



People's Democratic Republic of Algeria  
Ministry of Higher Education and Scientific Research



Saad Dahleb University - Blida 1

Faculty of Sciences

Computer Science Department

**Thesis Submitted in Partial Fulfillment of the Requirements for the  
Master Degree of Information Systems Security**

Authors:

**KERBICHE MOHAMED ELOUALID**

**LABRECHE MASSINISSA**

**Host organization: ETB/IFEG (Sonelgaz)**

**SECURITY AND DATA TRANSFER IN DECENTRALIZED  
APPLICATIONS USING BLOCKCHAIN TECHNOLOGY**

**Jury President: Mme. S.AROUSSI**

**Jury Examiner: Mr. Y.DOUGA**

**Advisor: Mme N.BOUSTIA**

**Supervisor: Mr. M.ZATOUT**

2020/2021

# Abstract

Blockchain technology has received great attention recently for enhancing data security in decentralized applications. It enables data to be exchanged between different contracting parties that act anonymously within a network and serves as a promising alternative to the current organizational and technical infrastructure. Blockchain transactions are administered by smart contracts, software programs that are executed automatically as soon as a given condition is met. They offer the possibility of creating a new form of agreement between parties without the need for intermediaries.

In this thesis, we have studied Blockchain technology and decentralized applications. . Then, we addressed our host organization “ETB/IFEG” needs and problematic that mainly represent the absence of a concrete solution to store and data files. To answer this problematic, we proposed a decentralized application as a solution based on Blockchain technology, decentralized storage and data file encryption. We implemented a Blockchain Smart Contract, developed a decentralized application and established an encryption for data files.

Through this process, we have concretized the security and data transfer in our decentralized application using the cryptography and immutability provided by Blockchain technology.

**Keywords:** Blockchain, Decentralized Application, Smart Contract, Data Security, p2p.

## ملخص

قد حظيت تكنولوجيا بلوكشاين باهتمام كبير في الآونة الأخيرة لتعزيز أمن البيانات في التطبيقات اللامركزية. وهو يتيح تبادل البيانات بين مختلف الأطراف المتعاقدة التي تتصرف دون الكشف عن هويتها داخل الشبكة، ويعمل كبديل واعد للهيكل الأساسية التنظيمية والتقنية الحالية. وتدار معاملات بلوكشاين بعقود ذكية، وهي برامج حاسوبية تنفذ تلقائيا بمجرد استيفاء شرط معين، وهي تتيح إمكانية إنشاء شكل جديد من الاتفاق بين الأطراف دون حاجة إلى وسطاء.

في هذه الأطروحة درسنا تكنولوجيا بلوكشاين والتطبيقات اللامركزية ثم تناولنا احتياجات ومشاكل المنظمة المضيفة " المدرسة التقنية لسونلغاز بالبليدة "والتي تبين عدم توفرهم على وجود حل ملموس لتخزين ملفات البيانات. وللإجابة على هذه المشكلة، اقترحنا تطبيق لا مركزي كحل يستند على بلوكشاين، التخزين اللامركزي وتشفير ملفات البيانات.

**الكلمات المفتاحية :** بلوكشاين ، التطبيق اللامركزي ، العقد الذكي ، أمن البيانات ,p2p.

## Acknowledgements

*We first thank ALLAH the almighty who guided us and gave us the strength and the willingness to realize this Master Thesis.*

*We would like to express our gratitude to our advisor Mme Narhimene Boustia for supporting this research project and for having guided, helped and advised us throughout the project. We would also like to thank our supervisor Mr Zatout Mohamed for his constant advice, orientation, guidance and support.*

*We thank our dear parents, for their unconditional support and encouragement, which has been a great help.*

*We address our sincere thanks to all the people who by their words and advice guided our reflections and contributed to the success of our master thesis.*

*Finally yet importantly, our thanks go to the jury the members for reviewing and judging our work.*

# Table of Content

GENERAL INTRODUCTION .....	9
CHAPTER I: DECENTRALIZED APPLICATION .....	11
I. Introduction .....	11
II. Decentralized applications.....	11
1. Definition: .....	11
2. Centralized Systems:.....	12
3. Decentralized Systems:.....	13
4. Centralized vs Decentralized .....	14
III. Conclusion .....	15
CHAPTER II: BLOCKCHAIN.....	16
I. Introduction: .....	16
II. Blockchain .....	16
1. Backstory of Bitcoin: .....	16
2. Definition of Blockchain: .....	16
3. Blockchain Components: .....	17
4. Layers of Blockchain: .....	19
5. Types of Blockchain: .....	20
6. Distributed Consensus Mechanisms: .....	22
7. Cryptography behind Blockchain: .....	24
8. Smart Contracts: .....	25
9. Choosing a Blockchain framework: .....	29
III. Conclusion: .....	30
CHAPTER III: CONCEPTION .....	31
I. Introduction: .....	31
II. Needs analysis and specification: .....	31
1. Identified problems: .....	31
2. Identified needs:.....	32
III. Conception view:.....	33

1.	Architecture view: .....	33
2.	Logical view:.....	35
3.	Process view: .....	36
4.	Deployment view:.....	40
IV.	Conclusion: .....	41
<b>CHAPTER IV: IMPLEMENTATION .....</b>		<b>42</b>
I.	Introduction: .....	42
II.	Development Tools:.....	42
1.	Node JS & React JS:.....	43
2.	Web3 JS: .....	43
3.	Truffle: .....	43
4.	Ganache:.....	44
5.	Remix IDE:.....	45
6.	Metamask: .....	45
7.	IPFS Desktop: .....	45
III.	Implementation details:.....	47
1.	Ethereum Blockchain:.....	48
2.	Ethereum Smart Contract:.....	52
3.	AES256 data file encryption: .....	60
4.	Transactions and Functionalities:.....	63
IV.	Conclusion: .....	75
<b>GENERAL CONCLUSION.....</b>		<b>76</b>
<b>Bibliography .....</b>		<b>77</b>
<b>Webography .....</b>		<b>80</b>

# List of Figures

Figure 1 : Centralized Architecture[5].....	12
Figure 2 : Decentralized Architecture[5].....	13
Figure 3 : Blockchain Blocks[9] .....	17
Figure 4 : How Blockchain Works[11] .....	18
Figure 5 : Blockchain Layers[13] .....	19
Figure 6 : Public Blockchain[14] .....	20
Figure 7 : Private Blockchain[14] .....	21
Figure 8 : Flow of PoW[17] .....	23
Figure 9 : Flow of PoS[17] .....	23
Figure 10 : Chained hash Blockchain blocks[18].....	24
Figure 11 : Functionality of smart contracts[25] .....	27
Figure 12 : Architecture .....	33
Figure 13 : IPFS Merkle DAG[45].....	34
Figure 14 : General class diagram .....	35
Figure 15 : Authentication sequence diagram.....	36
Figure 16 : Upload data file sequence diagram .....	37
Figure 17 : Share data file sequence diagram.....	38
Figure 18 : Retrieve data files sequence diagram.....	39
Figure 19 : Deployment diagram .....	40
Figure 20 : Implementation Architecture .....	42
Figure 21 : Workspace Main Interface.....	44
Figure 22 : IPFS desktop .....	46
Figure 23 : IPFS API configuration.....	46
Figure 24 : Mapping Hash table[50] .....	53
Figure 25 : Smart Contract addUser function .....	54
Figure 26 : Smart Contract getAllUserInfo function .....	54
Figure 27 : Smart Contract uploadFile function.....	55
Figure 28 : Smart contract remove hash function .....	56
Figure 29 : Smart contract get file function .....	56
Figure 30 : Smart contract upload file function .....	57
Figure 31 : Smart contract get file key function .....	57
Figure 32 : Smart contract deployment.....	58
Figure 33 : Compilation of smart contract[54] .....	59
Figure 34 : AES256-ctr encryption process[57] .....	60
Figure 35 : Convert data file to buffer .....	60
Figure 36 : Data file array buffer .....	61
Figure 37 : AES256-ctr auto generation of key and iv .....	61
Figure 38 : Encrypt aes256-ctr function.....	61

Figure 39 : Encrypted data file buffer .....	62
Figure 40 : Decrypt aes256-ctr function .....	62
Figure 41 : Metamask access .....	63
Figure 42 : Ganache account private key.....	63
Figure 43 : Connect Dapp to ganache.....	64
Figure 44 : Handling Ethereum network connection.....	64
Figure 45 : Registration form .....	66
Figure 46 : Register input validation .....	66
Figure 47 : Register transaction .....	67
Figure 48 : Dapp upload file .....	69
Figure 49 : Confirm upload transaction .....	70
Figure 50 : Upload Data File transaction details.....	71
Figure 51 : Dapp share data file .....	72
Figure 52 : Share Data File transaction details .....	72
Figure 53 : Delete data file.....	74

## List of Tables

Table 1 : Blockchain types comparison.....	22
Table 2 : Challenges of smart contracts[23] .....	28
Table 3 : General Comparison of Blockchain Frameworks[39, 40].....	29



# Glossary

PoW = Proof Of Work.

PoS = Proof Of Stake.

PBFT = Practical Byzantine Fault Tolerance.

EVM = Ethereum Virtual Machine.

ABI = Application Binary Interface.

P2P = Peer-To-Peer.

DAG = Directed Acyclic Graph.

Dapp = Decentralized Application.

SC = Smart Contract.

DHT = Distributed Hash Table.

IPFS = InterPlanetary File System.

SHA-256 = Secure Hashing Algorithm-256.

UML = Unified Modeling Language.

Blob = Binary Large Object.

AES-256-CTR = Advanced Encryption Standard 256 Counter mode.

## GENERAL INTRODUCTION

Nowadays, organizations transfer and store huge amount of data. From a security perspective, data in transit is always data at risk since data in transmission presents an opportunity for interception. Unauthorized access can also occur when data is stored at rest for download on a file transfer server. Moreover, there is always the chance of files being delivered to an unintended recipient or mishandled by end users that receive the files.

Decentralized applications solutions has emerged to resolving data transfer risks and creating a secured and an optimal workflow. One the most popular solution for a secured data transfer has emerged called Blockchain. Blockchain is the new wave of disruption that has already started to redesign business, social and political interactions, and any other way of value exchange.

Every organization will have different ideas as to what information is most sensitive. Financial information will always be in that top bucket because. The Technical School of Blida (ETB/IFEG) offers a stimulating environment conducive to learning. Even though the organization has a major problem for transferring data files between its divisions.

### **Problematic:**

There is an absence of a decentralized and secured solution, which allows the management of the different types of data files at Technical School of Blida. As many other organization, they use traditional ways for transferring data files like emails , flash disks or paper printed documents, which can affect the integrity and the availability of the data files .They even use their own email network, which becomes saturated because of the space limit restriction that forces the organization users to delete their old emails . Therefore many data files with sensitive data becomes unreachable, may be stolen and all of that without traceability.

### **Objective:**

The objective of our thesis design and implementation of a secure distributed application that allows Technical School of Blida to store and share their data files between their different divisions in a secured, reliable and decentralized way.

Our thesis is divided in four chapters:

- **Chapter I:** in which we have given a general overview about Decentralized Application.
- **Chapter II:** in which have studied Blockchain technology.
- **Chapter III:** where we have exposed and discussed our solution conception.
- **Chapter IV:** in which we have presented our implementation process.

As general conclusion, we will summarize our work and will open an axis towards the improvement of our solution with perspectives.

# CHAPTER I: DECENTRALIZED APPLICATION

## I. Introduction

A new model for building massively scalable and profitable applications is emerging. These applications paved the way with their peer-to-peer technology. This feature provide a starting point for building a new type of software called decentralized applications, or Dapps. In this chapter, we will speak about Decentralized Application, as we will highlight on both centralized and decentralized systems.

## II. Decentralized applications

In a business perspective, decentralization means that business can make decisions locally. A business unit can choose the way to use local resources to fulfill objectives for that unit. The unit must cooperate with other units in the company perhaps also externally, and must report to management in a specified way. However, there is a freedom of action to perform within each business unit. In addition, in a decentralized organization there must be a central coordination. Without management and control, the organization ends up in anarchy. Even if this central coordination such as standardization can set restrictions for each unit, the main criteria for decentralization is the right and responsibility to form an efficient inner structure in each business unit, using local resources to fulfill objectives and tasks that are set for the unit. Changes in this inner structure should not affect other units.[1]

### 1. Definition:

Decentralized applications are digital applications or programs that exist and run on a P2P network of computers instead of a single computer, and are outside the purview and control of a single authority. By using Blockchain technology Dapps grant users more control over their data by eliminating the need for centralized intermediaries to manage the data, thus making the service decentralized.[2]

**Decentralized** simply means not centralized. The control, as opposed to with a single entity, lies with the end users. Where the peers are responsible for the availability of content on the network. More peers seed the data, higher is the availability & download speed. Decentralized systems have no single points of failure. Even if several nodes go down, the network as a whole is still up. There is no single entity control, so there is zero possibility of the network going down anytime unless all the nodes go down simultaneously which is a rare possibility.[3]

## 2. Centralized Systems:

Centralized systems are systems that use client/server architecture where one or more client nodes are directly connected to a central server. This is the most commonly used type of system in many organizations where client sends a request to a company server and receives the response[4]. Component of Centralized System are:

- Node (Computer, Mobile, etc.).
- Server.
- Communication link (Cables, Wi-Fi, etc.).

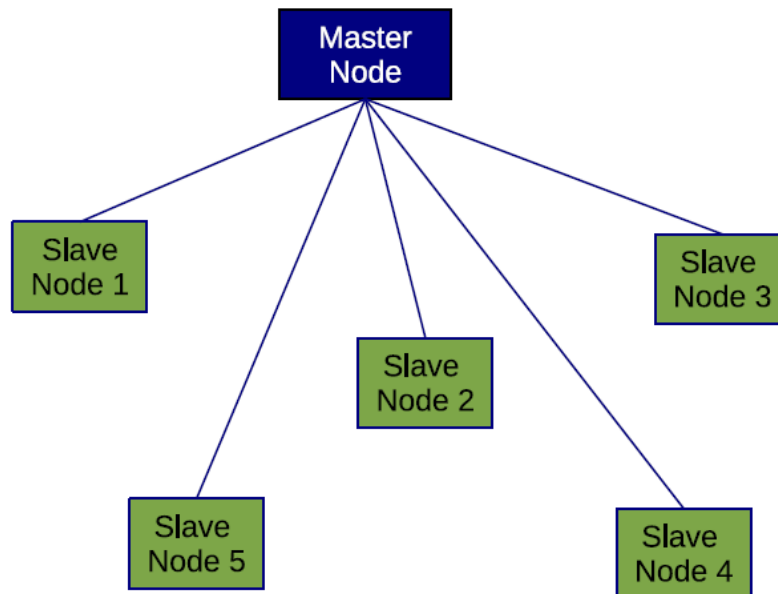


Figure 1 : Centralized Architecture[5]

### 3. Decentralized Systems:

As opposed to the centralized systems, in the decentralized case all or at least, several nodes have to host at least the base information, capabilities related to query processing, and management. Thus, requesters do not need to ask a central instance for query analysis and receiving information about the desired resources every time. Accordingly, the system becomes more failure resilient and in particular much more scalable.[5]

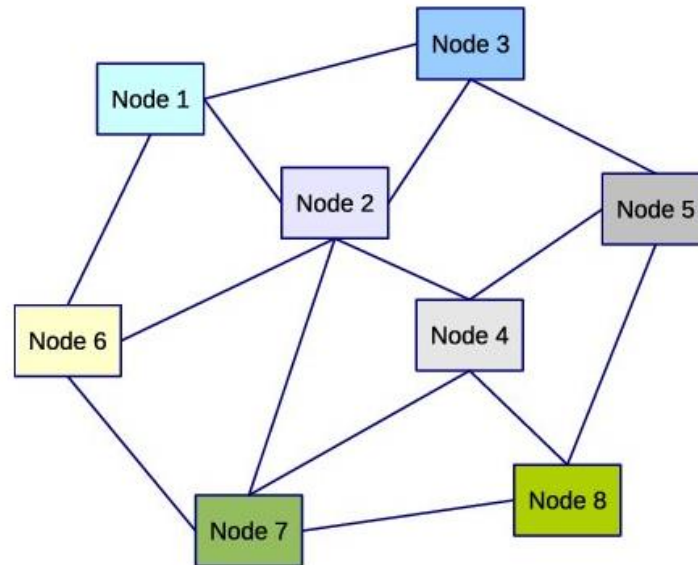


Figure 2 : Decentralized Architecture[5]

The concept of Peer-to-peer introduces many significant advantages in different aspects of resource discovery including scalability (due to collaborative resource sharing between peers), reliability (fault-tolerance due to the equality essence of peers), and robustness due to self-organization against peer or system failures. For resource discovery in P2P systems. [3]

#### 4. Centralized vs Decentralized

Decentralization should be applied where it makes sense. The goal of any Blockchain solution is to deliver what the users of that solution need, and this may or may not include certain levels of decentralization. For a better understanding of decentralized networks, the table below breaks out how decentralized networks compare to the more common centralized networks.[6]

	<b>Centralized</b>	<b>Decentralized</b>
<b>Network/hardware resources</b>	Maintained & controlled by single entity in a centralized location.	Resources are owned & shared by network members; difficult to maintain since no one owns it
<b>Solution components</b>	Maintained & controlled by central entity	Each member has exact same copy of distributed ledger
<b>Data</b>	Maintained & controlled by central entity	Only added through group consensus
<b>Control</b>	Controlled by central entity	No one owns the data & everyone owns the data
<b>Single Point of Failure</b>	Yes	No
<b>Fault Tolerance</b>	Low	Extremely high
<b>Security</b>	Maintained & controlled by central entity	Increases as the number of network members increase
<b>Performance</b>	Maintained & controlled by central entity	Decreases as the number of network members increase
<b>Example</b>	ERP Systems	Blockchain

Table 2: Centralized vs Decentralized [5]

We looked at the design aspects of centralized and decentralized systems and got some idea of the technical benefits of decentralized systems over centralized ones. Decentralized system allows for more data privacy. It is much more difficult to track information in such a system as information passes through a variety of points and not just through a single entity. An example of a decentralized network can be Blockchain.

### III. Conclusion

In this chapter, we have given an overview about Decentralized application. After we have spoken about Decentralized applications. We listed the different components and exposed the features of centralized and decentralized systems and their architecture. We ended this chapter with a comparison between centralized and decentralize approaches in order to understand how decentralized application can benefit organizations.

In the next chapter, we will speak about Blockchain technology as it an emerging concept of decentralisation.



## CHAPTER II: BLOCKCHAIN

### I. Introduction:

Blockchain is the new wave of disruption that has already started to redesign business, social and political interactions, and any other way of value exchange. In this chapter, we will explain what Blockchain is all about in order to understand its importance in building Decentralized Applications.

### II. Blockchain

#### 1. Backstory of Bitcoin:

When the Internet was made accessible to the public through the World Wide Web in the early 1990s, it was supposed to be more open and p2p. This is because it was built atop the open and decentralized TCP/IP. People adapted to the WWW revolution and leveraged the benefits it had to offer in every possible way. As a result, the World Wide Web started shaping up in a way that might not have been the exact way it was imagined. It could have been more open, more accessible, and more peer-to-peer. Satoshi Nakamoto must have felt that the monetary systems were not touched by the technological revolution since the 1980s. The whole of commerce relies on banks as trusted third parties to process payments. Financial institutions increase cost and time to settle a transaction, and limits transaction sizes. So, Satoshi invented a cryptocurrency called Bitcoin that was enabled by the underlying Blockchain technology.[7]

#### 2. Definition of Blockchain:

Blockchain is a system of records to transact value in a peer-to-peer fashion. It is a shared, decentralized, and open ledger of transactions. This ledger database is replicated across a large number of nodes. Each block contains its hash pointer, previous hash block, and data to be transferred in its structure. This ledger database is an append-only database and cannot be changed or altered. There is no need for trusted third parties to serve as intermediaries to verify, secure, and settle the transactions. Blockchain technology was designed to enable true decentralization.[7, 8]

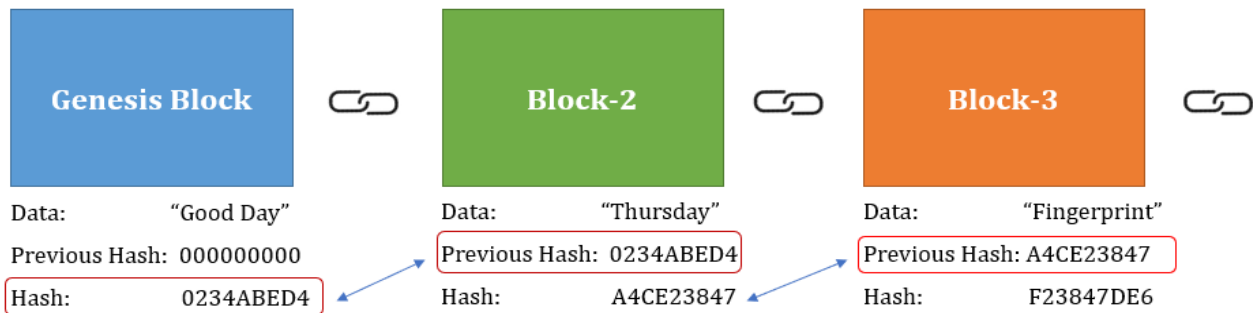


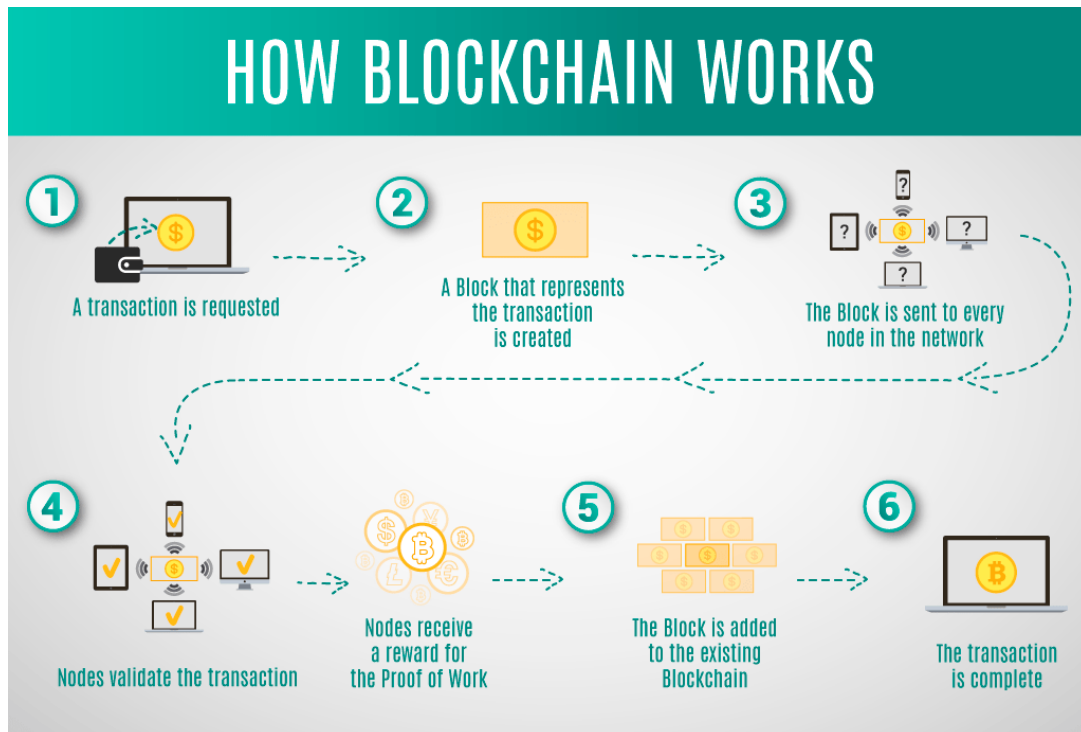
Figure 3 : Blockchain Blocks[9]

### 3. Blockchain Components:

These are the core Blockchain architecture components [10]:

- **Node** : user or computer within the Blockchain architecture (each has an independent copy of the whole Blockchain ledger)
- **Transaction**: smallest building block of a Blockchain system (records, information, etc.) that serves as the purpose of Blockchain.
- **Block**: a data structure used for keeping a set of transactions which is distributed to all nodes in the network
- **Wallet**: A Blockchain wallet is a digital wallet; it is used to securely store the private information of the user.
- **Ledger**: Every node in the Blockchain network is used to maintain a ledger of all transactions and this transaction maintains the state of the data that is being stored on the Blockchain network.
- **Peer Network**: It is peer-to-peer network contains a group of independent computers called as nodes that are interconnected to share data among each other with any centralized authority.
- **Chain**: a sequence of blocks in a specific order
- **Miners**: specific nodes which perform the block verification process before adding anything to the Blockchain structure
- **Consensus**: a set of rules and arrangements to carry out Blockchain operations.
- **Events**: It is used to create notifications for operations performed on the Blockchain.

The figure below resumes the general interaction between components when a transaction is requested.



**Figure 4 : How Blockchain Works[11]**

The transaction process works following this steps[12]:

- The sender sends a new request to start a new transaction.
- The request is created as a new form of transaction.
- The new transaction is broadcasted to the network for validation.
- A group of people verify and validate the broadcasted transaction.
- The valid transactions are approved and then added in the existing network.
- The verified transaction is stored as a new block.
- The newly created block is added to the existing block structure.
- The extent block structure sends a transaction to the destination node.
- The receiver can view the all the transactions done through the senders.
- All the transactions are maintained through a public ledger

#### 4. Layers of Blockchain:

In order to have a clear understanding of Blockchain there is five major layers that are present in a Blockchain node and are considered as a common point between Blockchain technologies. This illustration bellow represents those five layers.

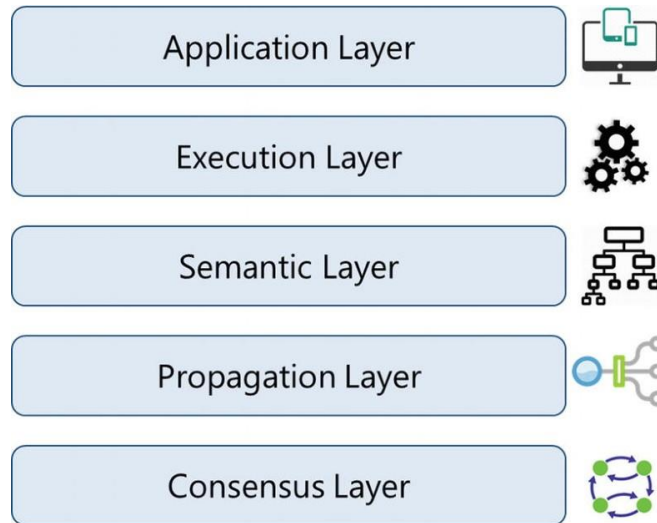


Figure 5 : Blockchain Layers[13]

- **Application Layer:**

In this layer, we find all functionalities codes in order to make application for the end users. It implies client-side programming constructs, scripts, APIs, development frameworks, etc.[13]

- **Execution Layer:**

All the nodes in a Blockchain network have to execute a script or a program to ensure a proper execution of the transactions. For that, the execution layer is where the ordered functions executions by application layer are taking place.[13]

- **Semantic Layer:**

The semantic layer is seen as a logical layer because it defines the logic and the rules of the system. In which all executed functions instructions in the execution layer get their validation in semantic layer. And to ensure the logic of the Blockchain, the linking of blocks with each other is defined in this layer.[13]

- **Propagation Layer:**

Considered as a the peer-to-peer communication layer the propagation layer allows the nodes to discover , talk and synchronize with each other according to the logic of the network . To guarantee all of that and ensure the stability of the Blockchain network, the transaction and network propagation are defined in the propagation layer. Therefore, when a transaction is made or a node wants to propose a valid block it is broadcasted to the entire network.[13]

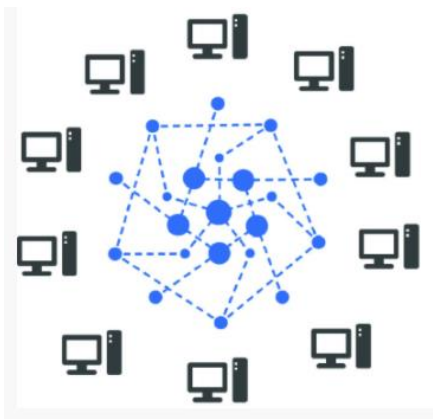
- **Consensus Layer:**

The Consensus Layer is considered as the base layer for Blockchain. Its primary purpose is to get all the nodes to agree on one consistent state of the ledger. There could be different ways of achieving consensus among the nodes, depending on the use case. [13]

## 5. Types of Blockchain:

There are different types of Blockchain from fully open, permissionless to permissioned depending on various features and covering large array of systems.

- **Public Blockchain:**



**Figure 6 : Public Blockchain[14]**

A user can join or leave the network as per his wish in the permissionless open public Blockchain. Here the consent from any central entity is not needed. Only user needs to join the network and add transactions to the digital record keeping register, which is generally a computing system where in the necessary software is installed.[14, 15]

- Private Blockchain:

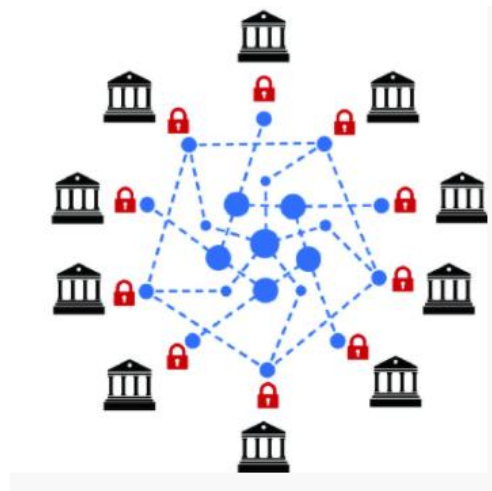


Figure 7 : Private Blockchain[14]

The rules for the ledger are pre-set by the network administrators in the private Blockchain. Nodes need to take permission from the network administrator for joining the network and initiating a transaction. The identity of the network nodes is verified before they join the network.[14, 15]

- Consortium Blockchain:

It is a hybrid combination in between low trusted permissionless and a single highly trusted authority model of permissioned Blockchain (combination of public and private Blockchains). This type of Blockchain generally functions under the group of board of managing directors or higher authorities. Consortium Blockchain is the best solution for inter-discipline, cross-industry applications, financial services, life sciences and health care, energy and resources, technology, media and telecommunications, public sector, and consumer and industrial products.[15]

- Comparison:

	Public	Private	Consortium
<b>Advantages</b>	<ul style="list-style-type: none"> <li>▪ Independence</li> <li>▪ Transparency</li> <li>▪ Trust</li> </ul>	<ul style="list-style-type: none"> <li>▪ Access control</li> <li>▪ Performance</li> </ul>	<ul style="list-style-type: none"> <li>▪ Access control</li> <li>▪ Scalability</li> <li>▪ Security</li> </ul>
<b>Disadvantages</b>	<ul style="list-style-type: none"> <li>▪ Performance</li> <li>▪ Scalability</li> <li>▪ Security</li> </ul>	<ul style="list-style-type: none"> <li>▪ Trust</li> <li>▪ Auditability</li> </ul>	<ul style="list-style-type: none"> <li>▪ Transparency</li> </ul>
<b>Use cases</b>	Cryptocurrency	Verify Ownership	Banking

**Table 1 : Blockchain types comparison**

From the comparison above, it may seem that public and private Blockchains cannot coexist, but in reality, several large-scale ventures implement them at the same time to address different requirements in their systems. It is a good idea to use private Blockchain if you are a company and want to use it without making everything public. More so, if you want more transparency to your network, then going for a public platform is a good option. But they are not that much suited for enterprise use cases.

## 6. Distributed Consensus Mechanisms:

The goal of consensus is to ensure that the network is robust enough to sustain various types of attacks. Now days there is two main use consensus algorithms

- **Proof of Work:**

The idea of PoW was invented in 1993 by Cynthia Dwork and Naor Moni . Markus Jacobson and Ari Juels coined the idea as "proof of work" in 1999 . PoW uses computing power for mining process by solving complex mathematical puzzles. The idea behind the PoW algorithm is that certain work is done for a block of transactions before it gets proposed to the whole network. Here comes the proof of work as an answer to the mathematical problem (crypto puzzle) which is considered as the solution that is difficult to find but it is easily verified. [8, 16].

## Proof of work Mechanism

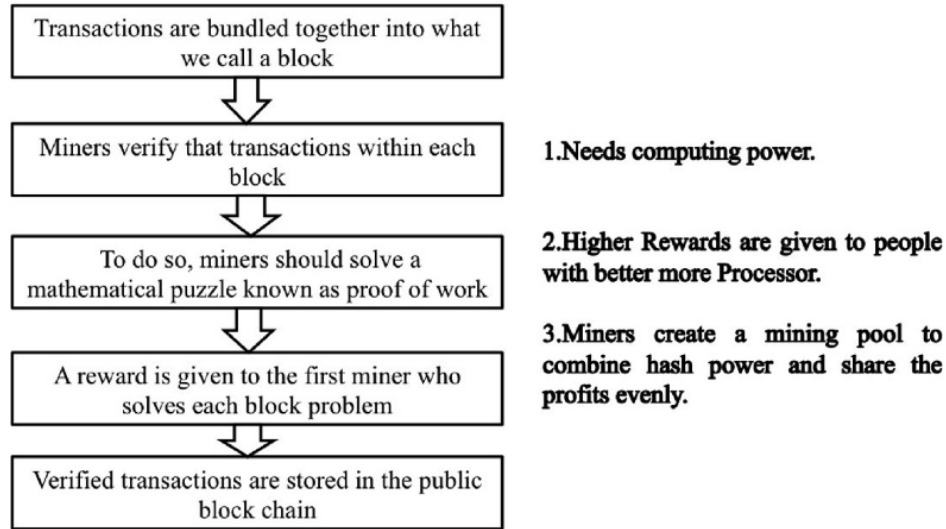


Figure 8 : Flow of PoW[17]

### ■ Proof of Stake:

The PoS algorithm is about validating blocks of transactions not mining. There are only transaction fees for the miners and for that all the digital currencies need to be created in the beginning and their total amount is fixed all through. Since the creator of a block in a PoS system is deterministic (based on the amount at stake), it works much faster compared with PoW systems. The PoS systems may provide better protection against malicious attacks because executing an attack would risk the entire amount at stake.[8, 16]

## Proof of stake

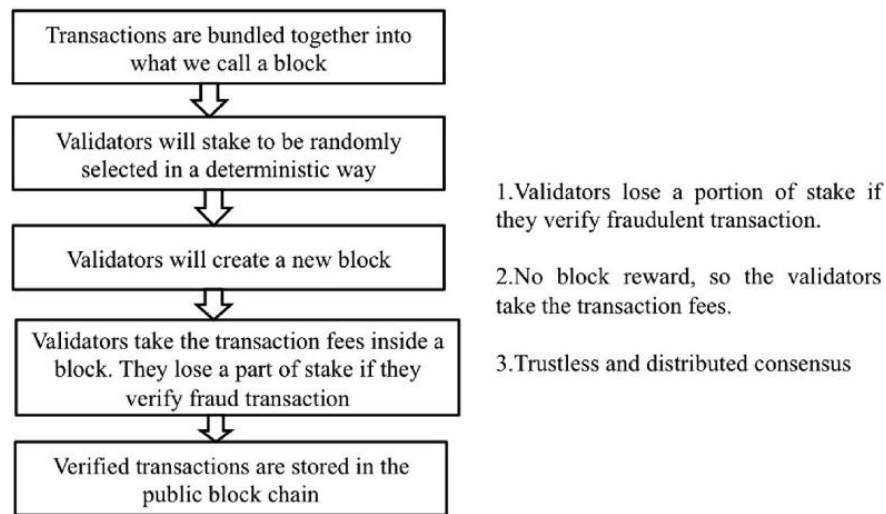


Figure 9 : Flow of PoS[17]



- **Comparison:**

In PoW and PoS consensus, the attacker needs a large amount of computing power or the stake for the creation of a long chain instead of the valid chain. Pow requires all miners to attempt to solve a complex puzzle, with the winner determined by the person who has the most powerful/quantity of hardware devices.

## 7. Cryptography behind Blockchain:

Cryptography is the most important component of Blockchain and used for the following cases:

- **Cryptographic hash Block:** Block chaining using Hash functions for example SHA256. As Each block consists of four fundamental elements: the hash of the previous block; the data of the block; the nonce of the hash; and the hash of the block. Each successive block enhances the authenticity claim for the preceding block by incorporating the hash of the previous block.[18]

The block hash represents a digital signature, which is a sequence of bits that is calculated from a message using the public-key procedure [15].In Blockchain technology; digital signatures are used to confirm that the transactions are derived from resources owner. In addition, Hash values allow a relatively clear and simple identification of the data; they are used as references in Blockchain technology. It also ensures that the block contents are protected against manipulation.[16]

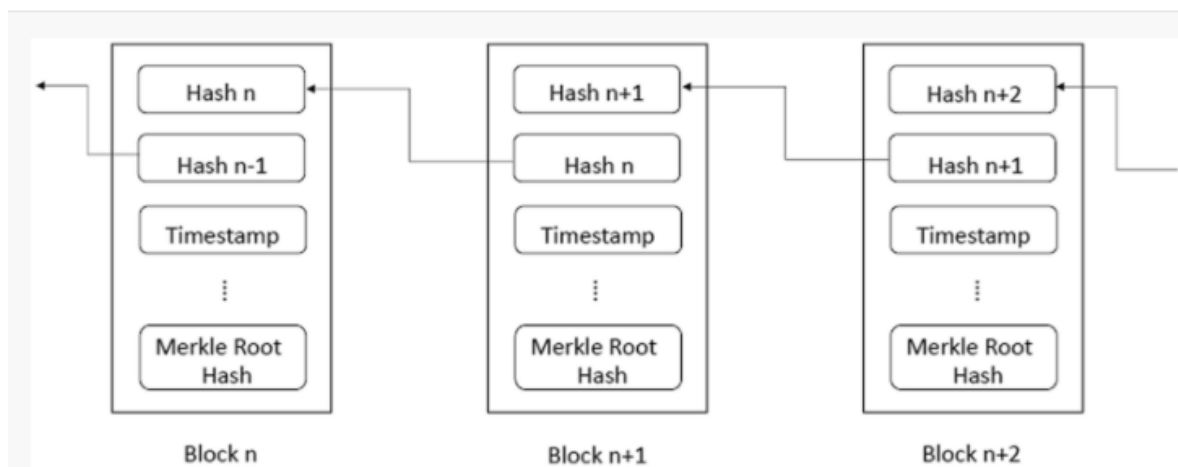


Figure 10 : Chained hash Blockchain blocks[18]

- **Asymmetric cryptography:** It is used by Blockchain to identify, verify, and validate transactions without revealing the users identities. Asymmetric cryptography is accomplished by using two distinct keys: public and private keys. The private key is kept private by each user, while the public key is available on the Blockchain network.[19]
- **User Identification and Addresses:** Addresses are used for the purpose of user identification in many Blockchain applications that offer multi-signature addresses. Several private keys are generated for this purpose with the intention of increasing security.[20]

## 8. Smart Contracts:

The term smart contract and the underlying idea date from long before the emergence of Bitcoin and Blockchain technology. Nick Szabo (1994) defined a smart contract as piece of computerized transaction protocol that satisfies contractual conditions such as payment terms, confidentiality or enforcement, reduces exceptions and minimizes the need for trusted intermediaries. A "smart contract" is simply a program that runs on the Blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Blockchain [21] .They run when predetermined conditions are met and typically are used to automate the execution of an agreement.[22]

### 8.1. Life cycle of a smart contract:

The life cycle of a smart contract typically consists of four broad phases: Creation, Deployment, Execution and Completion.

- **Creation of smart contracts:**

Several involved parties first negotiate contract obligations, rights, and prohibitions. Lawyers or counselors may assist parties in drafting an initial contractual arrangement. The arrangement is then converted into a smart contract written in programming by software engineers. Plan, execution, and validation are all steps in the smart contract conversion process, much as they are in app development. It is worth mentioning that the creation of smart contracts is an iterative process involving with multiple rounds of negotiations and iterations.[23, 24]

- **Deployment of smart contracts:**

The validated smart contracts will be deployed to Blockchain-based networks. Due to the immutability of Blockchain, contracts stored on them cannot be changed. Any alterations necessitate the creation of a new deal. After smart contracts are deployed on Blockchain, both parties will be able to access the contracts through the Blockchain .[23, 24]

- **Execution of smart contracts:**

A smart contract is made up of a series of declarative statements linked by logical relations. When a condition is met, the following declaration is executed immediately, resulting in a transaction being executed and checked by miners in the Blockchain. Following that, the committed transactions and modified states were stored on the Blockchain. [23, 24]

- **Completion of smart contracts:**

Once a smart contract is executed, all the parties' states are updated. As a result, the transactions that occur during the execution of smart contracts, as well as the modified states, are recorded in Blockchain. In the meantime, digital properties have been moved from one entity to another. As a result, the digital capabilities of the parties concerned have been unlocked. The smart contract has now finished its whole life cycle. [23, 24]

Smart Contract on the Blockchain:

A smart contract on the Blockchain is a piece of code that represents the terms of an agreement among parties. The obligations are enforced via the consensus process when the parties deploy the contract[25]. A smart contract on the Blockchain enables participants to:

- Inspect the code to ensure it meets the agreed clauses.
- Be reassured that an agreed contract, once registered on the Blockchain, is tamper-proof.
- Be certain the contract executes in the same way for all participants.
- Execute predefined actions according to the predefined occurred events.

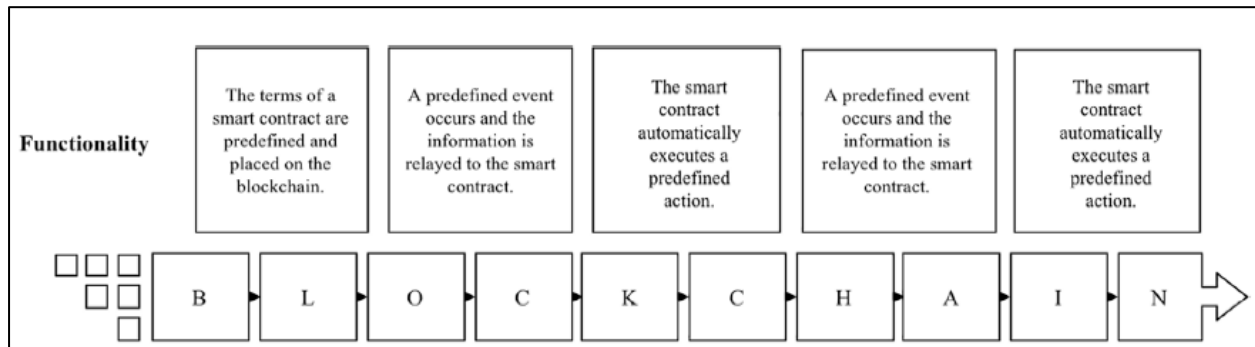


Figure 11 : Functionality of smart contracts[25]

## 8.2. Advantages of smart contracts:

Listed below the main advantages of smart contracts [26]:

- **Autonomy:** There is no need to rely on intermediaries to validate the deal since you are the one who makes it.
- **Trust:** Your documents are encrypted on a shared ledger.
- **Backup:** The data documents are duplicated many times over.
- **Safety:** Cryptography, the encryption of websites, keeps all documents safe.
- **Speed:** Smart contracts use software code to automate tasks, thereby shaving hours off a range of business processes.
- **Savings:** Smart contracts save you money (no presence of an intermediary).
- **Accuracy:** Automated contracts are not only faster and cheaper but also avoid the errors that come from manually filling out heaps of forms.

## 8.3. Challenges of smart contracts:

Challenges	Problems	Solutions
<b>Creation challenges</b>	<ul style="list-style-type: none"> <li>• <b>Readability:</b> smart contracts are written in different programming languages.</li> <li>• <b>Functional issues:</b> recalling interrupted function again and exploit this vulnerability to steal digital currency, sensitive information ...etc.</li> </ul>	<ul style="list-style-type: none"> <li>• Recover source code[27]</li> <li>• Human readable code and execution[28, 29]</li> <li>• Overcharging [30]</li> </ul>
<b>Deployment challenges</b>	<p><b>Contract correctness:</b> Impossible to make any revisions after deployment.</p> <p><b>Dynamic control flow:</b> The interaction of smart contracts result an increased number of interconnected contracts over time.</p>	<p>Bytecode and Source code analysis[31, 32]</p> <p>Graph based analysis[33]</p> <p>Path-searching[34]</p>
<b>Execution challenges</b>	<ul style="list-style-type: none"> <li>• <b>Transaction-ordering dependence:</b> The order of transactions is not deterministic due to the uncertainty of the bisected Blockchain branches</li> <li>• <b>Execution efficiency:</b> Smart contracts are serially executed which limits the system performance.</li> </ul>	<ul style="list-style-type: none"> <li>• Sequential execution[35]</li> <li>• Inspection of contract[36]</li> </ul>
<b>Data storing and sharing challenges</b>	<ul style="list-style-type: none"> <li>• <b>Data Volume:</b> Storing and sharing a large amount of data can affect the performance.</li> <li>• <b>Operations cost:</b> Storing and sharing data needs may be expensive.</li> </ul>	<ul style="list-style-type: none"> <li>• Use third party services.[37, 38]</li> </ul>

Table 2 : Challenges of smart contracts[23]

## 9. Choosing a Blockchain framework:

A Blockchain framework is software solution that contains infrastructure and libraries to develop the application. The network infrastructure or simply infrastructure consist of nodes and software running on them. It provides features and capability such as user identity, transaction details, consensus protocol and controls the identity management for Blockchains.

	Bitcoin	Ethereum	Hyperledger	Corda
<b>Privacy features</b>	No private transactions	Private transactions	Private channels, private transactions	Private transactions
<b>Access</b>	Public	Public, but supports permissioned networks	Permissioned	Permissioned
<b>Consensus</b>	PoW	PoW	Multiple approaches	Multiple approaches
<b>Smart Contracts</b>	Not available	Solidity	Java , Go, Java Script	Kotlin , Java , C#
<b>Industry focus</b>	Finance	Cross-industry	Cross-industry	Finance

**Table 3 : General Comparison of Blockchain Frameworks[39, 40]**

Ethereum is the best in terms of documentation and support, development and scalability.it did not have any major limitations. It has also been around for longer than the other platforms and has already been used to create fully developed applications for any field. Ethereum is also a platform that makes it possible for any developer to write Smart Contracts using high-level language solidity, which is nearly close to well-known programming languages

### III. Conclusion:

In this chapter, we have given an overview about the Blockchain technology. We have dug into Blockchain different components and transaction process followed with Blockchain layers and types. We also have described its main distributed consensus mechanisms and cryptography. After that, we have presented our study about smart contracts, their different characteristics, life cycle, its importance, advantages and challenges. We ended this chapter with a comparison between Blockchain frameworks and choosing Ethereum framework based on the comparison results

In the next chapter, we will speak about our conception process in which we will response to our problematic and expose its solution

## CHAPTER III: CONCEPTION

### I. Introduction:

In this chapter, we will present the conception steps of our proposed solution and that based on our previous research results in the previous chapters. As first step, we will present our host organization identify major issues and extract the needs to respond to our problematic that represent the absence of a concrete solution to manage data files in ETB . Therefore, we will describe our proposed solutions using different UML diagrams.

### II. Needs analysis and specification:

In our general introduction problematic, we have spoken about the need to guarantee the security and the data files transfer in any organization, which matches the needs of ETB School. After speaking with our internship supervisor who oriented us to the head master of the commercial and marketing, we have identified specific needs in data files transfer and security, which affects both the software and hardware security aspects.

#### 1. Identified problems:

We have listed below problems that could affect the security of data files and both software and hardware aspects.

##### 1.1. Limited sharing method:

The ETB School has a local network email with a limited storage for each user (700mb). Therefore, a user must constantly be aware of his storage, which may become full and oblige him to delete some previous data files to keep others. As result, the sharing and storage a drastically limited.

##### 1.2. Data files security:

Despite having sensitive data in the organization-shared files, there is no concrete security methods to guarantee the integrity and confidentiality of the shared files between the different entities.

##### 1.3. Large file sharing:

The Organization has bunch of important files with high size storage, which causes many troubles to diffuse or preview them. It affects mainly the hardware side and causes storage saturation and



latency in processing. As a result, the costs increases due to the need of high storage capacity in order fulfill the exigency of the organization.

#### 1.4. Excessive duplication of files:

With the absence of proper solution for sharing data files, the organization resorts to impractical solution as printing copies files, transferring files by emails, using flash disks and hard drives .This can lead to information leak, stilling and altering sensitive data files.

#### 1.5. Centralization:

Many data files are stored and available only in one general host of each division without being accessible remotely. All the data files existing in their respective divisions are accessible in a centralized way. Therefore, the accessibility and availability is practically limited.

### 2. Identified needs:

According to the resorted problems, there are needs to be full field to resolve the previous listed problems.

#### 1.1. Data security:

When properly implemented, robust data security strategies will protect an organization's information assets against cybercriminal activities, but they also guard against insider threats and human error, which remains among the leading causes of data breaches.

#### 1.2. Decentralization:

Decentralization represent an essential step into building a suitable solution for data files transfer. Decentralizing the management of and access to resources in an application provides benefits to the organization:

- Provides a trustless environment
- Reduces points of weakness
- Optimizes resource distribution

All the data files may have huge sizes and sensitive information. Therefore, all of them need to be shared and stored in an efficient and secured way.

### III. Conception view:

This Section describe our Dapp logic by exposing different uses case using different conception views.

#### 1. Architecture view:

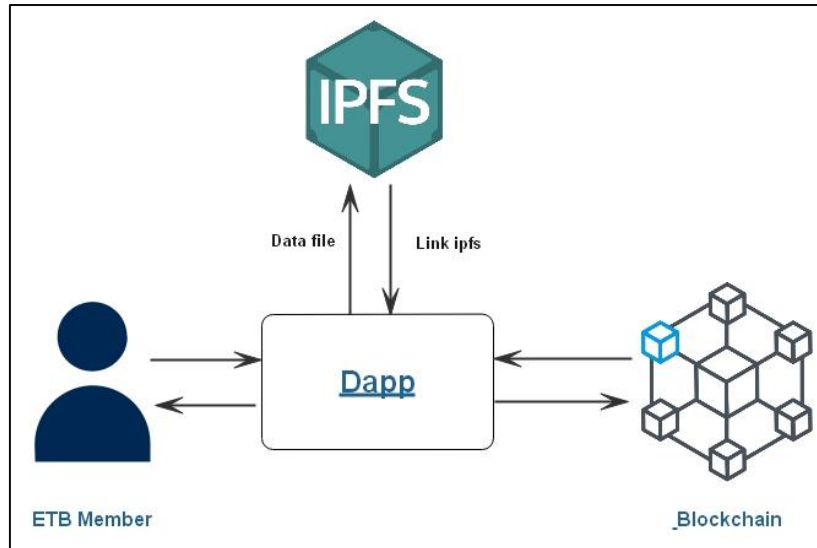


Figure 12 : Architecture

#### 1.1. InterPlanetary File System:

IPFS is a protocol and network designed to create a peer-to-peer method of storing and sharing data. It was initially designed by Juan Benet and is now an open source project developed with help from the community. IPFS connects all computing devices with the same system of files. In some ways, IPFS is similar to the Web.[37, 41]

We used IPFS as a peer-to-peer Decentralized Storage in our solution as it uses three main concepts to guaranty data files decentralizations:

- **Content Addressing:** IPFS uses content addressing to identify content by what is in it rather than by where it is located. Every piece of content that uses the IPFS protocol has a content identifier, or CID, that is its hash. The hash is unique to the content that it came from, even though it may look short compared to the original content.[42]

- **Directed acyclic graphs (DAGs):** IPFS takes advantage of a data structure called directed acyclic graphs or DAGs called Merkle DAGs. A Merkle DAG is a DAG where each node has an identifier, which is determined by hashing the node's contents, including any hidden content carried by the node, and the list of identifiers of its children, using a cryptographic hash function such as SHA256. Merkle DAG nodes are immutable. Any change in a node would alter its identifier and thus affect all the ascendants in the DAG, essentially creating a different DAG.[43, 44]

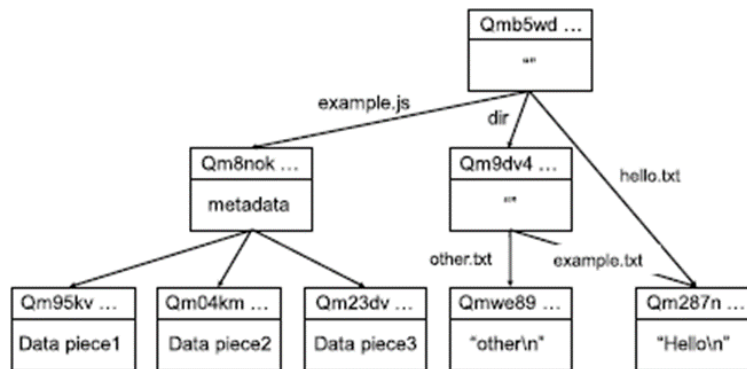


Figure 13 : IPFS Merkle DAG[45]

- **Distributed hash tables (DHTs):** IPFS uses a distributed hash table or DHT to find out which peers are hosting the content you are looking for. A hash table is a database of keys to values. A distributed hash table is one where the table is split across all the peers in a distributed network and to find content, you ask these peers.[46]  
The IPFS ecosystem includes the libp2p project, which provides the DHT and handles peer connections, communication.[47]

## 1.2. Blockchain:

The Blockchain is used to store users information, data files encryption keys and data files hashes which are returned from IPFS after upload IPFS on it. Users get access to the Blockchain network using their predefined account.

## 1.3. The Dapp:

The Dapp is the center of our solution as it allow the user to interact with the Blockchain network and the Decentralized storage (IPFS). The Dapp has a client-side where the user can execute different functions on the data files and the server-side on which all users queries and data file encryption are handled.

## 2. Logical view:

We can resume our system into three main classes:

- **ETB Member:** The main user of the Dapp as he is identified with his unique Blockchain account address. As a user, he can perform different function using our solution (authentication, register, upload data file, share data file, retrieve data file).
- **Data File:** The main class of our Dapp on which ETN member will execute different operations. As each data is identified with a unique ID on the Blockchain and its cryptographic keys.



Figure 14 : General class diagram

### 3. Process view:

The process view contains the interactions between the actors and the different components of the system.

#### 3.1. Authentication:

To access the Dapp functionalities , the following authentication steps are required:

- Use a wallet to get acces to the Blockchain network.
- Introduce Blockchain account credential access.
- Acces Dapp login page and confirme login.
- The Dapp check if the user is registred to the list of users.
- If yes the user accesses the Dapp functionalities.
- If not , the user is redirected to registration page.
- After registration, the user get access to the Dapp functionalities.

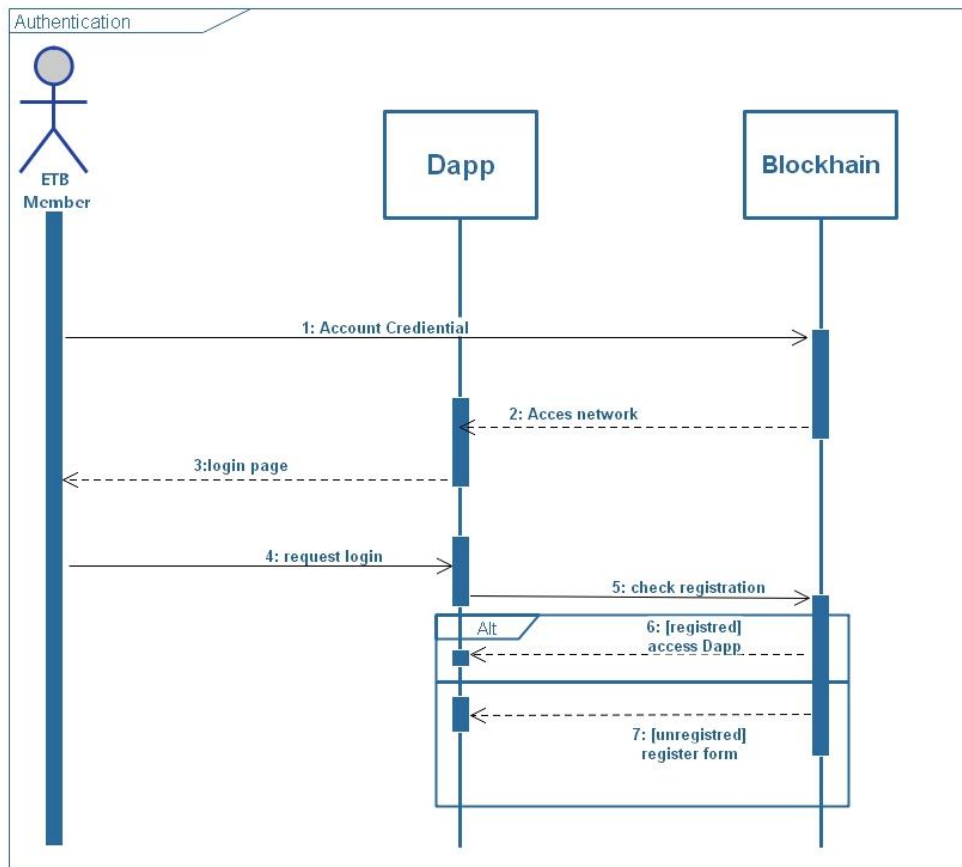


Figure 15 : Authentication sequence diagram

### 3.2. Upload data files:

A registered ETB member can upload his data files in IPFS in a secure way and save their information access in Ethereum Blockchain to ensure immutability and protection. Steps are:

- Select a local data file and confirm the upload transaction.
- The Dapp generate randomly a 16bits aes256 key and an initialization vector of 8bits for counter mode.
- The data file is converted to a buffer, then encrypted with aes256-ctr using they key and the initialization vector.
- The encrypted file is uploaded to ipfs, which returns the data file hashed path to the Dapp.
- The hashed path , aes256 key , aes256 iv are all stored to Ethereum Blockchain using an immutable and encrypted mapping structure in which each user has his data files access information stored safely.
- At the end, the Dapp interfaced updated with new added data file to the displayed files.

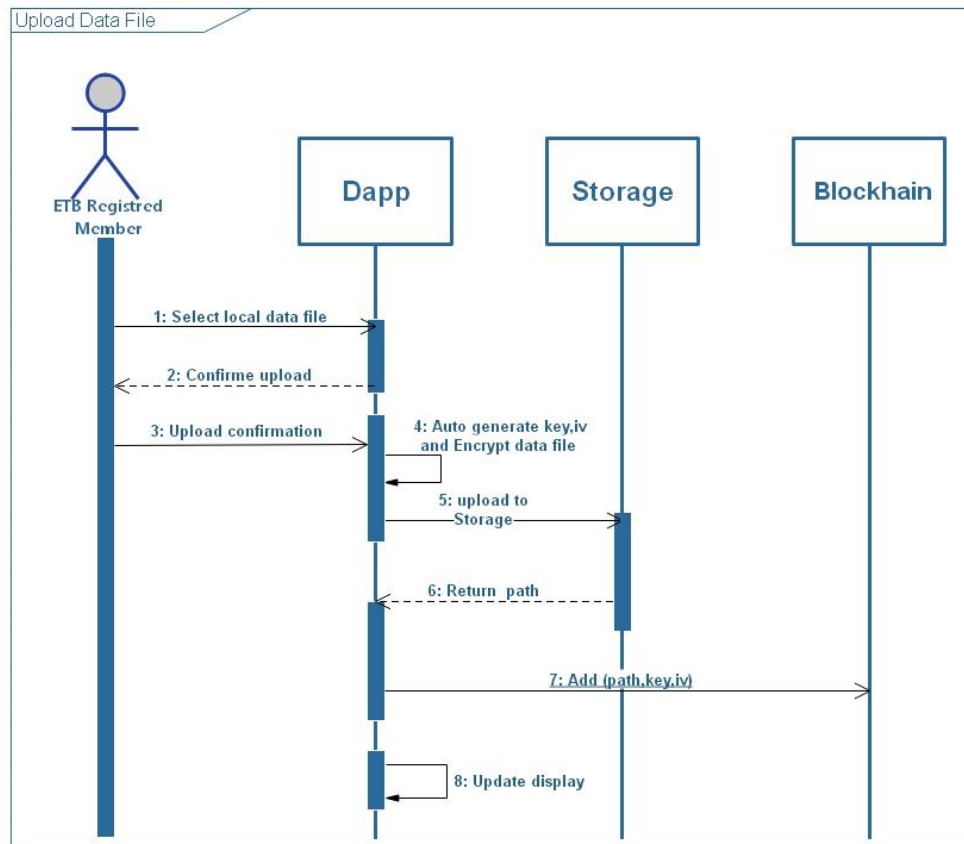


Figure 16 : Upload data file sequence diagram

### 3.3. Share data files:

A registered ETB member can share files with others by transferring his data files access information to the desired receiver. To do that he must:

- Select a data file from the ones displayed on the Dapp interface.
- The Dapp request the data file access information and users wallets addresses from the Ethereum Blockchain.
- Ethereum response after checking the identities of the requester.
- The user chooses the data file receiver to confirm the transaction.
- The hash, aes256 key and aes256 iv are transferred to the receiver using Ethereum Blockchain secured transaction.
- A message of success is displayed to the user at the end.

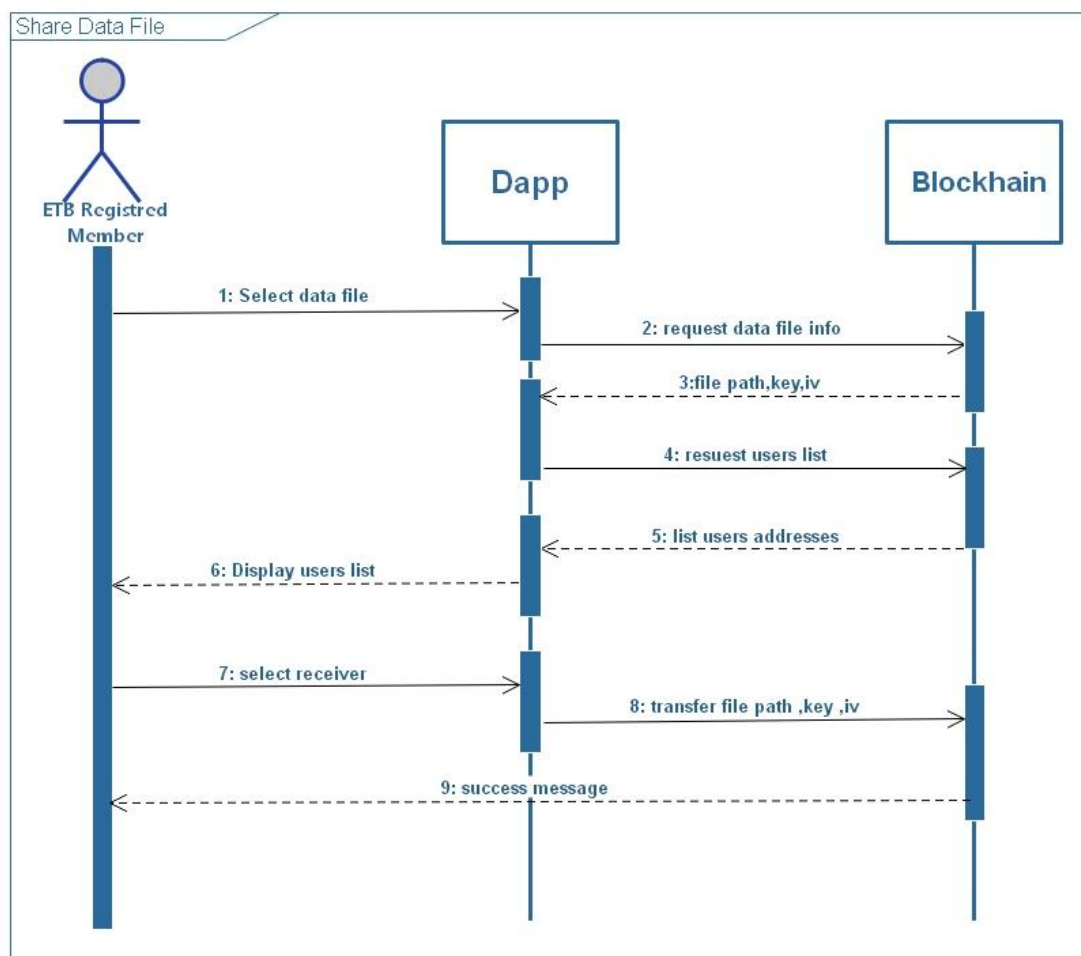


Figure 17 : Share data file sequence diagram

### 3.4. Retrieve data files:

A registered ETB member can retrieve his data files by following this step:

- Select one of the data files displayed on the Dapp interface.
- Dapp request property right to retrieve the path hash, aes256 key and aes256 iv.
- Ethereum Blockchain respond to the Dapp after checking property right.
- The Dapp retrieve the file from ipfs using the hashed path.
- The Dapp convert the data file to a buffer in order to decrypt it.
- The data file is decrypted using aes256-ctr with the key and iv.

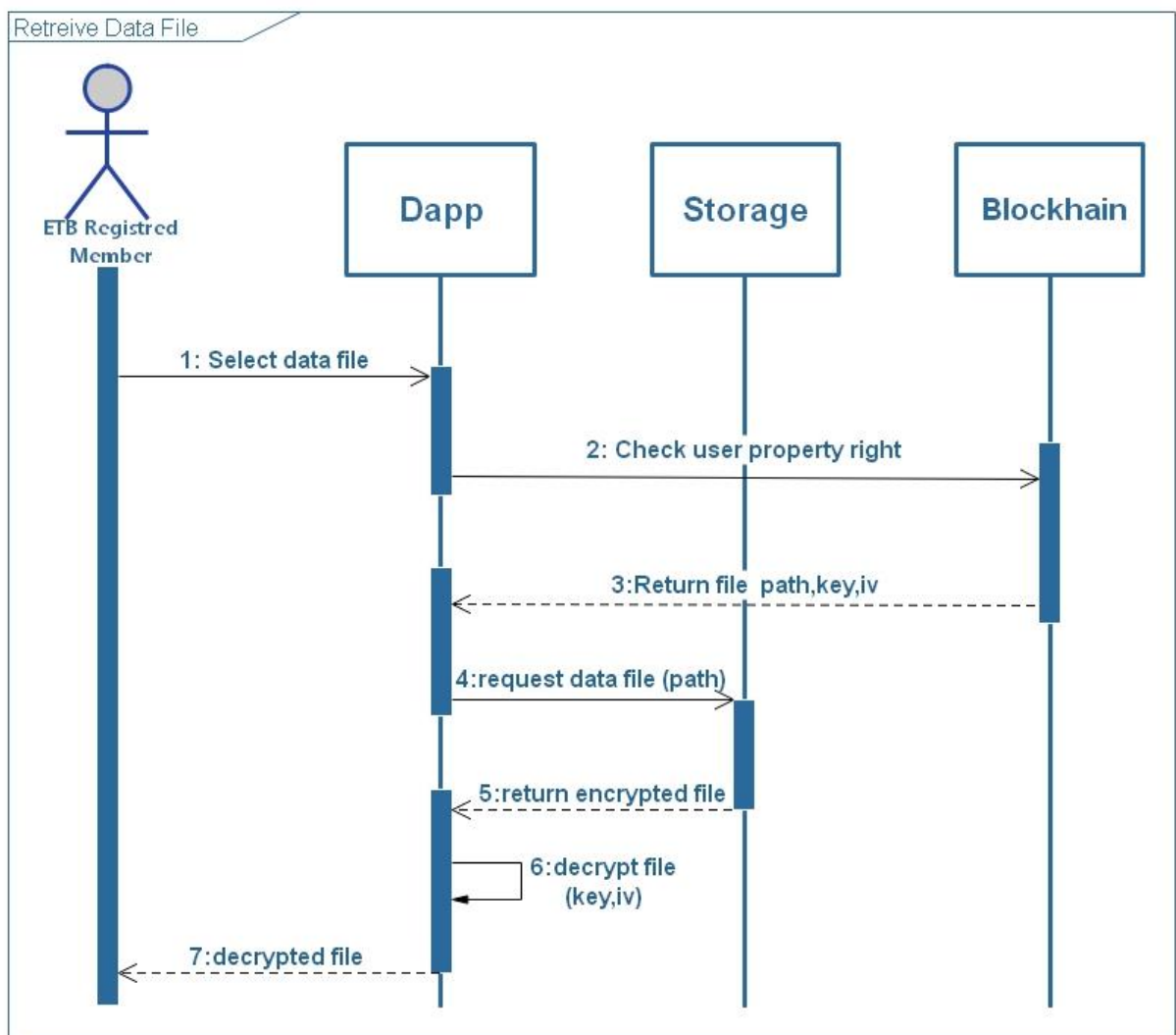


Figure 18 : Retrieve data files sequence diagram



#### 4. Deployment view:

The deployment view describes the arrangement of components, which makes the Dapp.

- **Web browser:** It provides the interface for the users to interact with the Dapp. Users need an Ethereum wallet browser to manage their accounts. The web interface provides the functionality to store data files information and users information on the Blockchain, as well as start the execution of different functionalities on data files.
- **Blockchain:** The system where the smart contract is deployed and provides an Application Binary Interface (ABI) to the users to call the contract methods in order to read or write information on the Blockchain.
- **Storage:** Where all data files should be stored.
- **Application Server:** a server specifically designed to run applications. The server has both the hardware and software components that provide an environment for the Dapp to run.
- **AES-256-ctr:** AES-256 counter mode is used to encrypt buffer converted data files. It uses a key and an IV (Initialization vector) which represent the counter value. This cryptographic algorithm has a good parallelization that improves the cypher speed.

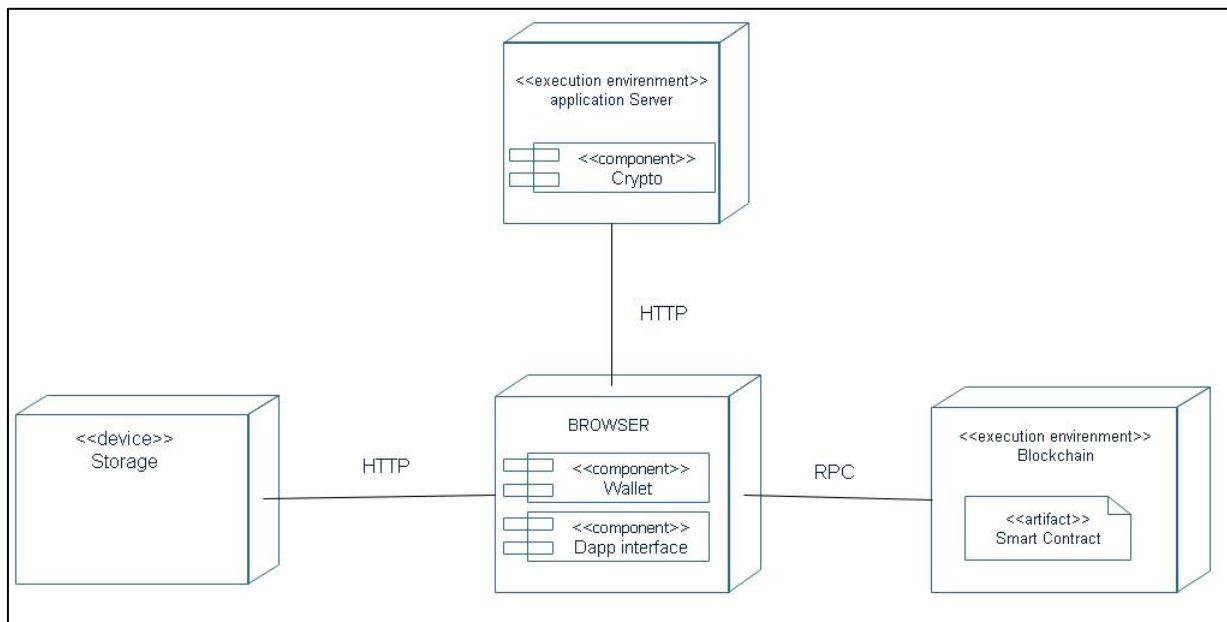


Figure 19 : Deployment diagram

#### IV. Conclusion:

In this chapter, we have described the functionalities and software architecture of our proposed Dapp solution. The Decentralized application aim to secure data files transfer and storage using cryptography and securing credential access and transfer using Blockchain security and immutability. We used different conception view to highlight our solution architecture, process logic and deployment architecture.

In the next chapter, we will speak about our implementation process in which we will concretize the results of our conception process.

## CHAPTER IV: IMPLEMENTATION

### I. Introduction:

In this chapter, we will introduce and bring a detailed presentation of our implementations. We will describe the different tools and approaches used in this process in order to achieve the realization of our solution.

### II. Development Tools:

This figure bellow represent an overview of over implementation architecture. It contains all the component .

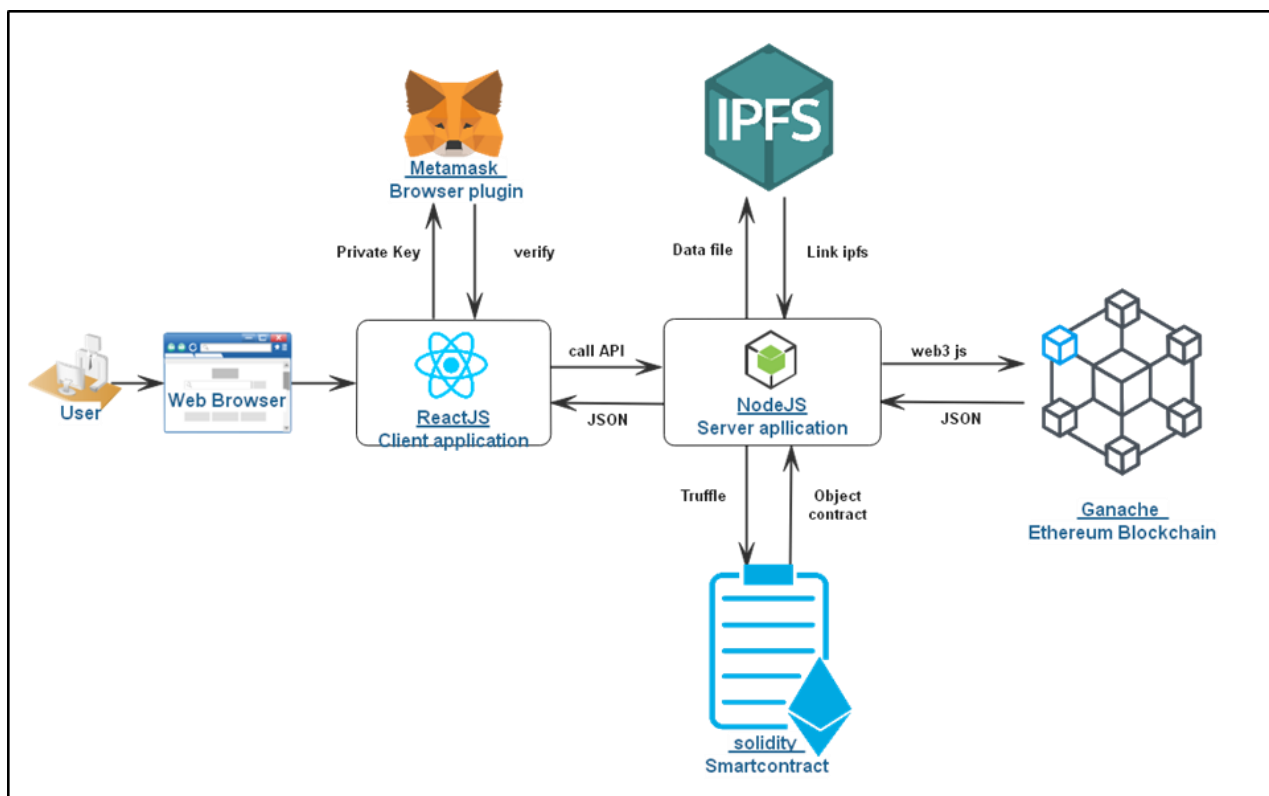


Figure 20 : Implementation Architecture

### 1. Node JS & React JS:

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

React.js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. The main purpose of React is to be fast, scalable, and simple. It works only on user interfaces in the application. This corresponds to the view in the MVC template.

Node.js is the most convenient platform for hosting as well as running a web server for a React application. It's because of two main reasons:

- Using an NPM (Node Package Manager), Node works alongside the NPM registry to easily install any package through the NPM CLI.
- Node bundles a React application into a single file for easy compilation using webpack and several other Node modules.

### 2. Web3 JS:

Web3.js talks to The Ethereum Blockchain with JSON RPC, which stands for "Remote Procedure Call" protocol. Ethereum is a peer-to-peer network of nodes that stores a copy of all the data and code on the Blockchain. Web3.js allows us to make requests to an individual Ethereum node with JSON RPC in order to read and write data to the network.[48]

### 3. Truffle:

A development environment, testing framework and asset pipeline for Blockchains using the Ethereum Virtual Machine (EVM).

With Truffle, we get built-in smart contract compilation, linking, deployment and binary management. Network management for deploying to any number of public & private networks. We install it on Node JS using npm with the command: **npm install -g truffle**.

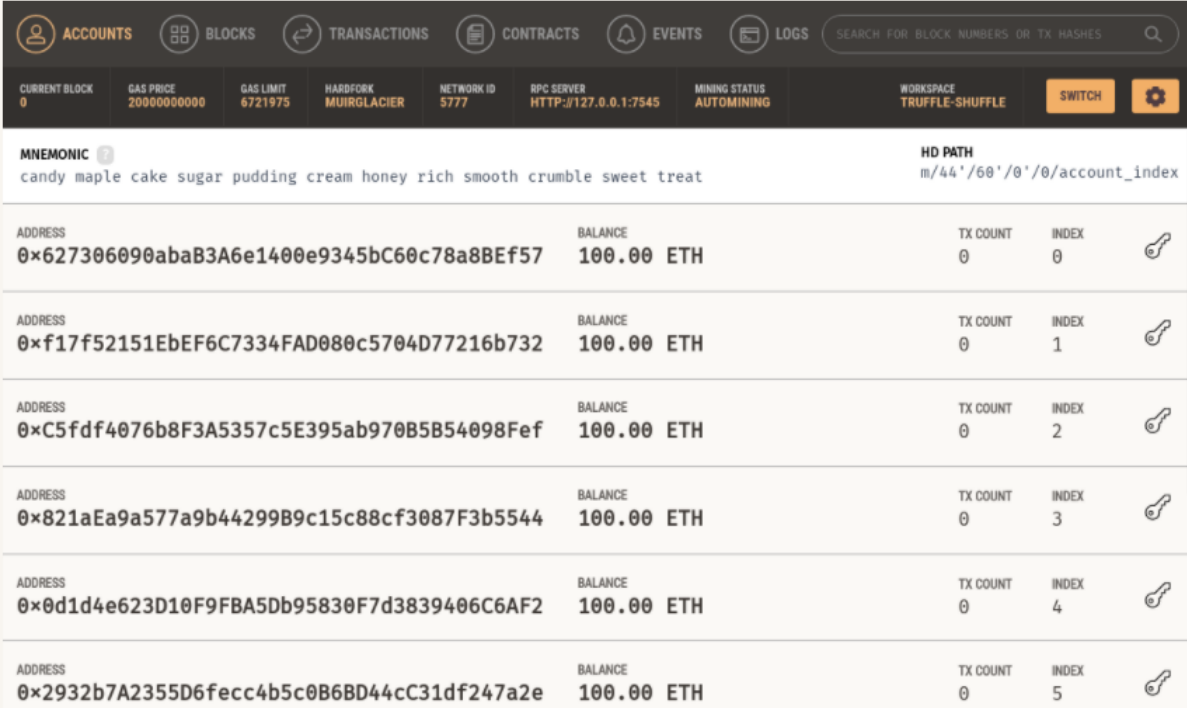
We execute on cmd: **truffle init** to create a project tree structure with the following items:

- contracts/: Directory for Solidity contracts
- migrations/: Directory for scriptable deployment files
- test/: Directory for test files for testing your application and contracts
- truffle-config.js: Truffle configuration file

#### 4. 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.

Once a workspace is created, the screen will show some details about the server, and list out a number of accounts. Each account is given 100 ether. Having ether automatically in all accounts allows you to focus on developing your application.



The screenshot shows the Ganache workspace main interface. At the top, there are navigation tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below the tabs, there is a search bar and a status bar with various metrics: CURRENT BLOCK (0), GAS PRICE (2000000000), GAS LIMIT (6721975), HARDFORK (MUIRGLACIER), NETWORK ID (5777), RPC SERVER (HTTP://127.0.0.1:7545), MINING STATUS (AUTOMINING), and WORKSPACE (TRUFFLE-SHUFFLE). The main content area displays the MNEMONIC (candy maple cake sugar pudding cream honey rich smooth crumble sweet treat) and the HD PATH (m/44'/60'/0'/0/account\_index). Below this, there is a table listing six accounts, each with an ADDRESS, BALANCE (100.00 ETH), TX COUNT (0), and INDEX (0-5). Each account entry also includes a key icon.

ADDRESS	BALANCE	TX COUNT	INDEX
0x627306090abaB3A6e1400e9345bC60c78a8BEf57	100.00 ETH	0	0
0xf17f52151EbEF6C7334FAD080c5704D77216b732	100.00 ETH	0	1
0xC5fdf4076b8F3A5357c5E395ab970B5B54098Fef	100.00 ETH	0	2
0x821aEa9a577a9b44299B9c15c88cf3087F3b5544	100.00 ETH	0	3
0x0d1d4e623D10F9FBA5Db95830F7d3839406C6AF2	100.00 ETH	0	4
0x2932b7A2355D6fecc4b5c0B6BD44cC31df247a2e	100.00 ETH	0	5

Figure 21 : Workspace Main Interface

There are six available pages on the interface:

- **Accounts:** shows the accounts generated and their balances (default view after creation).
- **Blocks:** shows each block as mined on the Blockchain, with gas used and transactions.
- **Transactions:** lists all transactions run against the Blockchain.
- **Contracts:** lists the contracts contained in your workspace's Truffle projects. For more information on how Ganache handles contracts, see our Contracts Page documentation.
- **Events:** lists all events that have been triggered since this workspace's creation. Ganache will attempt to decode events triggered by contracts in your Truffle project. For more information on events, see our Events Page documentation.
- **Logs:** shows the logs for the server, which is useful for debugging.

#### 5. Remix IDE:

Remix IDE is an open source web and desktop application. It fosters a fast development cycle and has a rich set of plugins with intuitive GUIs. Remix IDE allows developing, deploying and administering smart contracts for Ethereum Blockchain. We will use it for developing process only of the smart contract.

#### 6. Metamask:

MetaMask is a cryptocurrency wallet but also a web browser extension (available in Chrome, Firefox and Brave) used to store, send and receive Ethereum. It allows users to store and manage account private keys and make Ethereum transactions through regular websites. MetaMask can be used to store keys for Ethereum cryptocurrencies only.

#### 7. IPFS Desktop:

IPFS Desktop bundles an IPFS node, file manager, peer manager, and content explorer into a single, easy-to-use application. If you already have an IPFS node on your computer, IPFS Desktop will act as a control panel and file browser for that node. If you don't have a node, it'll install one for you. In addition, either way, IPFS Desktop will automatically check for updates[41, 42].

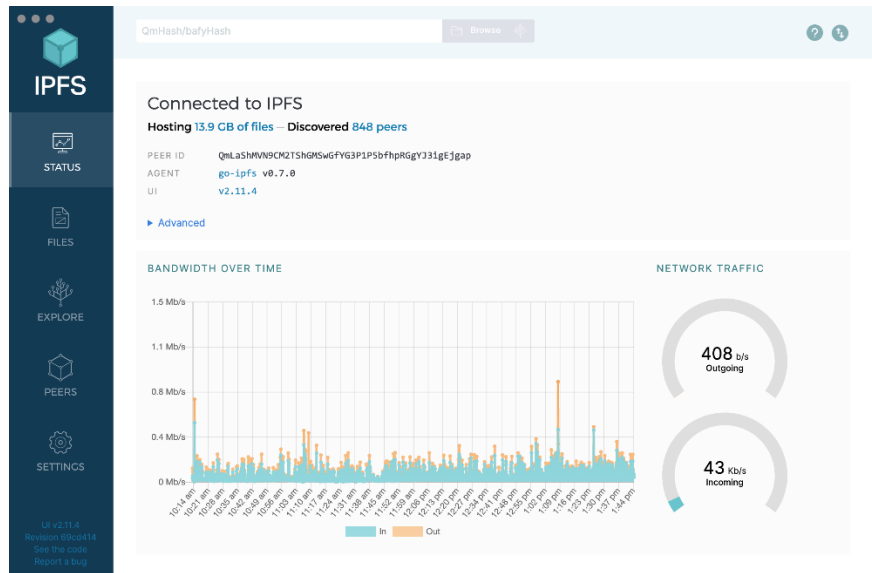


Figure 22 : IPFS desktop

We configure our IPFS API HTTP headers, as we set:

- Allowed access methods for our Dapp: Put, Post, and Get.
- Allowed access origin: The Dapp IP application server.

By doing this configuration steps we ensure only our Dapp access IPFS.

```

1 {
2   "API": {
3     "HTTPHeaders": {
4       "Access-Control-Allow-Methods": [
5         "PUT",
6         "POST",
7         "GET"
8       ],
9       "Access-Control-Allow-Origin": [
10        "http://127.0.0.1:3000/"
11      ]
12    }
13  },
14  "Addresses": {
15    "API": "/ip4/127.0.0.1/tcp/5001",
16    "Announce": [],
17    "Gateway": "/ip4/127.0.0.1/tcp/8080",
18    "NoAnnounce": [],
19    "Swarm": [
20      "/ip4/0.0.0.0/tcp/4001",
21      "/ip6:::tcp/4001",
22      "/ip4/0.0.0.0/udp/4001/quic",
23      "/ip6:::udp/4001/quic"
24    ]

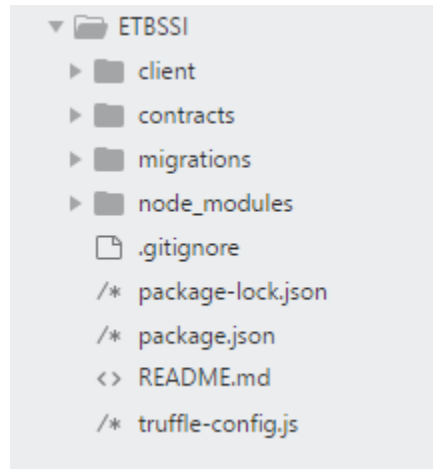
```

Figure 23 : IPFS API configuration

### III. Implementation details:

This is Bellow our main project development structure as we have main folders:

- **Client:** Contains all the scripts, files and node modules, which are all necessary four development and testing step of our Dapp.
- **Contracts:** Contains the Dapp smart contract.
- **Migrations:** Contains the migration script for the smart contract deployment.



- **Truffle-config.js:** A configuration script file where we have specified our development network and solidity compiler version.

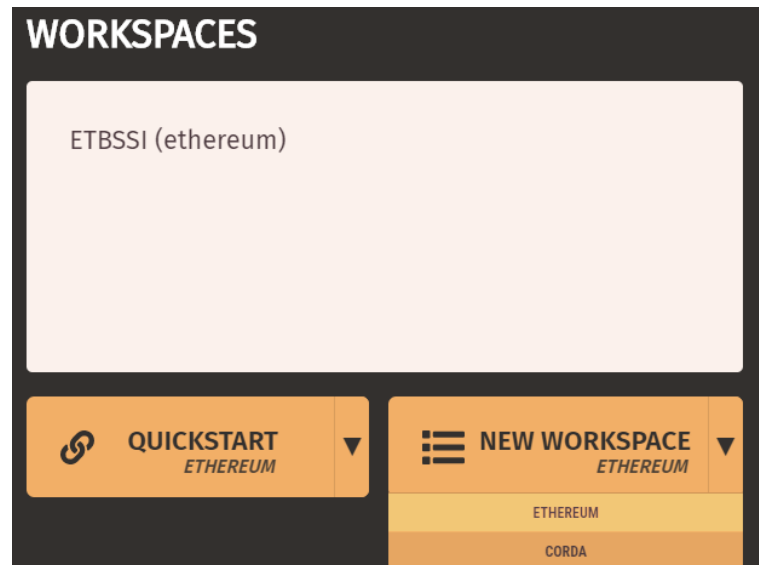
```
networks: {  
  development: {  
    host: "127.0.0.1",  
    port: 8545,  
    network_id: "5777"  
  },  
},  
compilers: {  
  solc: {  
    version: "^0.5.3",  
    docker: false  
  }  
}
```



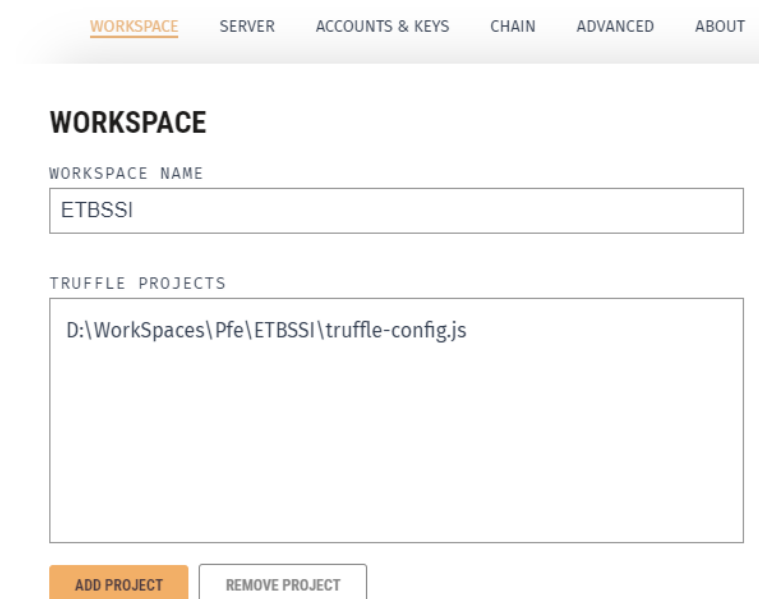
## 1. Ethereum Blockchain:

To deploy our Smart contract we used personal Ethereum Blockchain Ganache from Truffle to run tests, execute commands, and inspect state while controlling how the chain operates. To proceed we followed these steps:

- **Create an Ethereum Workspace:** We select Ethereum as a type of Blockchain.



- **Truffle config:** We add our Dapp truffle configuration path, which will allow deploying our Smart contract.



- **Blockchain Server:** To allow users access to our Blockchain network we configure its server interface in our network and port 8545.

WORKSPACE **SERVER** ACCOUNTS & KEYS CHAIN ADVANCED ABOUT

## SERVER

HOSTNAME  
192.168.1.103 - Wi-Fi ▼

PORT NUMBER  
8545

NETWORK ID  
5777

- **Ethereum Accounts:** We set up the number of accounts “10” to be generated and their balance 100 Ether.

WORKSPACE SERVER **ACCOUNTS & KEYS** CHAIN ADVANCED ABOUT

## ACCOUNTS & KEYS

ACCOUNT DEFAULT BALANCE  
20

TOTAL ACCOUNTS TO GENERATE  
10

- **Gas Price & Gas Price:** GAS is used to measure how many steps a transaction will need on EVM. It is straightforward: your transaction is complex, which means it needs more computation resources (such as CPU time and memory), you will need to pay more GAS. All opcodes on EVM will cost GAS. The smallest metric of GAS is wei, and  $1 \text{ eth} = 10^{18} \text{ wei}$   $\text{wei} = 10^9 \text{ gwei}$ .

Thus, the work of the Ethereum network is ensured [49]by:

- **Ether (ETH)** is the Ethereum network's native cryptocurrency, the second-largest by market cap on the crypto market.
- **Gas** is the unit of calculation that indicates the fee for a particular action or transaction.
- **The Gas Limit** is the maximum amount of Gas that a user is willing to pay for performing this action or confirming a transaction (a minimum of 21,000).
- **The price of Gas (Gas Price)** is the amount of Gwei that the user is willing to spend on each unit of Gas.

## GAS

GAS LIMIT

6721975

GAS PRICE

20000000000

- **Transaction tracking:** after Starting our Ethereum Blockchain, we can know list, track all transactions and check their details.

CURRENT BLOCK 15	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER	NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:8545	MINING STATUS AUTOMINING	WORKSPACE ETBSSI	SWITCH	⚙️
<b>TX HASH</b> <span style="font-family: monospace; font-size: 0.9em;">0×44c5dc120bfded626ae292a841180c79eb79dc4b937dbcc716cbfd84aaf82b17</span> <span style="float: right; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px; font-size: 0.8em;">CONTRACT CALL</span>									
FROM ADDRESS	TO CONTRACT ADDRESS			GAS USED	VALUE				
<span style="font-family: monospace; font-size: 0.8em;">0×63Ec6C7cC124ff762F91c71dA4E783999c537244</span>	<span style="font-family: monospace; font-size: 0.8em;">Doc2eth</span>			<span style="font-family: monospace; font-size: 0.8em;">266172</span>	<span style="font-family: monospace; font-size: 0.8em;">0</span>				
<b>TX HASH</b> <span style="font-family: monospace; font-size: 0.9em;">0×3f24ae6cebf216b608e5e4afd4f2061f818a2e8f13aab189bfbfb2a402dea4bc</span> <span style="float: right; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px; font-size: 0.8em;">CONTRACT CALL</span>									
FROM ADDRESS	TO CONTRACT ADDRESS			GAS USED	VALUE				
<span style="font-family: monospace; font-size: 0.8em;">0×63Ec6C7cC124ff762F91c71dA4E783999c537244</span>	<span style="font-family: monospace; font-size: 0.8em;">Doc2eth</span>			<span style="font-family: monospace; font-size: 0.8em;">329900</span>	<span style="font-family: monospace; font-size: 0.8em;">0</span>				
<b>TX HASH</b> <span style="font-family: monospace; font-size: 0.9em;">0×41b66754c28a23d716001d9038dbc38b8fa48dbf812c708ee9997b6868a260d</span> <span style="float: right; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px; font-size: 0.8em;">CONTRACT CALL</span>									
FROM ADDRESS	TO CONTRACT ADDRESS			GAS USED	VALUE				
<span style="font-family: monospace; font-size: 0.8em;">0×63Ec6C7cC124ff762F91c71dA4E783999c537244</span>	<span style="font-family: monospace; font-size: 0.8em;">Doc2eth</span>			<span style="font-family: monospace; font-size: 0.8em;">281172</span>	<span style="font-family: monospace; font-size: 0.8em;">0</span>				
<b>TX HASH</b> <span style="font-family: monospace; font-size: 0.9em;">0×2c5fd84d55710204eabdf121fc62d1be33aa87380f5a1746678cb0d179e69c21</span> <span style="float: right; border: 1px solid #ccc; border-radius: 5px; padding: 2px 5px; font-size: 0.8em;">CONTRACT CALL</span>									
FROM ADDRESS	TO CONTRACT ADDRESS			GAS USED	VALUE				
<span style="font-family: monospace; font-size: 0.8em;">0×d0b77Cfc5b2379c01CEfae8FFBe123d7815D1655</span>	<span style="font-family: monospace; font-size: 0.8em;">Doc2eth</span>			<span style="font-family: monospace; font-size: 0.8em;">329838</span>	<span style="font-family: monospace; font-size: 0.8em;">0</span>				

- **Blocks listing:** We can check all created blocks information details.

BLOCK 15			
GAS USED 266172	GAS LIMIT 6721975	MINED ON 2021-07-12 16:15:47	BLOCK HASH 0xa87af8b72a1f62ac8c0663320c08df72d8e7ce519efa40b6520ad2a61313dda2
TX HASH 0x44c5dc120bfded626ae292a841180c79eb79dc4b937dbcc716cbfd84aaf82b17			<a href="#">CONTRACT CALL</a>
FROM ADDRESS 0x63Ec6C7cC124ff762F91c71dA4E783999c537244	TO CONTRACT ADDRESS Doc2eth	GAS USED 266172	VALUE 0

- **Blockchain events:** All occurring events in block chain can be tracked using Ganache logs.

EVENT NAME FileShareUploaded	CONTRACT Doc2eth	TX HASH 0x44c5dc120bfded626ae292a841180c79eb79dc4b937dbcc716cbfd84aaf82b17	LOG INDEX 0	BLOCK TIME 2021-07-12 16:15:47
EVENT NAME FileUploaded	CONTRACT Doc2eth	TX HASH 0x3f24ae6cebf216b608e5e4afd4f2061f818a2e8f13aab189bfbfb2a402dea4bc	LOG INDEX 0	BLOCK TIME 2021-07-12 16:13:18
EVENT NAME FileShareUploaded	CONTRACT Doc2eth	TX HASH 0x41b66754c28a23d716001d9038dbcf38b8fa48dbf812c708ee9997b6868a260d	LOG INDEX 0	BLOCK TIME 2021-07-10 10:52:22
EVENT NAME FileShareUploaded	CONTRACT Doc2eth	TX HASH 0x2c5fd84d55710204eabdf121fc62d1be33aa87380f5a1746678cb0d17	LOG INDEX 0	BLOCK TIME 2021-07-10 10:08:11

## 2. Ethereum Smart Contract:

As we have opted for Ethereum framework for as a Blockchain. Our smart contract is programmed using Solidity language. We opted to use one Smart contract to handle our users and data files.

### 2.1. Structures:

In our Smart Contract, there is two main object structures, the users and the file object.

- User object structure:

```
struct User{
    string service;
    string firstname;
    string lastname;
    address userwallet;
}
```

A user object stores the different information about users and we mainly differ between users by using their wallet address because of it unicity.

- File object structure:

```
//File structure attributes
struct File {
    string fileId;
    string fileHash;
    uint fileSize;
    string fileType;
    string fileName;
    string key;
}
```

A File object stores the different information about data files and file ID and file hash are both important. Therefore, file ID helps to identify file hash on our Blockchain and the file hash to identify the file in the IPFS network .As result, without those two attributes no one can retrieve the data files.

- Mapping File structure:

Mappings is useful for associations to associate unique Ethereum addresses with associated value types. Mappings act as hash tables that consist of key types and corresponding value type pair. We used it to associate users with their files according to their wallet address to guaranty the proof of property.

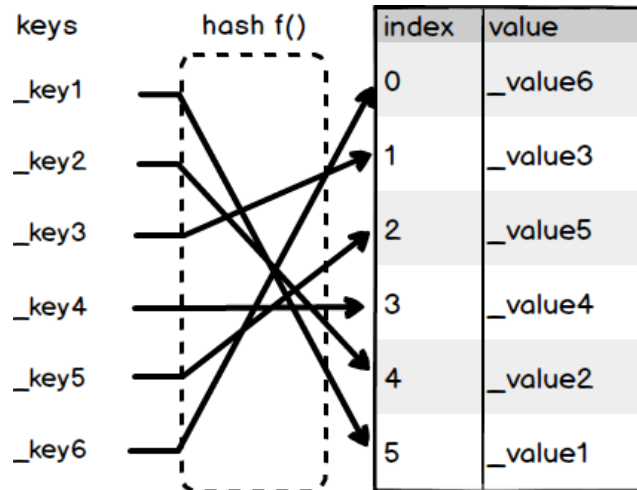


Figure 24 : Mapping Hash table[50]

As we know, altering smart contract rules is impossible, as result files property can be changed only by its main holder.

```
//get all the current user files hashes that have been uploaded by the current user
mapping(address => File[]) public users;
// get all shared files hashes that have been shared to the current user
mapping(address => File[]) public shares;
```

In our case, we have to mappings:

- One for user proper data files: it permit to retrieve all user files by linking them with address wallets.
- On for the files shared to the user: it permit to retrieve all files shared to a user by linking them with address wallets.

## 2.2. Functions:

There is two categories of functions; one is for data files and one for users.

### 2.2.1. User functions:

- **addUser**: a function that register the user into the Blockchain users list , then we can have a list of address wallets shared only between the Dapp users. Each three parameters take a string length greeter than 0 , if one of them does not respect this condition an error message is thrown according to the matching case ('Invalid Service name', 'Invalid First Name', 'Invalid Last Name') and this is assured by using **require** function that guarantee the validity before insertion.

```
function addUser(string memory _service,string memory _firstname,string memory _lastname,address _userwallet) public payable
    require(bytes(_service).Length > 0,"Invalid Service Name");
    require(bytes(_firstname).Length > 0,"Invalid First Name");
    require(bytes(_lastname).Length > 0,"Invalid Last Name");

    allUsers.push(User(_service,_firstname,_lastname,_userwallet));
    userservice=_service;
    ufirstname=_firstname;
    ulastname=_lastname;
    uaddress=_userwallet;

    emit UserAdded(_service,_firstname,_lastname,_userwallet);
}
```

Figure 25 : Smart Contract addUser function

- **getAllUserInfo**: a function that returns the attribute values of a user using its index. This function is set to view so that the state cannot be changed, only read is allowed. We used **require** to check the index validity.

```
function getAllUserInfo(uint _index) public view returns
(string memory userservice,string memory ufirstname,string memory ulastname,address uaddress) {

    require(_index>=0);
    User memory tmp = allUsers[_index];
    userservice=tmp.service;
    ufirstname=tmp.firstname;
    ulastname=tmp.lastname;
    uaddress=tmp.userwallet;

}
```

Figure 26 : Smart Contract getAllUserInfo function

- **getCountuser**: It's an utility function that returns the number of users registered addresses . This a view function cannot modify the state.

```
function getCountusers() public view returns(uint){
    return allUsers.length;
}
```

### 2.2.2. File functions:

- **uploadFile:** A function that register mainly data files hashes and their unique GUIDs (Globally Unique Identifier which is generated by the dapp server side) into the Blockchain mapped files, with their different attributes values (**user address, file Id, file Hash, file Size, file Type, file Name, key: concatenation of key+iv**). As result, we have a mapped list of files hashes and their information. All parameters have a condition check using **require** function, if one of them does not respect its condition an error message is thrown according to the matching case. This function emit the upload event and returns the current uploaded file attributes values.

```
// function of uplading a file
function uploadFile(address _address,string memory _fileId,string memory _fileHash,
uint _fileSize, string memory _fileType, string memory _fileName,string memory _key)

    require(bytes(_fileHash).length > 0,"Invalid File Hash");
    require(bytes(_fileId).length > 0,"Invalid File ID");
    require(bytes(_fileType).length > 0,"Invalid File Type");
    require(bytes(_fileName).length > 0,"Invalid Service Name");
    require(_address != address(0),"Invalid address");
    require(_fileSize>0,"Invalid File Size");
    require(bytes(_key).length > 0,"Invalid key size");
    //require(bytes(_iv).length > 0,"Invalid key size");

    users[_address].push(File(_fileId,_fileHash, _fileSize, _fileType, _fileName,_key));

    emit FileUploaded(_fileId,_fileHash, _fileSize, _fileType, _fileName);

    uint length = users[_address].length;

    File memory file = users[_address][length-1];

    return (file.fileId,file.fileHash,file.fileSize,file.fileType,file.fileName);
}
```

Figure 27 : Smart Contract uploadFile function

- **removeHash:** This functions allow us to remove a file hash from the Blockchain by funding the mapping list of the user using its address. After that we search for the hash using GUID by looping into the list and compare id bits with **keccak256** and **abi.encodePacked**



```

//remove file hash from the blockchain
function removeHash(string memory _fileId, address _address) public {
    File[] storage f = users[_address];
    for (uint i = 0; i < f.length; i++) {
        if (keccak256(abi.encodePacked(f[i].fileId)) == keccak256(abi.encodePacked(_fileId))) {
            delete users[_address][i];
        }
    }
}
}
//share file

```

Figure 28 : Smart contract remove hash function

- **getFilesOfUser:** A function that returns the attribute values of a data file user using its index and the address wallet of user. This function is set to view so that the state cannot be changed, only read is allowed. We used require to check the index validity.

**NB:** The function does not return the cryptographic keys of the file.

```

function getFilesOfUser(uint _index, address _address)
public view returns(string memory, string memory, uint, string memory, string memory) {
    require(_index >= 0);
    File memory file = users[_address][_index];
    return (file.fileId, file.fileHash, file.fileSize, file.fileType, file.fileName);
}

```

Figure 29 : Smart contract get file function

- **uploadShareFile:** A function allow to share a data file information and credential access (key). It get data file information from the sender mapping file structure and transfer them to the receiver mapping shared file structure.

```
//share file
function uploadShareFile(address _to,string memory _fileId,string memory _fileHash, uint _fileSize,
string memory _fileType, string memory _fileName,string memory _key) public returns(string memory

require(bytes(_fileHash).length > 0,"Invalid File Hash");
require(bytes(_fileId).length > 0,"Invalid File ID");
require(bytes(_fileType).length > 0,"Invalid File Type");
require(bytes(_fileName).length > 0,"Invalid Service Name");
require(_to!=address(0),"Invalid receiver address");
require(_fileSize>0,"Invalid File Size");
require(bytes(_key).length > 0,"Invalid key size");

shares[_to].push(File(_fileId,_fileHash, _fileSize, _fileType, _fileName,_key));
emit FileShareUploaded(_fileId,_fileHash, _fileSize, _fileType, _fileName);

uint length = shares[_to].length;
File memory file = shares[_to][length-1];

return (file.fileId,file.fileHash,file.fileSize,file.fileType,file.fileName);
}
```

Figure 30 : Smart contract upload file function

- **getFilekey:** An essential function that retrieve a data file key. The file key is a concatenation of the aes256 key and aes256 iv. The function require the correct user address to be connected with his Ethereum account, otherwise the file key cannot be retrieved.

```
function getFilekey(uint _index, address _address)
public view returns(string memory) {

    require(_index>=0);

    File memory file = users[_address][_index];

    return file.key;

}
```

Figure 31 : Smart contract get file key function

- **getCount:** It is an utility function that returns the number of uploaded files hashes from a specific address user. It is view function that cannot modify the state.

```
function getCount(address _address) public view returns(uint){
    return users[_address].length;
}
```

- **getShareCount:** It is an utility function that returns the number of shared files hashes which a specific address user have access. It is a view function that cannot modify the state.

```
function getShareCount(address _address) public view returns(uint){
    return shares[_address].length;
}
```

### 2.3. Deployment:

We deployed our smart contract to ganache Ethereum network. The figure below shows the deployment details.

```
~/_deploy_contracts.js
=====

Replacing 'Doc2eth'
-----
> transaction hash:    0xd62dd3a9c55c21261fbd098b72109cb66e30c9c4f71c1db66b418846d16b374e
> Blocks: 0           Seconds: 0
> contract address:   0x0b987c15B1F901061697ca27AAaCaA114beAB4C1
> block number:       3
> block timestamp:    1623920188
> account:            0x6015f4F987cbdf36e3aeA00361464dEE4B60BCD6
> balance:            99.8926456
> gas used:           5161204 (0x4ec0f4)
> gas price:          20 gwei
> value sent:         0 ETH
> total cost:         0.10322408 ETH

> Saving migration to chain.
> Saving artifacts
-----
> Total cost:         0.10322408 ETH

Summary
=====
> Total deployments:  2
> Final cost:        0.10650758 ETH
```

Figure 32 : Smart contract deployment

### 2.4. Application Binary Interface generation::

The Application Binary Interface (ABI) is a specification in the form of a JSON that describes the contract. The ABI is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the Blockchain and for contract-to-contract interaction[51]. Data is encoded according to its type, as described in this specification. The encoding is not self describing and thus requires a schema in order to decode.

After compilation of smart **contract.sol** using a solidity compiler (eg., solcjs [52]), **contract.bin** file will be generated and contains contract's bytecode. After, we need to use solc to create ABI. The generated file is a contract template used for interface containing available methods in the contract. Using the following command to generate ABI file: **Solcjs contract.sol --abi** [53].

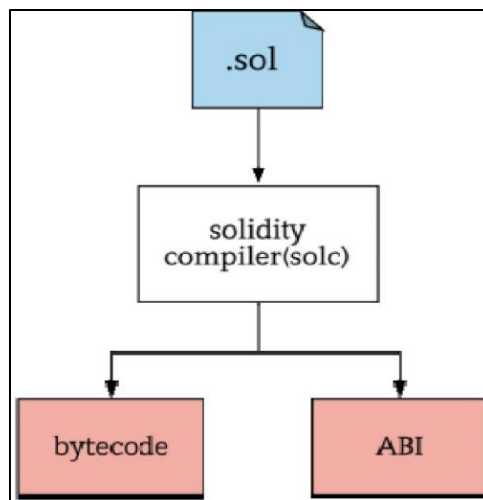


Figure 33 : Compilation of smart contract[54]

## 2.5. Read data from smart contract:

In order to read data from smart contracts with Web3.js, we use **web3.eth.Contract()** which takes two arguments : SC abi and SC address. And now we can use SC methods with **contract.methods.methodname().**[54]

### 3. AES256 data file encryption:

- We use aes256-ctr [55], which stands for aes256 counter mode. It takes a key of 16 bytes and an initialization vector of 8 bytes for counter mode. It allows good parallelization [56] (ie. speed).

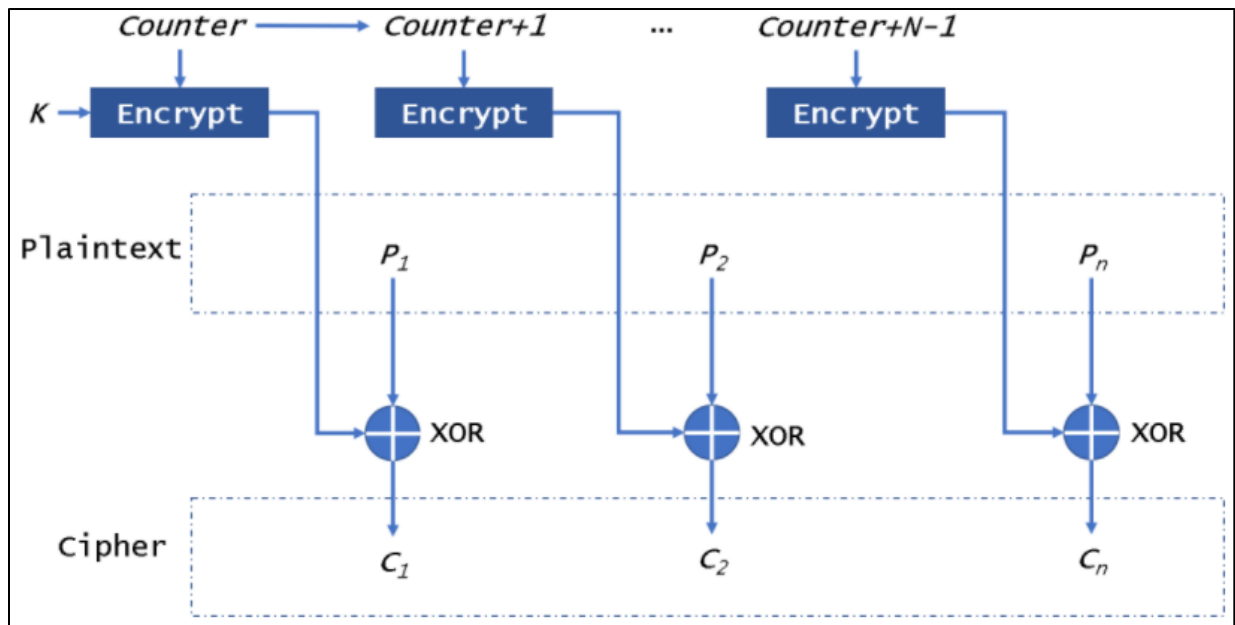


Figure 34 : AES256-ctr encryption process[57]

- To encrypt any type of data file, we convert it into an array buffer after reading the file. The ArrayBuffer object is used to represent a generic, fixed-length raw binary data buffer.

```
const file = e.target.files[0];
let reader = new window.FileReader();
reader.readAsArrayBuffer(file);
reader.onloadend = () => convertToBuffer(reader, file.type, file.name);
};

const convertToBuffer = async (reader, type, name) => {
  const buffer = await Buffer.from(reader.result);
```

Figure 35 : Convert data file to buffer



- We get as a result a new buffer with encrypted values.

```

Buffer encryptAES                                Dashboard.jsx:235
Uint8Array(342023) [105, 251, 244, 191, 114, 12
1, 113, 93, 7, 45, 159, 226, 81, 44, 143, 104, 1
2, 99, 29, 254, 244, 176, 80, 26, 141, 58, 213,
39, 64, 106, 68, 244, 37, 175, 74, 35, 2, 78, 18
6, 201, 115, 242, 125, 28, 72, 46, 235, 160, 21
▶ 5, 184, 93, 95, 115, 231, 248, 45, 149, 117, 14
8, 142, 147, 44, 70, 10, 52, 13, 129, 137, 160,
161, 13, 124, 97, 191, 122, 224, 133, 218, 243,
199, 112, 253, 159, 72, 110, 214, 193, 66, 136,
107, 222, 157, 38, 213, 113, 172, 86, 108, 46,
5, ...]

```

Figure 39 : Encrypted data file buffer

- For decryption, we use aes256 function with the according data file key and iv. As the function take the encrypted buffer and decrypt it.

```

export const decryptAES = (buffer, secretKey, iv) => {
  const decipher = crypto.createDecipheriv('aes-256-ctr', secretKey, iv);
  const data = decipher.update(buffer);
  const decrypted = Buffer.concat([data, decipher.final()]);
  return decrypted;
};

```

Figure 40 : Decrypt aes256-ctr function

#### 4. Transactions and Functionalities:

We named our Dapp Doc2Eth and here we will present all interfaces and functionalities using algorithms and Dapp screenshots.

##### 3.1. Authentication:

- A user must access firstly his wallet by introducing his password to unlock metamask.

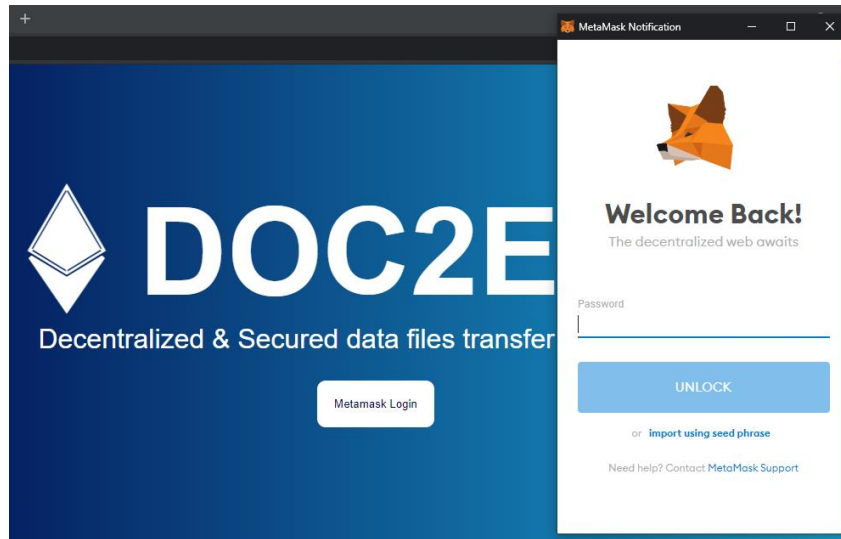


Figure 41 : Metamask access

- Then user get his private key to access Ethereum network. In our case, we used ganache to generate accounts with private keys. The user introduce it in metamask and connect to the Blockchain network.



Figure 42 : Ganache account private key



- The user can confirm the connection between the Dapp and Ganache.

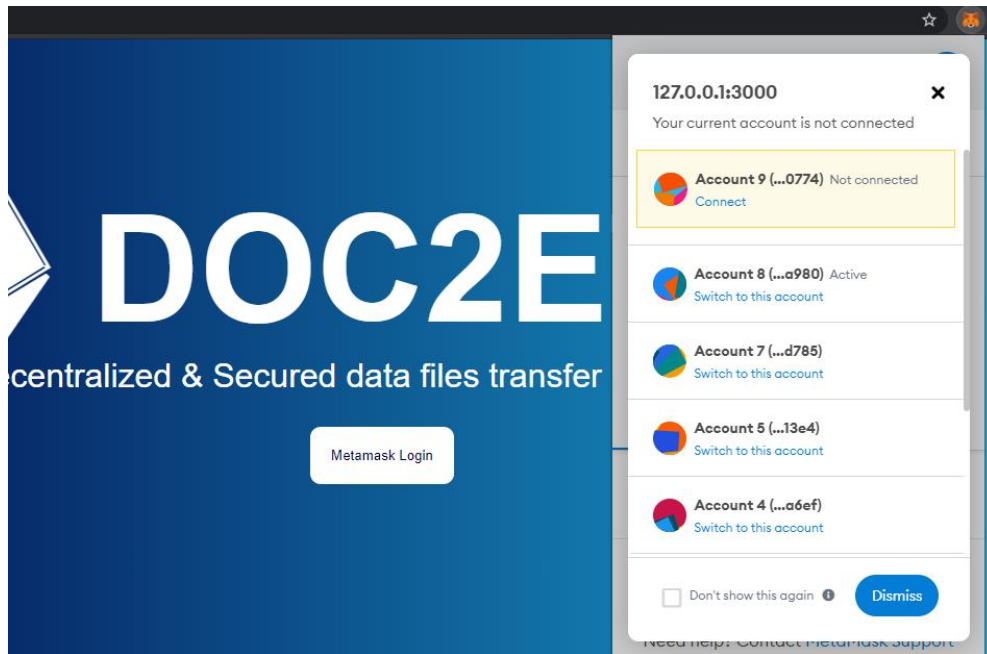


Figure 43 : Connect Dapp to ganache

- We used Web3 to retrieve Ethereum account (Ganache) information and connection state to the Dapp. As we get the account, network ID, deployed Network and the most important creating the smart contract instance.
- To create the smart contract instance, we use **Web3.eth.Contract()**. As it uses the ABI that allows the Dapp to interact with the smart contract, deployed Network and its address.

```
const web3 = await getWeb3();
const accounts = await web3.eth.getAccounts();
const networkId = await web3.eth.net.getId();
const deployedNetwork = Doc2eth.networks[networkId];
const instance = new web3.eth.Contract(
  Doc2eth.abi,
  deployedNetwork && deployedNetwork.address
);
```

Figure 44 : Handling Ethereum network connection

- If the Dapp is not connected to ganache Ethereum network using metamask, it throws an error as it cannot handle authentication and cannot read **web3.eth** instance.

```
✖ ▶ TypeError: Cannot read property 'eth' of undefined
    at authHandler (Home.jsx:33)
    index.js:1451
```

- If the user does not have an Ethereum account from the network where the smart contract has been deployed, the Dapp throws an error.

```
✖ ▶ Error: This contract object doesn't have address set yet, please set an address first.
    at Object.o._processExecuteArguments (web3.min.js:29729)
    at Object.o._executeMethod (web3.min.js:29733)
    at authHandler (Home.jsx:47)
    index.js:1451
```

### 3.2. Registration:

- When a user login to the Dapp using metamask, it checks if he is registered to list of users, which is created by the smart contract using the code in the bellow picture. The Dapp uses the smart contract function **getAllUserinfo()** .

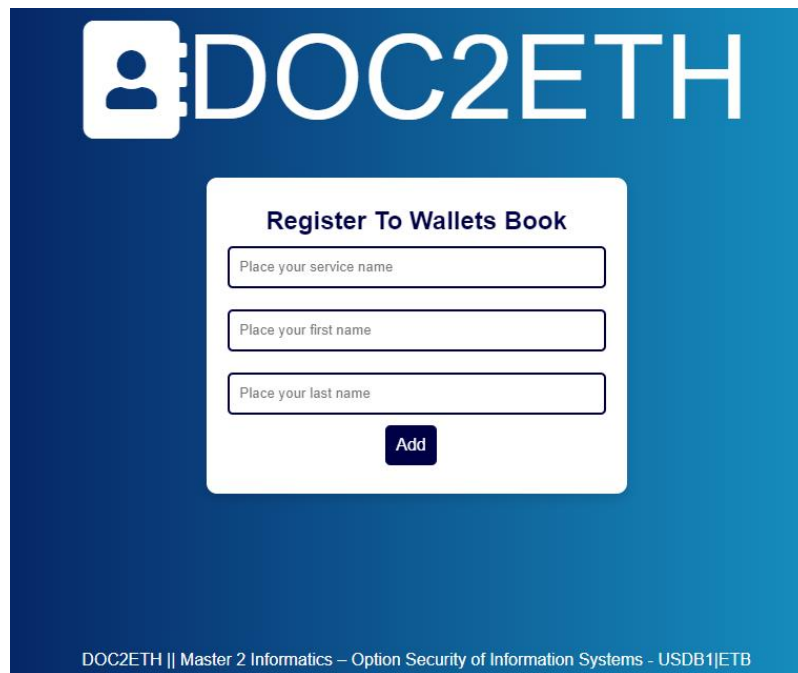
```
const usersCount = await instance.methods.getCountusers().call();

for (var userIndex = 0; userIndex < usersCount; userIndex++) {
  const USER = await instance.methods.getAllUserInfo(userIndex).call();

  allusers.push(USER);
}

const userExists = await allusers.some(user => user.uaddress === accounts[0]);
if(userExists) {
  history.push("/dashboard");
}else{
  history.push("/register");
  alert.show("Please add your wallet address to DOC2ETH!");
}
```

- If the user is not registered, he is redirected to the registration form.



The image shows a registration form titled "Register To Wallets Book" on a dark blue background. The form is centered and contains three input fields: "Place your service name", "Place your first name", and "Place your last name". Below the fields is a dark blue "Add" button. At the top left of the form area is a white icon of a person with a gear, and the text "DOC2ETH" is displayed in large white letters. At the bottom of the form area, the text "DOC2ETH || Master 2 Informatics – Option Security of Information Systems - USDB1|ETB" is visible.

Figure 45 : Registration form

- We use a react module **react-inputs-validation** and regular expression to control if any input field is empty and omit the special characters.



The image shows the same registration form as in Figure 45, but with validation messages. The "Place your service name" field is empty, and the message "Service cannot be empty" is displayed below it. The "Place your first name" field is empty, and the message "Firs tname cannot be empty" is displayed below it. The "Place your last name" field is empty, and the message "Last name cannot be empty" is displayed below it. The "Add" button is still present at the bottom.

Figure 46 : Register input validation

- After filling the form, the user can submit his information. The Dapp call the **adduser** function bellow. To finalize the registration, the user must confirm the transaction

```
const adduser = await state.contract.methods
  .addUser(
    _service,
    _firstname,
    _lastname,
    _adress
  ).send({ from: state.accounts[0] });
```

➔

GAS FEE 0.004848  
No Conversion Rate Available

Gas Price (GWEI) Gas Limit

---

AMOUNT + GAS FEE

TOTAL 0.004848  
No Conversion Rate Available

Reject
Confirm

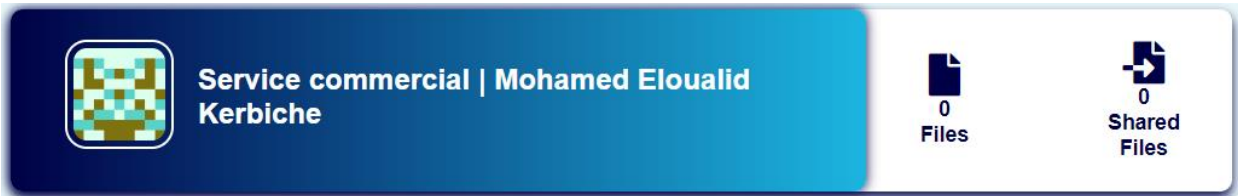
displayed by metamask plugin. The register pays 0.004848 ether to confirm the transaction.

**Figure 47 : Register transaction**

- Now, the user can login and directly access the Dapp main page and use its functionalities.

### 3.3. Main interface:

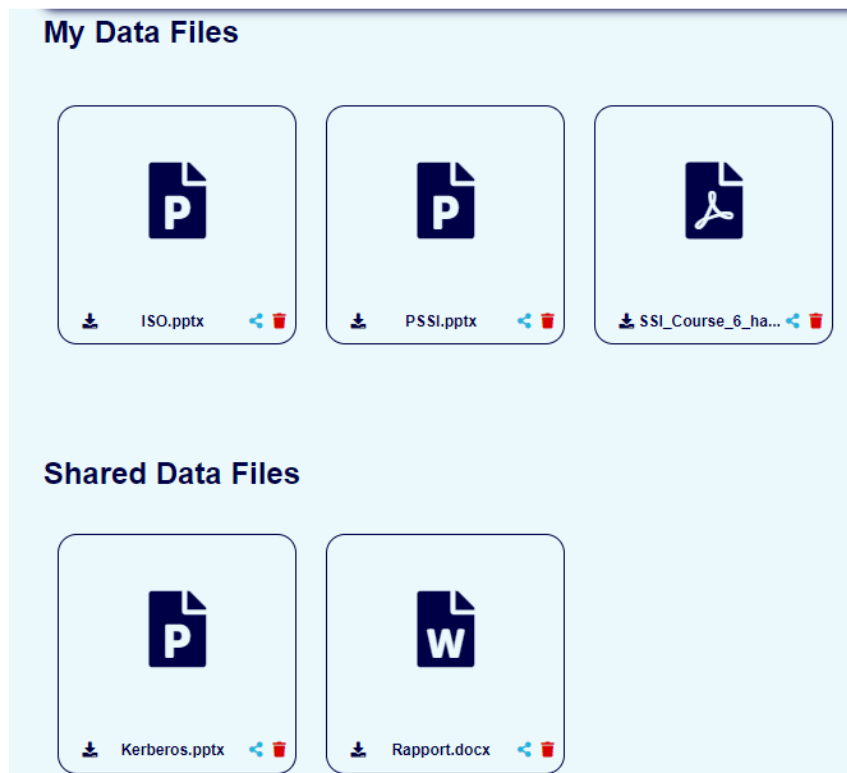
- The Dapp displays the service name, user first and last name.



- The Dapp displays the number of files and shared files.



- The user can visualize the list of his data files, and the list of the shared data files with him.



### 3.4. Upload data file:

- To upload a data file, the user click on the right bottom corner plus button to open the file explorer and choose a data file.

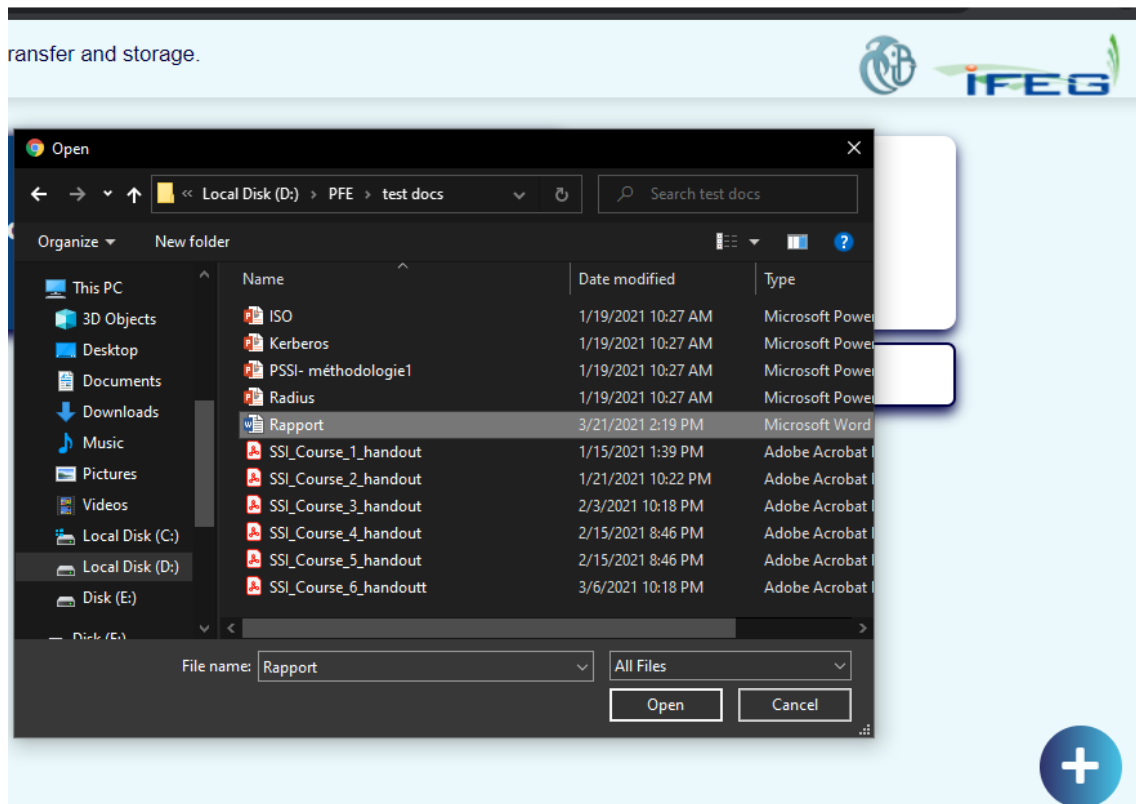


Figure 48 : Dapp upload file

- After that, the user can confirm upload or cancel it as it shows in the picture billow.



- After confirming upload, metamask asks to confirm upload transaction which costs 0.010344 ether.

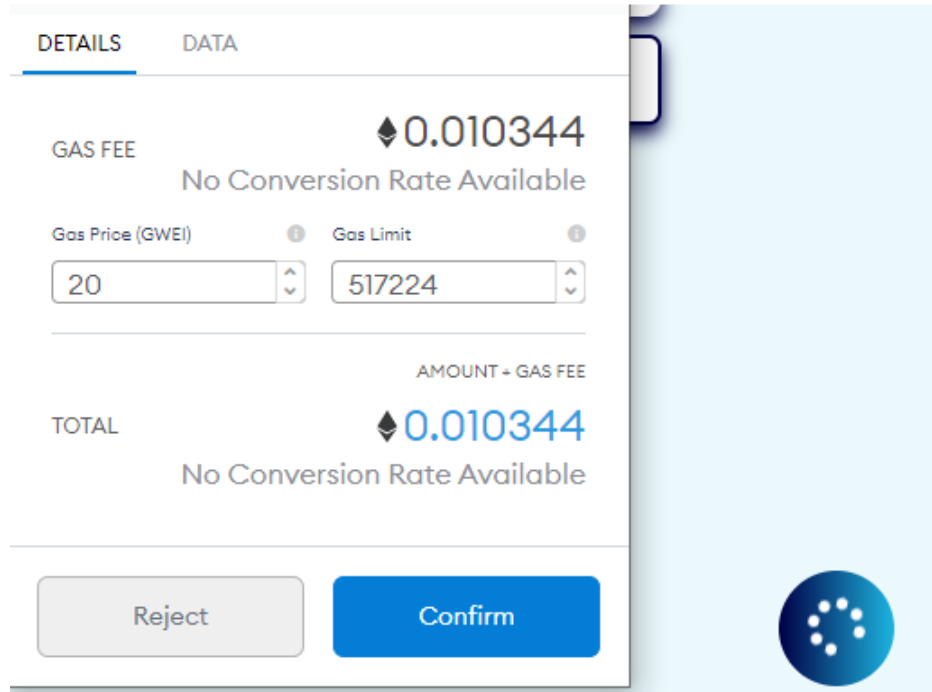
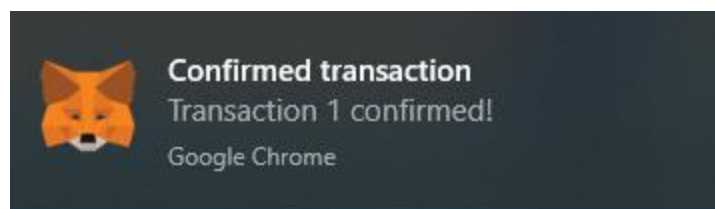


Figure 49 : Confirm upload transaction

- After confirmation and a successful transaction execution, metamask shows a message of transaction confirmation.



- We have also limited the type of files that can be uploaded in the IPFS for a better control and security. As files with no common extension cannot be uploaded.







### 3.6. Retrieve data file:

- To retrieve the desired data file, the user have to click on the download button.



- When retrieving data file, we have noticed that IPFS return the data file buffer by ordered blocks. Therefore, we implemented a merging function to build a unique buffer array of our data file to allow decryption operation.

```
export const mergearrays = (arraylist) => {
  // Get the total length of all arrays.
  var length = 0;
  arraylist.forEach(item => {
    length += item.length;
  });

  // Create a new array with total length and merge all source arrays.
  var mergedArray = new Uint8Array(length);
  var offset = 0;
  arraylist.forEach(item => {
    mergedArray.set(item, offset);
    offset += item.length;
  });

  return mergedArray;
};
```

- If someone tries to access the file using its hashed only without decrypting the data file with the Dapp, he will get an encrypted unusable data file.

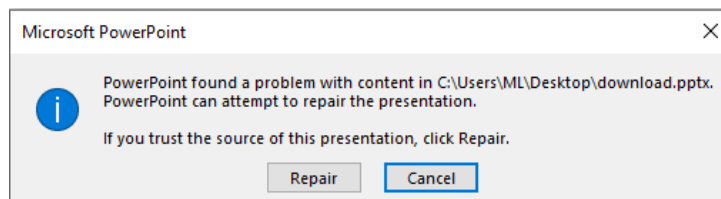
**Exp:** A test with a power point data file which ipfs hash link is

gateway.ipfs.io/ipfs/QmbfzAKBvVojhmBbZHg5jbEZ5pYqQVqM5mvMwBnztg91t8

- When trying to access the file directly, the browser sometimes fails to retrieve it as it show 504-timeout error.



- In the case, the data file can be downloaded but no OS programs can open it.



### 3.7. Delete data file:

- To delete the desired data file from his Blockchain mapping structure list, the user have to click on the delete button and confirm the transaction.

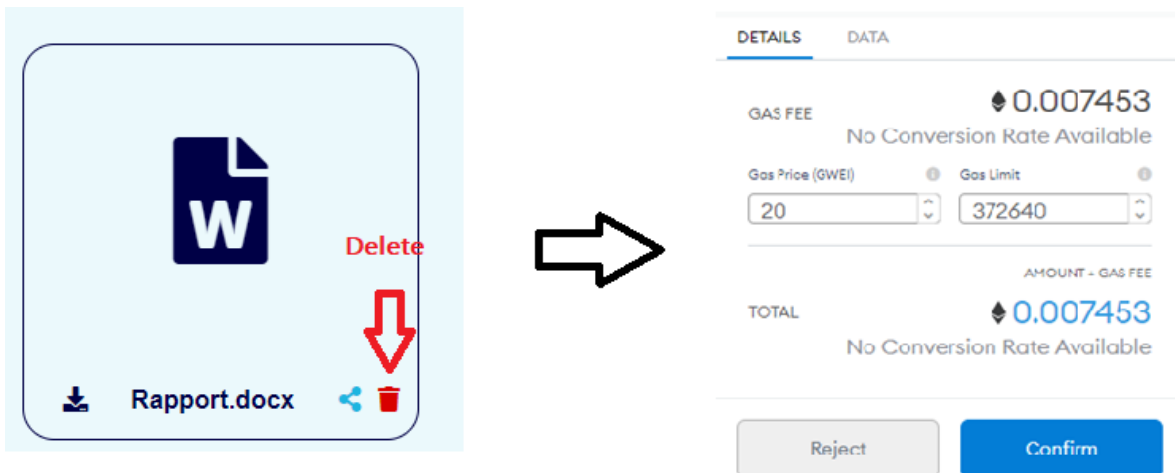


Figure 53 : Delete data file

#### IV. Conclusion:

In this chapter, we have presented our implementation process. As a first step, we presented our development environment and tools, which have used to build our Dapp and make fully functional. After that, we exposed two main component of our Dapp, which are the structure and the functions of our smart contract the encryption solution aes256 we used to secure our data files. We finished this chapter by presenting the Dapp different functionalities and interfaces with different test cases.

## GENERAL CONCLUSION

Blockchain has proven that is an especially promising and revolutionary technology in security field because it helps reduce risk, stamps out fraud and brings transparency. Adding to that its flexibility to work with other decentralized tools and concepts allowed to expand its fields of application.

Our thesis main purpose was to assure data transfer in decentralized application using Blockchain technology. As we aimed to use Blockchain cryptography and secured transactions property, to create a secured Dapp for Technical School of Blida (SONELGAZ) to assure data transfer and storage in a secure and optimal way.

To concretize our solution we developed a Smart Contract on Ethereum network to ensure a secured transfer and storage of data files access information. We used also Ethereum accounts authentication process to secure our Dapp access. As it is impossible to store huge data files on Ethereum Blockchain due to the need of high cost and resources, we opted to store the data files on InterPlanetary File System, which is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. We also added a cryptographic solution that encrypt data files using AES256-ctr algorithm and store user keys and initialization vectors into the Blockchain.

We used Truffle to deploy and test our smart contract and Ganache as Ethereum network test. We opted for Node JS as our Dapp application server and React JS as client side. After testing our Dapp, we concluded that only users who have an Ethereum account with the initialized network could access the Dapp and upload data files. In addition, only the users with hashed link, AES256 Key and AES256 iv can retrieve, share data files.

### Perspectives:

In the future, we could improve our Dapp in some aspects listed below:

- Use a decentralized network of oracles to retrieve and verify external data.
- Improve and upgrade the system to support all the three different Schools of SONELGAZ.
- Upgrade the Dapp to editing on data files based on access right

# Bibliography

1. Hugoson, M.-Å. Centralized versus Decentralized Information Systems. in History of Nordic Computing 2. 2009. Berlin, Heidelberg: Springer Berlin Heidelberg.
5. Zarrin, J., R. Aguiar, and J. Barraca, Resource discovery for distributed computing systems: A comprehensive survey. *Journal of Parallel and Distributed Computing*, 2017. 113.
7. 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.
8. Sanka, A.I., et al., A survey of breakthrough in blockchain technology: Adoptions, applications, challenges and future research. *Computer Communications*, 2021. 169: p. 179-201.
10. Patel, V., et al. A Review on Blockchain Technology: Components, Issues and Challenges. 2020. Singapore: Springer Singapore.
12. Subburaj, J., et al. Block Chain Technology: An Outline. in *ICDSMLA 2019*. 2020. Singapore: Springer Singapore.
13. Singhal, B., G. Dhameja, and P.S. Panda, How Blockchain Works, in *Beginning Blockchain: A Beginner's Guide to Building Blockchain Solutions*, B. Singhal, G. Dhameja, and P.S. Panda, Editors. 2018, Apress: Berkeley, CA. p. 31-148.
14. Hellwig, D., G. Karlic, and A. Huchzermeier, Blockchain Foundations, in *Build Your Own Blockchain: A Practical Guide to Distributed Ledger Technology*, D. Hellwig, G. Karlic, and A. Huchzermeier, Editors. 2020, Springer International Publishing: Cham. p. 3-27.
15. Prasad, R. and V. Rohokale, Blockchain Systems and Cryptocurrencies, in *Cyber Security: The Lifeline of Information and Communication Technology*, R. Prasad and V. Rohokale, Editors. 2020, Springer International Publishing: Cham. p. 274-277.
16. Sharma, A., et al., Introduction to Blockchain, in *Blockchain Applications in IoT Ecosystem*, T. Choudhury, et al., Editors. 2021, Springer International Publishing: Cham. p. 1-14.
17. Srیمان, B., S. Ganesh Kumar, and P. Shamili. Blockchain Technology: Consensus Protocol Proof of Work and Proof of Stake. in *Intelligent Computing and Applications*. 2021. Singapore: Springer Singapore.
18. Mohamed, K.S., New Trends in Cryptography: Quantum, Blockchain, Lightweight, Chaotic, and DNA Cryptography, in *New Frontiers in Cryptography: Quantum, Blockchain, Lightweight, Chaotic and DNA*, K.S. Mohamed, Editor. 2020, Springer International Publishing: Cham. p. 65-87.
19. Upadhyay, N., Rule of Code, in *UnBlock the Blockchain*, N. Upadhyay, Editor. 2019, Springer Singapore: Singapore. p. 11-24.
20. Gayvoronskaya, T. and C. Meinel, Technical Basics for a Better Understanding of Blockchain Technology, in *Blockchain: Hype or Innovation*, T. Gayvoronskaya and C. Meinel, Editors. 2021, Springer International Publishing: Cham. p. 15-33.
23. Zheng, Z., et al., An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 2020. 105: p. 475-491.

24. Sillaber, C. and B. Waihl, Life Cycle of Smart Contracts in Blockchain Ecosystems. *Datenschutz und Datensicherheit - DuD*, 2017. 41(8): p. 497-500.
25. Vigliotti, M.G. and H. Jones, Smart Contracts, in *The Executive Guide to Blockchain: Using Smart Contracts and Digital Currencies in your Business*, M.G. Vigliotti and H. Jones, Editors. 2020, Springer International Publishing: Cham. p. 133-149.
26. Nzuva, S., *Smart Contracts Implementation, Applications, Benefits, and Limitations*. *Journal of Information Engineering and Applications*, 2019.
27. Zhou, Y., et al. Erays: reverse engineering ethereum's opaque smart contracts. in *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018.
28. Ciatto, G., et al. From the Blockchain to Logic Programming and Back: Research Perspectives. in *WOA*. 2018.
29. Kasampalis, T., et al., Iele: An intermediate-level blockchain language designed and implemented using formal semantics. 2018.
30. Chen, T., et al., Towards saving money in using smart contracts, in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. 2018, Association for Computing Machinery: Gothenburg, Sweden. p. 81–84.
31. Albert, E., et al. EthIR: A Framework for High-Level Analysis of Ethereum Bytecode. in *Automated Technology for Verification and Analysis*. 2018. Cham: Springer International Publishing.
32. Brent, L., et al., Vandal: A scalable security analysis framework for smart contracts. 2018.
33. Fröwis, M. and R. Böhme. In *Code We Trust? in Data Privacy Management, Cryptocurrencies and Blockchain Technology*. 2017. Cham: Springer International Publishing.
34. Nikolić, I., et al., Finding The Greedy, Prodigal, and Suicidal Contracts at Scale, in *Proceedings of the 34th Annual Computer Security Applications Conference*. 2018, Association for Computing Machinery: San Juan, PR, USA. p. 653–663.
35. Mavridou, A. and A. Laszka. Designing secure ethereum smart contracts: A finite state machine based approach. in *International Conference on Financial Cryptography and Data Security*. 2018. Springer.
36. Bragagnolo, S., et al. SmartInspect: solidity smart contract inspector. in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. 2018.
37. Mohanty, D., *Advanced Programming in Oraclize and IPFS, and Best Practices, in Ethereum for Architects and Developers: With Case Studies and Code Samples in Solidity*, D. Mohanty, Editor. 2018, Apress: Berkeley, CA. p. 151-179.
38. Vimal, S. and S.K. Srivatsa, A new cluster P2P file sharing system based on IPFS and blockchain technology. *Journal of Ambient Intelligence and Humanized Computing*, 2019.
39. Quasim, M.T., et al., *Blockchain Frameworks, in Decentralised Internet of Things: A Blockchain Perspective*, M.A. Khan, et al., Editors. 2020, Springer International Publishing: Cham. p. 75-89.
42. Manoj Athreya, A., et al. Peer-to-Peer Distributed Storage Using InterPlanetary File System. in *Advances in Artificial Intelligence and Data Engineering*. 2021. Singapore: Springer Singapore.
43. Sanjuán, H., et al., Merkle-CRDTs: Merkle-DAGs meet CRDTs. 2020. abs/2004.00107.

44. Tennant, P.W.G., et al., Use of directed acyclic graphs (DAGs) to identify confounders in applied health research: review and recommendations. *International Journal of Epidemiology*, 2020. 50(2): p. 620-632.
45. Huang, H., et al., When Blockchain Meets Distributed File Systems: An Overview, Challenges, and Open Issues. *IEEE Access*, 2020. 8: p. 50574-50586.
46. Huang, H.-S., T.-S. Chang, and J.-Y. Wu, A Secure File Sharing System Based on IPFS and Blockchain, in *Proceedings of the 2020 2nd International Electronics Communication Conference*. 2020, Association for Computing Machinery: Singapore, Singapore. p. 96–100.
48. Lee, W.-M., Using the web3.js APIs, in *Beginning Ethereum Smart Contracts Programming: With Examples in Python, Solidity, and JavaScript*, W.-M. Lee, Editor. 2019, Apress: Berkeley, CA. p. 169-198.
53. Zheng, G., et al., Application Binary Interface (ABI), in *Ethereum Smart Contract Development in Solidity*, G. Zheng, et al., Editors. 2021, Springer Singapore: Singapore. p. 139-158.
54. Panda, S.K. and S.C. Satapathy. An Investigation into Smart Contract Deployment on Ethereum Platform Using Web3.js and Solidity Using Blockchain. in *Data Engineering and Intelligent Computing*. 2021. Singapore: Springer Singapore.
55. Rogawski, M., K. Gaj, and E. Homsirikamol, A high-speed unified hardware architecture for 128 and 256-bit security levels of AES and the SHA-3 candidate Grøstl. *Microprocessors and Microsystems*, 2013. 37(6): p. 572-582.
56. Fei, X., et al., Analysis of energy efficiency of a parallel AES algorithm for CPU-GPU heterogeneous platforms. *Parallel Computing*, 2020. 94-95: p. 102621.



# Webography

2. Hertig, A. What Is a Decentralized Application? [cited 2021 13/05/2021]; Available from: <https://www.coindesk.com/learn/ethereum-101/what-is-a-decentralized-application-dapp>.
3. Decentralized Vs Centralized Systems. [cited 2021 17/04/2021]; Available from: <https://medium.com/nhct-nanohealth-care-token/decentralized-vs-centralized-systems-cb31b95928c5>.
4. Comparison – Centralized, Decentralized and Distributed Systems. [cited 2021 17/04/2021]; Available from: <https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems/>.
6. Blockchain on AWS. [cited 2021 05/04/2021]; Available from: <https://aws.amazon.com/blockchain/>.
9. Blockchain key characteristics and the conditions to use it as a solution. Available from: <https://medium.com/swlh/blockchain-characteristics-and-its-suitability-as-a-technical-solution-bd65fc2c1ad1>.
11. Blockchain Architecture Basics: Components, Structure, Benefits & Creation. [cited 2021 13/07/2021]; Available from: <https://mlsdev.com/blog/156-how-to-build-your-own-blockchain-architecture>.
21. INTRODUCTION TO SMART CONTRACTS. [cited 2021 11/05/2021]; Available from: <https://ethereum.org/en/developers/docs/smart-contracts/>.
22. IBM. Smart contracts. [cited 2021 11/05/2021]; Available from: <https://www.ibm.com/topics/smart-contracts>.
40. Best Blockchain Frameworks You Should Know About. [cited 2021 17/05/2021]; Available from: <https://merehead.com/blog/blockchain-frameworks-you-should-know-about/>.
41. Juan Benet inventor of IPFS. Available from: <https://research.protocol.ai/authors/juan-benet/>.
47. WHAT IS LIBP2P? [cited 2021 24/05/2021]; Available from: <https://docs.libp2p.io/introduction/what-is-libp2p/>.
49. What Are Gas, Gas Limit, and Gas Price in the Ethereum Network? [cited 2021 28/05/2021]; Available from: <https://decenter.org/en/category/mining>.
50. Mappings in Solidity Explained in Under Two Minutes. [cited 2021 10-06-2021]; Available from: <https://medium.com/upstate-interactive/mappings-in-solidity-explained-in-under-two-minutes-ecba88aff96e>.
51. Contract ABI Specification. [cited 2021 18/05/2021]; Available from: <https://docs.soliditylang.org/en/v0.5.3/abi-spec.html>.
52. solc-js. [cited 2021 18/05/2021]; Available from: <https://www.npmjs.com/package/solc>.
57. The difference in five modes in the AES encryption algorithm. [cited 2021 10-06-2021]; Available from: <https://www.highgo.ca/2019/08/08/the-difference-in-five-modes-in-the-aes-encryption-algorithm/>.