocurrencies*. 2020, Springer International Publishing: Cham. p. 3-17.
4. Shi, M., et al. *Using Blockchain Technology to Implement Peer-to-Peer Network in Construction Industry*. 2021. Cham: Springer International Publishing.
5. Zhang, Y., *Blockchain*, in *Encyclopedia of Wireless Networks*, X. Shen, X. Lin, and K. Zhang, Editors. 2020, Springer International Publishing: Cham. p. 115-118.
6. Hellwig, D., G. Karlic, and A. Huchzermeier, *Smart Contracts*, in *Build Your Own Blockchain: A Practical Guide to Distributed Ledger Technology*. 2020, Springer International Publishing: Cham. p. 75-97.
7. Monrat, A.A., O. Schelén, and K. Andersson, *A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities.* IEEE Access, 2019. **7**: p. 117134-117151.
8. Singhal, B., G. Dhameja, and P.S. Panda, *Introduction to Blockchain*, in *Beginning Blockchain: A Beginner's Guide to Building Blockchain Solutions*. 2018, Apress: Berkeley, CA. p. 1-29.
10. Vigliotti, M.G. and H. Jones, *Bitcoin and Blockchain: The Fundamentals*, in *The Executive Guide to Blockchain: Using Smart Contracts and Digital Currencies in your Business*. 2020, Springer International Publishing: Cham. p. 41-70.
12. Attaran, M. and A. Gunasekaran, *Blockchain Principles, Qualities, and Business Applications*, in *Applications of Blockchain Technology in Business: Challenges and Opportunities*. 2019, Springer International Publishing: Cham. p. 13-20.
13. Fang, Z., et al. *A Blockchain Consensus Mechanism for Marine Data Management System*. 2020. Singapore: Springer Singapore.
14. Chen, F., et al. *Secure Scheme Against Compromised Hash in Proof-of-Work Blockchain*. 2018. Cham: Springer International Publishing.
15. Burmaka, I., et al. *Proof of Stake for Blockchain Based Distributed Intrusion Detecting System*. 2021. Cham: Springer International Publishing.
16. Gupta, S. and M. Sadoghi, *Blockchain Transaction Processing*, in *Encyclopedia of Big Data Technologies*, S. Sakr and A.Y. Zomaya, Editors. 2019, Springer International Publishing: Cham. p. 366-376.
17. Suzuki, J. and Y. Kawahara. *Blockchain 3.0: Internet of Value - Human Technology for the Realization of a Society Where the Existence of Exceptional Value is Allowed*. 2021. Cham: Springer International Publishing.
18. Dhillon, V., D. Metcalf, and M. Hooper, *Unpacking Ethereum*, in *Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make it Work for You*. 2021, Apress: Berkeley, CA. p. 37-72.
26. Praitheeshan, P., et al. *Attainable Hacks on Keystore Files in Ethereum Wallets—A Systematic Analysis*. 2019. Cham: Springer International Publishing.
28. Werner, S.M., P.J. Pritz, and D. Perez. *Step on the Gas? A Better Approach for Recommending the Ethereum Gas Price*. 2020. Cham: Springer International Publishing.
29. Mohanty, D., *Ethereum Architecture*, in *Ethereum for Architects and Developers: With Case Studies and Code Samples in Solidity*. 2018, Apress: Berkeley, CA. p. 37-54.

35. Vigliotti, M.G. and H. Jones, *Smart Contracts*, in *The Executive Guide to Blockchain: Using Smart Contracts and Digital Currencies in your Business*. 2020, Springer International Publishing: Cham. p. 133-149.
36. di Angelo, M. and G. Salzer. *Characterizing Types of Smart Contracts in the Ethereum Landscape*. 2020. Cham: Springer International Publishing.
37. Hu, B., et al., *A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems.* Patterns, 2021. **2**(2): p. 100179.
38. Khan, S.N., et al., *Blockchain smart contracts: Applications, challenges, and future trends.* Peer-to-Peer Networking and Applications, 2021.
39. Wilkens, R. and R. Falk, *Technische Grundlagen und Begriffe*, in *Smart Contracts: Grundlagen, Anwendungsfelder und rechtliche Aspekte*. 2019, Springer Fachmedien Wiesbaden: Wiesbaden. p. 5-15.
40. Karkeraa, S., *Smart Contracts*, in *Unlocking Blockchain on Azure: Design and Develop Decentralized Applications*. 2020, Apress: Berkeley, CA. p. 119-140.
42. Meng, W., et al. *Position Paper on Blockchain Technology: Smart Contract and Applications*. 2018. Cham: Springer International Publishing.

Webography

9. *Blockchain: What are nodes and masternodes?* [cited 2021 04/23]; Available from: https://medium.com/coinmonks/blockchain-what-is-a-node-or-masternode-and-what-does-it-do-4d9a4200938f.
11. *Ethereum*. [cited 2021 03/06]; Available from: https://ethereum.org/en/whitepaper/#ethereum.
19. *What Is Ethereum?* [cited 2021 02/06]; Available from: https://www.coindesk.com/learn/ethereum-101/what-is-ethereum.
20. [cited 2021 02/06]; Available from: https://ethereum.org/en/what-is-ethereum/.
21. *Ethereum Whitepaper*. [cited 2021 05/10]; Available from: https://ethereum.org/en/whitepaper/.
22. *Ethereum State Trie Architecture Explained*. [cited 2021 29/05]; Available from: https://medium.com/@eiki1212/ethereum-state-trie-architecture-explained-a30237009d4e.
23. *Ethereum EVM illustrated*. [cited 2021 10/06].
24. *web3.js - Ethereum JavaScript API*. [cited 2021 30/05]; Available from: https://web3js.readthedocs.io/en/v1.3.4/.
25. *Web 3: A platform for decentralized apps*. [cited 2021 03/05]; Available from: https://ethdocs.org/en/latest/introduction/web3.html.
27. *GAS AND FEES*. [cited 2021 27/05]; Available from: https://ethereum.org/en/developers/docs/gas/.
30. *Turing Completeness and Smart Contract Security*. [cited 2021 30/05]; Available from: https://medium.com/kadena-io/turing-completeness-and-smart-contract-security-67e4c41704c.
31. *Turing Complete*. [cited 2021 01/06]; Available from: https://academy.binance.com/en/glossary/turing-complete.
32. *Introduction to Smart Contracts*. [cited 2021 02/06]; Available from: https://docs.soliditylang.org/en/v0.6.2/introduction-to-smart-contracts.html.
33. *INTRO TO ETHEREUM*. [cited 2021 03/06]; Available from: https://ethereum.org/en/developers/docs/intro-to-ethereum/.
34. *NETWORKS*. [cited 2021 06/02]; Available from: https://ethereum.org/en/developers/docs/networks/.
41. *Smart Contract Development*. [cited 2021 20/06]; Available from: https://hack.bg/dlt-blockchain-development-services/smart-contracts-development/.
43. *How Blockchain Technology Can Prevent Voter Fraud*. [cited 2021 10/06]; Available from: https://www.investopedia.com/news/how-blockchain-technology-can-prevent-voter-fraud/.
44. *What is Node.js?* [cited 2021 11/06]; Available from: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm.
45. *TRUFFLE OVERVIEW*. [cited 2021 11/06]; Available from: https://www.trufflesuite.com/docs/truffle/overview.
46. *Ganache overview*. [cited 2021 13/06]; Available from: https://www.trufflesuite.com/docs/ganache/overview.
47. [cited 2021 11/06]; Available from: https://www.ibm.com/topics/smart-contracts.