UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Blockchain-Based E-Voting System

David Vives Conesa

Director: Ruben Tous Liesa
June 2022

Specialization in Information Technologies

Contents

Abstract

The purpose of this present paper is to study the state of electronic voting. This term has gained strength in the past few years, and has the intention to improve election processes on paper, by increasing efficiency and reducing errors that may come from this process, in which a lot of people are involved. Blockchain has gained a lot of popularity and has some characteristics that can help develop these systems by giving confidence to the people involved in the elections.

This project is divided into 5 main characters. The first one explains what the project is about, the problem that has to be solved, the scope, and what are the expected results. The second one is how the project will be planned and developed, together with the budget and sustainability reports. In the third chapter, there is a historical and theoretical explanation of voting and e-voting, its implementations, advantages, and disadvantages. Following in chapter 4, the same theoretical explanation is done with blockchains. The aim of chapters 4 and 5 is to get a good background and understanding of the main concepts before diving into a practical part or any conclusions. Finally, in chapter 5, an electronic voting platform based on an Ethereum blockchain is implemented.

In the end, conclusions about the overall project are given and the corresponding references are attached.

Resumen

El objetivo del presente trabajo es estudiar el estado del voto electrónico. Este término ha cobrado fuerza en los últimos años, y tiene la intención de mejorar los procesos electorales en papel, aumentando la eficiencia y reduciendo los errores que puedan surgir de este proceso, en el cual participan gran cantidad de personas. Blockchain ha ganado mucha popularidad y pose características que pueden ayudar a desarrollar estos sistemas dando seguridad a las personas involucradas en las elecciones.

Este proyecto se divide en 5 capítulos principales. El primero explica de qué trata el proyecto, el problema que hay que resolver, el alcance y cuáles son los resultados esperados. El segundo es cómo se planificará y desarrollará el proyecto, junto con el presupuesto y los informes de sostenibilidad. En el tercer capítulo, se hace una explicación histórica y teórica de la votación y el voto electrónico, sus implementaciones, ventajas y desventajas. A continuación, en el capítulo 4, se realiza la misma explicación teórica sobre Blockchain. El objetivo de los capítulos 4 y 5 es conseguir una buena base y comprensión de los conceptos principales antes de entrar en la parte práctica o en las conclusiones. Finalmente, en el capítulo 5, se implementa una plataforma de votación electrónica basada en una blockchain de Ethereum.

Al final, se dan las conclusiones sobre el proyecto en general y se adjuntan las referencias correspondientes.

Chapter 1: Generalities

1.1 Context

This is a Bachelor Thesis of the Computer Engineering Degree, specialization in Information Technologies, done in the Facultat d'Informàtica de Barcelona of the Universitat Politècnica de Catalunya directed by Ruben Tous Liesa. In this project, we will evaluate an application of blockchain as a service to implement distributed electronic voting systems.

1.2 Stakeholders

This project is aimed at organizations with electoral purposes, for example, Governments, OSCE (Organization for Security and Co-operation in Europe)… But not only big organizations can benefit from this project, but also small businesses or institutions that require elections to be held, of any kind.

1.3 Problem to be solved

Online voting is becoming a reality today. Countries like Brazil, India, Estonia, and also some states in the US already allow their citizens to vote via the internet. However, some people do not support online voting. There are concerns regarding the higher risk of impersonation, fraud, and also cybersecurity risks compared to traditional voting systems.
To sum up, four essential key points should be dealt with:

- Security: Most online voting systems claim to be secure referring to cybersecurity and resistance to cyberattacks. However, this is not enough. A voting system is secure in the sense that we can trust that the results of an election are fair and correct while protecting voters' privacy (for example by avoiding double voting).
- Voter anonymity: The voter must be able to trace the effect of her vote on the result, while on the other hand, privacy requires that a vote cannot be traced back from the result to a voter.
- Accessibility: This is essentially achieved by an easier voter registration method, giving alternatives to polling places, and reducing the crimination and privacy concerns.

- Voter authentication: To ensure the “one voter, one vote” principle.

Here is where Blockchain can be used to enhance the e-voting systems:

- Reliably verify identities and anonymity of votes
- Securely store data (data records are immutable)
- Transparent transaction history
- Resistance to cyberattacks

1.4 Justification

E-voting systems have searched for new ways to improve the physical elections, trying to improve efficiency, and reduce mistakes that could be made in this complex procedure with so many people involved, the key to developing society. Blockchain is an emerging technology that poses characteristics that could bring the development of secure systems, capable of giving confidence to the population that takes part in elections. It also improves the lack of confidence from the population in their governments modifying illegally the results with a more trustworthy system. With this project, we are making an effort in implementing a blockchain-based e-voting system.

1.5 Scope

Having in mind the problem presented above, this project has as the main objective, the analysis, design, and implementation of an e-voting system under legal and technical standards that provide transparency and robustness in the phase of preparation, vote registration, and vote casting, and auditing.

1.6 Expected results

- Research into the voting systems. What they are, what methods are available, their history, and how time changed these systems. We want to have a good background before taking any action or consideration.
- Research into Blockchain technology. Discover the origins, the pros and cons, what uses it may have, and how compatible would this technology be, when applied to vote.

- Implement an open-source voting system that manages the information of the voting process in a decentralized manner for all the election processes.
- Implement a web application using a smart contract to run an election. To do so, we are going to implement the following: a voting emission module with the use of a blockchain, a vote-counting module in a blockchain, and a repository with all the source code of the software. Also, a client-side will be added to make usability more user-friendly.

1.7 Methodology and rigor

The methodology to implement the above-mentioned objective is the following:

- **Initiation phase:** In this phase, the new project is defined and the approval is searched to pass to the following phases. In this project, I searched for approval from the director of the project.
- **Planification phase:** Phase that determines the objective, scope, limitations, along with the requirements of the project.
- **Execution phase:** The activities defined by the project get executed.
- **Control phase:** A constant revision pattern by the director is defined to ensure the proper development of the project.
- **Closing phase:** The last phase of the project, defines the ending of the project and the acceptance level from the expected final result of the project. For this final phase, all the documentation required will be handed in.

In every phase, the corresponding documentation will be delivered.

Chapter 2: Project planning

This project will last approximately 540hours. More or less in a time length of about half a year. Starting from the end of January until the end of June of 2022. The idea is to work a few hours every day, although other factors such as exams or work may affect this intention.

2.1 Description of tasks

Following, are presented all the tasks that will form this project. The main objective of developing a concrete plan is to complete the thesis on time. For each task, a concrete number of hours will be detailed.

2.2 Project management

Below are the tasks dedicated to GEP (Project management):

- **Project scope:** We have to define the scope of the project in the context of its study. Mention the general objective of the TFG, the context, the reason for selecting the subject area, and how the project will be developed. The total amount of hours (25h).
- **Temporal planning:** Once the scope has been set, planning for the entire execution of the TFG is needed. A description of the phases, and the resources and requirements associated with each one. The total amount of hours (10h).
- **Economic management:** After all, tasks have been identified, it is important to know what the cost of the project will be. Hence, in this section, we focus on making a budget. (15h)
- **Meetings:** Meetings are scheduled every two weeks, to discuss how the project is evolving. Since covid, most likely these will be via Google Meets or a similar method. The approximate time for each meeting is around 1h. The number of hours (20h).
- **Final document writing:** Group all the documents mentioned above, and mix them all to create the final document. And modify those parts that were wrong. The total amount of hours (20h).

The main pillars of the project are discussed here. Many of these tasks match the ones explained in the objective session.

2.3 Research

To ensure the project will be well organized, clear, and understandable a big foundation is needed. Therefore, most of the time in the initial weeks is dedicated to the research of information.

- **Programming Languages:** We work with NodeJS (server-side), JavaScript (web functionalities), ReactJS (user interface), Solidity (implement Smart Contracts). Minimum research on how to work with these languages is required to ensure clean and optimized code.
- **Blockchain-based concepts:** Understand how a Blockchain works internally, what is Ethereum and how we can use this platform. How Smart Contracts work, and other tools like Ganache, MetaMask, and Truffle.
- **Election and legal-based concepts:** Understand how elections are held, and how to develop a system that recreates a physical election ensuring the same level of security and trustiness.

2.4 Programming part

Once the foundation is well-formed and all the concepts are deeply acquired and we are capable to explain and work on them. The programming phase starts. Below are the tasks related to code writing:

- Implement an open-source e-voting system that manages the information of the electoral process in a decentralized manner.
- Implement a vote emission module in a blockchain network.
- Implement a vote-counting module inside a blockchain.
- Create a GitHub repository with all the source code.

2.5 Resources

The resources needed for the proper development of the project are divided into two groups. Human resources and material resources.

2.5.1 Human resources

Three main human resources are distinguished in the development of the project.

- The researcher, is the one developing the project and researching all the information.
- The director of the project, is in charge of making sure the project is going in the right direction and can meet the deadlines.
- The GEP tutor, is in charge of giving feedback to the researcher to improve its project documentation.

2.5.2 Material resources

In terms of hardware resources, for the development of the project MSI desktop computer is used, some of the specifications (Intel(R) Core(TM) i7-9750H CPU 2.60GHz, RAM 32Gb).
The software resources used are the following: Visual Studio Code (for programming), Truffle (framework to develop smart contracts in Ethereum), Ganache (local blockchain), Metamask (plugin works as a bridge between Dapps and the browser), Git (open-source repository).
Finally, other devices are needed to make an election work, so any smartphone, computer, or device with an internet connection will work.

2.6 Risk management

During the development of the project, several issues may appear. These potential risks must be taken into account and evaluate possible alternative plans. Below are listed a few possible risk situations we might encounter, along with a risk level.

- **Meeting impossibilities [Medium risk]:** Since covid has happened, face-to-face meetings are less likely to happen, which makes things difficult, since time has to be spent to find software to arrange those meetings. Connection issues may appear and schedule complications too. Otherwise, the traditional email method will be used.
- **Deadline of the project [High risk]:** Since the planning of the project is all done before even starting, is more than likely that we will

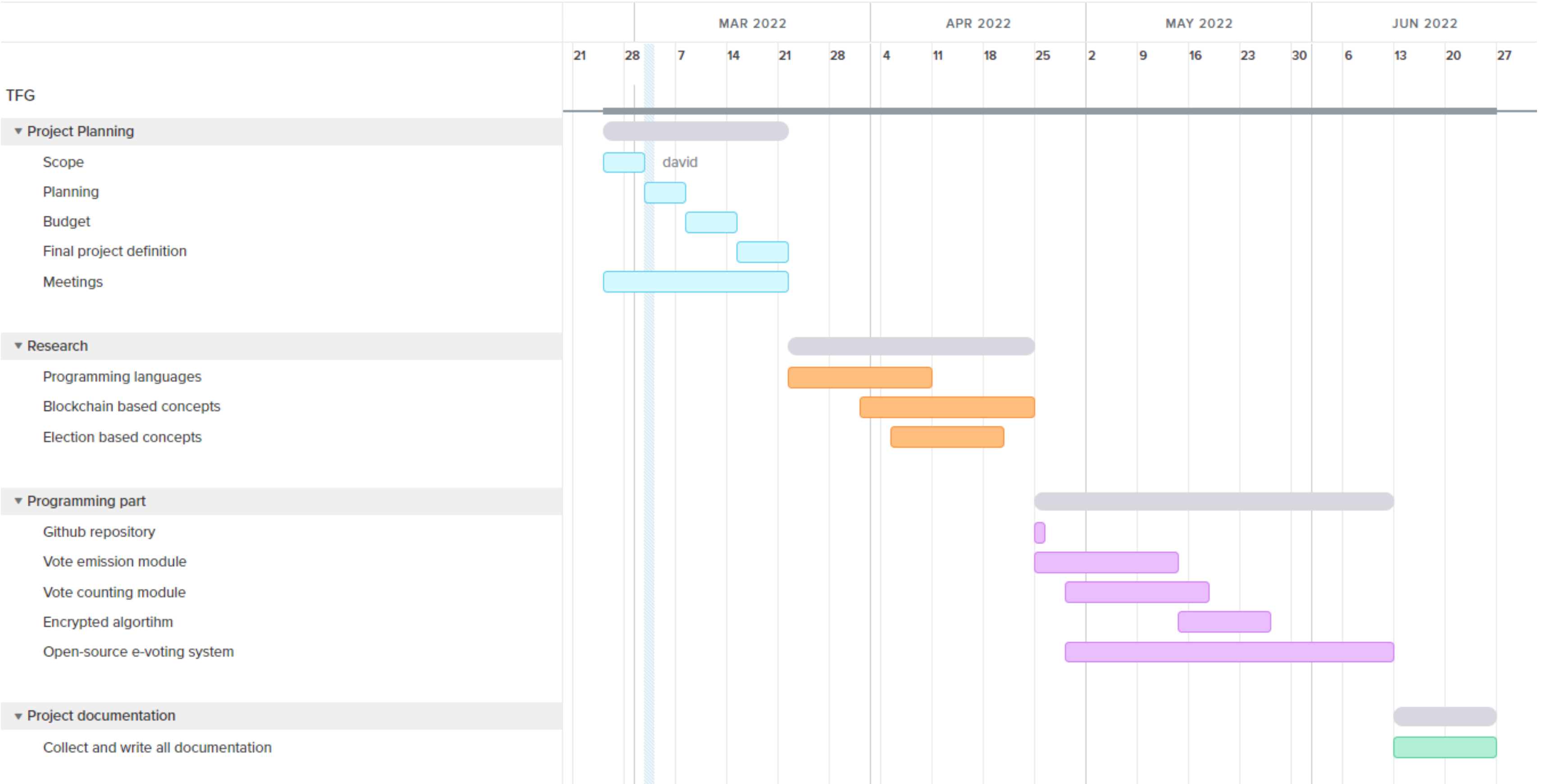
encounter problems and therefore not meet the proposed time plans we created at the beginning of the problem. The solution is to re-create a new time plan adapting to the problems faced.

- **Inexperience with the software and languages [Medium risk]:** Since we will be working with software that we have never used and starting to work on new programming languages. There is a chance we will encounter more issues than expected. If that is the case the simplest solution is to invest more hours into formation and learning.

2.7 Workload estimation

Task	Code	Hours	Dependencies
Project Planning	T1	**85**	
Scope	T1.1	25	
Planning	T1.2	10	T1.1
Budget	T1.3	15	T1.2
Final project definition	T1.4	20	T1.3
Meetings	T1.5	20	
Research	T2	160	T1
Programming Languages	T2.1	50	
Blockchain-based concepts	T2.2	80	
Election based concepts	T2.3	30	
Programming Part	T3	215	T2
Vote emission module	T3.1	40	
Vote counting module	T3.2	40	
GitHub repository	T3.3	5	
Encrypted algorithm	T3.4	30	
Open source e-voting system	T3.5	100	T3.1-T3.2-T3.4
Project Documentation	T4	80	
Collect and write all documentation	T4.1	80	
TOTAL		**540**	

2.8 Gantt diag



2.9 Budget and sustainability

In this section, we will discuss the economic cost of the project. We will start with the personnel cost per task, generic costs, and other costs. We will also explain the mechanism for controlling potential budget deviations, including numerical indicators. In the end, questions regarding sustainability in the project will be answered.

2.9.1 Staff costs

To define what are the costs, first, we will define all the different roles that are needed throughout the project, and then find the cost per hour. To calculate the total sum, the worker's salary is multiplied by the total amount of hours needed to finish the task.
All these tasks are all going to be performed by myself and the tutors, but we will anyways divide the work into different roles, for a better understanding of the global cost.

In this project, 5 different roles are identified:

- **Technical writer:** Document everything that involves the development and results of the project.
- **Researcher:** Look for information, experiment, analyze and draw conclusions from the information found.
- **Programmer:** Code all the systems, from backend to frontend.
- **Tester:** Verify that the currently written functions by the programmer are working properly and that all interfaces
- **Project manager:** Is in charge of making sure the project is developing in the right direction.

To compute the total cost for each task, we have to distribute the amount of work each role plays in every task. But first, we show how much each role earns every year and divide it by the approximate working hours to get a price per hour.

Role	*Annual Salary (€)*	*Including SS (€)*	*Price per hour (€)*
Technical writer	32.000€	43.200€	19.57€/h
Researcher	33.482€	45.200€	20.47€/h
Programmer	37.198€	50.217€	22.75€/h
Tester	20.347€	27.468€	12.44€/h
Project manager	48.312€	65.221€	29.54€/h

Table 1: Annual salary of the different roles. [1]

2.9.2 Generic costs

Amortization

For items such as hardware or software, amortization is required. In this project almost all the software used is free. But the most expensive part of the material costs will be the hardware.

More or less, it is calculated that each item is being used for a total of 150days at an average of 3.5 hours every day.

The only material cost we have to count is the desktop computer we are working on. All the software is free so it hasn't to be counted. A total of 544 hours result from computer usage.

Hardware	*Price (€)*	*Time used (h)*	*Amortization (€)*
MSI computer	1.600	544	414.17
Total			***414.17***

Table 2: Amortization of the hardware

To compute the amortization the following formula is used:

$$Amortization = Resource\ price \times \frac{1}{4years\ \times 150days\ \times 3.5hours} \times hours\ used$$

Equation 1: To calculate the amortization of a resource

Task	Hours	Technical writer	Researcher	Programmer	Tester	Project Manager	Cost (€)
Project Planning	**85**	**20**	**20**	**20**	**20**	**85**	**4.162,8**
Scope	25	0	0	0	0	25	738,5
Planning	10	0	0	0	0	10	295,4
Budget	15	0	0	0	0	15	443,1
Final project definition	20	0	0	0	0	20	590,8
Meetings	20	20	20	20	20	20	2.095,0
Research	**160**	**0**	**160**	**0**	**0**	**0**	**3.275,2**
Programming Languages	50	0	50	0	0	0	1.023,5
Blockchain-based concepts	80	0	80	0	0	0	1.637,6
Election based concepts	30	0	30	0	0	0	614,1
Programming Part	**215**	**0**	**0**	**115**	**100**	**0**	**3.860,25**
Vote emission module	40	0	0	40	0	0	910
Vote counting module	40	0	0	40	0	0	910
GitHub repository	5	0	0	5	0	0	113,75
Encrypted algorithm	30	0	0	30	0	0	682,5
Open source e-voting system	100	0	0	0	100	0	1.244
Project Documentation	**80**	**80**	**0**	**0**	**0**	**0**	**1.565,6**
Collect and write all documentation	80	80	0	0	0	0	1.565,6
TOTAL	**540**	100	160	135	120	85	**12.863,85**

Table 3: Estimated task cost

2.9.3 Deviations in the budget

2.9.3.1 Contingency

Unexpected events may appear during the project. We will have in mind those possible obstacles to creating a contingency fund. A 15% addition to the budget has to be added to face these events. The result is: 12.863,85 + 414.17 = 13.278,02 x 0,15 = 1.991,70. The computed contingency cost is 1.991,70 €.

2.9.3.2 Incidental costs

As mentioned in the risk management section, some possible inconveniences have been taken into account. Such as, meeting impossibilities, deadline of the project, or inexperience with the software and programming languages. To calculate the cost for each incident, we multiply the price of the event by the probability that it happens:

Incident	*Estimated Cost (€)*	*Risk (%)*	*Cost (€)*
Meeting Impossibilities	0	30	0
Deadline for the project	547,05	80	437,66
Inexperience with software	583,8	50	291,9
TOTAL			**729,56**

Table 4: Incidental costs

2.9.4 Total cost

Finally, a sum-up of all the costs calculated will result in the total cost of the entire project:

Activity	*Cost (€)*
Staff cost	12.863,85
Generic cost	414,17
Contingency	1.991,70
Incidental	729,56
TOTAL	**15.999,28**

Table 5: Total cost of the project

2.9.5 Management control

Although the incidental part of the budget has already been defined. It is yet to be explained how the potential budget deviations are going to be modeled. We are going to plan how to detect any alteration in the project's cost.

In each task, we will calculate its deviation from the estimated cost. Following the formulas and calculations used in each different task.

- **Estimated cost:** To calculate it, we will use the estimated cost of each task declared in (Table 3).
- **Real cost:** To calculate the real cost of each task. A new recalculation of all values (staff cost, generic costs, contingency, and incidental) will have to be re-adjusted. With this procedure, we can identify which task has been miss-estimated and therefore, adjust it accordingly.
- **Deviation:** This value will be the difference between the estimated cost and the real cost of each task.

In the case of the deviation, two possible outcomes exist. Either a positive deviation, which means that the task has been over-estimated and therefore there is money left, which we will use to cover other possible incidences. Or a negative deviation, which means that more money needs to be added to the task, which in this case we will take from the contingency fund to cover the deviation.

2.10 Sustainability report

After delivering the survey about sustainability, I have figured out that my knowledge about sustainability is quite low. I thought I would know most of the things since in my head everything seems very obvious. But I have never thought that there would be so many things involved.
I do believe that by studying my project and all the following aspects, not only will I have a more conscious point of view about the sustainability of my project, but also, I will improve my knowledge about all these topics.

2.10.1 Environmental dimension

Have you estimated the environmental impact of undertaking the project?
Yes, I have. I consider that this project has an important impact on the environment in a positive way. With the digitalization of the elections, no more physical resources will be wasted, such as all the papers, the fuel needed to make all the population go to the electoral college, electricity, and human resources. On the other side, there are also the hardware costs. All the population must have a device that can connect to the internet to vote. But I think that this downside is compensated since nowadays the vast majority of the population owns one device with Wi-Fi-connection.

Have you considered how to minimize the impact, for example by reusing resources?
Most of the resources used for the project can be reused. The highest cost is the first time the election is run since all the servers and hardware has to be bought, but once it is done, in every election the same resources will be used.

How is the problem that you wish to address resolved currently (state of the art)?
Nowadays very few countries give the option of electronic elections. And all the ones that offer that possibility also have the physical election college available.

In what ways will your solution environmentally improve existing solutions?
Since all elections are held physically, the adaption to an e-model would solve all the issues related to making all the population go to the electoral college, most likely the rate of voters would increase since going to vote will take just a few seconds, no fuel will be needed, no waste of paper will be made, and probably the electricity wasted will remain more or less the same. In the electronic model all the servers will be consuming electricity, but so does having all the election colleges with lights on all day.

2.10.2 Economic dimension

Have you estimated the cost of undertaking the project (human and material resources)?
Yes, I have, all the economic aspects of the project have been taken into account during the budget and sustainability chapter of the project.

In what ways will your solution economically improve existing solutions?
As previously discussed, none of the infrastructures needed to be held every election will be needed anymore. All the economic impact will be on the first election, after the first one, the cost will be reduced significantly. And now all this money can be invested into improving the interface, security, and functions of the software.

2.10.3 Social dimension

What do you think undertaking the project has contributed to you personally?
I think undertaking this project will help me apply all the concepts learned during the degree, and also enhance my knowledge in the election/political area. I will have the opportunity on developing a project with the support of professionals in the area that I am working in, which will give me plenty of new ideas for the future.

In what ways will your solution socially improve (quality of life) existing

Is there a real need for the project?

There is. I think that nowadays, in the industrialized world, where everything is already digitalized, elections won't take much until they all jump into the electronic era and start improving people's life. Everyone will be able to privately and securely vote from their homes, without the need of leaving work and having to move to the closest election college.

Chapter 3: Electronic voting and references

3.1 Cconcepts

The reader must be familiarized with the following concepts to understand the work correctly.

3.1.1 Blockchain

A blockchain is a distributed database that is shared among the nodes of a computer network. As a database, a blockchain stores information electronically in digital format. Blockchains are best known for their crucial role in cryptocurrency systems, such as Bitcoin, for maintaining a secure and decentralized record of transactions. The innovation with a blockchain guarantees the fidelity and security of a record of data and generates trust without the need for a trusted third party. [2] More in-depth information will be given in chapter 5, but this will do for now.

3.1.2 Internet

The large system of connected computers around the world allows people to share information and communicate with each other [3]

3.1.3 Distributed systems

A distributed system, also known as distributed computing, is a system with multiple components located on different machines that communicate and coordinate actions to appear as a single coherent system to the end-user. [4]

3.1.4 Electronic voting

Electronic voting (also known as e-voting) is voting that uses electronic means to either aid or take care of casting and counting ballots. Depending on the particular implementation, e-voting may use standalone electronic voting machines (also called EVM) or computers connected to the Internet. [5]

3.1.5 Smart Contracts

Smart contracts are trackable and irreversible applications that execute in a decentralized environment (blockchain). Once the smart contract has been deployed nobody can edit the code or change its execution behavior. Ethereum provides an open-source blockchain platform that can be used to deploy smart contracts. Ethereum introduced a new programming language called Solidity (similar to JavaScript) to write these contracts. We implement our e-voting system, as a system based on a smart contract in a blockchain. [6]

3.1.6 Zero-Knowledge proof

This is a concept that is not directly related to blockchain but is seen as an essential component for satisfying some of the requirements that we will mention below to build an e-voting system on a blockchain. A zero-knowledge proof is a cryptographical method by which one party, the prover, can prove to another party, the verifier that the prover knows a value x, without revealing any information other than the fact that
the verifier knows the value x. [7]

3.1.7 Voting

Voting is a tool of democratic systems, which is based on consulting the population about their opinion. They are some main characteristics to take into account:

- **Unique vote**: This vote can be used only once during the duration of the election.
- **Private vote**: This vote is private.
- **Vote integrity**: Once the vote is processed, no one can modify it.

3.2 Introduction

Before diving into what Blockchain technology is about, and implementing e-voting into this technology, it is pertinent to understand what voting systems are together with the results and the impact they have had. All to subsequently mention in which of these systems the Blockchain technology is compatible. From there, to expose the strengths and weaknesses that can be compared to the other voting alternatives.

3.3 Elections

Elections in a generalized context are a participatory mechanism for decision-making, in which the members of a given group or also called electors use the vote to elect one or more representatives. With the aim that the elected representatives will occupy positions of relevance in the group for a certain period. An example of positions that an elected representative can occupy is: president or mayor in a political context, leader of the same group or president of a company in a business context.

The vote mentioned above is how an individual makes a decision, based on his or her preferences, for a choice between two or more alternatives. In addition to this, voting is also used as a decision-making mechanism in political motions or proposals, such as the plebiscite, where laws or administrative acts are put to a popular vote.

Although the types and names given to the types of votes vary according to the environment and context in which the election is being held, the following will briefly describe the types of votes used by voters. An affirmative vote is intended to demonstrate support for an option, a blank vote shows for any of the options, a spoiled vote is one that was poorly cast and for this reason is not valid, the subtractive vote subtracts the support for an option, the useful vote is that which is made by supporting the option that has the best chance of winning, and the punishment vote is the vote for a different option to show one's disagreement with the management given by this option.

The types of votes can also be named according to how they are cast. The following is a brief description of these types of votes, the in-person vote is the one that the voter casts in person at the polling station. Remote voting is that which the voter casts by means other than at the polling station. Internet voting is voting by internet. Postal voting is voting by post, and e-voting is voting using electronic means to cast a vote or to count the votes.

3.4 Voting Systems

Suffrage is the constitutional right to vote for elected public office and has been seen in the vast majority of democratic systems as the preferred method of election to public office. Historically, the exclusion of certain groups from suffrage was quite common. However, with the advancement of societies and the advent of the concept of equal rights, it is now more common to see "universal suffrage" or "equal suffrage", where every adult has the right to vote without exclusion. And this, added to population growth, technological advances, and political culture of each nation, among other factors, has generated over time different needs in each country in terms of how voting should be carried out. Leading to the creation of different voting systems to adapt to these needs. [8]

3.4.1 What are voting systems?

In a general way, the term "voting systems" can encompass many concepts that define and govern the conduct of voting in a nation or organization, be it how votes are counted, how the winning candidate or decision is decided and so on. However, in this document the term "voting system" shall mean the systems through which the votes are recorded, cast, or counted.

Voting systems can be either analog or digital. Before discussing electronic voting, it is important to understand a little about the origin and functioning of traditional or analog voting systems. This may present various problems that could be solved through ICTs, bearing in mind the digital reality in which we live, and yet analog voting systems are still the most widely used in the majority of democratic countries.

3.4.2 Analog system

Analog voting as we know it today dates back to the 19th century when it was popularized in what were then the early democracies and the electoral processes they carried out. One of the first voting methods that could be seen at that time was the secret ballot. This consisted of meeting at an anonymous point with the voter to cast his or her vote using a ballot paper. However, with the growing idea of universal suffrage, which implied a greater number of voters, it was then necessary to generate other strategies for the collection of votes. Among them and the one that is used today is to generate meeting points with printed or at that time handwritten lists containing the name of the voters eligible to vote at these points.

In terms of the way votes are recorded and counted, different methods have been devised and implemented according to the technologies that have been developed throughout history. In ancient Athens around the 5th century B.C., voting was done by depositing-colored pebbles in boxes. While the use of ballots in voting dates back to 139 B.C. in Rome.

In terms of mechanical voting, various systems have been used. The first was in the United Kingdom in the 19th century, where a mechanism was used that counted bronze balls that voters dropped into the ballot boxes of their candidates. In the late 19th century in the United States, a voting machine was patented that allowed election with a series of buttons. Also in the United States, a lever mechanism was used for most of the 20th century, whereby the voter had to operate the lever corresponding to his or her choice. The system prevented the casting of more votes per person. In the same country, a punch-card voting mechanism was also used, where a machine punched the cards according to the voter's choice and then counted by a pneumatic mechanism. Other variants of these mechanisms have also been patented and/or used. However, the manual paper ballot, despite being the most archaic, is the most widely used today.

Analog voting by ballot, despite being the most widely used system, presents several problems, some of the most notorious of which are:

- Delay in vote counting
- Travel to the polling place
- Long wait time to vote
- Possibility of vote fraud

The latter is the most worrying, as it is a process that involves completely different people and the results may vary according to the intentions of some of the individuals or collectives involved in the different stages of the electoral process be it the counting of votes or the disregard of irregularities by oversight or regulatory bodies. But despite a few exceptions, analog voting is to date, the most trusted process and the one that gets the approval of the majority of the population. Therefore, making it one of the more secure processes

Many of the above-mentioned problems could be solved through electronic voting. However, many factors interfere with the modernization of voting systems, such as the high budget needed to plan and implement an electronic voting system, this major spending though, is only for the first election that is going to be held. After this, all the infrastructures carry over time to the next elections, making the overall cost in time very efficient. The more elections that are run, the cheaper it becomes over time.

Moreover, the lack of interest on the part of the entities in charge of the process, illiteracy or difficulty of access to computer resources for some voters, and even the distrust they may have of digital media.

For the most part, voters do not trust the technology as they compare the security of electronic voting processes to other activities related to the use of the internet and that they might lose their anonymity in voting, thus many voters prefer to vote in person and using pencil and paper ballot papers.

3.5 Electronic system

Systems in which the registration, casting, or counting of votes in elections involves the use of information and communications technology, or by its abbreviation ICT, may be called electronic voting systems. E-voting is a tool that, with proper planning and careful design, makes electoral processes efficient, since it makes voting secure and speeds up the processing of results, and facilitates the voting process.

E-voting systems, when implemented in ICTs, have a wide variety of ways to be implemented. However, the two types of e-voting systems that cover the totality and variety of existing e-voting systems are the on-site e-voting systems and remote voting systems.

Electronic voting systems at the polling station (on-site), as the name suggests, are electronic voting systems in which the voter must be present at the polling station. Usually enters the polling station with a physical credential that verifies the person's identity. In this type of system, the voting information and the voter are not sent through the internet or other networks. The vote is usually cast through direct electronic voting machines, the usual interface in this type of machine is a touch screen or a scanner that scans the ballot paper for the voter's mark. The information that is scanned is stored in the machine itself.

Remote voting systems are more complex than on-site e-voting systems, given all the planning and logistics behind them, as well as being more unpopular among the people given the insecurity of not knowing where and who can see or even manipulate their vote. Remote voting systems use the internet as a means of sending voting information. In these systems, the voter can vote from anywhere with a computer or device, such as a smartphone or tablet that has access to the internet. They can vote with an application for this purpose, voters will be able to vote freely as long as they have the necessary credentials such as a username and password. Although voting systems usually use the internet to send information, means such as postal mail, fax, or landline telephone can be used as a method of sending information. [9]

3.5.1 Methods

This section describes the main means by which the different types of electronic voting are implemented. It is made clear that these means can range from simple technological components that are designed for the casting, receiving, and counting of votes, to complex networks of distributed technological components, where each of the components is intended to perform a single function, to enhance features such as the security of e-voting.

A Direct-Recording Electronic voting machine (DRE voting machine) or also known as a virtual ballot box, are on-site voting machines that record the voter's vote using a representation of a digital ballot paper. These machines are physically designed to have buttons and a screen or a touch screen, these machines process the votes through embedded software and record them on removable memory components such as a USB, CD, or an SD card. The most current advancement in DRE machines is the public network DRE voting systems, which have as an addition the method of recording votes through telematic networks to be stored on a central server. One of the most important points about these machines is that, they may or may not have an integrated voter-printed component, which is known as the Voter-Verified Paper Audit Trail or VVPAT. VVPAT component is intended to provide physical proof of the vote cast.

OMR Optical Mark Recognition are on-site voting machines that use an optical scanner to identify the marks previously made by the voter on a ballot paper, the voting information obtained from the ballot paper can be counted by the machine itself or it can be recorded and sent by telematic means. Examples of the use of this machine are the United States and Argentina, both of which use this technology in their elections.

Mixed systems, also known as electronic ballot printers, are systems that use DRE machines for the voter to select and print the ballot paper with the chosen option already marked on it. The paper ballot is then entered by the voter into an OMR machine to be automatically scanned and registered. An example of this type of system can be found in Belgium, where it is mostly used to conduct elections.

Internet voting systems or online voting systems are systems that allow votes to be cast from any device such as public computers, ballot boxes, or personal devices that are enabled for this purpose. One of the most important characteristics of this type of system is that voting can be done from any place with internet access. Votes in this type of system are cast from one of the aforementioned devices, after which the vote is recorded and counted on a central server. Examples of this type of system are applied by Switzerland, Norway, Estonia, the United States, the United Kingdom and Ireland for their elections.

3.5.2 Impact of implementation

Among the challenges for the implementation of e-voting, two factors are always constant. The cost required for the implementation of these systems and the mistrust that it generates for voters and the entities in charge of carrying out the voting. Together with the fact that the most widely used e-voting systems, turn voting into an automated process, without the need for human interference. Countries and organizations that had the idea of using e-voting, have gone through the planning phase, or that have directly implemented it, are evidence of the impact of e-voting. Which can be positive or negative, compared to traditional or manual voting systems. In the following, some of these implementation cases will be described, focusing mainly on the cases of e-voting implementation by countries, as these require high standards in terms of user capacity, transparency, and security of the vote. The most important factors are accuracy and veracity in the calculation and counting of voting results.

In Brazil, electronic voting was implemented in 1996 to make voting more transparent about electoral fraud. This country has a one hundred and forty-eight million voters. With the implementation of electronic voting in Brazil, there were advantages such as; ease of access to voting for voters in rural parts of the country, ease of voter registration, the counting of votes and the dissemination of results is done in a short period, however, the most important advantage that Brazil gained by implementing the electronic voting is the elimination of fraud in vote counting and the aggregation of results.

In India, electronic voting was implemented in 1998 and by 2004 it was established as the only means of voting. Thanks to the rapid publication of the results of the elections. India has a population of 1,400 million by 2022, making it the country with the largest number of voters in the world. It is also a country that has a high level of illiteracy, rurality, and multiplicity of languages, which implies several challenges for the implementation of e-voting. Machines used in India are portable, domestically designed and assembled, and their power source is a battery. Challenges such as geography and the multiplicity of languages in India were overcome quickly.

The United States is a special case among the countries that apply electronic voting systems, it implements four different voting systems to conduct its elections. First of them is the punch card scheme which is an analog voting system, which has already been described. The most important feature, and the reason why this voting system is still used, is because for the elderly who are used to this system, it is easier to continue
and on the other hand, there are some people from certain groups who are distrustful of the security and veracity of the results of the other three systems that are implemented. The second system that is used for voting is direct electronic registration, which is used by one-third of voters and is also used in the two versions described above. The third method implemented by the United States is a mixed voting system. Finally, the last system is the least used, as it is a relatively new implementation in this country and it is the internet voting system, which is used mostly by young voters and voters outside the country. [10]

3.5.3 Advantages and disadvantages

Individual e-voting systems have advantages, strengths, disadvantages, and weaknesses. However, this section will focus on briefly and concisely explaining the characteristics of electronic voting over the type of voting that is done in an analogous and on-site manner, which is known as the traditional voting method.

The commonly known advantages of electronic voting systems can be summarized in four, the certainty and reliability that these systems present, by eliminating human error at the stage of determining the voting results. The time and cost savings in the phase of determining the results of the vote and by avoiding the logistics, coordination and printing of different resources. In the book “Secure Electronic Voting” by the Author (Dimitris A. Gritzalis) [11], the following advantages are mentioned:

- Facilitate the voting process by providing reliable and rapid data on the receipt of votes and results.
- Allows people to vote from anywhere in the world
- Voters can check the election in which they are voting at any time.
- Official results are obtained and published a few hours after the closing of the voting process.
- Financial resources are saved, as there is no need to print election ballots and certificates, less "polling stations" are deployed.
- Given the facilities of these systems, which immediately and inexpensively collect the decision of voters, e.g., governments, organizations or communities could for example, hold votes for different purposes that have a participatory democracy model, at any time and in any place.
- The use of an electronic ballot box will not only lighten the workload of officials overseeing the voting process, but can also reduce human error, simplify tasks, increase the speed of obtaining and disseminating results and, additionally, generate significant savings in the documentation and materials used for voting.
- There is a high rate of voter increase, as voters can vote from anywhere, like home, work, or even school.
- The loss of voter time caused by waiting in long queues. If there are large numbers of voters on the day of the election, it completely reduces the voter's time spent waiting in long queues.
- The ecological factor by reducing the consumption of raw materials in stationery and cardboard boxes.

The disadvantages faced by this system and presented in the same book are the following:

- It can lead to unemployment, as many people working in the voting process are dismissed or no longer employed.

- The costs of implementation are high for both hardware and software, including maintenance, licenses, support, and training. This infrastructure can be deployed in different voting environments, so the large expense is only at the beginning.
- The privacy and secrecy of the election may not be guaranteed. It can be manipulated if the necessary IT security structure and adequate training of human resources are not in place.

Apart from the pros and cons taken from Dimitris' book, we can add the following strengths and weaknesses enumerated in the book "Electronic Voting Machines the True Story" by Alok Shukla [12], which are taken into account different aspects. The strengths are:

- Efficiency in handling complicated electoral systems that require laborious counting procedures.
- Increased accessibility, for example, through the use of "audio-voting" for visually impaired voters, and online voting for voters living abroad.
- Multilingual interfaces for countries or companies where more than one language is spoken, a more practical solution than paper ballots.
- Decrease in the number of invalid ballots as the voting system can warn the voter when a vote will be invalidated.

The weaknesses of electronic voting are:

- Increased infrastructure and environmental requirements. Associated with power supply, communications technology, temperature, and humidity.
- System certification is required, but there are no widely accepted parameters for such certification.
- High dependence of the system on the supplier and/or the technology.
- Increased security requirements to protect the voting system during and between elections, including during transportation, storage, and maintenance.

- Increased costs for the purchase and maintenance of the e-voting system.
- Risk of manipulation by internal staff with privileged access to the system, or by "hackers" outside the system

3.6 Related work

To make the present project, several projects have been checked to see how other people presented the same situation and what solution they faced. Here are some interesting projects with plenty of knowledge.

In [13], Owonubi. J.S developed a similar e-voting system using Django, which is a Python Framework. In this project, there is an administrator interface added differently to our case, where the administrator part is done on the smart contract. We do this to avoid possible changes in the rules of the election. Thus, making it impossible to make changes unless a new contract is deployed. Also, no theoretical part is added to his project.

The group, Kashif. M, Junaid. A and Muhammad. M, in [14] from the NED University of Engineering and Technology in Pakistan, and the University of West London, UK. They presented a project where they developed a system with a frontend developed with Java. And stored all the addresses manually in a MySQL database. This application also supports importing data using MS Excel to perform bulk import views. In their case, they use Multichain as the blockchain platform to create a private blockchain. In the end, their project achieves a very similar result, with some different functionalities.

In this research paper [15] Uzma. J, Zarina. S and Mohd. J, described the different challenges that this process requires. They dive deep into the theory about the implementation of these systems. A very interesting point they make comes from the following table:

Online Voting Platforms	Framework	Language	Cryptographic Algorithm	Consensus Protocol	Main Features (Online Blockchain Voting System)							
					Audit	Anonymity	Verifiability by Voter	Integrity	Accessibility	Scalability	Accuracy/ Correctness	Affordability
Follow My Vote	Bitcoin	C++/Python	ECC	PoW	✓	✓	✓	✓	✓	✗	✓	✓
Voatz	Hyperledger Fabric	Go/JavaScript	AES/GCM	PBFT	✓	✓	✓	✓	✓	✗	✓	✓
Polyas	Private/local Blockchains	NP	ECC	PET	✓	✓	✓	✓	✓	✗	✓	NA
Luxoft	Hyperledger Fabric	Go/JavaScript	ECC/ElGamal	PBFT	✓	✓	✓	✓	✓	✗	✓	✓
Polys	Ethereum	Solidity	Shamir's Secret Sharing	PoW	✓	✓	✓	✓	✓	✗	✓	✓
Agora	Bitcoin	Python	ElGamal	BFT-r	✓	✓	✓	✓	✓	✗	✓	✓

Figure 1: Comparison of current blockchain-based voting systems

They give a lot of attention to the fact that these systems do not allow scalability. This is because there is a barrier from an architectural standpoint. Transactions in the blockchain are very slow. Which is the major concern for enterprises that want to adopt this method.

Chapter 4: Blockchain Technology

4.1 Introduction

The main intention of this chapter is to introduce the reader to all the details of Blockchain technology. Starting from the origins, going through the functionalities of it, what is it made of, its main characteristics, the security and what applications it has. This has the intention of making the reader capable of understanding what will happen on the development phase of the project. And also understanding the capabilities and advantages it has in order to implement an electronic voting system in terms of security against fraud and functionalities.

The blockchain is mostly based on cryptographic concepts, theory of games and computation science. By using cryptography, it is possible to validate what users are doing the transactions and avoid fraudulent movements among the accounts. The theory of games provides a more in-depth look on how the system is designed, a block is not considered malicious or not, it works by consensus. Depending on its application, the degree of complexity of the Blockchain can change, but the principle of maintaining a record always remains. Last, computation science is the one that lets implement the logic and mathematical components of the decentralized applications.

4.2 Origin

¿How did the Blockchain appear?
In 1991, researchers named Stuart Haber and Scott Stornetta published a report where they provided a solution to the modification of timestamps in digital files. This solution used a chain of blocks with cryptographical security that stored the documents with their time stamps.

One year later, in 1992 Stuart Haber, Scott Stornetta and Dave Bayer improved the design by adding Merkle Trees, this provided the possibility to add multiple documents to a single block which enhanced efficiency. Sadly, this technology never made it anywhere and it is not considered the first blockchain since the patent expired in 2004.

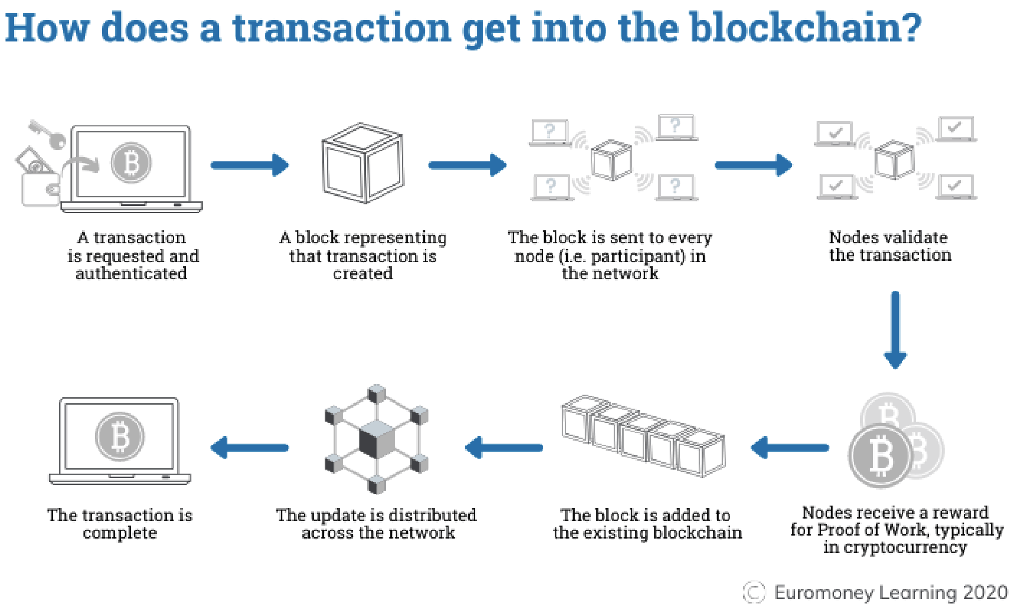


Figure 2: Proof Of Work diagram

It was in 2004 when a cryptographer and programmer Harold Thomas Finney developed a system that he named Proof of Work (PoW). This system worked in a way that gave a token to a user after doing a proof of work. This means that the user solves a complex mathematical operation, with the computation power of his computer using the resource of its computer and electricity to get tokens or access to activities in the system. This token was based on the algorithm HashCash (a proof of work system created in 1997 by Adam Back in order to avoid spam in the mail). This token was not interchangeable between users. In exchange, when this happened, the system created and exact token signed by RSA5 which was interchangeable between people.

Given its operation, the PoW system is considered to be the initial prototype of cryptocurrencies since it came up with a solution to the issue known as the double spending, which happens if the tokens are not correctly controlled by the system. The issue with the double spending is a constant duplication of the same token to be used repeatedly.

¿How did PoW solve this double spending issue?
The system registered each token with its corresponding owner to an independent and secure server. Due to this security, availability and integrity,

this server was designed so that despite the location of the users, it was possible to prove the exactitude and integrity in real time of all the tokens.

Nevertheless, this concept did not have much relevance until 2008, thanks to Satoshi Nakamoto. He is known to be the brain behind Blockchain technology. There is not much known about him, some people say it could be just one person, and others say it is a group of people working in Bitcoin, the first application of this technology. Nakamoto published a whitepaper named "The Cryptography Mailing List" as a decentralized electronic currency called Bitcoin. This network was based in a HashCash algorithm to place timestamps in the blocks and without the need of them to be signed by a third party. Then, he added a difficulty value so that the creation of blocks is limited to ensure a constant growth of the network.

On the other side, the double spending issue is solved by a decentralized protocol Peer-to-Peer which traces and verifies every transaction. This design was implemented as the core of the cryptocurrency Bitcoin by Satoshi Nakamoto in 2009.

Nowadays there is a huge offering in different platforms created based on Blockchain. These platforms present applications different to Bitcoin, with different purposes, most of them without the intention to become a currency but a project. In 2013, Vitalik Buterin, programmer and founder of the Bitcoin magazine saw the necessity of creating a scripting language to develop decentralized applications in the Bitcoin network. Nevertheless, he never reached an agreement with Bitcoin, to which we developed a distributed computation network based on the Blockchain technology that he called Ethereum. Among other functionalities, it allowed the creation of smarts contracts, which will become the foundation of our voting system project. [16]

4.3 Distributed ledger technology (DLT)

Before talking about how a Blockchain works it is important to mention what distributed ledger technology (DLT) is. It allows users to store all the information from a program, simultaneously and permanently shared by a group of users or servers. This information is always kept public.

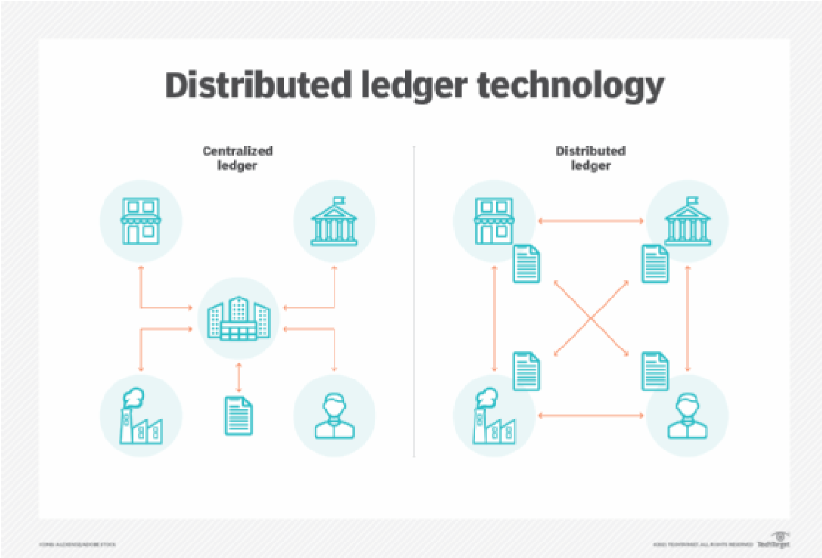


Figure 3: Operating of a distributed ledger technology [16]

From another point of view, DLT is like a decentralized database that is managed by a multitude of users and there is not a unique authority entity. The fact that there is a distributed registry allows a great level of transparency, making it very difficult to modify the information. To add data into a DLT system, cryptographical keys are used, this data is distributed among different nodes, it gets replicated and synchronized using consensus algorithms.

DLT is classified into three main groups. The first one, permission-less, are completely public and any user from the internet can access them and participate, for example, Bitcoin. Secondly, permissioned, these can either be public or private, the public ones can be accessed by anyone, but the addition of data is limited to authorized entities, in private examples, only allowed entities can access and add data.

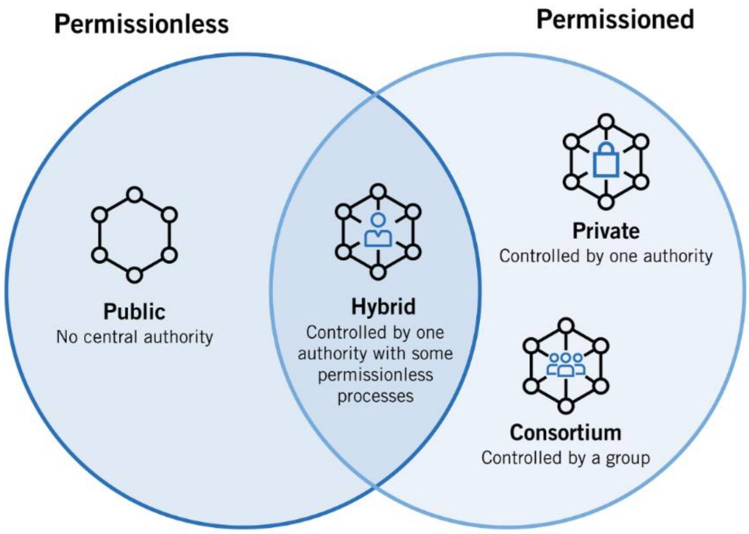


Figure 4: Permissioned vs Permissionless Ledger [17]

In conclusion, a Blockchain is in essence a DLT with certain particularities.

4.4 How it works

In a Blockchain, data is distributed among information blocks that are chained sequentially to other blocks called hashes. This way, a chain of inalterable blocks is created. In every block, the following information is stored: a registry with the valid transactions, information about the block and the bonding with the hash of the previous block of the chain.

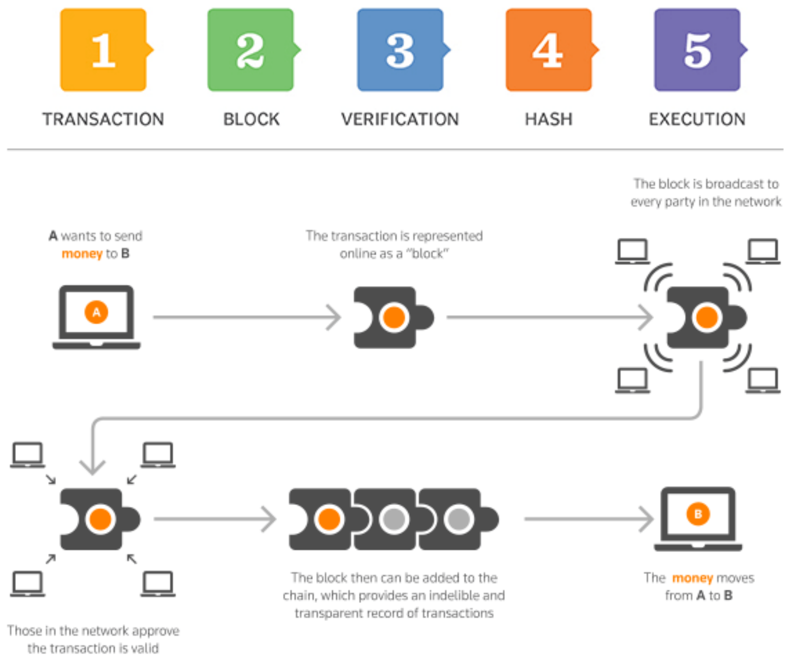


Figure 5: Blockchain operating

Each block has a unique and unchangeable place in the chain, it is this way because if a block was to be moved or deleted, then all the connection in the chain would be lost since every block has the information about its previous block, and then the following block would not be able to chain to the previous one. Additionally, each node that participates in the network is in charge of storing an exact copy of the blockchain.

4.5 Cryptography

Cryptography can be defined as a group of techniques or a method with the finality of altering the information to make it unreadable to unauthorized people. The main objective of cryptography used to be confidentiality, however, with the invention with the internet and the fast advance in technology, it was possible to find new uses for cryptography since new necessities for creating more advanced techniques to ensure integrity and security of the information.

In this chapter some key elements about the cryptography that Blockchains use will be implemented:

4.5.1 Asymmetric cryptography

Blockchain is fundamentally based on asymmetric cryptography also known as public keys. It is based on two keys, one public and one private. In asymmetric cryptography either the sender com the receiver must use the same algorithm.

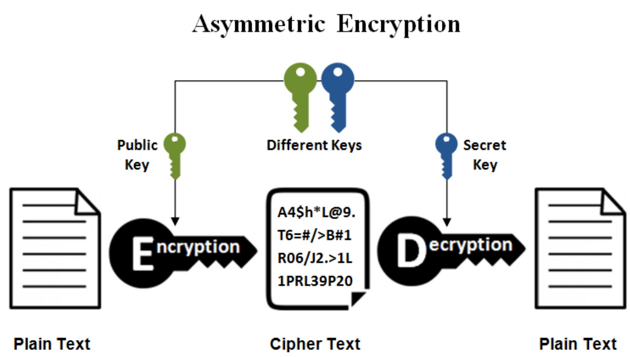


Figure 6: Asymmetric Encryption

In the first place, an encryption algorithm must be chosen, these algorithms are based on mathematical systems, the more complex the math's are, the more secure the algorithm will be. Having an algorithm, public and private keys must be generated. First, the private key is created, this can be formed from random values. Only the owner must have access to this key, to ensure the privacy of the information. After that, a public key is created. This one is created from the private key and is the one shared with all the users of the network so that messages can be received.

The sender will use the public key of the sender to encrypt the message, and, on the other side, only the receiver will be able to decipher the message by using its private key, associated with the public. Moreover, if a message is encrypted using the private key, this can be deciphered using the public key. In addition, no message can be decipher using only the key which has been used to encrypt the message, in any case, both keys are required to decipher it.

4.5.2 Address

An address is an alphanumerical chain that can be shared to receive transactions, these addresses are produced from public keys making use of hash functions. For example, a Bitcoin address is an identifier of between 27 and 34 characters, starting from the number 1 or 3, which represent the destination of payment in bitcoins. This address can be simply created by using any program client to Bitcoin.

4.5.3 Digital signature

Digital signatures are fundamental in Blockchain, they work as proof that the transaction has not been modified in the sending process. This signature is linked to the transaction that has been executed and its unique for each transaction. They are calculated using cryptographic techniques from the private keys from the sender and the information included in the transaction.

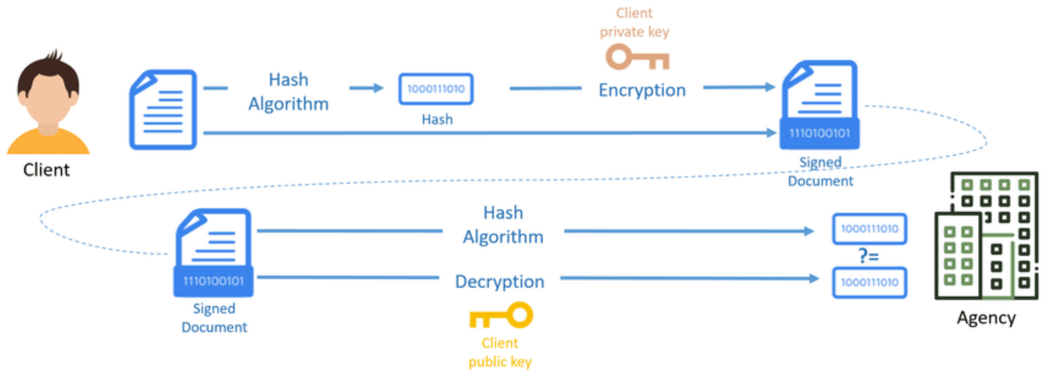


Figure 7: Digital signature

Since this signature has been generated from a private key, proving the integrity and security of the message is as easy as deciphering it using the public key of the sender (which is sent with the transaction), if the information coincides, then it is a legit transaction.

4.5.4 Hash functions

Hash functions are mathematical functions that make up the data structures of a blockchain and are a fundamental part of its cryptography. These functions get a variable number of characters as parameters and converters it to and alphanumerical chain of a fixed size (hash) as an outcome.

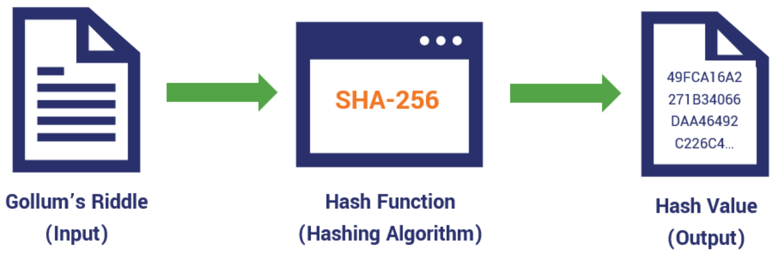


Figure 8: How hashing works

Hash functions must follow a series of principles to achieve its main design purpose.

- Entries can be of any length, but their outcome must be fixed to a certain size.
- The same parameters given to the same hash function must always return the same result.

- There cannot be two different entries that generate the same result.
- The smallest change to the message must change the hash completely so that no one can correlate the new hash to the previous one before making the change.

Hash functions are used in different ways inside a Blockchain, they are commonly used to prove the integrity and authenticity of the information, they can be used to index information in hash tables and make searching tasks faster, and they are also used to securely authenticate users without the need of storing passwords locally, among other functions that can be assigned depending on the blockchain.

4.6 Composition

In this section the essential components that make up a Blockchain are mentioned, which make the fundamental principles possible.

4.6.1 Peer-to-Peer network

In a Blockchain, communication is always made with peers, there is not a central node in which information is transmitted. Each participant is in charge of storing information about the movements that have been made in the Blockchain, and, after that, this participant has to share it with others.

4.6.2 Transactions

A transaction is an aggrupation of data that stores all the token transactions between users. Transactions must carry a digital signature from the sender, and, after that, these transactions are transmitted among the network where will be validated and subsequently ordered together with the rest of transactions to form the blocks in a Blockchain.

4.6.3 Tokens

A token is commonly the unit of value emitted by a private entity, in its form, a token is more than a coin or a currency, since uses can be assigned to it. For example, in a private network, a token can work to retrieve some privileges, as an incentive, among other uses. Finally, the person that designs the system, who decides how will the value of the token be composed. From there come all the different uses that can be assigned to it.

4.6.4 Consensus algorithm

Are algorithms used by the Blockchain network to achieve a consensus and define which transactions are valid and which are not. What the algorithm reveals will become an absolute truth to all the nodes that form the network. Consensus algorithms follow a protocol, which are the main rules that are followed by a Blockchain.

For example, we have Bitcoin, its consensus algorithm is the Proof Of Work. This is the first consensus algorithm and the most popular to date. This algorithm is used to confirm transactions and create new blocks. Miners compete to complete transactions and receive rewards.

4.6.5 Blocks

Blocks are made out of a collection of confirmed transactions, in addition to the information regarding the identification and concatenation of the mentioned blocks. When a block reaches its limit in terms of capacity (number of transactions), it then becomes the moment of validating it. This is where miners come to place. After to its validation, the block connects to the chain and a new block is created which will store the newer transactions.

A basic block contains and index, a hash that identifies it, a hash pointer that connects with the last block, and timestamps. In its content are all the transactions stored in the block together with the list of transactions.

4.6.6 Nodes

Nodes are computers connected to the Blockchain network using a software that is in charge of storing and distributing updated copies of the chain of blocks in real time to all the participants on the network. When a block becomes validated and therefore added to the chain, this change is then made public to all the other nodes, these will then store a copy of the block. For example, nowadays each block of the Bitcoin network weighs 4MB, which means that the total weight of the network is around 320Gb.

Making the average transaction size of around 343bytes. On average the network processes around 250.000 transactions each day. [18]

4.6.7 Mining

Miners are a fundamental aspect of the public side of the Blockchain technology, miners are the nodes that fulfill the function of generating new blocks, apart from validating transactions and registering them inside them. Rewards are given to the first miner that is capable of solving the first valid block of transaction generating its hash code. The block must be valid by the proof of its hash by the rest of members of the network so that this can be aggregated to the chain of blocks.

4.7 Security

Throughout this chapter, we have mentioned several characteristics that are part of the basic operation of a Blockchain. From these characteristics, it is possible to mention a series of qualities that Blockchain technology can offer about security issues.

4.7.1 Immutability

Thanks to the cryptographic algorithms that make it possible to guarantee and verify the integrity of the data recorded on the Blockchain and its nature as distributed records, it is possible to say that the data is immutable. Atomic transactions take place on the blockchain, and once the transactions have been distributed and recorded, they are unalterable in the system, and the integrity of this information can be verified later on by using the hash corresponding to the block in which it is stored.

4.7.2 Counterfeit resistance

One might think that the fact that a Blockchain is public makes it vulnerable to multiple types of attacks, including counterfeiting. But it is quite the opposite, as it is a decentralized network that makes counterfeiting impossible. Decentralized networks make forgery impossible. When a person makes a transaction, that person must sign it with his or her own private key. Then a hash algorithm is applied to the transaction, before it is sent to the network. Is through this information that all the other nodes can verify the identity of the person. Other nodes can verify the identity of the user and the integrity of the transaction. If any member of the network receives transactions that do not comply with the above verification, which have been modified in some way, then that transaction will be discarded.

4.7.3 Pseudo-anonymity and transparency

In a public Blockchain all participants can see all the information contained in the blockchain (including its hash values) without violating the privacy of users. Anyone using the Blockchain can be anonymous or can share their identification with others. All that can be observed in a Blockchain are records of transactions between users belonging to the network. On the other hand, this can vary when it comes to private or hybrid Blockchains. In these cases, an identification is usually required to be able to grant permissions on the network.

4.7.4 Democratic

At least in the public Blockchain, which uses a peer-to-peer system, there should be no one entity in the system that is more powerful than the others. All participants in the network should have equal rights in any situation, and decisions are made when the majority reaches a consensus. Thus, when one person somehow manages to alter the information in a block, that alteration will not be taken into account as it would only be one vote among many.

4.7.5 Registration consistency

The qualities mentioned before, guarantee to a certain extent consistency in the general recording of information in the Blockchain. But what would happen in cases where, unsynchronized nodes that are not connected consider some transactions to be fraudulent?

This is where the consensus mechanisms come into play. Many consensus mechanisms exist that have into account different factors to make voting the most legitimate as possible.

4.7.6 Auditable

Since all the blocks on the Blockchain are connected in a chain to the previous block until they reach the initial block, the task of auditing is made much easier, and if you want to verify a transaction, your search should be relatively quick thanks to the timestamps. Furthermore, there is no need to be concerned about the integrity of the information as the data is unalterable.

4.8 Types of blockchain

Blockchain initially emerged as a single type of blockchain, which is known as public blockchain, but due to the large fields of action in which this technology can be applied, depending on necessity different types have emerged, public blockchains, private blockchains, and a combination of these two, the Hybrid Blockchain, which is the best known. However, a last type of Blockchain has been gaining momentum, this type of Blockchain is implemented in the cloud and is called BasS meaning Blockchain as a Service. Now let's take a look at the other types of Blockchain.

4.8.1 Public

The public Blockchain is decentralized because all nodes in the network are equal, it is distributed because each node in the network maintains an up-to-date copy of the database. The database is in turn maintained by the users of the network. A public Blockchain is consensual because it handles decentralized consensus algorithms such as proof-of-work or proof-of-stake. Public Blockchains tend to be slow.

Anyone can access a public Blockchain, make transactions and participate in the network without permission and anonymously. All that has to be done to access a public Blockchain is to download the application and connect to the network of nodes. This type of Blockchain usually has an associated cryptocurrency and sometimes rewards block miners. Examples of Public Blockchain are: Bitcoin, Ethereum, Monero Dash,
Litecoin

4.8.2 Private

Private Blockchains are not open to everyone in the world, the only way to access a private Blockchain is by invitation. The control of the whole system is held by a single entity, which is responsible for the maintenance

of the blockchain, this entity grants permission to users so that they can participate, propose transactions and accept blocks on the network.
Because the Blockchain is managed by a central entity, the Database is only stored on central servers and cannot be accessed by anyone as an invitation is required. This Blockchain is mostly used by the financial sector and is usually fast. Examples of Private Blockchains are Monax, Hyperledger Fabric, and Ripple.

4.8.3 Hybrid

To access and participate in a Hybrid Blockchain you need to be invited, i.e., this type of Blockchain is not open to everyone in the world like the private ones. Unlike private, all the transactions made in this type of Blockchain are public. Transactions carried out in this type of Blockchain are public, the management of the database is usually carried out by several entities. Hybrid Blockchains do not have an associated cryptocurrency, so they do not reward mining either.

This type of Blockchain is usually used by governments, associations and companies where large amounts of transactions take place. Finally, hybrid Blockchains occur when several companies are grouped to benefit each other. Examples of this type of Blockchain are B3, EWF and R3.

4.9 Different applications

The Blockchain technology was designed in 2008 and implemented in 2009 by Satoshi Nakamoto solely as the core for the Bitcoin cryptocurrency, but over time, programmers and enthusiasts have seen Blockchain's potential to be developed in different types of environments and industries. Blockchain as of the date of this paper not only serves as the core of the cryptocurrency, but thanks to Bitcoin the digital money or cryptocurrencies are constantly growing.

Why are cryptocurrencies used? For the user who uses them, the main advantages of cryptocurrencies are the anonymity of the user and the speed of transactions, due to the elimination of unnecessary intermediaries. An important aspect to highlight is that, the fact of eliminating intermediaries, will not make cryptocurrencies less trustworthy, since thanks to the

Blockchain design, each transaction has its record, which allows for authentication it.

Let’s now see Blockchain in different contexts and analyze how its design can be implemented and how it helps improve the infrastructure:

4.9.1 Financial sector

One of the mentioned environments is the financial sector, where blockchain technology has made its way into the FinTech industry. The name of this industry is given by the words of the Financial Technology, this industry aims to bring together digital technologies and financial services. FinTech aims to provide efficient, agile, convenient and reliable financial services. The customers of the FinTech industry are individuals, businesses and governments. Examples of the industry's applications are mobile payment systems, person-to-person lending and crowdfunding schemes.

The most commonly used application of Blockchain technology in the financial sector is the KYC verification or "know your client" which is a process used by companies to identify and verify the identity of each of their customers. Another use is the simplification of the process of settling purchase and sale of any item, this is expressed in a document called a remittance. Other applications include reliable payment solutions, records of storage and management, fast processing speed.

With each of the above-mentioned applications, great advances have been made to solve complex and redundant difficulties in the financial sector. These difficulties arise due to the use of technology to offer services. We will talk about how Blockchain technology helps in these applications.

Given the public-key/private-key design of the Blockchain, many financial institutions tend to think that this technology is dangerous because of the anonymity it can offer. However, if the system makes intelligent management of the public and private keys, this feature does not represent any danger. This is why Blockchain technology is used for the authentication, verification and storage of electronic records by the financial companies, which generates KYC utility for stock exchanges.

Blockchain's design facilitates efficient, transparent and secure payment processing services. As it uses encrypted distributed ledgers to ensure real-time verification of transactions, without the need for intermediaries such as banks. Blockchain applications make transactions fast, secure and low-cost.

Blockchain is driving the emergence of digital insurance by enabling the tracking and management of transactions, making audit trails permanent, helping insurers and their customers to automate claim processing by implementing smart contracts and preventing insurance fraud.

If each bank were to use the same general ledger to record every transaction that takes place, as opposed to the current situation, where each bank has its own ledger, then transactions would be way faster. Blockchain offers a faster processing speed as it provides a distributed ledger that allows all parties to communicate with each other in real time.

4.9.2 Registration application and data verification

Traditional recording and verification data applications use databases that are usually administered and managed by intermediaries. These generate records that can be easily altered, in Blockchain, as mentioned above, intermediaries are eliminated. The records that Blockchain generates are immutable and unalterable, which makes the data in a Blockchain more secure than in traditional databases.

All sectors do registration and verification of data, however, we will describe some specific applications, such as the health sector, where a single medical record can be created for each patient's data and medical history.
In the real estate sector, where fraud and manipulation by the owner of a property or land can be avoided if a single register is created, in which for each property or land, the owner and all property transactions are recorded. In the automotive sector to register vehicles. Other applications such as protecting intellectual property and digital products such as music, photos and books, by encrypting and storing these as a transaction on a Blockchain designed to serve that purpose.

4.9.3 Supply chains

In the production and food sectors, so-called supply chains are very difficult to monitor due to the immense amount of information they can handle. Using Blockchain, the complexity of tracking the entire supply chain process is reduced, because everything is stored in records. The immutability of these records ensures the validity of the most important information in the supply chain, such as the origin of the different products that make up the final product.

4.9.4 Cloud distributed storage

In a Blockchain-based storage network, data or files would be stored in a decentralized peer-to-peer system. Meaning that because this network allows nodes to be created in different geographical locations, the network would be able to withstand the failure of any node, as another node would replace it in its place and would generate a higher availability of services such as Google Drive or Dropbox.

4.9.5 Smart contracts

Smart contracts are digital contracts that are executed after the fulfillment of one or more conditions, this type of contract is facilitated by the Blockchain due to the immutability feature of the records. The contracts stored on the Blockchain have the terms needed, and once these are met, the contract self-executes. Among the many applications that can be given to these contracts, the one that stands out the most is the leasing of goods such as real estate or vehicles.

4.9.6 Digital identities

In Blockchain, each user has a private cryptographic key associated with their account and this in turn is associated with a public key.
This public and private key system mean that the identities of users cannot be manipulated. So, no user can impersonate another user unless he/she has the credentials. But these cannot be extracted from the Blockchain.

4.9.7 Automated security

This application is the combination of the digital identities and smart contracts of the Blockchain together with electronic locks. The security application would be completely automatic, as based on whether or not a person has permissions, it would let the individual access or not a given location.

4.9.8 Voting systems

Blockchain solves some of the most important problems of e-voting,
with the use of the private key and the public key, voter anonymity and voter identity theft is covered. Also, this system is known to be one of the most secure as with the decentralization of the Blockchain, a malicious visitor will not be able to get anything from the users, unless they give their key and password to someone else. And finally, one of the most important problems with voting systems is the uncertainty as to whether or not there was electoral fraud. This is solved thanks to the immutability feature of the Blockchain.

Chapter 5: Implementation

5.1 Introduction

In this chapter, we will describe the development of a decentralized voting application. The application starts with a simple design that later gets improved with more functionalities. It is a proof-of-concept that has the following features:

- Smart contract that holds an election between two or more candidates.
- Votes coming from verified voters are computed by the smart contract and stored in the blockchain.
- Votes are displayed in real time for everyone.
- Code deployed to the local Ethereum blockchain.
- Client-side frontend.
- Tests to prove codes' robustness.

The aim is to prove that blockchain is a technology highly compatible with an electronic voting system. But it is also to prove if blockchain is, or not, the way to go when we want to ensure security, trustiness, and usability.

5.2 Tools used

Here, all the tools used to successfully develop the project will be described in detail

- **Node JS**
  Node JS is an environment of JavaScript oriented to certain events. In our case, NodeJS is used on the server side.
- **Java Script**
  JavaScript is a programing language that has the intention of programming web functionalities. These programs are called scripts and can be written directly in the HTML of a webpage and execute themselves automatically. The main benefit is the compatibility with HTML and CSS.
- **React JS**

React is a JavaScript library for the creation of user interfaces. In our case, it is used to develop the front-end. It is a tool that facilitates the creation of different HTML components efficiently.

- **Ethereum**

  Ethereum is an open-source platform based on blockchain technology, which allows smart contracts to be executed over it. To execute transactions inside the blockchain, “Gas” is spent. This “Gas” unit has a monetary value in ether units, which is the cryptocurrency of Ethereum. Ethereum uses gas due to its high volatility and price change, which means that no matter what the price of ether is, the gas price will always be the same for every transaction.

- **Solidity**

  Solidity is a programing language used to develop smart contracts. Moreover, we will use this language because is the most used and therefore has a lot of forums and information online.

- **Truffle**

  Truffle is an open-source development framework. It allows the creation of decentralized applications on the Ethereum Blockchain. It provides a set of tools that let you develop smart contracts in the Solidity programming language.

- **Ganache**

  Ganache is a personal blockchain used in the development of Dapps (Decentralized applications) in Ethereum. Ganache allows you to recreate blockchain environments locally, which lets you test your programs without the need of spending real ether. In our project, Ganache is used to test all the functionalities before the deployment in an actual blockchain.

- **Metamask**

  Metamask is a plugin that works as a bridge between Dapps and your search engine. In our project, we choose Metamask because it is compatible with the framework Truffle. This plugin will connect to our local Ethereum blockchain with our account, and interact with it.

- **Visual Studio Code**

  Visual Studio Code is a source code editor developed by Microsoft. It has an extension that allows the development of smart contracts.
- **GitHub**

  GitHub is an open-source website used as a huge repository where all users can post their code to maintain the different versions of their code. GitHub is used in the project to ensure the proper order of development and the good organization of the data.

5.3 Centralized and decentralized applications

Beginning with the basics, we will analyze how a centralized web app works without any blockchain or decentralization taking place. This will allow us to understand the basics and the foundation of web development to further introduce the uses of the blockchain and its implementation.

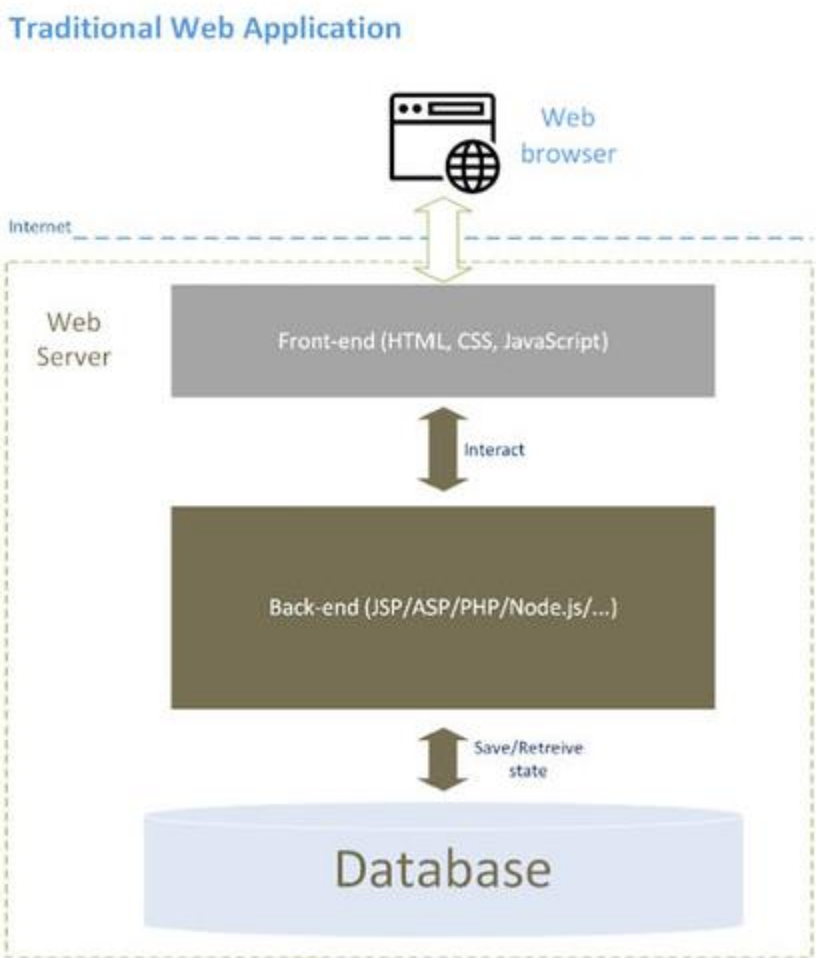


Figure 9: Client-Server architecture at a high level [20]

The most basic representation of a web app and how the interactions work is represented in figure 1.
Whenever you want to access a web application, you use a web browser to connect to a web server over the internet. This server contains all the client-side files like HTML, CSS, and JavaScript. The server contains all the backend codes, business logic, and any data that is stored in the application in the database. A request is received and the server talks to the database/cache to treat the requested resources.

When you want to create a web app, you write a client-side application. This application does some business logic on a web backend, that makes some reading/writing/modifications and puts it into a database. Which happens on a central server.

This is a very common architecture, that works perfectly fine most of the time. But imagine an application that would require this database to be public and also securely accessible by everyone without the requirement of the web app owner having all the data. An example presented in [19] is eBay. If you are a power seller who has hundreds of good reviews and for some reason your account gets closed, or you move to another platform. You lose all these reviews which could hardly impact your business. What would be very nice, is the ability to take all the reviews and ratings and move them to another platform. In these situations, is where decentralized applications take place. And the easiest tool to use to develop these Dapps is Ethereum.

Analyzing what an Ethereum Dapp looks like:

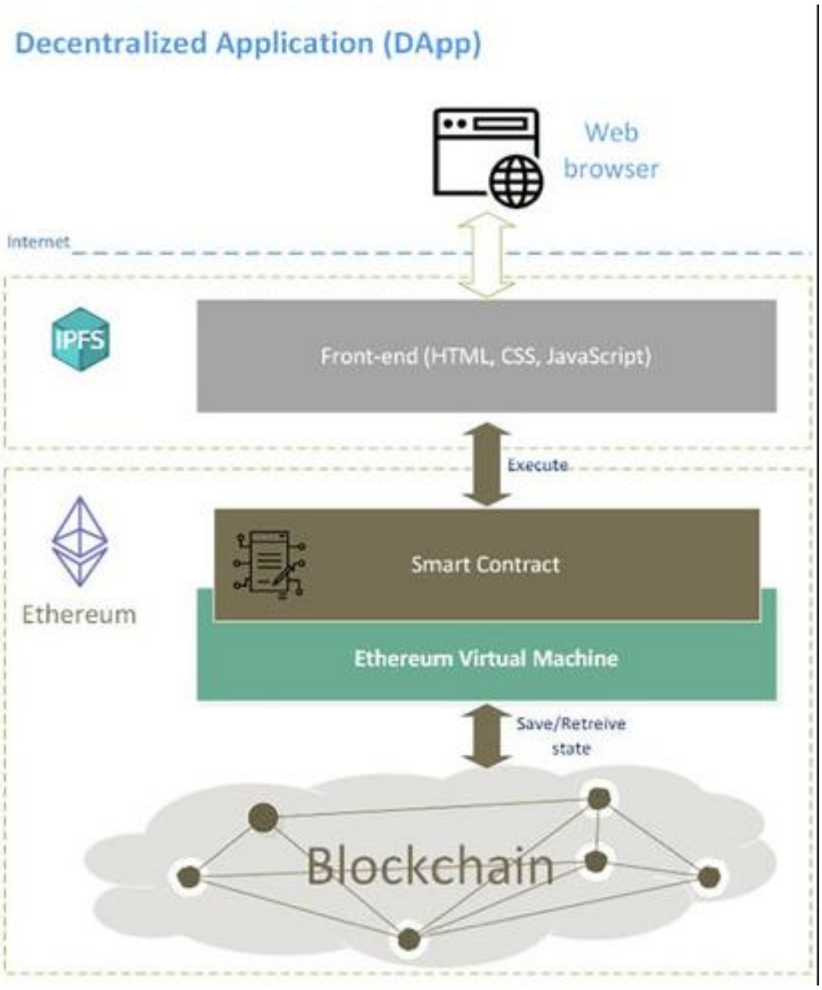


Figure 10: Ethereum Dapp architecture [20]

Before explaining how the diagram in figure 2 works. The most important difference between a traditional web app and a Dapp is that, in the first one, each client connects to the same central server, in the Dapp, each client communicates with its instance of the application.
This means that before you are allowed to interact with the application you must download all the blockchain. This sounds impossible but we will see how it is done.

On a Blockchain, instead of connecting directly to a server, we access the web app via browser, and connect to the client-side application, the difference is that this web application is not going to talk to a web-backend and a database, instead, it's going to talk directly to the blockchain.

On the blockchain, we are going to have code that is written with Ethereum smart contracts. This smart contract contains all the business logic for our application, and all the items will be stored in the blockchain itself.

5.4 Centralized Voting Webapp problems

We will analyze what could happen if we build a centralized web app voting system. First, as seen in previous sections, since all of the data is stored in a database in a centralized web app, this means that all the data could change at any time, causing changes in our votes or deletion of them. Secondly, all the code of the election is stored in the server, which could also change at any time, meaning a modification of the rules of the election. In conclusion, we do not want to build a centralized web app. The solution is to build a decentralized application so we can ensure that the rules (code) of the election remain the same during the entire process. That only one vote is counted and this can never be changed nor deleted. And finally, the candidate with the greatest number of votes is the one winning the elections.

5.5 Blockchain

We already know how a blockchain works, but we will dive into the two main parts of the Blockchain that take place for this project:

- **Database**: In the example shown in the previous chapter, our client-side application is talking to the blockchain which is a separate network. A Blockchain is a peer-to-peer (P2P) network of nodes that all talk to one another in a distributed network. This means that different computers/machines talk to one another. We can connect to an individual node on the blockchain to use it. That is what the web application in the previous example is doing. All the nodes on the network participate in running the network. They all contain a copy of the code and the data on the blockchain. All of the data on the blockchain is contained in bundles of records called blocks. These blocks are chained all together to make up the blockchain. All the nodes on the blockchain participate in ensuring that the data on the blockchain is secure and unchangeable. And that is what makes the blockchain so powerful.

- **Code:** The application codes, called contracts, are programs written in Solidity. A Solidity compiler is used to convert these contracts to Ethereum Byte Codes to be then deployed to the blockchain. There are some alternatives to this method, but Solidity is the most popular language.

5.6 Simple voting contract

We will start by describing the most basic form of the Voting application. We will initialize a set of contestants, let everyone vote, and finally display the total votes received. For now, the idea is to learn the process of compiling, deploying, and interacting with the Dapp.

Using Solidity, we will write the first version of this Voting Dapp smart contract. As mentioned, it has two methods, one to retrieve the total votes and one to add votes.

```
pragma solidity ^0.8.13;

contract SimpleContract {

  // candidate name stored as bytes32 and the total votes as uint256
  mapping (bytes32 => uint256) public votesReceived;
  bytes32[] public candidates;

  // Constructor (called only once)
  constructor(bytes32[] memory nameList) public {
    candidates = nameList;
  }

  // Total votes for candidate
  function sumVotes(bytes32 candidate) view public returns (uint256) {
    require(isAllowed(candidate));
    return votesReceived[candidate];
  }

  // Cast a vote
  function addVote(bytes32 candidate) public {
    require(isAllowed(candidate));
    votesReceived[candidate] += 1;
  }

  // Checks if allowed to vote
  function isAllowed(bytes32 candidate) view public returns (bool) {
    for(uint i = 0; i < candidates.length; i++)
      if (candidates[i] == candidate)
        return true;
```

Figure 11: Simple voting smart contract

For this first model, everyone is allowed to vote. To make sure no one votes for anyone that is not a candidate, a function isAllowed(candidate) is created. With this function, we make sure that every transaction is done to the correct candidate or display an error otherwise. To do this, we check the array of candidates and see if it exists inside the array.
To keep track of the votes of each candidate, we created a map with the name as the key and the votes as the value. So, when we cast a vote, we call the function addVote(candidate). This function first checks if this candidate is a real one, and then adds 1 to the value of the map. Finally, to retrieve the total amount of votes a candidate has, it is just a matter of checking the value of the map of the certain candidate. Also, in this function sumVotes(candidate), the check of a legit candidate is made.

5.7 Compile and deploy

So far, the project only has a backend, and in this state, to see the progress, we need to compile the solidity code using an npm module called solc. We won't go deep into this, but we will explain what two files are created after compiling the Solidity code to understand what will happen on the Frontend.
The compiler outputs these 2 files after compiling:

- **SimpleContract.bin:** This file contains the bytecode generated after compiling the SimpleContract.sol. This is the bytecode that will be deployed to the blockchain.
- **SimpleContract.abi:** This file tells the contract user what methods are available. This is a definition file that represents an interface or template of the contract.

We will explain the steps needed to run the project without an actual frontend developed yet. These steps are not very important since we will implement everything in the front end later. But they help us understand what is the required process to successfully deploy and use a smart contract.

Once the project is compiled, we need to deploy the application to interact with it. To do so, we get into the node console and query all the accounts in the blockchain that we get from the ganache.

To finally deploy the contract, we have to create a deployedContract object, this object is used to deploy and initiate contracts in the blockchain. Load the bytecode and the abi file. And then use the web3 deploy function.

The following arguments need to be defined when we deploy and send functions:

- **data:** The .bin file that we discussed above that contains the bytecode
- **arguments:** The arguments required by the constructor. In our case, the only required argument is the nameList that represents the names of all the candidates.
- **from:** The blockchain requires knowing the account that deployed the contract. In our local scene, we created 10 accounts using ganache, and we can use the one we want. But in the real blockchain we have to prove that we own the account using the passphrase.
- **gas:** This will be the amount of gas (ether) to insert the code into the blockchain. This is the money it costs to work with the blockchain, which goes to miners.
- **gasPrice:** Price associated with the unit of gas.

5.8 Interacting with the contract

Now the contract is deployed to the network and it is possible to interact with it. It is only required to call the instance deployedContract that we created before.

Since we are going to create an entire frontend to be able to interact with the contract. We will not go deep into the interaction through the console but will see an example and explain it.

Now that the contract is all deployed, we will use the NodeJS console to interact with it. There are only two available functions to call right now. The sumVote would be like so:

```
root@mint:/home/mint/Desktop/voting_dapp# deployedContract.methods.sumVotes(web3
.utils.asciiToHex('Pep')).call(console.log)
```

This is a straightforward call. The first object deployedContract is the main instance of our contract. The methods.sumVotes is used to select the function we want to use from all available methods. The web3.utils.asciiToHex is used to transform our plain text (name of the candidate) to a hexadecimal value. In the Solidity code, we can see that the only argument required by the method sumVotes is a candidate in byte32, in our case we are passing ‘Pep’. Finally, we add the .call function to execute the call and print the result. This function returns two values: the sum of the total votes and the transaction id. In a blockchain, every transaction has its id. This is that all transactions in the blockchain are immutable and you can always referee back to it using the id that is returned once a transaction has been validated.

For the addVote we use a command like this:

```
root@mint:/home/mint/Desktop/voting_dapp# deployedContract.methods.addVote(web3.ut
ils.asciiToHex('Pep')).send({from:'Metamask_Address'}).then((f) => console.log(f))
```

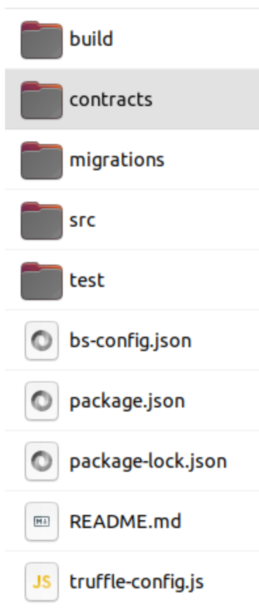
Everything remains as the last one but this time we have the .send function where we have to specify which account is doing the transaction. This has to be filled with our account address. This function does not have any return value, but we still print the console.log to check that the transaction has been executed properly and to get the transaction id.

5.9 Advanced Application

The main idea of this section is to go over all the projects, section by section, and explain what has been done and why this implementation has been chosen. Mainly covering the structure of the directory, the smart contract and the frontend. Finally, at the end of the document, there is an annex showing how the application works and how to interact with it.

5.9.1 Directory

Having made a simple application and created a few functions. We will develop more functionalities together with a frontend. To begin the project, we will use a truffle feature to create the base of the project. This is called pet-shop and will create all the main folders and



files needed to develop a decentralized application, so you don't have to do it manually. It can be understood as a base template to start developing new decentralized applications. As seen in the figure above is the structure that gets created.

5.9.2 Smart Contract

Starting with the variables created for this contract, we have added a struct. In the simple smart contract that we have written before, we have to go over the array of candidates each time to check if it exists. Therefore, no attributes can be assigned to it, and we have to use maps for each attribute. A more efficient friendly approach would be to create a struct of candidates. This way, each candidate has its own set of attributes and can be accessed directly. Its attributes are its name, id, and vote count.

```
struct Candidate {
    uint id;
    string name;
    uint voteCount;
}
```

Figure 12: Struct of the candidate

The next thing is to store all the Candidates in one place so that we can reference them in the future. By using a map, we can assign every candidate to a unique id, therefore making the search much more efficient. In solidity, by setting a variable public, an automatic getter is created, and therefore it is not required to implement it. Since in solidity there is no proper way of knowing how many objects are inside a map, we will use a counter to monitor the number of candidates that we have. This is because if a function returns a key of an object from the map, which is not yet created, it will return the object with the default value. In our case, an empty candidate. To solve this, a simple uint variable is used to implement this counter.

```
mapping(uint => Candidate) public candidates;
uint public candidatesCount;
```

Figure 13: map of candidates and candidates' counter.

Now that we have the mapping of the candidates, we need a function dedicated to filling up this map. To do so, we create a function that gets as an argument the name of the candidate in string type. This function has to increment the candidatesCount uint (to monitor all the candidates) and has to add the new candidate to the map. In Solidity this is done the same way as you would on C++. Create a new Candidate object with the attributes and assign it to the map. It is important to mention that since we do not want anyone to add candidates, this function is set to private and therefore no one except the contract can add candidates.

```
function addCandidate (string _name) private {
    candidatesCount ++;
    candidates[candidatesCount] = Candidate(candidatesCount, _name, 0);
}
```

Figure 14: Adding candidate

To add candidates to the election, it would be as easy as to call the addCandidate function in the constructor. Passing as an argument the name of the candidate. To make an example, I will add two different candidates, so we can test them later once all the functionalities are set. The names of the examples are made up and therefore they do not represent any real person.

```
constructor () public {
    addCandidate("Guillem Folch");
    addCandidate("Raquel Mestres");
}
```

Figure 15: Adding candidates in the constructor.

5.9.3 Testing

Although not all functionalities are finished in the contract. We have only added a candidate, it is important to test the code to ensure that no bugs are found. As Edsger Dijkstra said:

> *"Testing shows the presence, not the absence of bugs." -Edsger Dijkstra-*

In a blockchain, it is of extensive preference to check all the possible bugs in a code. Once the contract is deployed to the blockchain, this code becomes immutable, thus making changes impossible. If bugs are found when the code is already deployed, a new deployment is required, which means that the state of the copy will change together with its address. Also, deploying contracts to the network has gas costs associated with it. The main idea is to spend an as little amount of Ether as possible. Finally, if the function from the contract happens to write to the blockchain and contains bugs, the user calling this function would waste Ether and consequently, the function would not behave as expected. Overall, we are sure that testing is a must, and even more on smart contracts.

In the main directory, we already have a test folder created where we will store all our test files. To write these tests, we will use the Mocha [20] testing framework. All the tests will be written in JavaScript. Following there is the code of the testing for the actual contract.

```
1 var Election = artifacts.require("./Election.sol");
2
3 contract("Election", function(accounts) {
4   var electionInstance;
5
6   it("it initializes the application with two candidates", function() {
7     return Election.deployed().then(function(instance) {
8       return instance.candidatesCount();
9     }).then(function(count) {
10      assert.equal(count, 2);
11    });
12  });
```

Figure 16: Testing of the initialization of the application

We will explain the code from the figure above. The first line of the code represents the association of the contract with a variable. This way we can reference it. From there, we write the contract function in line 2 which will contain all the tests. The "accounts" that appear in line 3 as an argument of the function is a callback that acts as the accounts from our local Ganache. In line 10 we check that the count value, returning from instance.candidatesCount equals two.

```
  it("candidates initialized with the corresponding attributes", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      return electionInstance.candidates(1);
    }).then(function(candidate) {
      assert.equal(candidate[0], 1, "correct id");
      assert.equal(candidate[1], "Guillem Folch", "correct name");
      assert.equal(candidate[2], 0, "correct votes count");
      return electionInstance.candidates(2);
    }).then(function(candidate) {
      assert.equal(candidate[0], 2, "correct id");
      assert.equal(candidate[1], "Raquel Mestres", "correct name");
      assert.equal(candidate[2], 0, "correct votes count");
    });
  });
```

Figure 17: Testing of candidates' attributes

In this piece of code, we check that the candidates are initialized with the correct attributes. This test is very straightforward. Line 17 and 22 return candidates 1 and 2. The following lines check that each attribute corresponds to the one we have assigned in the smart contract. This way we make sure that the program is working the way we intended them to do. Now we can run the tests and check that everything works as intended:

```
david@ubuntu:~/Desktop/Project2.0/election$ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


  Contract: Election
    ✔ it initializes the application with two candidates
    ✔ candidates initialized with the corresponding attributes (61ms)


  2 passing (150ms)
```

Figure 18: Running main tests

5.9.4 Frontend

Next, the development of the main frontend is described. It represents what happens in the smart contract. We will use the Bootstrap framework [21] to avoid the necessity of writing any CSS during the project. The code on the frontend is quite easy to follow since it is made using HTML and JavaScript code and does not contain any complex features. Also, thanks to the template acquired by the pet-shop box feature, we already have the main structure made. We will not describe the code but will mention a few key points that are more related to what our interest remains which is Blockchain and the usability with the smart contract.

We make use of a JavaScript library mentioned web3.js. This allows our frontend to talk directly to the blockchain in our case the Ganache. All that has to be done is configure it. It can be found in the initWeb3 function. To allow it to interact with our contract, all that we require is to put the deployed instance of the contract into the function. Finally, the render function is in charge of loading the contents of the contract to the frontend.

Then, we go over the map of candidates to render each one of them, and finally, we also added a function to show the current user address.

So far, the frontend looks like this:

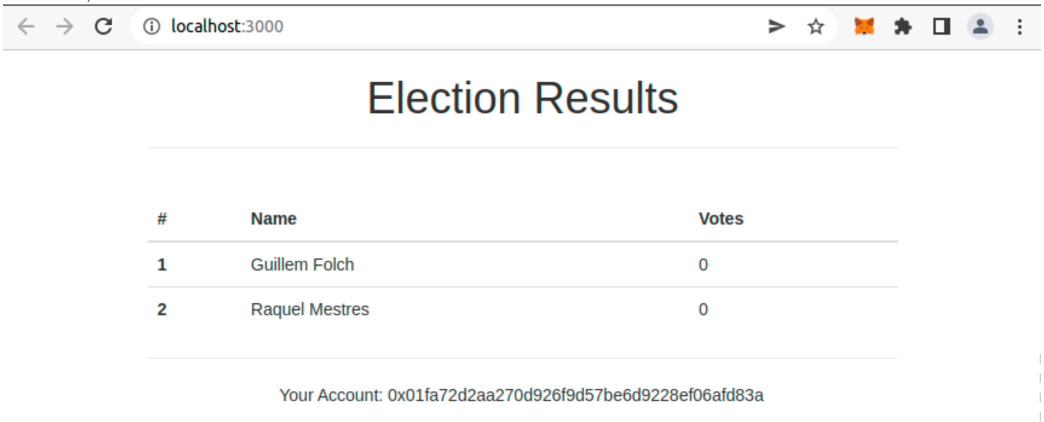


Figure 19: Initial Frontend

It is important to mention that to view this, it is required to “log in” to the blockchain. This means that you have to import one Ganache account into the Metamask extension. Once all is set up, running the command “npm run dev” will execute the application which is run in the port 3000.

5.9.5 Vote casting

So far, we have created the simplest working interface. A smart contract that creates two candidates and a frontend connected to the backend using the Metamask as a bridge. This is so far the toughest part of the project. From this point, it all comes to creativity, time, and effort to start implementing new functionalities and testing them, making sure they are secure. We say that the toughest part is done because we no longer have to link both backend and frontend, everything is now linked.

In this section, we are going to introduce the vote casting for an election to take place. The first thing needed will be a map to monitor the accounts that have already voted. And consequently, a function to allow the voting.

```
34    mapping(address => bool) public voters;
35    function vote (uint _idCandidate) public {
36
37        require(!voters[msg.sender]);
38        require(_idCandidate > 0 && _idCandidate <= candidatesCount);
39
40        voters[msg.sender] = true;
41        candidates[_idCandidate].voteCount ++;
42    }
```

Figure 20: Voting function

In line 34 is the map dedicated to storing the voter's addresses. Following, in line 35 is the voting function. It gets as an argument the id of the candidate that is being voted. The function is set to public since any account should be allowed to call it. Inside the function, we placed two requirements, the first one is meant to check that the account hasn't yet voted. And the second requirement is intended to check that the candidate id is valid. Finally, if these requirements are met, the address of the voter is stored on the map and set to true. Now, if this account would try to vote again the first requirement should stop it from happening. Finally, we increment the number of votes of the candidate.

5.9.6 Vote Testing

With the implementation of the voting function, several things should be tested to ensure the proper operation of the contract. The main one, check that the vote count of a candidate gets incremented once a vote is executed. Then, check that the account of the voter is added to the map and set to true. The test would look like this:

```
it("voter can cast vote", function() {
  return Election.deployed().then(function(instance) {
    electionInstance = instance;
    candidateId = 1;
    return electionInstance.vote(candidateId, { from: accounts[0] }).
  }).then(function(receipt) {
    return electionInstance.voters(accounts[0]);
  }).then(function(voted) {
    assert(voted, "the voter set to true");
    return electionInstance.candidates(candidateId);
  }).then(function(candidate) {
    var voteCount = candidate[2];
    assert.equal(voteCount, 1, "vote count incremented");
  })
});
```

Figure 21: Test to allow voters to cast a vote

Another test should be implemented to check that the function returns an error when there is an invalid candidate. When testing, it is possible to test for errors, which means that, to pass the test, an error must pop up. In our case, we will not only seek for an error, but we will dig into the error, in search of a keyword. This is to make sure that the returned error from the function is the error that we want and not another one instead.

```
  it("throws exception if candidate is invalid", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      return electionInstance.vote(99, { from: accounts[1] })
    }).then(assert.fail).catch(function(error) {
      assert(error.message.indexOf('revert') >= 0, "message contains revert");
      return electionInstance.candidates(1);
    }).then(function(candidate1) {
      var voteCount = candidate1[2];
      assert.equal(voteCount, 1, "candidate did not recieve a vote");
      return electionInstance.candidates(2);
    }).then(function(candidate2) {
      var voteCount = candidate2[2];
      assert.equal(voteCount, 0, "candidate did not recieve a vote");
    });
  });
```

Figure 22: Test for an invalid candidate.

Now, the final test for this function is to check that no account is allowed to vote twice. This is quite straightforward. We will get an account that has not voted yet. We will vote and afterward, we will try to vote again. This situation should produce an error. Finally, it is also important to check that although the error has occurred, no votes have been counted when attempting this. This is how the test looks:

```
  it("exception thrown when double voting", function() {
    return Election.deployed().then(function(instance) {
      electionInstance = instance;
      candidateId = 2;
      electionInstance.vote(candidateId, { from: accounts[1] });
      return electionInstance.candidates(candidateId);
    }).then(function(candidate) {
      var voteCount = candidate[2];
      assert.equal(voteCount, 1, "first vote");
      // Try to vote again
      return electionInstance.vote(candidateId, { from: accounts[1] });
    }).then(assert.fail).catch(function(error) {
      assert(error.message.indexOf('revert') >= 0, "message contains rever");
      return electionInstance.candidates(1);
    }).then(function(candidate1) {
      var voteCount = candidate1[2];
      assert.equal(voteCount, 1, "candidate did not recieve a vote");
      return electionInstance.candidates(2);
    }).then(function(candidate2) {
      var voteCount = candidate2[2];
      assert.equal(voteCount, 1, "candidate did not recieve a vote");
    });
  });
});
```

Figure 23: Test for double voting

When we run all the tests together an output similar to the following one should appear:

```
david@ubuntu:~/Desktop/Project2.0/election$ truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


  Contract: Election
    ✔ it initializes the application with two candidates (39ms)
    ✔ candidates initialized with the corresponding attributes (61ms)
    ✔ voter can cast vote (192ms)
    ✔ throws exception if candidate is invalid (327ms)
    ✔ exception thrown when double voting (194ms)


  5 passing (905ms)
```

Figure 24: Output from running the tests

5.9.7 Vote client-side

Next up is to develop the client-side of the new voting function. The easiest idea we have come up with is to add a form in the HTML file. This form will contain all the available candidates that we can get from the smart contract. Then, thanks to the onSubmit handler from the form we can make a call to a function that will cast the vote from the app.js file. We will have to add the function there. The HTML file should look similar to this:

```
<form onSubmit="App.castVote(); return false;">
  <div class="form-group">
    <label for="candidatesSelect">Choose Candidate</label>
    <select class="form-control" id="candidatesSelect">
    </select>
  </div>
  <button type="submit" class="btn btn-primary">Vote</button>
  <hr />
</form>
```

Figure 25: HTML code for the voting form

We do not have to worry about any CSS since the Mocha framework will make everything look modern and stylish. We will see the result once we render the final application. We now have the HTML file done but this will not work since the app.js file does not contain the logic yet. There are two things to be done. First, we have to load all the candidates inside the form. Secondly, the castVote function has to be developed.

For the first part, we have to update the render function. The render function must first load the account data and check for errors. Then, load the contract data. After this, we need to iterate over the map of candidates from the first one to candidatesCount. To render them, we have to add all the candidates together. Since I am not an HTML expert and I have no other idea of how to go over this, I have seen online that I can append them in the following way:

```
var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" +
      voteCount + "</td></tr>"
candidatesResults.append(candidateTemplate);
```

Figure 26: Candidates append for render

We can do the same for the ballot option, like so:

```
var candidateOption = "<option value='" + id + "' >" + name + "</ option>"
candidatesSelect.append(candidateOption);
```

Figure 27: Candidates' ballot option

Also, since no account can vote twice, we hide the form once the hasVoted function returns true. Here is the code for the loop over the candidates and the hiding part of the form.

```
for (var i = 1; i <= candidatesCount; i++) {
  electionInstance.candidates(i).then(function(candidate) {
    var id = candidate[0];
    var name = candidate[1];
    var voteCount = candidate[2];


    var candidateTemplate = "<tr><th>" + id + "</th><td>" + name + "</td><td>" +
          voteCount + "</td></tr>"
    candidatesResults.append(candidateTemplate);

    var candidateOption = "<option value='" + id + "' >" + name + "</ option>"
    candidatesSelect.append(candidateOption);
  });
}
```

Figure 28: Candidate iteration in app.js

```
}).then(function(hasVoted) {
  // Do not allow a user to vote
  if(hasVoted) {
    $('form').hide();
  }
  loader.hide();
  content.show();
}).catch(function(error) {
  console.warn(error);
});
```

Figure 29: Hide form when a voter has already voted

Finally, we have to develop a function that gets called every time the form is submitted. The castVote function is in charge of executing the vote function from the smart contract. To do so, we get the candidate Id from the candidateSelect from the form and pass it as an argument to the instance.vote. This will result in the account voting for this specific candidate. We have tested this feature before so we know that the function will act as expected. We also hide and wait to show the results again until the votes get updated.

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

Figure 30: Cast votes in the app.js

Finally, this is what the frontend should look like:

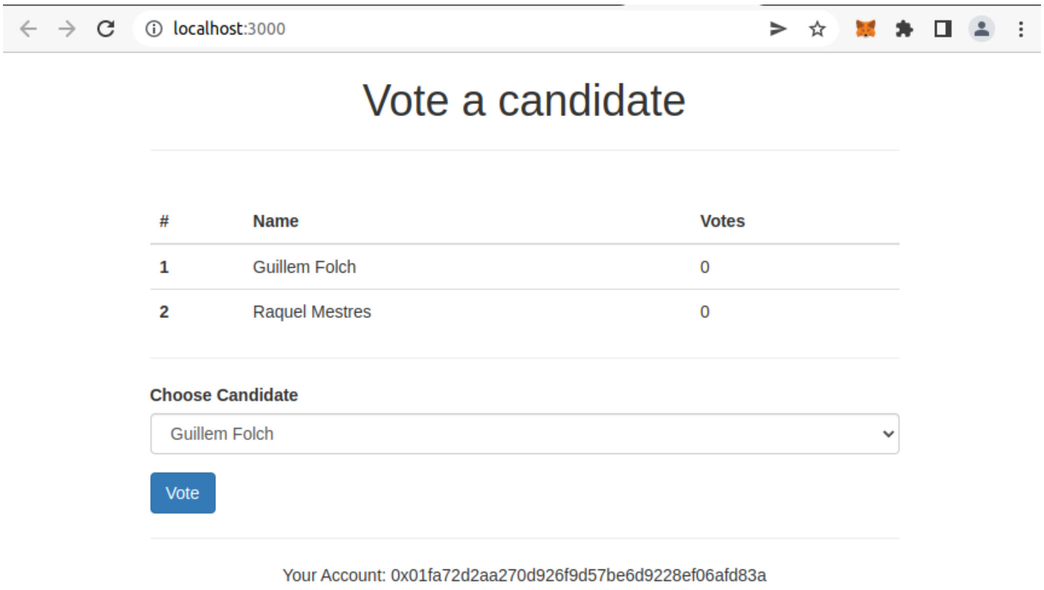


Figure 31: Frontend

This already allows us to cast a vote. The operation of the application is shown in the Annex at the end of the project. So far, the only issue with the application is that you must refresh the page to see the changes in the vote count. In the following section, we will add events so that this is no longer an issue.

5.9.8 Event trigger

The very final step of the practical part is to integrate event triggers on the application and update the frontend once voting occurs. This is indeed a very straightforward process and should not be difficult. To add an event is as easy as to declare it on the smart contract. Then, we just need to add this event into the function that we want the trigger to happen. In our case, we add the event to the vote function. Like so:

```
event votedEvent (
    uint indexed _idCandidate;
);
```

Figure 32: Event declaration in Election.sol

```
function vote (uint _idCandidate) public {

    require(!voters[msg.sender]);
    require(_idCandidate > 0 && _idCandidate <= candidatesCount);

    voters[msg.sender] = true;
    candidates[_idCandidate].voteCount ++;

    emit votedEvent(_idCandidate);
}
```

Figure 33: event added to the vote function in Election.sol

It is good to notice that once the smart contract is updated, we have to do new migration to deploy the contract again into the blockchain. Otherwise, the implementation will not take place.

On its own, this event will do nothing to our application, we have to update the client-side application to notice when an event happens. Consequently, refresh the page once this specific event happens. To do this, we update the app.js file.

```
listenForEvents: function() {
  App.contracts.Election.deployed().then(function(instance) {
    instance.votedEvent({}, {
      fromBlock: 0,
      toBlock: 'latest'
    }).watch(function(error, event) {
      App.render();
    });
  });
```

Figure 34: Configure event to refresh the frontend

Essentially what this code does is pay attention from block 0 to the latest, which means that it checks all the events that happen on the blockchain. The watch function will execute every time a votedEvent is registered. This function is set to render again the whole frontend. Consequently, the vote count gets automatically updated and the user is no longer in charge of refreshing the page to see the new results.

Several issues appeared when trying to make this work. And apparently, there is a common issue with MetaMask extension from Chrome that makes events not work as intended. Below is the link to the issue. Therefore, by restarting Chrome the issue seems to solve itself. Despite that, I link the issue. Issue: https://github.com/MetaMask/metamask-extension/issues/2393

We now have a fully developed and tested a decentralized application to run elections on the Ethereum blockchain. It is possible to vote from the frontend and watch the votes happen in real-time.

Conclusions

The main goal of this research is to analyze and evaluate the current research on blockchain-based electronic voting systems. During the project, there is a deep theoretical aspect about both voting systems and blockchain. Finally, in the last part, both are combined to create a fully operating system. This developed system is thought to be expanded with plenty of new features such as login, administrator interface, and timer… but yet the process to get to this stage has shown all the important phases. From choosing the tools, developing the contract and the frontend, testing the functionalities, and finally linking the smart contract together with the frontend to make everything work. From now on, it is only needed to add and test the new functionalities.

To conclude with a personal opinion, electronic voting is theoretically achievable, the major challenge of electronic voting lies in converging a social system and behavior with an electronic electoral process. The design of physical architecture to support the blockchain network as well as the cryptographic algorithms to be implemented are vital to support a process of such magnitude as a popular election. Sooner or later, an electronic electoral process must be carried out, either because of adaptation to the new technological reality or because society itself will demand it, as maximum transparency is necessary for the processes, as well as being auditable. Society must be prepared for regulatory, legislative, and judicial matters, among others, because both the social system and technology, must live in a new social-technological era. Electronic voting with blockchain is not the solution to an electoral process, but rather the evolution of this process, given that society has been evolving according to the use of technology, so must the electoral system.

Annex: User manual

System functionalities:

- Load all the available candidates
- Choose and anonymously vote among all the candidates
- Real-time update of all the votes

PREPARE THE SETUP

Open Ganache and create a new workspace with the Quickstart button. This will generate 10 test accounts and save the workspace. Then go to the /election directory and run "npm install" to install all dependencies.

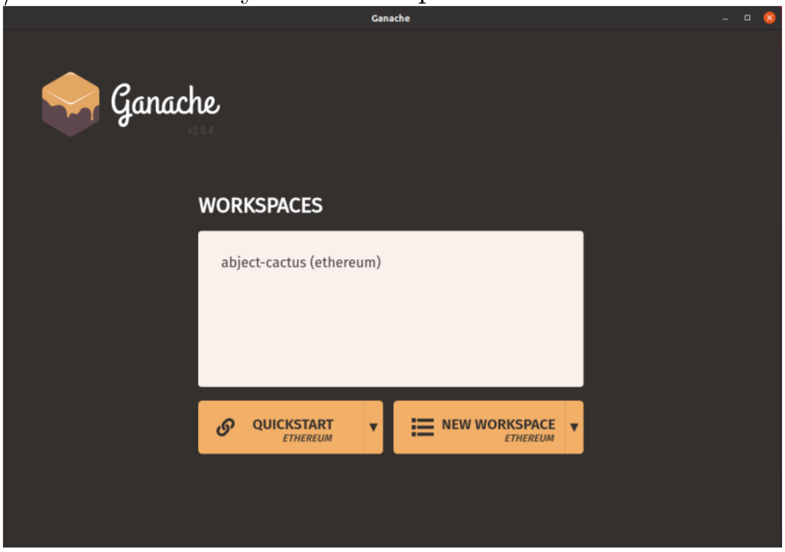


Figure 35: Main page ganache

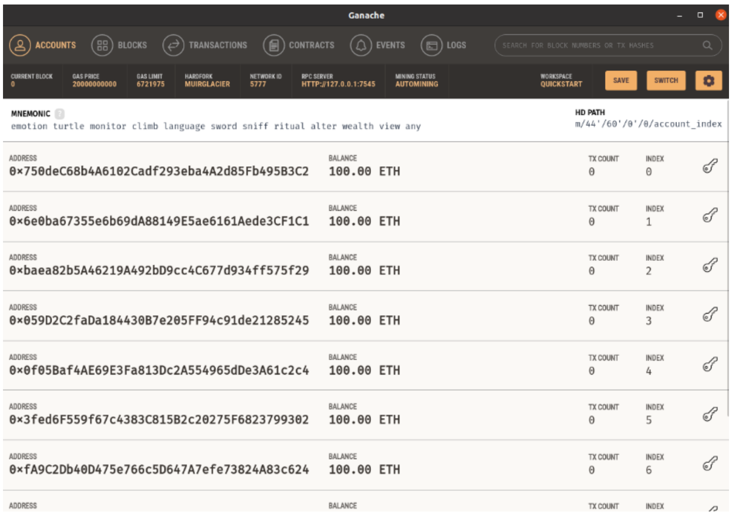


Figure 36: Ganache workspace and account

LOAD THE NETWORK TO METAMASK

Open chrome and Metamask extension. Click on Add Network. An add a new network with the values shown in the Figure. The RPC Url comes from the ganache interface.

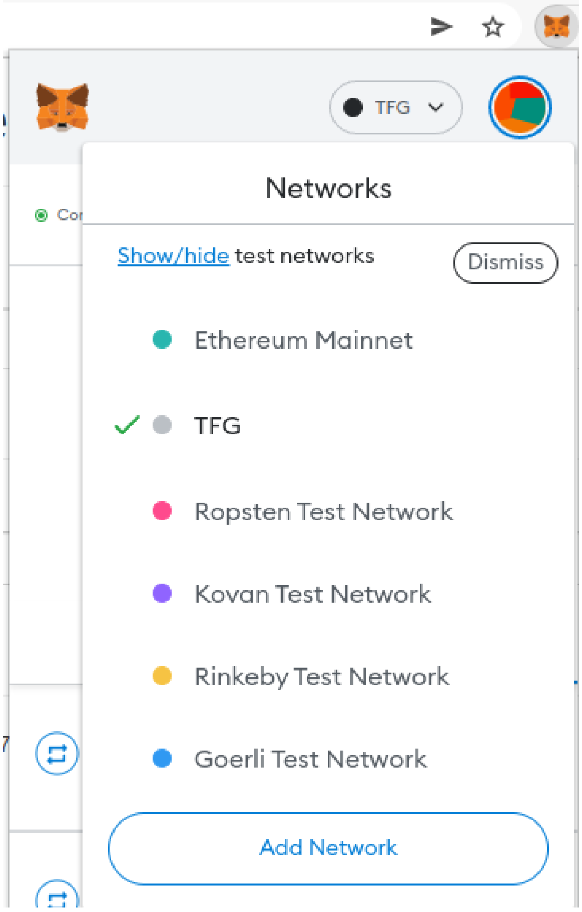


Figure 37: Main page Metamask

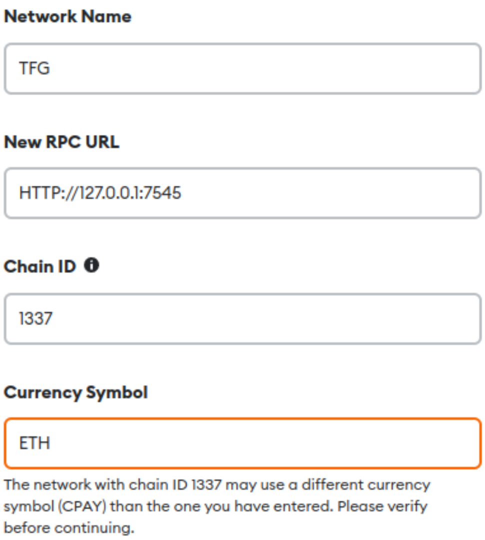


Figure 38: Network parameters

IMPORT GANACHE ACCOUNTS TO METAMASK

To connect the Metamask extension to the Ganache local blockchain we have to import the Ganache accounts into Metamask. Copy the private keys from Ganache and add them to Metamask. This will connect both services. Once the account is imported, it has to be connected.

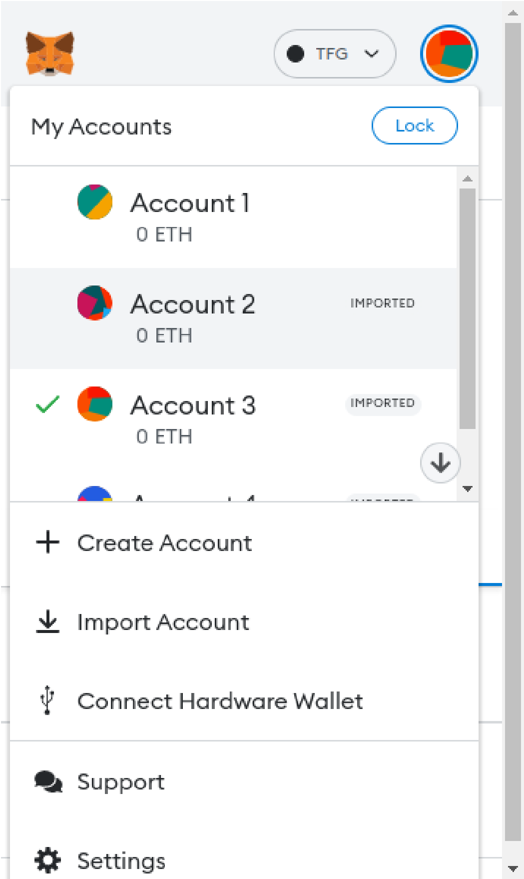


Figure 39:Import accounts

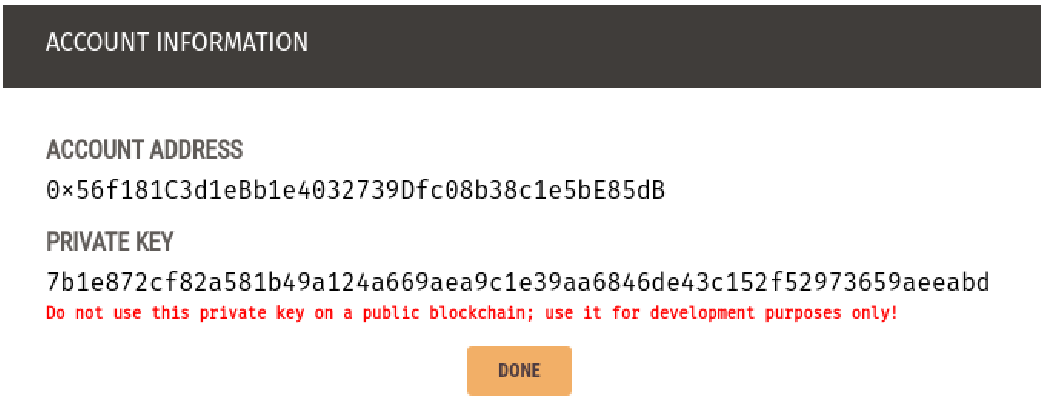


Figure 40:Private key of one account

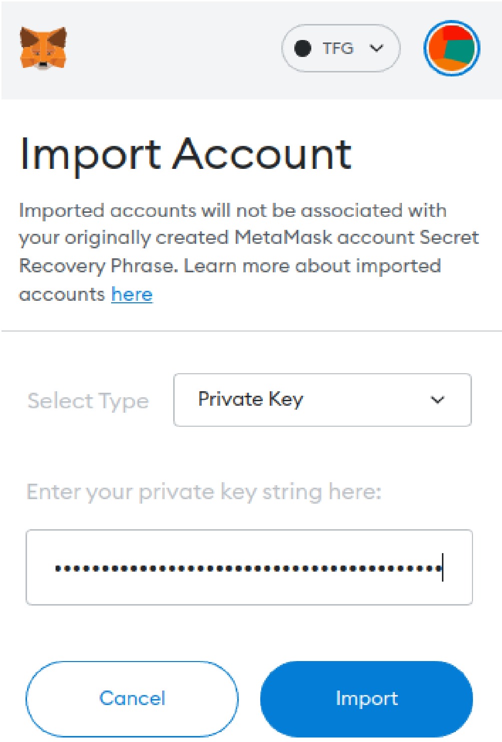


Figure 41: Importing account

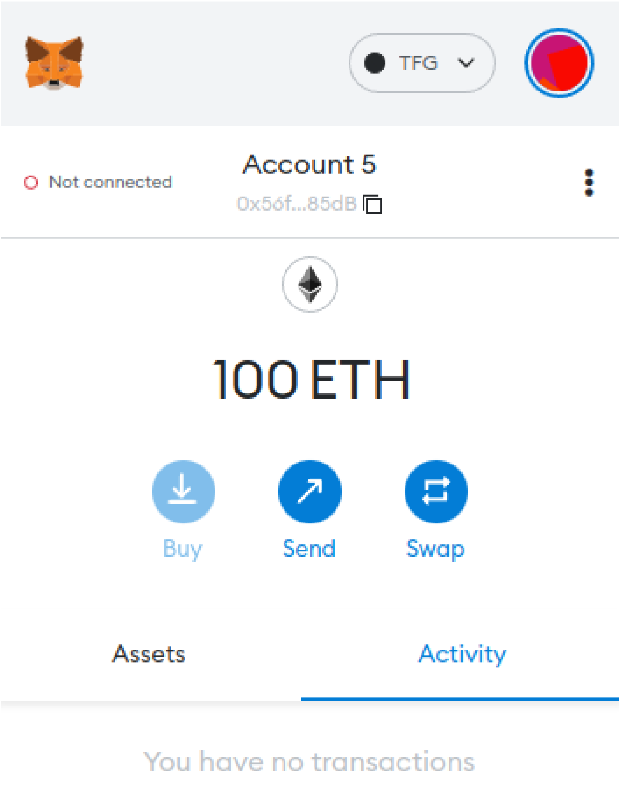


Figure 42: Account imported but not connected

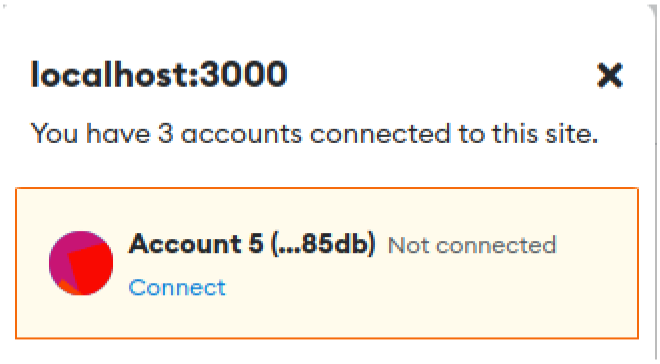


Figure 43: Connect account

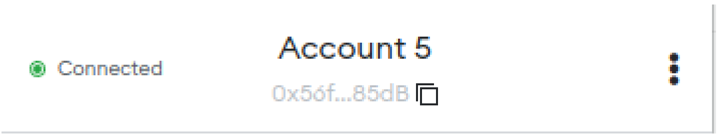


Figure 44: Connected

VOTE A CANDIDATE

Select a candidate from the form a click on Vote. After a few seconds, a Metamask notification will pop up to confirm the transaction. Once accepted, this transaction will then be registered on the blockchain. After that, the frontend will automatically refresh and the vote button will no longer be available, unless we change to an account that has not yet voted.

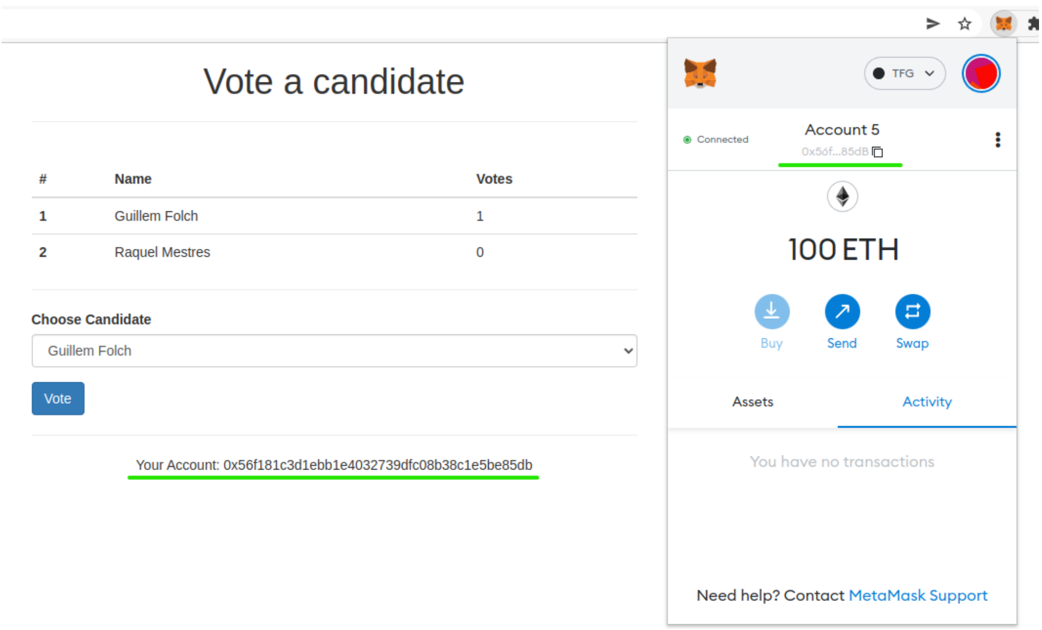


Figure 45: Same accounts on frontend and Metamask

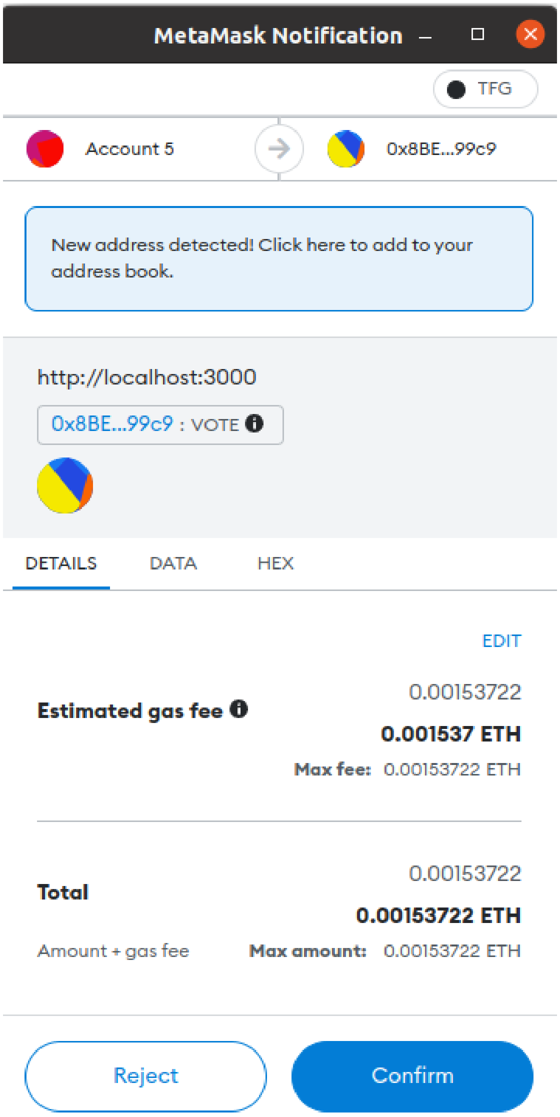


Figure 46: Transaction to vote

```
[3:07:25 AM] eth_sendRawTransaction
[3:07:28 AM]   Transaction: 0x23e819cb5d5b8f1b595b8835e42323546445141f868678b11c68ba3479efd789
[3:07:28 AM]   Gas usage: 51241
[3:07:28 AM]   Block Number: 158
[3:07:28 AM]   Block Time: Fri Jun 17 2022 03:07:28 GMT-0700 (Pacific Daylight Time)
[3:07:28 AM] eth_getBlockByNumber
```

Figure 47: Logged transaction in the Blockchain

Figure 48: Transaction history in Ganache. (FROM ADDRESS = our account)

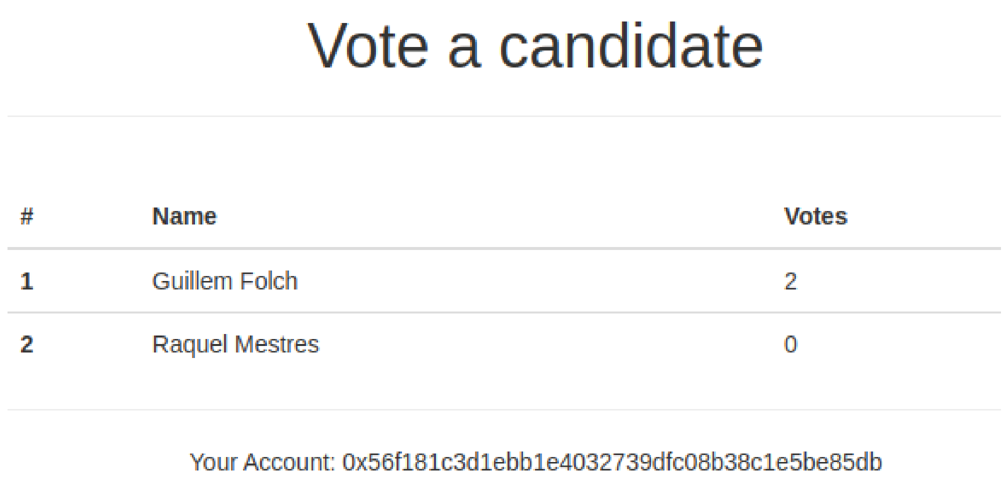


Figure 49: Vote is counted and it is no longer possible to vote again with this account

Table of figures

References

[1] "Statista," [Online]. Available: https://www.statista.com/statistics/419513/average-annual-wages-spain-y-on-y-in-euros/. [Accessed 03 March 2022].

[2] A. Hayes, "Blockchain Explained," Investopedia, [Online]. Available: https://www.investopedia.com/terms/b/blockchain.asp. [Accessed 01 March 2022].

[3] C. Dictionary, "Cambridge," 15 05 2022. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/internet.

[4] R. Gibb, "Stack Path," [Online]. Available: https://blog.stackpath.com/distributed-system/. [Accessed 15 05 2022].

[5] Wikipedia, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Electronic_voting. [Accessed 15 05 2022].

[6] @minimalism, "INTRODUCTION TO SMART CONTRACTS," Etherum.org, [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/. [Accessed 01 March 2022].

[7] "Zero-Knowledge proof," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Zero-knowledge_proof. [Accessed 01 March 2022].

[8] M. Hogan, "Duval County," [Online]. Available: https://www.duvalelections.com/General-Information/Learn-About-Elections/History-Of-Elections.

[9] T. Kohno, A. Stubblefield, A. Rubin and D. Wallach, "IEEE Xplore," [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1301313.

[10] J. B. Carlos Vegas, Overview of Current State of E-Voting Worldwide, 2016.

[11] D. A. Gritzalis, Secure Electronic Voting, Springer Science, 2003.

[12] D. A. Shukla, Electronic Voting Machines - The True Story, 2018.

[13] O. J. Sunday, "GitHub," [Online]. Available: https://github.com/jobic10/e-voting-with-django.

[14] J. A. M. M. Kashif. M, "University of West London," [Online]. Available: https://core.ac.uk/download/pdf/155779036.pdf.

[15] M. J. Z. S. Uzma. J, "Sensors," [Online]. Available: https://www.mdpi.com/1424-8220/21/17/5874/pdf.

[16] M. K. Pratt, "Tech Target," [Online]. Available: https://www.techtarget.com/searchcio/definition/distributed-ledger. [Accessed 15 06 2022].

[17] A. Kumar, "Vital Flux," [Online]. Available: https://vitalflux.com/what-is-blockchain-how-does-it-work/. [Accessed 15 06 2022].

[18] I. Mitic, "fortunly," Blockchain Statistics, [Online]. Available: https://fortunly.com/statistics/blockchain-statistics/. [Accessed 20 05 2022].

[19] M. Murthy, "Medium," 2 January 2017. [Online]. Available: https://medium.com/@mvmurthy/ethereum-for-web-developers-890be23d1d0c. [Accessed 28 April 2022].

[20] "Mocha," [Online]. Available: https://mochajs.org/.

[21] "BootStrap," [Online]. Available: https://getbootstrap.com/.

[22] "nvotes," [Online]. Available: https://nvotes.com/security/.