UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DESIGN OF A DECENTRALIZED SOCIAL NETWORK

JAVIER HERNÁNDEZ SÁNCHEZ

2023

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Título: Design of a decentralized social network.

Autor: D. Javier Hernández Sánchez

Tutor: D. Alejandro Antonio Alonso Muñoz

Ponente: D. ……

Departamento: ……

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……

Secretario: D. ……

Suplente: D. ……

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 2023

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DESIGN OF A DECENTRALIZED SOCIAL NETWORK

JAVIER HERNÁNDEZ SÁNCHEZ

2023

SUMMARY

This project explores the potential of decentralized technologies to address the problems of traditional platforms. The main problem with current platforms is their high centralization, being controlled exclusively by a single company. This centralization allows the company to exploit its market power for its own purposes, generating other serious problems for its user base. Therefore, the design of a standard is proposed that platforms can adopt for the development of decentralized and multi-company applications, which effectively face centralization. Subsequently, a prototype social network is developed which will serve as a proof of concept, to evaluate the designed standard in a realistic environment.

RESUMEN

Este proyecto explora el potencial de las tecnologías descentralizadas para abordar los problemas de las plataformas tradicionales. El principal problema de las plataformas actuales es su elevada centralización, estando controladas exclusivamente por una única compañía. Esta centralización permite a la compañía explotar su poder de mercado para fines propios, generando a su vez otros problemas de gran gravedad sobre su base de usuario. Por tanto, se propone el diseño de un estándar que las plataformas pueden adoptar para el desarrollo de aplicaciones descentralizadas y multicompañía, que enfrenten de manera efectiva la centralización. Posteriormente, se desarrolla una red social prototipo que servirá como prueba de concepto, para evaluar el estándar diseñado en un entorno realista.

KEYWORDS

Decentralization, social media, blockchain, digital identity, self-sovereign identity, peer-to-peer networks, content platforms, Ethereum, censorship resistant, artificial intelligence, standard.

PALABRAS CLAVE

Descentralización, redes sociales, blockchain, identidad digital, identidad auto soberana, redes peer-to-peer, plataformas de contenido, Ethereum, resistente a la censura, inteligencia artificial, estándar.

Content

Figures

Tables

Equations

Abbreviations and acronyms

2FA	Two-Factor Authentication
AGI	Artificial General Intelligence
AI	Artificial Intelligence
AICF	Artificial Intelligence Content Filter
AS	Access Signature
API	Application Programming Interface
Blockchain	Chain of blocks
Bootstrap	CSS Library
CSS	Cascading Style Sheets
DAG	Directed Acyclic Graph
DApp	Decentralized Application
DeFi	Decentralized Finance
DNS	Domain Name System
DoS	Denial of Service
Drizzle	ReactJS library
E2E	End-to-end
ECC	Elliptic Curve Cryptography
EIP	Ethereum Improvement Proposals
ETH	Ether
Ethereum	Ethereum blockchain
EVM	Ethereum Virtual Machine
ExpressJS	NodeJS framework
Filecoin	Filecoin blockchain
Ganache	Ganache blockchain
GnuPG	Gnu Privacy Guard
Gossip protocol	Gossip protocol is a message distribution protocol
GPT	Generative Pre-trained Transformer
Hash	The output value of a hash function
Hash function	Function that maps data of arbitrary size to fixed-size values
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFS	Interplanetary File System
JS	Javascript
Man-in-the-middle	Cyberattack where the attacker has interposed himself between two communicating parties
Merkle trees	Tree in which every node is labelled with a cryptographic hash of the child nodes

Mine	The act of issuing a new block in PoW blockchains
Miners	The nodes that mine blocks in blockchain
MPT	Modified Merkle Patricia Trie
NAT	Network Address Translation
NodeJS	Javascript runtime environment
Nonce	Solution to a mathematical puzzle in PoW blockchains
NPL	Natural Language Processing
Npm	Node package manager
Off-chain	Processing a transaction outside the blockchain
OpenAI	OpenAI Company
P2P	Peer to peer
Plug-and-play	Easy to connect
PoR	Proof of Retrieval
PoS	Proof of Stake
PoSt	Proof of Spacetime
PoSto	Proof of Storage
PoW	Proof of Work
Proxy	Server that acts as an intermediary between two clients
Radix trie	Radix tree
ReactJS	JavaScript framework for developing client interfaces
Redux	Open source JavaScript library
Recharts	Open source JavaScript library
RFC	Request for Comments
RPC	Remote Procedure Call
RSA	Ron Rivest, Adi Shamir and Leonard Adleman cryptographic algorithm
SAS	Shared Access Signature
Solidity	Smart Contract Programming Language
SSI	Self Sovereign Identity
SSS	Shamir's Secret Sharing
Trie	Tree
Truffle	Open source JavaScript library
Tx	Transaction
U.S.	United States
W3C	World Wide Web Consortium
Web3	World Wide Web ''version'' which incorporates decentralization
Web3	Open source Javascript library for interacting with blockchains
ZKP	Zero-Knowledge Proofs

1. Introduction to decentralized social media

In today's interconnected world, social media and content platforms have become an essential part of our daily lives, allowing us to connect with others, share information, and express ourselves in ways that were previously unimaginable. However, despite the many benefits of these platforms, their centralized nature has given rise to several critical issues, most of them common among all players in the industry. Among these concerns, we find censorship, content manipulation, lack of data governance, massive bot attacks, impersonation issues, as well as privacy and security vulnerabilities.

Centralized platforms are controlled by a small group of individuals or corporations, who have the power to impose their vision, manipulate content, and collect and monetize user data. This has led to concerns about data governance, the spread of misinformation, and the suppression of free speech. To address these issues, there is a growing need for decentralized platforms that prioritize user privacy, security, control, and provide a more transparent environment for social interaction and information sharing.

This research project aims to solve the problems of centralized platforms by proposing a technical solution. For this purpose, several cutting-edge technologies will be used such as blockchain, cryptography, and even artificial intelligence. To achieve this, not only the technical challenges will be covered, but also the scalability and the economic feasibility will be considered. Since a computing infrastructure is needed for the adoption to take place, the proposed solution should be financially feasible, so that when it is released to the market, it is attractive enough for the infrastructure firms.

To demonstrate the utility and effectiveness of the proposed design, a prototype social network will be developed and tested within a realistic environment. This proof of concept will allow the proposed solution to be evaluated, allowing for interaction between different network actors and simulating real-world conditions.

The findings of the project will contribute to the ongoing efforts towards decentralization and data governance in the digital age, offering a promising alternative to traditional platforms and building a future where individuals have greater control over their data.

1.1. Objectives and work plan

The main objective of this project is to conceptualize and build a decentralized social network that mitigates the existing problems of centralized platforms. To accomplish this we will follow the work plan illustrated in **Figure 1**. First, in **Chapter 2** we will undertake a deep analysis of the problems of centralized platforms. After having a clear view of the issues, in **Chapter 3** we will define the requirements that the decentralized platform we will design must implement.

Following this, in **Chapter 4**, we will explore the main decentralized technologies. These technologies will equip us with the necessary technical knowledge to design a decentralized social network. This design is carried out in **Chapter 5**. Once the design is clear, the solution is technically implemented in **Chapter 6**. This implementation will serve as a proof of concept that will be tested later in a realistic environment with different network participants during **Chapter 7**. Finally, we will extract the main conclusions and difficulties encountered during the development of the project, which will serve as a basis for future developments in the field of decentralization.

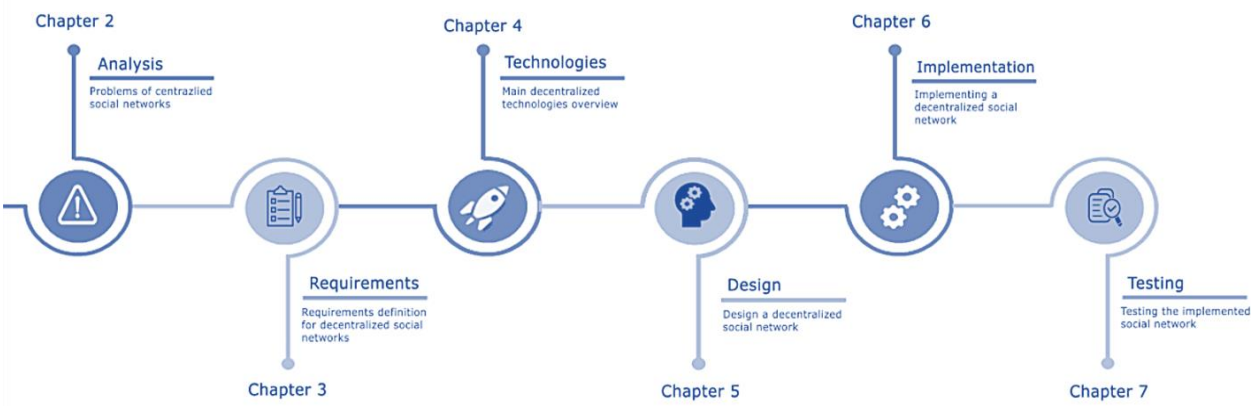

Figure 1 - Work plan

2. Analysis: The problems of centralized social media

Over the past few years, social media has become a fundamental part of our lives and has revolutionized the way we communicate and interact with each other. However, as the use of social media platforms continues to increase, so too, do the problems associated with their use. Of all the issues, centralization is probably the biggest concern, as it is arguably the root of most social media problems.

Throughout this document, you will notice that I am going to keep using the word "centralized" to refer to the fact that social media is operated and controlled by a single company or provider, and not to the fact that social media is centered designed relying on a single server that performs all the heavy lifting. In fact, social media despite being operated by a single company is highly distributed across many computers and storages around the world. Distributed systems have many advantages like fault tolerance, higher availability and lower latency through replication and sharding techniques. **Figure 2** shows a centralized and a distributed architecture.

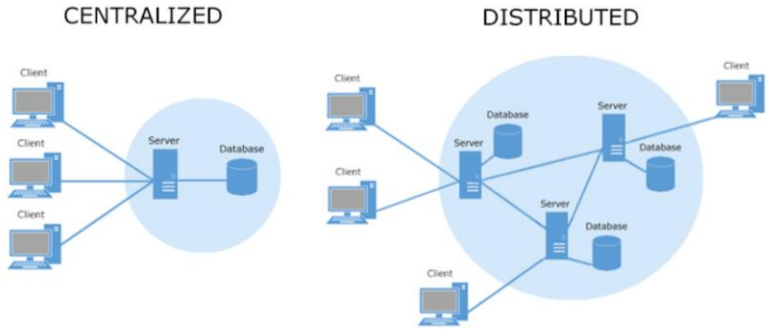

Figure 2 - Centralized vs distributed architectures

However, this section does not explore the disadvantages of using centralized architectures since social networks are already distributed, it explores the disadvantages of centralized platforms in terms of ownership, in other words, platforms controlled by a single infrastructure company. **Table 1** shows the main social media platforms alongside their holding company, and it illustrates the majority of this platforms are ruled by one single corporation. However, the impact of centralization is not limited to social media alone, rather, it affects other content platforms, including audio and video streaming services. This is what **Appendix A** illustrates.

SOCIAL MEDIA
Meta
Meta
ByteDance
XCORP
Snap Inc.

Table 1 - Owners of the top social media platforms

If we want to build a decentralized platform that solves the existing problems, first we must understand the current problems. For this reason, in this chapter we will explore the issues surrounding centralized social media, complemented by real-world situations.

2.1. Platforms censorship and its threats to free speech

The biggest problem of centralized platforms is probably censorship. Centralized social media have often been criticized for attacking and threatening free speech. These platforms have considerable power to control the flow of information and shape public discourse. However, this power can be subject to abuse, whether intentional or unintentional.

Most platforms rely on automated tools or human moderators to assess and remove content that violates their internal policies and guidelines, based on criteria such as hate speech, pornography, or violence.

Besides enforcing their own policies, these platforms face external pressures from governments, interest groups, and advertisers to restrict or remove content that could be deemed a threat to national security, public order, or offensive in nature. Consequently, censorship can come from both internal and external sources.

While some of these requests may be legitimate, they can lead to excessive restrictions on free speech. In some cases, extra-legal means have been employed to force platforms to remove content, raising concerns about the erosion of free speech and the potential for censorship to be used as a tool of repression.

The emergence of "cancellation culture" exemplifies this trend, as the suppression of unpopular or minority viewpoints becomes more widespread. This phenomenon not only threatens free speech, but also undermines the diversity of ideas and perspectives that are essential to a healthy democratic society.

Platforms face the challenge of striking a balance between promoting free speech and protecting users from spreading harmful content. This conflict gives rise to the "security versus freedom" dilemma, illustrated in the Venn diagram of **Figure 3**, where a choice must be made between freedom or security, the two options being mutually exclusive.

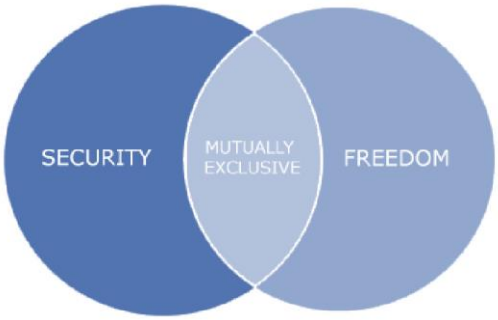

Figure 3 - Security versus freedom dilemma

Although censorship might provide some degree of safety and protection, protecting users from the spread of harmful content, it also restricts the fundamental right to free speech. This clearly demonstrates the mutual exclusivity between the two concepts.

The **Figure 4** provides a visual timeline of several historical events related to the regulation of free speech and censorship, illustrating the challenges of finding a balance between these two opposing forces.

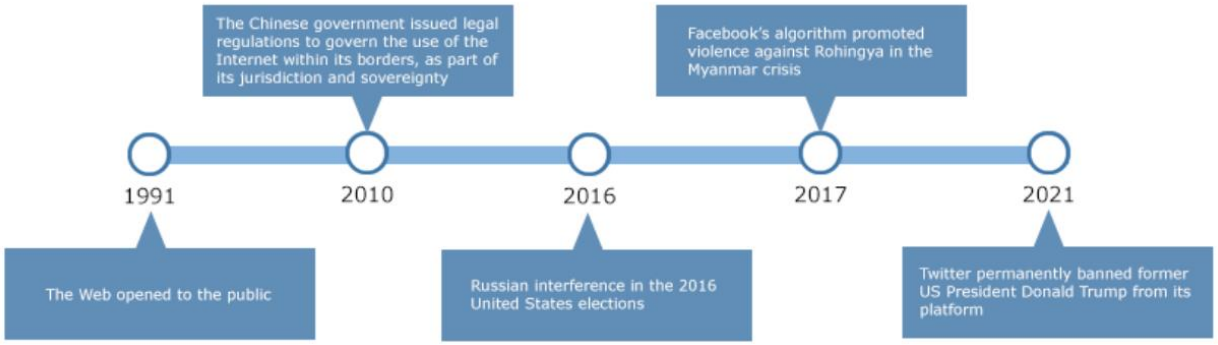

Figure 4 - Timeline of historical events related to censorship and free speech

On one side of the spectrum, we find Twitter's permanent ban of former U.S. President Donald Trump and the Chinese government's censorship, which imposed unilateral content restrictions on all platforms in China. Both events exemplify the potential risks of censorship, as they threaten to undermine free speech and diversity of ideas.

On the other side, Facebook's handling of the Myanmar crisis and the use of social media by Russian agents to interfere in the U.S. presidential election in 2016 demonstrate the potential consequences of free speech absolutism, where harmful content and misinformation can spread causing significant harm.

These historical moments capture the complex challenges social media platforms face when balancing free speech and censorship. Striking a balance between protecting free speech and ensuring security is a challenging task. While some degree of moderation is necessary to maintain a safe and respectful environment, platforms must be committed to upholding the principles of free expression and open discourse.

According to the "security versus freedom" dilemma, it is impossible to simultaneously choose between unrestricted free speech and security at the same time, for this reason, this project will prioritize free speech over security to build a censorship-resistant network. However, it will also be essential to build additional layers that can be placed on top of this network to achieve a high level of security. These will be discussed later, but for the time being, the focus is still on the remaining existing problems of centralized social networks.

2.2. Biased platforms and manipulation of content

It is no secret that there is an inherent bias within social media, due to the centralized nature of these platforms, which compromises their impartiality and gives rise to non-neutral decisions on content. For this reason, content manipulation and ideological imposition on users is probably the second major concern regarding centralized platforms. The companies behind these platforms have the power to control and shape user interactions, even without users realizing what is happening behind the scenes. This chapter examines the mechanisms by which centralized platforms can influence and control their user base, as it is crucial to understand them in order to tackle them later when developing the standard.

There are several ways in which social networks can manipulate their users, algorithmic manipulation being the foremost and most important of them all. Given the overwhelming volume of content present on these platforms, manual review becomes impractical, leading these platforms to rely mostly on automated tools. Centralized platforms can develop algorithms that prioritize certain types of content over others in search or recommendation processes, leading to a distorted perception of reality. This may amplify extreme views, suppress minority viewpoints, and create "echo chambers" where users are exposed only to ideas that confirm their existing beliefs. Usually, these algorithms have artificial intelligence behind them, which can be biased during the training process [1]. Although this technique is often used unintentionally, it can also be used intentionally as well.

Censorship, an issue previously discussed, is another technique platforms appeal to manipulate content. The consequences of censorship have already been seen in the previous section, mainly threatening free speech. By removing certain profiles or content that does not align with their interests, these platforms exert immense control over the narrative presented to users.

A more subtle form of censorship is shadow banning. Through this practice, platforms selectively reduce the visibility of certain profiles or posts without removing them directly. Instead, they are selectively hidden making it more difficult to find and discover them

Another manipulation technique is highlighting, which is the opposite of shadow banning. Highlighting or promoting refers to actively boosting a user's content to a wider audience, providing them with additional visibility. Platforms can promote certain types of content or viewpoints effectively shaping public opinion and influencing the way people think about certain issues. This can be particularly problematic when platforms have a strong ideological bias or are beholden to particular interest groups.

Finally, the last manipulation technique is disinformation and propaganda. As we have seen in the previous chapter, this technique appears on social networks where free speech is a fundamental priority since any idea can be spread. It is the only technique that is not directly used by the platform itself. State-sponsored actors or interest groups can exploit these platforms to disseminate false information and shape public opinion in ways that serve their interests. This can have severe consequences for democratic societies, as it can erode trust in institutions if it is the case.

In retrospect, addressing content manipulation and ideological imposition on centralized platforms is critical to preserving the principles of free speech and neutrality. As we move forward in this paper, we will explore how these issues can be addressed during the design of a decentralized social media.

2.3. The Network Effect and its influence on monopoly formation and low innovation

Understanding the network effect and its influence on innovation and monopoly formation is crucial when discussing centralized platforms. The Network effect, also known as Metcalfe's Law (2.1), states that the value of a network is proportional to the number of unique connections (N) within the network, or what is the same, equal to the number of users squared (n^2) if users tend to infinite. This means the more users a network has, the more valuable it becomes for its users. However, while this principle seems advantageous for the growth of a network and its users, it can be a source of problems, often leading to stagnation of innovation and the emergence of monopolies.

$$\lim_{n\to\infty} N = \lim_{n\to\infty} \frac{n(n-1)}{2} \cong n^2 \qquad (2.1)$$

The concept of vendor lock-in illustrates this problem. Users can become dependent on a particular platform due to their connections and data stored on it. As a result, users find it difficult to switch platforms without losing their connections, creating a form of dependency.

Established companies with large user bases create high entry barriers for new companies attempting to enter the market. Attracting users to new platforms becomes an arduous task due to the millions of connections and the dominance of existing platforms. This enables established companies to maintain their market dominance and control, as companies with extensive user bases can leverage their dominance to replicate features from competitors.

This practice generates a cyclical effect of dominance and replication, where defeating the competence and maintaining the market power becomes an easy task. This behavior restricts innovation and obstructs the development of novel products, leading to limited diversity in the social media market, restricting users to a few dominant platforms.

Moreover, the network effect can exacerbate content manipulation and censorship by platform owners. Given the limited alternatives, it allows platform owners to maintain certain control over content moderation without being penalized by losing users. The Network Effect also affects data governance, a concept introduced in the next section.

In conclusion, while network effect can provide substantial benefits to communication networks and their users, it can simultaneously pose significant challenges to new market entrants leading to monopolistic practices by stablished companies and lack of innovation. Addressing these challenges requires the implementation of regulatory measures and fostering an environment that encourages innovation and competition, as seen in many other industries such as the telecommunications sector. This project proposes a decentralized standard as a base layer for social media platforms to build upon. This approach aims to facilitate the development of next-generation social media eliminating the Network Effect without relying on external regulations. However, this will be discussed in subsequent sections. For now, our focus continues to be on the remaining problems.

2.4. Data governance and privacy concerns

In the digital realm, where user data is a valuable asset, centralized platforms use their power to collect, analyze, and monetize user data, leaving users with limited control over their information. This issue is known as data governance. Data governance is a significant problem, as users typically cannot decide how their data is managed or used. This lack of control can lead to unauthorized access or misuse of user information, which could have serious consequences for individuals.

The lack of data governance is in part a network effect problem. The network effect creates data portability issues as it is too expensive for users to change platforms without losing their connections, challenging users who may become dependent on a particular platform. Provider lock-in coupled with lack of alternatives could lead to bad practices in the use of user data by social media, since established platforms cannot be punished by their users, as the threat of losing users to competitors is low.

Privacy, which is inherently linked to data governance, is another important issue. Centralized platforms often do not provide adequate privacy measures, exposing users and their data. The free-to-use model of these platforms typically involves an exchange where the company's revenues derive from user data. However, it should be considered that some users may be willing to pay instead of giving away their data. This can be further compounded by the lack of data governance, as users typically cannot decide how their data is managed or used.

Encryption is another security related aspect. Centralized platforms may not offer end-to-end encryption which is vital to maintain private communications. This is exemplified by cases such as the Cambridge Analytica scandal, which highlighted the importance of data privacy and end-to-end (E2E) encryption where millions of conversations where filtered.

In summary, users should have complete sovereignty and control over their data, as social media should be the product and not the users themselves. Therefore, social networks must be user-centric and serve the users ultimately. These considerations will be one of the fundamental parts of the standard developed in the following chapters, the details of which will be discussed later.

2.5. Impersonation, deep fakes and the need for a self-sovereign identity

As the digital world expands, impersonation and identity theft loom large, amplifying the urgent necessity for a digital and self-controlled identity. This concept is called Self-Sovereign Identity (SSI). Self-Sovereign Identity is an innovative approach in which individuals have absolute control and ownership of their identity, personal information and interactions, thus eliminating dependence on third parties.

In a world where artificial intelligence (AI) continues to improve, *deepfakes* will continue to grow. Deepfakes are AI-generated synthetic media such as videos, images or audio that convincingly imitate the appearance and voice of a person. The **Figure 5** shows a deepfake featuring Morgan Freeman. In this example, Bob de Jong, the person in the right of the picture, creates a speech that is then transformed by AI into an accurate replica of Morgan Freeman's speech, complete with matching voice and facial expressions.

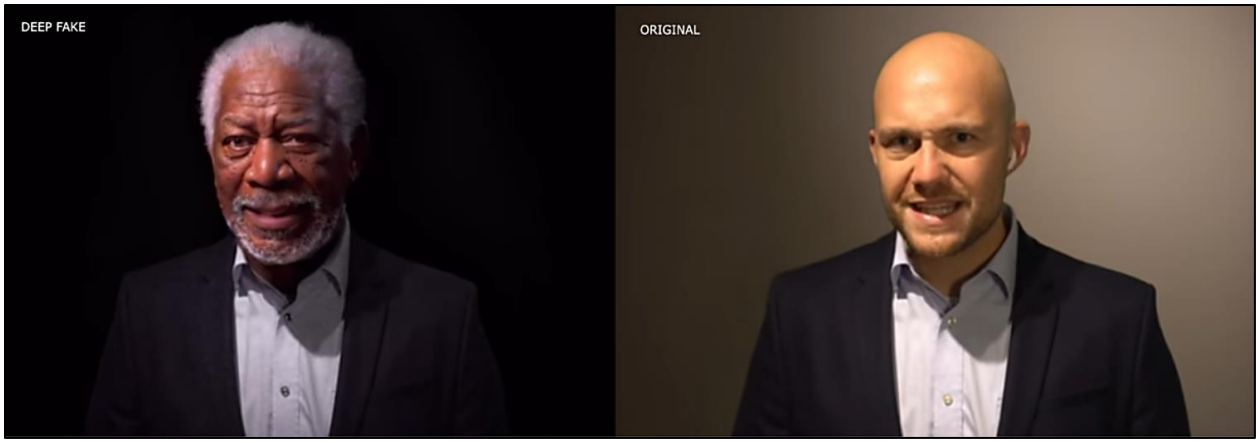

Figure 5 - Morgan Freeman Deepfake [2]

With the escalating boom of deep fakes, a verifiable source of truth becomes essential. It is in this context where digital and self-sovereign identity plays its role. Without SSI, anyone could control a user's identity and manipulate their content. This is why SSI is needed. SSI puts the user in control of their identity, information and interactions. All the content and interactions are digitally signed by its owner and all non-proprietary content should be linked back to the original. This is the only way we can know such content is real.

The need for Self-Sovereign Identity (SSI) combined with Zero-Knowledge Proofs (ZKP) is fundamental to establishing trust. Zero-knowledge proofs refer to cryptographic protocols in which one party can prove to another that it knows a given value, without revealing any other unnecessary information. Essentially, it is a way to verify authenticity without revealing private information. Implementing SSI and zero-knowledge proofs will serve to further strengthen the verification process. This combination promises a new horizon for digital identity that is considerably more secure, resistant, and most importantly, respectful of individual privacy.

In conclusion, the rise in identity theft, impersonation, and deepfakes underscores the urgent need for a more secure digital identity. Self-Sovereign Identity, backed by Zero-Knowledge Proofs, stands out as a powerful solution that gives users complete ownership and control of their personal data and interactions. The transformation of social media platforms to integrate SSI is a necessary step to mitigate privacy risks and ensure content authenticity. Therefore, the proposed solution will incorporate SSI, as the ultimate goal is to achieve a decentralized and user-centric identity that respects individual privacy while enforcing strong authentication and verification.

2.6. Massive bot waves and the spread of fake content

In recent years with the rise of AI powered technologies we have witnessed an increase in the fake content generation on social media. This surge has been largely driven by massive bot waves, which has led to debates about possible solutions, such as the content censorship. However, as discussed in prior sections, censorship may not be the most suitable solution as free speech could be threatened on some occasions specially if the platform is biased. Furthermore, stopping waves of censored bots is an uphill battle, given the incredible realism of its content and the number of accounts involved.

In light of these considerations, it becomes clear that our focus should shift towards development of more advanced anti-bot systems. Traditional defenses, such as CAPTCHA systems and many others, are no longer effective as modern AI can bypass such obstacles with ease. The cost of launching large-scale bot operations remains alarmingly low and deterring them through financial means appears to be the only resilient solution.

In this landscape, the only resistant system that remains effective against bots seems to be paid verification which increases bot cost exponentially. Although this might be viewed as a barrier to platform adoption, we will explore in the following sections the beneficial aspects of this strategy, which extends beyond bot mitigation.

Additionally, digital identity provides an additional security layer to safeguard against impersonation and dissemination of false information against someone. This stems from the fact that all content associated with an account is digitally signed by the account owner. Consequently, the people who use the content of other accounts must link that content to the original, facilitating the verification of its authenticity. So, we should be reluctant to believe unsigned and unlinked content. Alongside this measure, we will later discuss the potential development of sophisticated AI anti-bot tools that could further improve our defenses.

Unfortunately, companies have not found an effective solution to this problem yet. This uncertainty has led to a lack of investment by social media companies since they do not see an apparent solution and source of income. The companies view this as a waste of resources. Therefore, the best strategy right now is to increase bot deployment cost in some way to stop bot hordes. Twitter has attempted this strategy through paid verification. However, many people are reluctant to pay for what they perceive as an ineffective and useless verification service. In subsequent chapters, we will demystify the ineffectiveness of this solution.

In conclusion, the rise of AI technologies and the proliferation of massive bot waves present significant challenges in maintaining security on social media. Introducing paid verification, advanced anti-bot systems, and the use of digital identity tools can be effective strategies to counter the spread of fake content in the era of AI-driven social media.

Throughout this chapter, we have seen the main problems of centralized platforms, setting the stage for the development of a decentralized social network that effectively addresses these problems. For this reason, in the next chapter we will define the functional and non-functional requirements that the designed social network must satisfy.

3. Requirements

Based on the previous analysis, we can extract the functional and non-functional requirements of the decentralized social network that we will design in **Chapter 5**.

3.1. Functional requirements

Regarding the functional requirements, the main objective of these will be to solve the problems of the centralized platforms extracted from the previous analysis. The functional requirements are shown in detail in **Table 2**.

Code	Name	Description
FR1	Public Publications	Users must be able to upload content publications to their profile.
FR2	Content security	Content security: Since the network must be censorship resistant (NFR1), harmful content can easily spread, so users need security tools to filter the content. These tools should be configured by the user, respecting the individual freedom and decision-making of the users
FR3	Content manipulation resistant	Content manipulation resistant: The social media should be resistant to content manipulation. For this reason, the recommendation engines and search engines must be plug-and-play and easily replaceable by others
FR4	Open network	Open network: The social network must be a multi-company network where any infrastructure provider could join the network. A standard protocol must be designed so that the providers are compatible with each other. Users can decide where their data is stored, thus selecting their providers and changing them whenever they want. This aims to defeat the network effect and its negative results on monopoly formation and innovation.
FR5	Data governance	Users must be able to manage their data the way they want with full sovereignty. Users must be able to select their providers through signed agreements. The signed agreement has certain conditions chosen by the user, which the provider must firsts accept if it wants to become their provider. Users must be able to change their providers whenever they want if they do not honor the signed deal.
FR6	Bad actors penalization	There must be penalties to providers who do not honor their users' deals.
FR7	Private Publications	The network shall provide a means for its users to upload private content to the network. Users can decide whether their content is public or private, and even who can see their content.
FR8	Private communication channels	End-to-end private communication channels: There must be private communication channels between users.
FR9	Content and impersonation verification	Verification and anti-impersonation systems: Users cannot trust any actor within the network since it is a trustless network. For this reason, a digital identity system must be implemented so that any interaction within the network is signed by its author. This is achieved by checking all network interactions and content before displaying them on the screen
FR10	Digital identity	As commented in (FR9), the network must implement a digital identity system where every user interaction within the network must be digitally signed by the user. The users control their identity, interactions, and publications
FR11	Global username	Users can reserve a global and unique username within the network. This username can only be used by the user himself
FR12	Content redundancy	The users should be able to store their information on multiple servers of different infrastructure companies
FR13	Bot resistant network	The network should be resistant to massive bot attacks

Table 2 - Functional Requirements

3.2. Non-functional requirements

In the case of the non-functional requirements, the main objective of these will be to guarantee maximum performance and quality of the decentralized platform. The non-functional requirements are illustrated in **Table 3**.

Code	Name	Description
NFR1	Censorship resistant	The social media must be censorship resistant.
NFR2	Storage security	The local information like private keys and private data must be protected against attacks and must not be disclosed or accessible from outside the client application.
NFR3	Portability and compatibility	The social network must be compatible with any browser and OS.
NFR4	Scalability	The social network must scale as the number of users grows. The performance must not be degraded.

Table 3 - Non-Functional Requirements

Throughout this chapter, we have seen the functional and non-functional requirements that any decentralized social network must satisfy. These requirements guarantee that any social network that implements them will end with the problems of centralized platforms shown during the analysis of **Chapter 2**.

For this reason, in **Chapter 5**, we will design a decentralized social network with these requirements in mind. Then, in **Chapter 6** we will implement and develop a prototype social media. Finally, in **Chapter 7**, we will test the social network prototype in a real environment with different network actors.

But first to navigate towards this goal, it is essential to comprehend the main decentralized technologies that will compose the social network. That is what **Chapter 4**, ''Overview of the main decentralized technologies'' is all about. The next chapter will equip us with the technical knowledge necessary to turn this theoretical concept into a tangible reality.

4. Overview of the main decentralized technologies

This chapter provides an overview of the main decentralized technologies used during the design of the standard of **Chapter 5**. These technologies will be essential to understand and design the standard.

4.1. Hash functions

Hash functions are functions that map input data of arbitrary size to a fixed-size output value. This is illustrated in **Figure 6**. The values returned by *hash functions* are called *hash outputs* or *hash values*. *Hash functions* are really important and widely used in computer science because of their properties.

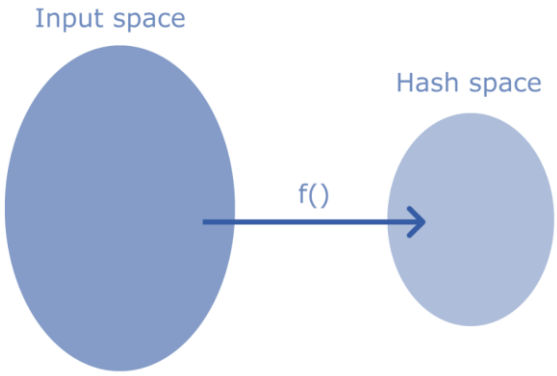

Figure 6 - Hash functions

To understand the importance of hash functions and their use cases we must first know the most important properties of *hash functions*:

- Determinism: Given the same input, a *hash function* always returns the same output.
- Fixed size: Regardless of the size of the input data, the output of the *hash function* is always fixed.
- Pre-image Resistance: Given a *hash value* h, it should be computationally unfeasible to find any input message m such that h = hash(m).
- Second pre-image resistance: Given an input m1, it should be difficult to find a different input m2 such that hash(m1) = hash(m2).
- High Speed: hash functions should be able to process data as fast as possible.
- Uniformity and collision resistance: A good hash function distributes its *hash outputs* uniformly across the entire output space, which reduces the likelihood of collisions.
- Avalanche Effect: A tiny change in input should produce drastic changes in output in order to have uncorrelated *hash outputs*.

These properties make *hash functions* really useful in a wide variety of applications, from data storage and data retrieval systems to complex cryptographic protocols. Throughout this chapter we will look at different technologies and use cases of *hash functions*. For the time being, it is important to know that it is desirable for *hash functions* to have these properties.

Note that not all hash functions are suitable for all purposes. There are hash functions that are excellent for storage applications but may not be secure enough for cryptographic use, and vice versa.

4.2. Hash tables

Hash tables, also known as hash maps, are one of the main applications of *hash functions*. *Hash tables* are powerful and efficient data structures that use *hash functions* to provide efficient insertions and lookups. These features make hash tables widely used in various applications such as databases and caching.

Hash tables are key-based storage, enabled by *hash functions* that generate a unique identifier for each entry. This is illustrated in **Figure 7**, where an alphanumeric key is passed through a *hash function*. The output of the *hash function* is the key of the data storage. Finally, the value is stored in the data storage within this key.

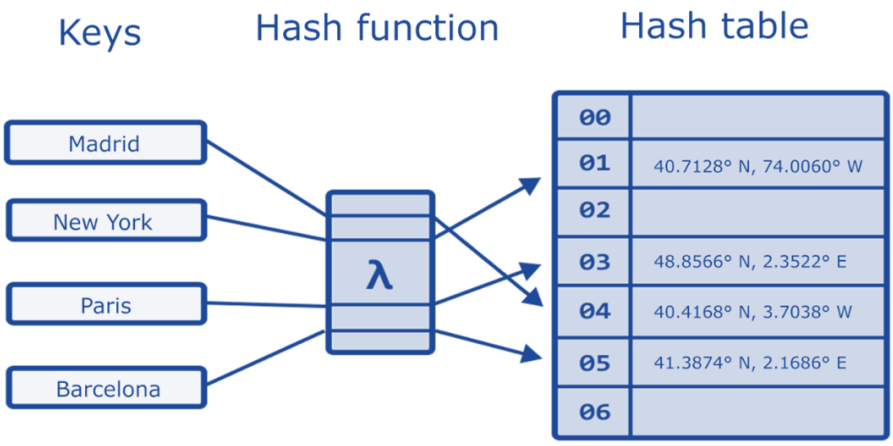

Figure 7 - Hash tables

Thanks to the Uniformity property of *hash functions* the keys are distributed equally across all the data structure making *hash tables* a really efficient data structure for load balancing. These facilitate sharding and replication techniques.

Despite having uniformity property, collisions can occur. Collisions lower down the efficiency of *hash tables*, but there are mechanisms to manage them. *Hash tables* also need to use fast *hash functions*, to retrieve and store the data as fast as possible.

In conclusion, *hash tables* are an essential tool in computer science and many other fields, offering fast data storage and retrieval. With an understanding of how they work, their strengths and limitations, we can use them effectively later.

4.3. Asymmetric cryptography

Asymmetric cryptography, often referred to as public key cryptography, is a unique cryptographic method that uses key pairs to securely encrypt or sign data. This model of cryptography is used for both data signing and encryption processes.

The most important feature of asymmetric cryptographic algorithms is the key pair generation, where each key pair has a private key and a public key. As the terminology suggests, the public key is shared and is used to verify signatures or encrypt. On the other hand, the private key is designed to remain confidential and can be used to sign or decrypt. As the name asymmetric implies, the public key can be derived from the private key, but the reverse is virtually impossible. This is the most important property of this encryption method which is illustrated in **Figure 8**.

Figure 8 - Asymmetric cryptography key pair

The most important asymmetric algorithms in these days use a variety of mathematical and computational techniques to achieve the key asymmetry property. These include Prime Numbers (RSA), Elliptic Curve Cryptography (ECC) or even *Hash Functions*. Although we will not go into the details of how they achieve this asymmetry, it is important to know this property.

In the next subchapters we are going to see two applications and use cases of asymmetric cryptography which are asymmetric encryption and asymmetric digital signatures.

4.3.1. Asymmetric encryption

The first use case of asymmetric cryptography is asymmetric encryption. As it is described in **Figure 9** the public key is used for encryption whereas the private key for decryption. When this method is applied to encrypted communications, the public key is shared, and the private key is kept secret to decrypt the data. With this approach, anyone can send us an encrypted message that only us we can decrypt.

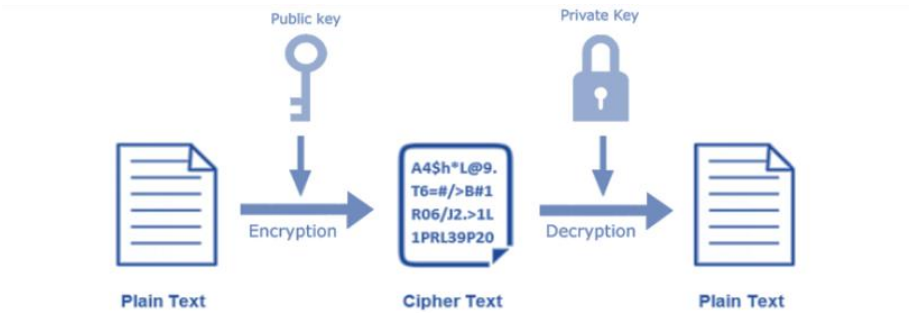

Figure 9 - Asymmetric encryption

4.3.2. Asymmetric digital signatures

The second use case of asymmetric cryptography is asymmetric digital signatures. Digital signatures guarantee the authenticity and ownership of the messages. Thanks to digital signatures we can be 100% sure that the person sending the data is really the expected sender, reducing the risk of impersonation. Asymmetric digital signatures are explained in **Figure 10**.

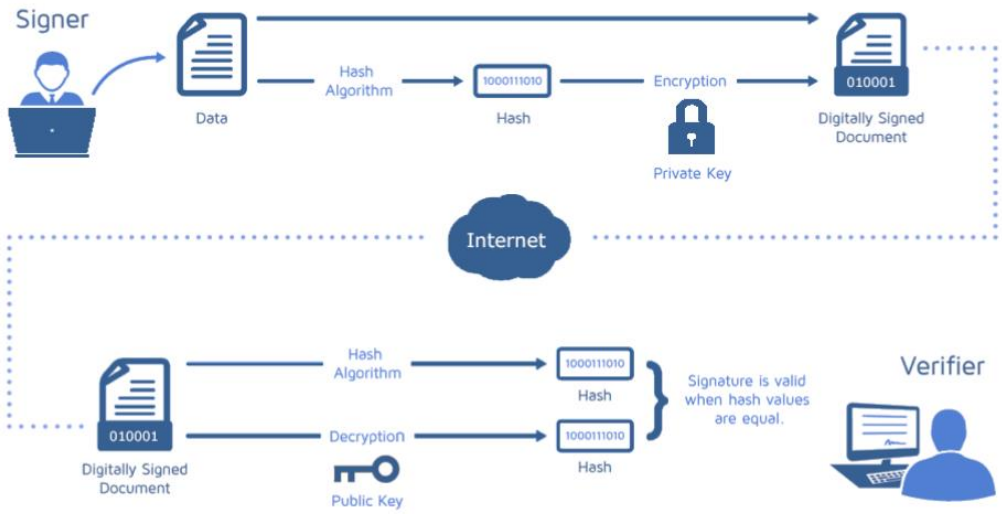

Figure 10 - Asymmetric signing

In first place, the signer passes the data through a *hash function*. Then, the *output hash* is encrypted with the private key and the final result is the signature. Finally, the data is sent along the signature.

On the opposite side of the communication, the verifier receives the data and the signature. The verifier passes through the *hash function* the data as the signer did before and obtains the *hash output*. The verifier then decrypts the signature and obtains another *hash code*. If the signature is correct, the two *output hashes* must be equal, otherwise the signature would not be valid.

As you may have seen, asymmetric cryptography is essential for maintaining private and secure communications through data encryption and signatures. These cryptographic methods will be used throughout the project recurrently, so it is important to understand them before moving forward.

4.4. Blockchain technologies

This section delves into blockchain technologies, examining its foundations and the different technologies that make it up that will be of vital importance for the construction of the standard.

4.4.1. Blockchain data structures

This section delves into the different data structures used by blockchain networks. Data structures play a very important role in achieving data integrity in blockchains.

The first and most basic data structures that we will explore are Radix tries. **Figure 11** illustrates how Radix trees are structured. In a typical trie, each node represents a single character, and edges connect parent nodes to child nodes. A radix trie is a data structure where each node represents a common substring. This means that each path from the root of the trie to a leaf node represents a single word.

However, this approach can be wasteful, especially if there are many strings with common prefixes. This data structure is designed to reduce storage by collapsing chains of nodes with only one child into a single node, effectively condensing the trie. However, this improvement has not been represented in **Figure 11**.

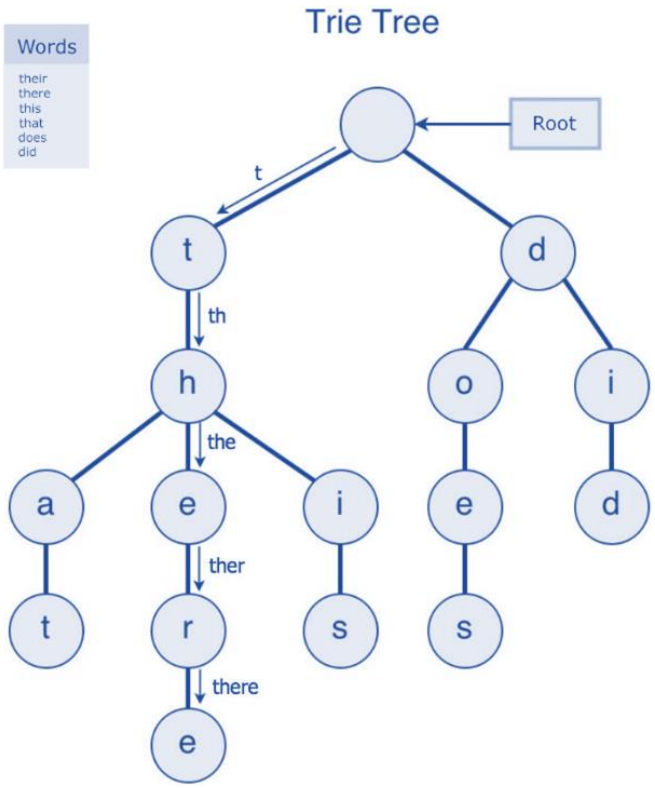

Figure 11 - Radix tree

A unique path from the root to each node corresponds to a specific key (string), allowing for efficient prefix-based search operations. The structure supports operations like insertion, deletion, and search, which are performed by navigating down through the tree.

The next blockchain data structures are Merkle trees, named after the computer scientist Ralph Merkle. This data structure is represented in **Figure 12**. Merkle trees are binary trees composed of a set of nodes, in which each leaf node in the tree is a *hash output* of a block of data, and each non-leaf node is a cryptographic hash of its child nodes.

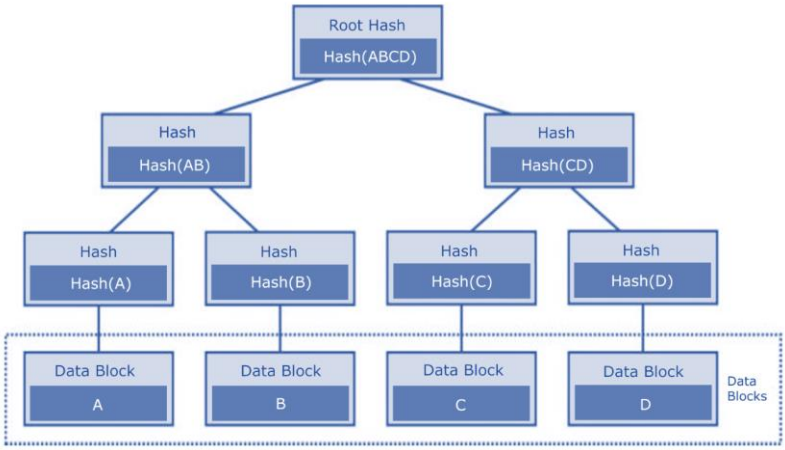

Figure 12 - Merkle tree

One of the most powerful aspects of Merkle trees is that you can confirm whether a specific data block is included in a set of data represented by a given Merkle Root with just a small amount of data – the hash of the data block in question, and the hashes of the parent nodes (called the Merkle Path or Merkle Proof).

Merkle trees are widely used in blockchains, and it will be used to present storage proofs in the standard as we will see later, so Merkle Trees will be of vital importance.

The last blockchain data structure presented in this chapter is Modified Merkle Patricia Trie (MPT). The Modified Merkle Patricia Trie is known for being the main data structure behind Ethereum blockchain as we will see later in this chapter.

The Modified Merkle Patricia Trie is a data structure that combines Radix trees and Merkle trees operations. **Figure 13** shows how this data structure works. The basic operations of MPT consist of four key steps:

1. Radix Trie formation: First, all the keywords in the system form a Radix trie structure. The Radix trie can group together common prefixes of keys, which in turn, increases the efficiency of key retrieval and search operations.
2. Leaf Node Hashing: After the Radix trie structure is formed, the values of each node are hashed.
3. Extension Node Hashing: After hashing the leaf nodes, the MPT applies hashes to the extension nodes. The extension nodes are essentially intermediate nodes that do not hold any data, but only serve to represent the shared parts of the keywords of their child nodes. The hash of each extension node is derived from its child nodes, similar to how data is hashed in Merkle trees.
4. Root Hashing: Finally, the root hash is created. This hash is derived from the combination of all the child hashes of the root.

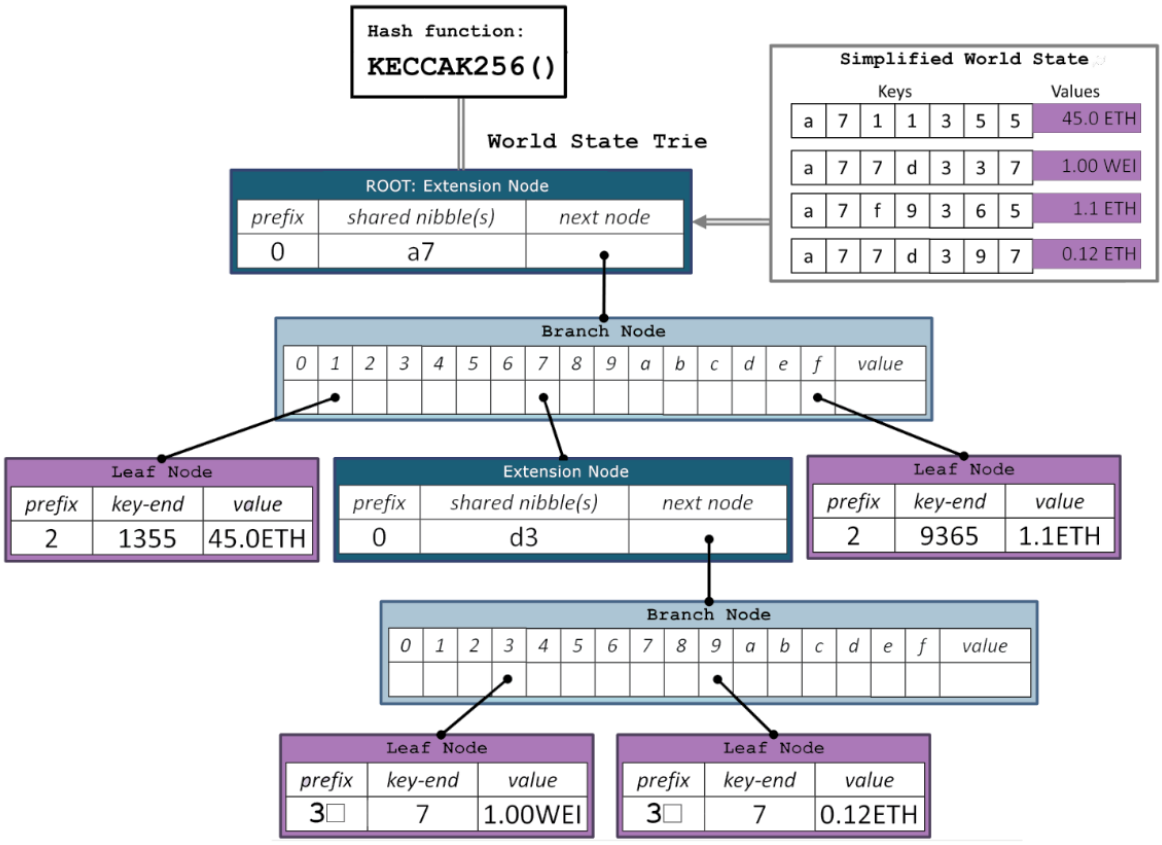

Figure 13 - Modified Merkle Patricia Trie [3]

4.4.2. Directed Acyclic Graphs (DAG)

The most important property of blockchain data structures is that they are acyclic data structures. This means that the data structures do not have loops, thus forming acyclic graphs. Directed Acyclic Graphs (DAGs) allow for very efficient searching algorithms within this data structures. We will not explore how these search algorithms work, but it is worth noting. A directed acyclic graph (DAG) is shown in **Figure 14**.

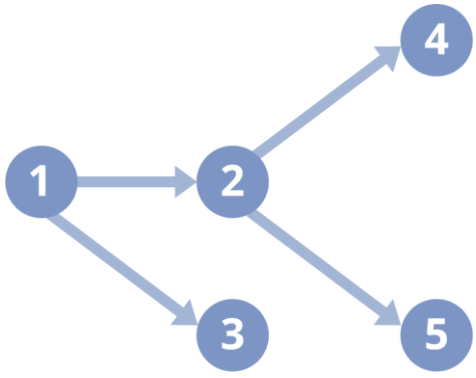

Figure 14 - Directed Acyclic Graph (DAG)

4.4.3. Peer-to-peer networks

Peer-to-peer (P2P) networks are decentralized networks in which participants or nodes interact directly without intermediaries. In the context of blockchain, *peer-to-peer* networks are crucial for enabling decentralization, consensus, security and trust. The communications within the network are done directly between nodes through peer-to-peer connections.

In **Figure 15** a *peer-to-peer* network of blockchain nodes is depicted. Blockchains are public ledgers or state machines that are redundantly stored by all nodes. Furthermore, operations can be performed by the nodes to change the state of the ledger. These operations are executed individually by each node and then broadcasted to the rest of the nodes. All this information is distributed throughout the entire network using complex P2P algorithms such as the Gossip Protocol to finally establish a consensus among the entire network on its state.

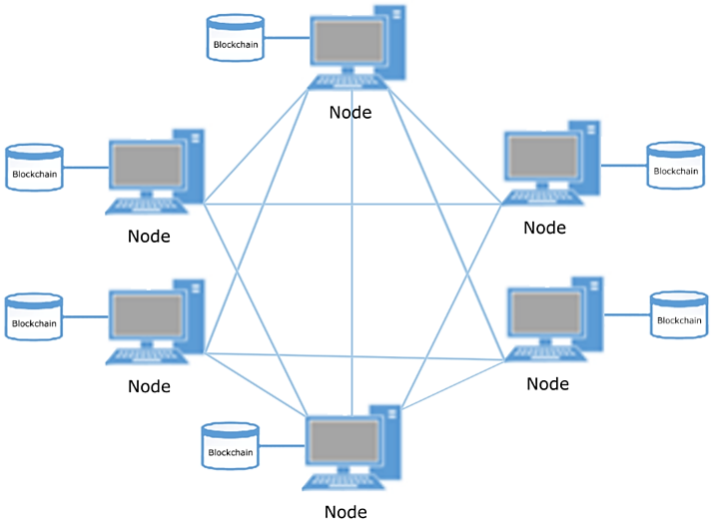

Figure 15 - Peer to peer network

4.4.4. Blockchain

The concept of blockchain technology was first introduced to the world in 2008 by an individual or group of individuals using the pseudonym Satoshi Nakamoto. The Bitcoin white paper [4] theoretically described decentralized network of monetary transactions now known as blockchain. The bitcoin network was launched by Satoshi Nakamoto in January 2009. Since then, blockchain technology has expanded beyond Bitcoin and is being used in a variety of applications.

As we have seen, blockchains are distributed databases or state machines that are stored and executed redundantly across multiple nodes. At a basic level, as shown in **Figure 16**, a blockchain starts with an initial state, S. This initial state is initialized by the first block of the chain, commonly referred to as the "genesis block." The genesis block is usually an empty block. Then, subsequent blocks are added to the blockchain containing a series of transactions each. When a transaction is executed and added to the blockchain, it changes the state of the blockchain from state S to a new state S'.

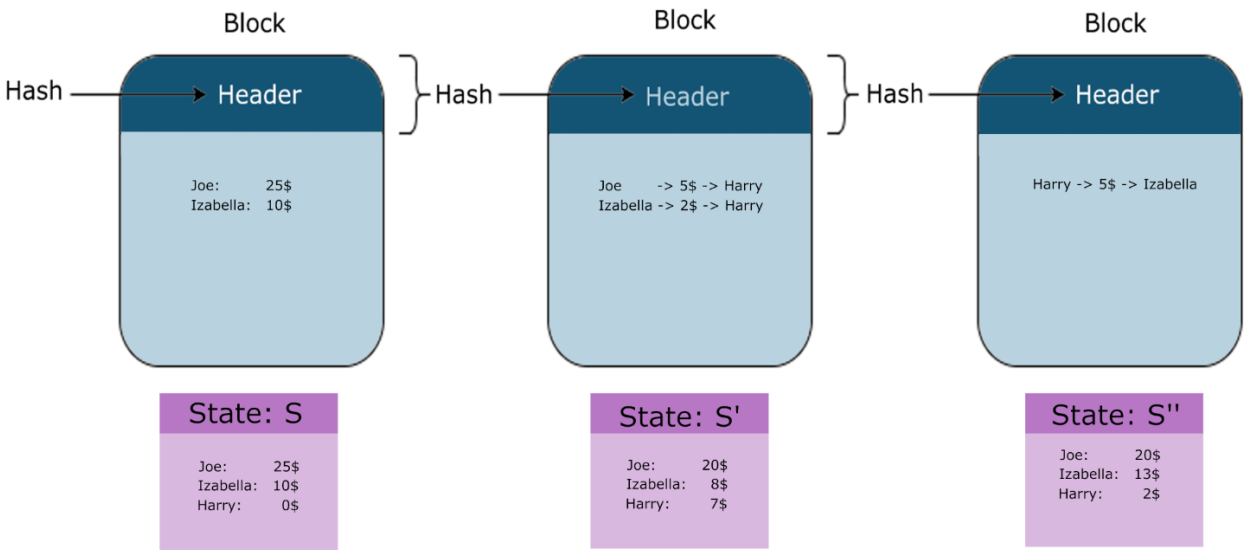

Figure 16 - Blockchain

A simpler way to understand how blockchains work is to think of them as state machines at first place. But blockchains are more than just a simple state machine. Blockchains are distributed and censorship resistant state machines. The question then arises: how do blockchains get those properties from?

The process starts with the creation of the genesis block. First, the genesis block is created with the information needed to initialize the initial state of the blockchain. Once, the blockchain is initialized transactions can occur within the network. These transactions are broadcasted to the rest of the nodes, and then aggregated and bundled into blocks.

New blocks are added periodically to the blockchain. The time frame between blocks is called block period. During the block period, the network nodes receive multiple transactions broadcasted by the rest of the nodes. Once the block period ends a new block is issued aggregating all the transaction received within that time frame.

The issued blocks contain the received transactions and some information related to that block. Additionally, the previous block is passed through a hash function which produces an output hash code that is called block hash. The block hash of the previous block is then added to the new emitted block. This process is repeated for each issued block, thus creating a chain of blocks linked by their output hashes.

This chain of hashes makes blockchains immutable, because changing any block also changes the subsequent blocks due to the properties of the hash functions. But now a new question arises: who decides which is the next block? There are several ways in which the nodes can select the next block, this is where consensus algorithms come to play as we will see next.

4.4.5. Consensus algorithms

Blockchain networks are systems based on mistrust, so they rely on a system of checks to ensure that the blockchain state transitions to a valid state. For this reason, nodes, or network participants play a critical role in this process by verifying the integrity of each new issued block.

To choose the next block, the nodes check first if the block is valid. What this means is that the block must contain only valid transactions, where the transitioned state is correct. Imagine that the current state of the blockchain is the state represented in **Figure 17**. Joe cannot make a transaction greater than $25 because it would lead the blockchain to an incorrect state, since Joe has a balance of only $25, so it would not be a valid transaction. Transactions are also assigned an incremental number associated to prevent the same transaction from being repeated multiple times in the blockchain. Also all the transactions are signed with the account private key of its owner, to prevent other users from spending your funds.

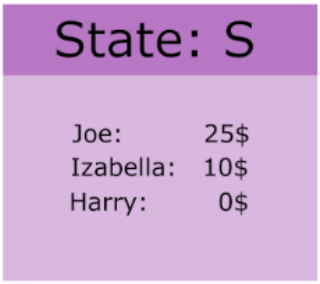

Figure 17 - Blockchain state

Once the nodes validate that the transactions within the block are correct a consensus should be made between al the network participants to decide which one is the next issued block. This is where consensus algorithms play their part.

There are multiple consensus algorithms to choose the next valid block in blockchain networks, but the most important ones are Proof of Work (PoW) and Proof of Stake (PoS).

In Proof of Work a complex mathematical puzzle needs to be solved in order to issue the next block. The block will be valid if a node solves the mathematical puzzle and attach its solution to the block. The mathematical puzzle is a game of trial and error where the nodes need to find a number that in combination with the block data gives a specific hash. This hash has to meet specific conditions. The difficulty of this puzzle is adjusted dynamically based on the computational power of the network in order to maintain a constant block period.

Therefore, the nodes will iterate over different numbers to guess the solution of the puzzle (nonce). The node who guesses the solution earns some money for validating and securing the network, and then the block is issued by the node. The nodes of PoW networks are called miners due to the great amount of computational effort they have to make to issue or mine the next block.

The rest of the nodes will accept the block because it contains valid transactions and the puzzle solution or nonce. The longest chain is the valid chain since it is the chain with most computational effort. This means it is the most secure chain where the majority of nodes add its blocks. This way, if a malicious node wants to hack the network it would need the 51% of computational power of the network to build the

longest chain, and make the other nodes accept its chain. This is extremely difficult since there is. Even If a node hacks the network for a period of time it could only do two things, not including transactions or lag some of them for a specific time. If the malicious actor includes not valid transactions in the blocks, it would not be accepted by the rest of the network. So even if the network is hacked a certain period of time the hacker could do little and probably will lose money doing so. However we are not going to see in detail this issue because it is not the scope of the project, so we will assume it is extremely difficult to hack the network.

The PoW algorithm is considered to not being an environmental responsible algorithm since it requires a big amount of energy for solving the mathematical puzzle, thus some people considering a waste of energy and resources.

On the other hand, in Proof of Stake the nodes are required to "stake" or commit a certain amount of their cryptocurrency holdings to participate in the process of validating and adding new blocks. The validator nodes block a certain amount of money to validate the block and behave honestly because if they do not, they could lose their stake. The nodes also receive money for validating the network and incentive this process.

PoS is an environmental respectful algorithm since it does not require the validator nodes to solve complex problems. In PoS the validator node that issues the next block is chosen randomly. The selected node validates the blockchain transaction and issues the new block while the other nodes validate the validity of the block.

Both consensus algorithms reach the Nash equilibrium. This means that most of the nodes and participants of the network reach an equilibrium in which they behave honestly to validate the network transactions due to the incentives and penalties of both protocols.

Both consensus algorithms are different with its advantages and disadvantages. However we are not going to go deeper in this discussion because it is not the scope of the project to discuss this issue. Both consensus algorithms reach their main objective which is to maintain a secure blockchain consensus within the network.

4.4.6. Self-Sovereign Digital identity

Digital identity and ownership are achieved within blockchains thanks to asymmetric cryptography. Accounts are created within blockchains with an asymmetric key-pair. The public key is used as an account network identifier whereas the private key is used to sign transactions. This way digital identity and ownership are ensured. The most important thing is that the user owns their own identity and decide what to do with it, thus achieving self-sovereignty identity.

4.4.7. Smart contracts

Smart contracts are computer programs deployed over the blockchain. **Figure 18** shows how these programs are deployed in blockchains.

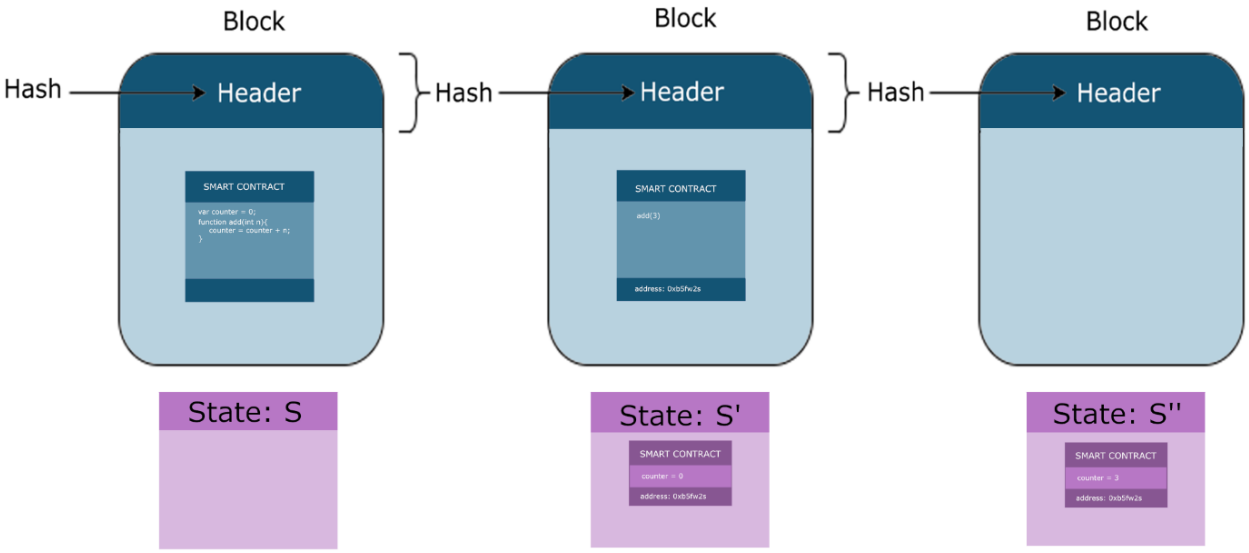

Figure 18 - Smart contracts

First, a contract deployment transaction is broadcasted to the network. This transaction includes the code of the program. The code is usually programmed with a high level language and then compiled into machine code. The deployed code is machine code that can be understanded by the blockchain virtual machine. The nodes run the blockchain virtual machine locally and execute the code. In the **Figure 19** is shown the smart contract code deployed on the blockchain of **Figure 18**.

Figure 19 - Smart contract code

The smart contract can have variables and its own state, that is saved within the blockchain state. As you can see in the smart contract of **Figure 19**, the *counter* variable is set to zero initially. So, once the contract is deployed this state variable will be zero too. In **Figure 20** the initial state of the contract is shown. As you can see in **Figure 20**, the smart contract is deployed to an address on the blockchain. This address is needed to interact with the contract, execute its functions and check its state.

Figure 20 - Smart contract state

Then, a contract method is executed by someone. For this reason, a transaction with the contract method execution is broadcasted over the network and added to the next block if it is valid. This transaction is shown in **Figure 21**.

Figure 21 - Smart contract function execution

Before adding the transaction to the blockchain, the nodes execute the code of the contract and if the state transition is valid they update their state and include the transaction in the blockchain block. The rest of the nodes will also execute the transaction locally to verify it does not lead to an invalid state transition. If the transaction and the block that includes it are valid the state of the smart contract is updated in each node. In **Figure 22** the new state of the smart contract is shown.

Figure 22 - Smart contract state update

The smart contract shown in this chapter is a very simple code, but more complex programs can be built. Smart contracts bring multitude of opportunities and applications to blockchains like Decentralized Applications (DApps), Decentralized Finance (DeFi), Decentralized Autonomous organizations (DAOs), and many more use cases. In conclusion smart contracts present numerous opportunities for innovation bringing decentralization, security and digital ownership to multitude of applications.

4.4.8. Ethereum network

The Ethereum network is one of the first blockchains in the crypto ecosystem and the first to bring smart contracts to the real world. Ethereum was conceived in 2013 by programmer Vitalik Buterin who wrote the Ethereum whitepaper [5]. Ethereum was founded in 2014 by Vitalik Buterin, Gavin Wood, Charles Hoskinson, Anthony Dilorio and Joseph Lubin. The technical design of Ethereum took place during that year and finished with the publication of the Ethereum yellow paper [6] by Gavin Wood, in which the technical details were specified.

The network went live on 30 July 2015. Ethereum was initially designed as a PoW network, but recently transitioned to PoS, thus being more environmentally friendly. This transition took place in the early 2023. This transition was called the merge and it is shown in **Figure 23**. The next step in Ethereum's roadmap is sharding which will skyrocket the scalability of the network, but it will make Ethereum less decentralized as consequence.

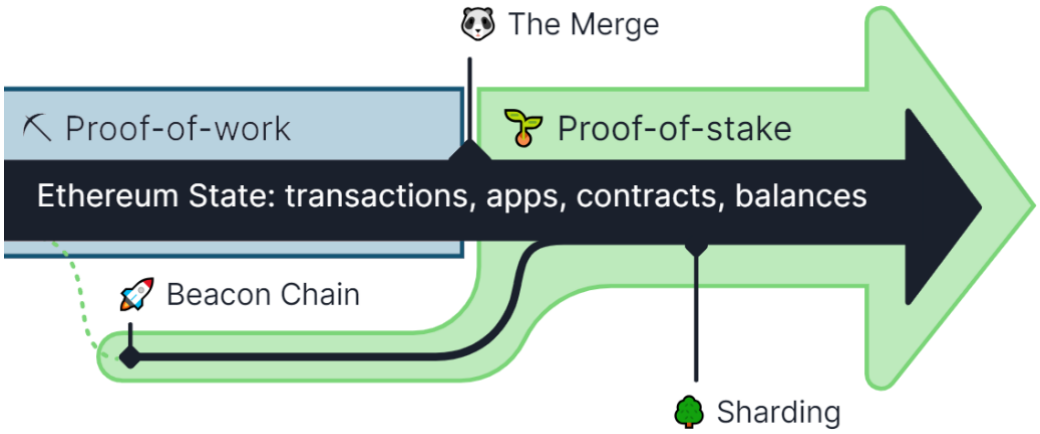

Figure 23 - The Merge: Ethereum transition to PoS and Sharding [7]

Ethereum is the second biggest blockchain network in the world behind Bitcoin. But Ethereum is the biggest smart contract blockchain, not only because of the number of blockchain nodes but also because of number of users and it has the largest developer community.

Some of the most important technical aspect of Ethereum are included in **Table 4**. The Ethereum currency is called Ether, which have divisible units called Weis. The current supply of Ethers is 120M. Thanks to the new Ethereum fee model that introduced in 2021 with EIP-1559 [8], Ethereum can be deflationary or inflationary depending on the block space demand. The maximum block size is 30M gas. If the block is filled above 15M gas, Ethereum is deflationary, otherwise inflationary.

Name	Ethereum
Symbol	ETH
Founded Date	2014
Founded By	Vitalik Buterin, Gavin Wood, Charles Hoskinson, Anthony Dilorio and Joseph Lubin.
Currency	$1\ \text{Ether} = 10^9\ \text{Gwei} = 10^{18}\ \text{Wei}$
Current supply	120M Ethers
Tokenomics	Deflationary and Inflationary (depends on the amount of ether burned which depends on the block space demand)
Transactions per second	~ 30 Tx/s
Block period	~ 12 s
Maximum Block size	30M gas = 1.875 MB
Consensus Algorithm	PoW → PoS
Minimum Stake	32 Ether per validator
Ethereum programming language	Solidity
Ethereum compiled code	EVM code (Ethereum Virtual Machine code)
Applications	DApps, DeFi, DAOs
Most important standards	ERC20, ERC721

Table 4 - Ethereum technical details

To become an Ethereum validator the nodes must download and run an Ethereum client. The most popular client software is *Go Ethereum* [9]. Then, the nodes have to stake a minimum of 32 ETH before becoming validators.

Ethereum enables a large variety of applications to be built on top of it. Its decentralization and its powerful smart contract functionality make Ethereum an ideal platform for exploring new use cases of blockchain technology. This is the reason why we will use the Ethereum network in the development chapter.

Throughout this chapter, we have seen the main decentralized technologies. This chapter has equipped us with the technical knowledge necessary to develop a decentralized social network that addresses the problems of centralized platforms presented in **Chapter 2**.

For this purpose, in **Chapter 5**, we will first design a standard using the technologies seen in this chapter and implementing the requirements of **Chapter 3**. Then, in **Chapter 6** we will develop a decentralized platform that implements the standard. Finally, during **Chapter 7** we will test the application in a real environment.

5. Design of a decentralized social network

In this chapter, a decentralized social network will be designed. The chapter presents the technical proposal to solve the existing problems of the centralized platforms. As discussed in the previous chapters, the main challenge facing existing platforms is centralization, as it is the root of all their problems. Therefore, the first step will be to achieve decentralization. Decentralization can be achieved by storing the information in multiple servers or what is called a distributed architecture. In **Figure 24** a distributed architecture is shown.

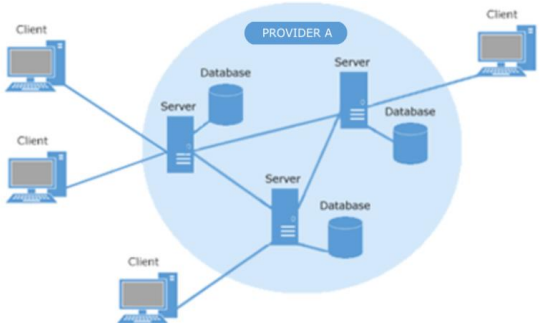

Figure 24 - Distributed architecture

As you remember from the second chapter the distributed architecture is already present in current platforms. The problem appears when all the servers are from the same provider. Distributed architectures are not enough to stop censorship since all the servers are from one single company, so the users are affected by the company decisions which can coordinate them for a specific task. This is illustrated in **Figure 24**, where a single provider is the owner of the whole computing infrastructure. Therefore, this company can coordinate all these servers with the sole purpose of censoring certain accounts or content.

In order to develop a censorship resistant network, we would need to store the data in multiple providers. The more providers an account has, the more likely it is that one provider will provide its information, thus defeating censorship. Therefore, it is extremely important to develop a multi-company platform, open for any infrastructure provider. So, the first step will be to redundantly store each account's data across multiple providers. This is what **Figure 25** illustrates, image in which some clients are connected to one or more server provider networks.

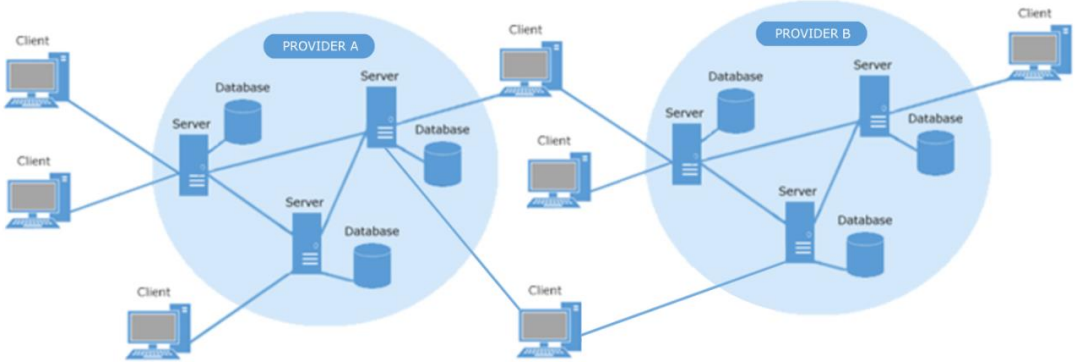

Figure 25 - Decentralized architecture

As it is shown in **Figure 25** the internal architecture of each provider can be distributed as well. Nevertheless, this is not the scope of this project, so henceforth we will assume that the internal architecture of each provider is only a single server, although it is not. This is just for simplicity. From now on we will draw each provider network as a single server computer as it is shown in **Figure 26**, where a client is connected to three different providers.

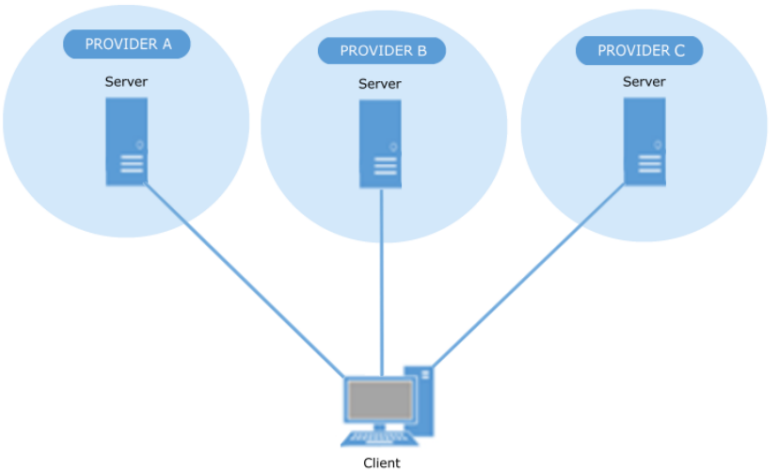

Figure 26 - Decentralized architecture II

Users want their information to be distributed across multiple servers from different providers to achieve fault tolerance, increased accessibility, lower latency, disaster recovery and be censorship resistance. After introducing multiple providers to the network, inconsistencies may appear between providers. However inconsistency is not a real problem, since as we will see later a mechanism will be implemented to solve it.

Furthermore, because it is a multi-provider network, users do not know where the other accounts are located. Therefore, users need a Domain Name System (DNS) to locate other users across the Internet. The proposed solution will use blockchain-based DNS, which is a great alternative to centralized DNS as we will see soon. The next section introduces and describes with more detail the general architecture based on these assumptions.

5.1. General architecture

This section introduces the general architecture. The network has a decentralized architecture like the one in the **Figure 27**, with blockchain nodes acting as DNS servers and different providers storing and serving user data. Finally, users with a client-side application installed are able to interact with the providers and the blockchain nodes thanks to a standardized design.

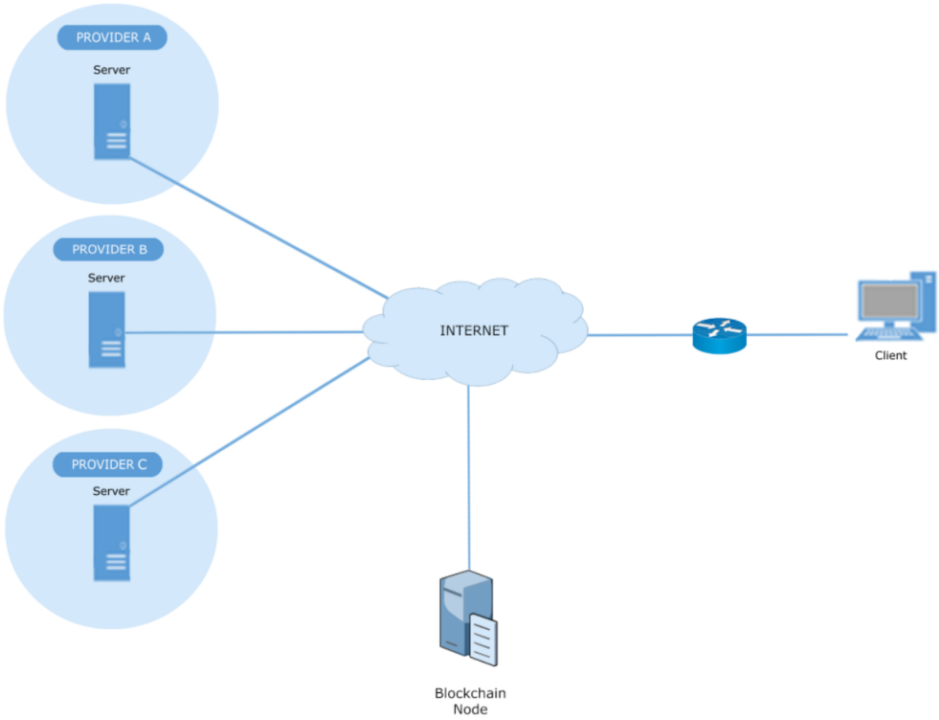

Figure 27 - General architecture of the decentralized social network

The diagrams in **Figure 28**, **Figure 29** and **Figure 30** show how the social network works. First, the infrastructure providers publish their deals on the blockchain ledger. These deals are the service conditions that the infrastructure companies are willing to provide to their users.

Once the deals are published in the blockchain users can search for them. They can explore the blockchain and search for specific deals or providers based on certain conditions. Users can choose the deal they want. For example, if they want more privacy, they will choose deals that respect their privacy, if they want a free deal, they may be willing to give their data in return, and so on. Once, the users know which deal and which provider want, the users send a deal request to the blockchain.

Then, the providers decide which deals they want to accept. After all, it is the providers who have the last word. Finally, after the deal is accepted, users can start uploading content to their providers, thus making their data available through them. The deal process is explained in detail in **Figure 28**.

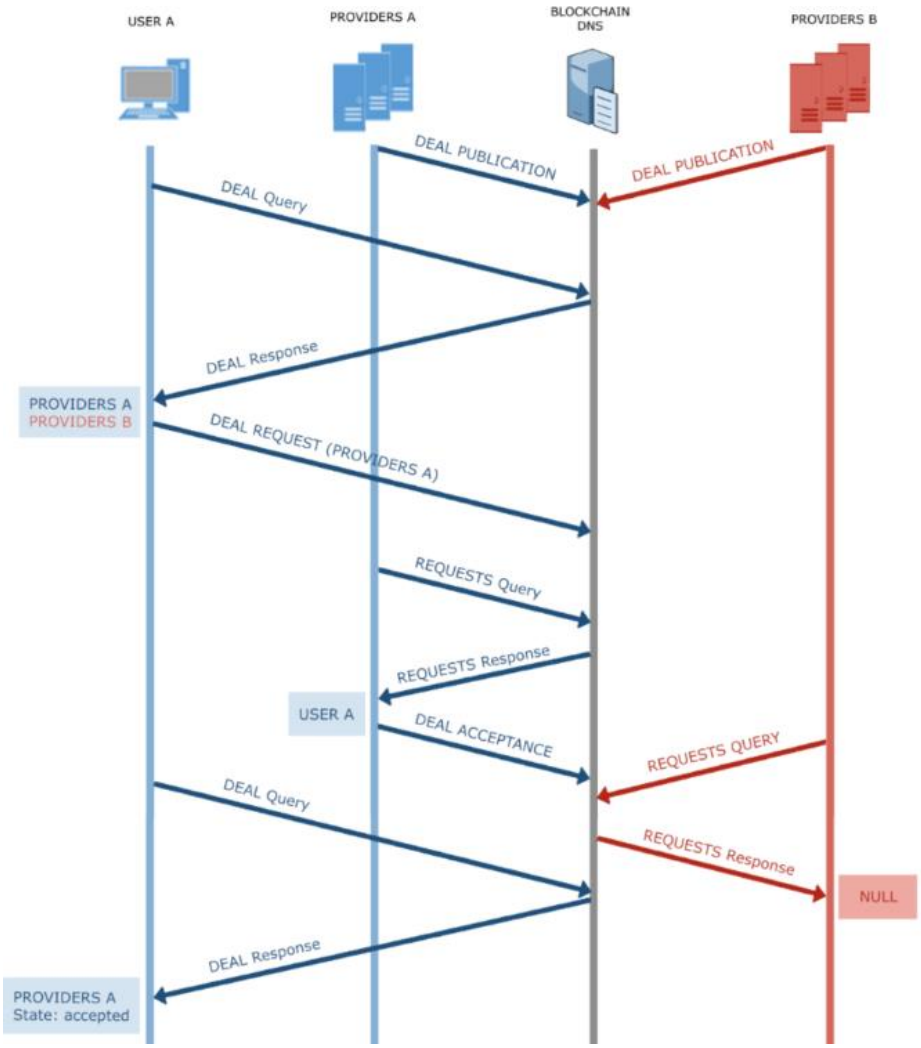

Figure 28 - Deal process

It is important to note that each provider publishes its own deal. Here, in **Figure 28**, we have aggregated the providers of ''*USER A*'' under the same name ''PROVIDERS A'', but in reality this process is done for each provider individually. It has been represented that way for simplification reasons.

Once, the deal process is finished and the account has at least one provider, other users can view their profile by requesting it to their providers. Even though the profile would be accessible from their providers, few users would be able to access it, because the DNS is not configured yet. Therefore, the account would not be searchable by username.

The username reservation process and the DNS are explained in **Figure 29**. First, the user sends a ''*username reservation*'' transaction to the blockchain DNS. Along this transaction, the user sends the username it wants to reserve and the IP addresses of their providers. If the username is not reserved, the username is successfully reserved for that user. Once, the DNS is set up, other users can now find and locate a user by searching for their username.

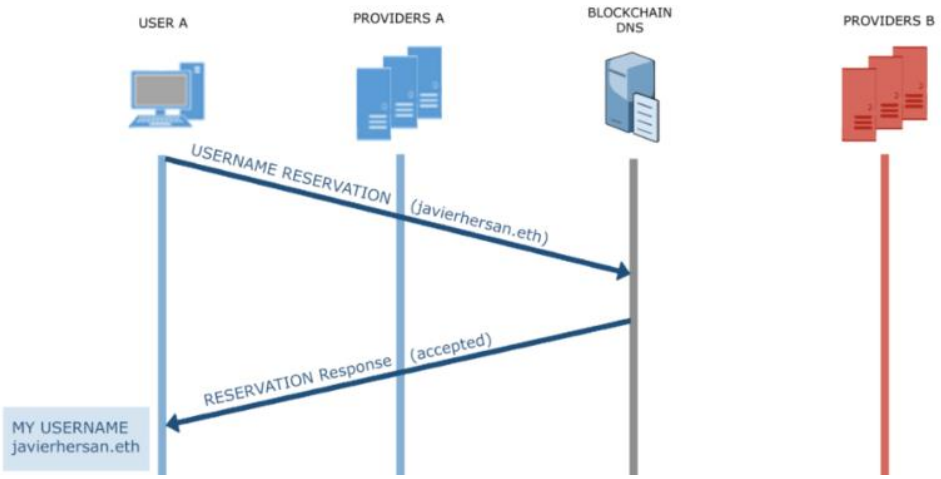

Figure 29 - DNS and username reservation process

Once, the DNS is configured, other users can request their profile and information easily. This process is explained in **Figure 30**. First, the users will have to request the DNS, asking for the providers of a specific account. Once, the users know the IP addresses of the providers for that account, they can send a profile request to them. There are different types of profile requests depending on the resource they want to consult or view.

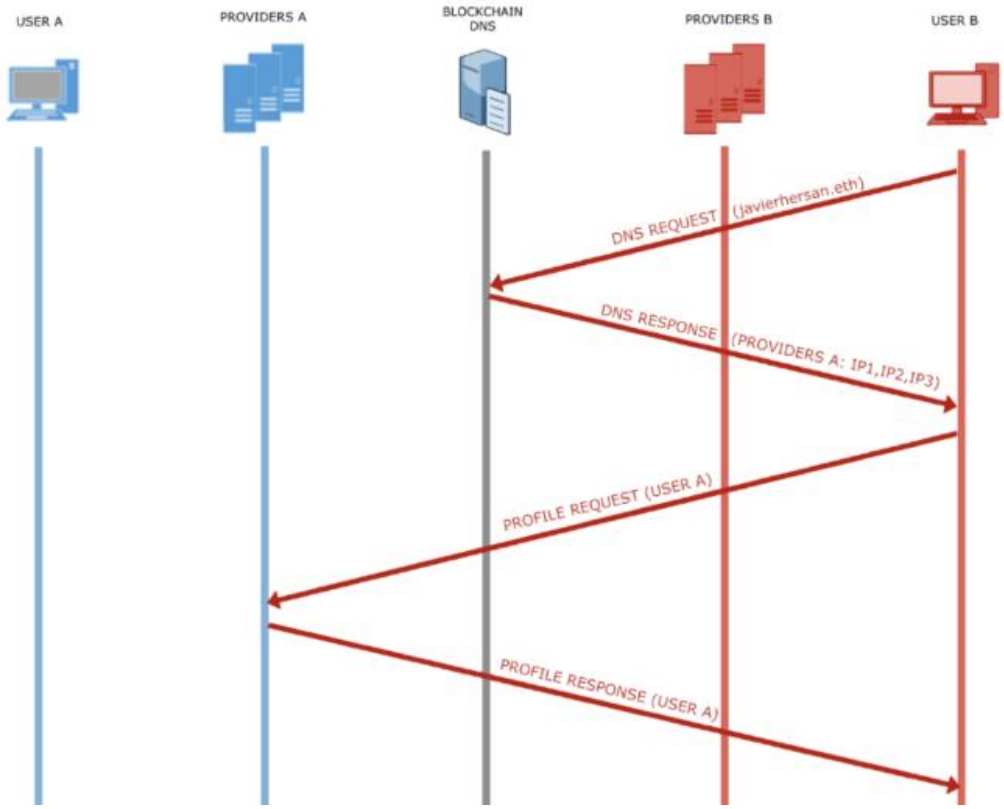

Figure 30 - Profile request

Once, the general architecture and the main workflows of the social network are clear we can analyze in detail each network actors and their internal components.

5.2. Server architecture

During this section we will see the design of the server. The providers will have to implement this design in order to be compatible with the client application used by users. The main architecture of the server is represented in **Figure 31**. While users can upload content to their profile, other users can also request content from other accounts. For this reason an API module is required to upload content and expose it.

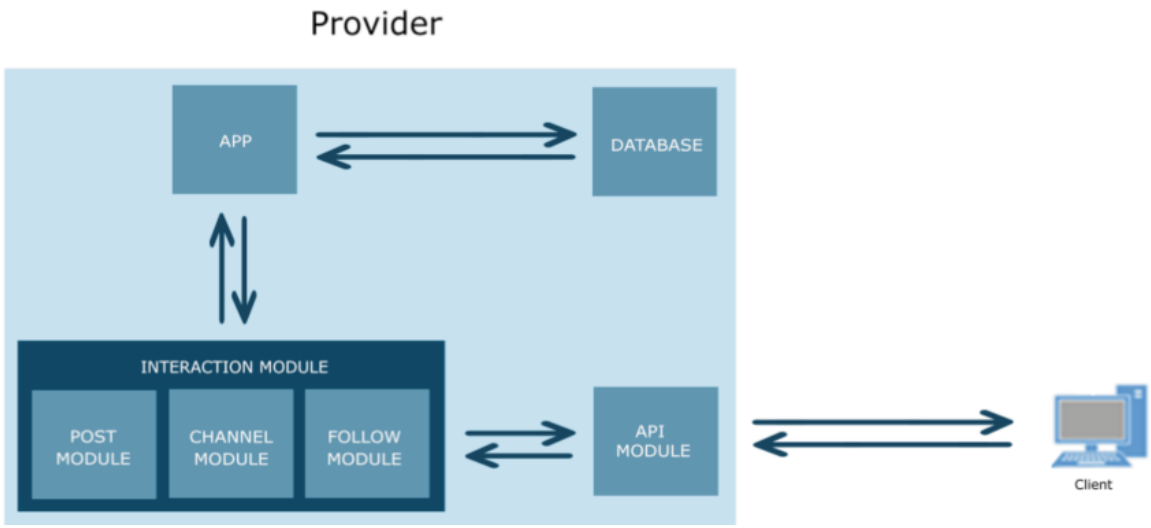

Figure 31 - Server architecture

Then, after the API requests are processed by the API module, the requests go into the interaction module. The interaction module ensures that only the owner of the account is the one who can upload the content or perform an interaction on their behalf. This is achieved thanks to content headers, these headers will have a signature attached to ensure content authenticity and prevent impersonation.

Since all the publications and interactions within the network are signed by its owner, users can detect identity impersonation. Even if a malicious provider stores data that has been uploaded by an account other than the owner, users can detect it on the client side. However, this module is needed in the server side to do not store unnecessary information despite the ability of users to detect fake content and impersonation. Content headers are explained later, but it will be one of the most important features that the interaction module implements.

The interaction module also ensures that always the last version of the data is uploaded and earliest versions of the content do not overwrite the latest content if they are sent again by error or by malicious actors. This feature is achieved thanks to content headers as well.

Furthermore, the interaction module has inside other submodules specialized for each different type of interaction. These submodules ensure the content has the correct format depending on the interaction type. There are different types of interaction within the social network. The first type of interactions are public and private publications. Users can upload content to their profile and can also request content from other profiles. Therefore, the post module is in charge of uploading and editing publications, and exposing them so that other users can see them. The second type of interaction is the channel module which is needed for managing private communication channels between users. Finally, the follow module for managing following and unfollowing interactions.

Once, the request is processed by the interaction module, the application can store the interaction and its content in the database. If it is the case that the request is a content request the app will manage the database to get the desired information and expose it through the API responding back.

There is also a blockchain module for publishing and accepting the deals, but we have omitted it in order to simplify the architecture, however, this module will look quite similar to the module included in the client-side application explained later.

5.3. Client architecture

In this section we will explore the architecture of the client-side application. This architecture is described in **Figure 32**. The user can interact with this application through a user interface (UX). These interactions can range from posting public and private publications, viewing content from other user profiles, sending private messages, or even following/unfollowing other accounts. These interactions are the same as those we have just seen in the server architecture section.

Then, the app module sends these interactions and the associated content to the interaction module. This module is in charge of signing the interactions and its content before sending it to the providers and publishing it. This module also ensures that other user interactions are signed before been shown in the user interface, actuating as a filter for fake and unsigned content. The signatures are needed since the network is a trustless network. The identity and content authenticity are guaranteed with the signatures and content headers. The content header is explained later, in subsequent sections.

As we have seen before the user will need an API module for requesting and uploading content. This module will be the responsible of interacting with the providers. The application also has a local storage module that will store private information like private data or keys.

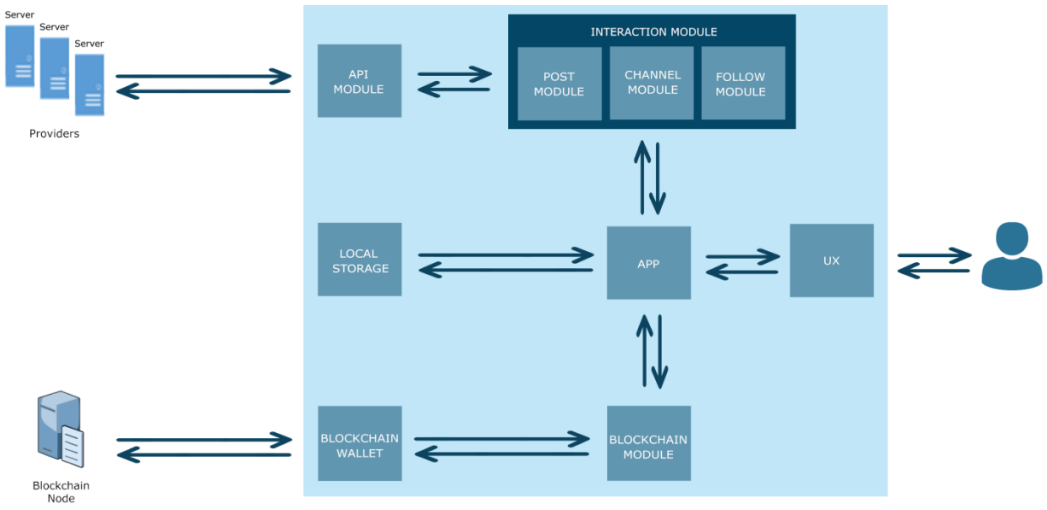

Figure 32 - Client architecture

Finally, the application will have a blockchain module for interacting with the blockchain. This module is needed for reserving usernames or make requests to the blockchain DNS. Moreover, this module is also needed to select the providers and carry out the deal process. The blockchain module interacts directly with a blockchain wallet which is the one in charge of sending the blockchain transactions to the blockchain nodes.

5.4. DNS architecture

After introducing several providers in the network, users do not know where each account is located. Therefore, in order to locate users across the Internet, users need a Domain Name System (DNS) to translate usernames into IP addresses. As we have mentioned before we will design a blockchain-based DNS, which is a great alternative to centralized DNS.

Due to the distributed nature of the social network and the absence of a central authority to govern it, a blockchain-based DNS becomes essential for username registration within a multi-provider network. Blockchains are decentralized public ledger that can be used as a name registration system. The blockchain DNS ensures decentralization and ownership of the domain names through smart contracts. For this reason a smart contract blockchain will be needed to deploy such functionality.

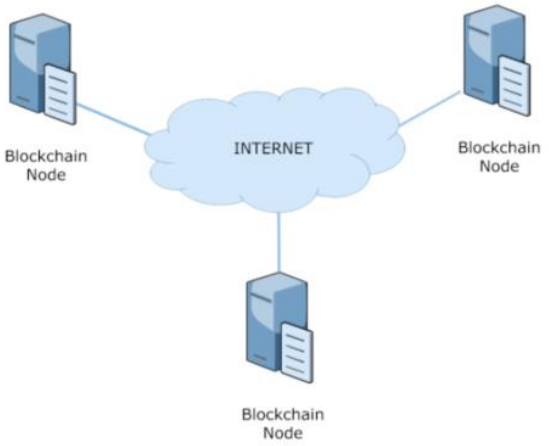

Figure 33 - Blockchain DNS

As we have seen in **Chapter 4**, the smart contracts are programs deployed on the blockchain. In consequence, these types of programs can be used to deploy different types of decentralized applications. In our case, the smart contract will handle the DNS logic. This smart contract will have more functionalities like the deals feature, but for now the focus is on the DNS.

The **Figure 34** shows a DNS request. First, a user requests a username to the blockchain node, in this case ''javierhersan''. Then, the node responds him with the owner of that username, which in this case is the blockchain address ''0xb37fq6gk5rxa''. Then, the user asks for the server providers of that account address. This double request is needed to store on chain as little memory as possible and save on storage costs, since storing information on blockchains is really expensive. Both requests and responses can be aggregated into a one single request using a smart contract method that aggregates both in order to reduce latency, but the figure exemplifies how, behind the scenes, there is actually a double memory request.

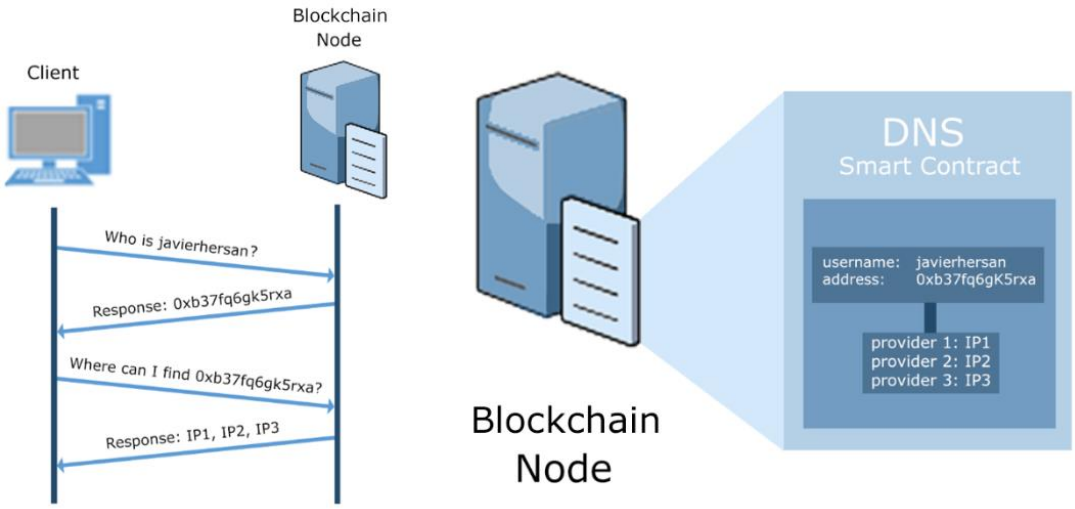

Figure 34 - Blockchain DNS

In the next figure, **Figure 35**, is shown the DNS storage. This smart contract will need two mapping structures. Mapping structures are very efficient in lookup operations since mappings are hash table structures. These mappings will be used to store the owners of usernames and the providers of users as is described in **Figure 35**.

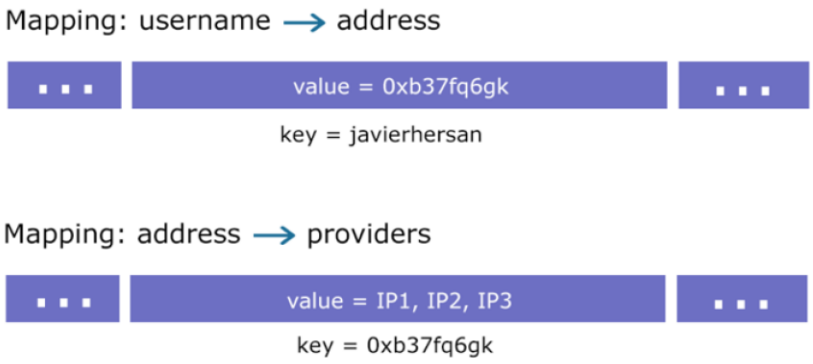

Figure 35 - DNS Smart contract storage

Users will need a method for registering usernames and another method for changing their providers. The most important thing to be developed is the ownership property. The smart contract should not enable to reserve usernames that are already in use by another account and either do not alter providers of other accounts. An expiration date property can be added also to the DNS usernames.

We have already seen how the blockchain DNS works, we just need to see how the deal system works to fully understand the smart contract. The deal system will be explained in detail in subsequent sections. In the next section of the design chapter we will look at the different type of interactions available in the social network, and how they relate to the different components of the different actors of the network.

5.5. Network Interactions

Within the decentralized social network that we are designing, there are five types of interactions available. These interactions are the following:

- Public publications: Users can publish public posts to their profile.
- Private communication channels: These communication channels allow end-to-end private chats between users and additional features such as private publications.
- Private publications: Users can publish private posts on their profile. This interaction is a subset of the first interaction, public publications, made possible by private channels.
- Deal system: The deal system is the process in which each user chooses their providers.
- Following and unfollowing interactions: These interactions allow users to follow their friends and other accounts.

Throughout this section, we will look in detail at how each of these interactions work and how they are integrated into the components of the different network actors.

5.5.1. Public publications

Public publications are the first type of interactions allowed within the social network. Publications will be stored across different providers, so to maintain compatibility and interoperability between providers, a standardized protocol is needed for both access and storage of information.

Compatibility is crucial to ensure data potability and destroy provider lock-in, which prevents data governance. A standardized protocol ensures data governance by allowing users to transfer their data and publications to any provider that implements it.

For this reason, a standardized API will be used to access the information. Therefore, if a user wants to view the profile of a particular account, they will first search for **https://<server-provider-ip>/<account>/** which is the root entry-point of any account. The entry-point will retrieve account metadata.

To request an account-specific resource, an URI extension will be added to the entry-point. For example, if a user wants to view the post number *n*, they will request **https://<server-provider-ip>/<account>/post/<n>**. The **Table 5** shows the different *publication* endpoints available for a given account. However, this is a reduced version, to check a more advanced API implementation navigate to **Appendix B**. More API endpoints will added later to the standard for the remaining interactions.

REST API Endpoints		
HTTP	Endpoint	Action
GET	https://<server-provider-ip>/<account>/	**Root entry-point.** Returns <account> metadata.
GET	https://<server-provider-ip>/<account>/post/<n>	Get the post <n> of <account>.
POST	https://<server-provider-ip>/<account>/post/<n>	Publish the post <n> of <account>.
PUT	https://<server-provider-ip>/<account>/post/<n>	Update the post <n> of <account>.
DELETE	https://<server-provider-ip>/<account>/post/<n>	Delete the post <n> of <account>.

Table 5 - Rest API endpoints: Publications

Before using the API, users will need the IP addresses of the providers of the account they want to view, which will require a DNS lookup before using the API. Once the providers of an account are known, users can search for its profile across the Internet. This is what **Figure 36** illustrates, where a user requests ''account *A*'' profile by sending an HTTP request to all their providers through the standardized API. The more providers an account has, the more censorship resistant it will be. This is the purpose of having more than just one provider. But later we will analyze this problem in detail, for now the focus is that there may be more than one provider and that it is important to defeat censorship.

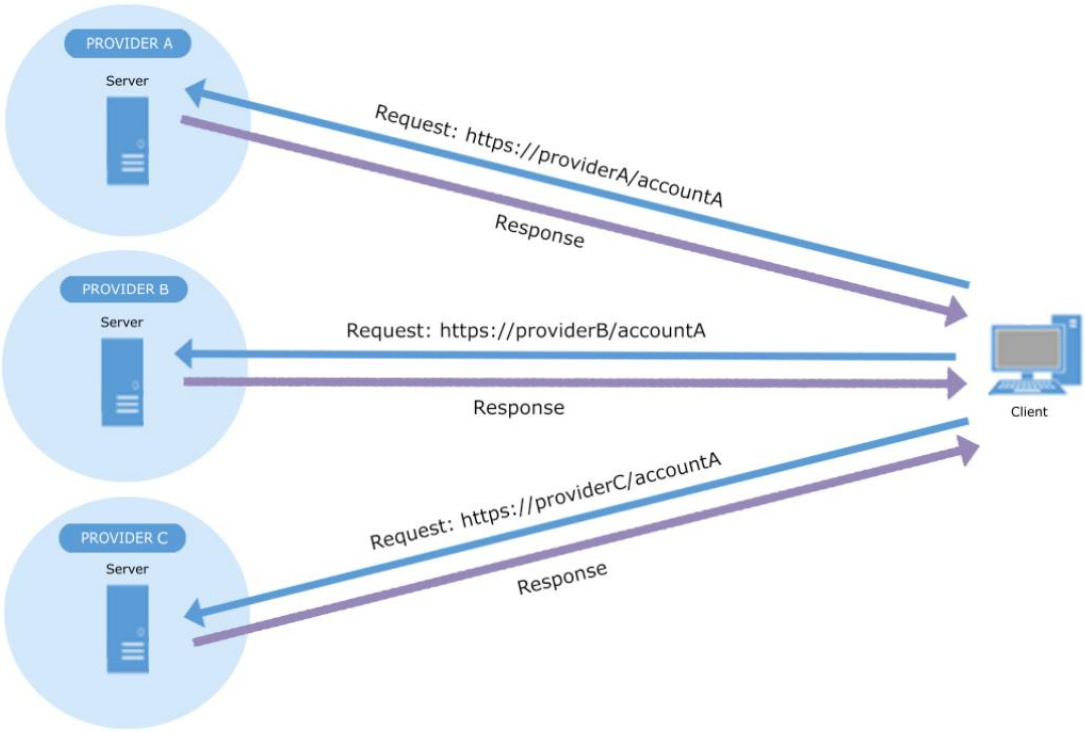

Figure 36 - Multi-provider request

Now we can improve the current system, taking it a bit further. Somehow, we will need an authentication process or a digital identity system. Users need to be 100% sure that only the owner of the account is the one who uploads content to its profile. So, a signature feature will be developed to digitally sign each post or interaction within the platform.

Since an Ethereum account is needed to interact with the Ethereum blockchain why not using the same account and its key-pairs for the signature process too. The signature feature is described in **Figure 37**. First, the content (text, audio, photo, or video) and some associated metadata are bundled and passed through a hash function. This process is called hashing. The hashing process return an output hash code. Then, using the output hash, a header of the post is generated. Finally, this header is digitally signed by the owner of the content.

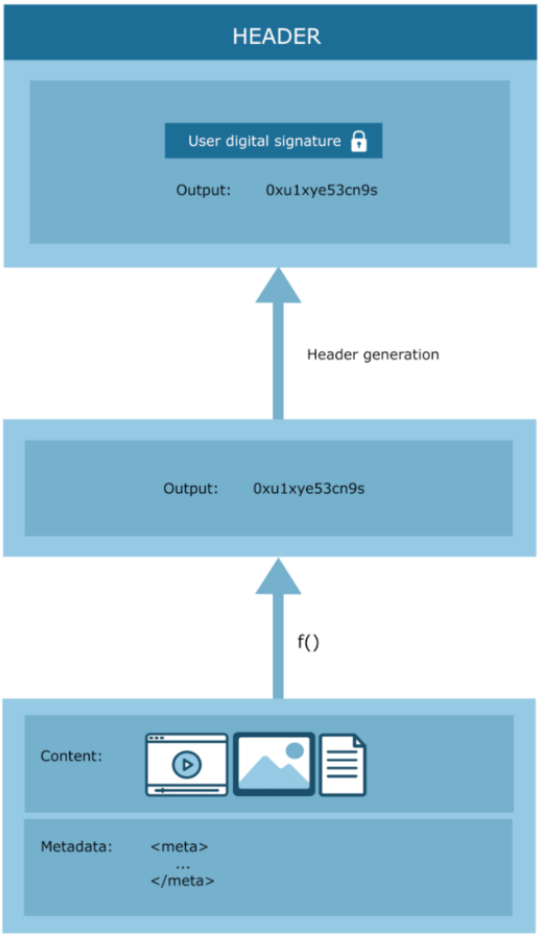

Figure 37 - Post headers I

Recalling the concepts of chapter three, with this feature, the authenticity of the content is guaranteed thanks to the digital signature and the content verification hash. As a result, users are completely sure that the post is authentic and belongs to the profile owner. Thus, every content request will have the post header as an attachment in order to check the authenticity of the content on the client side before displaying the post on the screen. So, it is mandatory for the providers to store block headers along with the content of their users.

The post header has many advantages, but users cannot yet check the order of the posts. An incremental number can be added to the header denoting the post number. Also, another incremental number can be added to indicate the number of times a publication has been updated. Since the header is signed, users can be 100% sure that these incremental numbers, the block-id and update-id, are real. Henceforth, the block header will have the aspect of **Figure 38**.

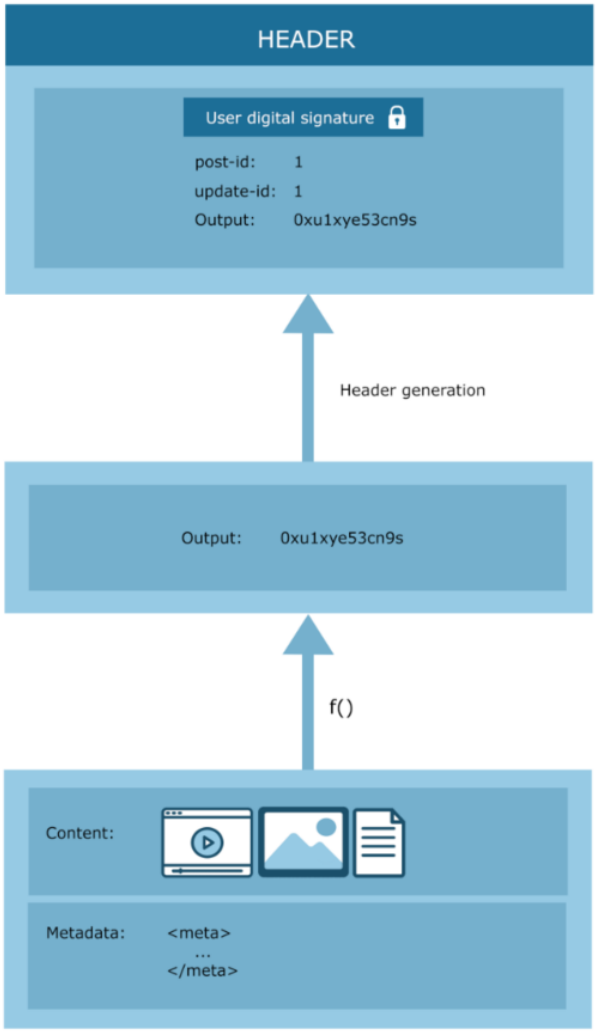

Figure 38 - Post headers II

This is where having more than one provider plays a very important role, let's see why. Imagine a user requests the most recent publication from a specific account, and the two providers of that account respond with two different posts. This situation is illustrated in **Figure 39**.

Initially, the user requests the latest publication from the account. To accomplish this, the user makes an HTTP GET request to both providers targeting the entry-point of the account, in order to get the associated metadata. Both providers return different metadata. According to *Provider A,* the seventh post is the most recent, whereas *Provider B* claims it is the eighth. Which provider should the user trust? In order to resolve their doubts, the user will request a proof of it and this proof is the content header we have just seen.

For this reason, the user will make a second request to Provider A and Provider B requesting those posts alongside their headers. *Provider A* returns a header with *post-id*=7 and *Provider B* returns *post-id*=8. Which provider should the user trust now? The answer is the one that has the longest chain or largest *post-id*. Therefore, in this case, the user should trust the one that returned the publication with *post-id*=8, because the header is digitally signed by the owner, so the user is 100% sure the owner published that post. This is where we see the importance of content headers and having more than one provider, to avoid censorship.

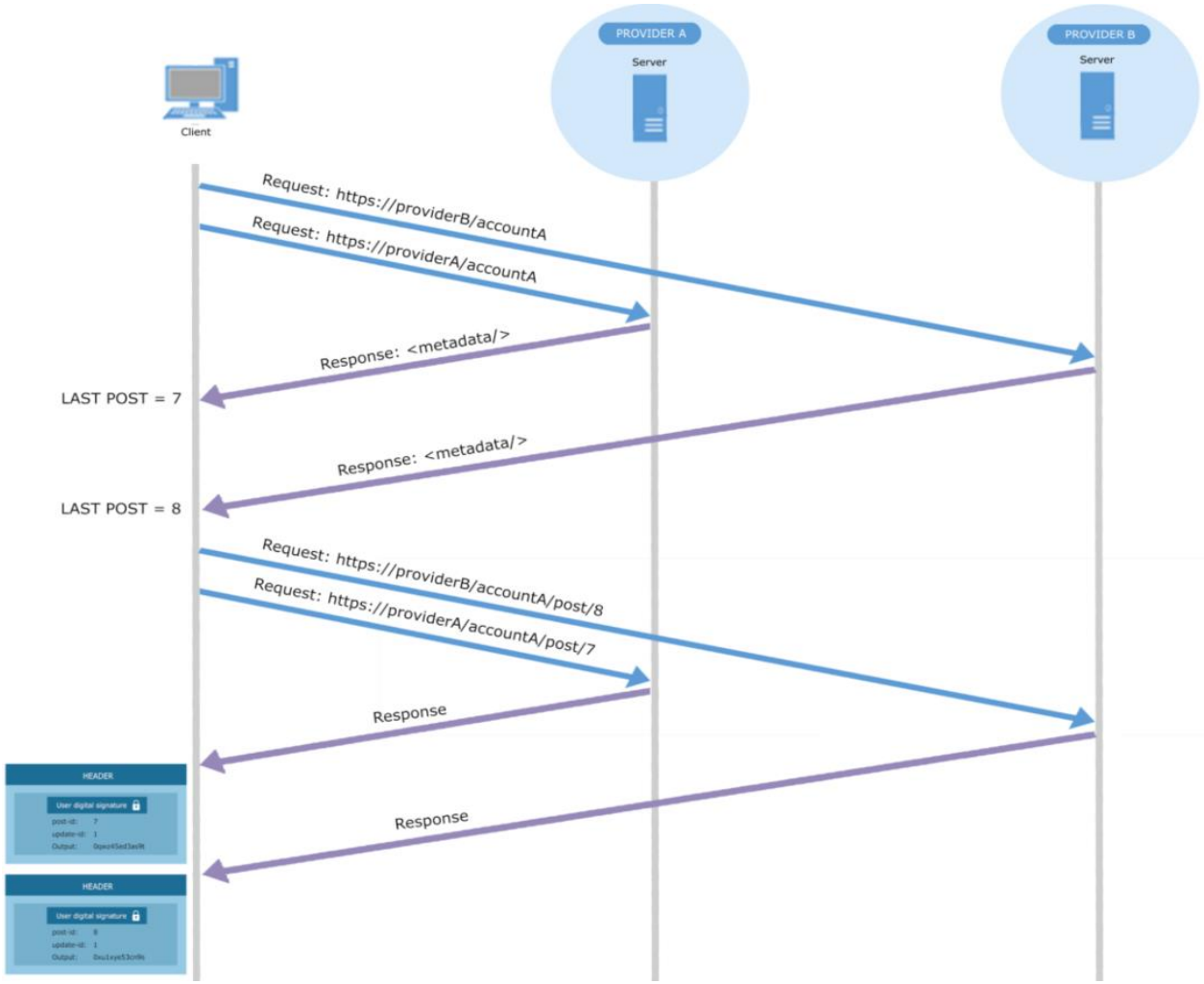

Figure 39 - Requesting the latest post from an account

The requests made in **Figure 39** can be aggregated into a single request to avoid latency, but the optimization processes are discussed later in the implementation chapter. Users also need the *update-id* field in the header for a similar reason. A post can be updated many times, therefore the last update is the one that should be shown in the screen, so the same logic applies here.

Previously we mentioned inconsistency problem between providers, and we have just seen how this problem has been satisfactorily solved thanks to the header system, despite the fact that the providers had different information.

The process for checking the headers is described in detail in **Figure 40**. The first stage is header verification, in which the signature of the user and the output hash are checked. The second stage is header comparisson in which the user compares the headers returned by each provider, displaying the content corresponding to the largest correct header.

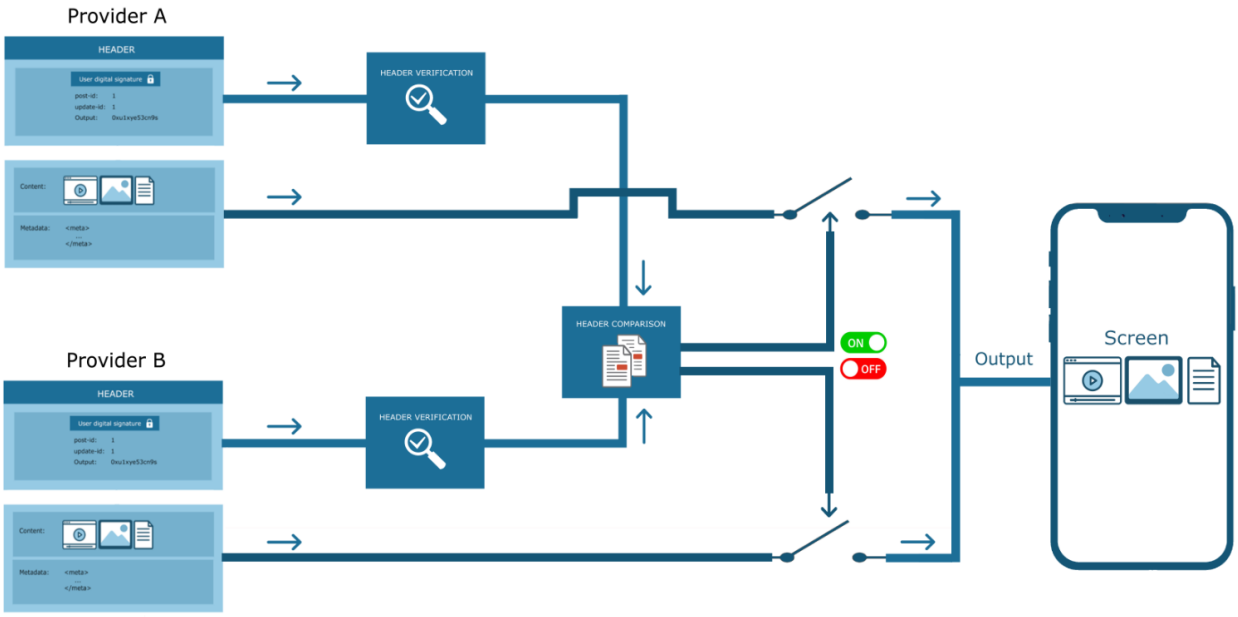

Figure 40 - Header Verification System

The header verification system of **Figure 40** is included in the interactions module. This module is present both in the client and server architectures to compare headers and check their correctness.

Now, the standard is censorship resistant thanks to the header system and the decentralization of providers. So, as long as one provider offers the content, the censorship will be defeated. For this reason, users will need as many providers as possible, so they can later compare the headers returned by each provider. After checking the headers, users will display the content corresponding to the longest valid header. Therefore, the more providers, the greater the resistance to censorship. A local copy can also be made to avoid the loss of information in case all the user's providers remove the user's information. This way, the user himself could have a backup copy and even upload it again to other providers.

However, having more providers requires more network bandwidth and storage resources. Nevertheless, this approach saves a lot of resources because the header size is really small compared to the size of media content. The total storage size of a single publication follows the equation (4.1).

$$S_{TOT} = N_{PROVIDERS} \cdot (S_{POST} + S_{HEADER}) \qquad (4.1)$$

As can be seen from equation (4.1), more storage is consumed as the number of providers increases. Having more providers is better for censorship resistance, but it must be known that more network and storage resources are spent. These resources may or may not be free depending on the providers and their available services. So, this should be considered when choosing the number of providers for your account.

What we know for sure is that blockchain should not be used to solve this problem. Because users do not need that level of redundancy. Also the publications need to be mutable and erasable, which is also not possible with blockchains. Hence, the header system approach is the best suited for decentralized social networks in terms of decentralization and resource savings.

So far, we have solved the censorship problem. But if we recall from **Chapter 2**, when censorship disappears, a new associated problem comes up. Censorship resistant networks can be exploited by malicious actors who can spread false and harmful content. This is a direct consequence of free speech absolutism.

Therefore, another tool should be implemented on top of the standard to filter the content. This tool is the last stage represented in **Figure 41**, which is a content verification module. This stage will filter the content based on certain criteria set by the user. It is important to note that it is the user who ultimately decides the filtering conditions and whether to apply them or not, unlike current social networks where the company itself decides not to show certain content. This module is configurable by the user, thus respecting the individual freedom.

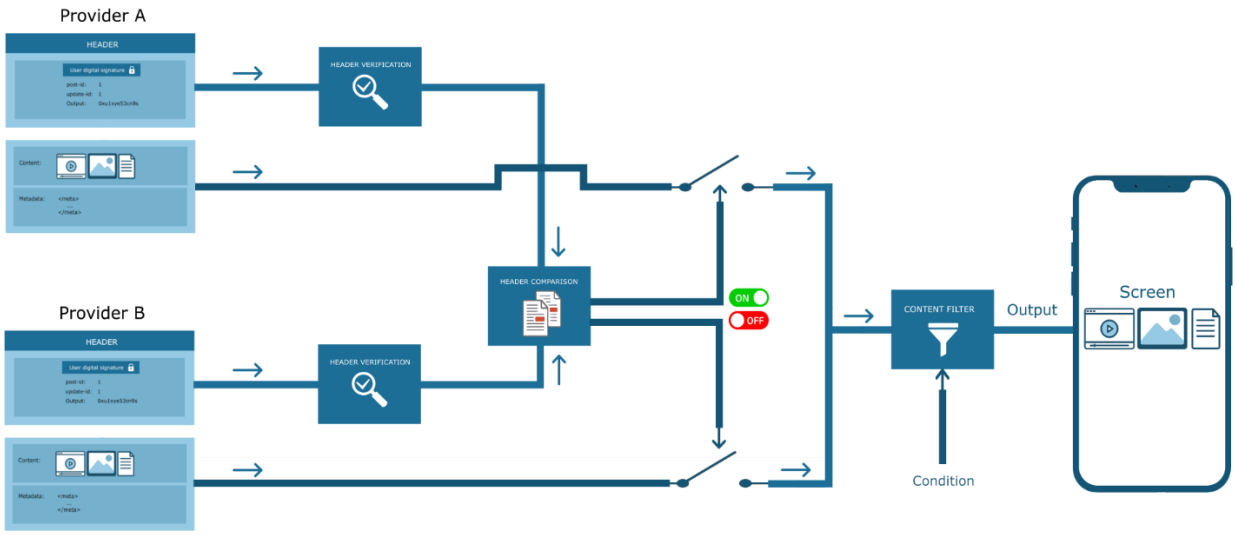

Figure 41 - Content display pre-processing

The filter module is shown as a black box system, since its implementation is out of the scope of the standard. In most cases the filter may be probably an Artificial Intelligence Content Filter (AICF). However, in the next chapter an AICF will be developed in detail during the development chapter. The filter can be implemented in many different ways, but let's leave it aside for now. This filter is also added to the interaction module of the clients.

5.5.2. Private communication channels

The next type of interaction is private channels. Private channels are private communications between two users, in which the communication is fully encrypted *end-to-end* where no *man-in-the-middle* can have access to the encrypted channel. This is what **Figure 42** illustrates. Imagine *user A* and *user B* want to set a private communication channel. First, *user A* will generate a key-pair to afterwards send the public key to *user B*. With the public key, *user B* can encrypt all its messages and be completely sure that only *user A* can decrypt its messages, since *user A* is the only user who has the decryption key.

Due to the architecture of the Internet, messages cannot be sent directly between users, for this reason the providers will be used as proxy servers. This is mainly due to the presence of NATs and the variability of end-user IP addresses. Thus, it is the user who requests any updates to its channels by sending an HTTP GET request to its providers. Once, *user B* receives the public key of *user A*, *user B* generates a key-pair and shares the public key to *user A*.

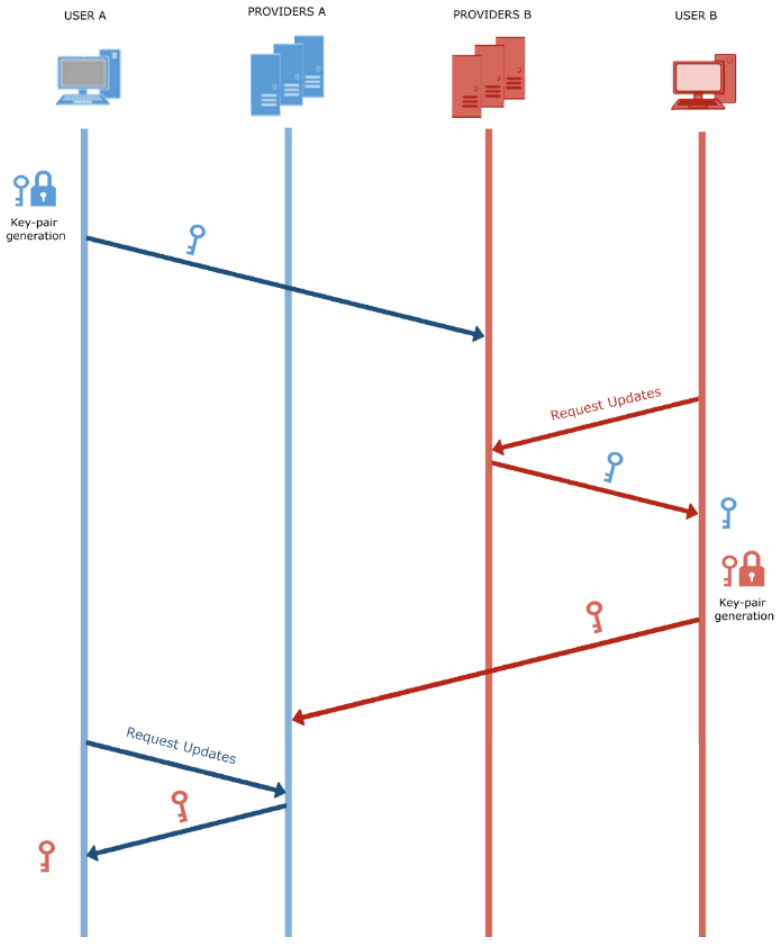

Figure 42 - Setting up a private communication channel

Once, *user A* and *user B* have their public keys to encrypt their messages, they have successfully established a private communication channel. This is what **Figure 43** shows. First, *user A* encrypts a plain message and sends it. Then, *user B* gets the message by requesting to its providers the latest channels updates. After receiving the message, *user B* decrypts it and reads the plain message successfully.

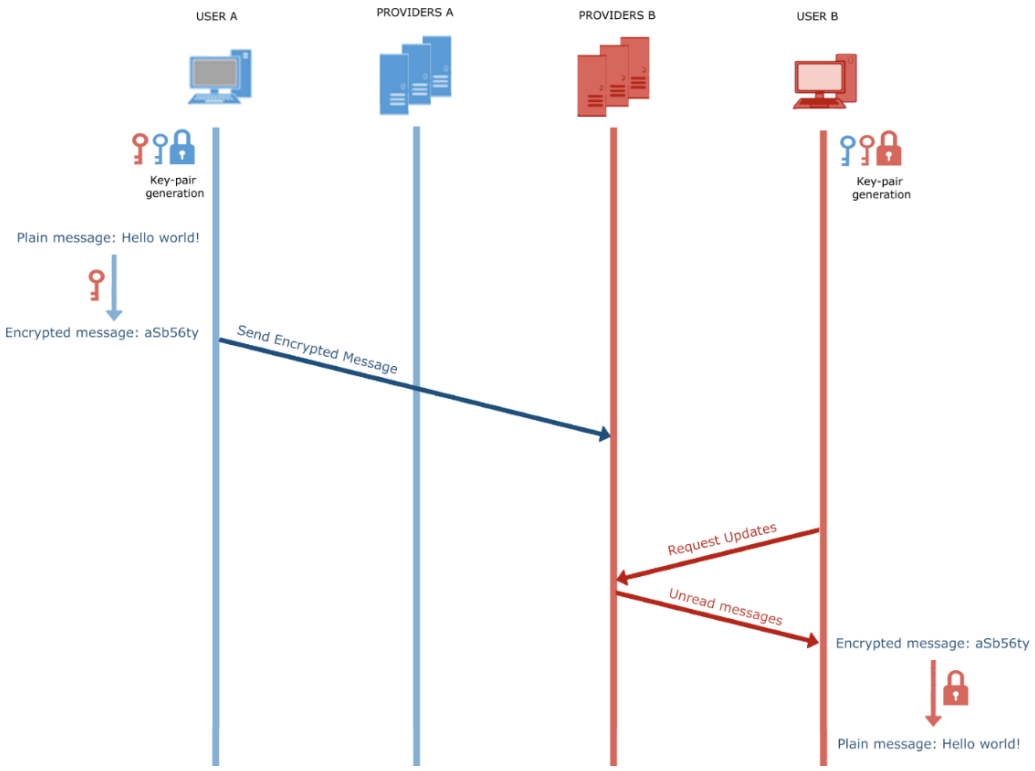

Figure 43 - Sending messages over private channels

The key pairs can be reestablished periodically and at any time for security reasons, in order to maintain a more secure channel over the time. All messages come along with content headers. This way, users can be 100% sure that it is the user who is sending these messages. This is why header signature and content verification are present throughout the entire encryption and message processes. Also, incremental numbers are needed in those headers to know the order of the messages.

New API endpoints are required to access the latest channel updates and send messages through them. This API endpoints are available in **Table 6**. This table is a reduced version of the API, for a more complete API implementation navigate to **Appendix B**.

REST API Endpoints		
HTTP	Endpoint	Action
GET	https://<server-provider-ip>/<account>/channels	Get the latest messages from the different channels
POST	https://<server-provider-ip>/<account>/channels	Send a message to an <account> channel

Table 6 - Rest API endpoints: Private channels

To read the latest channels updates, the user will need to make an HTTP GET request to **https://<server-provider-ip>/<account>/channels** and its providers will respond with the latest messages from the different channels. This endpoint is public despite the messages being private, but since all messages are encrypted, it is not a big problem. To send a message to a specific user, an HTTP POST request will be send to **https://<server-provider-ip>/<account>/channels**. The shared keys are stored by the users in the local storage module of the application. This module is a very secured module and access from the outside is not allowed.

An access signature (AS) could also be implemented in each API request to stop DoS attacks over providers by allowing only authorized access or real user requests. With this signature present in each request, the provider can detect the user trying to reach the public endpoint and deny its access if it is not a real user. Providers can verify if they are real users or bots by checking the blockchain activity, because real users have at least performed one operation in the blockchain DNS. The access signature (AS) can be implemented with a header similar to the content header, however, we will not go into the details, but leave it as a possible protocol enhancement.

5.5.3. Private publications

Once the private channels are implemented, we can start implementing private publications. Private publications have to be encrypted too, so a new key-pair is required. This key pair is stored locally on the user's device, specifically in the local storage module.

Figure 44 - Profile encryption key-pair

The user will use the public key to encrypt all the publications and, counterintuitively, the user will share the private key with the accounts the user wants to give access to their profile. This key will be shared secretly through the private channels the user has set up before with those accounts, this way, no authorized users can access the decryption key. **Figure 45** shows how this private key is exchanged through a private channel.

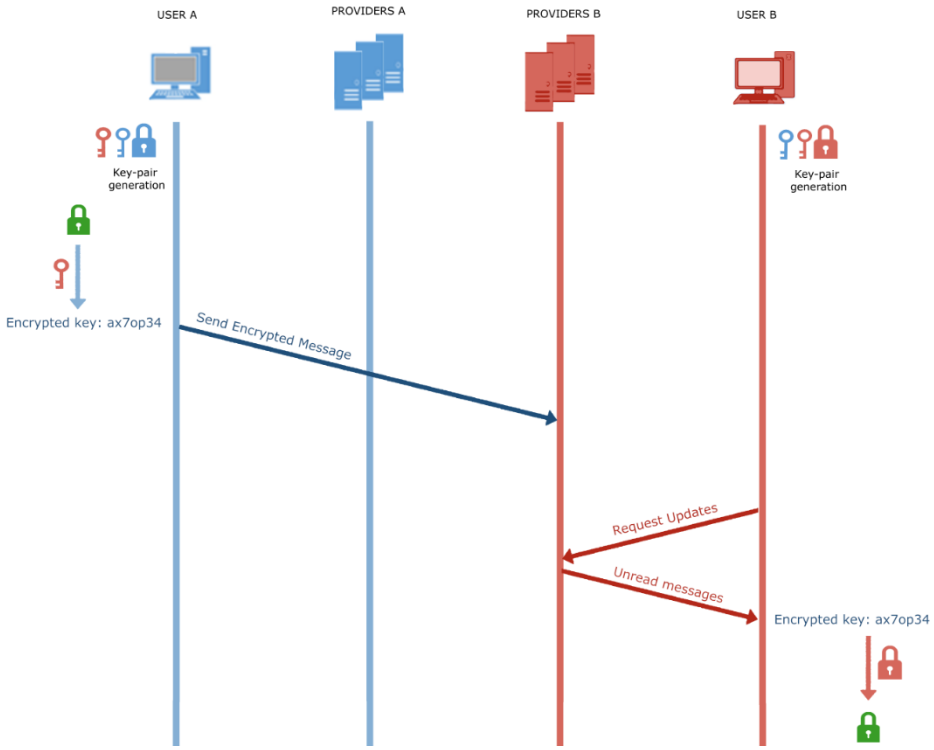

Figure 45 - Profile Encryption: Private key exchange through private channel

Figure 46 illustrates how post-encryption and new header generation work. The user encrypts the content with the public key. Then, the encrypted content is passed through a hash function giving as output another hash code. Both output hash codes, the encrypted and unencrypted are included in the header.

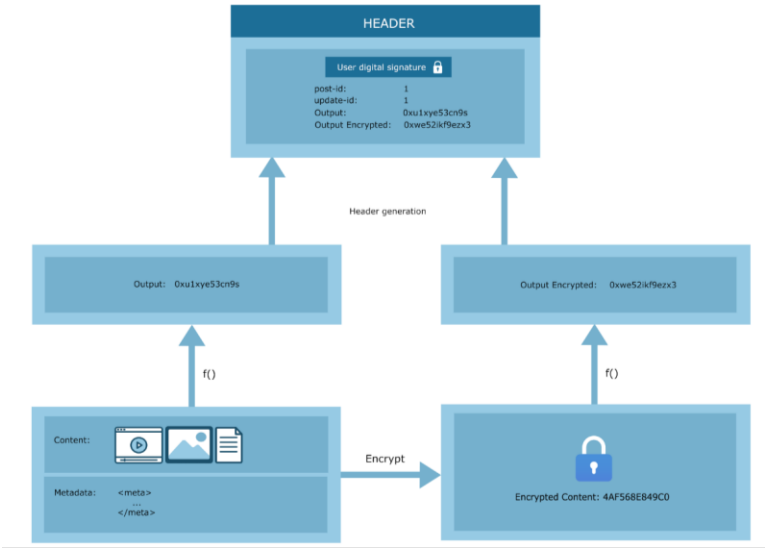

Figure 46 - Profile Encryption: Header generation and encryption process

Once the header has been generated, the user publishes the publication and sends it to their providers, sharing only the encrypted content and the header. In this way, the providers are not even able to access the publication. Then, the user shares the private key to decrypt the publications with the users that they want to give access to, through the private channels as we have seen before.

A new module is added to the user application. This module is the decryption module. This module is added be before the content filter as it is shown in **Figure 47**. This module is also added to the interactions module of the client.

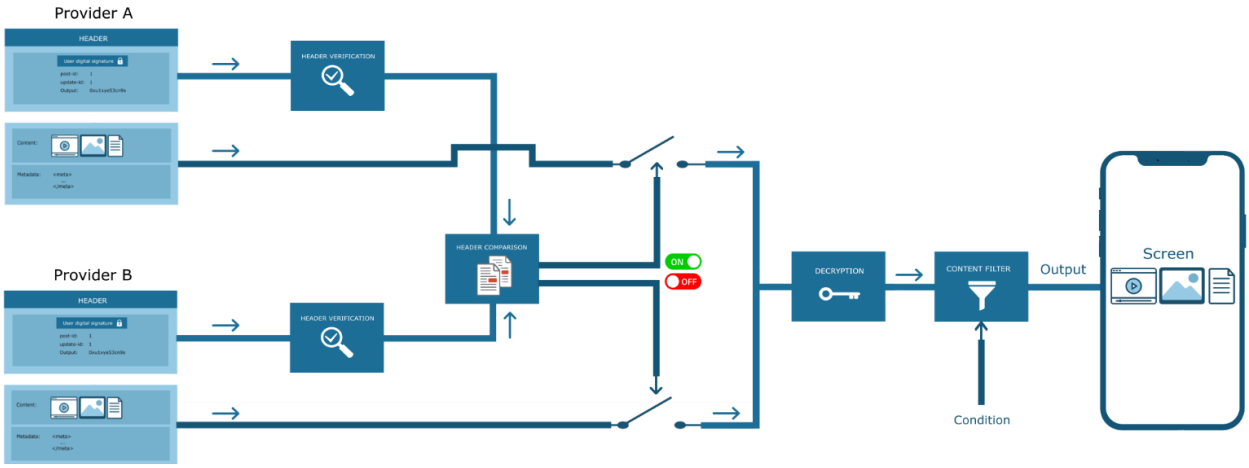

Figure 47 - Content decryption component

This is how private publications are managed within the social network. However, other alternative privacy models can be built. From now on, we can say that the standard is data-governance compliant, as users have complete control over their data, identity and now privacy.

5.5.4. Following and unfollowing interactions

For unfollow and unfollow interactions, users just have to reach the endpoints of **Table 5**. To avoid impersonation, users will attach the content header along the request body. For a more detailed implementation of the API navigate to **Appendix B**.

REST API Endpoints VI		
HTTP	Endpoint	Action
GET	https://<server-provider-ip>/<account>/following/<address>	Get follow list.
POST	https://<server-provider-ip>/<account>/following/<address>	Follow/Unfollow an account with <address>.

Table 7 - Rest API endpoints: Following and unfollowing interactions

5.5.5. Deals process

The last type of interaction allowed in the social network is the *deal interaction*. As you may recall, *deal interaction* is the process in which providers upload the service level agreements to the blockchain, and the users choose the provider and service that best suits them.

To summarize, the *deal* process is the process in which providers are chosen from the perspective of the user and what infrastructure companies get in exchange for providing those services to users. These benefits may not necessarily be an economic benefit, the benefits may be of a different nature as we will see soon. In **Figure 48** the whole deal process is explained.

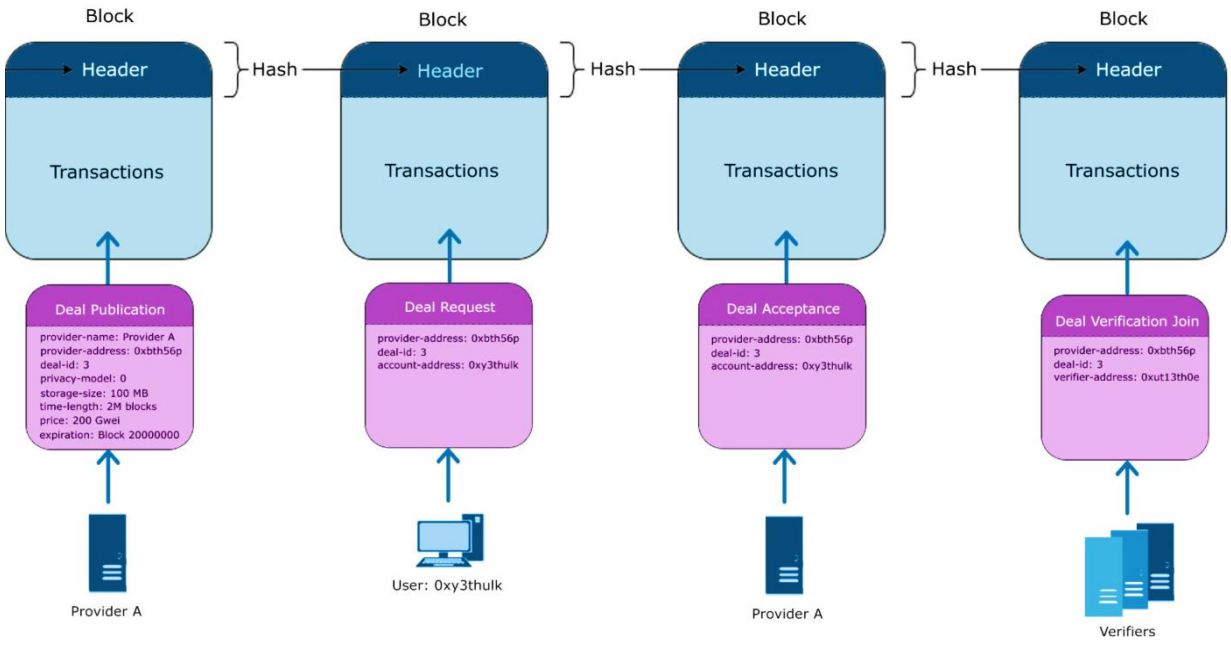

Figure 48 - Provider deals

First, providers publish their deals on the blockchain ledger. After reviewing the deals from different providers, users choose the ones they like the most, to do this, they send a deal request to the blockchain. Later, it is the provider who must decide whether to accept it. Finally, validating nodes can optionally join the deal. Validator nodes will provide an additional layer of security to users as we will see soon.

In **Figure 49** a deal publication is shown in detail. First, we find the provider's name and provider address. Providers can reserve names in the same way that users reserve usernames. This is done through a provider domain name. This is done to filter deals by provider name and no by their address, which is by far less user-friendly. Then, the deal has an id associated (*deal-id*) to identify it between multiple deals within the same provider. Finally, we found the deal options which are *privacy-model*, *storage-size*, *time-length*, *price* and *expiration*.

Figure 49 - Deal publication

The *privacy-model* field indicates the privacy of the user's profile. For more details, see (4.2).

$$privacy\text{-}model \begin{cases} 0, \ public \\ 1, \ private + provider\ access \\ 2, \ full\ private \end{cases} \tag{4.2}$$

The *storage-size* indicates the maximum amount of storage the profile can occupy on memory. On the other hand, *time-length* indicates the time duration of the deal in Ethereum blocks. We have taken Ethereum as example because it is the blockchain we have decided to use later in the implementation chapter. If you follow Equation (4.3), you can see 2M blocks length is approximately 277 days, since a new block is added to the Ethereum blockchain every 12 seconds.

$$time\text{-}length\ (days) = 2M\ blocks\ \cdot 12\ s/block\ \cong\ 277\ days \tag{4.3}$$

Then, the *price* field which is 200 Gwei, which is approximately 0.03$ as you can see from Equation (4.4). This deal is extremely cheap, but users have to consider the network fees for transacting in the blockchain, so a little bit more is expected. Deals can also be signed *off-chain* to save on transaction costs, but due to the complexity we will not go to deep on that during the project.

$$price\ (\$) = 200\ Gwei\ \cdot\ \ 2 \cdot 10^{-7} \cdot\ 1800 \frac{\$}{ETH} \cong\ 0.03\ \$ \tag{4.4}$$

Finally, the expiration which is set in block 20,000,000. At the time of writing, the Ethereum blockchain is in block 17,463,430. If the Ethereum blockchain issues a block every 12 seconds, the deal will expire in about a year. In **Figure 50**, the *Deal Request*, the *Deal Acceptance* and the *Deal Verification Join* transactions are shown in detail. In the three transactions the deal is identified by the *provider-address* and the *deal-id* fields. Then, the *account-address* field indicates the account that wants to set the deal. Finally, the *verifier-address* which is the Ethereum address of the verifier joining to monitor the deal.

Figure 50 - Deal request, deal acceptance and deal verification

This is the process by which users select their providers, first they search in the blockchain for specific deals by filtering them and select the most suitable for them. But you are probably asking yourself why there are verifiers joining those deals. This is to ensure that providers are acting honestly by fulfilling their deals. Here, is where Proofs play a fundamental role. The verifiers can be any user wanting to secure the deals or can be other providers of that same account, as they have to store the user's information in the same way, so why not join this process too.

The unique proof we will examine in this chapter is Proof of Storage (PoSto), but other proofs can coexist. In Proof of Storage, the content is fragmentated first in several chunks. For instance, in **Figure 51** the content is fragmented into four chunks.

Figure 51 - Data fragmentation

Then these chunks or data blocks are passed through hash functions. The resulting hashes are hashed up to form a Merkle tree similar to the one shown in **Figure 52**. As we have seen in chapter three, any child hash can be proven that belongs to the root hash of the tree, so Merkle trees becomes an extremely useful tool when proving if some chunk of data is present within a *dataset*.

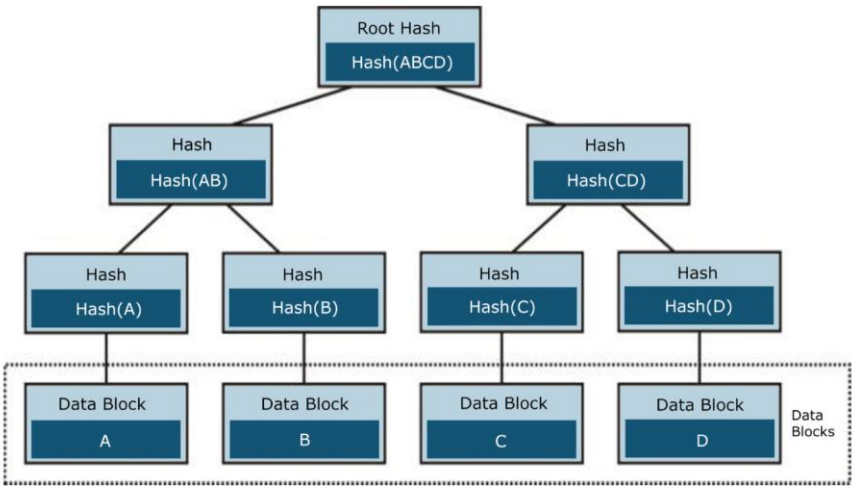

Figure 52 - Proof of Storage Merkle tree

So Merkle trees can be used as a Proof of Storage. The verifiers can request to providers a Proof of Storage for a specific chunk. If the providers failed to provide the proof, they are penalized with some Ether, that is why providers will need to block some money during the deal, to be penalized in case they actuate dishonestly. The proofs are the parent hashes of the data chunk. Finally, these hashes can be verified that belong to the root hash, thus successfully completing the Proof of Storage process.

The Proof of Storage process can start after the deal is set up. The Proof of Storage process is depicted in **Figure 53**. The Proof of Storage process starts as follows: First, the user fragmentates the content of their profile to finally calculate the root hash of the Merkle tree. Then the user sends a INIT FLAG to the blockchain with the root hash. Immediately after that, the user sends the data to their providers and deal verifiers. The providers and verifiers do the same process and calculate the root hash of the Merkle tree. Finally, the providers and verifiers send a START FLAG to the blockchain confirming the root hash. If the root hashes match the Proof of Storage process has started successfully.

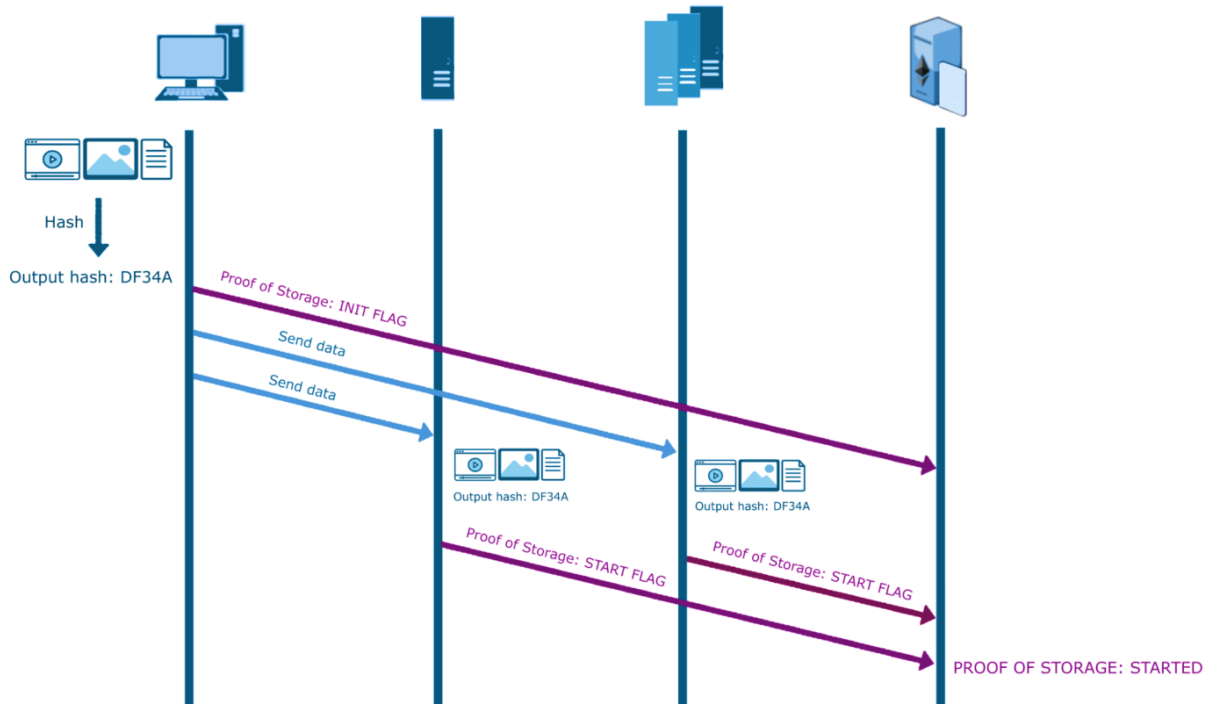

Figure 53 - Proof of Storage starting process

During the starting process, the providers lock some ether to be punished if they do their job badly. Once the Proof of Storage process starts, any validator node or even the user itself can request a proof to the providers. This is what **Figure 54** represents.

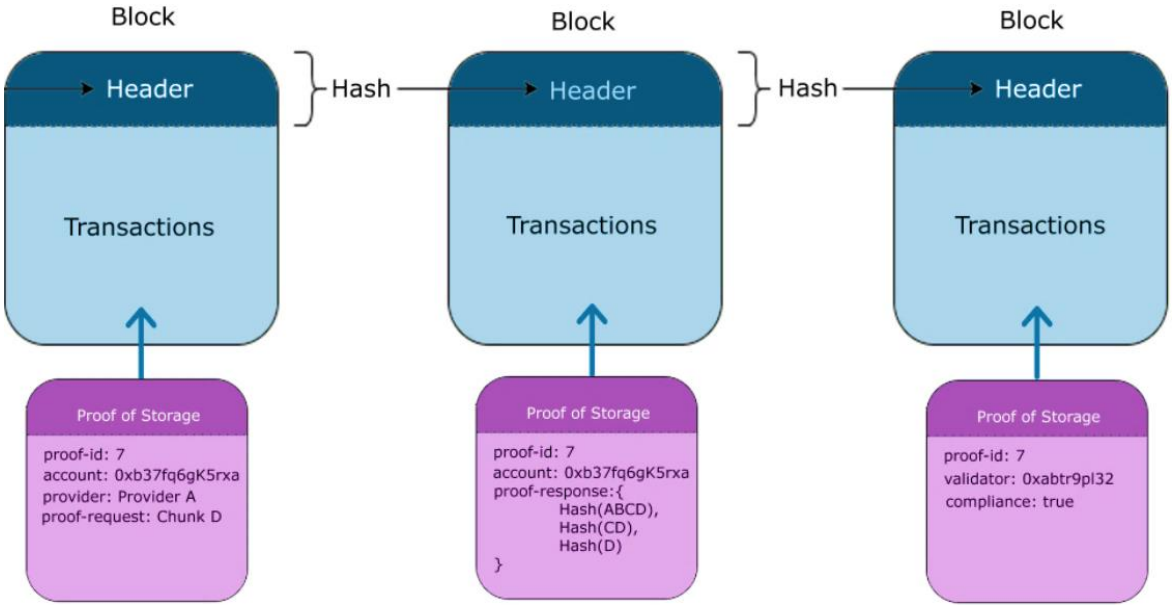

Figure 54 - Proof of Storage: Presenting a proof

First, a verifier requests a proof of storage for Chunk D. Then, provider provides as proof the hash path. If the proof is valid the verifiers set the Proof of Storage to compliance mode, if not the provider is penalized. With Proof of Storage, users force their providers to always store their information since they can be requested to present a proof at any time, being penalized for not fulfilling the deals. With this mechanism, users can be 100% sure that the providers are going to store their information.

However, users cannot be 100% sure that their information is retrieved to other users when requested. For this a more complicated and sophisticated technique needs to be developed which is Proof of Retrieval (PoR).

Proof of Retrieval could be implemented in the blockchain, but it would be an expensive proof and not a feasible one, since requesting the proof every time a user makes a request, would imply a blockchain transaction. More sophisticated techniques need to be implemented in combination with *off-chain* transactions to scale it up and reduce the costs. IPFS [10] and Filecoin [11] for example do not implement Proof of Retrieval since they use a retrieval payment [12] each time a user makes a request. However, this approach is not feasible within the social media landscape, so Proof of Retrieval is a better option.

Due to the great complexity of Proof of Retrievals and the limited time frame of the project, this functionality will not be developed. However, at least Proof of Storage should be enough in many cases, as the provider is forced to store all the data during the deal.

The design is now finished. Now we are prepared to build and implement the decentralized social media that implements this design, this is what **Chapter 6** is all about.

6. Development of a decentralized social media platform

In this chapter, we will review the technical implementation of the standard designed during **Chapter 5**. We will implement both the frontend and the backend parts of the application. On the client and server side we have used JavaScript language. On the other side the DNS smart contract has been implemented with Solidity language to deploy it later on a Ethereum compatible blockchain used for testing. All the code is available at [13]. **Figure 55** shows a network architecture with the main technologies used during development.

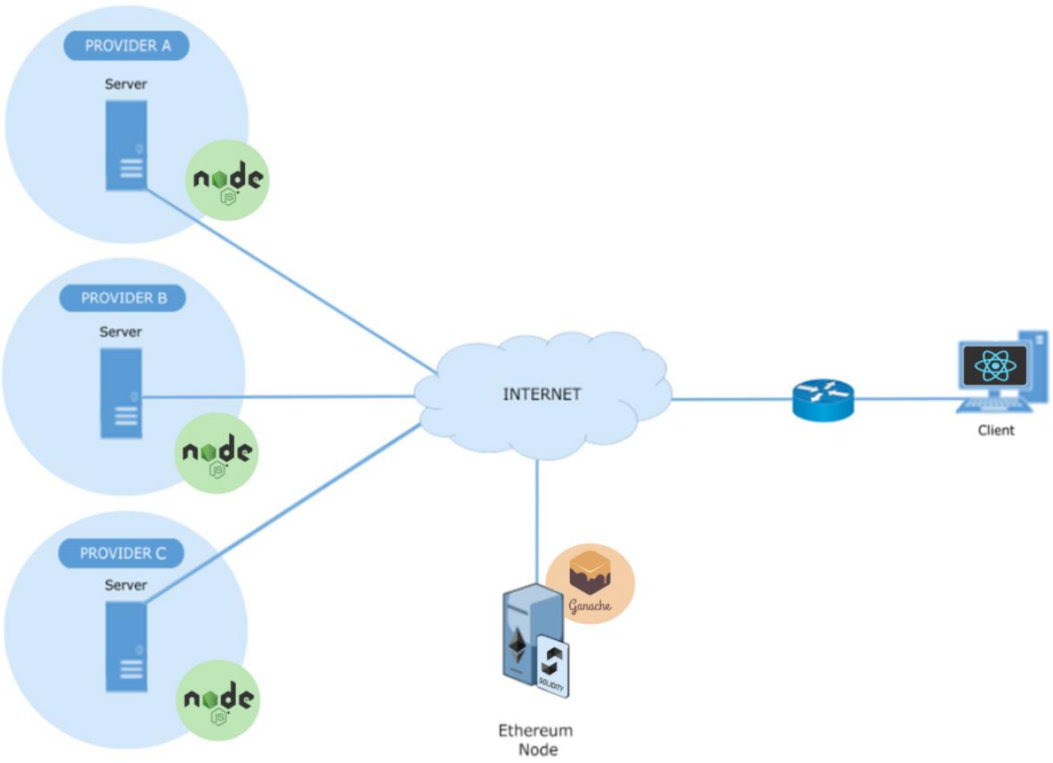

Figure 55 - Technical implementation of the standard

6.1. Server

The server is implemented with JavaScript and NodeJS [14]. These servers use ExpressJS [15] which is a very useful framework for developing REST APIs on NodeJS.

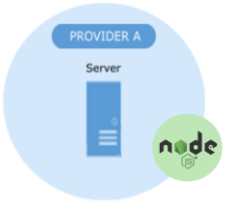

Figure 56 - NodeJS Server

If you recall from **Chapter 5** and **Figure 31**, the server has the following submodules:

- **API Module**: ExpressJS.
- **Interaction Module**: JavaScript self-made libraries.
- **App Module**: NodeJS.
- **Database Module**: JavaScript lightweight database.

First, we will start with the API Module implementation. We have built a REST API that implements all the endpoints viewed during the design to support the different network interactions available. These endpoints are available in **Appendix B** with all their details. These details give relevant information about the endpoint route and the json schema of the request and the response bodies. The file structure of the server is available in **Figure 57**.

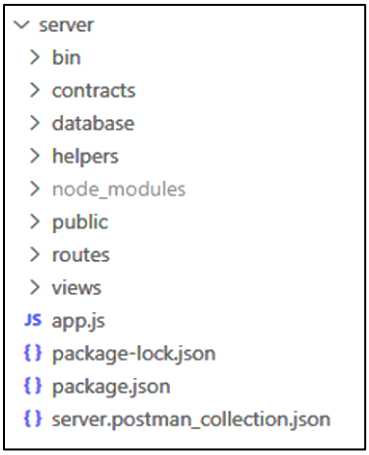

Figure 57 - Server File Structure

The API is implemented with ExpressJS which is a lightweight framework for building APIs. The API implementation is inside the *routes* folder in the *index.js* file. When the API receives a request, the server sends the interaction with its request body to the interaction module.

The interaction module has been developed with the help of some self-made JavaScript libraries. These libraries are available in the *helpers* folder. These libraries will help the server to perform header verification, which allows providers to store content that has only been published by the owners of the publications. This is extremely important because these headers act as a layer of authentication.

The interaction module also compares the request body with a json schema object to check if the body and its content are correct depending on the type of the interaction. These schemas are available in the *database* folder which is also used for the database. Finally, other checks are done depending on the endpoint, to avoid storing redundant or outdated information inside the server. If the request meets the requirements the interaction is executed successfully.

Finally, within the *bin* folder we find the *www* file. This file allows us to change the configuration of the server, such us the port it listens on.

6.2. Blockchain

To simulate the Ethereum blockchain, Ganache network will be used. The Ganache [16] network is a single node blockchain network and it is perfect for testing scenarios since it is a lightweight blockchain. The deployed node mines a new block every time it receives a transaction.

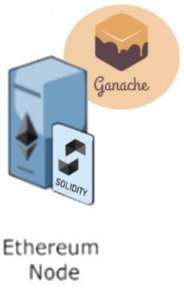

Figure 58 - Ganache blockchain node

Ganache is compatible with Ethereum based applications and Ethereum smart contracts, so we can deploy Solidity smart contract over Ganache. First, we will start a *Truffle* project. *Truffle* [17] is a suite of tools for developing smart contracts. To start a Truffle project we will execute the commands of **Figure 59** in order.

```
$ mkdir truffle
$ npm init -y
$ npm install truffle
$ npx truffle init
```

Figure 59 - Init Truffle project

Once, we initialize the Truffle project, a file structure like the one in **Figure 60** should have been created. The most important folder is *contracts* where all the Solidity smart contracts are going to be placed. The *build* folder is where the compiled smart contract will be placed. Solidity needs to be complied into Ethereum Virtual Machine (EVM) code.

Figure 60 - Truffle File Structure

The *test* folder is used for testing the smart contracts. It is really important to test the smart contracts before deploying them in production, since the funds of many users are at risk. The *migrations* folder is where we deploy the smart contracts following the configuration of the *truffle-config.js* file. We will not go into too much detail, since we will cover it later when deploying the smart contract.

Now, we will create the *DNS.sol* smart contract within the *contracts* folder. This smart contract will have two mapping structures. Mapping structures are very efficient in search operations since mapping are hash table structures. These mappings will be used to store the owners of usernames and the providers of users as is described in **Figure 35**.

In consequence, users will need a method for registering usernames and other for changing its providers. The most important thing to be developed is the ownership property. The smart contract should not enable to reserve usernames that are already in use by another account and either do not alter providers of other accounts.

An expiration date property can also be added to the DNS usernames which can be easily implemented. For example, imagine we want to add a one year expiration date. One year is equivalent is equivalent to 2.6M blocks in Ethereum. So, anyone can reserve a username if the owner of that username has not renewed it after that period. To implement this feature we will need another mapping structure to store the block number when the registration took place and add some logic to check how many blocks have passed since that block. However, we will leave the expiration feature for future lines.

Other important variables of the contract are:

- *Deprecated*: if the contract version is deprecated and there is a new one.
- *Current version*: current version of the contract.
- *Smart contract deployer address*: the address of the account that has deployed the smart contract.

Once, developed the DNS code the Deal code can be added to the contract. However due to time constraints, it has not been implemented. Once, we finished programming the smart contract, we can deploy it on the blockchain. However, we leave the deployment for **Chapter 7**.

6.3. Client

The client application has been implemented using JavaScript. If we recall the architecture of the client from **Figure 32**, we can extract that the main components of the client application are the following:

- **UX Module**: ReactJS
- **API Module**: JavaScript.
- **Interaction Module**: JavaScript self-made libraries.
- **App Module**: JavaScript.
- **Local Storage Module**: JavaScript lightweight database.
- **Blockchain Module**: Drizzle and Web3.js
- **Blockchain Wallet**: Web3 compatible wallet

Users interact with application with the UX Module which is the user interface. This module has been developed with ReactJS [18. ReactJS is a free and open-source front-end JavaScript library for building user interfaces. This library is probable the most used library within the JavaScript ecosystem.

Figure 61 - Client application

First, we create a new react application, for this reason, we will need to create a NodeJS project and then execute ''*npx create-react-app <name>*''. In our case the name we have given to the app is *client*. A folder structure will be created automatically by React. However, React allows great customization of the project structure and we have taken advantage of this to build our own structure. The file structure that we have used is represented in **Figure 62**.

Figure 62 - ReactJS File Structure

The *components* folder contains the different screens and React components of the decentralized application or DApp. The UX Module is responsible of capturing user interactions and send them to the app module. If the interaction requires sending a blockchain transaction, the interaction will be sent to the Blockchain Module. The Blockchain Module is needed to interact with the DNS and to select a provider through the deal system. The blockchain module is connected to the Wallet Module which is the one in charge of sending the transactions to the blockchain nodes via RPC (Remote Procedure Call) protocol.

We have implemented the Blockchain Module with *Drizzle* [19] framework and Web3.js JavaScript library. *Drizzle* and Web3.js provide the necessary tools for interacting with the blockchain and the deployed smart contract. The *drizzle.js* file will instantiate the *Drizzle* and *Web3* objects. These objects include the necessary connection parameters to connect the Wallet Module to Ganache blockchain.

The *index.js* file will initialize the React application components including *Drizzle*. The *Drizzle* object has been included in a React *context* to save it on memory. React *contexts* allow to have permanent object instances between DOM re-renderings. The project has also more *contexts* within the *contexts* folder. These *contexts* have been developed to share some global variables along the different React components.

On the other hand, in the *contracts* folder there is the output file result of the deployment and compilation of the smart contract. This folder is needed in order to *Drizzle* finds the deployed contract in the blockchain and interact with it.

If the interaction is not a blockchain interaction, the App Module will send the interaction to the Interactions Module. As we have seen in **Chapter 5**, this module is in charge of signing publications and interactions as well as checking content headers or even comparing them. To implement the Interaction Module we have used self-made and 3rd party libraries.

One of the 3rd party libraries is Web3.js. Web3.js will allow users to sign their interactions with their Ethereum identity. These interactions despite being signed with the blockchain wallet, they are not blockchain transactions. Web3.js and wallets allow users to sign plain messages, so in order to minimize the digital identity systems of the users, we have taken this approach. Users will only have to worry for only one digital identity which is the Ethereum identity, both private key and public key.

To send the interactions to the network the API Module is implemented. For this JavaScript fetch will be used along react Hooks. The application can also send private information or private keys from the channels to a local storage module which is only accessible within the client application.

Finally, in the *services* folder, there are different self-developed libraries for allowing the interaction of the App Module with the local storage, the providers, and channel creation and management. To conclude, in the *helpers* folder there are some encryption and decryption functions. In this folder there is also the header implementation for header signatures and content hashing along with header verifications.

7. Testing the decentralized social media platform

During this chapter, the social media application developed in **Chapter 6** will be tested in a realistic environment. For this reason, is important to set up properly the testing environment first.

7.1. Testing Environment

The first step to test the social media will be to deploy the testing scenario of **Figure 63**.

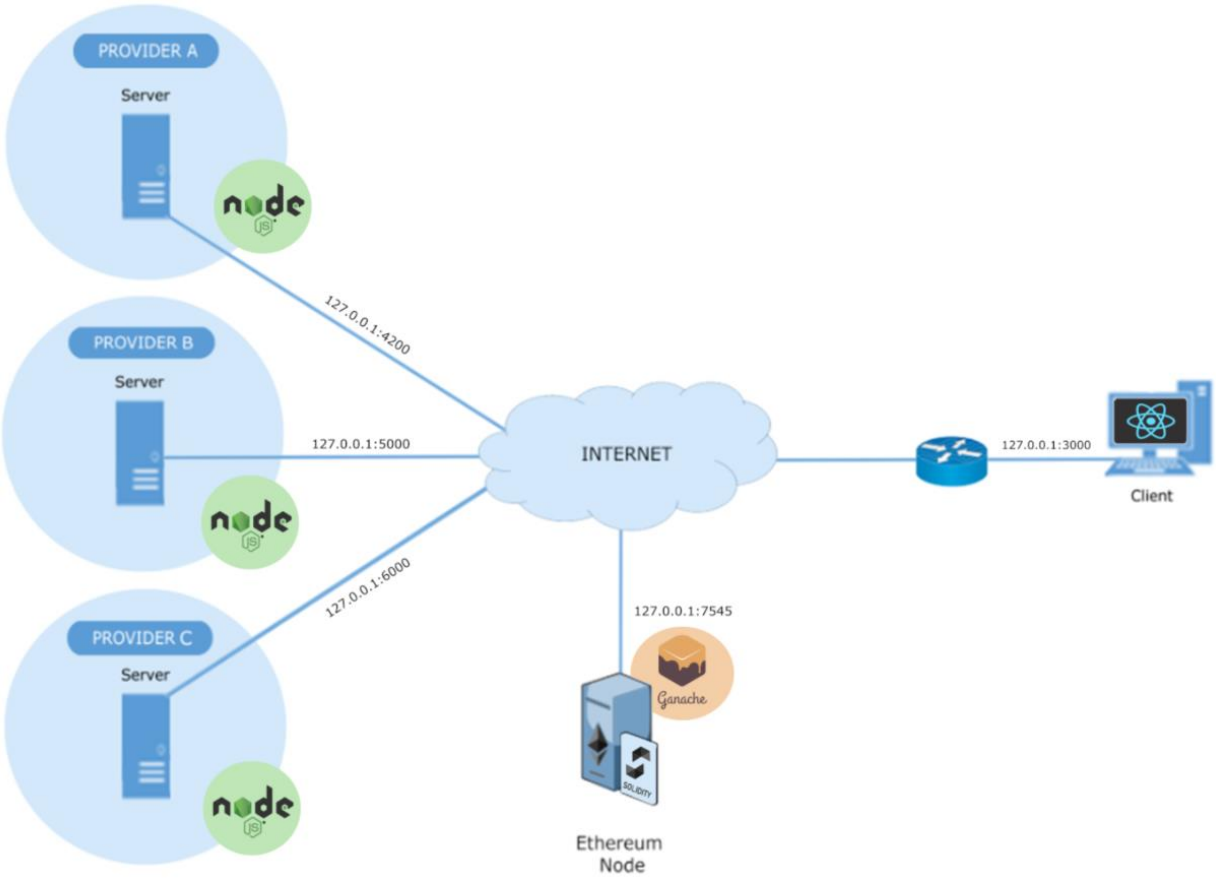

Figure 63 - Testing scenario

First, we will deploy the providers' servers, for this reason we will execute in three different consoles ''*npm start*'' to start the NodeJS servers. It is important to previously configured them to listen in different ports. In this case the ports selected for the servers are 4200, 5000 and 6000.

Then, we will deploy the client interface executing also ''*npm start*'' to initiate the ReactJS application. The client application will listen to port 3000.

Finally, we will need to initialize the blockchain, for this reason we will open Ganache and initialize the blockchain. Once the Ganache node is running, we can deploy the smart contract over the testing blockchain. For this reason, we will first configure Truffle to point to the Ganache node when deploying the smart contract. To do this we will edit *truffle-config.js* file as follows in **Figure 64**.

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*"
    }
  },
  compilers: {
    solc: {
      version: "pragma"
    }
 }
};
```

Figure 64 - Truffle configuration for deploying the smart contract over Ganache blockchain

Then, we will need to compile the smart contract. This will translate the Solidity code into Ethereum Virtual Machine (EVM) code. For this step we will execute in the Truffle folder the ''*npx truffle compile*'' command. Finally, to deploy the contract we will execute ''*npx truffle migrate*''. Once these steps are finished, the smart contract should be deployed and visible through Ganache interface like it is shown in **Figure 65**.

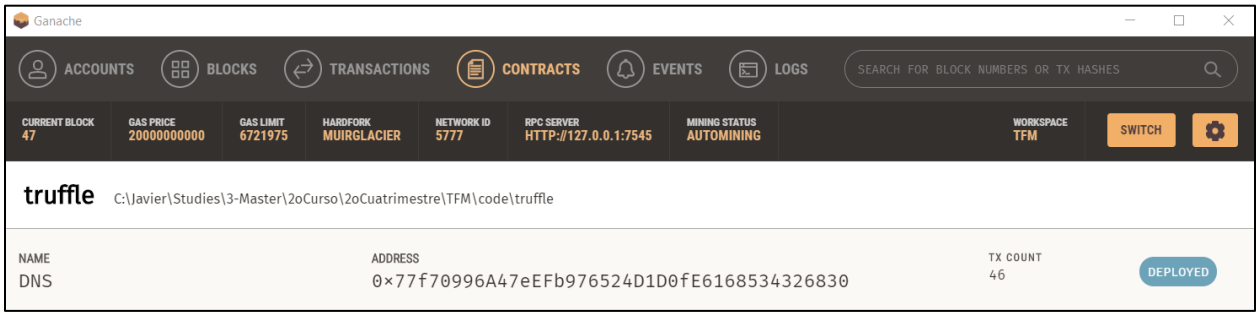

Figure 65 - Smart contract deployed over Ganache blockchain

As you can see in **Figure 65**, the smart contract was deployed successfully at the address *0x77f70996A47eEFb976524D1D0fE6168534326830* from which it is reachable. Finally, once all the parts of the application are deployed, we can start testing the social media application. This is exactly what we are going to do in the next subchapter.

7.2. Testing the decentralized social media platform

During these subchapters we will be testing the application through the client interface. For that we will need to open a web browser targeting ''localhost:3000''. The application will open the home page as it is shown in **Figure 66**.

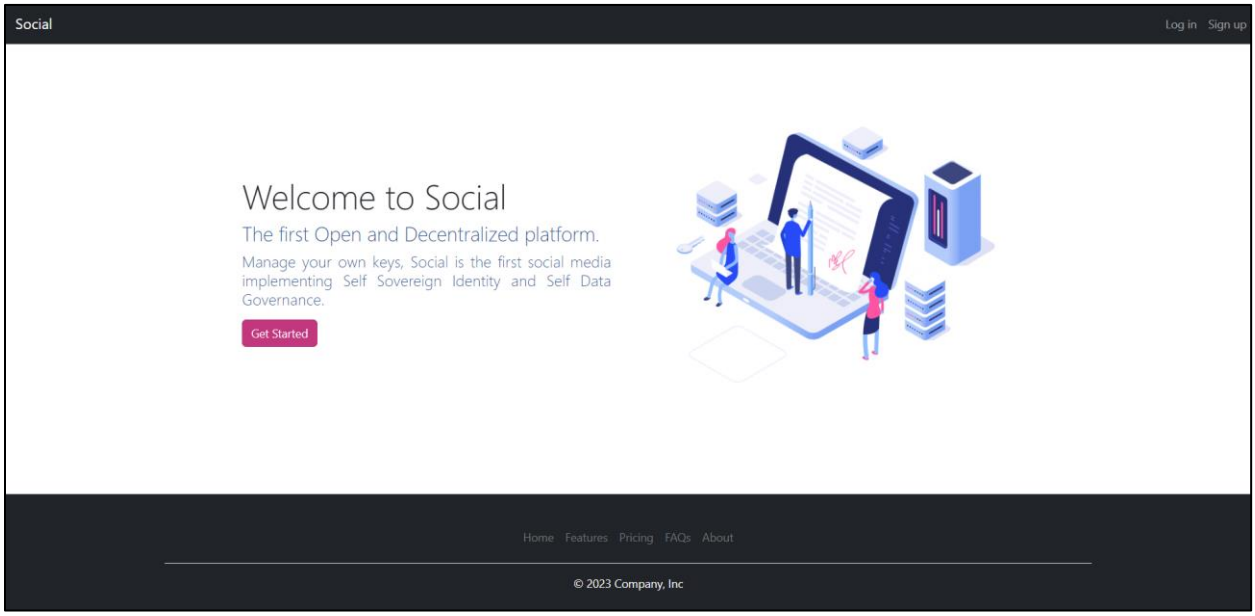

Figure 66 - Home page

First, we will try to log in or sign up. The first time you enter the application, you will probably see a warning like the one shown in **Figure 67**. You will need to install a *Web3* wallet. In this case the application requests you to install MetaMask but any *Web3* wallet will work as well.

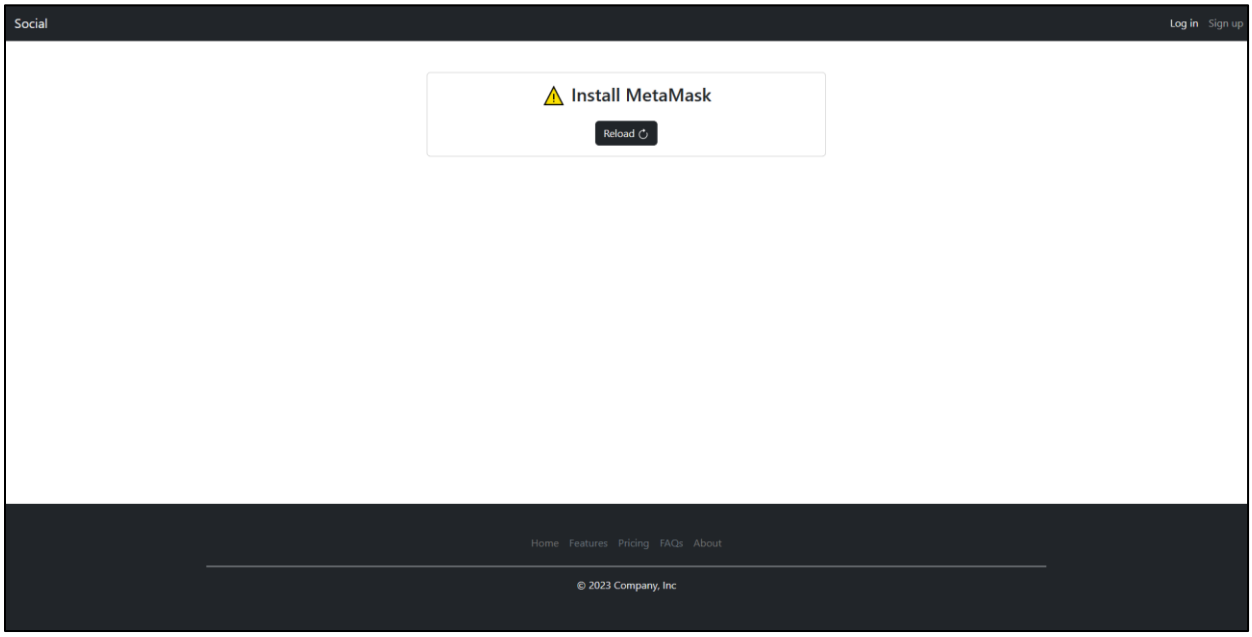

Figure 67 - Install MetaMask warning

Therefore, to start using the application we need to install MetaMask wallet extension in the browser to connect to our blockchain provider. The wallet is needed to interact with the Ethereum blockchain.

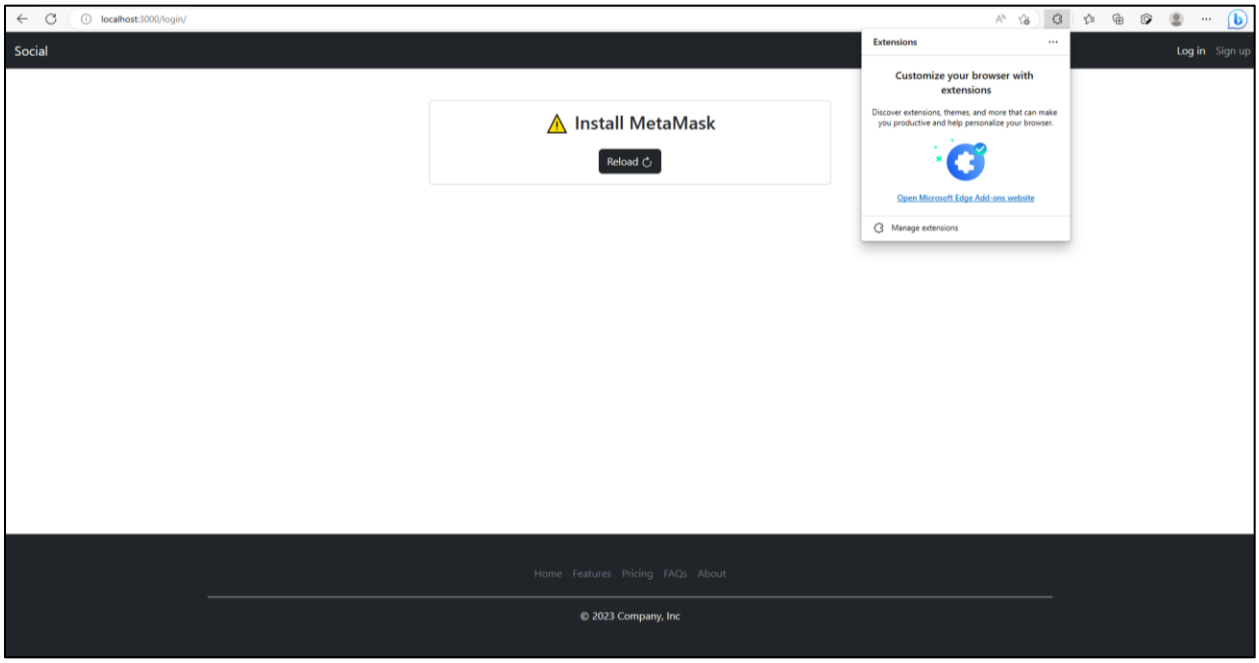

Figure 68 – Install MetaMask warning and extensions

You can find MetaMask in the Web Extensions Store of your web browser. In the case of any Chromium based browser the extension is the following represented in **Figure 69**.

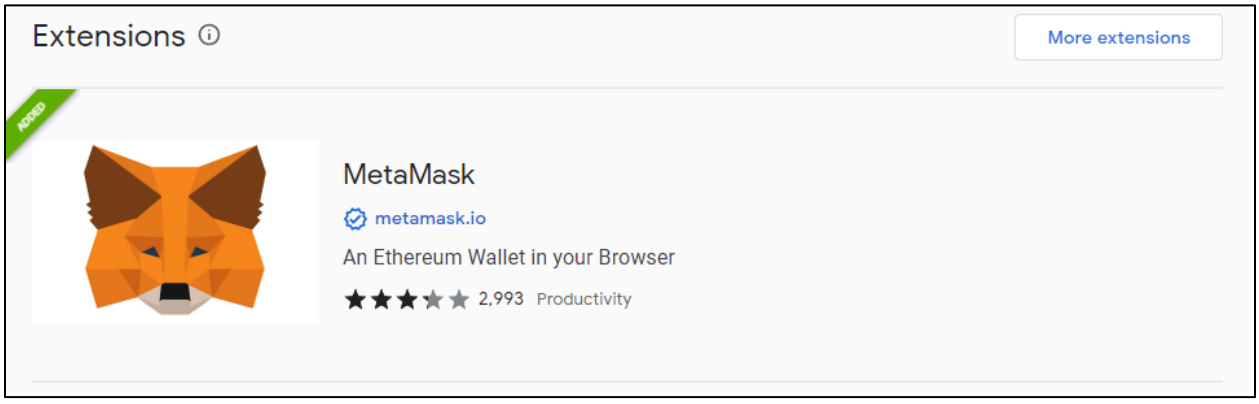

Figure 69 - MetaMask browser extension

Once we have MetaMask installed, we will need to connect the wallet to the right network. As you can see in the screenshot from **Figure 70**, we will need to connect to Ganache network, which is the one we are using in our testing scenario. In production environment, the smart contract will be directly deployed over the Ethereum Mainnet (0x1), but in this case Ganache network (0x539) will be used.

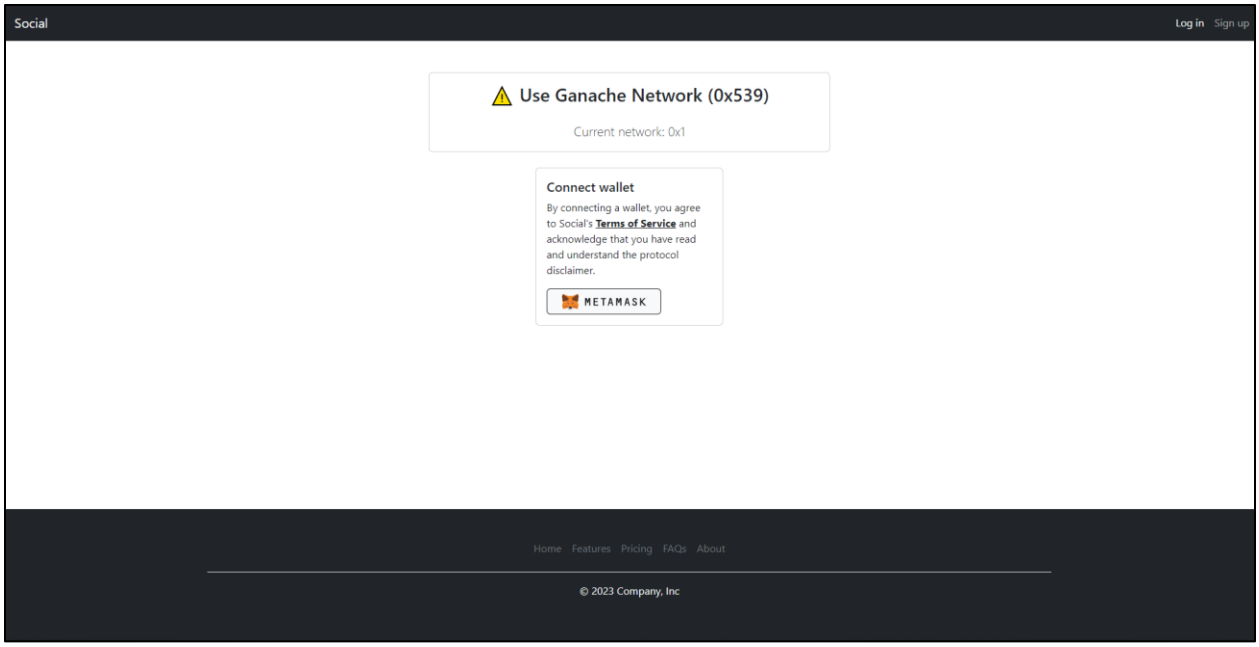

Figure 70 - Blockchain Network warning

For that reason, we will open the MetaMask extension and change the network connection. As you can see in **Figure 71**, by default the Ethereum Mainnet (0x1) is selected.

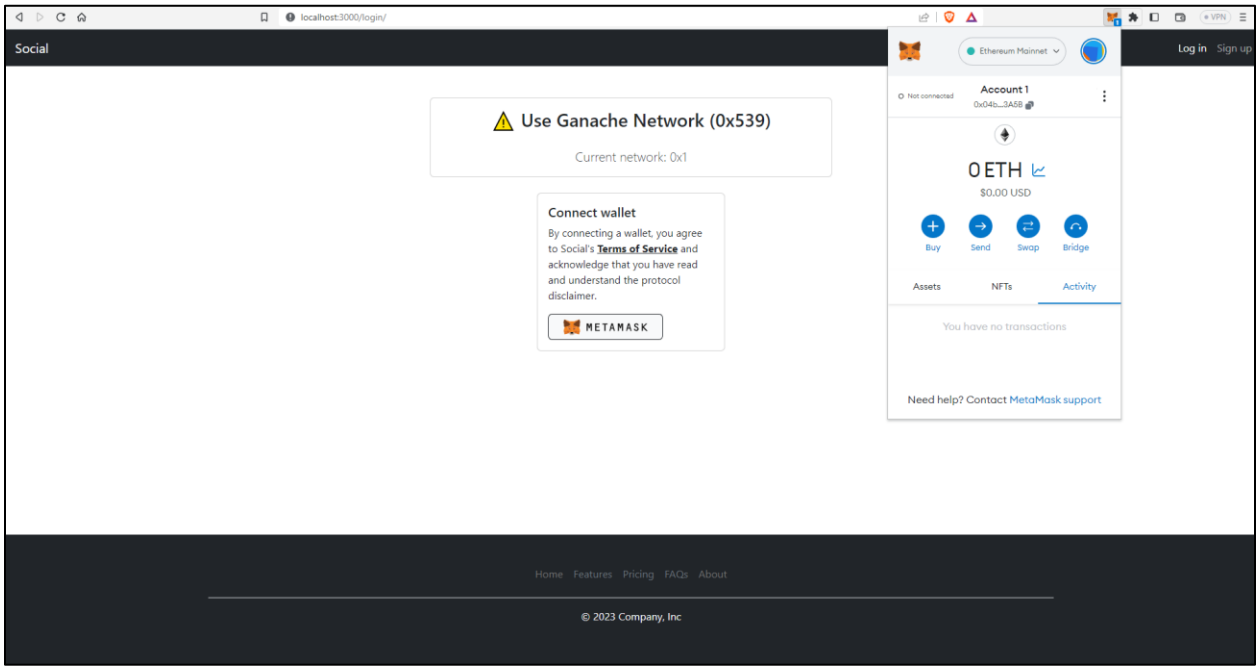

Figure 71 - Open MetaMask for the first time

To connect to Ganache network, first we will need to configure the connection to Ganache. To do this click on the account profile picture located in the top right of the screen of **Figure 72**, and then navigate to *Settings > Networks*.

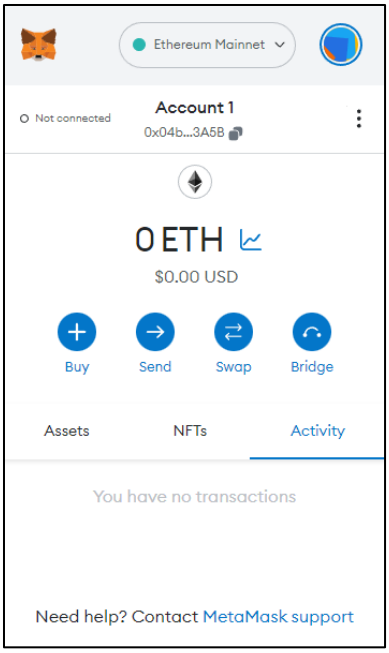

Figure 72 - MetaMask wallet interface

Once there, select "*Add a network manually*" to add Ganache network like it is done in **Figure 73**.

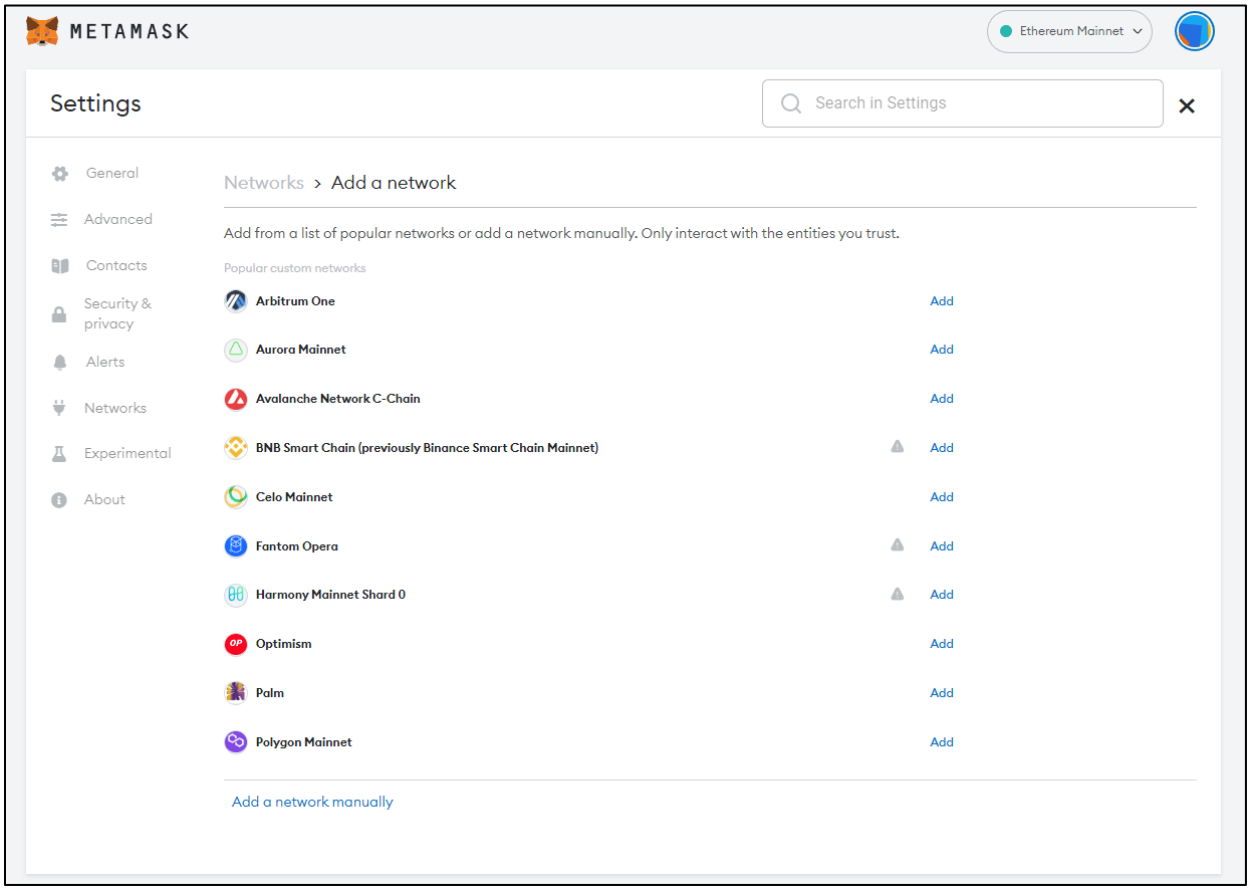

Figure 73 - Add a network manually in MetaMask

Now, we will have to introduce the Ganache node configuration and network details in the screen depicted in **Figure 74**.

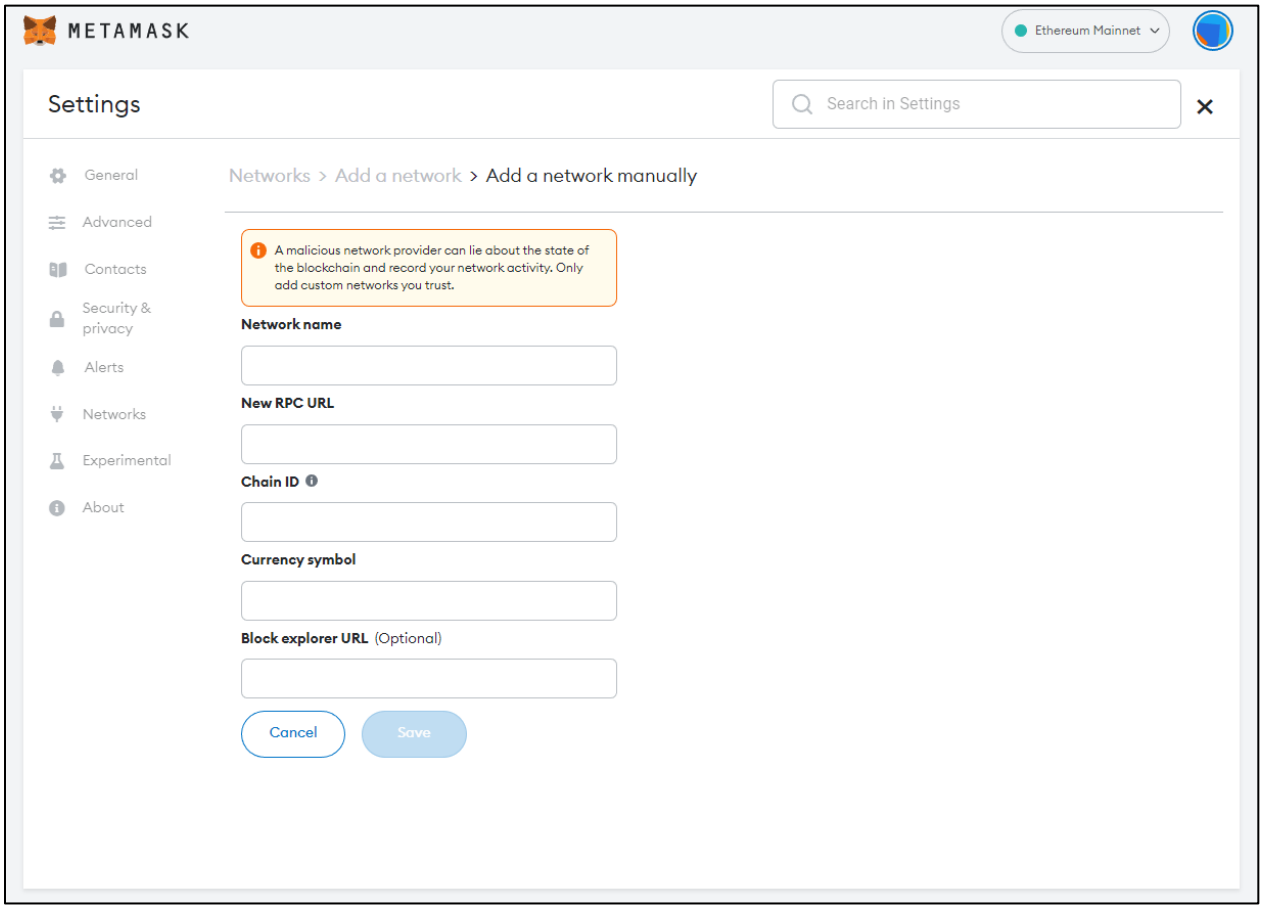

Figure 74 - Add a network manually in MetaMask II

Once you have introduced the Ganache node and network details, the new added network should look like the added network in **Figure 75**. All these details are provided by the Ganache node interface we have previously launched.

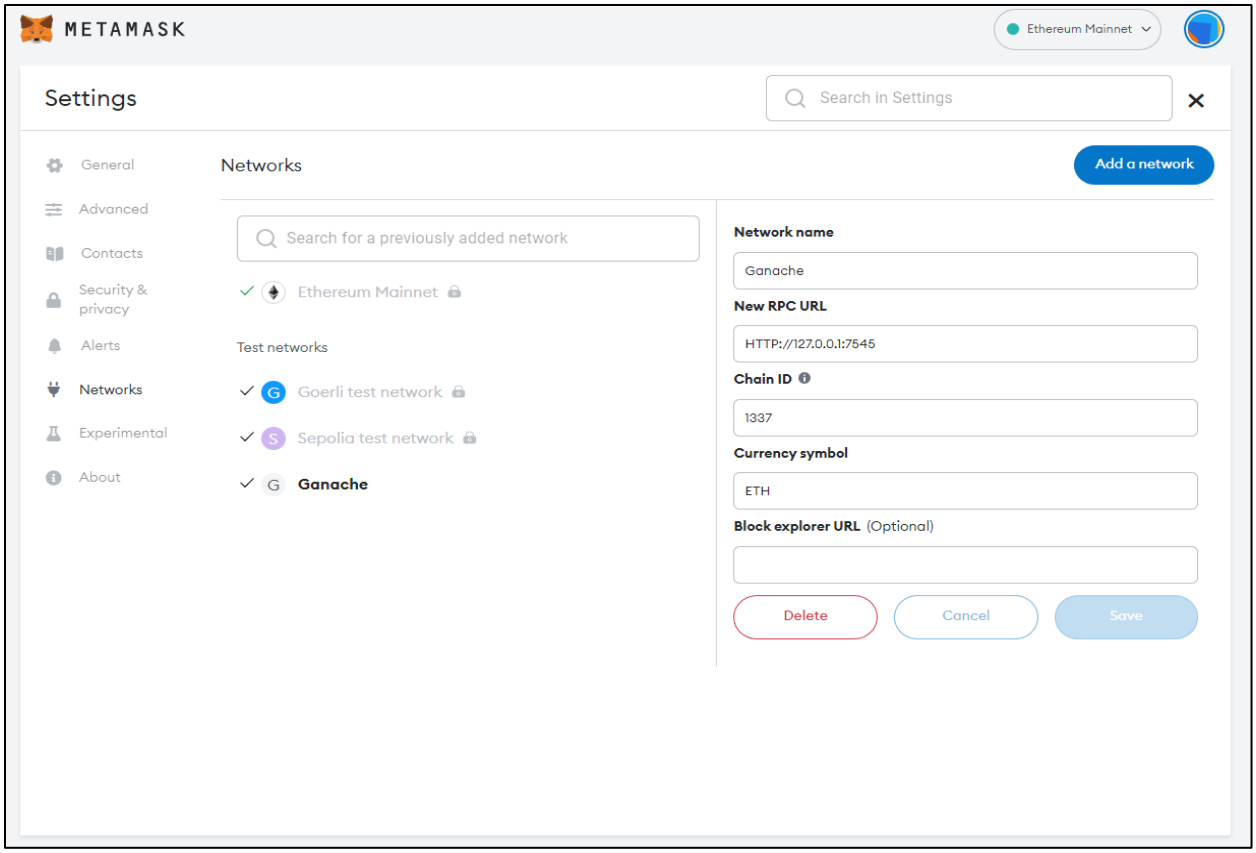

Figure 75 - Ganache Network details

Then, after saving the network configuration, we select Ganache in the network dropdown to connect MetaMask to the React decentralized application or DApp.

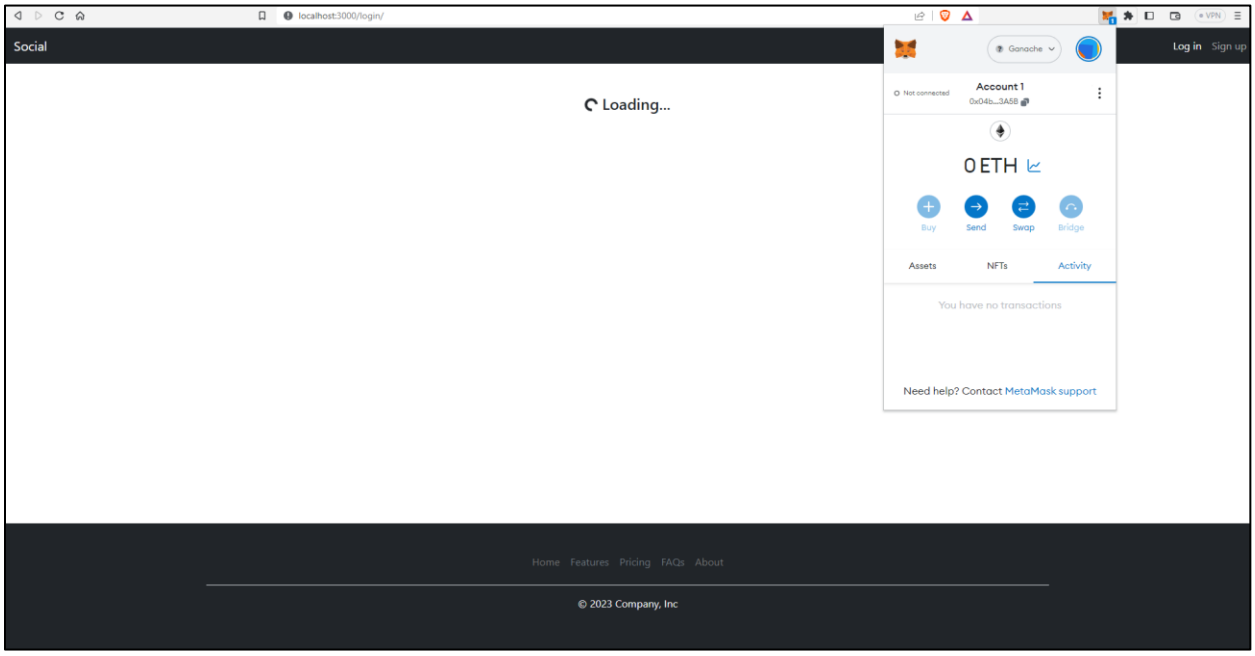

Figure 76 - Loading page

In some cases, you may be prompted with the following warning, in that case look it there is already another MetaMask window opened. This could happen if it is the first time MetaMask reaches this *Web3* site. If this is the case connect your wallet to the DApp from that window.

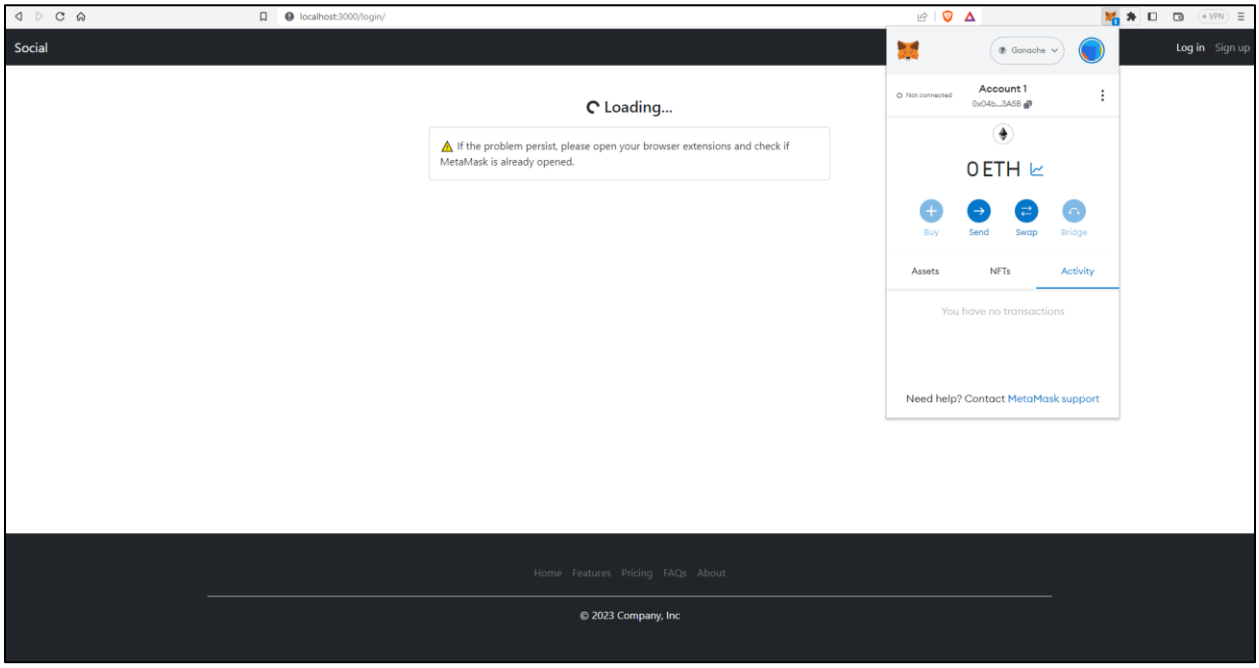

Figure 77 - MetaMask warning if there is already a MetaMask interface opened

If it is not the case and Ganache network is selected, click in the Connect wallet button from DApp to open MetaMask. This button is shown in **Figure 78**.

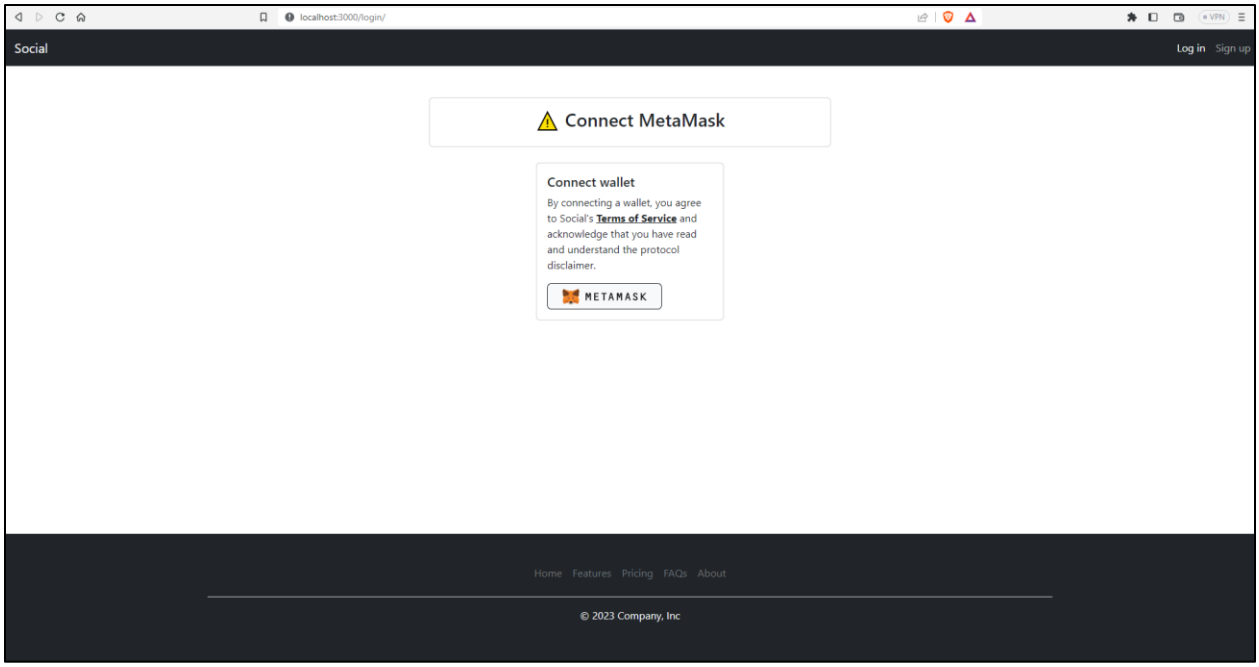

Figure 78 - Connect to MetaMask from the DApp

A MetaMask window will be opened when clicked like **Figure 79**.

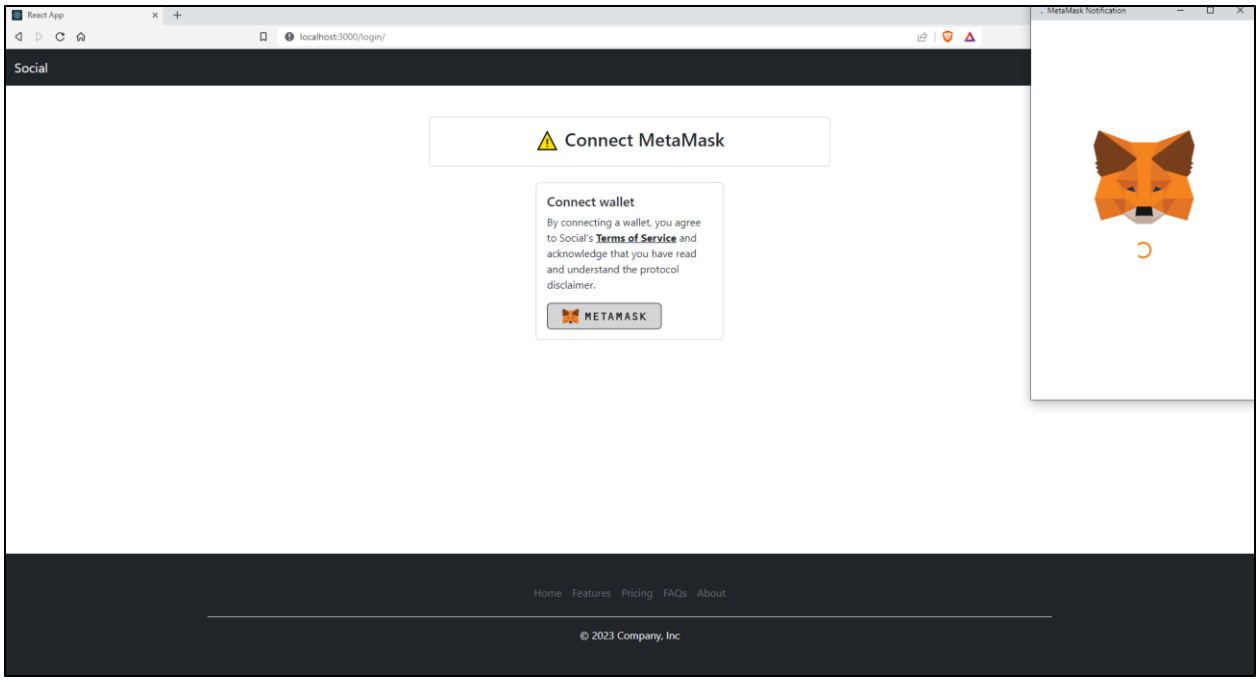

Figure 79 - Opening MetaMask from the DApp

Then, MetaMask will ask you to select the accounts you want to connect to the DApp. Select the ones you want to use like it is done in **Figure 80**.

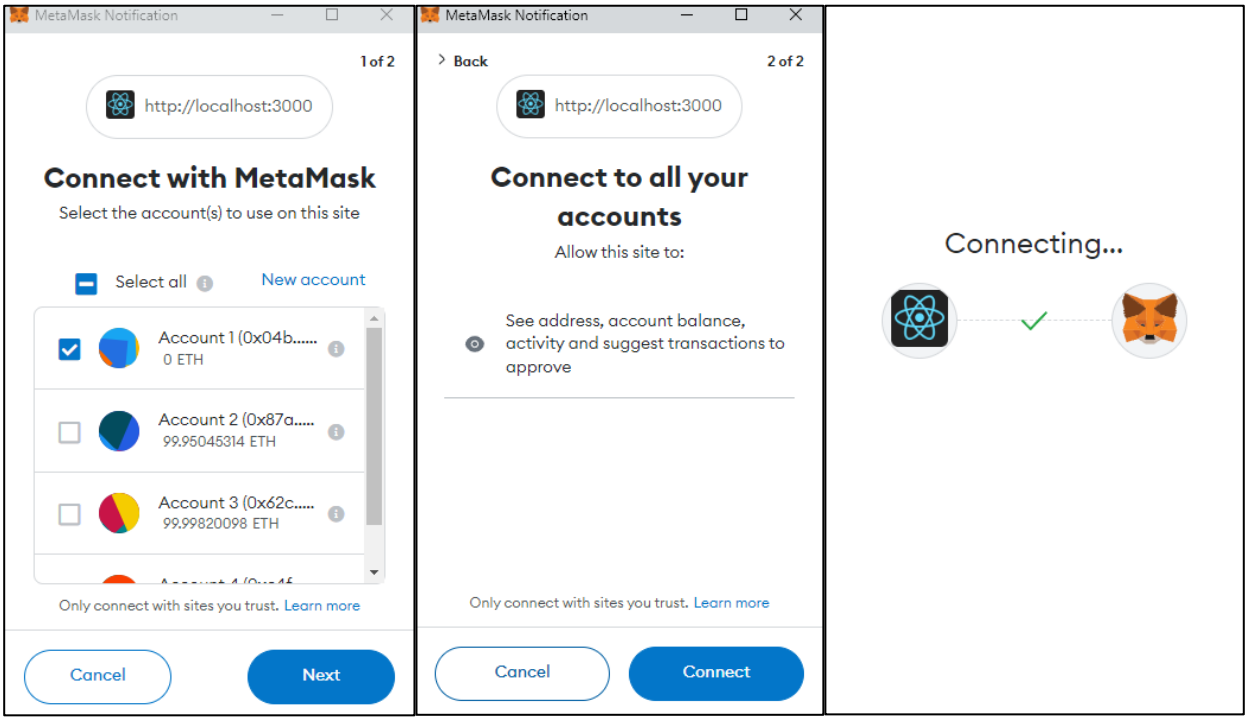

Figure 80 - Connect an account to the DApp from MetaMask

Once connected, we can check if the MetaMask accounts are connected to the DApp by checking a MetaMask account and seeing if the green ''Connected'' label appears in that account. This is shown in **Figure 81**.

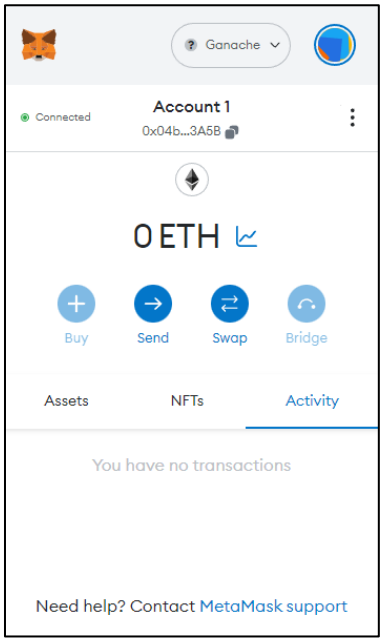

Figure 81 - How to know if MetaMask is connected to the DApp

Once connected, we are now ready to create an account or login into an existing one if we already have an account created. We should see the screen shown in **Figure 82**, with a label with the connected account address and a signup button.

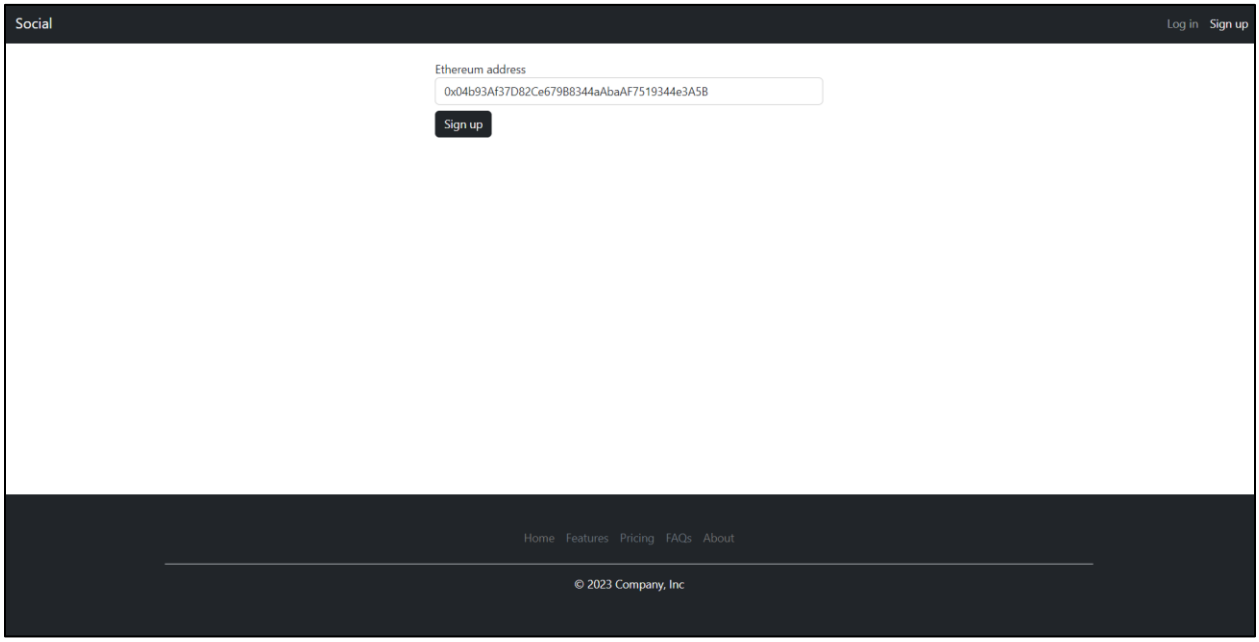

Figure 82 - DApp Sign up

The first step when creating the account will be to select the server providers in the screen shown in **Figure 83**. These providers are where your information will be stored and accessed. We will have to select them manually but in future releases, the providers will be able to publish their deals in the blockchain as we have seen in **Chapter 5**. This will allow users to search for a specific provider by filtering by price or deal types, or even discover providers by selecting them through a visual interface within the DApp, rather than manually prompting the provider's IP address, which is tedious.

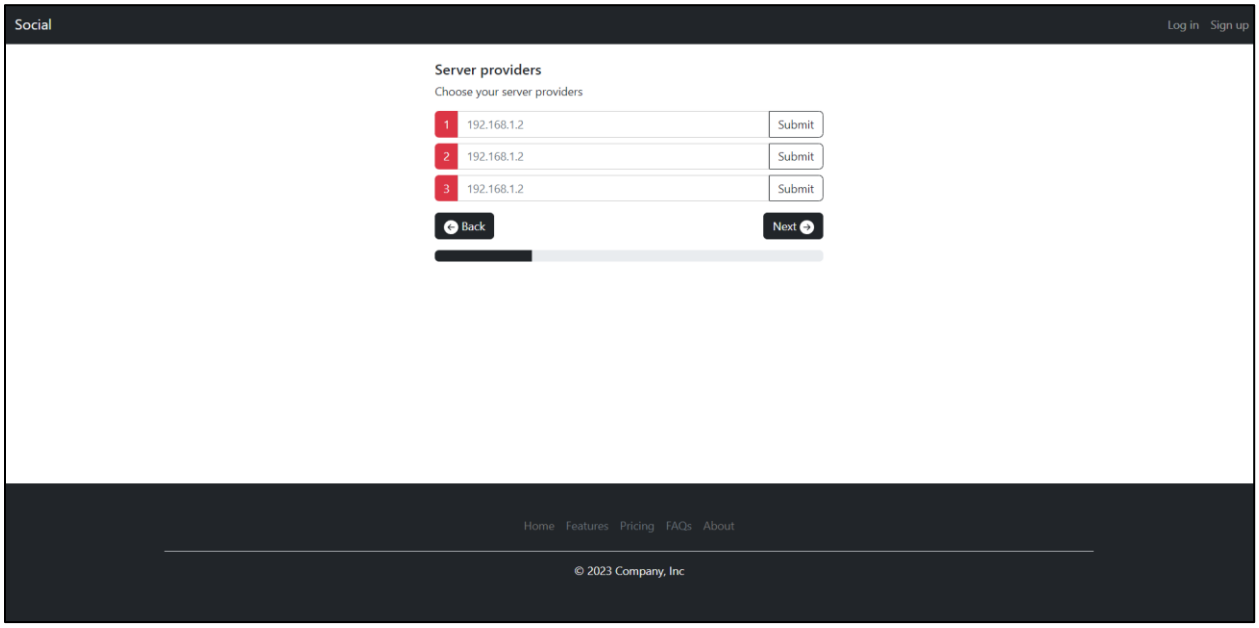

Figure 83 - DApp Signup: Selecting the providers

We will enter localhost:4200 as the first provider. This IP address corresponds to *Provider-A* address. A MetaMask window will be prompted to execute a blockchain transaction like it is shown in **Figure 84**.

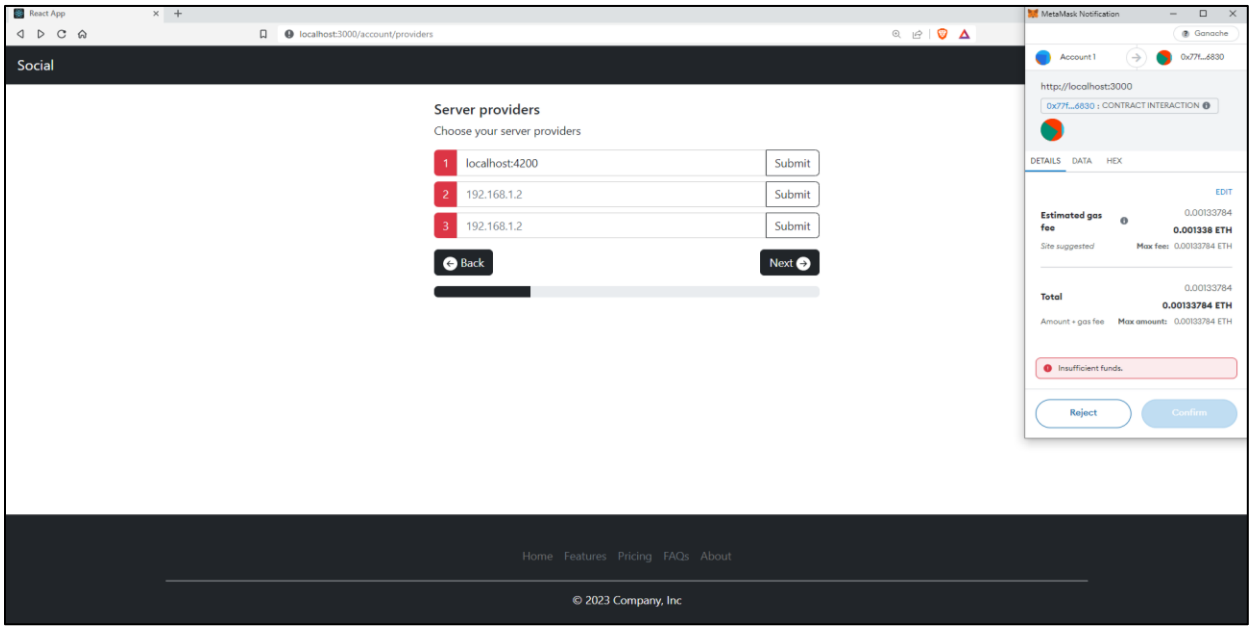

Figure 84 - DApp Signup: Selecting the providers II

Since the operation of selecting a provider is part of the blockchain DNS, it implies storing and modifying the blockchain state, this operation or transaction (Tx) will have some fees associated. As you can see from **Figure 85**, the fees are 0.000805 ETH, which is about $1.5 at the current price of $1.8K per ETH. There are two types of fees, one is gas fee which is the cost of executing code and storing information in the blockchain and the network fee which will depend on the transaction demand since there is a transaction amount limit per block. These fees will depend on the complexity of the executed code, the amount of information to store in the ledger, the network usage and obviously ETH price.

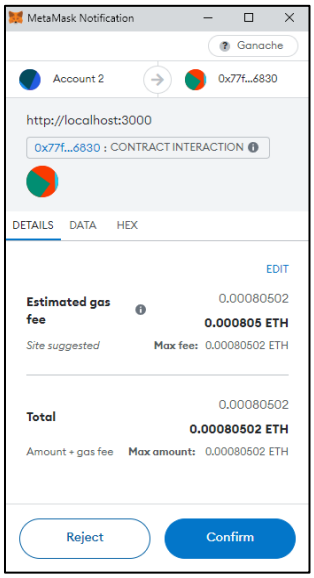

Figure 85 - DNS Blockchain transaction

As you can see, it cost only $1.5 approximately to save a provider in the blockchain DNS. Once the transaction is executed the provider's IP address is stored and saved on the blockchain. It is important to note that only the owner of the account can edit his own storage, which is probably the most important property of blockchains, which is ownership.

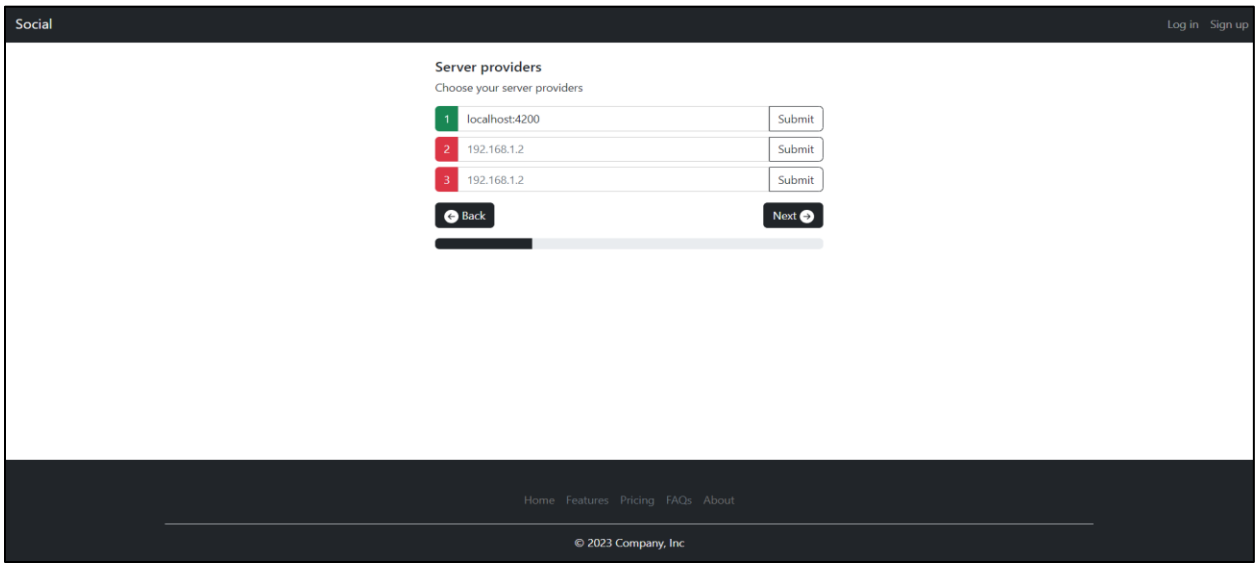

Figure 86 - DApp Signup: Selecting the providers III

As shown in **Figure 86**, if the provider is stored in the blockchain, it appears in green color, otherwise in red. Then in **Figure 87**, the rest of providers are selected, with a total accumulated cost of $4.5.

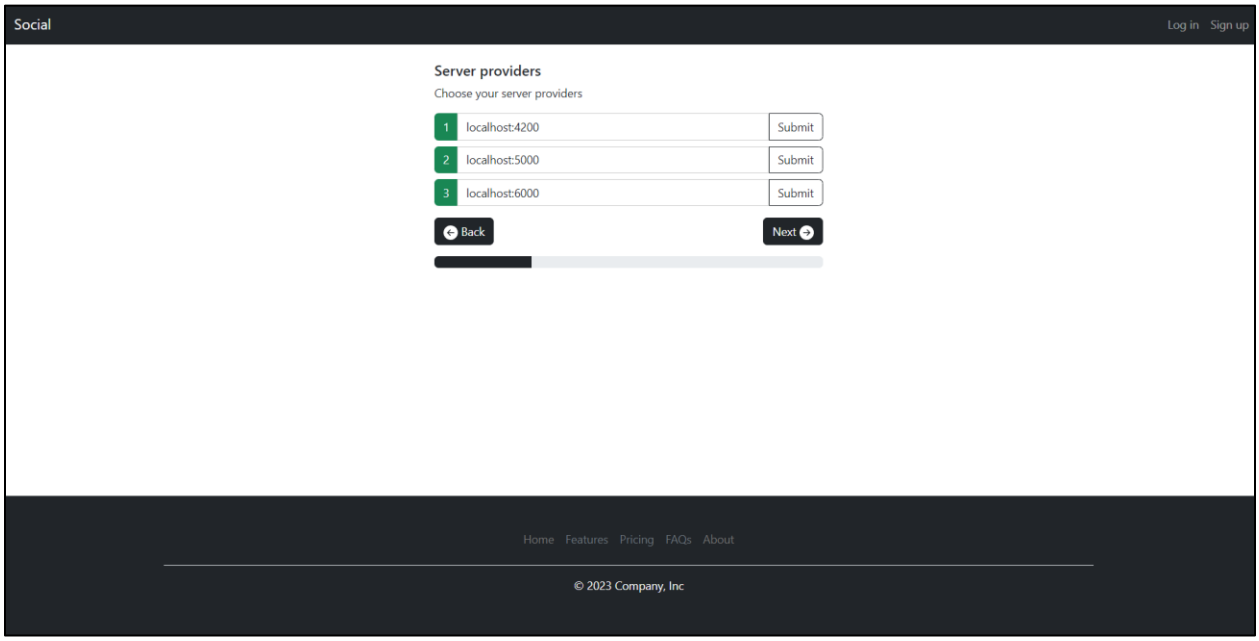

Figure 87 - DApp Signup: Selecting the providers IV

Afterwards, the username must be selected. This operation also implies blockchain interaction. As you can see in **Figure 88**, this name is already in use by another account, so it is not available to select it. We can query the blockchain for free, since we are not changing the ledger state, in this way we avoid making a useless transaction which will not let us choose the username and will make us spend some ETH fees on it.

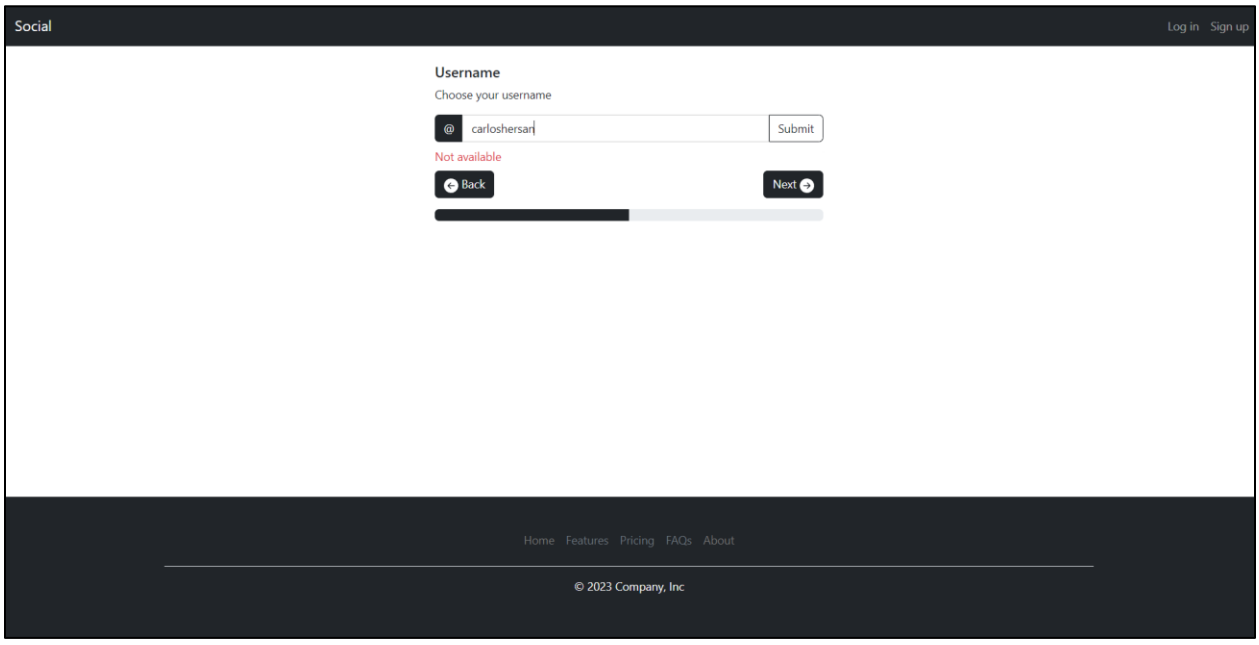

Figure 88 - DApp Signup: Selecting the username

In **Figure 89**, we select an available username. As before MetaMask will ask you to allow the transaction. The fees to reserve the username in the ledger is also $1.5, making an accumulated total of $6 for the entire DNS service. If we compare the blockchain DNS with the traditional DNS, we will see that is cheaper, more decentralized and even without expiration date if it is not implemented in the smart contract.

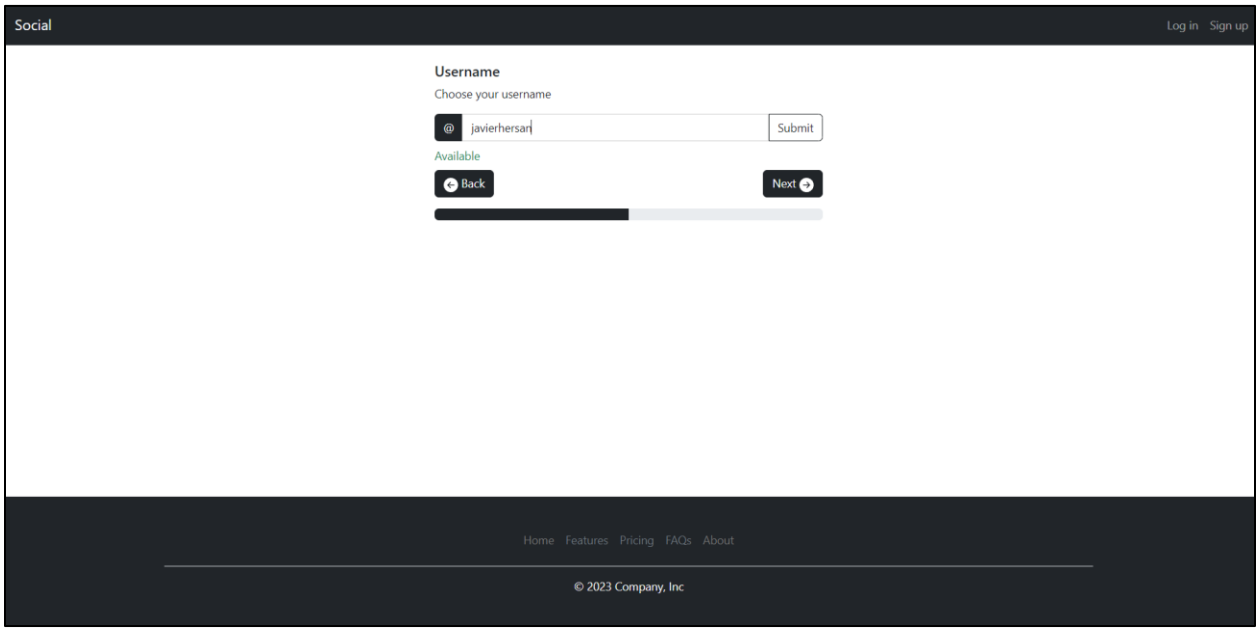

Figure 89 - DApp Signup: Selecting the username II

In case, you have reserved this username before, MetaMask will not be opened to avoid a useless transaction. Once the username is selected the "@" label will be in green color as shown in **Figure 90**.

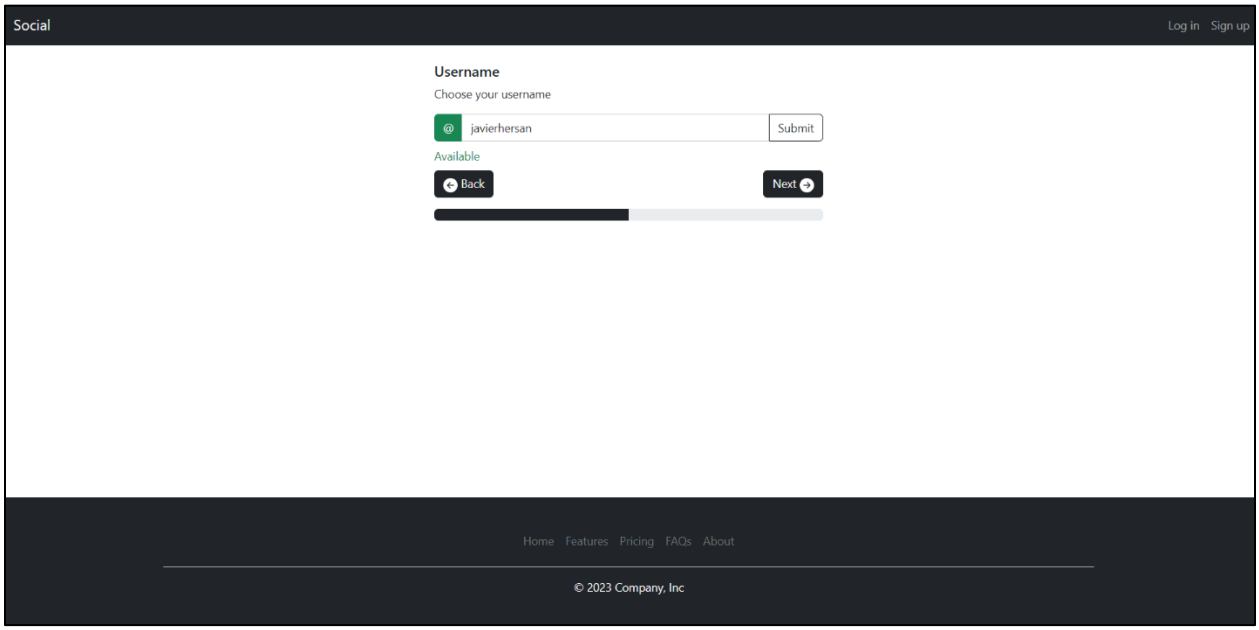

Figure 90 - DApp Signup: Selecting the username III

Then the personal data is prompted. This is illustrated in **Figure 91**.

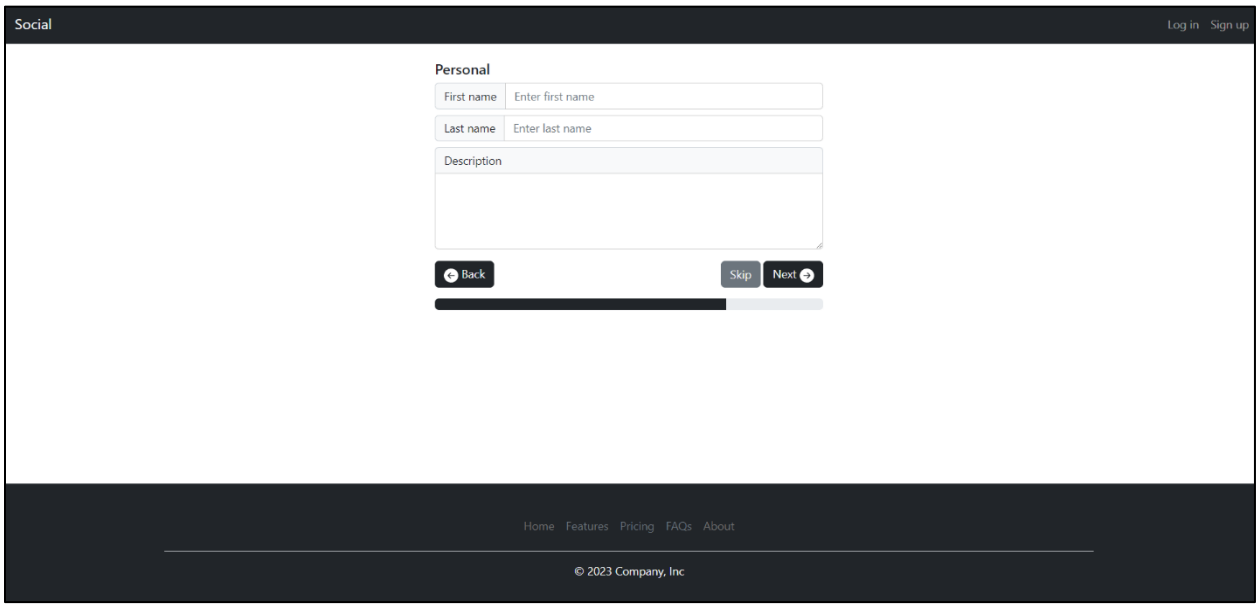

Figure 91 - DApp Signup: Personal data

Once we enter the personal data like in **Figure 92**, we can click *Next* to enter the application. We can also *Skip* this configuration since it is optional. This is the first operation that is saved in the providers. This information will be sent to the servers and the providers will save the personal information along the *post header*. As you remember the *post header* ensures content authenticity and that the information belongs to the profile owner, in order to avoid impersonation. Since all the interactions within the platform are signed, other users cannot authenticate with our account. The platform interactions and the related headers are signed in the background by MetaMask, since we have given permissions to do this previously.

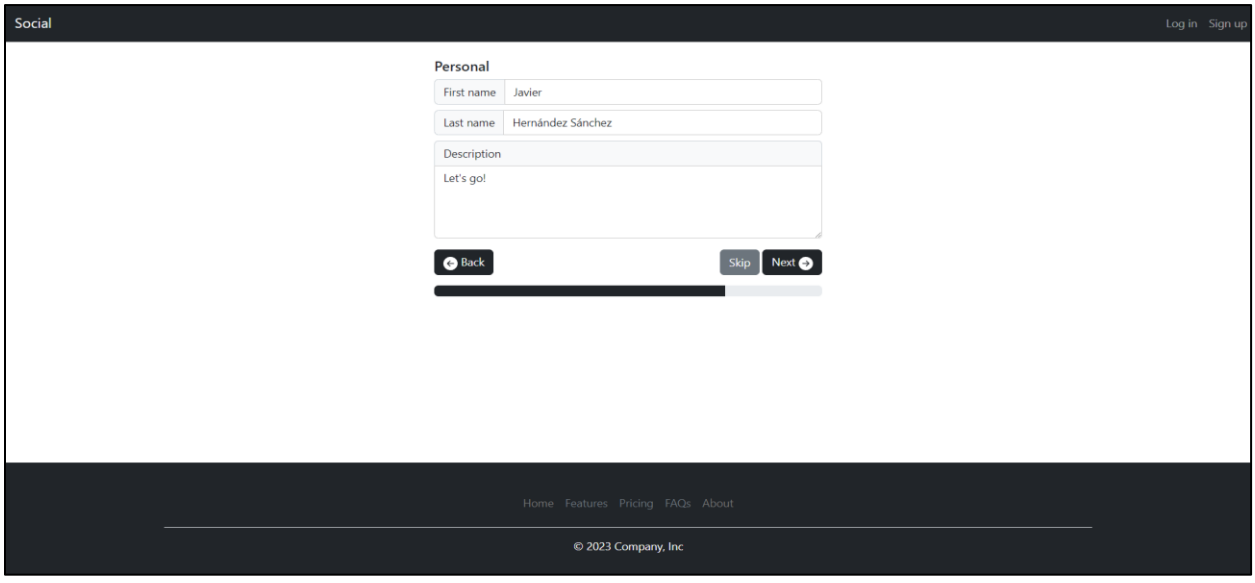

Figure 92 - DApp Signup: Personal data II

Now, that the account is already created we can enter into the DApp by clicking *Next* button, but first we will explore how the login works. Once MetaMask is connected to the site, the login page will request only the account username. This is illustrated in **Figure 93**. This seems like a strange login, since there is no text input for the password. Remember that the *Web3* wallet also works as a password manager and that every interaction we do in the DApp will be signed in the background by MetaMask, so there is no point in logging in as another user, since no interaction will be valid without the user's signature.

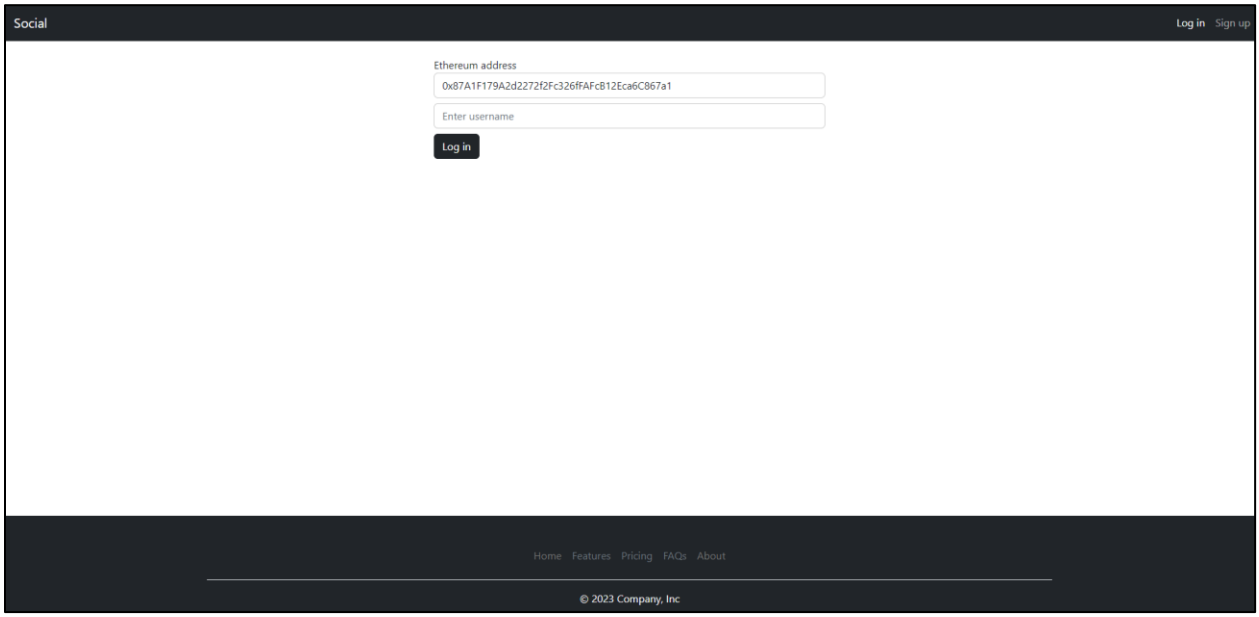

Figure 93 - DApp Login

Once we enter the social media platform either through the login or the signup page, the first page that is displayed is the *Feed*. The Feed is shown in **Figure 94**. This page displays recommended content. This content can be content from the accounts we follow, recommendations from your providers and even advertisements in case you have signed deal which includes them.

In this case, in **Figure 94**, only two posts are shown since the social media is not highly populated with content yet. The first publication is a post from an account I already follow, and the second one is an advertisement or recommended publication from one of my providers.

For the Feed page, a new API endpoint, **http://<server-provider-ip>/<account>/feed**, will be needed. Additional endpoints are required in the application, see **Appendix B** for more information about the different API endpoints.

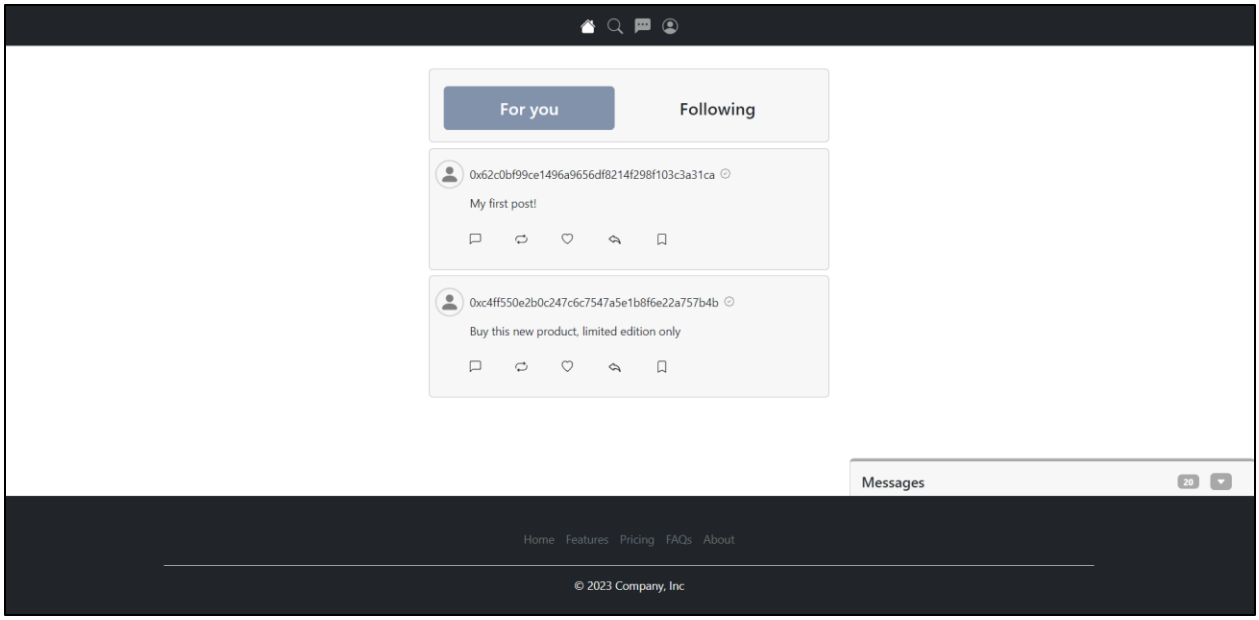

Figure 94 - DApp Feed

We can visit the user's profile by clicking on its profile picture or its Ethereum address. As you can see in **Figure 95**, we are already following Carlos, and we can unfollow him by clicking on the unfollow button. We can also go back to the *Feed* by clicking the *Back* button. New endpoints will be needed to manage the follower system, remember to visit **Appendix B** for more information on these endpoints.

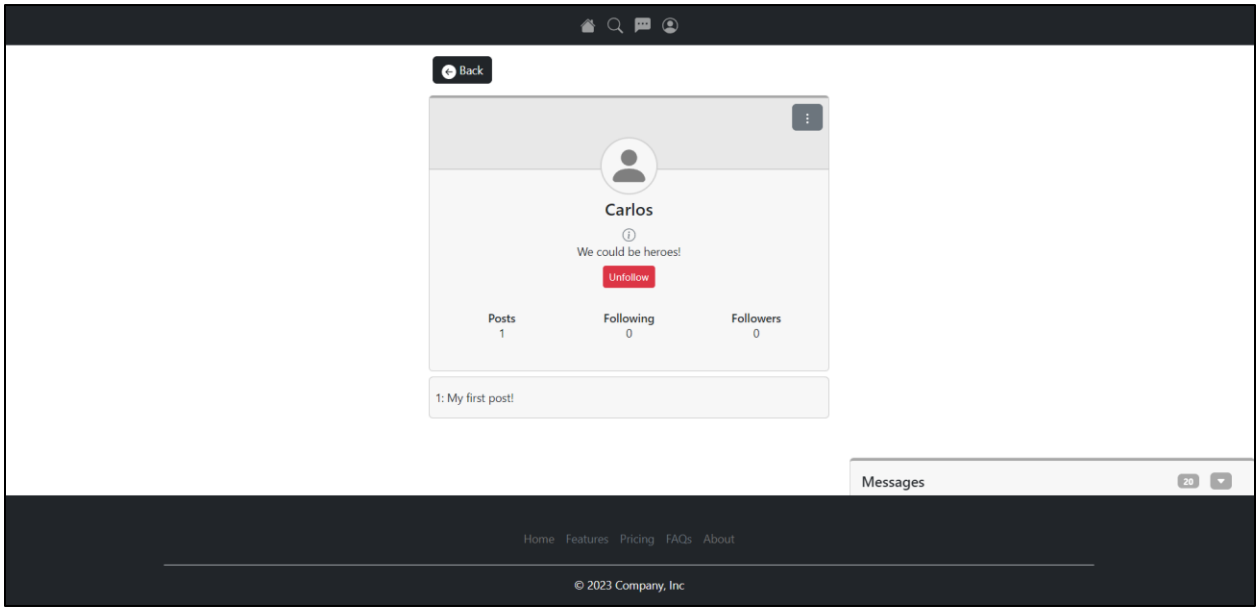

Figure 95 - DApp User Profile

Now we are going to visit the advertiser account, but instead of accessing it directly by the Feed page, we are going to search it. We can search accounts in the *Search* page. This page is described in **Figure 96**. We can search by Ethereum address or by username which is the user-friendly option for most users.

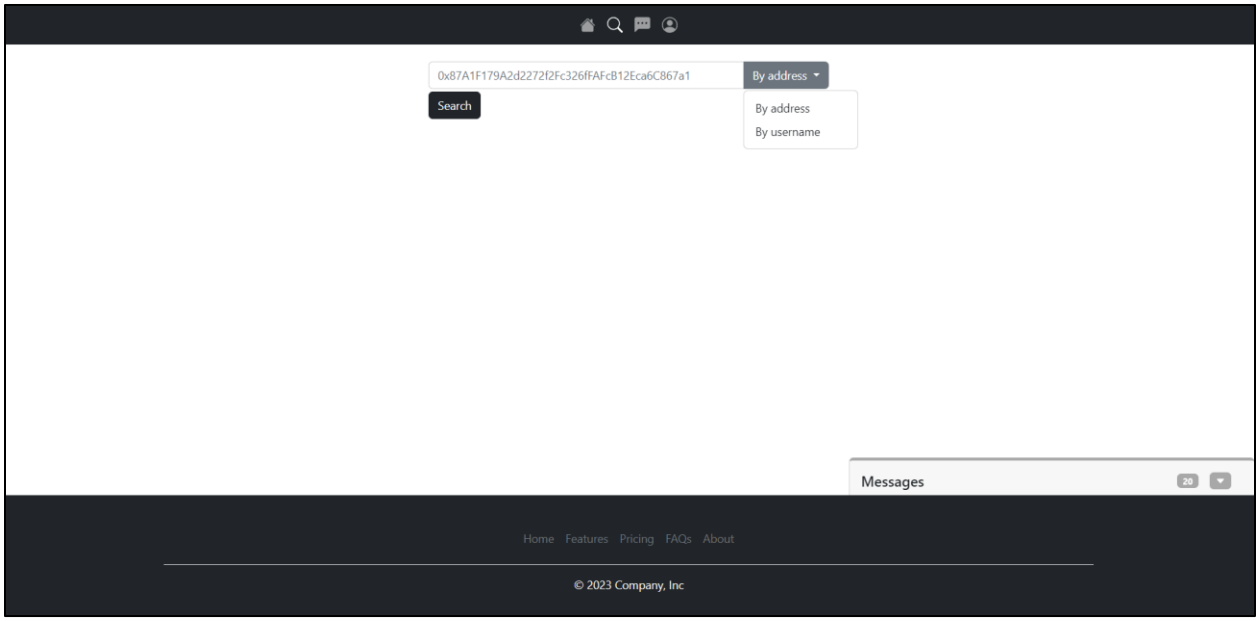

Figure 96 - DApp Search Page

If we introduce the Ethereum address of the advertiser, we can visit its profile as it is shown in **Figure 97**. We can follow the account by clicking on the blue button of the profile if we want to.

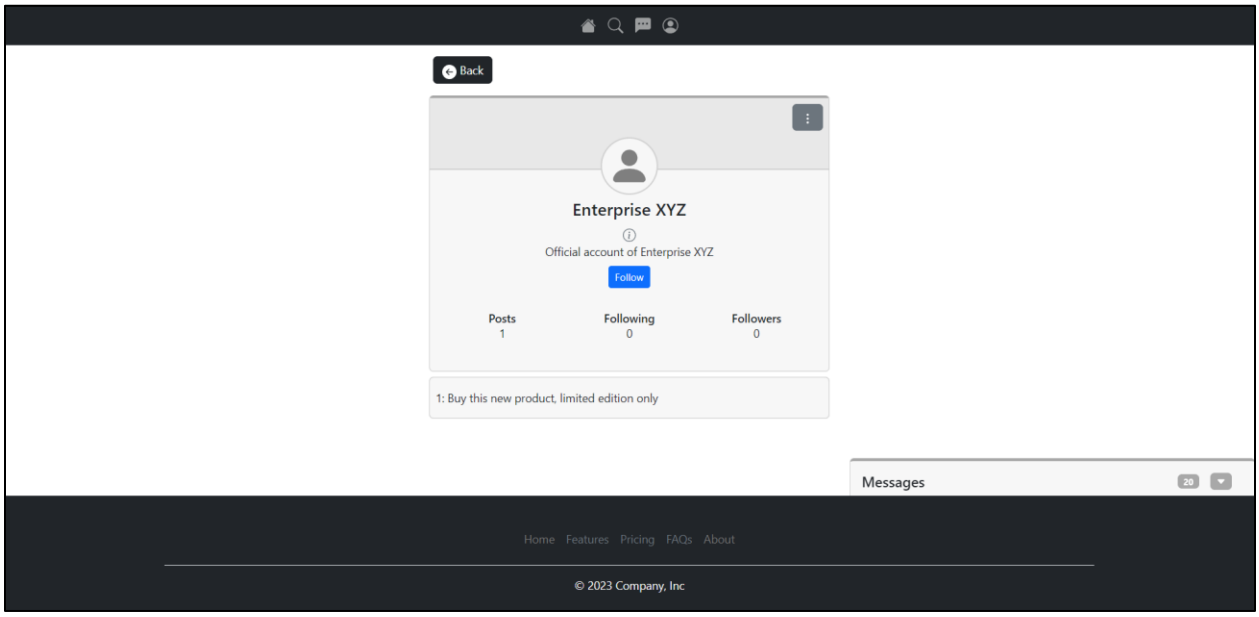

Figure 97 - DApp User Profile II

The next screen that we will explore is the *Profile* page. In this screen we can see our profile information and publications. This page is depicted in the screenshot of **Figure 98**. As you remember from **Figure 92**, we entered the personal information and sent it to our providers, so this is why this page displays our first name, last name and description. We can also upload new posts using the text box at the bottom of the profile.

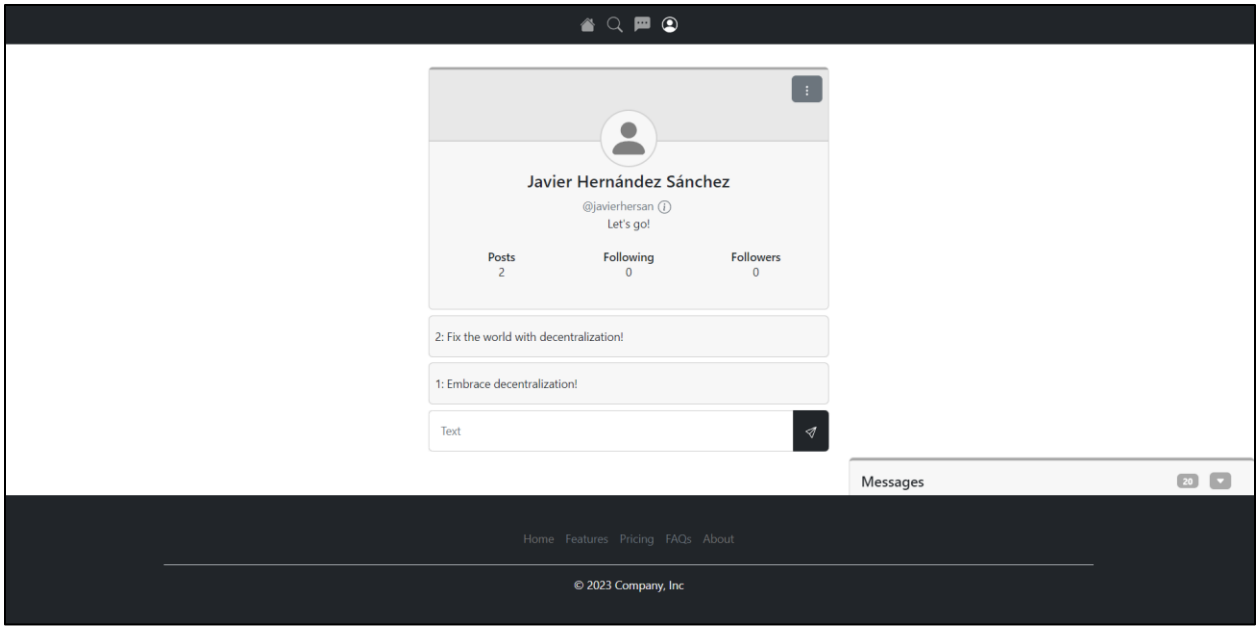

Figure 98 - DApp Profile

If we click on the *Info* icon, we can see more details about the account like the Ethereum address and the providers of the account. All the personal information and the posts are loaded by requesting it to the account providers. Then the information retrieved by each provider is compared and checked through the header subsystem, in order to display the latest correct content as we have seen in chapter four. All these processes are done in the application background. In the event of a provider outage or censorship, our account will still be operational as long a provider is up and acting honestly.

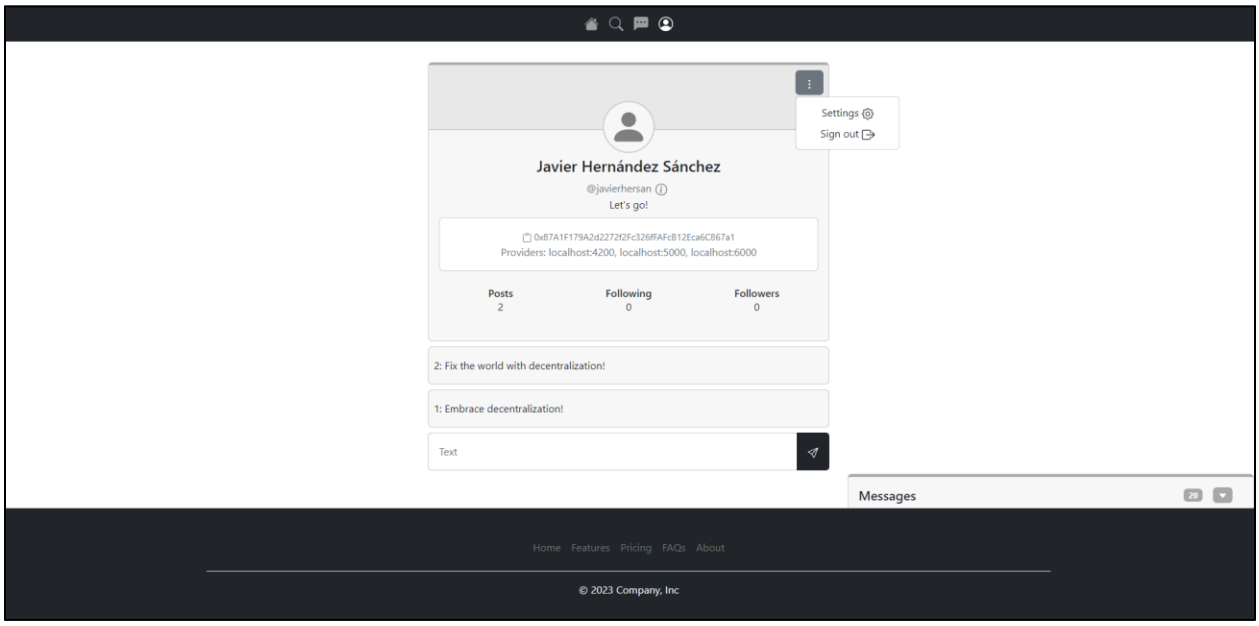

Figure 99 - DApp Profile II

We can also sign out or check the account settings, this is what is shown in **Figure 99**. In the settings page we can change our personal information, privacy and security settings, and even stats. In the *Stats* page available in **Figure 100**, we can review the performance of our providers. For example, the failed request, with this information we can assess if there is a provider that is not offering a good service and change it if it does not meet the standards of the agreement or deal. Other metrics can be further implemented such us censorship metrics which will be really useful to evaluate a possible change of provider. However, only the failed request metrics have been developed as an example. Here, the failed requests are high because we are in a testing scenario and sometimes a provider is turned off, thus bringing failed requests associated with.

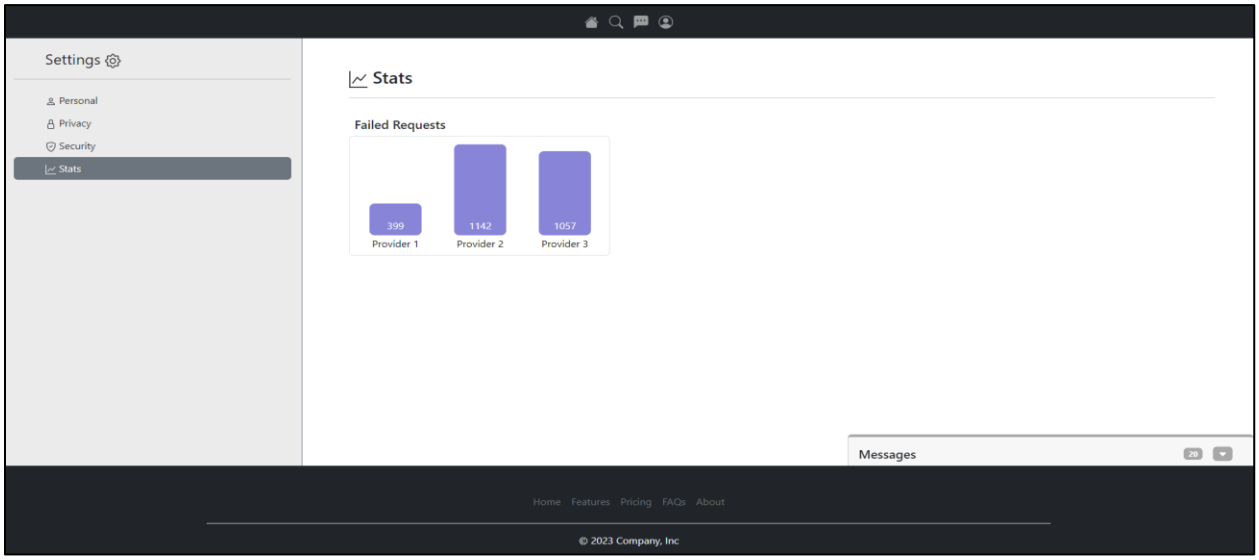

Figure 100 - DApp Account Settings: Stats

In the *Security* page, we can enable content filters. This feature available in **Figure 101** will allow users to filter harmful or offensive content. For more information about the design of this filter visit **Appendix C**.

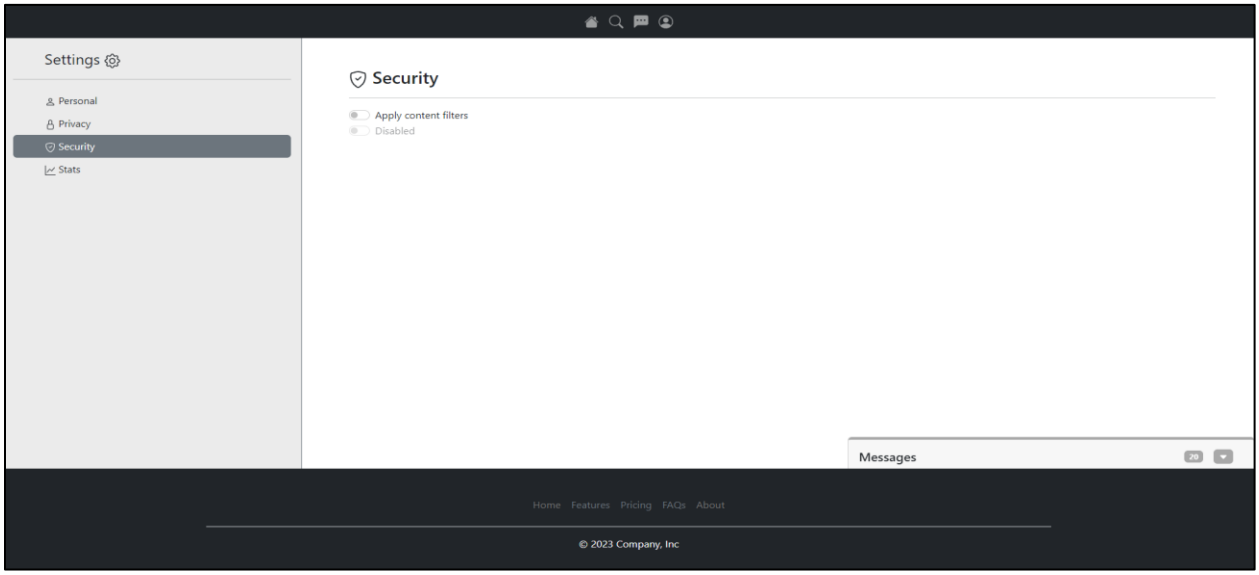

Figure 101 - DApp Account Settings: Security

We can apply the content filters by right clicking in the filter selector of **Figure 102**.

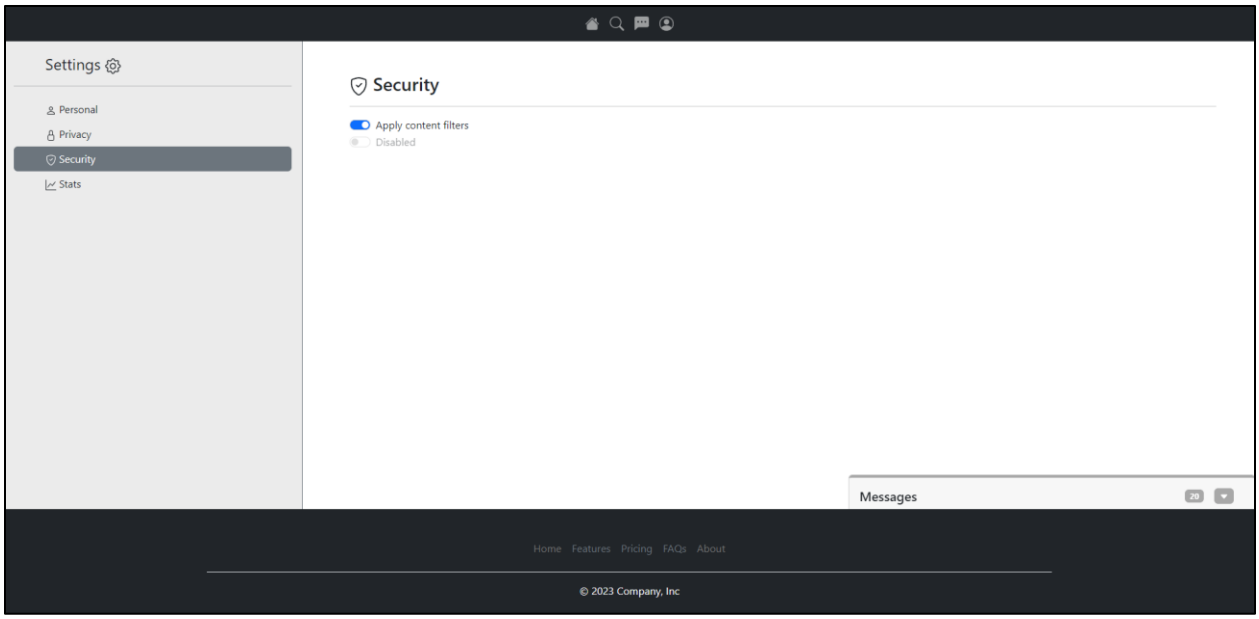

Figure 102 - DApp Account Settings: Security

Now, if we return back to the profile, we see a publication is hidden in **Figure 103**. This hidden post is a new publication that I have published before which contains harmful content. In this case the filter is enabled so the content will not be displayed unless we click on *Show*.

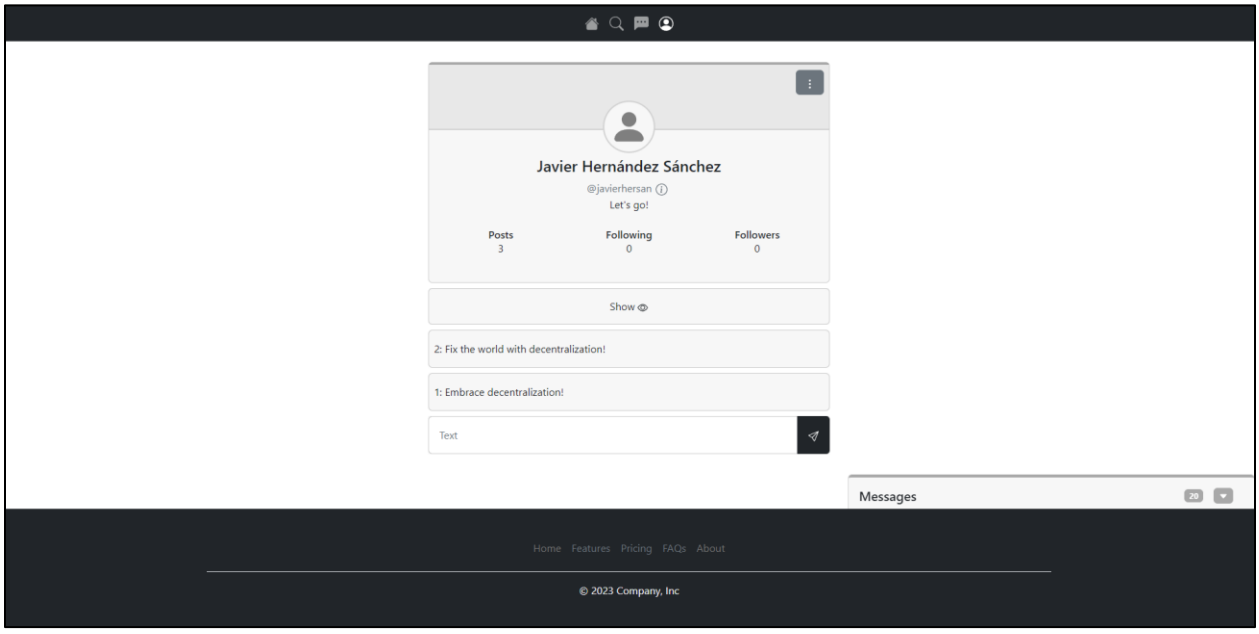

Figure 103 - DApp harmful content

Now we can navigate to the *Chat* page as shown in **Figure 104**. On this page we can create private channels and have conversations over them with other accounts. We can start a new private channel or conversation by introducing the Ethereum address of the account. Then, once the address has been introduced, we click on *new chat* button. A new chat will appear in the chat history.

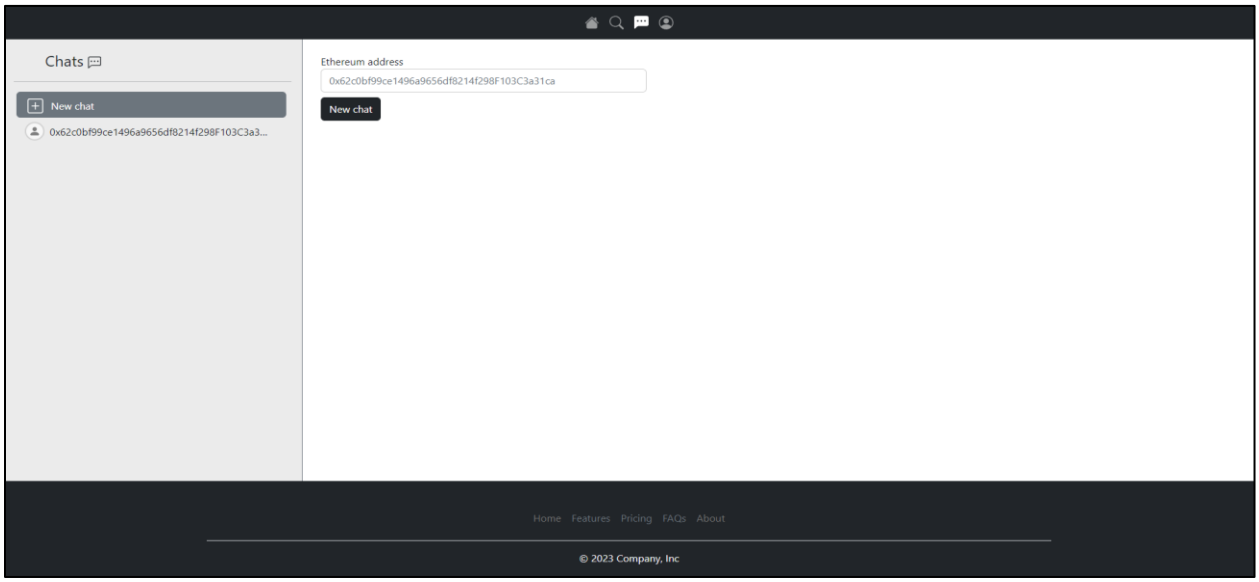

Figure 104 - DApp Creating private channels and conversations

Next, in **Figure 105** a private channel or conversation is shown. As you can see the first message of each account is a key sharing process in order to encrypt the channel end-to-end (E2E). The providers will actuate as proxy servers since direct communication between users is not possible due to the architecture of the internet. But this is not a problem since all the content is encrypted E2E, so it is impossible to suffer from *man-in-the-middle* attacks. We can change our keys whenever we want to and rotate them to have additional security.

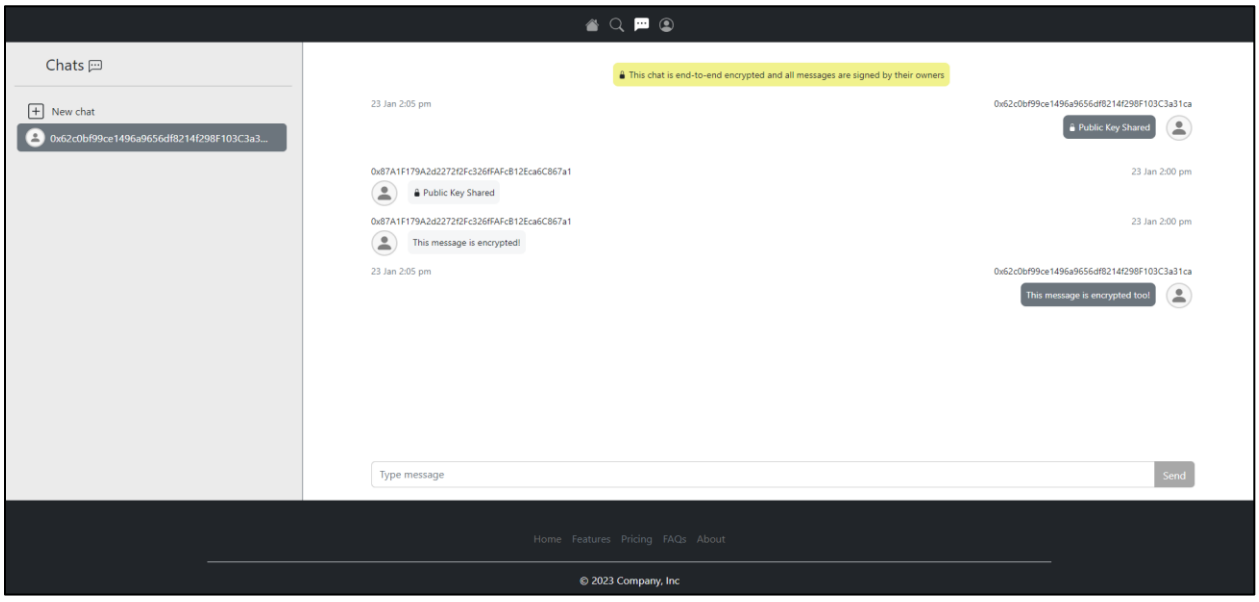

Figure 105 - DApp Private channel

Private channels can be used for other features other than chat, as we have seen when designing the standard in **Chapter 5**. However, due to time constraints, the channels still only support conversations for now.

Finally, in **Figure 106**, the *About* page is shown. This page illustrates all the tools, libraries and frameworks that have been used during the development of the social media platform.

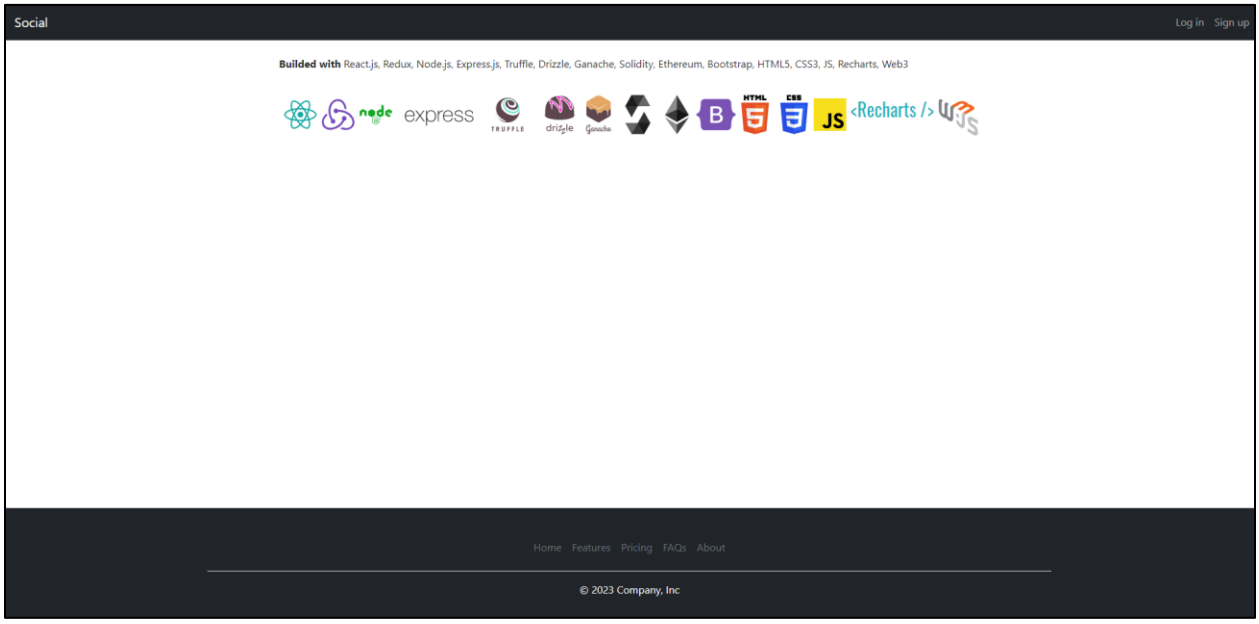

Figure 106 - DApp About page

7.3. Results

Once the DApp is tested, we can extract some results from the testing process. For this reason, **Table 8** and **Table 9** illustrate which functional and non-functional requirements have been completed. Starting with the non-functional requirements, we could say that the developed social media has completed all except NFR2. If we recall what NFR2 requirement was, it was about network security. The local information like private keys and private data must be protected against attacks and must not be disclosed or accessible from outside the client application. While it is partially true, because the local storage is not accessible from outside, all the data is saved in plain text which is not secure enough.

Code	Name	State
NFR1	Censorship resistant	Done
NFR2	Storage security	Loading
NFR3	Portability and compatibility	Done
NFR4	Scalability	Done

Table 8 - Non-functional Requirements results after testing the DApp

Regarding the NFR1 requirement which is probably the most important requirement, has been successfully completed. We have achieved a censorship resistant network, as long as users have enough providers. As the number of providers grows the more censorship resistant the network it is for a user.

Regarding NFR3, the client application is portable and compatible with any browser and OS. Finally, the scalability (NFR4) is ensured, since as the number of users grow, more providers or existing ones will increase their infrastructure to supply the demand. In the case of the functional requirements, these are presented in **Table 8**.

Code	Name	State
FR1	Public Publications	Done
FR2	Content security	Done
FR3	Content manipulation resistant	Undone
FR4	Open network	Done
FR5	Data governance	Loading
FR6	Bad actors penalization	Undone
FR7	Private Publications	Undone
FR8	Private communication channels	Done
FR9	Content and impersonation verification	Done
FR10	Digital identity	Done
FR11	Global username	Done
FR12	Content redundancy	Done
FR13	Bot resistant network	Done

Table 9 - Functional Requirements results after testing the DApp

Regarding the publication requirements, FR1 has been completed since public publications are available within the network, while private publications (FR7) has not been implemented. In the case of content security requirement or FR2, it has been accomplished thanks to the ability for users to activate content filters to protect them from harmful content.

In the case of FR3 or content manipulation resistant, it is still pending because the recommendation engines are not interchangeable and plug-and-play. These engines point to the engines of the user's providers. In terms of network openness, NFR4 is completed since any infrastructure company can join the network and become a provider.

The FR5 requirement depends on private publications (FR7), which has not been implemented. Full sovereignty is achieved when the user can select their account privacy so technically FR5 is incomplete. Also, the deal system has not been implemented which makes FR7 unfinished too. Moreover, users cannot penalize their providers (FR6) for not fulfilling their deals because the deal system and Proof of Storage have not been implemented within this version.

Regarding FR9 and FR10, these requirements have been met as the digital identity and verification systems are achieved with the Ethereum account and content headers. In the case of FR11, this requirement is fulfilled thanks to the uniqueness of DNS usernames. In terms of content redundancy, FR12 is accomplished because users can have multiple providers and store their information on them at the same time. Finally, the network is bot resistant (FR13), since valid accounts have a username reserved in the blockchain, which means they have transacted in the blockchain and they have spent money on the network. Deploying massive bot attacks is extremely difficult with this network, since it will be really expensive.

8. Conclusions

The research conducted in this project has conclusively shown that decentralized platforms offer an innovative solution to the many problems that plague traditional and centralized social media.

Decentralization of social media and content platforms, underpinned by blockchain and other decentralized technologies present significant opportunities for restoring the power back into the hands of users. We have designed a censorship-resistant system to protect users' right to freedom of expression, and that, in turn, is capable of protecting them from the dissemination of harmful or unwanted content.

The design also defeats bots, as there is a cost associated with the account creation, due to the DNS blockchain transactions. Therefore, deploying massive bot attacks is extremely expensive at least until AI is smart enough to make money easily and autonomously. Then, in that case the economic cost will no longer be an impediment and will not be able to stop artificial intelligence. However, this is a future and hypothetical scenario where the internet is said to be ''broken''. With the arrival of Artificial General Intelligence (AGI), more complex tools will be needed, such as human 2FA, but this is beyond our scope.

The design also removes the network effect and provider lock-in, as any provider that implements the standard design will be compatible with the others, removing the dependency on a particular provider. The standard design will empower the users by allowing them to decide about their data, recovering self-data governance, as users can move their data to the provider they want at any time. This is what is illustrated in **Figure 107**.

Figure 107 - Standard design implementation over different providers

The standard design can be implemented also in other applications not only in social media. This is what **Appendix A** is all about, stating that centralization problems affect other industries such as the streaming platforms. This standard design is really powerful as can be used in many more applications.

Having a standardized design and defeating the network effect allows for much more innovation, as it allows users to plug-and-play different interfaces, applications, and tools on top of the standard as long as they are compatible and compliant with it. This is what **Figure 108** illustrates, where different applications can be built on top of the standard design. This makes it possible to deploy new applications easily on any provider, simply by deploying standard software. Infrastructure providers only have to worry about running the standard, they do not have to worry about what application is running on top of them, acting like some kind of database.

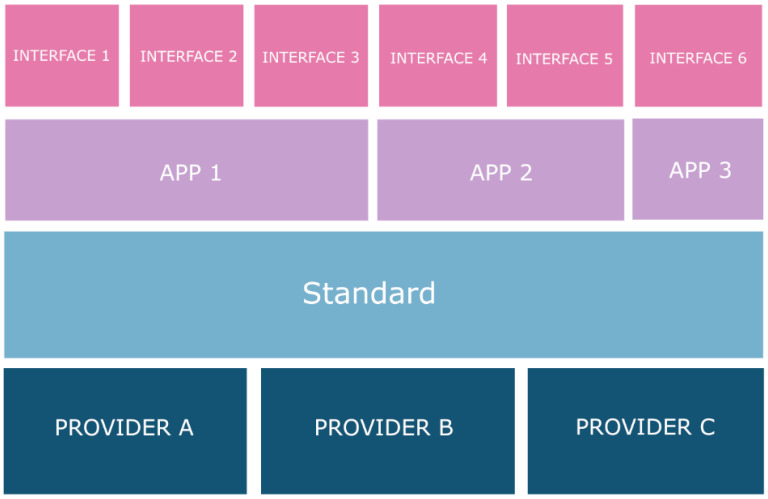

Figure 108 - Standard layers

Moreover, multiple interfaces can be used for a single application enabling high client customization. This is also shown in **Figure 109** where different client interfaces in combination with installable tools enable a lot more customization. These tools can be *plug-and-play*, allowing for a lot of innovation in the social media landscape. The **Figure 109** represents how different tools can be installed on top of each interface. The tools are chosen by the users who can select the tool that suits them best. The standard design is open to many new features through open client-side customization and the use of metadata fields in publications.

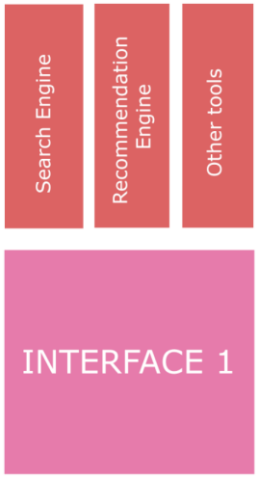

Figure 109 - Plug-and-play tools

The client-side interfaces, applications and tools can be open source, single payment, pay-per-use, freemium, or data and ad-based revenue. The trend will probably be to open source most of the applications, interfaces and tools, since most of the profits and non-economic benefits will come from the server side, the business will be mainly in being an infrastructure provider.

The impersonation problem has been solved too, since the Ethereum digital identity has been used to sign all publications and interactions within the network. Thus, it solves the self-sovereign identity problem

because users have full control of their identity and interactions within the network. Also if a fact needs to be verified by an entity, users can use the Ethereum blockchain and zero-knowledge proofs to present any proof. This is what Proof of Storage (PoSto) does. Other SSI standards can also be used, such as *W3C, Hyperledger* [22], or *IETF*.

As we have seen, many of the problems of current centralized platforms have been solved with the standard design. Decentralization of social media and content platforms, underpinned by blockchain and other decentralized technologies present significant opportunities for restoring the power back into the hands of users by ensuring data governance, privacy, higher level of trust, and encouraging innovation in the social media space.

The design presented in this work provides a framework that can be adopted widely by developers to create social media platforms and other decentralized applications on top of it. The implementation and testing of decentralized platform prototype demonstrated the technical feasibility of this approach and confirmed its potential to create a more user-centric and secure environment.

However, despite these promising results, it's important to note that the transition towards decentralized social media is not without challenges. Technical complexities, scalability issues, user adoption, and regulatory compliance are some of the obstacles that need to be overcome. These aspects have been covered in **Annex A**.

Regarding the future lines, more work needs to be done to refine and improve the standard design for decentralized platforms. More applications need to be tested and a broader consensus needs to be reached regarding the specifics of this standard design. But this is achieved with time and the project will gain maturity as it grows. The future lines are available in the next section.

In conclusion, while challenges exist, the future of social media and content platforms lie in decentralization. By addressing the problems of traditional social media and centralized platforms, we can create an online and more user-centric environment that gives the power back to the users. The findings of the project will contribute to the ongoing efforts towards decentralization and data governance, offering a promising alternative to traditional platforms and building a future where individuals have greater control over their data.

8.1. Future lines

Regarding the future lines, there are many enhancements that can be added to the design. However these improvements can be achieved by doing and extensive testing. We will have to test more applications on top of the standard design which would help identify its deficiencies. We will, nonetheless, concentrate the focus of the future lines primarily on the development, given that certain features have remained unimplemented due to the inherent complexities of the project and time limitations.

Full data governance has not yet been achieved in the development, since users cannot publish private publications. For this reason private publications are left for future implementations. Similarly, features such as the deal system and storage proofs have also been deferred to future stages, due to time constraints and its complexity.

The implementation also can lead to content manipulation by providers since search and recommendation engines are not yet plug-and-play components. These engines point to the engines of the user's providers and users cannot change their engines by others yet. This feature remains unimplemented and is therefore earmarked for future development.

In summary, the future development of this project is marked by the objective of reinforcing the design through exhaustive application tests and developing the remaining parts of the design.

9. Appendices

9.1. Appendix A: Centralization in audio and video streaming platforms

The centralization phenomenon is not only affecting social media platforms, it is becoming evident that affects other content platforms like audio and video streaming services. **Table 10** provides an overview of the ownership of some of the top streaming platforms. In this table you can see how centralized these platforms are.

STREAMING PLATFORMS
Google
amazon
Spotify
Apple
NETFLIX

Table 10 - Owners of the top streaming platforms

Everything we have seen so far also applies to streaming platforms, so the standard design can be applied to them as well. Apart from the benefits of the standard design, additional tools such as decentralized payments and decentralized finance (DeFi) can be added to the standard. The use of these tools would dramatically change the monetization landscape, empowering content creators by directly benefiting them and eliminating the need for intermediaries.

To conclude, centralization is a pervasive problem that extends far beyond social media and affects all digital platforms. The adoption of the proposed standard, empowered by DeFi tools, provides a path towards fairer platforms that serve their users and content creators, placing them at the center of development.

9.2. Appendix B: API endpoints of the standard

This appendix presents all the different endpoints required in the standard through the following tables, **Table 11**, **Table 12**, **Table 13**, **Table 14**, **Table 15**, and **Table 16**.

In **Table 6** the metadata endpoint is described. In this endpoint, the user can request basic account information, such as the most recent post. For that reason, it receives the header of that publication.

REST API Endpoints I				
HTTP	Endpoint	Action	Request Body (JSON Object)	Response Body (JSON Object)

GET	https://<server-provider-ip>/<account>/	**Root entry-point**
		Returns <account> metadata.
				metadata: {
				last_header: {
				signature: string,
				post_id: int,
				update_id: int,
				address: string,
				output_hash: string
				}
				}

Table 11 - REST API Endpoints I

With the endpoints of **Table 12**, the user can request or publish the profile information (first name, last name, and description). These endpoints are a subcase of the endpoints of **Table 13**.

REST API Endpoints II				
HTTP	Endpoint	Action	Request Body (JSON Object)	Response Body (JSON Object)

GET	https://<server-provider-ip>/<account>/post/0	Get the profile details of <account>.		{
				content: {
				first_name: string,
				last_name: string,
				description: string
				},
				header: {
				signature: string,
				post_id: int,
				update_id: int,
				address: string,
				output_hash: string
				}
				}

POST	https://<server-provider-ip>/<account>/post/0	Publish Get the profile details of <account>.	{
			content: {
			first_name: string,
			last_name: string,
			description: string
			},
			header: {
			signature: string,
			post_id: int,
			update_id: int,
			address: string,
			output_hash: string
			}
			}	

Table 12 - REST API Endpoints II

In **Table 13** more endpoints are shown. With these endpoints the user can publish, edit, delete and view any profile publication.

REST API Endpoints III				
HTTP	Endpoint	Action	Request Body (JSON Object)	Response Body (JSON Object)
GET	https://<server-provider-ip>/<account>/post/<n>	Get the post <n> of <account>.		{ content: string, header: { signature: string, post_id: int, update_id: int, address: string, output_hash: string } }
POST	https://<server-provider-ip>/<account>/post/<n>	Publish the post <n> of <account>.	{ content: string, header: { signature: string, post_id: int, update_id: int, address: string, output_hash: string } }	
PUT	https://<server-provider-ip>/<account>/post/<n>	Update the post <n> of <account>.	{ content: string, header: { signature: string, post_id: int, update_id: int, address: string, output_hash: string } }	
DELETE	https://<server-provider-ip>/<account>/post/<n>	Delete the post <n> of <account>.	{ content: string, header: { signature: string, post_id: int, update_id: int, address: string, output_hash: string } }	

Table 13 - REST API Endpoints III

In **Table 14** the channel endpoints are presented. Thanks to these endpoints the users create private channels and see the latest updates on them.

REST API Endpoints IV				
HTTP	Endpoint	Action	Request Body (JSON Object)	Response Body (JSON Object)

GET	https://<server-provider-ip>/<account>/channels	Get the latest messages from the different channels.		[
				{
				content: string,
				header: {
				signature: string,
				post_id: int,
				update_id: int,
				address: string,
				output_hash: string
				}
				},
				...
				]
				
				* "content" field is a JSON stringify object.
				
				content: {
				address: string
				messageID: int
				message: string
				}

POST	https://<server-provider-ip>/<account>/channels	Send a message to an <account> channel.	{
			content: string,
			header: {
			signature: string,
			post_id: int,
			update_id: int,
			address: string,
			output_hash: string
			}
			}
			
			* "content" field is a JSON stringify object.
			
			content: {
			address: string
			messageID: int
			message: string
			}	

Table 14 - REST API Endpoints IV

With the endpoints from **Table 15**, users can get an array of recommended publications from their providers based in their interests.

REST API Endpoints V				
HTTP	Endpoint	Action	Request Body (JSON Object)	Response Body (JSON Object)

GET	https://<server-provider-ip>/<account>/feed	Get recommended publications.	[
			{
			content: string,
			header: {
			signature: string,
			post_id: int,
			update_id: int,
			address: string,
			output_hash: string
			}
			},
			...
			]	

Table 15 - REST API Endpoints V

Finally, in **Table 16** the remaining endpoints are presented. With these endpoints the users can follow/unfollow accounts and obtain a list of the accounts they follow.

REST API Endpoints VI				
HTTP	Endpoint	Action	Request Body (JSON Object)	Response Body (JSON Object)

GET	https://<server-provider-ip>/<account>/following/<address>	Get follow list.		[
				{
				content: {
				address: string,
				following: bool
				},
				header: {
				signature: string,
				post_id: int,
				update_id: int,
				address: string,
				output_hash: string
				}
				},
				...
				]

POST	https://<server-provider-ip>/<account>/following/<address>	Follow/Unfollow an account with <address>.	{
			content: {
			address: string,
			following: bool
			},
			header: {
			signature: string,
			post_id: int,
			update_id: int,
			address: string,
			output_hash: string
			}
			}	

Table 16 - REST API Endpoints VI

9.3. Appendix C: Artificial Intelligence Content Filters

In this appendix, we will see how Artificial Intelligence Content Filters or AICF are implemented within the social media application. **Figure 110** represents an AICF as a black box system. First, some input content enters the AICF, and it is then filtered by the AICF based on certain input conditions. Finally the content will exit the output if the restriction conditions are not met.

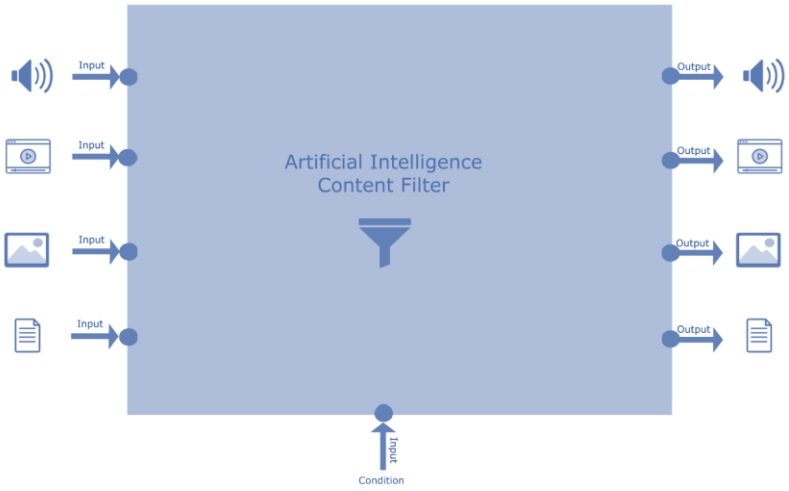

Figure 110 - Artificial Intelligence Content Filter

The AICF can be implemented in many different ways. One example of implementation is the **Figure 111**. First a computer vision and speech recognition modules for audiovisual content is placed. Then the output of that modules will enter a large language model classifier or LLM. In the case of text, it enters directly the LLM classifier module. This classifier module would classify our content based on some directives we have given to the model, and if the conditions are met it will hide the content, not show it at the output. The **Figure 111** shows the concatenation of several unimodal models, but content can be analyzed also with just a single multi modal model.

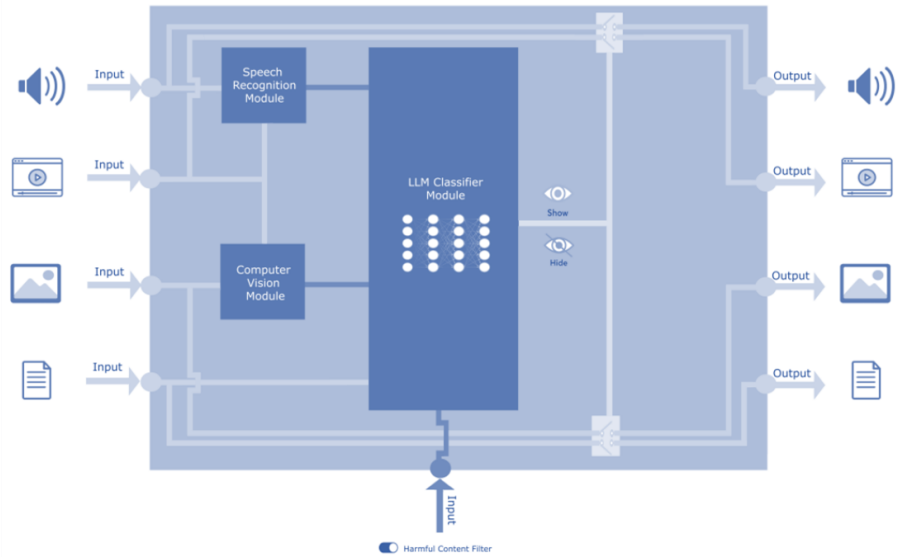

Figure 111 - Artificial Intelligence Content Filter

The Large Language Models or LLMs are a key part of AICFs. **Figure 112** and **Figure 113** show an example of how these models are really powerful for filtering content. First, in **Figure 112** a CNN political new is presented.

Figure 112 - CNN Political News

Imagine, you as a user do not want to view any political or related content in your application. You can activate a filter that writes the prompt of **Figure 113** to the LLM in the background. If the LLM returns true, the application will hide the content for you and send a warning message. Otherwise, the application will display the publication content.

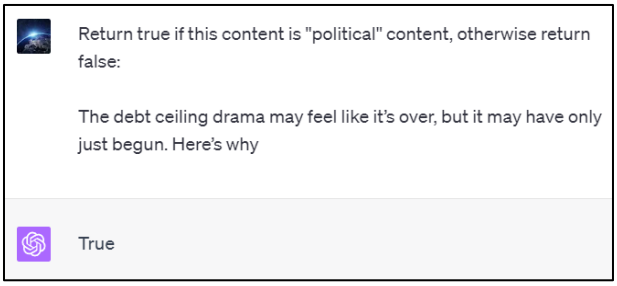

Figure 113 - LLM Content Filter: Filtering a CNN new

The same mechanism has been implemented in the application for filtering just harmful content. However, a wide variety of conditions can be imposed by the user. The content should be filtered in the client side, letting the users configure what content they would not like to see, thus respecting user freedom. However, these models are still under development and despite being a lot of open source models available, the implementation that we have used is GPT4 OpenAI model, accessing the API running in the Microsoft Azure cloud.

10. References

[1] "*Can machine-learning models overcome biased datasets*," MIT News, 21-Feb-2022. [Online]. Available: https://news.mit.edu/2022/machine-learning-biased-data-0221

[2] B. de Jong, "*Morgan Freeman Deepfake Singularity*," 2022 [Video]. Available: https://www.youtube.com/watch?v=F4G6GNFz0O8

[3] L. Thomas, "*An interpretation of the Ethereum Project Yellow Paper*," 2016.

[4] S. Nakamoto, "*Bitcoin: A Peer-to-Peer Electronic Cash System,*" 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] V. Buterin et al., "*Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform,*" 2013. [Online]. Available: https://ethereum.org/en/whitepaper/

[6] G. Wood, "*Ethereum: A Secure Decentralised Generalised Transaction Ledger,*" 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[7] Ethereum, "*The Merge*," 2023. [Online]. Available: https://ethereum.org/en/roadmap/merge/

[8] Ethereum, "*EIP-1559: Fee market change for ETH 1.0 chain,*" 2019. [Online]. Available: https://eips.ethereum.org/EIPS/eip-1559

[9] Ethereum, "*Go Ethereum*," [Online]. Available: https://github.com/ethereum/go-ethereum

[10] Protocol Labs, "*IPFS*," [Online]. Available: https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[11] Protocol Labs, "*Filecoin: A Decentralized Storage Network*," 2017 [Online]. Available: https://filecoin.io/filecoin.pdf

[12] Filecoin, "*How storage and retrieval deals work on Filecoin*," 09-Mar-2021 [Online]. Available: https://filecoin.io/blog/posts/how-storage-and-retrieval-deals-work-on-filecoin/

[13] J. Hernández Sánchez, "*GitHub repository*," [Online]. Available: https://github.com/javierhersan/Social

[14] Node.js, [Online]. Available: https://nodejs.org/en/

[15] Express.js, [Online]. Available: https://expressjs.com/

[16] Truffle Suite, "*Ganache, One Click Blockchain*," [Online]. Available: https://trufflesuite.com/docs/ganache/

[17] Truffle Suite, "*Truffle, Ethereum Development Environment*," [Online]. Available: https://trufflesuite.com/

[18] React, [Online]. Available: https://reactjs.org/

[19] Truffle Suite, "*Drizzle, Front-End Libraries for DApp Development*," [Online]. Available: https://trufflesuite.com/docs/drizzle/

[20] Truffle Suite, "*Truffle, Smart Contract Compilation, Deployment and Testing*," [Online]. Available: https://trufflesuite.com/docs/truffle/

[21] Web3.js, "*Web3.js, Ethereum JavaScript API*," [Online]. Available: https://web3js.readthedocs.io/en/v1.10.0/

[22] Hyperledger, [Online]. Available: https://www.hyperledger.org/

[23] Hyperledger Indy, Aries, [Online]. Available: https://www.hyperledger.org/use/hyperledger-indy, https://www.hyperledger.org/use/aries

[24] S. Despres, "*DNS on Blockchain: The next evolution of domain names*," Nameshield Group, 08-Apr-2020. [Online]. Available: https://blog.nameshield.com/blog/2020/04/08/dns-on-blockchain-the-next-evolution-of-domain-names/

[25] A. Shamir, "*How to Share a Secret*," 1979. [Online]. Available: https://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf

Annex A: Ethical, economic, social and environmental aspects

The main goal of the project was to design a decentralized social network that would solve the existing problems of the current platforms. This goal has been successfully achieved, but to achieve this, not only the technical challenges have been covered, but also the scalability and the economic feasibility have been considered when designing them.

Since a computing infrastructure is needed for the adoption to take place, the proposed solution should be financially feasible, so that when it is released to the market, it is attractive enough for the infrastructure firms.

Providers can earn money or other non-financial benefits by providing their infrastructure to users. But these companies also face costs such as operational expenses, development costs, and the return on investment for stakeholders. Therefore, the benefits of providing their computing infrastructure must be similar or greater to those currently provided by traditional social media companies.

Earnings can come directly from payments made by payment accounts in exchange for providing the infrastructure. But maybe there are users that are not willing to pay and do not care too much about their privacy, for these cases the social media can provide a free mode in which companies can extract and use user data.

Provider earnings can come as well from advertisers, since providers have search and recommendation engines, advertisers could pay for being recommended to users. These alternatives are the traditional alternatives and have been working since the start of world wide web.

From the user perspective, the platform and the standard must generate a network effect around it, so that the users themselves see the positive results of it, expanding the connections they have and the user base. Users also need content to consume so it is extremely important to attract content creators. This could be achieved through traditional methods or by using the powerful tools that blockchains give us in the economic realm. Here is where decentralize payments and DeFi play a very important role, since the payments can be done without intermediaries, paying directly to content creators. This empowers them and could be a powerful reason to join decentralized platforms.

Users also need a seamless, quick, and efficient experience while using the application. As we have demonstrated this has been achieved when testing the proof of content platform.

Regarding the blockchain nodes, their benefit is clear, they earn cryptocurrency in this case ether for verifying the transactions on the network. All the network actors are covered, and the proposed solution has clear economic potential behind it to each of them, so it should not concern us.

From a scalability perspective, it is important to consider that as user base or transaction volume grows, the solution should effectively accommodate the increase, maintaining high performance levels. Scalability also impacts the environment, so it should be an efficient solution and environmental responsible. The technologies employed can be scaled easily and are really efficient and lightweight technologies. The unique possible pain point and scalability impact is the blockchain. We have choose Ethereum for a variety of reason but the most important one being scalability. Ethereum is environmentally friendly since The Merge update where it updated its consensus protocol to PoS which is by far more efficient than the other existing protocols and do not require too much computing and

energy resources to run it. Also, sharding chains will be introduced later this year in 2023, reducing the amount of storage nodes have to store in memory. Ethereum infrastructure team has in it is development core these issues in mind.

On the provider side, scalability is ensured as well because the infrastructure can be adjusted and grown with demand, since more providers could join the network or even the existing ones can increase their infrastructure.

In the ethical side, we have seen the objectives of the project were to develop a user centric experience where the user has more control over their data, their identity and connections which is by far more ethical than current platforms which do not respect user preferences. These types of platforms will affect our society giving more individual freedom.

Finally, in the case of economic effects, these types of platforms are very important to encourage innovation and end with monopolies. For smaller companies trying to enter the market this decentralized and open models are very beneficial. At this time, the *unicorn companies*, which are companies with at least one billion dollars valuation, are the ones that lead the social networks and content platforms. However, with this standard it is expected to reduce these monopolies and increase innovation thanks to the end of network effect. This has big implications for Europe, as it lacks large companies in the market. The biggest tech companies are from the US and China, putting Europe far behind these giants economically. This is a great opportunity for European countries to join a market where monopolies dominate the digital landscape.

Annex B: Economic budget

The section "Requirements for the international accreditations EUR-ACE and ABET" of the TFT Regulations of the ETSIT-UPM states that "*The memory of the TFT of the GITST, GIB and MUIT, and in general those qualifications that have been obtained or for those who want to apply for an international accreditation EUR-ACE or ABET, must include an economic budget*".

		hours	Price/hour	TOTAL
LABOR COST (direct cost)		840	15 €	**12,600 €**
COST OF MATERIAL RESOURCES (direct cost)				
	Purchase	Use in months	Amortization in years	TOTAL
Personal computer (Software include)	1,000.00 €	6	5	100.00 €
Laser printer	500.00 €	6	5	50.00 €
Other equipment				
TOTAL				**150.00 €**
GENERAL EXPENSES (indirect costs)	15%	over DC		**1,912.50 €**
INDUSTRIAL PROFIT	6%	over DC+IC		**879.75 €**
FUNGIBLE MATERIAL				
Print				**100.00 €**
Binding				**300.00 €**
BUDGET SUBTOTAL				**15,942.25 €**
APLICABLE VAT			21%	**3,347.87 €**
TOTAL BUDGET				**19,290.12 €**

Table 17 - Economic budget of the project (TFT)