

# Blockchain in social networks

Bc. Ahmed Al-Doori

---

Bachelor's Thesis  
2023



**Tomas Bata University in Zlín**  
Faculty of Applied Informatics

---

Academic year: 2022/2023

# **ASSIGNMENT OF BACHELOR THESIS**

(project, art work, art performance)

Name and surname: **Ahmed Hamid Taha Al-Doori**  
Personal number: **A19895**  
Study programme: **B0613A140021 Software Engineering**  
Type of Study: **Full-time**  
Work topic: **Blockchain pro sociální síť**  
Work topic in English: **Blockchain in Social Networks**

## **Theses guidelines**

1. Create a literature review and describe the terminology of blockchain technologies related to the thesis topic, including smart contracts and decentralized applications.
2. Choose appropriate technologies.
3. Design a social network decentralized application build on blockchain technologies.
4. Implement the proposed solution of the decentralized application using modern blockchain technologies, including smart contracts.
5. Appropriately test the application, present the results, and discuss the further possible development of the application.

Form processing of bachelor thesis: **printed/electronic**

Recommended resources:

1. GERARD, David. Attack of the 50 foot blockchain: bitcoin, blockchain, ethereum and smart contracts. [England]: David Gerard, 2017, 179 s. ISBN 978-1-9740-0006-7.
2. HOLBROOK, Joseph. Architecting enterprise blockchain solutions. Indianapolis: Sybex, 2020, 1 online resource (401 pages). ISBN 9781119557722.
3. BASHIR, Imran. Mastering blockchain: a deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, ethereum, and more, 3rd Edition. 3rd ed. Birmingham: Packt Publishing, Limited, 2020, 1 online zdroj (xx, 788 stran). ISBN 9781839211379.
4. ASHURST, Stephen a Stefano TEMPESTA. Blockchain Applied: Practical Technology and Use Cases of Enterprise Blockchain for the Real World. New York: Productivity Press, 2021, 1 online zdroj.
5. BAMBARA, Joseph J. a Paul R. ALLEN. Blockchain: a practical guide to developing business, law, and technology solutions. New York: McGraw-Hill Education, [2018], 1 online zdroj (xviii, 302 stran). ISBN 9781260115864.
6. TYAGI, S. S. a Shaveta BHATIA. Blockchain for business: how it works and creates value. Hoboken, NJ: Wiley-Scrivener, 2021, 1 online resource. ISBN 9781119711056.

Supervisors of bachelor thesis: **doc. Ing. Roman Šenkeřík, Ph.D.**  
Department of Informatics and Artificial Intelligence

Date of assignment of bachelor thesis: **October 5, 2022**

Submission deadline of bachelor thesis: **May 12, 2023**



**doc. Ing. Jiří Vojtěšek, Ph.D. m.p.**  
Dean

**prof. Mgr. Roman Jašek, Ph.D., DBA m.p.**  
Head of Department

In Zlín October 10, 2022

**I hereby declare that:**

- I understand that by submitting my Diploma thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Diploma Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Diploma/Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlin, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Diploma Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlin has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Diploma Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlin with a view to the fact that Tomas Bata University in Zlin must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Diploma Thesis include the use of software provided by Tomas Bata University in Zlin or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Diploma Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Diploma Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

**I herewith declare that:**

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlin; dated: 23/05/2023

Ahmed Al-Doori m.p  
Student's Signature

## **ABSTRAKT**

Bakalářská práce se věnuje využití blockchain technologií, specificky smart kontraktům, za účelem implementace koncepce decentralizovaných aplikací v oblasti sociálních sítí. Cílem této práce je ověření vlastností, využitelnosti a bezpečnosti blockchain technologií v doméně sociálních sítí a decentralizovaných aplikací (DApps). Navržený a preferovaný přístup zahrnuje Web3.0 technologie. Bakalářská práce je rozdělena do dvou hlavních částí. Nejdříve jsou v teoretické části popsány obecně blockchain technologie, smart kontrakty, decentralizované aplikace, sociální sítě, dále moderní technologie pro budování DApps, a architektura samotných DApps. Následující praktická část je pak zaměřena na samotný vývoj, implementaci a demonstraci web DApp v oblasti sociálních sítí pomocí moderních Web3.0 technologií jako smart kontraktů, ReactJS, NextJS (JavaScript), Solidity, a dalších.

Klíčová slova: Blockchain, smart kontrakty, web3.0, solidity, NextJs, decentralizovaná aplikace, DApp, sociální sítě, JavaScript, ReactJS.

## **ABSTRACT**

This bachelor thesis will deal with the utilization of blockchain technologies, specifically smart contracts, to implement the concept of decentralized applications in the field of Social Networks. The goal of the thesis is to verify how blockchain technologies can be helpful and safe in the domain of Social Networks and decentralized applications (DApps). The preferred approach will include Web3.0 technologies. The thesis will be divided into two main parts. Firstly, the theoretical part will contain a description of blockchain technology in general, smart contracts, decentralized apps, social networks, modern technologies for building DApps, and architectures of DApps. Secondly, the practical part will focus on building a demonstration social network web DApp using modern Web3.0 technologies and smart contracts, such as ReactJS, NextJS (JavaScript), Solidity smart contracts, etc.

Keywords: Blockchain, smart contracts, web3.0, Ethers.js, solidity, NextJS, Decentralized application, DApp, social networks, JavaScript, ReactJS.

## **ACKNOWLEDGEMENTS**

I would like to express my deepest gratitude to my supervisor, Ing. Roman Šenkeřík, Ph.D., for his invaluable guidance, support, and expertise throughout the entire process of completing my final thesis. His insightful feedback, patience, and dedication have been instrumental in shaping my research and academic growth. I am truly grateful for the knowledge and skills I have gained under his supervision.

I would also like to extend my heartfelt thanks to my family, especially my mom and dad, for their unwavering love, encouragement, and support. Despite being thousands of kilometers away from them, their constant belief in me and their willingness to go the extra mile have been a tremendous source of strength and motivation.

I would like to acknowledge the contribution of Ing. Zuzana Virglerová, Ph.D., for her support and encouragement. Her mentorship and guidance have played a significant role in pushing me to exceed my own expectations and become a better student.

Lastly, I would like to express my gratitude to all the individuals who have provided assistance, encouragement, and inspiration throughout my academic journey. Your support has been invaluable, and I am deeply grateful for the opportunities and experiences I have had as a result.

Thank you all for being a part of my journey and for contributing to my personal and academic development.

I hereby declare that the print version of my Bachelor's/master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

# Contents

<b>INTRODUCTION .....</b>	<b>12</b>
<b>I. THEORY.....</b>	<b>14</b>
<b>1 BLOCKCHAIN .....</b>	<b>15</b>
1.1 BLOCKCHAIN BLOCKS .....	15
1.2 TYPES OF BLOCKCHAIN .....	18
1.2.1 PUBLIC BLOCKCHAIN.....	19
1.2.2 PRIVATE BLOCKCHAIN .....	19
1.2.3 HYBRID BLOCKCHAIN.....	19
1.2.4 CONSORTIUM BLOCKCHAIN .....	19
1.3 WHY DO WE USE BLOCKCHAIN.....	20
1.4 BLOCKCHAIN AND DISTRIBUTED LEDGER TECHNOLOGY.....	21
1.5 BLOCKCHAIN USE CASES.....	21
1.5.1 MONEY TRANSFER USING DISTRUSTED LEDGER TECHNOLOGY. ....	21
1.5.2 DECENTRALIZED APPLICATIONS .....	22
1.5.3 DIGITAL IDENTITY .....	23
1.5.4 SUPPLY CHAINS .....	24
1.5.5 BLOCKCHAIN IN GOVERNMENT .....	24
1.5.6 SMART CONTRACTS .....	25
1.5.7 DIGITAL WALLETS.....	26
1.6 HOW BLOCKS ARE ADDED TO THE BLOCKCHAIN .....	27
1.6.1 MINERS .....	28
1.6.2 PROOF-OF-WORK.....	28
1.6.3 PROOF-OF-STAKE .....	29
1.7 CRYPTOCURRENCIES .....	30
1.7.1 HOW DOES CRYPTOCURRENCY WORK? .....	30
1.8 BLOCKCHAIN LIMITATIONS IN GENERAL .....	31

1.8.1	LACK OF FLEXIBILITY .....	31
1.8.2	HUGE ENERGY CONSUMPTION .....	31
1.8.3	ILLEGAL BEHAVIORS AND BAD ACTIVITIES .....	32
1.8.4	SMART CONTRACT AS A LIMITATION .....	32
<b>2</b>	<b>POPULAR BLOCKCHAINS AND CRYPTOCURRENCIES.....</b>	<b>34</b>
2.1	BITCOIN.....	34
2.2	ETHEREUM .....	35
2.2.1	CONNECTING TO THE ETHEREUM BLOCKCHAIN.....	35
2.2.2	ETHEREUM SMART CONTRACTS .....	36
2.2.3	ETHEREUM VIRTUAL MACHINE (EVM).....	38
2.2.4	ETHEREUM STATE.....	39
2.2.5	EVM STORAGE.....	40
2.2.6	EVM GAS FEES AND EXECUTION PROCESS.....	40
2.2.7	ETHEREUM NODE VARIATIONS .....	42
2.2.8	ACCOUNTS ON THE ETHEREUM BLOCKCHAIN.....	43
2.2.9	TYPES OF ACCOUNTS IN THE ETHEREUM BLOCKCHAIN .....	44
2.2.10	ETHEREUM DRAWBACKS AND LIMITATIONS .....	45
<b>3</b>	<b>SIDE CHAINS .....</b>	<b>46</b>
3.1	SIDE CHAINS .....	46
3.2	POTENTIAL OF SIDE CHAINS .....	46
3.2.1	ADAPTABILITY TO CHANGE. ....	47
3.2.2	UPDATES AND EXPERIMENTATIONS .....	47
3.3	EXAMPLES OF SIDENCHAINS .....	47
3.3.1	POLYGON .....	48
3.4	DOWNSIDES OF SIDENCHAINS.....	49
<b>4</b>	<b>BLOCCHAIN AND WEB .....</b>	<b>50</b>
4.1	WEB 3.0 VS WEB 2.0 .....	50



4.1.1	WEB 2.0 .....	50
4.1.2	WEB 3.0 .....	57
4.2	HOW DOES WEB 3.0 OPERATE? .....	57
4.3	REQUIREMENTS TO CREATE WEB 3.0 APPLICATION. ....	57
4.4	SOCIAL NETWORKS .....	58
4.4.1	PROBLEMS WITH CENTRALIZED SOCIAL NETWORKS .....	59
4.5	DECENTRALIZED SOCIAL NETWORKS .....	60
4.5.1	APPLICABILITY OF DECENTRALIZED SOCIAL NETWORKS .....	61
4.5.2	GENERAL ARCHITECTURE OF A DECENTRALIZED SOCIAL NETWORK .....	61
4.6	REAL LIFE EXAMPLES OF DECENTRALIZED SOCIAL NETWORKS .....	63
4.6.1	STEEMIT .....	64
4.6.2	MINDS .....	66
4.6.3	LENSTER AND LENS PROTOCOL .....	67
4.6.4	COMPARISON BETWEEN THE DIFFERENT PLATFORMS .....	70
4.7	DRAWBACKS OF DECENTRALIZED SOCIAL NETWORKS.....	72
<b>II.</b>	<b>PRACTICAL .....</b>	<b>74</b>
<b>5</b>	<b>APPLICATION ANALYSIS AND TECHNOLOGIES USED.....</b>	<b>75</b>
5.1	APPLICATION/PROJECT DESCRIPTION.....	75
5.2	APPLICATION OVERVIEW AND REQUIREMENTS.....	75
5.2.1	APPLICATION FUNCTIONAL REQUIREMENTS .....	75
5.2.2	APPLICATION NON-FUNCTIONAL REQUIREMENTS .....	76
5.2.3	USE CASES AND ACTORS.....	77
5.2.4	CLASS MODEL .....	78
5.3	FRONT-END TECHNOLOGIES AND UI INSPIRATION .....	81
5.3.1	NEXTJS .....	82
5.3.2	TAILWIND CSS.....	85
5.3.3	OTHER FRAMEWORKS TO IMPROVE USER EXPERIENCE.....	86
5.3.4	USER INTERFACE DESIGN AND TAKEN APPROACH.....	87
5.4	BACK-END TECHNOLOGIES .....	90

5.4.1	SERVER-SIDE FRAMEWORKS AND LIBRARIES.....	90
5.4.2	APIs AND DATABASES.....	91
5.5	WEB3 TECHNOLOGIES.....	91
5.5.1	DEVELOPMENT ENVIRONMENT.....	91
5.5.2	DECENTRALIZED DATA STORAGE - IPFS .....	92
5.5.3	DEVELOPING SMART CONTRACTS.....	93
5.5.4	INTERACTION WITH THE SMART CONTRACT AND BLOCKCHAIN.....	96
<b>6</b>	<b>APPLICATION'S SECURITY ARCHITECTURE .....</b>	<b>98</b>
6.1	SECURITY ARCHITECTURE .....	98
6.1.1	JSON WEB TOKEN .....	98
6.1.2	SECURING THE APP .....	101
<b>7</b>	<b>APPLICATION'S ARCHITECTURE AND SETTING UP</b>	
	<b>DEVELOPMENT ENVIRONMENT .....</b>	<b>102</b>
7.1	HIGH LEVEL OVERVIEW OF THE APPLICATION ARCHITECTURE .....	103
7.2	LAYERS COMMUNICATION.....	105
7.3	SETTING UP THE PROJECT'S DEVELOPMENT ENVIRONMENT .....	105
7.3.1	LOCAL BLOCKCHAIN .....	105
7.3.2	DIGITAL WALLET.....	107
7.3.3	MS SQL DATABASE AND ASP.NET CORE API.....	108
7.3.4	HARDHAT CONFIGURATIONS (CONTRACT'S DEPLOYMENT ENVIRONMENT) .....	112
<b>8</b>	<b>IMPLEMENTATION AND EVALUATION .....</b>	<b>115</b>
8.1	APPLICATION'S SMART CONTRACT .....	115
8.1.1	CONTRACT'S STATES AND VARIABLES.....	115
8.1.2	CONTRACT'S MAPPINGS .....	116
8.1.3	CONTRACT'S EVENTS .....	117
8.1.4	CONTRACT'S FUNCTIONS.....	118
8.1.5	CONTRACT'S MODIFIERS .....	126
8.2	DEMONSTRATION OF THE MAIN FUNCTIONALITIES.....	128

8.2.1	REGISTERING OF NEW USERS TO THE PLATFORM .....	128
8.2.2	LOGIN OF EXISTING USER TO THE PLATFORM.....	133
8.2.3	CREATION OF NEW POST .....	136
8.2.4	COMMENTING ON A POST.....	137
8.2.5	LIKING A POST .....	139
8.2.6	COMMENTS PAGINATION AND POST MODAL .....	141
<b>9</b>	<b>TESTING OF THE SMART CONTRACT .....</b>	<b>143</b>
9.1	AUTOMATED TESTING .....	143
9.1.1	USER TEST SUITES .....	143
9.1.2	POST TEST SUITES.....	147
9.1.3	COMMENT TEST SUITES .....	154
9.1.4	AUTOMATED TESTING TOTAL TESTING RESULT .....	156
<b>10</b>	<b>FUTURE ENHACMENT AND IMPORVEMENTS.....</b>	<b>158</b>
10.1	SMART CONTRACT IMPROVEMENTS .....	158
10.2	GAS FEES IMPROVEMENTS AND CHOOSING OF THE SUITABLE BLOCKCHAIN NETWORK.....	159
10.2.1	USAGE OF SO-CALLED META-TRANSACTIONS .....	159
10.3	FRONT-END AND CLIENT-SIDE IMPROVEMENTS.....	160
	<b>CONCLUSION .....</b>	<b>161</b>
	<b>BIBLIOGRAPHY .....</b>	<b>162</b>
	<b>LIST OF ABBREVIATIONS .....</b>	<b>171</b>
	<b>LIST OF FIGURES .....</b>	<b>173</b>
	<b>LIST OF TABLES .....</b>	<b>177</b>
	<b>APPENDICES.....</b>	<b>178</b>

## INTRODUCTION

The rapid advancement of technology and the growing influence of the internet have revolutionized the way we communicate and interact with others. Social networks have become an integral part of our lives, connecting people from all around the world and providing a platform for sharing information, ideas, and experiences. However, these social networks are fully centralized, controlled by a few strong entities that have access to and control over user data. Centralization brings with it a set of concerns and limitations, including issues of privacy, data security, and censorship. As a response to these challenges, decentralized social networks have emerged as a promising alternative that leverages the power of blockchain technology to create a more transparent, secure, and user-centric online social environment.

This thesis aims to explore the concept of decentralized social networks and the role of blockchain in enabling their operation. It provides an in-depth analysis of blockchain technology, its types, and various use cases. Furthermore, it delves into popular blockchains and cryptocurrencies such as Bitcoin and Ethereum, highlighting their strengths and limitations. The thesis also examines the potential of side chains and their impact on scalability and adaptability in the context of decentralized social networks. The study recognizes the evolution of the web from Web 2.0 to Web 3.0 and discusses how Web 3.0 operates, focusing on its decentralized nature and the requirements to create web applications in this new paradigm. It critically examines the problems associated with centralized social networks and emphasizes the need for decentralized alternatives. The general architecture of decentralized social networks is explored, along with real-life examples such as Steemit, Minds, and Lenster and Lens Protocol.

Furthermore, the thesis delves into the analysis and evaluation of an application that embodies the principles of a decentralized social network. It outlines the technologies used in the front-end, back-end, and web3 layers, along with the security architecture of the application. The implementation and demonstration of the main functionalities are also presented, showcasing the registration of new users, login procedures, post creation, commenting, and liking features. Finally, the thesis concludes with a discussion on future enhancements and improvements for decentralized social networks, including smart contract

improvements, gas fee optimizations, and client-side enhancements. It emphasizes the potential of decentralized social networks in reshaping online interactions and promoting a more democratic and user-controlled online ecosystem. By exploring the theory, analyzing popular blockchains and cryptocurrencies, discussing the potential of side chains, examining the intersection of blockchain and the web, and evaluating an implemented application, this thesis contributes to the existing body of knowledge surrounding decentralized social networks. It provides insights into the advantages, challenges, and prospects of decentralized social networks, paving the way for a more decentralized and user-centric online social landscape.

## **I. THEORY**

# 1 BLOCKCHAIN

First, let's talk about the naming and what does the word **BLOCCHAIN** means and from where the name is obtained. The name of this technology refers to the description of how it does work and how it stores data differently than other technologies. The data is stored into blocks, and each block of data is linked to the other block, and they all form a chain, in this way no one can alter one block of data without making the other block knowing about this change and that the data of this block is altered, basically it is not possible to alter or delete a block, it is only possible to add a new block to the chain. Blockchain technology is not a name for a company or for an application, but it is a completely new strategy to store data on the internet. This technology uses hashing algorithms to safely secure the data and to create a safe link between the blocks of data. The most beneficial advantage of this technology is that people feel safe and free when they send or store their data because their data will not be stored in the hand of someone or somewhere, the data will be stored in the blockchain in a decentralized way, where there is no one controlling this data or as we said altering this data. We can describe this technique of storing data like a family, each member of the family trusts the other one and protect the other one, this family does not let any guests enter their house without letting all the family members knows that there are guests coming and they should all be satisfied that someone is visiting them. The same goes for all their secrets, they don't tell their secrets to anyone outside this family. [1]

## 1.1 Blockchain blocks

It is important to know how blocks of the blockchain are constructed and what do they look like from the inside. First it is better to know that a block should have some data in it otherwise it is not possible to insert or connect the block into the blockchain. In the *Figure 1* this is what a blockchain structure looks like in general.

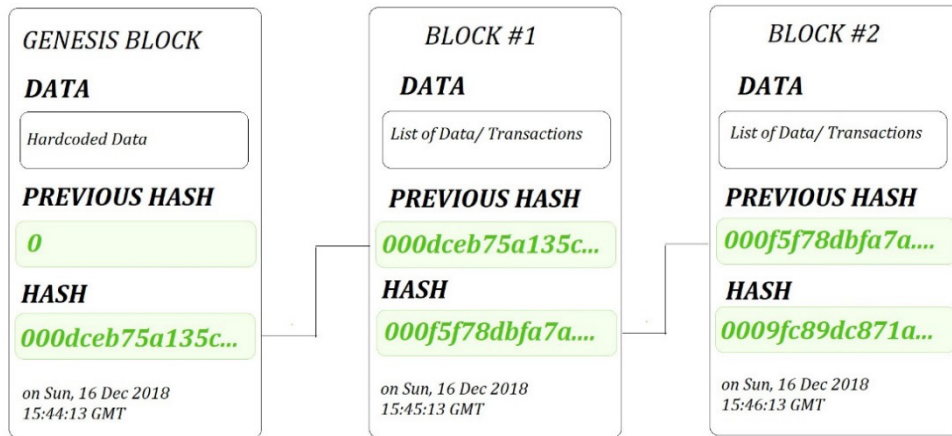


Figure 1. Blocks of blockchain [2]

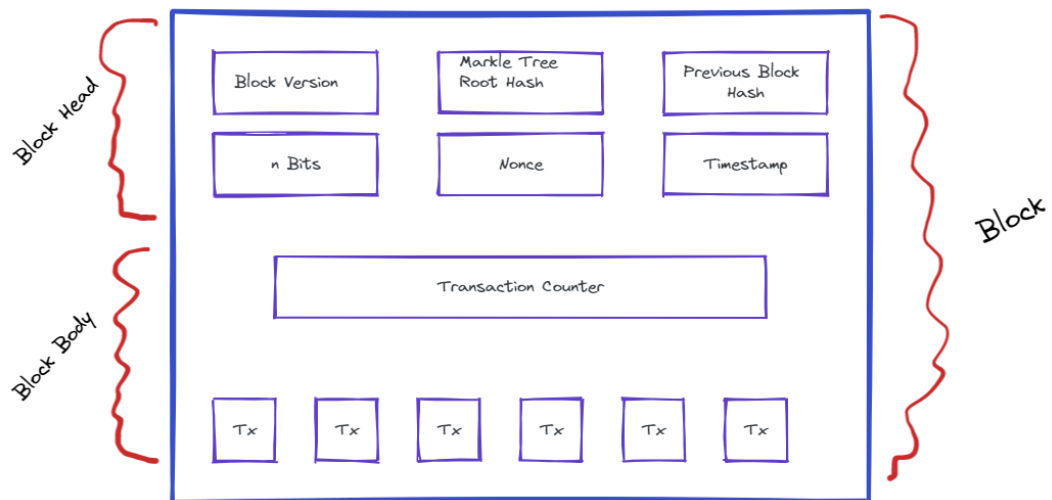


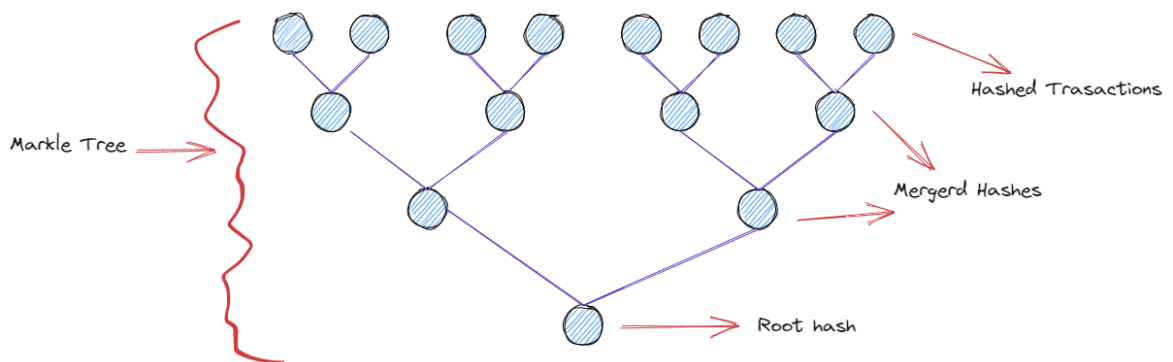
Figure 2. Block structure [2]

As shown in the *Figure 2*, a block is divided into two main parts. First is the block head, second is the block body. The block head is divided into six different parts those are:

- Version number.
- Previous block hash.
- Merkle tree root hash.
- Nbits.
- Nonce.
- Timestamp.



The Block version or *version number* tells which set of validation standards this block should follow, this is useful for people who want to insert a new block to the blockchain network, in other word mining. If someone wants to insert a new block to the blockchain network, he needs to check the rules and regulations by checking the version block or number. The Merkle tree root hash as shown in the *Figure 3* is a kind of data structure that is used in bitcoin and other cryptocurrencies this data structure is used to encode data inside the blockchain in a sufficient and efficient secure way, it is also called “Binary hash trees”, It works in way that it collects all transactions in a block and generates a digital fingerprint of the entire set of the transactions. The data or the transactions is hashed first then hashes that used to encrypt the data are hashed again and merged to only single hash that is finally called the root hash.



*Figure 3. Markle tree [2]*

Each block except the first block in the blockchain, contains the previous block hash and because that each new block has the hash of the previous block that's why it called blockchain and that's why all the blocks are connected. This sequence of linking goes back to the first block in the blockchain which is called the *genesis block*. On the other hand.

The *n Bits* part. Is also called the difficulty target, it specifies the complexity and the computation power that is necessary to mine the network if the n bits part is too big it means we need a stronger expensive computational machine to mine it, different kind of hashes algorithms are used to make it more stronger to mine it, we should remember that when using hashing function it is not possible to get the input from the output and that's why hashing process is also called digestion process which is lead to a really powerful encryption algorithm.

The *nonce* stands for “number used once” the nonce is used to validate the information within a block. A nonce is a four-byte number that the miners who's trying to insert a new

block to the blockchain should find. In order for the miners to create or insert a new block to the blockchain they must do these following procedures:

1. A collection of transactions will be gathered by the miners that they want to include in the new block.
2. This collection of transactions will be combined with other data such as the timestamp, and the hash of the previous block, to form the block data.
3. A nonce will be generated by the miner's software and which usually an integer number.
4. The miners will combine the block's data with the nonce to create a new hash.
5. The hash should be checked with the **target difficulty level**, which is a numerical value that specifies the level of difficulty that the miners must overcome to create a new block in the blockchain, it is not a standard or a hash value it is rather a value that is agreed upon by the blockchain network. The target difficulty level is set by the number of leading zeros required in the block's hash, the more leading zeros required, the more difficult is to find a hash that meets the target. The target difficulty level is modified in proper way by the blockchain network to make sure that the rate of inserting new blocks to the network remains consistent. And this modification is based on how strong is the computing power in the network and the rate of block creating.
6. The miner successfully created a block and can insert a new block only if the hash meets the target difficulty level.
7. If the generated hash does not meet the target difficulty level, the miner will try to change the nonce and repeat the process until a hash met the requirement of the difficulty level.

Nonce has a huge role in the process of mining the blockchain, as it ensures that only new blocks are added if they meet the required level of difficulty. The *timestamp* part is data that is stored in each block. The main purpose of this timestamp is to determine the exact moment of each block when it was mined and validated by the network, this piece of data shows us how blocks in the blockchain are linked in a chronological order way, also the timestamp is the amount of time that was required to create this block. The *transaction counter* is basically a place where all the transactions are performed, here will be stored the reference for each transaction that happened in the block. [3] [4]

## 1.2 Types of blockchain

To understand how the blockchain works, it is necessary to describe the difference and know the various types of blockchain, there are four different types of blockchain. **Public**, **private**, and **permissioned** or constructed by a group of people called **consortium**.

### 1.2.1 Public blockchain

This is the first type of blockchain and the most famous and used nowadays. A public blockchain is a non-restrictive network, anyone with internet access can sign on to a blockchain platform to become an authorized node. The advantage of such network is that it is independence basically it is decentralized means no one is responsible for these blocks but the users themselves are and can be trusted but it has less performance and scalability than the other types of blockchain due to the number of the blocks or nodes that grows day by day. Public blockchain most use case is the mining and exchanging the cryptocurrencies like Ethereum or bitcoin. [5][6]

### 1.2.2 Private blockchain

This type of blockchain is more secure and much faster due to the concept that it is centralized network because it is managed by an administrator, due to that only people with invitations can join this type of network. This kind of network is used by special businesses and organizations such as healthcare, financial services, government organizations because it provides these organizations with database services that are fully secure and scalable and more fault tolerance.

### 1.2.3 Hybrid blockchain

A hybrid blockchain is a unique type of blockchain that tries to get the benefits of both public and private one, the transactions in hybrid blockchain are made private. The hybrid blockchain members can decide who can participate in the blockchain or which transactions are made public. This brings the best of both worlds and ensures that a company can work with its stakeholders in the best possible way. [7]

### 1.2.4 Consortium Blockchain

This type of network is best for organizations where there is a need for both types of blockchain, public and private. It is almost like hybrid one the only difference is that hybrid controlled by one entity only. For consortium blockchain there is more than one central in-charge, or more than one organization involved who provides access to pre-selected nodes for reading, writing, and altering the blockchain. [8]

The *Table 1* shows a summary of differences between different types of blockchains.

Table 1. Types of blockchain overview

Features	Public	Private	Hybrid	Consortium
Accessibility	Anyone	Single Person/ Central In charge	Single organization controlling the nodes	More than one organization act as center of the data
Participate	Anyone	Permissioned and known identities	Permissioned and known identities	Permissioned and known identities
Transaction speed	Slower	Faster	Faster	Faster
Decentralization	Complete decentralization	Less Decentralized	Less Decentralized	Less Decentralized

### 1.3 Why do we use blockchain

It is essential to highlight the benefits of blockchain technology and the reasons for its utilization and significance. When considering the advantages of this technology, the primary and most significant benefit is its decentralization concept. Consequently, blockchain can be regarded as an immutable public digital ledger, where recorded transactions cannot be altered. Additionally, the implementation of encryption algorithms and hashing functions ensures the constant security of the blockchain. Each transaction is made or each inserting of a block should be proven by all the participants in the network. It is also possible to talk about why changing one block in the chain would be impossible, because each block has a hash value, and this hash value can be obtained by encrypting/hashing all the data inside the block and the next block will be linked to the previous block by using its own hash value as link between the two. [9]

## 1.4 Blockchain and distributed ledger technology

The phrase "Distributed Ledger Technology" (DLT)<sup>1</sup> refers to the technological architecture and protocols that allow for concurrent access, record validation, and immutable record updating via a network that is spread over several businesses or locations. The DLT technology was introduced by bitcoin, this technology is all about the idea of a decentralized network against centralized network. DLT is all about the idea of distributed network, distributed network gets rid of the need for a central authority to keep a check against data manipulation. The way DLT stores data is in a secure and accurate way by using cryptography. The same that can be accessed using "keys" and cryptographic signatures. If the information is stored it becomes an immutable database meaning no one can alter or delete anything, and the information is governed by the rules of the network. The idea of a DTL is not completely new, many organizations keep their data at different locations, but still each location is connected to a central system, which keep an eye for them periodically, this approach makes the central database much stronger and vulnerable to cyber-attacks, this approach also decreases the delays.

## 1.5 Blockchain use cases.

Blockchain is not only about Bitcoin and cryptocurrencies, but it is also much more than that it is a really great way of storing data in a decentralized way. Most important use cases for us as technicians' people or software engineers is that it used in Gaming, Healthcare, NFT<sup>2</sup> marketplaces, IoT (internet of things), government voting, smart contracts, DApp (Decentralized applications), original content creation.

### 1.5.1 Money transfer using distrusted ledger technology.

Blockchain can save the largest banks in the world a lot of money and time. Let's take an example, let's say John he lives in France, and he want to send money to his friend Daniel in Czech Republic the process for sending/receiving the money using banks will usually takes more than 1 day across different countries because a bank will be connected to other bank that will process the money and connect with the main bank in that country to connect

---

<sup>1</sup> (DLT) Distributed ledger technology [10]

<sup>2</sup> NFT (non-fungible token): <https://www.forbes.com/advisor/investing/cryptocurrency/nft-non-fungible-token/>

with the branch bank and makes sure everything is valid, with the help of blockchain and DTL this whole process can be done in seconds. With the blockchain each user or participant will be connected to the blockchain network and will be able to connect to every other user. Let's talk about how the payment can be done using DLT by taking real life example as shown in the *Figure 4*. Jenny and Alex are part of the safe secured blockchain network, and both hold a distributed ledger, and now it is just the matter of adding new transaction to the chain. And as Alex request the bank for the transfer, the bank creates a record for the transaction and that has details as, timestamp, debit account credit account, sending bank, amount, currency, receiving bank, the bank put this record on the chain with the required encryption hash. When the record is added to the chain its validated by all the nodes on the network with their key for integrity and the block is finalized. The block now contains all the information that is accessible by all participants and denotes that Alex want to send money to Jenny who has account with the specified account address, and now the receiving bank start the accounting process at their side thus the payment is done within seconds. [11]

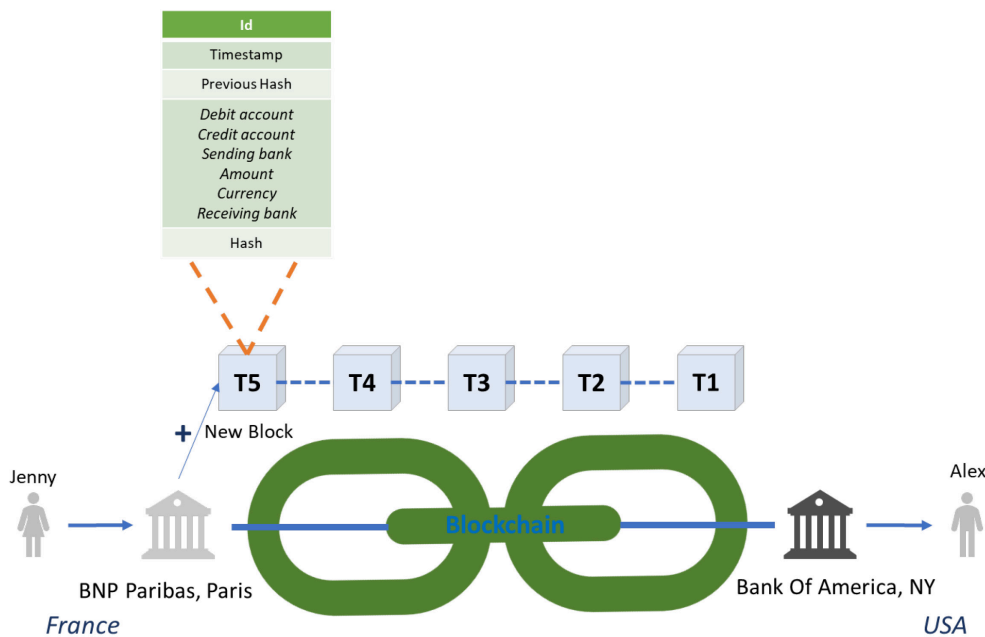


Figure 4. Money transfer blockchain [12]

### 1.5.2 Decentralized applications

Decentralized applications are apps that are running on the blockchain which is as we said a network of nodes or computers, instead of running on single computer or server. The good thing about DApps is that the users inside the apps own their data. There is no center authority between the user and the data that can control the user data. DApps which are often

built on the Ethereum blockchain can be coded for different purposes including social media, gaming, finance and more. Decentralized applications can be hosted on a public blockchain so that all people can access it around the world easily.

### 1.5.3 Digital identity

When talking about digital identity in the world of the internet, the first thing that comes to the mind is that what we share in this big world that we call the internet, digital identity is not only about our social media profiles or email addresses. Our digital identity is made up of everything we have on the web including images, shopping preferences, website usage behavior, bank account information. To know more about what we mean by digital identity, one of the examples of the good usage of digital identity is the National Digital Identity or NDI system in the Smart Nation initiative in Singapore<sup>3</sup>. The NDI system would help citizens secure access to e-governance services. Most of us are worried about our personal information and if it is leaked into this huge world or it has been sold by specific organizations or it has been used by someone else. Blockchain can be a great solution for decentralized identity, since we know that blockchain addresses are extremely unique, and we can use these addresses to make digital identity. Decentralized digital identities will be a huge player in the case of banking, e-commerce, gaming, healthcare, Insurance, loan, and most importantly social media, payments, travel. When talking about decentralized digital identity in the world of the blockchain we mean by that SSI (Self-sovereign identity), if a user has SSI, he/she can control who can access their information, maintain their identities themselves, use it for verifying their identity online, and many more. When it comes to SSI a platform should have at least the following principles, so it is considered as a decentralized digital identity provider. These principles are first, all users need to have independent existence in the platform. Second, users should have full control over their identity. Third, is the protection, no matter what happens the platform should maintain and secure user personal information. Fourth, is the agreement, if for any reason a third party wants to access a specific user identity then the user should give permission for that, without permission no third party is able to access their personal information. After all, that's why blockchain technology is a good example of such a platform. [13]

---

<sup>3</sup> Smart Nation initiative in Singapore: <https://www.smartnation.gov.sg/about-smart-nation/transforming-singapore/>

#### 1.5.4 Supply chains

Using blockchain technology in the field of supply chains can provide so many benefits. First, the immutability, since the data cannot be altered in the blockchain, and it is almost impossible to change or manipulate a specific transaction, this makes the use of blockchain technology in supply chains highly resistant to fraud. Second, the traceability, Due to its extensive and integrated linking of operations, blockchain makes it simple to map and visualize the processes in a supply chain. This implies that every stage of the process, from the source of raw materials through the delivery of the finished product to the client, can be tracked. By lowering the possibility of fraud, mistakes, and counterfeiting, as well as increasing openness and accountability, the supply chain is made more effective overall. Third, lowering the cost, since blockchain allows cross-border transactions. Due to that the need for central authority or middleman to make a money transaction will not be necessary any more business that will use the blockchain in the supply chains will not only save money but will also save time. [14]

#### 1.5.5 Blockchain in government

Blockchain technology can be useful in the e-government field from the side of data protection to the side of tracking the data. The government can benefit from this technology in many aspects. First, it increases the efficiency and transparency and reduction of costs, blockchain can help reduce the paperwork and the manual data entry by creating secure and immutable records of transactions and data, this leads to more efficiency and transparency in majors such as taxation, land registry and public services, no one will be able to change any contracts or paper data, thanks to the blockchain technology and the immutable feature. Second, digital identity management, government organizations can provide the citizens with a secure and convenient way to access services and ensure that only authorized people can access sensitive information. This way it can prevent fraud and minimize the cost of identity verification as much as possible. Third, validation of qualifications, the steps of validating qualifications are usually time consuming and costs a lot. This is since there is no single repository of qualifications, and every authority has its own steps for the validation process. Creating a shared, immutable proof of record of all qualifications using blockchain technology can help to ease this process. The UAE (United Arab Emirates) government already has launched several initiatives for encouragement to the use of blockchain technology in the country, one of the most was the “The UAE Blockchain Strategy 2021”,



which is aimed to make Dubai the first city that is fully powered by blockchain, the strategy was focusing on government efficiency, industry creation, and international leadership. As they are saying that they want to make Dubai a paperless city. [15]

### 1.5.6 Smart contracts

Smart contracts are no more than a programming code that is executed on the blockchain network. They are innovated for the purpose of reducing the paperwork and reducing the need for third party between two sides, also for exchanging the money and goods and services even real estates. In 1996 *Nick Szabo* was the first person who described the principle of smart contracts. From Szabo's view, smart contracts are digital information transfer protocols that use mathematical algorithms to automatically execute a transaction only and if the established conditions are met. Even to this day, since 1996, this definition is still valid and accurate. Smart contracts are immutable, verifiable, and autonomous pieces of code. Since the blockchain technology has the feature of immutability, a deployed smart contract or a verified transaction cannot be altered or deleted. smart contracts are:

- **Fast enough:** writing and processing contracts by hand can be really time consuming, it also requires both sides to be present, while smart contracts from the other hand do not require personal involvement, it is an automated process.
- **Independence:** the need for a third party between two sides is eliminated with the usage of smart contracts.
- **Reliable:** once the data inserted into the blockchain, it cannot be deleted or altered in the future
- **Less expensive:** while real contracts require operational costs, and attendance of two parties and middleman, smart contracts do not require all of this.

It is also worth it to mention the disadvantages of smart contracts in real world, these can be:

- **Reliability:** reliability can also be considered as a disadvantage, if some conditions provided to a smart contract and the smart contract is deployed to the blockchain, there is no way the conditions can be modified, that's why governments still thinking about jumping to make real contracts implemented using the blockchain technology.
- **Implementation challenges:** implementing smart contracts in the real world often takes too much time and effort.
- **The lack of knowledge:** while smart contracts are good to use and have many advantages. People and organizations and even the governments still don't know a lot about smart contracts and the world of blockchain.

Talking about the development of smart contracts, it is worth to mention not all types of blockchains support the concept of smart contracts. Bitcoin itself does not support the concept of smart contracts. Ethereum on the other hand is considered as the most popular blockchain for smart contracts, Ethereum has its own programming language that is used for the development of smart contracts, it's called *Solidity*, Solidity is an object-oriented programming language, it is a statically typed programming language. Also, *Hyperledger Fabric* blockchain, *Corda* blockchain, *Stellar* blockchain, all these blockchains<sup>4</sup> support developing of smart contracts. [16]

### 1.5.7 Digital wallets

Blockchain digital wallet is the container for different types of accounts on different types of blockchain networks. They allow the users to manage different types of cryptocurrencies such as Bitcoin or Ethereum. A blockchain wallet is almost the same as the normal wallet, it helps the users or the owners to exchange currencies and make transactions easily. Blockchain wallets are accessible from web or mobile, Transactions in digital wallets are highly secured they are hashed with cryptographic functions. The process of receiving and sending currencies in blockchain wallets is very similar to once in bank accounts or payable accounts. An example of Blockchain digital wallets can be MetaMask, Coinbase Wallet and more. Talking about how do blockchain wallets operate, it is important first to talk about private and public keys to explain the operation process of wallets. Whenever a blockchain wallet is created by a user, a user will be provided with a private key that is important to access the wallet any time. And provided by a public key at the same time as shown in the *Figure 5*. To explain more how these keys are important a real-life example will be taken. If someone gave me his phone number it does not mean I can use the phone number to make calls from it, instead the ability to send that user a message and make a call. For me to be able to use the user phone number for making calls or sending messages I must get the SIM card. Same goes with the blockchain wallet the public key is used for receiving/sending transactions, while the private key acts as the SIM card or as my password for my email.

---

<sup>4</sup> Blockchains that's supports smart contracts: <https://cointelegraph.com/learn/smart-contract-development-platforms>

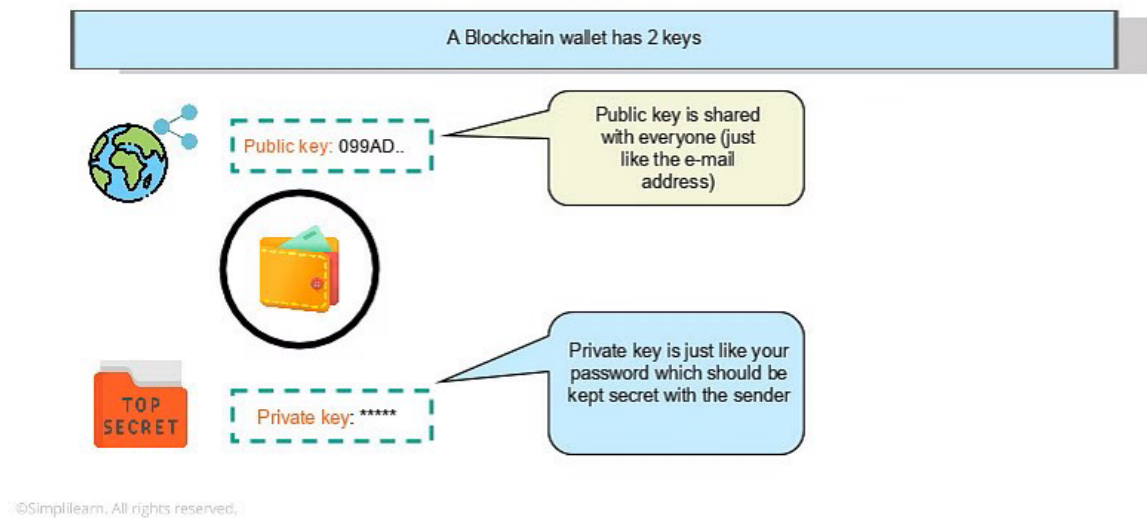


Figure 5. Blockchain wallet [17]

Advantageous of blockchain wallets can be:

- A highly secured way of storing digital currency, the only thing that the user must secure and not share with anyone is the private key. Because this is how anyone/he can access the wallet and make transactions.
- Allowing transactions using different types of cryptocurrencies.
- Allowing transactions across different countries throughout the world.

There are currently only two types of blockchain wallets, hot wallets, and cold wallets. Hot wallets are online wallets which are usually used to transfer cryptocurrencies in a fast way. MetaMask and Coinbase wallets are an example of a hot wallet. Private keys in hot wallets are stored in the cloud for faster transfer. On the other hand, cold wallets are digital offline wallets, the transaction in these wallets is made offline and then confirmed and applied online, these types of wallets are maintained offline to have high security. The private key in these types of wallets is either stored on a paper document or on a hardware device. Such wallets are Trezor and Ledger. [17]

## 1.6 How blocks are added to the blockchain

As mentioned before, a block of blockchain is no more than a container for data that is strongly protected and encrypted and linked to the other blocks. There are different ways and mechanisms used to verify transactions and adding new blocks to the blockchain, talking

about the cryptocurrency world which is the most common use cases currently in the world of blockchain technology, there are two common methods, and these are proof-of-work and proof-of-stake.

### 1.6.1 Miners

Miners are the people who add new blocks into the chain, this process called mining. Special software is used by the miners to solve complex math puzzles, for example they must find a nonce that generates an accepted hash. A nonce stand for **number used once**. This number is a random number that can be used only once. It is a 32-bit string that is subjected to modifications by miners for making sure that is valid to use in hashing block's value. This number when combined with the information provided in the block and passed to a hashing function must result in a value that satisfies given conditions. When the conditions are met the other nodes in the network will verify the validity of the outcome results. Therefore, the miner is rewarded with some crypto coins because we were able to append a block into the network. Therefore, it is impossible to add a new block into the head chain without finding a valid nonce which is responsible for generating the solution for a specific block. Each validated block contains a block hash that represents the work done by the mine and from this concept this why the mechanism proof-of-work called proof-of-work.

### 1.6.2 Proof-of-work

Proof of work was originally dated back to 1993, the proof of work concept was created or developed to prevent service attack, in 2009 Bitcoin introduced a good way of using proof-of-work as consensus algorithm, it is like an agreement between people to prove that something is correct, it is like witnesses in the court. This algorithm has been widely used by many cryptocurrencies mainly by Bitcoin and Ethereum, Litecoin and much more. The concept is that miners on the network will try to compete in solving complex computational problems by using their computers as shown in the *Figure 6*. Basically, it is cracking of the code, all the miners on the network will have to verify that the solution found by a miner is correct, so that a miner is able to add a block to the blockchain. Talking about bitcoin, bitcoin is a blockchain-based system that is maintained by a group of distributed decentralized nodes. Some of these nodes are known as miners and they are responsible for adding new blocks into the blockchain, to the miners to be able to add block to the blockchain they need to guess the nonce. [18]



Figure 6. Proof of work Process [19]

### 1.6.3 Proof-of-stake

Proof-of-stake is the successor of proof-of-work since proof-of-work mechanism requires powerful computing devices that can solve very big complex mathematical problems, proof-of-work causing the miners to use massive amounts of energy. Because of this, people are making large mining houses or farms in general form. These farms contain a huge number of GPUs that are responsible for the solving the complex mathematical puzzles, bitcoin miners use about 54 TWh of electricity that is enough to provide five million houses in the US or even power the entire country of New Zealand. Because of all these disadvantages a new consensus algorithm that is as effective as proof-of-work or even better is introduced. The basic idea is letting all the nodes or the users on the chain compete against each other but without using the concept of mining, instead an election process is done in which 1 user or node is randomly chosen. So, they are no longer called miners, instead they are Validators. The basic idea here is that each validator will pay specific amount of money to the network, the higher the validator pays the higher chance he gets to be selected by the network as validator to add his block to the blockchain but the validator will not get a reward for making a block but so the block creator will take a transaction fee. [20]

## 1.7 Cryptocurrencies

A cryptocurrency is a digital form of currency. That is highly secured by cryptography. Cryptocurrencies are not owned by anyone they are distributed among a network of computers or as we referred to it as a collection of nodes. Cryptocurrencies does not rely on any authority to verify transactions. It's a peer-to-peer system that can enable anyone anywhere to transfer and receive payments anytime. When talking about the terminology and where it came from cryptocurrency received its name because "Crypto" refers to the various encryption algorithms and cryptographic techniques that protect these entries such as elliptical curve encryption, public-private key pairs, and hashing functions. Cryptocurrencies are nothing else other than encrypted string of data or a hash, encoded to signify one unit of currency. They have no physical form and exist only in the network. The first known cryptocurrency was Bitcoin, which was founded in 2009 and remains the best known today. People are more interested in cryptocurrencies for the purpose of trading for profit. [21]

### 1.7.1 How does cryptocurrency work?

Cryptocurrencies or digital cash was discovered or found as a side product of another invention. The inventor/inventors of Bitcoin or the inventor of the first blockchain *Satoshi Nakamoto* (it is the name used by the person or people who developed bitcoin as a Nickname), in one of his announcements of Bitcoin, Satoshi said he created or developed "A peer-to-peer Electronic cash system". In his speech he said the following "*Announcing the first release of Bitcoin, a new electronic cash system that uses a peer-to-peer network to prevent double-spending. It's completely decentralized with no server or central authority*" – (*Satoshi Nakamoto, 09 January 2009, announcing Bitcoin on SourceForge*<sup>5</sup>). The most important thing of the *Satoshi's* invention was that he found a way to build a decentralized digital cash system, as described before means there is no third party between two people that is communicating for the purpose of money transaction or contract creation. Before Bitcoin there were many tries to invent a digital cash system that is in somehow decentralized, *DigiCash* was a digital money company that wanted to make a system where digital payments are anonymous but unfortunately their plan did not go well because they got bankrupt according to Wikipedia the company was founded in 1989 and

---

<sup>5</sup> SourceForge bitcoin's activities history: <https://shorturl.at/cirtE>

defunct in 1998. After that *Satoshi* realized that there were many failed attempts to build a digital cash system without a central entity. Like a peer-to-peer network for file sharing. This decision became the birth of cryptocurrency. There were just little pieces that were missing that *Satoshi* was able to find to invent such a decentralized system. To get or observe digital cash we need a payment network, with accounts, balances, and transaction. In a decentralized network, every peer in the network needs to maintain a list with all transactions to examine if future transactions are valid or an attempt to double spend. If one peer in the network disagrees about a single minor balance, everything then will be broken. It is necessary to get an absolute consensus. [22][23]

## **1.8 Blockchain limitations in general**

With all the progress and the trend of blockchain technology that was made, blockchain technology is still missing a lot of features and has huge limitations for different type of solutions. Blockchain technology has a long way to go before it can reach its full potential. As mentioned, the best feature of the blockchain technology is decentralization, however the openness and the lack of centralized controller may have unintended effects that restrict the system's usage.

### **1.8.1 Lack of flexibility**

Blockchain is a complex huge data structure system. That consists of a variety of concepts and protocols that are optimized and adopted to one another. Such a huge environment or system can be very difficult to change or alter. Till now there have been no established procedures for how to change or upgrade the blockchain components. For example, talking about the cryptographic techniques that is been used by a specific blockchain, these techniques must be valid for the lifetime of the blockchain. There is also a huge problem for enterprises and people that are developing on the top of a specific blockchain caused by the immutability of the system and the idea of data cannot be altered or deleted, even for developers it will be difficult to fix bugs or adjust the blockchain protocol. These problems make blockchain technology more difficult to deal with than other technologies.

### **1.8.2 Huge energy consumption**

Blockchain technology implementation requires a lot of energy, specifically if we talk about the Bitcoin blockchain, since Bitcoin works with the POW concept, means miners needs

more energy and more computational resources to mine a Bitcoin, as mentioned in 1.6.2 and 1.6.3.

### 1.8.3 Illegal behaviors and bad activities

Cryptocurrencies and blockchain technologies can also be used as a means of exchange for a variety of unlawful and dishonest activities. Such activities include examples of cybercrimes. The following activities are the main threat for the blockchain technology:

- **Darknet markets and illegal trade:** since privacy is number one feature in the blockchain technology this make it easier for thieves and drug dealers to do their operations easily, in other word no one owns the data in the blockchain and there is no third party that can reveal their secrets.
- **Crypto jacking:** A lot of people and even gamers fell into the trap of free games, free movies, free books and so on. Crypto jacking is also known as computer hijacking, it is the practice of using a user computer' resources to mine cryptocurrencies against their will.
- **Illegal content posting and blogs:** In case of we talk about decentralized social networks and decentralized blog platforms the problem with that is any user can post illegal and violet content and unwanted content and the main problem here is that the user is unknown, and no one can track. But in case of not showing the unwanted content, some AI modeling with the mix of front-end technologies this content can be hidden from other users and user that was responsible for posting this content can be banned from such a platform.

### 1.8.4 Smart contract as a limitation

While smart contracts are a good feature of the blockchain technology, they are still a big threat for large numbers of enterprises and companies, since once a smart contract is deployed on the blockchain it can't be altered or the code cannot be changed anymore, that's why smart contracts that are about to get deployed on the blockchain (main nets) should be tested well and deployed first to private/local blockchain and make sure that there is no vulnerability to hack them or no bugs. We can take an example of **DAO (decentralized autonomous organization) hack**. This organization was launched in 2016 by one of the Ethereum protocols engineers on top of the Ethereum blockchain. The organization was launched after they raised \$150 million USD worth of ether (ETH) through a token sale. As a result of the weak programming foundation of the smart contract codes that were deployed, the DAO was attacked and hacked. At that time to recover the stolen funds, the Ethereum blockchain finally underwent a hard fork. However not everyone in the network agreed with



this choice. For the DAO attack even before the DAO started to sale the tokens, computer programmers and scientists were afraid that there was a bug in the DAO's smart contracts that would allow them to be drained. While developers attempted to fix the bug, a hacker took advantage of the bug and started the attack. [24]

## 2 POPULAR BLOCKCHAINS AND CRYPTOCURRENCIES

It is necessary to talk about different types of public blockchains since they are free to join and because everyone on the network can read and write ongoing activities on it. Public blockchains can be extremely useful because they can be used as the backbone for almost every decentralized solution. When mentioning public blockchains, a developer or someone that knows about blockchain technology, the first thing that will come to his mind is the following blockchains (Bitcoin, Ethereum). These can also be referred to as cryptocurrencies because these technologies/cryptocurrencies have their own blockchain that's why they are also referred to as blockchain. In this section the focus on these blockchains will lie and mostly focusing on Ethereum since the application in the practical part will be built on that blockchain.

### 2.1 Bitcoin

When talking about Bitcoin, it also refers to it as a cryptocurrency. Bitcoin has its own public blockchain, that is working under the concept of proof-of-work. Bitcoin was introduced to the public in 2009, and from that time it became the most famous cryptocurrency. At that time the first Bitcoin block was mined, and it is called Block 0. It is also commonly famous as the "*genesis block*". Bitcoin is easy to understand as a digital currency, if someone owns a Bitcoin, he/she can use the cryptocurrency wallet to send specific amount or small portion of it for the purpose of getting goods or services. Bitcoin as a blockchain technology uses the SHA-256 hashing algorithm to encrypt the data that is stored on the chain. The mechanism of blocks linking in the Bitcoin blockchain technology is that whenever a transaction happens on the blockchain, the information from the previous block is copied to the new block with the new data that is encrypted using the SHA-256 algorithm. The new block can be added to the blockchain only if the transaction is verified by all the miners in the network. When that block is added to the blockchain a Bitcoin is created and given as a reward to the miner(s) who was/were responsible for validating the transaction. About the transactions in the blockchain, all the transactions in the blockchain are placed into a queue to be verified by the validators or the miners on the network, all the miners in the network trying to validate the same transaction at the same time. Those miners use special hardware's and software's to solve the nonce, which is a 32 bits number that is included in the block header, the generated nonce is then compared to the target difficulty, if it meets the requirements then the block is added. The target is used in mining, it is just a number that a

block hash must be below the target number otherwise the block will not be added to the blockchain. The target in the Blockchain network manipulates the speed at which new blocks are added to the blockchain, the target is adjusted automatically by the network to maintain a constant block production rate. [25]

## 2.2 Ethereum

Ethereum is an open source blockchain, it is mostly known for its smart contracts and the usage of smart contracts, it is founded by *Vitalik Buterin*, and it was established in 2015. It is also famous because it is digital/crypto currency **ether** or as known **ETH**. The Ethereum platform was created to be fast, scalable, secure, decentralized, and programmable. From huge enterprises to small ones, the Ethereum blockchain was the choice for building solutions and decentralized applications on top of it because of its features. Bitcoin and Ethereum have a huge similarity, but the main difference is the ability to use smart contracts in the Ethereum blockchain and how the cryptocurrency is rewarded to the miners or the validators. Ethereum firstly was operating on the proof-of-work concept as Bitcoin, but in September 2022 the platform started to operate on the proof-of-stake concept. Proof-of-stake mechanism is known that is more secure and more energy efficient compared to the proof-of-work mechanism. Since the launch of Ethereum blockchain, ether as the digital cryptocurrency now standing as the second largest cryptocurrency in terms of market value, surpassed only by Bitcoin. In Ethereum blockchain there is a single computer embedded in it, and it is called the Ethereum virtual machine (EVM). [26]

### 2.2.1 Connecting to the Ethereum blockchain

All nodes' computers in the Ethereum blockchain relate to each other, and for this connection to happen all of them must run software. Or as commonly known a client, there are different types of clients/software's that's connect the nodes to the real Ethereum blockchain for example:

- **Through web3 providers**, that allow us to interact with the Ethereum blockchain and send/receive transactions and query the blockchain data without the need for running our own full node. These providers provide the user with an API that he can use to interact with the blockchain, an example of these providers is **Infura**, **Alchemy** and **QuickNode**
- **Through running our own node**, this process involves directly to communicate with the Ethereum blockchain without the need for third party services, this operation includes

downloading and syncing the entire Ethereum blockchain on the computer, to run our own full node it requires to use software's/clients such as Geth, Parity.

- **Through digital wallets**, such as **MetaMask, Brave Wallet, Web3Auth, Coinbase**. Usually, these wallets let the nodes connect to the Ethereum blockchain through a web3 provider.
- **Through a blockchain explorer**, such way will not let us send/receives transactions, but rather only viewing transaction details and querying the blockchain data.

Running our own node of the Ethereum blockchain have huge benefits, such as:

- Allows us to directly communicate with the Ethereum blockchain.
- Decreasing the reliance on a third-party service and this means direct communication with the chain.
- Sending and receiving transactions to/from other accounts and nodes on the chain.
- Interacting with smart contracts and querying blockchain data.

staying connected and active with the other blockchain nodes, requires an online connection.

Each client that the node runs has its own implementation of the EVM. [26]

### 2.2.2 Ethereum smart contracts

As mentioned in *1.5.6*, smart contracts are just a piece of code that is executed by the Ethereum blockchain engine or virtual machine. For the code/contract to be able to get deployed and run on the blockchain the process has several steps such:

1. Writing code using a programming language supported by the Ethereum blockchain such as **Solidity**.
2. **Code compilation:** The solidity code should be compiled into an EVM bytecode that can be used by the Ethereum virtual machine. The byte code represents the smart contracts with its functions, states, mapping and more. A bytecode is a collection of machine-readable instructions that can be executed by the Ethereum virtual machine. An example of Ethereum virtual machine byte code can be:

```
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe60806040523
48015600f57600080fd5b506004361060285760003560e01c8063037a417c14602d575b
600080fd5b60336049565b6040518082815260200191505060405180910390f35b60006
00190509056fea265627a7a7230582050d33093e20eb388eec760ca84ba30ec42dadbd
e8edf5cd8b261e89b8d4279264736f6c634300050a0032
```

A bytecode is not a human readable code, but it is readable for the machine. A bytecode is in the form of hexadecimal. Each opcode (operation code) in the byte code corresponds to a specific operation that the EVM can perform, such as adding two numbers and memory

access and control flow. Each opcode is represented by a single byte (8 bits) and has a corresponding hexadecimal value, some opcodes may also have associated data, which is used as input for the operation. The length of the EVM bytecode vary depends on the complexity of the smart contracts. The byte code of smart contract is really the smart contract but in other form that will be deployed on the blockchain, so the owner that is responsible for deploying the smart contracts needs to make sure that there are no bugs, vulnerabilities in the code because it might seriously affect the smart contract's integrity and security as well as the security of the entire Ethereum network.

- 3. Execution of the byte code by the EVM:** The owner of the contract will send a transaction object to the specified Ethereum net (main or test or local one) that contains the bytecode (in other word the smart contract) and gas limit and other data that is required for the deployment of the contract. The transaction object is signed by the contract owner using his private key. The transaction is subsequently broadcast to the network, where the nodes in the network process it. The nodes verify the transaction's validity by examining its bytecode, gas limit, and other metadata. The transaction is added to the pool of pending transactions if it is valid, and miners then compete to process it by adding it in the following block. Once the transaction is included in a block and the block is added to the blockchain, the contract is deployed to the network and becomes available for use. We can take an example of how the EVM reads and execute the smart contract's bytecodes. Let's say we have the following contract as shown in *Figure 7*.

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract SimpleContract {
6     uint public num;
7
8     function setNum(uint _num) public {
9         num = _num;
10    }
11 }
```

*Figure 7. Solidity smart contract example*

When this contract compiled using the Solidity compiler, the bytecode of the contract will be obtained, which is a long hexadecimal number, and it will look something like the following:

```
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe60806040523
48015600f57600080fd5b506004361060285760003560e01c8063037a417c14602d575b
600080fd5b60336049565b6040518082815260200191505060405180910390f35b60006
00190509056fea265627a7a7230582050d33093e20eb388eec760ca84ba30ec42dadbd
e b8edf5cd8b261e89b8d4279264736f6c634300050a0032
```

The EVM when it receives the byte code it starts to read them, one instruction at a time, each instruction in turn has a specific code that tells the EVM what operation to perform. For example, the first opcode in the bytecode **0x60** or **60**, tells the EVM to push a 32-byte value onto the stack. Followed by **0x80** that tells the EVM to duplicate the top stack Item. The EVM continues to execute each opcode in turn until it reaches the end of the bytecode. Once the bytecode has been fully executed, the contract is deployed and is ready to be used. Each opcode or instruction in the Ethereum engine has a specific amount of fee to be executed, so the owner of the contract needs to have an enough amount of ETH to be able to deploy the contract.

- 4. Deployment of the contract:** For the deployment of the contract, after the smart contract gets compiled, two types of bytes code are obtained. **First** is the *full byte code* which contains the whole contract. **Second** is the *deployed byte code*, the deployed byte code is the one that will be deployed to the network, it has only the necessary information about the contract for the purpose of execution.

### 2.2.3 Ethereum virtual machine (EVM)

The Ethereum virtual machine is the heart or the engine of the Ethereum blockchain, it is considered as the run time environment for the smart contracts and decentralized applications on the blockchain, it is a decentralized virtual machine that compiles and runs codes written in the Solidity programming language or any other languages that's support it. The Ethereum virtual machines act as a software platform or as a virtual computer that is used by the developers for creating decentralized applications, smart contracts. The Ethereum virtual machine also works as a huge database for keeping the state of all the accounts and balances on the Ethereum blockchain. The reason behind the word "machine" is because this "Virtual computer" has the ability for executing machine code. The EVM has its own instruction for executing machine codes called "opcodes". All smart contracts codes get compiled into instructions called **bytecode** which can be executed by the **EVM**. The EVM is also known as a "state machine" which is not a real machine but rather it is a huge data structure of wallets, balances and even sets of rules, like how to make/add new blocks,

and how much gas fee for the execution of each operation. Each blockchain provider such as web3 provider or clients such as Geth or parity has its own implementation of the EVM, which is used to run codes on the Ethereum network. [27]

#### 2.2.4 Ethereum state

Ethereum state is a large data structure, that holds all the information in the blockchain such as accounts information and balances. Each account has its own address that is a reference to it. When a computer runs a blockchain provider/client it will be responsible for adding this computer as a node to the network. When a computer becomes a node in the network as shown in the *Figure 8*, this node will begin to download the whole blockchain network data and start verifying the validity of each block in the network. This data or entire blockchain data is known as the “**Ethereum world state.**” As mentioned before, this Ethereum world state contains all the information about accounts, balances, and smart contracts. If a new block is added to the Ethereum network the client will be responsible for updating the local copy of the Ethereum world state. This will ensure that the node is always up to date with the Ethereum blockchain. Each web3 provider or client has its own implementation of storing the world state. [27]

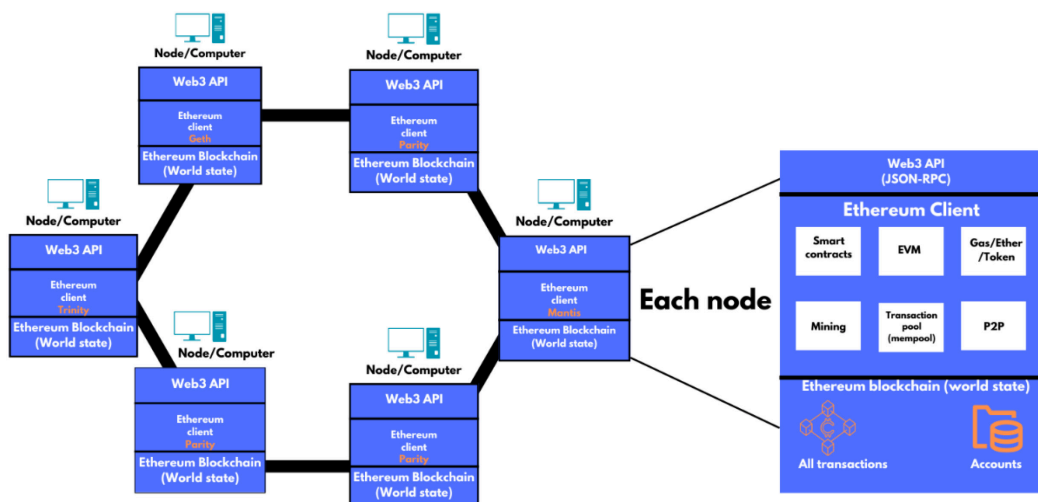


Figure 8. The Ethereum world state and nodes

### 2.2.5 EVM storage

Generally, there are three types of storage in the Ethereum virtual machine:

1. **Normal storage (Account storage):** This type of storage is used for persistent data, such as transaction data, or even personal information. Account storage can be accessed and modified only by the account owner and smart contracts authorized by the owner.
2. **Contract storage:** The data and the information of contract storage can be accessed only by the smart contract owner that owns it; this type of storage is different from normal/account storage.
3. **Memory:** Memory storage is like the RAM in the computer, data stored is used for run-time execution so after the contract finishes the execution this data will be whipped out. So, the data in the memory is used for computational purposes only.
4. **Stack:** Is the type of storage that is responsible to store data from the bytecode (the instructions), the stack has the LIFO data structure (Last in first out), for example from the long byte code we can take this example „0x30“ this is an instruction that will be stored in the stack to be executed by the EVM, each instruction will be pushed to the stack and then start executing from the last inputted one (LIFO). The stack can hold 1024 elements, comprised of words of 256 bits (32 bytes). [27][28]

### 2.2.6 EVM gas fees and execution process

So, the discussion before has covered the types of storage in the EVM and the concept of bytecode. It is necessary now to examine how the instructions from bytecode are read and executed by the EVM and the associated costs. The Ethereum blockchain has its own set of opcodes instructions and what each instruction and byte code mean and how much gas fee each cost. Looking at a specific part of smart contract's byte code such as “0x6080604052348015600f57600080fd5b50604580601d6000396000f3fe”

The EVM executes byte by byte the bytecode, such as the first 60 byte. According to the Ethereum yellow paper 60 stands for “PUSH1” and means place 1 byte item on the stack so the following byte 80 will be placed on the stack. So, it is like “PUSH 0x08”. The part 0x6080604052 is almost exists on every smart contract byte code, this bytecode is a hexadecimal representation of the Ethereum Contract Application Binary Interface (ABI) encoded constructor. Almost every contract has a constructor. And for “0x34” when it gets executed will results or will output how much Ethers we are sending with the transaction. Talking about the gas in the Ethereum, gas indicate it name, it is the fuel of the Ethereum blockchain, whenever a contract or a transaction is being executed on the blockchain a



specific amount of fee/gas should be paid to the network, so that the network keeps operating. Gas in formal terminology is the unit of measurement for the computational effort required to execute an operation or smart contract on the Ethereum blockchain. For example, the execution of a contract required 3 gas units. And each unit price is paid with the small Ethereum currency unit which is *gwei* ( $1 \text{ Gwei} = 0.000000001 \text{ ether}$ ), the calculated amount of gas for an operation varies depending on how complex is the operation and, and the *gwei* required for each gas unit vary on the market and the Ethereum blockchain network activities, in general the more crowded the network the higher the gas price will be. If we look for how much does the "PUSH1 0x80" operation costs, according to the yellow paper and Ethereum blockchain official documentation website as shown in *Figure 10*, the PUSH1 opcode has 3 gas units and an additional gas cost for each byte of data being pushed, so for a single byte of data pushed using the PUSH1 opcode, the total would be 3 gas units. And as date of today "*Friday, April 7, 2023 11:15 PM*" the cost for 1 gas is as shown in the *Figure 9*.

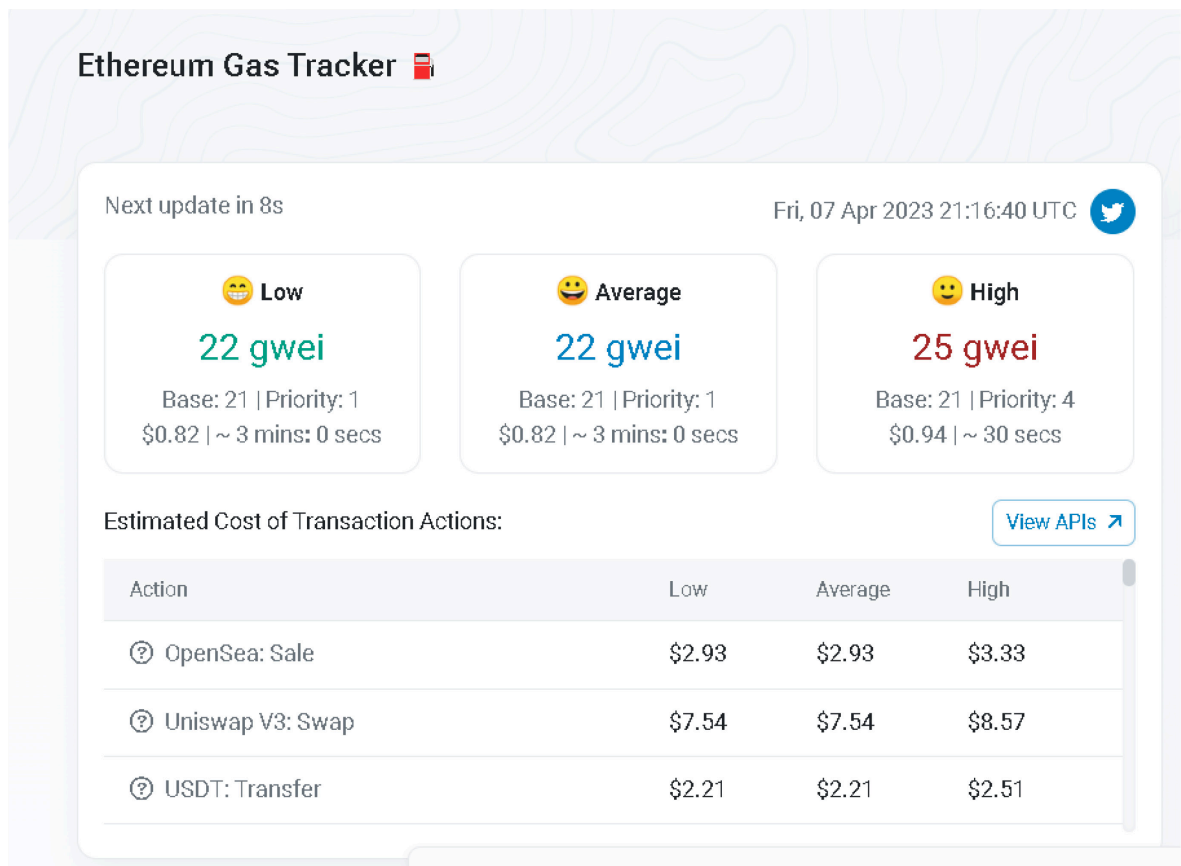


Figure 9. Ethereum gas price<sup>6</sup>

<sup>6</sup> Ethereum gas price tracker: <https://etherscan.io/gastracker>

As we can see in the *Figure 9*, there is **low**, **average**, and **high**. Low and high gas prices represent the minimum and maximum gas prices paid for transactions included in the last few blocks. The average gas price is the median of the gas prices that have been paid for these transactions. Users may choose to select a higher gas price if they want their transactions to be confirmed right away, while users who are patient may choose to set a lower gas price. [28][29]

FOR THE COST OF THIS INSTRUCTION.				
$\mu'_s[0] \equiv \mu_s$				
0x5b	JUMPDEST	0	0	Mark a valid destination for jumps. This operation has no effect on machine state during execution.
60s & 70s: Push Operations				
Value	Mnemonic	$\delta$	$\alpha$	Description
0x60	PUSH1	0	1	Place 1 byte item on stack. $\mu'_s[0] \equiv c(\mu_{pc} + 1)$ where $c(x) \equiv \begin{cases} I_b[x] & \text{if } x < \ I_b\  \\ 0 & \text{otherwise} \end{cases}$ The bytes are read in line from the program code's bytes array. The function $c$ ensures the bytes default to zero if they extend past the limits. The byte is right-aligned (takes the lowest significant place in big endian).
0x61	PUSH2	0	1	Place 2-byte item on stack. $\mu'_s[0] \equiv c((\mu_{pc} + 1) \dots (\mu_{pc} + 2))$ with $c(x) \equiv (c(x_0), \dots, c(x_{\ x\ -1}))$ with $c$ as defined as above. The bytes are right-aligned (takes the lowest significant place in big endian).
⋮	⋮	⋮	⋮	⋮
0x7f	PUSH32	0	1	Place 32-byte (full word) item on stack. $\mu'_s[0] \equiv c((\mu_{pc} + 1) \dots (\mu_{pc} + 32))$ where $c$ is defined as above. The bytes are right-aligned (takes the lowest significant place in big endian).
80s: Duplication Operations				
Value	Mnemonic	$\delta$	$\alpha$	Description
0x80	DUP1	1	2	Duplicate 1st stack item. $\mu'_s[0] \equiv \mu_s[0]$
0x81	DUP2	2	3	Duplicate 2nd stack item. $\mu'_s[0] \equiv \mu_s[1]$
⋮	⋮	⋮	⋮	⋮
0x8f	DUP16	16	17	Duplicate 16th stack item. $\mu'_s[0] \equiv \mu_s[15]$

Figure 10. Ethereum yellow paper [29]

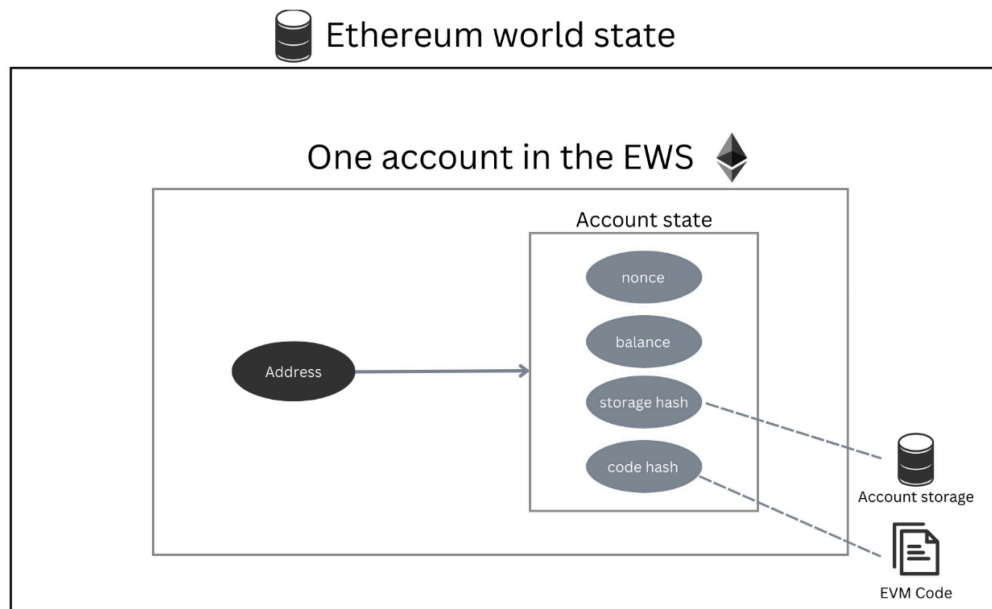
### 2.2.7 Ethereum node variations

There are different types of Ethereum blockchain nodes, and the most commonly known ones are **full nodes**, **light nodes**, and **Archive nodes**. Full nodes from their names indicate that's they store the full blockchain data and participate in the network by validating transactions and blocks. Once the full nodes fully synced with the Ethereum network, can query all the Ethereum blockchain data. Light nodes from the other hand are much smaller than the full nodes, and the main difference from the full nodes that they do not contribute to the process of validating the blocks, these nodes can only query data from the blockchain,

they don't store the state. Archive nodes are nodes that maintain storage of historical blockchain states, these nodes already have the information stored in a local storage and more efficient from the side of performance when dealing with this type of requests. [30]

### 2.2.8 Accounts on the Ethereum blockchain

Accounts on the Ethereum blockchain are like the accounts that we have from the banks. But for ethers cryptocurrencies, where the digital currency can be transferred or held to other accounts, the main difference is that these accounts can be used to execute smart contracts in the Ethereum blockchain. Accounts in the Ethereum blockchain are simply objects or entities that are stored in the Ethereum world state as shown in the *Figure 11*. Each account on the Ethereum blockchain has the following properties, **Address**, address is a specific number that references the account in the Ethereum blockchain network, each address in the Ethereum network is unique and immutable. An address is a number that consists of **20 bytes** (40 digit each two numbers represent one byte) and it is in form of **hexadecimal**, this address is stored in Ethereum state. "0xb794f5ea0ba39494ce839613fffa74279579268." Is an example of an account address. Where "0x" represents the hexadecimal format. **Balance**, balance simply indicates how much eth the account has such as **10eth**. **Nonce number**, nonce number describes how many transactions are made from the specific account.



*Figure 11. Contract based account in the Ethereum world state*

### 2.2.9 Types of accounts in the Ethereum blockchain

There are two types of accounts in the Ethereum network, *Externally owned account* and *contract account*. Externally owned accounts as shown in the *Figure 12* are the most basic accounts in the Ethereum network, it functions similarly to a Bitcoin account. The address of the externally owned account is created using private key and public key and then both keys are hashed. Creating such account costs nothing and can initiate transactions. These accounts are also controlled by the users, this control usually happens through software such as a wallet application. Externally owned accounts are simple accounts without any associated code or data storage. A normal person can have as many as accounts he wishes. An account is created whenever a wallet is created, and it is made with a private key this key can be used to:

- Check balances.
- Send and receive transactions.
- Establish and create smart contracts.

Transactions that are made from an external account to a contract account can trigger codes that can execute many different actions, such as creating new contracts or different types of functionalities.

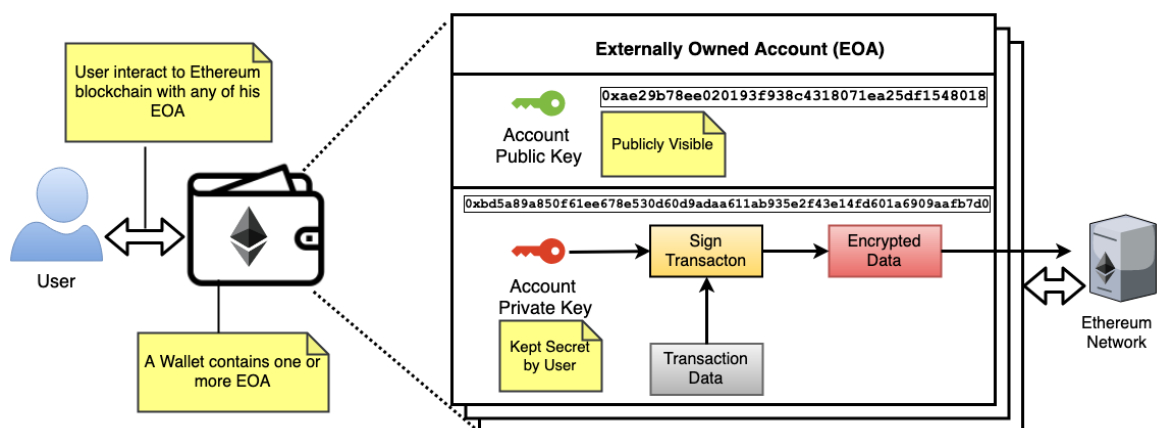


Figure 12. Ethereum externally owned account [30]

From the other hand contract accounts as illustrated in the *Figure 13* is controlled by contract code and surely a contract account can do all the functionalities that an externally owned account can do, contract accounts unlike an externally owned accounts they are created when a contract code is deployed, while externally owned accounts are formed when a digital

wallet is created. Contract accounts are governed by the contract codes. The good thing about contract accounts is they can have some storage data (storage hash) to store some data. Creating a contract account has some disadvantages such as the gas fee for each a contract account it is required to pay some gas fees to the network. These fees are used for storage resources and valuable computation power. Contract accounts can't trigger a transaction on their own, instead they can only trigger them responding to other transactions. [30]

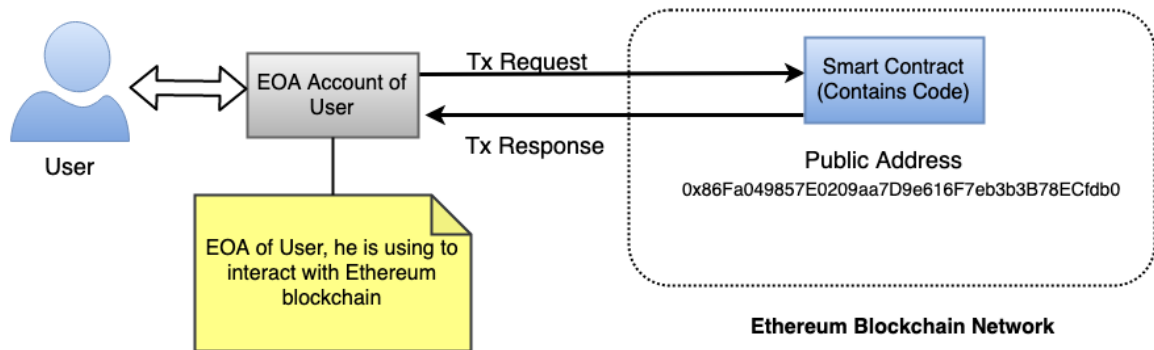


Figure 13. Contract account on the Ethereum network [30]

### 2.2.10 Ethereum drawbacks and limitations

Ethereum still has some downsides and some disadvantages such as:

1. **Slow transaction speed and speed in general:** The Ethereum network has an average speed of 15 TPS (Transactions Per Second), compared to Bitcoin it is not that difference, Bitcoin has an average of 7 TPS, that's why sidechains were invented to improve such limitations if we look at Polygon, it has an average of 65000 transaction per second. Slow transaction speed happens mostly because the network is too congested and because of the mechanism that the networking is operating on (POS, POW). In the case of Ethereum and because it uses POS the more crowded the more a node must pay gas fee to make his transaction accepted faster.
  2. **Expensive fees:** Ethereum is also known for the high gas fees that are requested for the transactions and for the deployment of contracts, a simple transaction on the network can cost up to 5\$. while a complex transaction including smart contract could cost between \$50 or \$100 to execute. Simple transactions between Alice and Bob can be reasonably priced. For most of Ethereum's existence, it was approximately \$0.10. A complicated smart contract with numerous phases will cost more because it needs more processing power.
- [31]

### 3 SIDE CHAINS

A side chain and layer 2 chain are a sperate blockchains that run independently of the main blockchain. They are both methods for scaling blockchain networks and improving their efficiency, transactions speed and more. But both differ from how they implement these improvements.

#### 3.1 Side chains

Sidechains and Layer 2 chains are both methods for scaling blockchain networks and improving their efficiency. However, they differ in their implementation and purpose. Sidechains are designed to provide an alternative platform for transactions, allowing users to transfer assets and data between the main chain and the sidechain as shown in the *Figure 14*. They can be used for experimentation and innovation and offer benefits such as improved scalability and faster transaction processing. On the other hand, Layer 2 scaling solutions are designed to build additional layers on top of the main chain, enabling faster and cheaper transactions, while maintaining the security and decentralization of the main chain. [33]

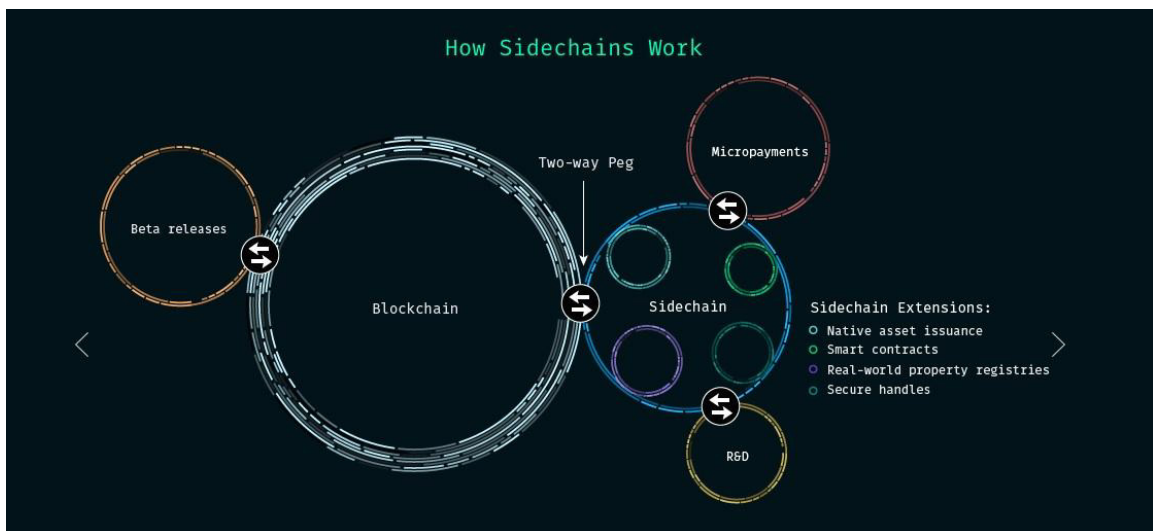


Figure 14. Side chains architecture [33]

#### 3.2 Potential of side chains

The main benefits of side chains are *adaptability to change*, *experimentation*, and *diversification*.

### 3.2.1 Adaptability to change.

Through various optimizations, such as shifting a certain type of transaction to another chain using a protocol designed just for that sort of transaction, a sidechain can provide faster and less expensive transactions. As a result, the first chain should become less clogged and speedier as well as more affordable. Sidechains can also make use of newer, more rapid, and effective methodologies.

### 3.2.2 Updates and experimentations

Sidechains allows new ideas and features to be tested and deployed without requiring a broad consensus from all stakeholders. This means that the developers can test and experiment with new ideas and features on the sidechain before implementing them on the main chain. Sidechains can be used to offload certain types of transactions from the main chain, freeing up space for more important transactions. [32][33]

## 3.3 Examples of sidechains

There are many examples of sidechains currently, but the most interesting ones are **Drivechain**, **SmartBCH** and **Polygon**. Talking about Drivechain, it is an example of a second type of sidechain with relation of “parent-child”. Bitcoin is the parent and Drivechain is the child. Drivechain does not have its own native token, instead it fully relies on BTC transferred from the Bitcoin network. Drivechain solves the problem that sidechains must have their own miners, by implementing the so-called blind merged mining (BMM). BMM gives the ability to the miner on the Bitcoin blockchain (the main network) to mine on Drivechain (child) without the need to run a full Drivechain node, and the miner is rewarded in BTC. The aim of Drivechain is to give the people that have BTC on the main network the ability to transfer bitcoins from the Bitcoin network to sidechains and back and this give the bitcoin holders access to variety range of blockchains.

SmartBCH on the other hand is independent from the main chain. SmartBCH is an Ethereum virtual machine (EVM) and Web3-compatible sidechain for **Bitcoin cash (it is bitcoin by with new improvements)**, SmartBCH does not have its own native token. The main aim of such a chain is to support smart contracts and the building of decentralized applications, improving transaction times. Providing a scalable and developer-friendly platform for building DApps. SmartBCH is designed to have low fees on execution of transactions and smart contracts. [32] [33]

### 3.3.1 Polygon

Polygon was created by a group of Ethereum experienced developers in 2017. Polygon was formerly known as the MATIC network (it is currency name). The MATIC network went live in 2020 and was then re announced to become Polygon in Feb 2021. Polygon runs on and alongside the Ethereum blockchain (Polygon is not independent from the Ethereum it is fully dependent on the Ethereum blockchain). Polygon is also considered as layer2 solution (A layer 2 is a different blockchain that extends Ethereum and takes on Ethereum's security assurances.). Polygon uses Ethereum framework called Plasma, which allows the creation of child chains that can process transactions before being finalized on the Ethereum blockchain. Polygon has its own cryptocurrency, called MATIC, MATIC is used to govern and secure the polygon network and to pay network transaction fees. The Polygon network, as a secondary scaling solution, is the goal to address the limitations of the Ethereum network, such as high transaction fees and slow transaction processing speeds. Talking about the Polygon architecture, Polygon is made up of four layers as shown in the *Figure 15*:

1. **The Ethereum layer:** this layer is not mandatory for Polygon to use, but Polygon can use this layer as a base layer for the chain, to take advantages of its decentralized nature and high security, this layer can be used for finality, checkpointing, staking, dispute settlement, relaying messaging between Ethereum and Polygon chains, and more. It is constructed as a collection of smart contracts on Ethereum.
2. **Security layer:** this layer provides the polygon network with a function that acts as validator. It allows Polygon to use sets of validators that can check the validity of the Polygon chain like transactions and blocks for a fee. This layer runs in parallel with Ethereum and can be used for the management of validators.
3. **Polygon networks layer:** this layer consists of multiple networks/chains or sidechains as the name indicates, these chains are running in parallel with the main network. Each chain can have its own functionality such as transaction collation.
4. **Execution layer:** This layer is the most important layer on the Polygon network, it is responsible for the execution of smart contracts and transactions, this layer includes the Ethereum virtual machine and different execution environments that support different programming languages. [34]



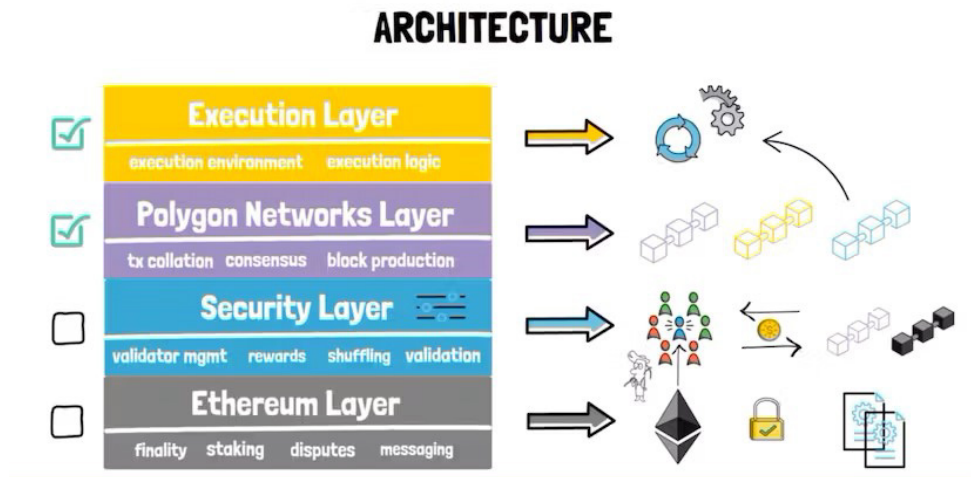


Figure 15. Polygon architecture [35]

### 3.4 Downsides of sidechains

Sidechains also have their cons; it is right they exist to improve the main chain's performance and functionalities, without affecting the main blockchain. But since sidechains are responsible for their own security; a sidechain's security is not inherited from the blockchain it connects with. This can be counted as a positive and a negative feature at the same time since poor security in one blockchain does not affect the security of the connected blockchain. On the other side this means popular blockchains like Bitcoin or Ethereum, cannot lead any security strength to smaller or less popular blockchains. It also applies to sidechains that they need their own miners, sidechains do not inherit the miners from the main chain, and since miners play huge factor in securing the blockchain, this can be a huge downside for sidechains, so a new side chain must do their best to grow the mining ecosystem. If the parent/main network is down or experiencing issues, it can potentially impact the child network, as the child relies on the parent as its base layer. [35]

## 4 BLOCCHAIN AND WEB

In the realm of web development and blockchain, the discussion revolves around decentralized applications, where the data of the users will not be controlled by anyone, meaning no one owns their data or no one can alter or remove their data. Standard social networks such as Facebook or Twitter are centralized applications, in other words means the user data is stored in a normal database or server basically, and this server or database is controlled by some people, it means our data is not fully secured, and is in the hand of someone. So, talking about a decentralized social network meaning that all users on the network can read the information available on the web, and all users are able to add information as far as they comply with the set of protocols, and the transfer of files or any data between any users are done without any third party and it is all done using the blockchain technology.

### 4.1 Web 3.0 vs Web 2.0

In the context of web development, it is important to consider different versions of the web. Specifically, we refer to web development in general, starting from the early days of the web known as web 1.0. During this era, users had limited capabilities and could only read information on the web. Web 1.0 represented the earliest version of the internet, where webpages primarily consisted of static content. These pages were hosted on web servers managed by Internet service providers or free hosting services. Users primarily utilized the web for accessing information and conducting research. It was not possible for users to store data or input any form of information. Essentially, webpages during this time were limited to text and images, offering read-only functionality.

#### 4.1.1 Web 2.0

Web 2.0 is basically the evolution of Web 1.0. The evolutions represent the shift from read only internet to read/write and store internet and moving from no styled pages to styled animated pages using CSS, in this period the social media networks started to appear, video casting, live streaming and much more as shown in the *Figure 16*.

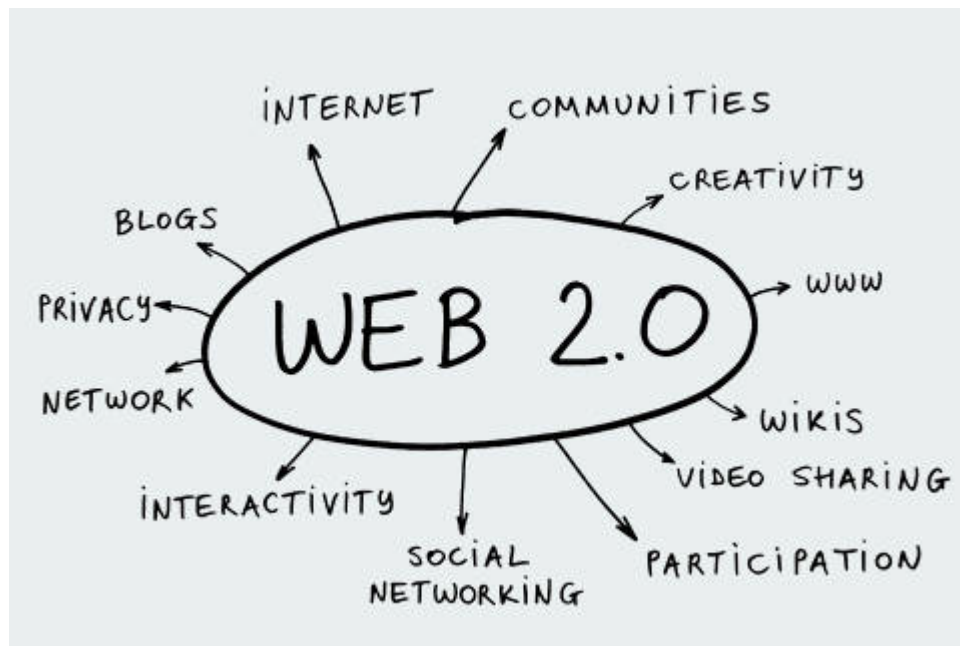


Figure 16. Web 2.0 features [36]

#### 4.1.1.1 How web 2.0 operates

Web 2.0 works in a way where users communicate with a web server where the website is stored in it. But direct communication is not established as in web 1.0, here the web server will process a user request using some kind of back-end language or any server scripts like shown in the *Figure 17*, after that the web server will connect to the database and send the right request and after that the response is send back to the user. [37]

#### 4.1.1.2 Requirements to create web 2.0 Application.

Nowadays the use of web 2.0 is still getting popular even though web 3.0 was introduced because web 3.0 website cannot be built without using web 2.0 technologies. To create a fully compatible web 2.0 application it is necessary to think about front-end technologies and back-end technologies and databases to store the website information and users' information. For the front-end technologies, it is mandatory to use HTML this language build the skeleton of the page this markup language was already introduced in the Web 1.0, then it is important to use the CSS to give some styling to the page the CSS was introduced in Web 2.0 it gives the ability to animate and colorize the website in many different ways, finally to make the website active and feels a life, the job of JavaScript language takes a place here, this language make the website interactive with what the user do, for example when a user clicks on button a dialog say hello should appear. Finally, talking about the

back-end side where the deal with the storing/retrieving of the data happen, the database is basically a place where a user saves his files, messages, pictures and retrieve them later, there are many different types of databases but to communicate with the database a scripting language such PHP, C# ASP.NET Core framework, Express.js framework should be used. These scripting languages or frameworks provide a way to communicate with the databases by retrieving requests from the users.



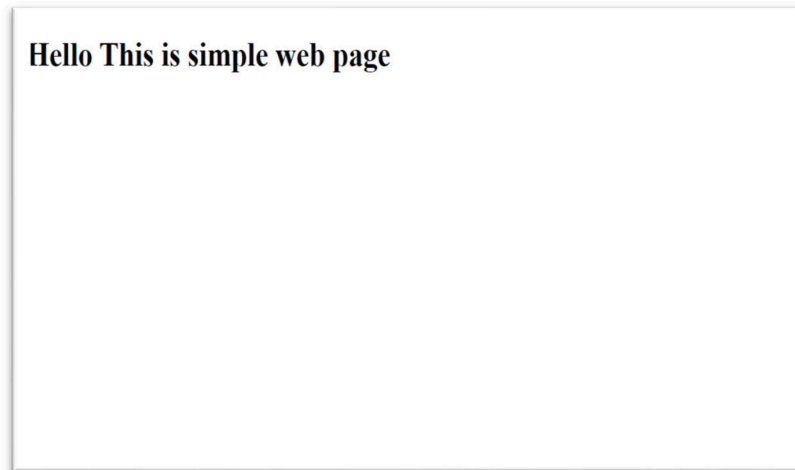
Figure 17. General Communication Architecture[38]

**HTML** is not really a programming functional language, but it is markup language means it works with tags, HTML has many different tags to represent content on the website such as image, heading title, video, paragraph, tables, links and much more. HTML is the standard language for creating web pages, it describes the skeleton of the page. Without HTML there is no website. The simplest code to build a web page is as shown in the *Figure 18*.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>My Web Page</title>
6 </head>
7 <body>
8   <h1>Hello This is simple web page</h1>
9 </body>
10 </html>
```

Figure 18. HTML simple page

This code will generate a simple web page that gives the result as shown in the *Figure 19*.



*Figure 19. HTML code result*

**CSS**, stands for cascading style sheets, it is a style sheet language means does not provide any functionality instead it describes how the HTML elements should appear on the screen, it is possible to write CSS code inside the HTML tag and this way of coding called inline styling or it is also possible to write inside the body of the head tag and in this case this way of writing style is called internal styling, or by creating an external CSS file with the extension .css as external cascading style sheet file, CSS provide us with many styling choices and animations not only this but also it provides the so called media queries which can decides how the web page should appear on different screen sizes or different devices. In the *Figure 20* is a demonstration of CSS inline styling, that results the web page showing in the *Figure 21*.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>My Web Page</title>
6 </head>
7 <body>
8   <h1 style="color: red;">Hello This is simple web
  page</h1>
9 </body>
10 </html>
```

Figure 20. HTML with inline CSS



Figure 21. CSS code result

**JavaScript**, every website that has interactive elements and 2D/3D interactive animation or data manipulation, it can be confirmed that JavaScript was included in that website, according to the w3 organization JavaScript is used as client-side programming language by 98.0% of all the websites, this is a huge percentage. With JavaScript the possibility to write code to select an element that exists in the HTML page and decide what can be done with that element, for example change it is inner content or delete it from the HTML DOM. Talking about JavaScript as a language, JavaScript is an object-oriented programming language, but it is not a class-based object-oriented language such as Java, C++, C# but since JavaScript follows the ECMAScript standard in ES6 JavaScript introduced the uses of classes. JavaScript is also Weakly typed language means that there is no need to declare a variable as integer or string or any other data type kind, but simply creating a variable without datatype and JavaScript will automatically detect the assigned value that is given to

that variable, or declare variable using *var*, *let*, *const* each one of these has their own advantages<sup>7</sup>. To write JavaScript that works on HTML web page, there are two ways either writing internal script inside the head tag body or at the end of the body tag or as external file with the extension .js, the code showing in the *Figure 22*. Demonstrates the internal JavaScript code that will select a button from HTML elements and when clicking on it will show alert, that the button is clicked as shown in the *Figure 23*. So, HTML, CSS, and JavaScript are essential components required to construct a fully functional website. [39]

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>My Web Page</title>
6
7 </head>
8 <body>
9   <h1 style="color: red;">Hello This is simple web
  page</h1>
10  <button class="myBtn">Click Me</button>
11  <script>
12    var myBtn= document.querySelector('.myBtn');
13    myBtn.onclick = function() {
14      alert("Button is Clicked")
15    }
16  </script>
17 </body>
18 </html>
```

*Figure 22. HTML with JavaScript*

---

<sup>7</sup> Differences between var, let, const: <https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/>

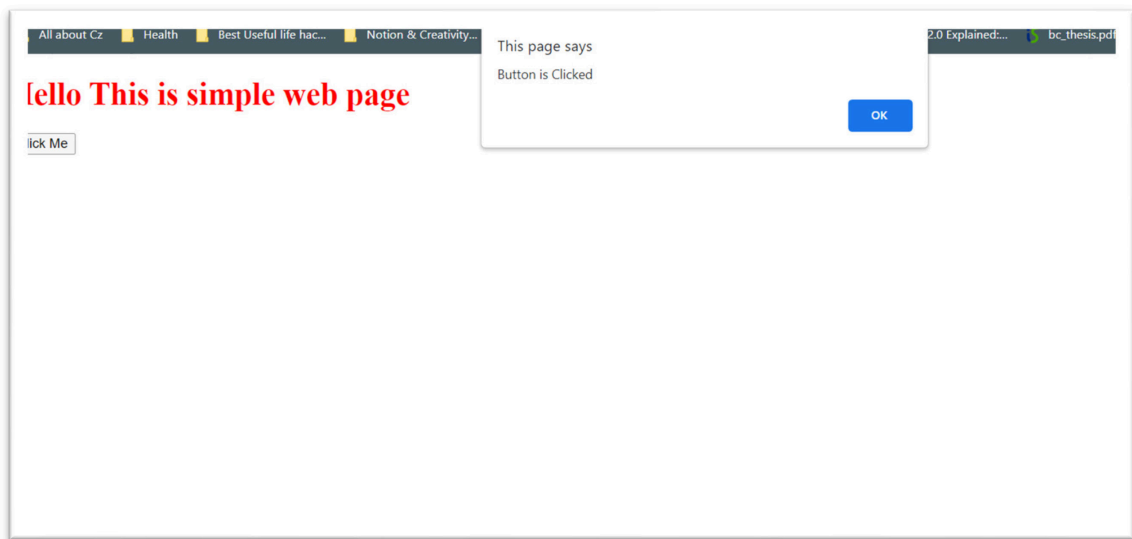


Figure 23. JavaScript Result Code

#### 4.1.1.3 Web 2.0 use cases and applications

Web 2.0 have helped us a lot with different services websites, such as social networks, and Blogs whether it is personal or professional or organizations blogs, Bloggers can simply create web pages and share up-to-date information and users visa-versa can like and share this news or even comment on them. Social media networks provide communication between different kinds of users from different places around the whole world, users can easily share their details with other people. Users can communicate and can also make communities or groups with different types of categories such a group for fitness or for studying or for coding and so on. Videos and image websites were also a huge move from web 1.0 where a user can upload videos on different kinds of platforms with different types of content.

#### 4.1.1.4 Web 2.0 in real life

Talking about social media networks, the popular ones for 2022 include Facebook, Twitter, TikTok, and Instagram. As for websites serving as information guides and sources, Wikipedia stands as a noteworthy example. However, it is worth noting that Wikipedia represents one of the potential risks associated with web 2.0 due to its open-editing nature, allowing anyone to modify its content. In terms of search engines, Google ranks among the most renowned.



### 4.1.2 Web 3.0

Web 3.0 is the third generation of web technologies. Web 3.0 was introduced to solve the fact that the data is centralized, which means the data is in the hand of someone or it is controlled by someone, but Web 3.0 is not only about the decentralization of the data, Web 3.0 will also make the use of machine learning and artificial intelligence to help building more intelligent and adaptive applications. Web 1.0 was a generation where static information was provided to different kinds of users for read only or rarely for write. Web 2.0 was a generation where enabled the user to collaborate between each other and give them the full ability to read and write information. Web 3.0 can be assumed that it will change both how websites are made and how the users interact with these websites. Web 3.0 is considered to be the successor for two older generations.

### 4.1.3 How does web 3.0 operate?

Web 3.0 introduces a paradigm where applications and services leverage blockchain technology while concurrently incorporating the foundational technologies and protocols of Web 1.0 and Web 2.0. The utilization of blockchain technology within Web 3.0 fundamentally challenges the notion of a centralized authority, emphasizing instead a distributed consensus framework. A noteworthy aspect of Web 3.0 is its inherent integration with cryptocurrency, enabling decentralized payment methods for goods and services. These cryptocurrencies are constructed and enabled through the utilization of blockchain technology. In contrast, Web 1.0 and Web 2.0 primarily relied on IPv4 addresses, a protocol that has become inadequate due to the exponential growth in internet users. In the context of Web 3.0, the adoption of IPv6 addresses predominates, facilitating scalability and accommodating the evolving demands of the digital landscape. [40]

## 4.2 Requirements to create web 3.0 Application.

Given the context of decentralized databases, there is no need to concern ourselves with creating a personal database. However, certain situations may necessitate its creation, such as when implementing authentication and authorization using JWT. In most cases, blockchain serves as the underlying foundation. Nevertheless, we still need to address the implementation of front-end and back-end technologies that support interactions with digital wallets and smart contracts. Solidity, for instance, is a programming language commonly used for smart contract development. For digital wallet functionality, options like MetaMask

or other compatible wallets supporting specific network/blockchain types can be utilized. A normal software or application architecture of a DApp should have:

- **Front-end:** Any type of front-end technology that supports the interaction with a web3 provider.
- **Web3 provider:** Any type of web3 provider/wallet that interacts with the blockchain we want to, but the provider should support that blockchain.
- **A smart contract programming language:** To be able to build smart contracts on the blockchain a smart contract programming language should be used. Such as Solidity or Rust or even Vyper, it is also possible to use JavaScript for building smart contracts.
- **IPFS (InterPlanetary File System):** For storing users' files, images, videos in decentralized way IPFS can be used. IPFS is a protocol, hypermedia, and file sharing peer-to-peer network for storing and sharing data in a distributed file system.
- **Back-end (optional) and central database:** Any type of back-end and database technology, the need for them will be in cases of adding additional features to the DApp such as:
  - Managing user accounts and authentication, authorization, the back end can store non-sensitive user information such as usernames, public keys, passwords, JWT tokens (access and refresh token), to ensure that only authorized users can access the DApp.
  - Analytics and monitoring, back-end can be used to track user activity and gather data on how different types of users interact with the DApp, such data can be used to improve the DApp's performance and usability over time.

### 4.3 Social networks

Social network/media is a place where billions of people share their information and daily activities. Such platform has many advantages for the people, including the ability to contact and exchange information with others, to share daily videos and share informative posts and videos with others, keep in touch with our long-distance families and friends. One of the first known social media platforms was introduced in 1997. It was called "SixDegrees", and it is still available<sup>8</sup>. A user on the platform could create a list of friends and send messages. Now as 2023. Social media has evolved a lot and now different platforms are competing between each other's to add more and more features, so as a result it gains more users and more

---

<sup>8</sup> One of the first known social networks: <http://sixdegrees.com>

market value. Some social media platforms currently support the idea of a so-called marketplace where people on the platform can sell their products or secondhand products. This idea helped many people to start growing their own business or find used products. An intriguing aspect of certain platforms like Facebook is their support for real-time gaming experiences within social media. This feature allowed users to engage in multiplayer games, compete with each other, and share their achievements. But all of this amazing feature and sharing of personal data and daily activities and live location, does a normal user know behind the scenes what is happening and where his data is stored, mostly the answer will be no. A normal user will be interested to have a safe password that no one should know and start dive into the network and share all his personal information and personal pictures without knowing how his data are stored or processed. Currently all popular and famous social media are centralized, means there is central authority that owns all the users' data. [41]

#### 4.3.1 Problems with centralized social networks

The main problem with centralized social networks is that all user's data are stored in a central way as shown in the *Figure 24*, means the data is stored somewhere that someone has control over it or in other words stored in a big database server. If we look at a general overview of a centralized social network architecture, we can see we have users and all their interactions, data, activities are sent to a central server or provider in the end.

#### Centralized Social Network

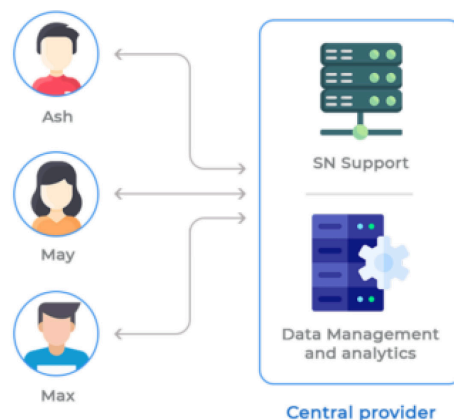


Figure 24. Centralized Social Network [42]

Centralized social media platforms try to capture valuable users' data such as interests, preferences, and activities. Then these platforms try to sell the captured data to markets for the purpose of advertisements, advertisers are the clients of centralized social media platforms like Facebook, Instagram, LinkedIn, Twitter, and YouTube, while user data is their main output. Another huge limitation of centralized social media is the fact that if the central server goes down, all users on the platform will notice this problem, all users will not be able to access their data at that moment. Centralized social media or any central system is exposed to 3rd party attacks, since they have a single point of entry that can be targeted by hackers looking to exploit any weaknesses. Most important topic to mention while talking about centralized social networks is the **freedom of speech and expression**, freedom of speech is the idea that everyone has the natural right to express themselves freely through any medium and without restriction from outside sources (such as censorship) or concern about retaliation (such as threats and persecutions). By barring users or restricting their access to the site for a certain period, most social networks in use today limit users' freedom of expression over certain and sensitive issues. The right to free speech and expression is nuanced. This is since freedom of speech is not absolute and carries with it some duties and responsibilities, therefore it could be subject to certain legal restrictions. However Certain types of speech that certain platforms deem hurtful, or offensive have been made illegal. While in certain circumstances this could be advantageous, it can also raise questions about censorship and the suppression of free speech. Instead of outright banning speech on social media platforms, I believe it would be better if there were government guidelines in place to impose restrictions. Given that, a balance between the right to free expression and the responsibility to protect others from harmful remarks must be found. To make sure that their programs are successful and do not unduly violate people's rights, the government should collaborate closely with social media sites. [43]

#### 4.4 Decentralized social networks

To solve the problems of centralized social networks, it is necessary to solve the main problem of it, which is the idea of someone controlling the data, or the data is stored in a place under authority of someone. Since blockchain is a decentralized storage system, it can be used to solve this problem where data is shared across the network nodes instead of one central server.

#### 4.4.1 Applicability of decentralized social networks

In the context of switching from centralized social networks to decentralized ones, it is essential to mention the problems that such an evolution can solve. Decentralized social networks have the potential to address the following issues:

- **Own the data:** Decentralized social platforms increase privacy, because they enable users to control and own their data, making it more difficult for large corporations or governments to access or misappropriate their information.
- **Security:** In addition, decentralized social networks are less susceptible to data intrusions because user data is distributed across a network of nodes rather than a central server. Users can create accounts without associating them with identifiers from the actual world, such as email addresses or phone numbers. Rather than relying on a single organization to safeguard user data, these networks frequently utilize public key cryptography for account security.
- **Censorship and freedom of speech:** Decentralized social media provides resistance to censorship and supports free speech. As no centralized authority can control or censor content, these platforms are ideal for free speech and expression. Since decentralized social media are largely unmoderated, their dark side can include political misinformation, cyberbullying, and criminal activity.
- **Rewarding users:** A decentralized social media platform can create a reward system. For instance, the platform could implement a system that compensates users with tokens when their posts receive a certain number of likes, remarks, or shares. The platform can create its own native cryptocurrency that can be used to reward the users for their activities.

[44]

#### 4.4.2 General architecture of a decentralized social network

Four or three layers are necessary for a decentralized social network to operate correctly. To explain this in a more detailed way, an example of creating a post on social media as shown in the *Figure 25*. Can be used to discuss the necessary layers for a decentralized social network architecture.

1. **Client-side:** A client side can be an API or a normal web UI application that consumes data. This side acts as the starting point of communication. For example, creating a post on social media will be triggered first from the client side.
2. **Web3 library/provider:** A web3 provider will give the client-side the ability to communicate with the blockchain, provide operation such as sending/receiving transactions

or scanning the content of the blockchain, or verifying if a contract exists and more. For our example the library will be used to communicate with a contract on the Ethereum blockchain to send a transaction that contains details about the post we want to create, the library then in turn will return the result of the transaction and acknowledge whether the transaction is successful or not.

- 3. IPFS provider:** Since the visual content of the social media should also be stored in a decentralized way, IPFS will be used to store user content on that platform, IPFS as explained in [5.5.2](#), it is a distributed file storage platform. Instead of running a node on IPFS network, there are providers that provide services and run their own node in the network. And provides functionalities such as publishing or retrieving content from or to the IPFS nodes. When talking about creating a post, the image or the visual content should be uploaded to the IPFS network, and in turn retrieve the CID and store it in the contract storage on the blockchain. Prior to uploading content to the blockchain, it is important to follow a specific process. This process involves initially waiting for the operation to upload visual content to IPFS (InterPlanetary File System), a decentralized file storage protocol. Subsequently, all the data, including the visual content, is sent together to the blockchain network.
- 4. Additional back end:** An additional back end and central database can be added to the architecture of a decentralized social network, for the purpose of improving the website user experience and improving the website security overall. An example of that can be using JWT token to protect the website routes and protect user contents and avoiding any middleman attack from taking over user actions, also the back end can be used to monitor number of users in the platform. Or handling the errors that occurred from the blockchain, like for example in case of blockchain goes down, it will not affect the overall platform experience.

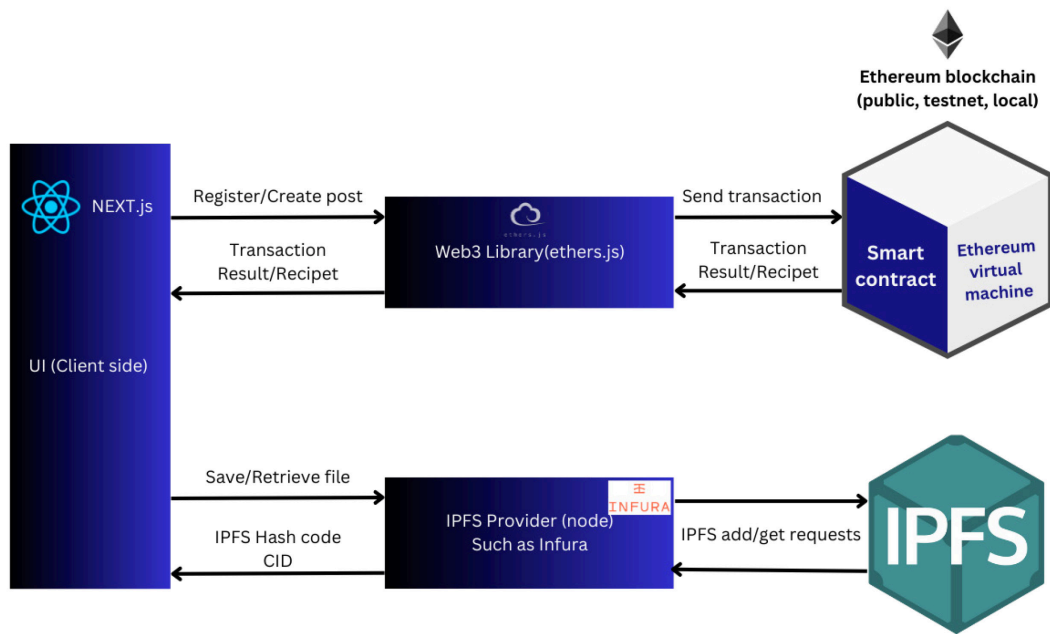


Figure 25. Decentralized social media architecture<sup>9</sup>

### 4.5 Real life examples of decentralized social networks

There are already some published decentralized social networks that are operating and have good amounts of users and have their own rewarding systems. It is anticipated that more decentralized social networks will emerge, each with unique features and expanding functionalities. Such most famous and used platforms will be explained in detail for better overview and which to choose and how they are compared to our current solution. When talking about different decentralized social media platforms we must focus on the following topics mainly:

1. Open source or not.
2. The used blockchain technology and the network.
3. The token or the cryptocurrency used in the platform.
4. When it was published.
5. The user interface of platform and ease of use.
6. If the platform provides a reward system for interaction with the platform’s community.
7. Cost of transactions and interaction on the platform
8. Ease of registration

<sup>9</sup> Decentralized social media architecture: <https://ijarsct.co.in/Paper4979.pdf>

### 4.5.1 Steemit

Steemit is a full open source Web3 social networking site Steemit was created on the STEEM blockchain. Steemit was considered one of the best decentralized social media platforms according to “**Sourceforge**”, also overall ratings/reviews that is done by users is 4.8/5. The STEEM token, is the native cryptocurrency of the STEEM blockchain, is used by the platform to compensate users for posting, sharing, and interacting with content. *Daniel Larimer* and *Ned Scott* started Steemit in 2016. The project's goal is to raise the standard of our online interactions by providing rewards for participation and community control. Talking about the UI of Steemit. Steemit has a comparable UI to Facebook's as shown in the *Figure 26*. It has a user-customizable automatic content stream that users may customize to their preferences. STEEM prizes are available for those that watch and read material. Engagement on the Steemit site determines how STEEM tokens are distributed. The most well-liked or up-voted content's creators get greater STEEM distributions. Users who interact with the platform's content and provide the greatest value also have the possibility to earn more tokens. The best thing about Steemit is that it **uses a zero-fee structure** so that the platform is accessible for all. One of the primary justifications for charging for Steemit social media interactions is that the Steemit blockchain has a "bandwidth-limiting mechanism" to reduce spam assaults. However, network users may increase their SP token holdings to get greater bandwidth. To Register to the platform a guest must provide the (username, phone number, email address) after writing the verification codes sent to email and phone number, the platform will automatically generate a password that will be used for login and generate a public key that is used for transferring and receiving tokens. Other keys are also generated for the purpose of account recovery and social actions and more as shown in *Figure 27*. All the keys that the platform will provide, will be served as a PDF file that will be asking the user to download it. [45][46][47]



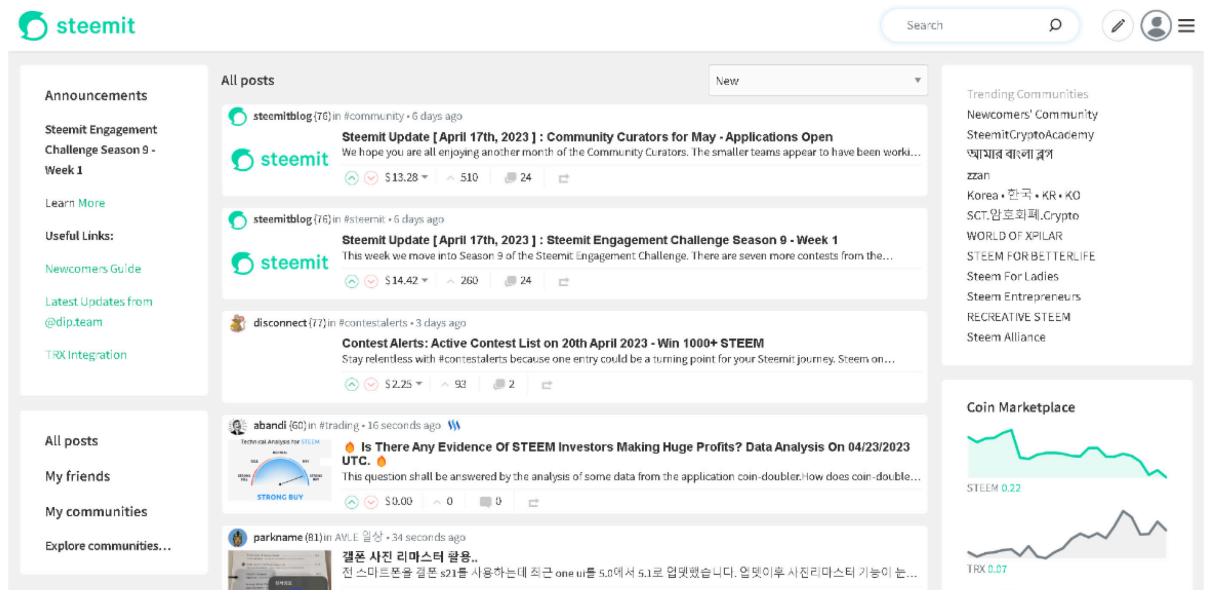


Figure 26. Steemit user interface

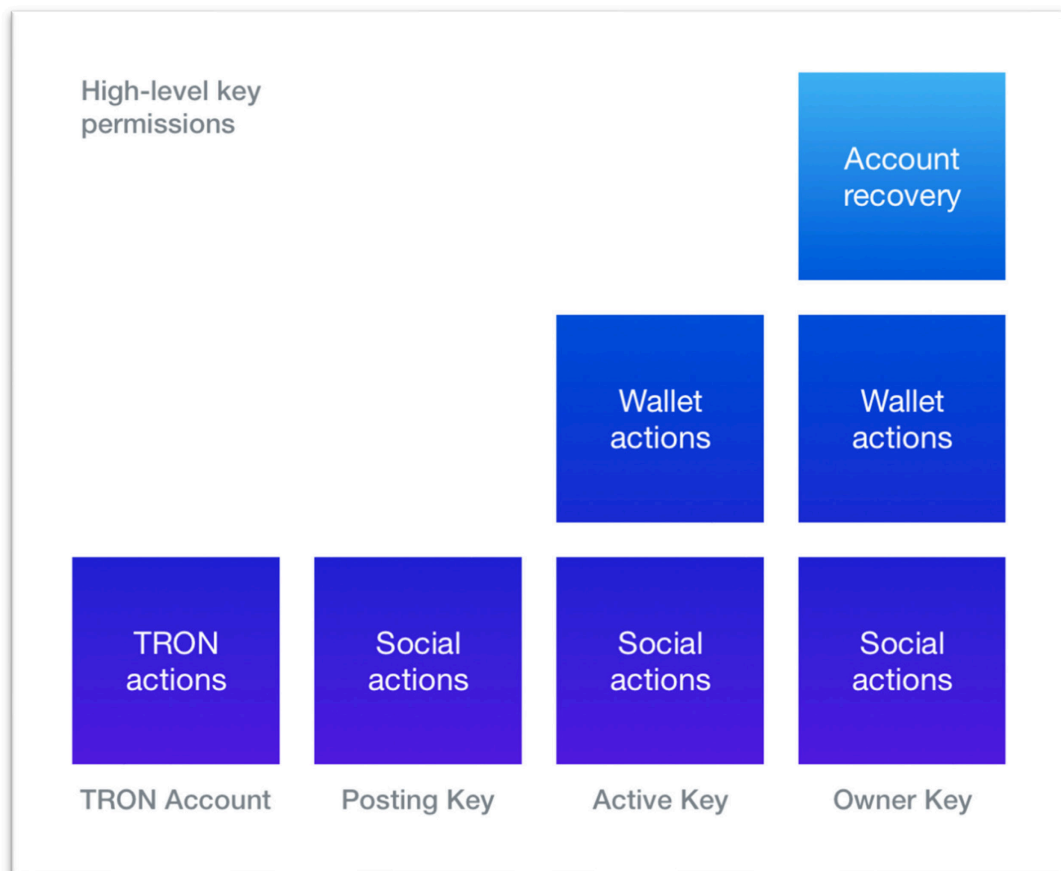


Figure 27. Steemit platform's keys

### 4.5.2 Minds

Minds is a free open source decentralized social media platform, that was founded by **Bill Ottman** in 2011. Minds built on top of the Ethereum blockchain. Minds creates its own tokens, known as *Minds tokens*. Its foundation is the Ethereum ERC20<sup>10</sup> tokens. The Minds' user interface is very user friendly, and it is so easy to navigate between different sections. The UI has a very similar architecture to Instagram as shown in the *Figure 28*. Minds also has its own reward system for creating, upvoting, sharing, and viewing content on the platform. On Minds social media platform, a user can earn rewards in the form of tokens for the contributions. This means that a user will be rewarded for the effort putted on, whether that involves creating exceptional content, being active on the platform, or assisting in the development of the code. Like YouTube, the social media platform Minds features channels. Users may design and administer their own channels on Minds, where they can post material and interact with their audience. Users may subscribe to channels that interest them to get alerts when new material is added, and channels on Minds can be personalized with distinctive branding and design. Minds also give the ability to exchange the tokens that a user owns to promote the content all over the network, or on a specific channel's audience.

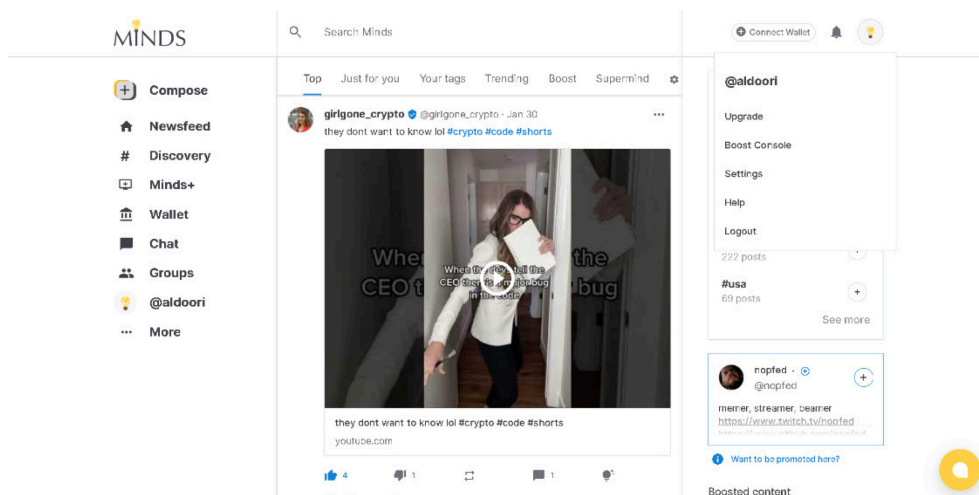


Figure 28. Minds' user interface

Talking about the data that a user must provide to register to the network. (Username, Email, Password) and the user can freely use the platform after the users verifies the code sent to the provided email. Actions on Mind's platform are free of charge, means there are no gas fees for actions like (Posting, liking, or commenting) that a user must pay directly. What

<sup>10</sup> ERC-20 Tokens: <https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/>

Minds do is that when user participate in the platform, they are earning tokens, and these tokens are then used to pay for the transaction fees, every change to the platform requires a transaction, which incurs a small fee in the form of cryptocurrency. Tokens in the platform are rewarded to user at 2:00 every day to **Offchain** wallet. **Offchain** wallets are just wallets that store cryptocurrency off the blockchain, instead of storing and recording balances and transaction on the real blockchain, all of this is made on the platform's wallet (that is considered as a server or database). Off-chain wallets may have several benefits over on-chain wallets, including quicker transaction times, fewer transaction fees, and more convenience. They do not offer the same level of protection and privacy as on-chain wallets, and they do carry some dangers including the potential for theft or hacks. [48][49][50]

### 4.5.3 Lenster and Lens protocol

Lens protocol is a decentralized social media protocol, or social media graph on Web3, built on top of the Polygon blockchain (Layer 2 scaling solution), "*Stani Kulechov*" founder and CEO of **Aave**, invented Lens Protocol based on discussions regarding the significance of digital identity control. On February 7, 2022, Lens Protocol was made accessible on Polygon's Mumbai test net. On May 18, 2022, Lens Protocol moved to Polygon's main net. Lens can be considered as a platform for blockchain developers to launch their social DApps on the top of it. The project seeks to enable Web3-based social media networks and profiles to be launched by developers. Lens protocol can also be considered as a set of smart contracts that are deployed on the Polygon network. Lens protocol does not offer a native token, instead it uses **ERC-721<sup>11</sup> token standard for non-fungible tokens on Ethereum**. Each user or profile created on the lens protocol is presented as a non-fungible token, that can even be sold or traded with. And this is what makes each user on the Lens network unique. Lens protocol acts as an umbrella of all the applications created on top of it. When a user has an account or a lens profile on the lens protocol, this account will be the same in all the applications created using the Lens protocol. This is what makes such a platform powerful. In other words, if they are two different social media websites for different purposes one for sharing videos and one for sharing posts, a lens profile will not need to register twice, a user can login simply with the same account on the two different platforms that are built on top of the lens protocol. Regards the ease of getting a profile on the lens protocol, it is a hard to

---

<sup>11</sup> ERC-721 Ethereum's tokens: <https://ethereum.org/en/developers/docs/standards/tokens/erc-721/>

get an account specially in 2023, A user must claim handle from the lens protocol to be a valid registered user, that can deploy applications on the lens protocol and can login to all different apps created on it. There are currently several ways to claim a handle from the lens protocol:

- Get invited from other DAO (decentralized autonomous organization).
- Buy a Lens profile from *OpenSea*<sup>12</sup> platform.
- Join different web3 discord servers that may give up lens profiles.
- Waiting for the lens protocol to open the access for the community to claim handles.

What makes Lens protocol different is that it makes it so much easier for blockchain developers to build decentralized social media platforms on top of it, like the applications shown in the *Figure 29*, by using their API, instead of writing whole application with smart contract and taking care of security and functionality. The API provides a good collection of end points, such as:

- Authentication using JWT.
- Gasless transactions – some conditions must be met.
- Follow – to follow other users on the platform.
- Create profile end point (only for the test net).
- Get profiles end point.

---

<sup>12</sup> OpenSea platform, lens profiles: <https://opensea.io/collection/lens-protocol-profiles>

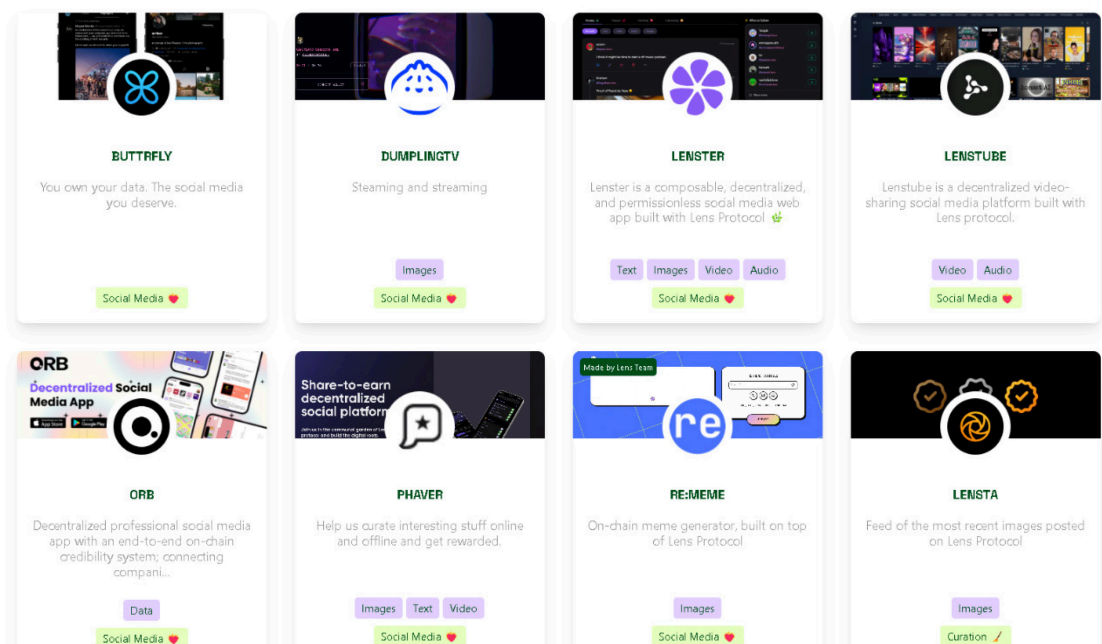


Figure 29. DApps on Lens protocol

Transactions and actions on the lens protocol tend to be free of gas from the user side. The protocol also makes use of a cutting-edge "Gasless transaction" mechanism, which enables users to communicate with the network without having to pay any gas prices. The Gasless transaction method works by having a network of owners who, in return for a nominal charge, pay the gas expenses on behalf of the consumers. people no longer need to have Ether to pay for gas, which lowers the barrier to entry for people interacting with the network. Talking about Lenster social media platform, it is one of the DApp deployed on the Lens protocol. It is a decentralized social media platform; it inherits all the features from the Lens protocol, also provides posting of different type of contents such (Text, videos, audio, images), it has an easy user interface like Instagram and twitter as shown in the *Figure 30*, however Lenster, does not support chatting between different users. [51][52][53]

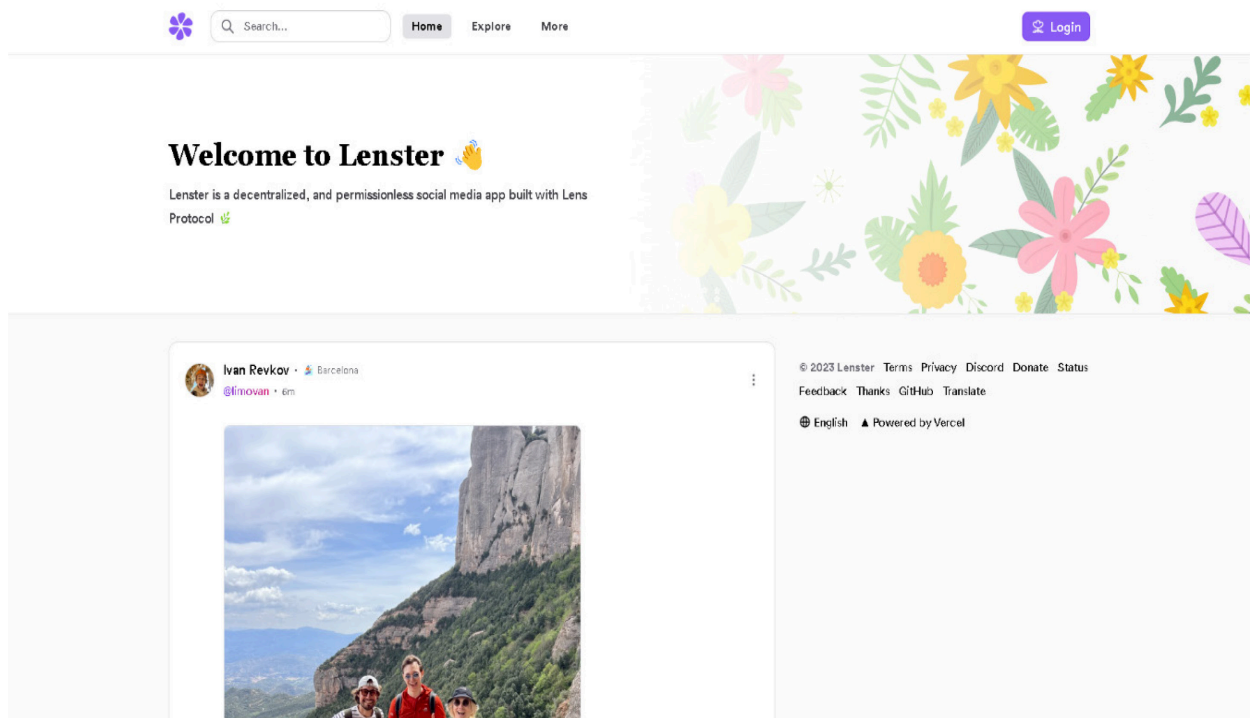


Figure 30. Lenster user interface

#### 4.5.4 Comparison between the different platforms

From the clarification made about each platform in chapters 4.6.1 - 4.6.2 - 4.6.3 above, one can summarize key points and create a comparison table as shown in *Table 2* and in *Table 3* where focusing on functionality and client-side perspective of each platform to gain a clear understanding of them.

Table 2. Decentralized social network comparison (functionality)

Platform	Blockchain or decentralization technology	Blockchain technique	Required registration data	Gas fee
Steemit	Steem blockchain	Delegated-Proof-of-Stake (DPoS)	<ul style="list-style-type: none"> <li>• Username</li> <li>• Phone number</li> <li>• Email address</li> </ul>	Gas-less transaction technique is used
Minds	Ethereum blockchain	Proof-of-Stake (PoS)	<ul style="list-style-type: none"> <li>• Username</li> <li>• Password</li> <li>• Email address</li> </ul>	Gas-less transaction technique is used
Lens Protocol	Polygon layer 2 scaling Ethereum blockchain	Proof-of-Stake (PoS)	<ul style="list-style-type: none"> <li>• Claiming of handle or token (NFT) from the platform</li> </ul>	Gas-less transaction technique is used

Table 3. Decentralized social network comparison (client-side perspective)

Platform	Ease of Registration	User-Friendly	Reward System and is transferable to Ethereum coin
Steemit	Easy, require understanding of tokens and keys	Kind of, like Facebook but with different structure.	Support reward system, tokens can be transferred to ETH or can be used to buy ETH
Minds	Very easy like other social media platforms	Very, like Twitter and Instagram.	Support reward system, token can be transferred to ETH since already built on Ethereum
Lens Protocol	Difficult, obtaining a token from the platform is challenging and require a good knowledge in web3	Depends on the deployed app.	Support reward system, tokens or profiles can be sold to others. And traded with ETH

#### 4.6 Drawbacks of decentralized social networks

As much as decentralized social media platforms provide benefits and solutions for centralized social media platforms problems, it still has some drawbacks, but these drawbacks can be addressed and solved in the future. These drawbacks can be:

- 1. Complexity:** Decentralized social networks are more difficult to use and comprehend than conventional ones, which acts as a barrier to adoption. Non-technical users are still intimidated by complex user interfaces and the need to delve into the realm of cryptography. For example, dealing with a digital wallet is something completely new for non-technical people or old people in general.
- 2. Lack of knowledge:** A lot of people still do not know about blockchain technology and what benefits it has over another technology.
- 3. People attraction:** Most people are acclimated to using major social networks such as Facebook, Twitter, and Instagram and may be reluctant to transfer.
- 4. Scalability challenges for decentralized social networks:** Centralized or conventional social media platforms typically necessitate high throughputs to support constant, rapid social interactions and proper operation. For decentralized social networks, scalability is a



problem because their decentralized nature limits their capacity to manage significant volumes of traffic and data. The speed and performance of the platform can be affected by the network that the platform has been deployed to. If Ethereum was the chosen one then transaction speeds are slow, that's why there is layer 2 blockchains that have much higher transaction speed per second.

5. **Crypto volatility in decentralized social networks:** Blockchain-based decentralized social networks with native crypto economies may be susceptible to cryptocurrency market volatility and may be affected by unanticipated occurrences. The market situation can rapidly affect the value of rewards earned by content creators and the stability of the social network.
6. **Lack of funds:** Moreover, a dearth of funds may result in the social network being shut down. In turn, this will cause consumers to lose their social connections. Platforms with sustainable economic structures are the solution. Using decentralized storage systems such as the InterPlanetary File System (IPFS), social networks can safeguard user data against exploitation and malevolent use.

[44]

## **II. PRACTICAL**

## 5 APPLICATION ANALYSIS AND TECHNOLOGIES USED

To showcase the practical application and advantages of decentralized social media, we will undertake the development of a customized platform with enhanced security features and unique characteristics.

### 5.1 Application/project description

First, it is important to provide a brief description of the application and the naming of it. The application is a decentralized social media, and its purpose is to eliminate the idea of letting someone own the data of the users in such a platform, by saving users' data on the blockchain. The application will have the name **“Social chain”**

### 5.2 Application overview and requirements

Whenever a software is created or an application, before the step where the coding part takes place or the UI takes a place, it is important to step back and setup the requirements of the application that must be fulfilled and create an outline that the developer will work on, or we can say in other words the goals that must be fulfilled. There are special softwares to do that and special methodologies such as using UML (Unified modeling language) diagrams with the software **“Enterprise architecture”**. This software gives the ability to model the architecture of an application and address what is needed and must be accomplished in the application before taking an immediate action. This step must take a lot of time and must be taken in the correct way and must be reviewed from time to time, since it is the basis of the application.

#### 5.2.1 Application functional requirements

When talking about the functional requirements, we refer to what is the application's functionality and what such an application can do, it is more about actions.

##### *System access*

- The application must provide signup/login/logout functionality.
- The application must support interaction with the MetaMask digital wallet.
- The application must prevent unregistered users from accessing the website's feed page or home page.

- The application must provide a remember me feature for the user so when next time he logs in he does not have to provide credentials (only in case of JWT refresh token is not expired).

### ***Post management***

A registered user:

- Can create/delete/edit a post on the platform.
- Can report a post.

### ***Comment management***

A registered user:

- Can create/delete/edit comment on the platform.
- Can report a comment.

### ***Account management***

A registered user:

- Can view his account settings.
- Can modify his account details.
- Can change his username for a limited number of times.

## **5.2.2 Application non-functional requirements**

Nonfunctional requirements encompass elements that are non-interactive or not explicitly defined as actionable operations. They can be considered as the aesthetic enhancements and refinements of the application.

### ***Security***

- The website must use JWT approach of authentication and authorization with the support of both access token and refresh token.
- The website routes must be protected from unauthorized users, by generating a JWT token for the registered user and only if they have valid JWT they can access the website protected routes.
- Logging out user automatically if the JWT is expired and ask him to login again in case user did not provide „remember me” feature.

### *Performance*

- The website should provide a pagination approach to retrieve the data from the blockchain, so that preventing huge load of data, such as loading of feed posts or loading of post's comments.
- The website should use client-side rendering to keep the website more user-friendly, dynamic, and fast.

### *Availability*

- The website is available only for users who have a website that's support digital wallets and they must have it as an installed extension.
- In the future the website should target specific blockchain networks since the contract will be deployed on a specific one.

### *Usability*

- The website is accessible through internet access by any different kind of users throughout the world.
- The website is almost responsive on all different kinds of screens.
- The website must provide simple UI friendly design, in terms of social media, it must be common, so user does not feel uncomfortable.

#### **5.2.3 Use cases and actors.**

For now, mainly the main actors of the application are the *unregistered user* and *registered user* as shown in *Figure 31* and in *Figure 32* means there is no admin for checking or manipulating the platform data or user's data. But an admin can be added in the future for website modifications such as layouts or blogs or different sections that can be used and integrated with a CMS<sup>13</sup> (content management system).

---

<sup>13</sup> Content management system: <https://kinsta.com/knowledgebase/content-management-system/>

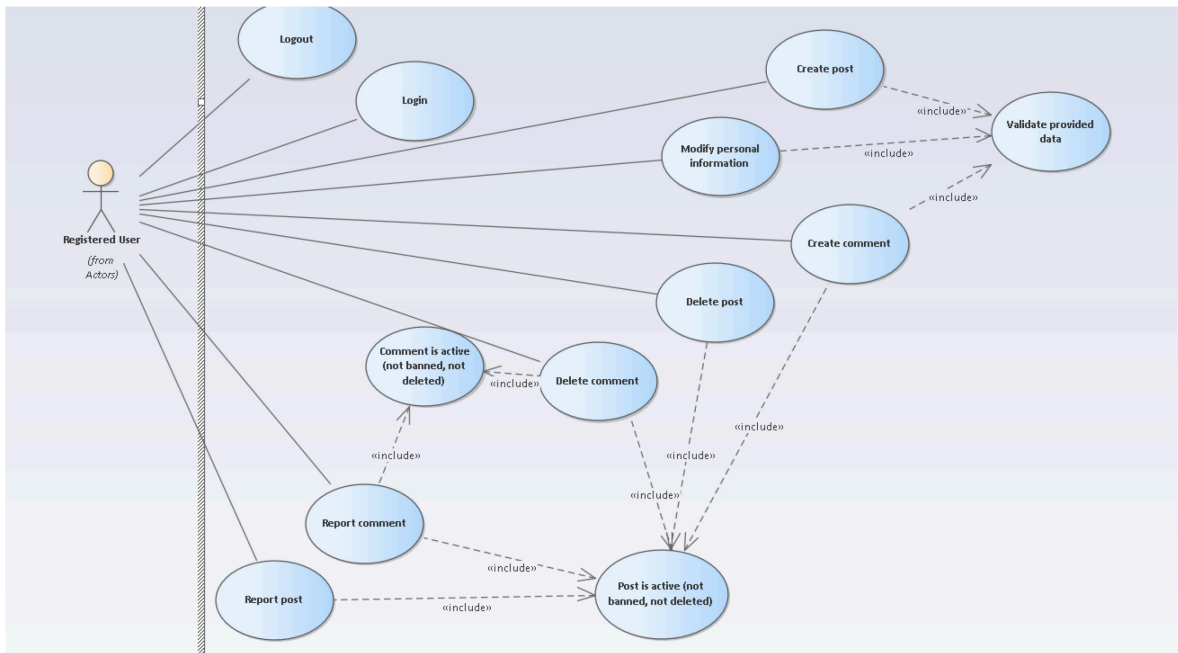


Figure 31. Registered user use cases

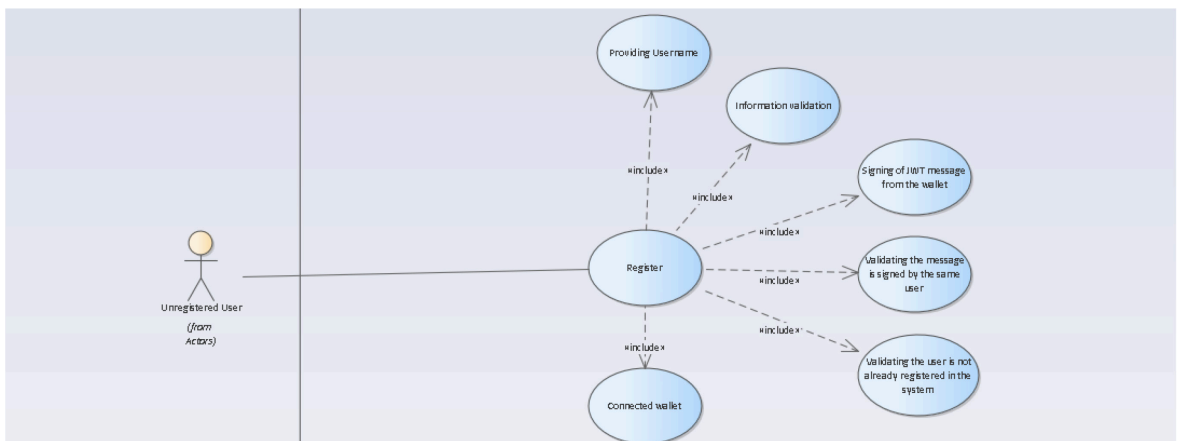


Figure 32. Unregistered user use cases

### 5.2.4 Class model

Since the application uses Solidity as the programming language for the back end of the blockchain but also C#, classes in Solidity are represented as structs and the app will have separated functions inside the contract that are outside the structs, that's why the classes do not have methods inside them in the diagram. In the back-end side of C# there is also some classes that does not have to be mentioned in the UML diagram because they do not represent the main core of the application, instead these are class used for example in handling

response and requests from to client-side, and also there are classes for handling errors they have properties such as (error message, error code). The aim is to focus on the classes that are most relevant and important to the application's domain and functionality as shown in the *Figure 33*.

So particularly in the application the class models will be divided into two sides:

1. **The C# side:** Mainly this side is used to generate JWT tokens and store user's address with the corresponding tokens, to authenticate the user to the front-end application and different protected routes and the C# side has mainly one important domain class.
  - a. User:
    - i. **AccountAddress:** It is a string and represents the Ethereum user's public key account address.
    - ii. **RefreshToken:** It is a string and represents the generated JWT refresh token that is used to request new access token.
    - iii. **TokenCreated:** It is a datetime data type that is used to know when the refresh token is generated for the user.
    - iv. **TokenExpires:** It is a datetime data type and used to know when the refresh token is expiring, so when it expires and user login with remember me feature, he has to login again to get new tokens.
2. **The Solidity side:** In this side all the data manipulation and user functionality are done here, this acts as the real back end of the application, classes in Solidity are represented as **structs** most of the time. And we will have the following structs and Enums:
  - a. **User (struct):**
    - i. **Id:** Id of the user represented as unsigned integer.
    - ii. **Name:** Name of the user represented as string.
    - iii. **userName:** Nick name of the user represented as string.
    - iv. **userBio:** Bio of the user small description about him represented as string.
    - v. **status:** Status is type of account status to represent if user is (banned, active, not present).
    - vi. **showUserName:** Used for each user to check if he wants to show his username publicly.
    - vii. **birthDate:** Is an unsigned integer, the reason it is an integer because Solidity does not provide built-in date time data type instead, we store date as Unix timestamp.
    - viii. **ethAddress:** The Ethereum public account address of the user, it is the type of address that is provided by Solidity.

- ix. **profileImageHash:** String that represents the hash of the image that is uploaded to IPFS platform.
  - x. **profileCoverHash:** String that represents the hash of the image that is uploaded to IPFS platform.
- b. **Post (struct):**
- i. **postId:** Id of the post represented as unsigned integer.
  - ii. **postDescription:** Content of the post represented as string.
  - iii. **status:** Status is type of post status to represent if post is (banned, active, not present).
  - iv. **author:** The Ethereum public account address of the user that posted the post, it is of type address that is provided by Solidity.
  - v. **imgHash:** String that represents the hash of the image that is uploaded to IPFS platform.
  - vi. **likeCount:** Number of likes on the post, represented as unsigned integer.
  - vii. **reportCount:** Number of reports on the post, this can be used in the future to hide and mark posts as deleted if specific number of reports is exceeded.
  - viii. **timeStamp:** It is an unsigned integer to remember when the post is created, Unix timestamp is used here since solidity does not provide built-in datetime data type.
- c. **Comment (struct):**
- i. **commentId:** Id of the comment represented as unsigned integer.
  - ii. **author:** The Ethereum public account address of the user that posted the post, it is of type address that is provided by Solidity.
  - iii. **postId:** Id of the post to attach comment to it. Represented as unsigned integer.
  - iv. **content:** Content of the comment represented as string.
  - v. **likeCount:** Number of likes on the post, represented as unsigned integer.
  - vi. **reportCount:** Number of reports on the post, this can be used in the future to hide and mark posts as deleted if specific number of reports is exceeded.
  - vii. **timeStamp:** It is an unsigned integer to remember when the comment is created, Unix timestamp is used here since solidity does not provide built-in data time data type.
  - viii. **status:** Status is type of commentStatus to represent if comment is (banned, active, not present).



**d. postStatus, commentStatus, userStatus (enums):**

The reason for now all Enums have same properties but still we have separated Enums is that, in the future we might add different properties for each Enum, such as privacy of a post, or hiding of account and so on.

- i. NP:** Not present.
- ii. Active:** Active, refer that the object (post, comment, user) is a live.
- iii. Banned:** Due to number of reports.
- iv. Deactivated:** When deleting this status is used.

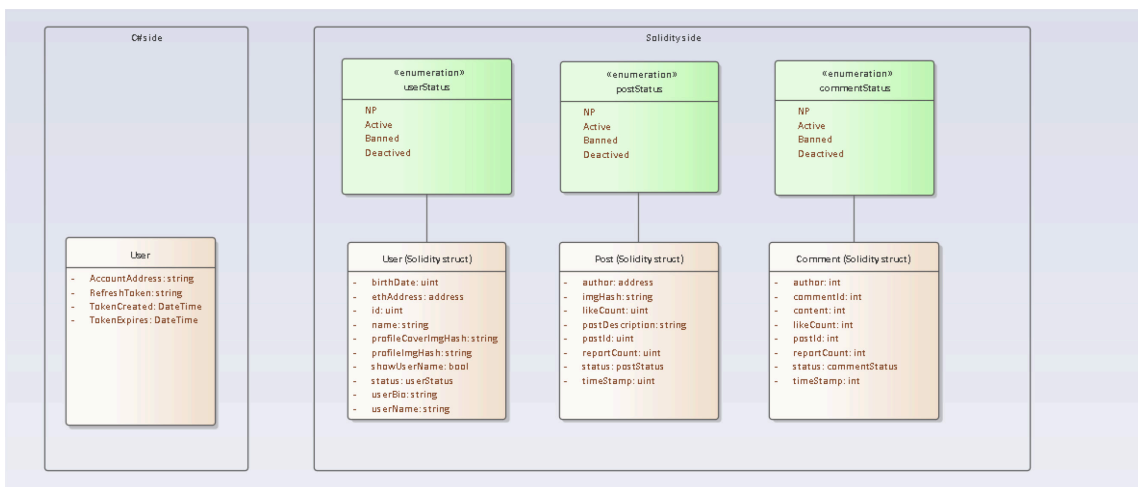


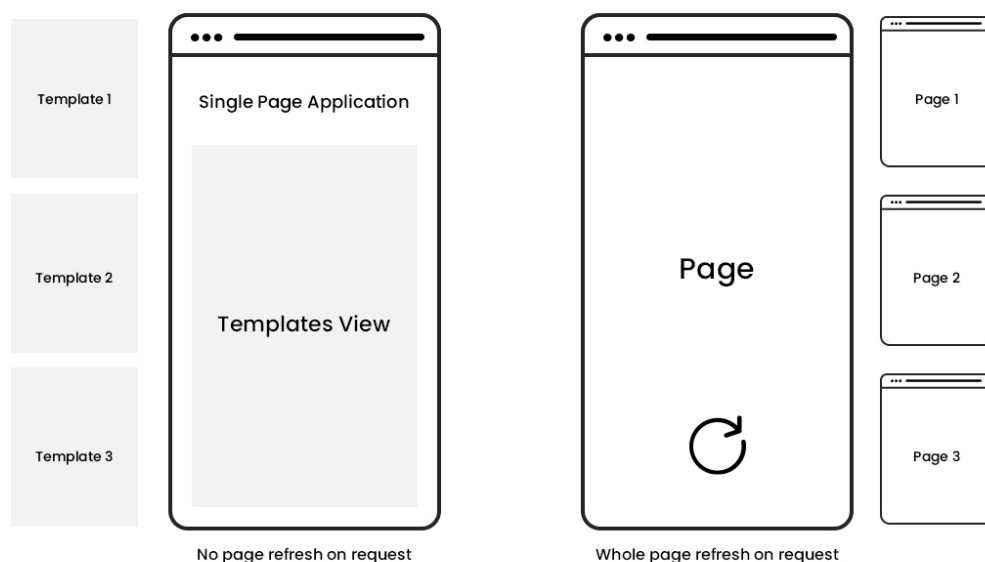
Figure 33. UML Class model

### 5.3 Front-end Technologies and UI inspiration

When building an app, it is necessary to choose appropriate front-end technologies. It is also important to consider the latest technologies used in the fields of blockchain and web development in general. When selecting technologies, a consideration should be taken to ensure that documentation, tutorials, and support are readily available, as well as a community to support the technology when encountering errors or bugs. It is necessary to choose the correct front-end technologies that support the interaction with smart contracts or with the blockchain in general. JavaScript is the main player from this side. Frameworks such as ReactJS, NextJS, VueJS that are built with JavaScript support the interaction with the blockchain through different type of wallets. The application that will be developed the NextJS framework will be used, so for the purpose of showing the data and the interaction with the blockchain NextJS will be used.

### 5.3.1 NextJS

NextJS as described by their official documentation<sup>14</sup> is “**The React Framework for production**” by that they mean developers will be still writing ReactJS code and still build ReactJS components and use ReactJS features. NextJS is a JavaScript framework, that is built on top of the framework ReactJS, the main purpose of this framework is to address the limitations of ReactJS and add a lot of improvements of how web applications can be created, and it make it so much easier. NextJS and ReactJS are frameworks to create SPA (single page application), meaning there is one HTML page and inside it there is the data, and the data is represented as components as shown in the *Figure 34*. The data is automatically changed on the client-side. SPA makes the web app more user friendly and does not refresh the website on client side on every request of new view or new section or new data.



*Figure 34. SPA vs multi page app [54]*

NextJS has one of the best features for developers which make it so much easier to build websites, it is the navigation between the different page routes and routes. NextJS pages are associated with a route based on their file name and this feature is named “**File-based routing**”. For example, in development:

- `pages/index.js` is associated with the `/` route.
- `pages/posts/first-post.js` is associated with the `/posts/first-post` route.

<sup>14</sup> NextJS official website: <https://nextjs.org>

Means the name of the folder inside the pages folder in a NextJS defined the parent route for child's routes inside it. One of the amazing features that NextJS provides is the different ways of rendering the data, with just simple lines of codes, NextJS provide different way of rendering such as:

- 1. Server-side rendering (SSR):** NextJS provides server-side rendering as a built-in feature. Server-side rendering is the ability to convert HTML files on the server into a fully rendered HTML page for the client. This eliminates the idea of single page application because server-side rendering requires the client-side to be refreshed to be able to see the new data. Still NextJS provides this feature for developers who want to load the data fully on the client-side, server-side rendering is useful in cases where we want to display a static data that does not require updates frequently, such as blogs websites, wallpapers websites. Server-side rendering has huge positive affect on the SEO (Search Engine Optimization) of a website because the content can be rendered before the page is loaded, which is ideal for SEO, while client side rendering the HTML code is empty which in result does not help with SEO at all. Rendering server-side may be ideal for static site generation, but frequent server requests and full page reloads can result in overall slower page rendering in more complex applications. To implement this feature in NextJS it really requires a little bit of code. In the *Figure 35*. The component is loaded only After the data is fully populated into the html code then the full code is sent to the client side.

A screenshot of a code editor showing NextJS server-side rendering code. The code is in a dark theme. It includes a function to fetch data from an external API and a component that renders the data. Two yellow boxes highlight the data fetching and the rendering logic. A yellow arrow points from the first box to the second.

```
1 //Server-side data fetching
2 export async function getserverSideProps() {
3   const resp = await fetch(
4     "https://jherr-pokemon.s3.us-west-1.amazonaws.com/index.json"
5   );
6   return {
7     props: {
8       pokemon: await resp.json(),
9     },
10  };
11 }
12 export default function Home({ pokemon }) {
13   return (
14     <
15       <Head>
16         <title>Pokemon list</title>
17       </Head>
18       <h1>Pokemon List</h1>
19       <div className={styles.grid}>
20         {pokemon.map((pokemon) => (
21           <div key={pokemon.id} className={styles.card}>
22             <Link href={` /pokemon/${pokemon.id}`}>
```

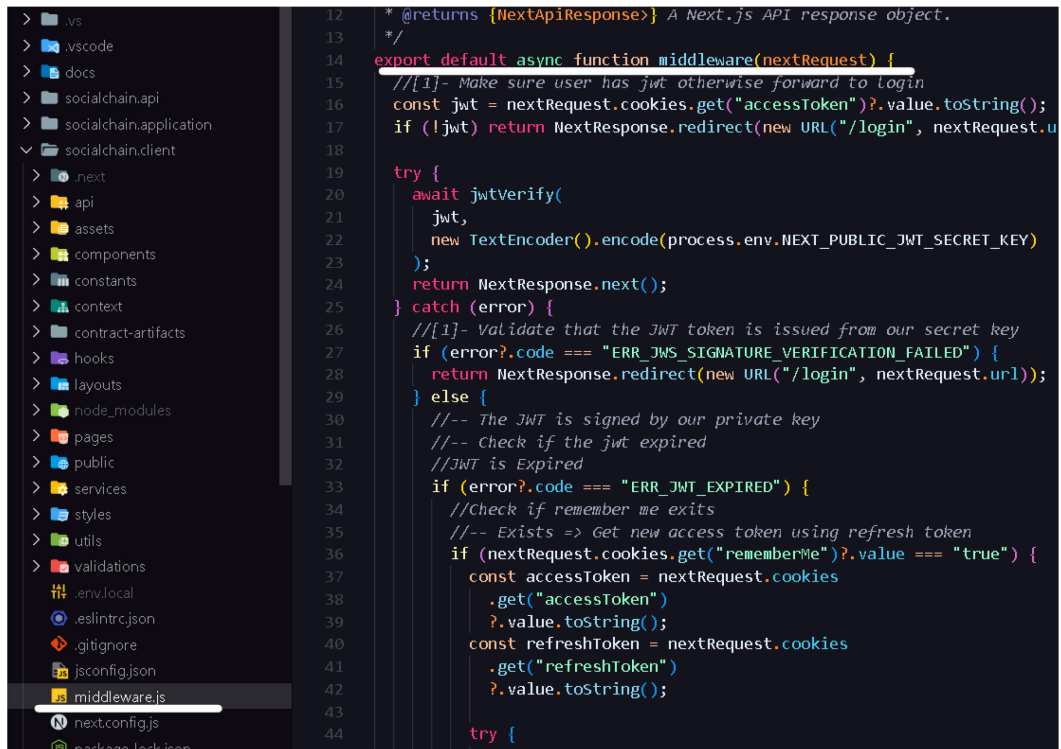
Figure 35. NextJS server-side rendering example.

2. **Client-side rendering (CSR):** By default, without specifying the way of rendering NextJS is a client-side rendering framework, which means the fetching of the data is done on the client side only. This way of rendering is useful when updates of the data frequently are required or when there are changes by different users on the data and other users want to see the changes frequently. Client-side rendering will be used in our application to render data, since social media requires frequent updates and dynamic refresh of data.

NextJS has also a really nice built-in feature, it is the “**middleware**”, middleware in general is something running between two sides, Taking an example where two people communicating between each other, and there is a man in the middle that is both sides knows about, this man is responsible to enhance the communication between the two sides he do some modifications before the message is transmitted between the two side. Middleware, as NextJS describes in their official documentation, is that before a request is processed, middleware enables you to run code. Then, depending on the incoming request, a response can be changed by rewriting, redirecting, changing the request or response headers, or by directly responding. Middleware can be useful in the following cases:

- Handling authentication and authorization, if user is not authenticated and he tries to access a protected route we can response or redirect the user to page where to show him he is not authorized.
- Handling errors and bad responses.

To run a middleware in a NextJS application, it is necessary to create a **middleware.js** file at the root of the NextJS application and write the necessary code in it as shown in the *Figure 36* below.



```

12  * @returns {NextApiResponse} A Next.js API response object.
13  */
14  export default async function middleware(nextRequest) {
15    //[[1]- Make sure user has jwt otherwise forward to login
16    const jwt = nextRequest.cookies.get("accessToken")?.value.toString();
17    if (!jwt) return NextResponse.redirect(new URL("/login", nextRequest.u
18
19    try {
20      await jwtverify(
21        jwt,
22        new TextEncoder().encode(process.env.NEXT_PUBLIC_JWT_SECRET_KEY)
23      );
24      return NextResponse.next();
25    } catch (error) {
26      //[[1]- Validate that the JWT token is issued from our secret key
27      if (error?.code === "ERR_JWS_SIGNATURE_VERIFICATION_FAILED") {
28        return NextResponse.redirect(new URL("/login", nextRequest.url));
29      } else {
30        //-- The JWT is signed by our private key
31        //-- Check if the jwt expired
32        //JWT is Expired
33        if (error?.code === "ERR_JWT_EXPIRED") {
34          //Check if remember me exists
35          //-- Exists => Get new access token using refresh token
36          if (nextRequest.cookies.get("rememberMe")?.value === "true") {
37            const accessToken = nextRequest.cookies
38              .get("accessToken")
39              ?.value.toString();
40            const refreshToken = nextRequest.cookies
41              .get("refreshToken")
42              ?.value.toString();
43
44            try {

```

Figure 36. NextJS middleware

### 5.3.2 Tailwind CSS

Tailwind CSS is a CSS framework, as the official website<sup>15</sup> describes it, it is a “utility-first CSS framework”. In other words, styling an HTML element or tag using classes names provided by the framework itself. Some of the most used styling and usually used in almost any websites the “**flex box**” CSS property to align items in the web application. To implement that in normal CSS the following can be done as shown in the *Figure 37* and *Figure 38*.



```

1 <div class="flex-container">
2   <div>1</div>
3   <div>2</div>
4   <div>3</div>
5 </div>

```

Figure 37. HTML flex box

<sup>15</sup> Tailwind CSS official website: <https://tailwindcss.com>

```

1 .flex-container {
2   display:flex;
3   background-color:gray;
4 }
5
6 .flex-container > div {
7   color:white;
8   background-color:black;
9   margin:10px;
10  padding:20px;
11  font-size:30px;
12 }

```

Figure 38. CSS flex box

While in tailwind all of this can be done without the need for a CSS file, as shown in the Figure 39.

```

1 <div class="flex bg-[#808080] text-white">
2   <div class="bg-black m-[10px] p-[20px] font-[30px]"></div>
3   <div class="bg-black m-[10px] p-[20px] font-[30px]"></div>
4   <div class="bg-black m-[10px] p-[20px] font-[30px]"></div>
5 </div>

```

Figure 39. Tailwind CSS example

Both will result will be as shown in the Figure 40.



Figure 40. Tailwind CSS and normal styling result

### 5.3.3 Other frameworks to improve user experience.

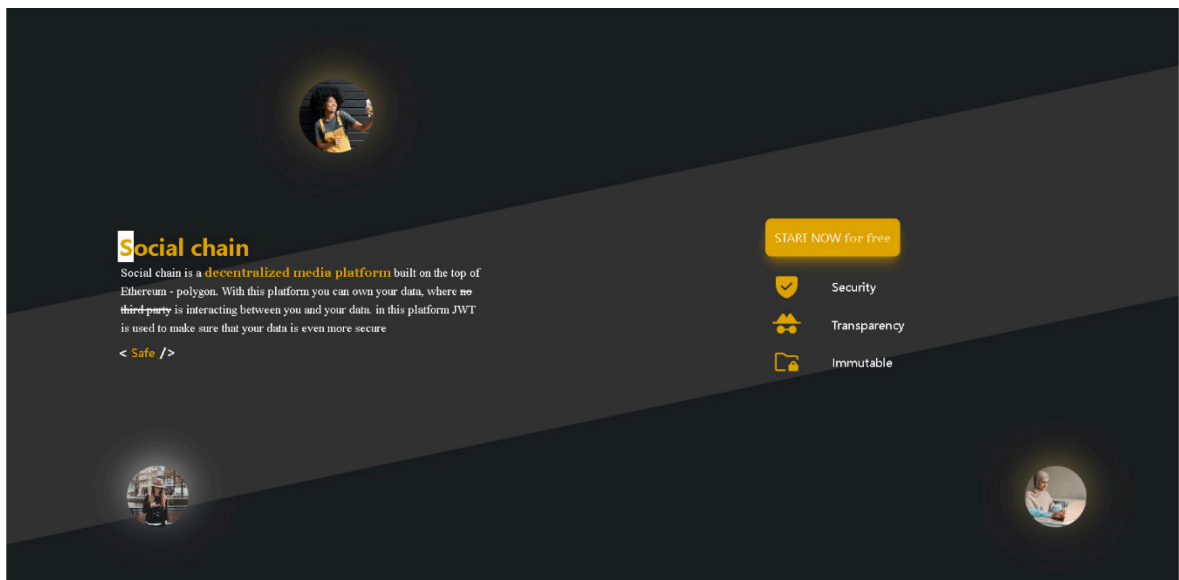
In the application some front-end frameworks will be installed and used that will improve user experience and make the website more attractive and user friendly, such as:

- **Framer motion:** Framer motion is a simple and powerful motion library for ReactJS.
- **phosphor-icons:** Light weight library for a useful collection of icons.

- **lottie-web**: Light weight library for running JSON file animations.
- **antd**: As described by their website<sup>16</sup> it is an enterprise-class UI design language and React UI library with a set of high-quality React components, one of best React UI library for enterprises.

#### 5.3.4 User Interface Design and taken approach.

To design the UI of the application, a distinct approach was adopted. Instead of creating the UI from scratch using specialized software or platforms like Adobe Illustrator or Figma, the process involved seeking inspiration from UI images on Google and blending various ideas derived from search results. Subsequently, the following results were achieved for the index page, or landing page with the route "`{websiteUrl}/`" (index root), as illustrated in *Figure 41*.



*Figure 41. Social chain index page*

For the login/register page that has the route "`{websiteUrl}/login`" has the UI as shown in the *Figure 42*.

---

<sup>16</sup> Ant design official website: <https://ant.designs>

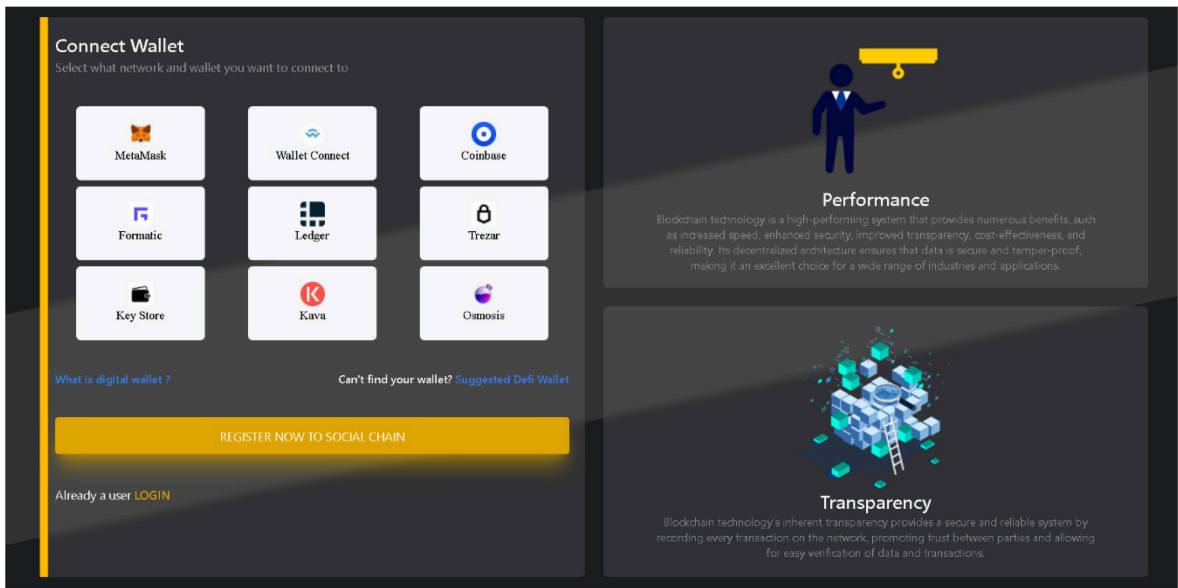


Figure 42. Social chain login page

For the register modal that when user want to register this has no special route it is just a pop-up modal that has an HTML form as shown in the Figure 43.

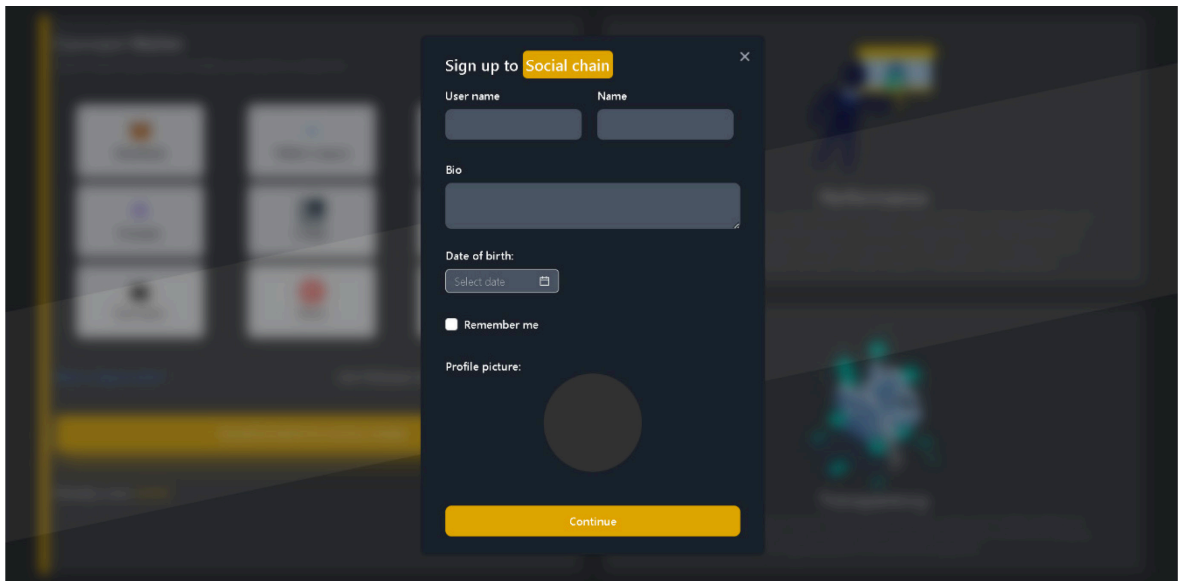


Figure 43. Social chain register modal

For the home page, or the feed page where users’ posts will be shown the route “{websiteUrl}/home” is set and has the following UI as shown in the Figure 44.



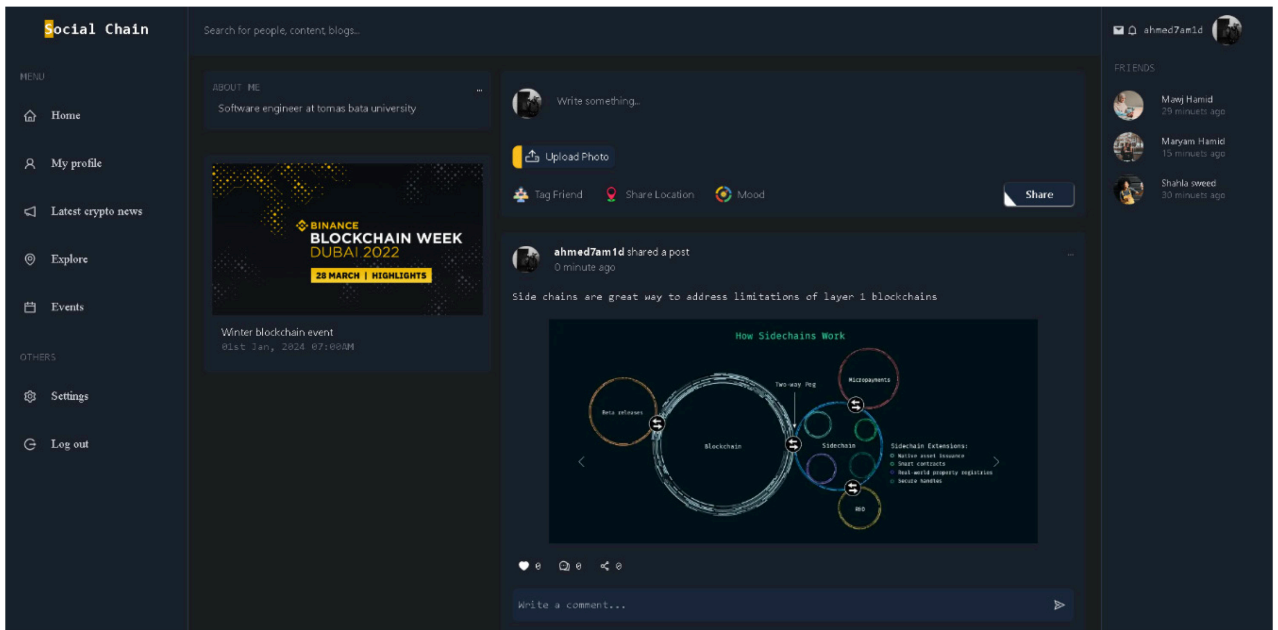


Figure 44. Social chain feed/home page

User’s profile page, it has the route “{websiteUrl}/home/profile” and has the UI as shown in the Figure 45.

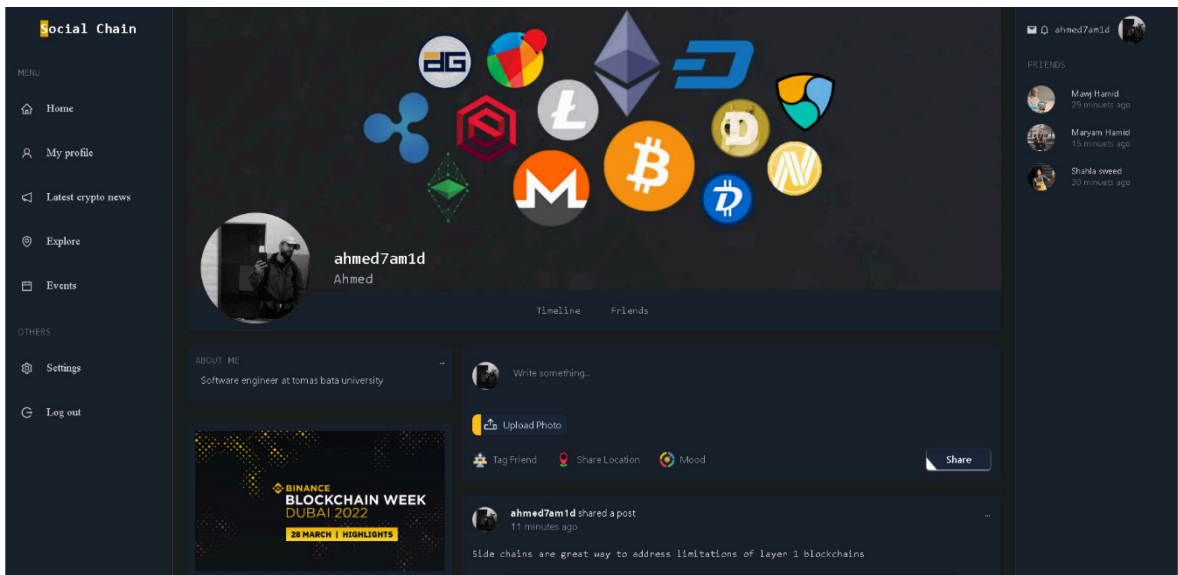
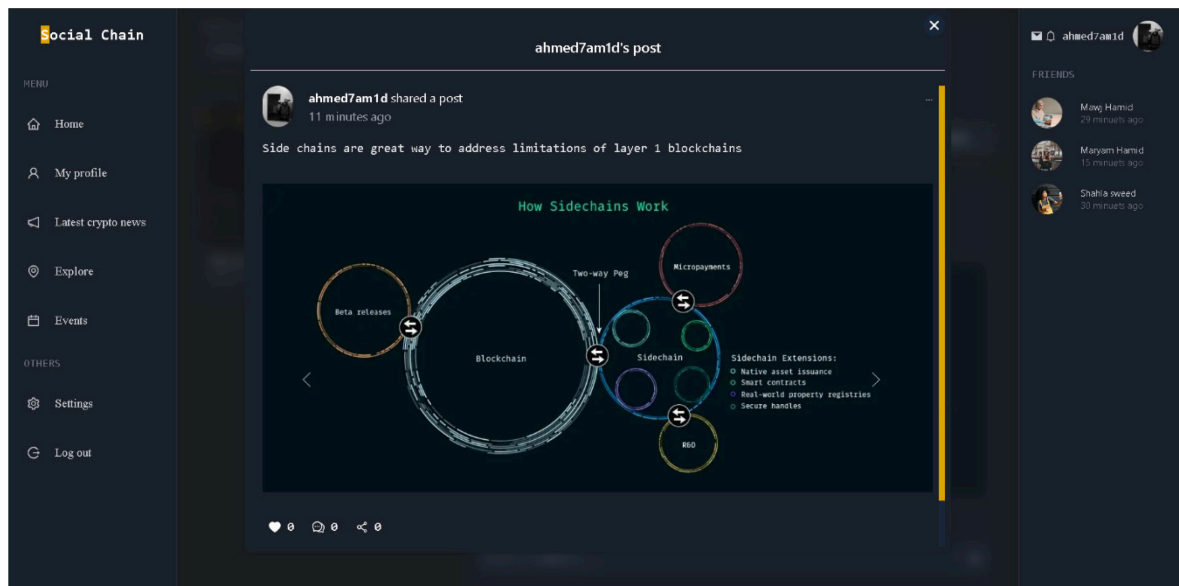


Figure 45. Social chain user profile page

Regarding the post modal, it is just pop-up modal that has no route it pops when a user click on “show more comment” or by clicking on the post’s image, the purpose of this modal is

to load more comments and to implement the idea of pagination and to have a better overview of the post, as shown in the *Figure 46*.



*Figure 46. Social chain post modal*

## 5.4 Back-end technologies

For the back-end technologies C# will be used as the main programming language, and .NET Core API will be used to build the website API, the database will be MS SQL Server database. The main purpose of the back end is to make the website more secure and protect the different type of routes. Using JWT tokens.

### 5.4.1 Server-Side frameworks and libraries

Some frameworks and libraries will be used in the application to make the work easier and cleaner. The most important library that is worth mentioning is the “**Nethereum.Web3**” this library is used for smart contract management and interaction with the Ethereum nodes using C#, whether they are public nodes or private nodes or even local nodes. And for handling the errors and exceptions that are returned to client-side “**ErrorOr**” library will be used and a smart way of handling errors using an endpoint that will handle all the throwed exceptions and errors throughout the application, the idea is that formatting exceptions thrown in the back-end application and return them to the client-side (NextJS) .

### 5.4.2 APIs and databases

The API will be created using ASP.NET core, Using .NET Core API framework is a great way to create APIs in a simple and fast way, Microsoft already has template for such a project. For the database MS SQL Server (Microsoft Structure Query Language Server) will be used, it is a relational database means data represented in tables of rows and columns.

## 5.5 Web3 Technologies

Bringing the decentralization feature of the blockchain to the world of web, requires special technologies used that helps us interact with smart contracts deployed on the blockchain, or with the blockchain itself.

### 5.5.1 Development environment

There are many development environments for setting up a project on the blockchain and start to develop smart contract on the blockchain whether it is local blockchain or public blockchain. “**Truffle**”<sup>17</sup> and “**hardhat**”<sup>18</sup>, these are development environments that provide a lot of tools and scripts to interact with live blockchain and build and compile and deploy smart contracts locally or live. Choosing which environment to use was a little bit challenging but looking at the advantages of each environment helped to choose the perfect framework. The main reason to choose hardhat over Truffle was that hardhat provides a built in local blockchain simulation, without the need for external software such as Ganache that is being used by Truffle most of the time. Talking about hardhat, Hardhat is not only about deploying and running smart contracts, but also provides a lot of features that make it more powerful. Such features are:

1. **Hardhat built-in network:** As mentioned before hardhat provides a local Ethereum network node that is designed for development purposes, making it easier and faster to make transactions on the network.
2. **Extensions:** Hardhat provides many extensions and plugins that improve its functionality. Such extension is „**nomiclabs/hardhat-ethers**” that allows developers to run the library ethers.js into the runtime of the environment. Or „**hardhat-gas-reporter**” this extension

---

<sup>17</sup> Truffle official website: <https://trufflesuite.com/>

<sup>18</sup> Hardhat official website: <https://hardhat.org/>

can be helpful because it can estimate how gas will be used for specific method in the contract.

- 3. Debugging inside the smart contracts:** Solidity as a smart contract programming language is hard to debug, to discover the errors it is necessary to execute the smart contract or deploy it and throw an exception. But hardhat provides one of its extensions that make it possible to run the (*console.log()*) function inside the contracts to investigate the changes and errors. [55]

### 5.5.2 Decentralized data storage - IPFS

There is no social network platform nowadays that has no photos or videos, every social media platform includes the usage of photos, videos, documents and sometimes even streaming. And since the plan it to make the application as decentralized as possible, all users' files including (videos, photos, documents) should be stored in a decentralized way. At the same time, a user should not be forced to pay fees to upload visual content, a user should only pay for the gas fee on that is required by the contract. IPFS (InterPlanetary File System), IPFS is a set of protocols for transferring data, it can be described as a decentralized network composed of peer-to-peer nodes that is open and participatory as shown in the *Figure 47*. It can also be described as a distributed file system, IPFS and blockchain share some similarities, such as both are decentralized, distributed and the use the power of cryptography for hashing the content and secure it. IPFS tries to solve the problem of accessing something by IP address, for example requesting a website using the location it is hosted on, IPFS instead tries to deliver for us the website by refereeing to it is content instead of the location and the content is collected from the nodes in the distributed system. Content in IPFS is immutable any updates to the content create new hash of the content, means it is not possible to update a file that is uploaded to the IPFS network. It is necessary to talk about life cycle of files in the IPFS and how they are stored and retrieved from such a system:

- 1. Uploading the file:** When first the file is uploaded to IPFS it is divided into blocks or chunks. Each of these blocks is hashed using a cryptography hashing function such SHA-256, for the purpose of creating a unique identifier called a CID (Content identifier) this step is called „**Content-addressable representation**” as IPFS describes.
- 2. Remembering where the hashes of chunks are stored:** The IPFS system has a collection of tables called Distributed Hash Tables (DHT) where it maps each CID to the per addresses that has the content, each peer or node in the network, has the same copy of DHT in case retrieving the CID from a node failed other nodes can provide it. But some nodes might be offline or are not updated, in that case in IPFS there is a step called Pinning.

3. **Pinning:** Saving the CID on the node does not mean the CID is retrievable, pinning allows the node/peer to advertise that it has the CID and provide it to the network.
4. **Retrieving the content:** when a node/peer or someone wants to access the file, the chunks of the requested file are retrieved to make up the requested file.

For the application that will be created to demonstrate the practical part. We will not be running our node in the IPFS network, instead the app will use web 3 services that runs node and provide limited data storage to upload files, such services provide an API to upload files to the IPFS network. In the application “**Infura**” will be used which is a development suite that provides instant, scalable API access to the Ethereum and IPFS networks.[56] [57]

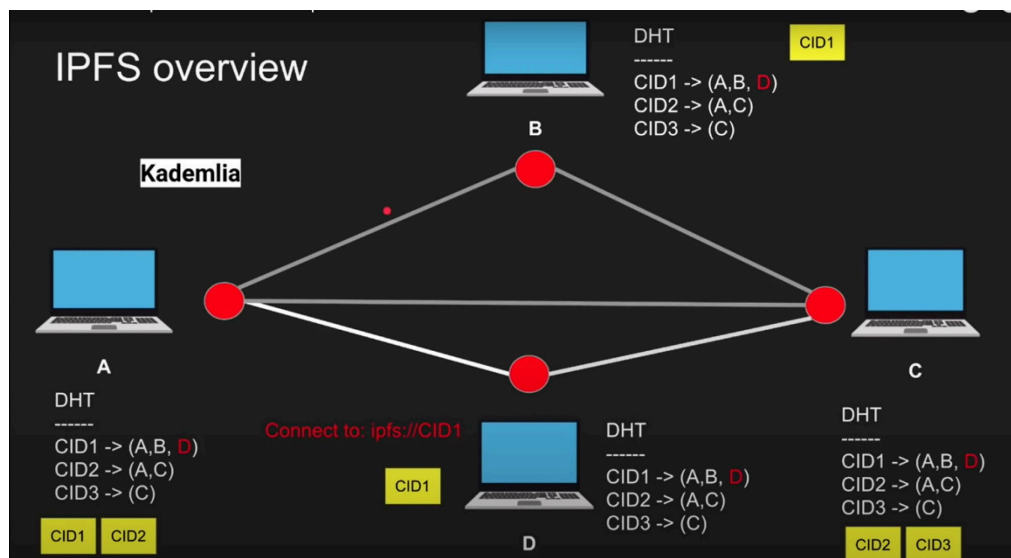


Figure 47. IPFS overview[57]

### 5.5.3 Developing smart contracts

When talking about the development of smart contracts, there are several programming languages that support writing smart contracts on the Ethereum blockchain. Such languages are **Solidity**, **Vyper**, **Rust**, these are the most famous and used programming languages but not the only one. In the application Solidity will be used as the language to write smart contracts. Solidity is a high-level typed object-oriented programming language; it has similar syntax as TypeScript or C++. Solidity is designed specifically for writing smart contracts that runs on the Ethereum blockchain. Solidity came alive in 2014. A smart contract file has the extension “**filename.sol**” and is wrapped always with a contract keyword, inside the

curly braces is the code of the contract, and inside the contract there can be different syntaxes and data representation such as:

- **Functions:** These are a collection of executable code that can be reused and re-called throughout the contract, functions in solidity can have the **view** keyword means they don't make any changes in the contract storage or code, instead they just return value (query function) otherwise a function that is considered as a command function that make changes is not marked with view keyword and such function require a gas fee and a transaction to be executed from outside.
  - **Variables and data types:** Variables in Solidity are typed means the data stored in memory, should be specified, Solidity provides the following data types:
    - **Boolean.**
    - **Integer.**
    - **Address:** Used to represent an Ethereum address. It is a 20-byte value that is unique to each account or contract.
    - **String.**
    - **Bytes.**
    - **Array.**
    - **Mapping:** Used to represent a collection of key-value pairs, where the keys are of one data type and the values are of another data type.
    - **Enum.**
    - **Struct.**
  - **Modifiers:** Modifiers in solidity are like any modifiers in other programming languages, they decide whether a variable or function or struct can be accessible from outside or inside, however solidity provide more built-in modifiers than others programming languages, and these are:
    - **Public, private, internal**
    - **View and pure:** Are functions modifiers.
      - **View:** Indicates that the function does not change the state of the contract it is used only for reading.
      - **Pure:** Indicates that the function does not read/write anything from/to smart contract states.
    - **Payable:** Indicates that the function or the specified state, can receive specific amount of Ether as a transaction value.
- Custom modifier:** These are modifiers that can have a custom name, they acts like a conditional function they have logic inside them, and the logic is usually conditional, for example we can have a modifier that checks if user is exists in a

specific mapping as shown in the *Figure 48*, and if so the function should continue the execution otherwise the modifier should throw an error message and stops the execution of the function a custom modifier can be written next to the function name, same as (public, internal, private). The sign „\_“ in the modifier scope at the end indicates that if the required condition passed successfully, continue the execution of the „*createPost*“ function otherwise throw the error message with string „*Not a Registered user*“.

```
1  modifier onlyAllowedUser(address userAddress) {
2      require(
3          users[userAddress].status == accountStatus.Active,
4          "Not a Registered User!"
5      );
6      _;
7  }
8
9  function createPost(string memory _postdescription, string memory _imghash)
10     public
11     onlyAllowedUser(msg.sender)
12     {
13         totalPosts = totalPosts + 1;
14         uint256 postId = totalPosts;
15         posts[postId] = Post(
16             postId,
17             msg.sender,
18             _postdescription,
19             _imghash,
20             block.timestamp,
21             0,
22             0,
23             postStatus.Active
24         );
25         //each user will have an array of postId that he posted
26         userPosts[msg.sender].push(postId);
27         postIds.push(postId);
28         emit logPostCreated(msg.sender, users[msg.sender].id, postId);
29     }
```

*Figure 48. Solidity modifier and function*

- **Events:** Events in Solidity are built in feature, mainly exists for the purpose of logging of an action occurred within the smart contract. Events in solidity have the „*event*“ keyword as shown in the *Figure 49*. And to be able to call or emit an event the keyword „*emit*“ is used before the name of the event, providing the required parameters that the event is looking for. The data that is emitted from an event is logged to the transaction log, which is then stored in the blockchain. All transaction logs in the blockchain are immutable even the ones that are emitted from smart contracts, means they will not be deleted, and they can't be altered. [58][59]

```
1 event logRegisterUser(address userAddress, uint256 userId);\n2\n3 function createPost(string memory _postdescription, string memory _imghash)\n4     public\n5     onlyAllowedUser(msg.sender)\n6     {\n7         totalPosts = totalPosts + 1;\n8         uint256 postId = totalPosts;\n9         posts[postId] = Post(\n10            postId,\n11            msg.sender,\n12            _postdescription,\n13            _imghash,\n14            block.timestamp,\n15            0,\n16            0,\n17            postStatus.Active\n18        );\n19        //each user will have an array of postId that he posted\n20        userPosts[msg.sender].push(postId);\n21        postIds.push(postId);\n22        emit logPostCreated(msg.sender, users[msg.sender].id, postId);\n23    }
```

Figure 49. Solidity event and emit.

#### 5.5.4 Interaction with the smart contract and blockchain

For the interaction with the blockchain there are few web3 libraries that making the process of the interaction with the blockchain much easier and more user friendly for developers than direct communication, these libraries provide functions such as:

- Reading from the blockchain network.
- Sending/receiving transactions to/from the blockchain.
- Interaction with the network's smart contracts.

There are few common and most used libraries for that purpose:

- **Etheres.js:** Is a JavaScript library, that aims to be a good solution for Ethereum development, this library was released or published a year after Web3.js and still has managed to become the most popular library for interaction with the Ethereum blockchain. Even though it is a JavaScript library, Etheres.js fully supports TypeScript. Etheres.js makes it possible to create A JavaScript object from any contract ABI, means a contract can be presented as JavaScript object and access the contract function easily. The library keeps the private keys in the client-side, so it is more secure and safer. The statement "Keep the private keys in the client-side " is to store the private keys on the users' client-



side devices, such as their personal computers or hardware wallets, rather than on a third-party server or service. This is because storing the private keys on a third-party service or server increases the risk of the private key being stolen or accessed by unauthorized parties, potentially resulting in the loss of the assets. By keeping the private keys on the client's side, we have greater control over the security of the assets.

- **Web3.js:** The Ethereum Foundation developed the open-source JavaScript (JS) library known as Web3.js. Web3.js' primary goal is to make it easier to connect with the Ethereum chain in a seamless manner. This interaction is made feasible by using the JSON-RPC<sup>19</sup> protocol to communicate with Ethereum nodes.
- **Web3.py:** If someone knows the programming language Python and likes to code in Python, this library can be a good option for them. This Python package web3.py allows users to communicate with Ethereum. The original API was based on the Web3.js JavaScript API, but it has since changed to better suit the requirements and conveniences of Python developers.

For the developed application the “**ethers.js**” library will be used, since it has all the functionality that is needed for the application and it uses JavaScript mainly, but also because the library has huge base of documentation and huge community support. [60]

---

<sup>19</sup> JSON-RPC protocol: <https://nonamesecurity.com/learn-what-is-json-rpc>

## 6 APPLICATION'S SECURITY ARCHITECTURE

It is important to make the application as secure as possible, since we are dealing with sensitive information or in other words we are dealing with real money, and since every request in the application or change in the contract storage require a transaction or real money, it is important to make the website as secure as possible. And prevent any unauthorized person from accessing it. Due to that I was thinking of a way to combine JWT authorization with the web3 without giving up the feature of decentralization. Using the strength of digital wallet authorization combined with the strength of .NET Core API. The only thing that will be stored in the database to prevent losing the feature of decentralization is the user's public account address and his (refresh token) also the expiration and creation of the token dates.

### 6.1 Security architecture

JWT authorization scheme and the use of the digital wallet will be the protector of every user and will be the protector of the website routes (website pages and users' content).

#### 6.1.1 JSON web token

JWT stands for JSON web token, it is an open standard used to share security information between two sides, mostly it is a client side and a server side. A JWT is no more than a three parts long string separated by dots [.] and each part encoded using base64. A JWT string contains important data about the user, this data called *claims*, these claims can be (name of user, Ethereum account address, expiration time of the token, when the token is issued to the user) and even custom claims can be added, such as phone number, address and more. But mostly a standard default claim contains:

- **„iss“ (Issure):** This identify who issued the token, in most of the cases it will be the company name, or the owner of the software, in our case it is „social chain“
- **„sub“(Subject):** This indicates unique data about the user who owns this token such as user identifier.
- **„iat“: (issued at):** This claim indicates when the token was created or issued it is mostly UNIX<sup>20</sup> time stamp.

---

<sup>20</sup> UNIX time: <https://kb.narrative.io/what-is-unix-time>

- „exp“:(expiration): This claim indicates when the token will be expired and it is not authorized any more, this must be set by the issuer, for example in the back end side.

An example of JWT can be:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMjQsInR5cCI6IkpXVCJ9.eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMjQsInR5cCI6IkpXVCJ9
```

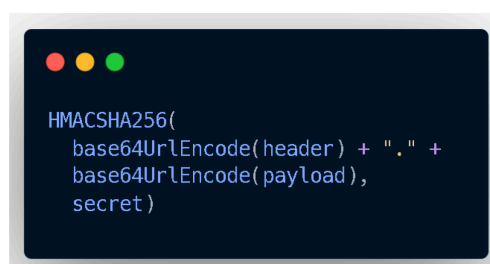
Here the first part called “**HEADER**” this part contains information about the algorithm used to encode the whole token and type of token before encoding this part using base64 this will result in:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

The second part is called “**PAYLOAD**” this part contains the claims and the necessary data. And the result before the encoding process is:

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

The third part is mostly called “**THE SIGNATURE**”. This part is formed by hashing the header and the payload and the secret all together using the hashing algorithm provided in the header, they are all putted together in a function such as the one shown in the *Figure 50*.



```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```

Figure 50. JWT hashing



So, this is how JWT is powerful and used for the authorization process, that's why it was decided to use it with the combination of the digital wallet in the application. The JWT token when storing it on the client side, it is important that it should be stored in a place where only server side can access it, rather than store it on local storage of the browser or session cookies, it is better to store it in an **HTTP Only cookie**<sup>21</sup>, to prevent any attacks that happens on the client side and the attacker will be able to take the JWT and act as the original user. [61][62]

### 6.1.2 Securing the app

There are three elements that controls the security of the application:

- From the client side.
- From the back-end side.
- From the blockchain side.

**In the client-side** there is a middleware that sit between every request and response made in the client application, the middleware is implemented using the middleware feature provided by NextJS. **In the back-end** side validating and registering the user to obtain the JWT token and refresh token that are generated using our own secret key, and in the client-side using our own secret key to validate that the JWT is not expired, and it is contains the right claims, if all of this are valid, user will be authenticated to access the website and website's routes. Talking about **the blockchain side**, specifically the smart contract of the application. The smart contract will be protected using conditions and modifiers, so every step must be checked to restrict unauthorized users from making any changes to the contract.

---

<sup>21</sup> HTTP only cookie: <https://www.cookiepro.com/knowledge/httponly-cookie/>

## 7 APPLICATION'S ARCHITECTURE AND SETTING UP DEVELOPMENT ENVIRONMENT

All software projects, whether personal or public, require a well-structured architecture to facilitate development and construction. Application architecture is the backbone of the software, especially when working on a software that more than one developer contributes to it. When choosing a specific software architecture pattern, it is necessary to keep in mind to choose the one that will result in a scalable, maintainable, testable, fault tolerant software, that can adapt to changes in the future. There are many software architectures that has different purposes, an example of them can be:

- **Layered pattern:** As the name indicates means the software will be divided into layers that are on top of each other, and each layer is responsible to handle specific operations and data, this type of architecture also has the name „*N-tier architecture*” an example of layered pattern can be **clean architecture** that will be used to create the demonstrated application. Such a design has typically four layers:
  1. **Layer of presentation:** Here is where the UI of the application sits.
  2. **Commercial or business logic layer:** This layer is responsible for the logic of the application, such as implementing and processing the request from the client-side.
  3. **The application layer:** This layer is responsible for exchanging the data between the data layer and the presentation layer.
  4. **Domain or data layer:** Layer where the data models are located.

Such a layer can be useful for E-shop web applications development.

- **Client-server pattern:** The client-server architecture has two primary parts. Multiple clients and a central server make up this system. Here, a client requests a specific file or other resource stored on the server. After receiving a request, the server generates a response.

There are also more layers to talk about but currently it is not the focus, the focus will be mainly on clean architecture since the application will be built using this type of architecture.

[63]

## 7.1 High level overview of the application architecture

In the application the used architecture will be the “clean architecture” or “the onion architecture” where the application is divided into different layers and each layer is responsible for specific tasks and data manipulation. This layer as mentioned is a type of “layered pattern”. Clean architecture was introduced by **Robert C. Martin (Uncle Bob) and promoted on his own blog<sup>22</sup>**. The whole purpose of such a pattern is that producing a system that is:

- **Testable:** The business logic of the application can be tested easily without the UI, database, or any external source.
- **Decoupled of the user interface:** The front-end of the application can be changed easily without changing or affecting the rest of the system. A Web UI for example can be easily replaced with a console application.
- **Independent of Database:** Changing or migrating to any database should not affect the application or business layer at all. For example, in the business layer we have a mapper such as entity framework that interacts with the database no matter what type of database it is by just changing some line of codes. [64]

Clean architecture contains 4 layers usually, and these are:

- **Domain layer:** In this layer sits most of the system entities such as User.
- **Business or application layer:** This layer is responsible for handling request from the presentation layer and how to process them individually, and handling response from the infrastructure layer and how to handle them, most of the error handling will be in this layer.
- **Presentation layer:** This layer refers to the part of the system that is responsible for handling user input and displaying output to the user, regardless of whether the user interface is a user interface (UI) such as a normal web application or an API.
- **Infrastructure layer:** Here in this layer all the interaction with the database will happen, reading, writing, migrations, and data contexts.

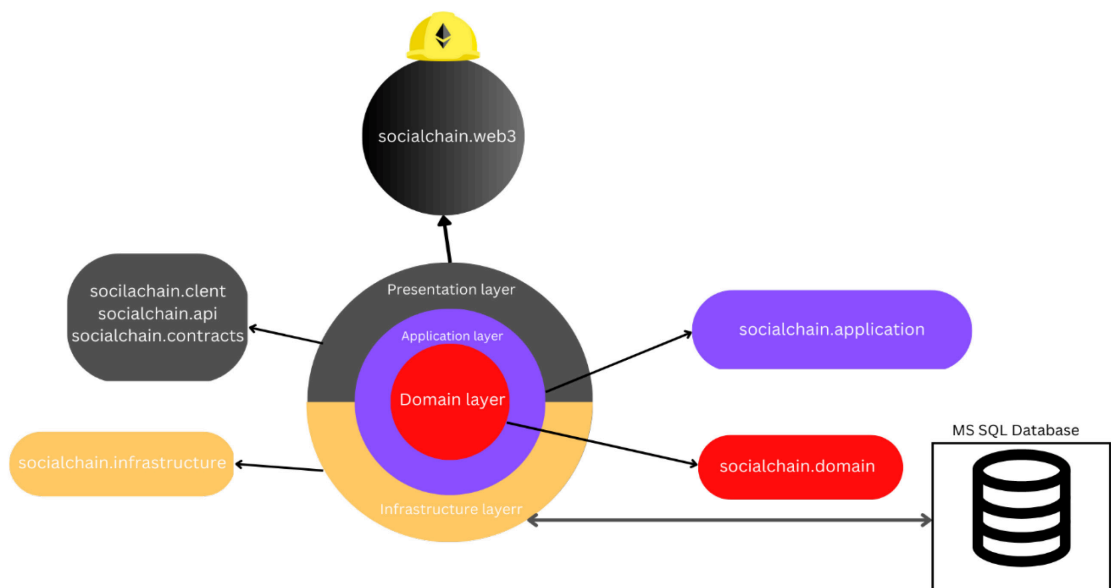
---

<sup>22</sup> The clean code blog (clean architecture): <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>

Since the application have the name “socialchain” it is a good practice to name the layers of the application with the name of the application as shown in the *Figure 52* and such as “socialchain.layername” The project will be in a folder called “socialchain” and have the following layers:

- socialchain.api
- socialchain.client
- socialchain.contracts
- socialchain.application
- socialchain.domain
- socialchain.infrastructure
- socialchain.web3

Where the **presentation layer** will be composed or represented of ( socialchain.api, socialchain.client, socialchain.contracts ) and the **application layer** will be the “socialchain.application”, the **domain layer** is the “socialchain.domain” and for the **infrastructure layer** it will be the “socialchain.infrastructure”, for the web3 project it will be a standalone project that the presentation layer will interact with, but it can joined with the clean architecture in the future. The **web3 layer** where smart contract is and where the blockchain development environment is sits in the folder “socialchain.web3”



*Figure 52. Application's architecture*



## 7.2 Layers communication

Since clean architecture is the used architecture, the communication between the layers will be from outside to inside, in other words the domain layer has no dependency on any of the layers above it instead the layers depending on it. In a clean architecture pattern the presentation layer and infrastructure layer are dependent on the application layer and the application layer in turn depends on the domain layer, but it is also important to mention that in some cases the presentation and infrastructure layer communicate between each other.

## 7.3 Setting up the project's development environment

It is important to mention how the development environment is settled, and how the layers are structured. Basically, for the development environment the following is needed:

- Local blockchain. to run the application on it and deploy the smart contract on it.
- ASP.NET Core API and MS SQL Database.
- Digital wallet.
- Deployment of the smart contract.

### 7.3.1 Local blockchain

For the purpose of testing the smart contract and avoid any mistakes when deploying to a public or test net network, it is important first to try deploying the contract on a local blockchain and try it is functionalities and even in the future it is important to try to attack the contract on a local blockchain to check if there are any loopholes that could stop the program from working in other word (penetration testing). It is right the application uses “hardhat” for the development and deployment of smart contracts and that “hardhat” provides it is own local blockchain. But ganache is used here as local blockchain, and the only reason for that is because ganache can remember the state of the blockchain (changes made), every time it runs, while hardhat local network blockchain, every rerun causes the loss of the old/previous state. So, creating a local blockchain in ganache is as sample as two clicks, creating a workspace and then we will have our own blockchain as shown in the *Figure 53*. So, a workspace called “**SOCIAL CHAIN**” is created for the application.

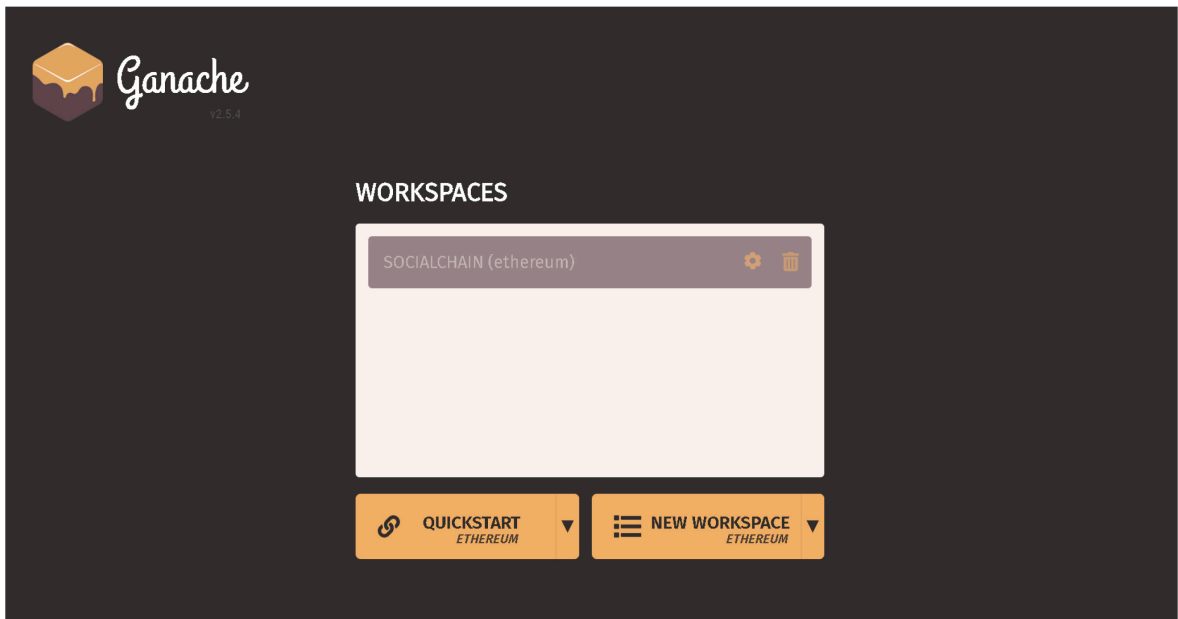


Figure 53. Ganache user interface and workspace

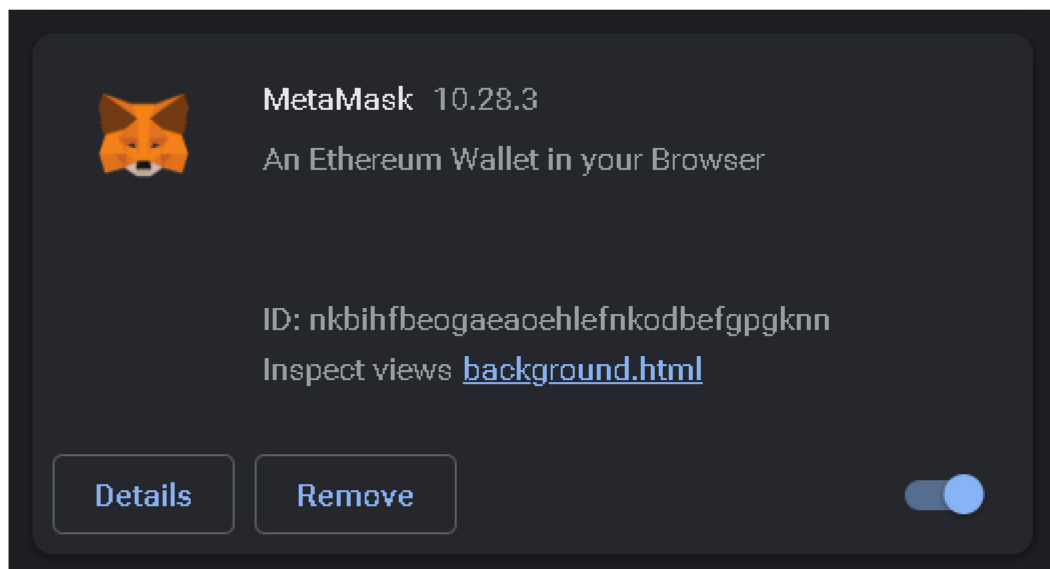
ADDRESS	BALANCE	TX COUNT	INDEX
0xa8aa6b53EcfaB02f6584cc32287Ae16801Cd7E24	99.92 ETH	1	0
0xDc567b6209CFe3bf7Cc2D2e1cEACc8b9d797879	99.97 ETH	8	1
0xA03Ca526b8AB680855b7552Eca8E084aFC3AFA69	99.96 ETH	15	2
0x67b2E68D3DBD67524fB74bb5b6615F2088c51159	99.96 ETH	9	3
0xed445A459c5106A927eC499dE0136Ff3d9EE12CE	100.00 ETH	0	4
0xd6c489f4d298aB754f483a0a22C415239b9379cD	100.00 ETH	0	5
0x4C6e7630D2f76147456eb35a00b595AE0BEe1005	100.00 ETH	0	6
0x577bb7B6CE2885BDEC5Ac9f9e45015823c053F1d	100.00 ETH	0	7

Figure 54. Ganache blockchain

The local blockchain provided by Ganache already has ten Ethereum accounts filled and prepared with 100 ETH each as presented in the Figure 54, and the ability to access their private/public keys also provided as a feature of Ganache. Then using the private keys to add the accounts to the MetaMask digital wallet in the browser.

### 7.3.2 Digital wallet

For the website to interact with the blockchain, a digital wallet must be used to act as a container for the accounts on the blockchain. For this purpose, digital wallets can be installed as extensions on the browsers or as applications in the system. Since the decentralized social network will be built as a web application, The extension must be installed and used in the browser as shown in the *Figure 55*. MetaMask will be used as the main digital wallet for the development environment. It is an Ethereum wallet that runs in the browser. MetaMask as extension for browsers is available in (Google Chrome, Firefox, Brave, and Microsoft Edge). It is also available as a mobile app for both iOS and Android devices.



*Figure 55. MetaMask Digital wallet*

For the digital wallet to communicate with specific blockchain network, it must be configured with a specific configuration such as RPC Server URL and port where the IP is listening to as shown in the *Figure 56*. This can be done by going to the settings of the digital wallet and to the network's settings, by adding our ganache local network configurations.

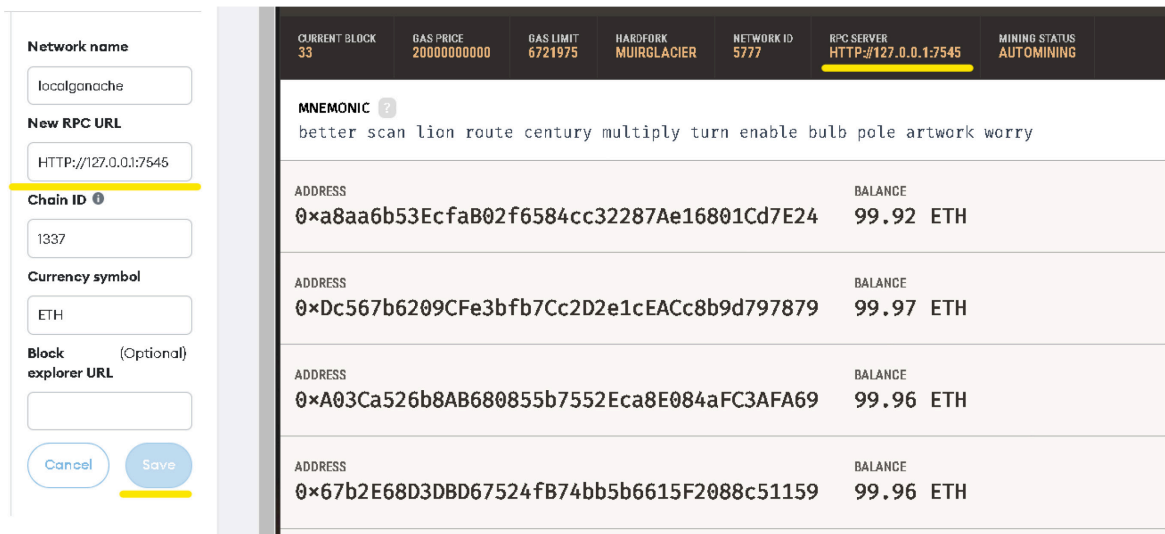


Figure 56. MetaMask network configs

And to add accounts that are created in the local network to the digital wallet that is connected to the same network, can be done by simply clicking on “import account” and add the private key from the ganache user interface application by clicking on the key icon next to the account we want to add as shown in the Figure 57.

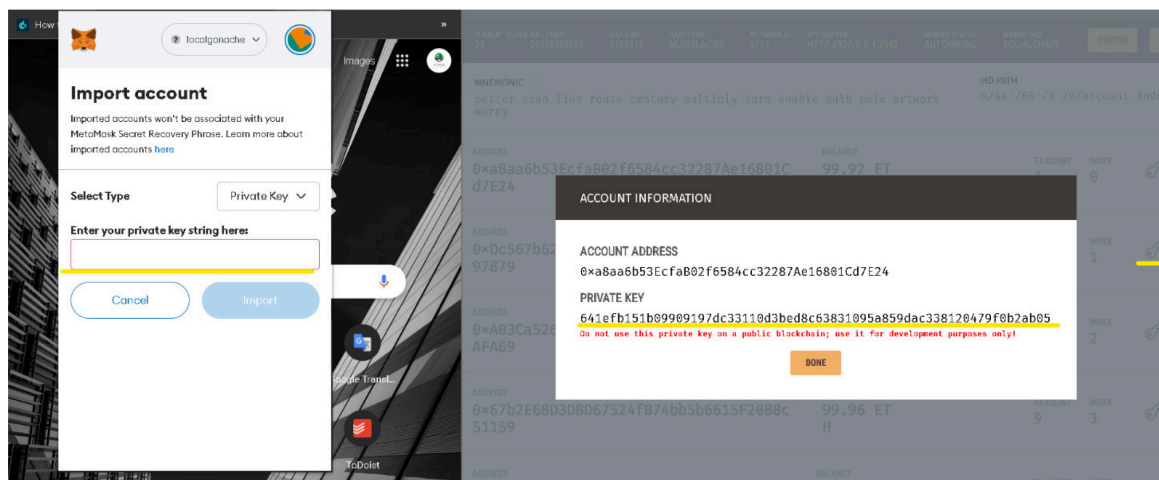


Figure 57. Adding Ethereum account to digital wallet

### 7.3.3 MS SQL Database and ASP.NET Core API

In order to implement the JWT authorization, the refresh token of all users must be stored, so every time the user request new access token using the refresh token, for purpose of accessing the website and to get new access token, the need to compare the user refresh token is equal to the one in the database, if so the user will be granted new access token that will be stored in the HTTP only cookie (server-side cookie). So, for that purpose an API with all the necessary end points to do all the job will be created, and that API communicates with

the MS SQL Database. The approach used in the application to generate/create the database is called “**code first migration**” where the domain classes’ a should be already created as shown in *Figure 58* and migration should be done and from that entity/class, the database is then created (if not already created), then the table is created in the database, all by providing the connection string of our database to the ASP.NET Core API application.

```
1
2 using System.ComponentModel.DataAnnotations;
3
4 namespace socialchain.domain.Entities;
5
6 public class User
7 {
8     [Key]
9     public string AccountAddress { get; set; } = null!;
10    public string RefreshToken { get; set; } = string.Empty;
11    public DateTime TokenCreated { get; set; }
12    public DateTime TokenExpires { get; set; }
13 }
14
```

*Figure 58. C# User entity/class*

In the *Figure 58* this entity will represent the table in the database, and “**AccountAddress**” here act the primary key of the table, that’s why it is annotated with the “[**Key**]” attribute (mainly because code first migration requires informing the database and Entity Framework about the primary key of the table). In the back-end *User* is the only and main entity of the application, to make the first migration from the code to database, there are several steps should be taken:

- 1. Object relational mapper package:** Object-Relational Mapping (ORM) is a technique used to map the objects in the code to the tables and columns in a database. It allows to interact with the database using objects rather than composing SQL statements directly. Entity Framework is a prominent Object-Relational Mapping (ORM) framework for .NET applications. It provides a set of tools and libraries that enable us to define the mapping between the objects and the database, and it manages communication with the database on our behalf. Entity Framework will be used as the object relational mapper in the application, it provides us with functionality such as migrations from code to database, or updating database tables according to our entities, or writing C# methods to get data from database instead of writing SQL Statements.

2. **Writing the DB Context code that is recognized by the Entity framework:** The `DbSet<User>` in the *Figure 59* represent the table in the database. And as shown also ensuring that the database should be created if not already created.

```

1 using Microsoft.EntityFrameworkCore;
2 using socialchain.domain.Entities;
3
4 namespace socialchain.infrastructure.DbContexts
5 {
6     8 references
7     public class DataContext : DbContext
8     {
9         0 references
10        public DataContext()
11        {
12        }
13        0 references
14        public DataContext(DbContextOptions<DataContext> options) : base(options)
15        {
16            Database.EnsureCreated();
17        }
18        4 references
19        public DbSet<User> Users { get; set; }
20    }
21 }

```

*Figure 59. Application's database context*

3. **Notifying the application about the DB Context:** While the run time process of the application the application should know about the database context and that it is related to our database (by providing the connection string from the “*appsetting.json*” file).

```

1 services.AddDbContext<DataContext>(options =>
2 {
3     options.UseSqlServer(builder.Configuration.GetConnectionString("SocialChainDB"),
4         x => x.MigrationsAssembly("socialchain.infrastructure"));
5 });
6 return services;

```

4. **Running the migration command:** Migration is the step where the code automatically turned to SQL Statements that is then executed by the back end to populate the database in order to do that, it can be done by the „*package manger console*“ provided by the Visual Studio IDE or using normal command line .

Using the package manger, we can run the following command:

```
PM> Add-migration "First-SocialChainMigration"
```

Using the command line:

```
dotnet ef migrations add "First-SocialChainMigration"
```

However, it is important to keep in mind that necessary packages are required to run the commands such as:

- Microsoft.EntityFrameworkCore.Tools
- Microsoft.EntityFrameworkCore.Desig

The result of the migration will be as shown in the *Figure 60*.

```

public partial class FirstSocialChainMigration : Migration
{
    /// <inheritdoc />
    0 references
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Users",
            columns: table => new
            {
                AccountAddress = table.Column<string>(type: "nvarchar(450)", nullable: false),
                RefreshToken = table.Column<string>(type: "nvarchar(max)", nullable: false),
                TokenCreated = table.Column<DateTime>(type: "datetime2", nullable: false),
                TokenExpires = table.Column<DateTime>(type: "datetime2", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Users", x => x.AccountAddress);
            });
    }

    /// <inheritdoc />
    0 references
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Users");
    }
}

```

Figure 60. EF migration result

5. **Updating the database from the migration:** In this step the database will be populated and the DB Context will run and try to run the necessary configurations, the following command must be used for „*console package manager*“:

```
PM> Update-Database
```

And from the terminal:

```
dotnet ef database update
```

And the command will affect the database as shown in the *Figure 61*.

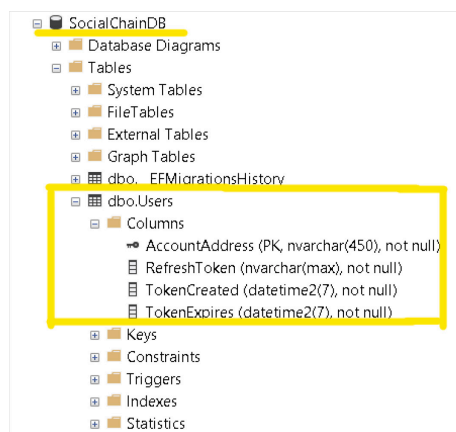
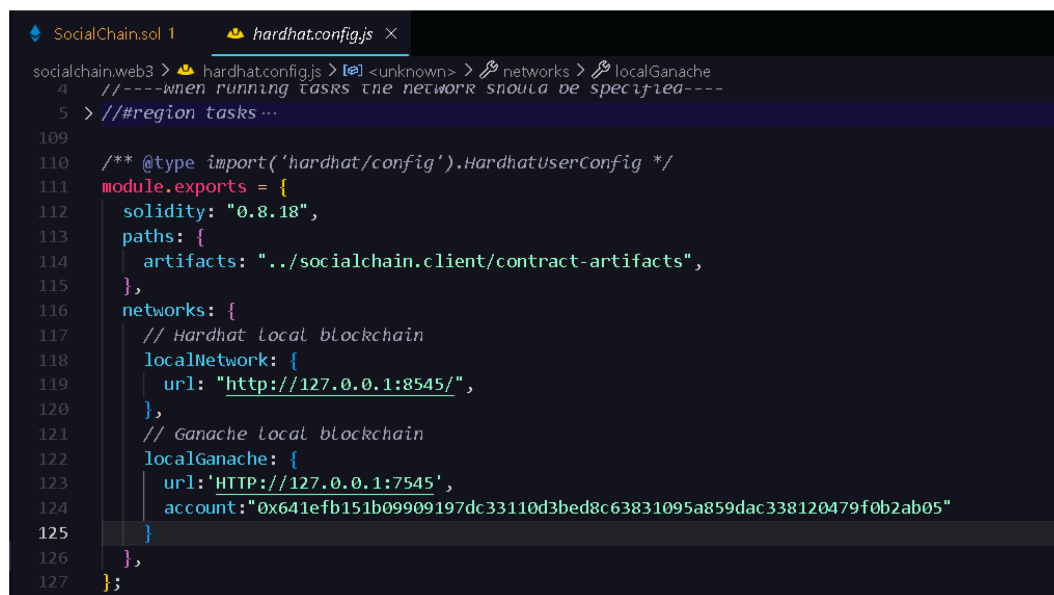


Figure 61. Updating database from migration result

### 7.3.4 Hardhat configurations (contract's deployment environment)

Hardhat is the main player here, without it, a test of the smart contract cannot be done, a deployment of the contract to the specified network cannot be done, without it debugging the smart contract is almost impossible. For the contract to be deployed to the specific network, there are few steps must be taken, and these are:

1. **Edit hardhat configs to connect to the local blockchain:** To make hardhat connect to the local Ganache blockchain, the hardhat configuration file must be modified and it is called „**hardhat.config.js**“, inside the module object there should be an object called **networks**, and this object can have as many objects as networks we want to connect to. Even with this file, deploying the contract to a real blockchain is possible. As shown in *Figure 62* below the "localGanche" object within the "networks" object, the RPC URL of the blockchain and the account for deploying the desired contract are available.



```

socialchain.web3 | hardhat.config.js | <unknown> | networks | localGanche
4 //---when running tasks the network should be specified---
5 > //#region tasks...
109
110 /** @type import('hardhat/config').HardhatUserConfig */
111 module.exports = {
112   solidity: "0.8.18",
113   paths: {
114     artifacts: "../socialchain.client/contract-artifacts",
115   },
116   networks: {
117     // Hardhat local blockchain
118     localNetwork: {
119       url: "http://127.0.0.1:8545/",
120     },
121     // Ganache local blockchain
122     localGanche: {
123       url: 'HTTP://127.0.0.1:7545',
124       account: "0x641efb151b09909197dc33110d3bed8c63831095a859dac338120479f0b2ab05"
125     }
126   },
127 };

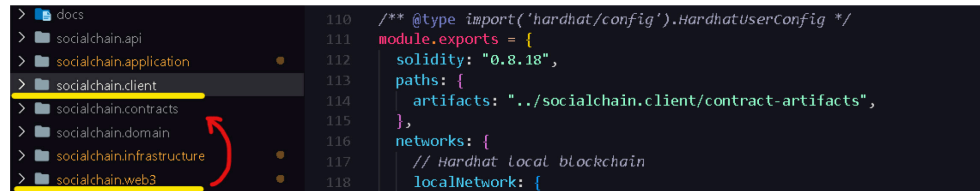
```

*Figure 62. Hardhat network configs*

2. **Compiling of the contract:** This step is crucial and mandatory prior to contract deployment. Compiling the contract results in two outputs. **First**, transferring it to a format that is accepted by the EVM. When the contract is compiled, the byte code is obtained which is a format considered as a low-level machine code that can be understood by the EVM. **Second**, the so-called ABI (Application binary interface) is obtained, this is a JSON file that let the client side interact with the smart contract and gets the functions, parameters, variables available in the smart contract, without the need for dealing with low-level machine code. To compile the smart contract using hardhat, there are several steps that must be taken:



- a. **Specifying the output location:** It is necessary to specify the location of the ABI and contract compilation results folder in the “**hardhat.config.js**” file as illustrated in the *Figure 63*. Instructing hardhat to place the compilation result in the “**sociachain.client**” layer, specifically in the “**contract-artifacts**” folder, is an important step. If the folder doesn't exist, it will be created, and if it already exists, any existing folders and files will be overwritten.



```

110  /** @type import('hardhat/config').HardhatUserConfig */
111  module.exports = {
112    solidity: "0.8.18",
113    paths: {
114      artifacts: "../socialchain.client/contract-artifacts",
115    },
116    networks: {
117      // Hardhat local blockchain
118      localNetwork: {

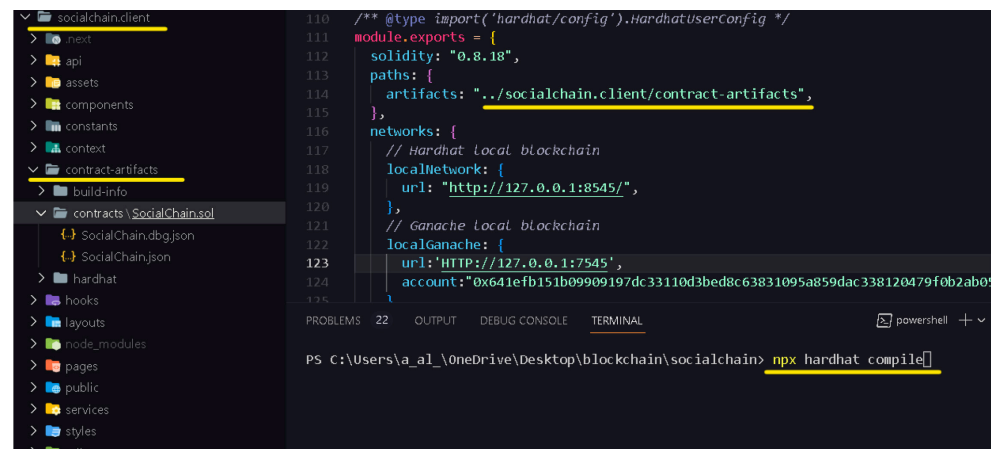
```

*Figure 63. Hardhat compilation configs*

- b. **Running the compilation command:** Now the provided command by hardhat can be executed, which is:

```
npx hardhat compile
```

The result will be as shown in the *Figure 64*.



```

110  /** @type import('hardhat/config').HardhatUserConfig */
111  module.exports = {
112    solidity: "0.8.18",
113    paths: {
114      artifacts: "../socialchain.client/contract-artifacts",
115    },
116    networks: {
117      // Hardhat local blockchain
118      localNetwork: {
119        url: "http://127.0.0.1:8545/",
120      },
121      // Ganache local blockchain
122      localGanache: {
123        url: "HTTP://127.0.0.1:7545",
124        account: "0x641efb151b09909197dc33110d3bed8c63831095a859dac338120479f0b2ab09
125

```

PS C:\Users\va\_al\OneDrive\Desktop\blockchain\sociachain> npx hardhat compile

*Figure 64. Hardhat compilation result*

3. **Deployment of the contract to specific network:** To deploy a specific contract to the network there are also some several steps to take care of:

- a. **Creating of JavaScript script file for deployment purpose:** a JavaScript script using the hardhat library must be created to be able to deploy the contract to the Ethereum blockchain, since already hardhat includes ethers.js as built-in library inside their library there is no need to import or install ethers.js library in the web3 layer (socialchain.web3), as shown in the *Figure 65*. The contract name, specified in the web3 layer, needs to be provided for deployment. Additionally, the

deployment of the contract is logged along with its specific address. This log is displayed when running the hardhat node, as logging in hardhat occurs during runtime. In Ganache UI, the latest deployed contract and its address are also visible.

```

socialchain.web3 > scripts > deploy.js > main
1  const hre = require("hardhat");
2
3  async function main() {
4
5    //[[1]- Deployment of the contract
6    const SocialChain = await hre.ethers.getContractFactory("SocialChain");
7    const socialChain = await SocialChain.deploy();
8    await socialChain.deployed();
9
10   console.log("SocialChain contract deployed to:", socialChain.address);
11 }
12
13 // We recommend this pattern to be able to use async/await everywhere
14 // and properly handle errors.
15 main().catch((error) => {
16   console.error(error);
17   process.exitCode = 1;
18 });

```

Figure 65. Hardhat deploying script

- b. Deployment of the contract:** The following command must be used to deploy the contract to the blockchain, but the network must be provided in which the contract will be deployed on:

```
npx hardhat run --network localGanache scripts/deploy.js
```

In Ganache the UI will be updated automatically and the following will be shown as in the *Figure 66*.

TX HASH	FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0x83118206f966d8d1fa2e1727704bd34ff605a12f67296045764ba234fc931273	0x93234C5d67487e8722F3291cC4105D1f0b483B6b	0x85365158Ed31cF2d8D9E8c070898d19419eA5B6E	189663	0

Figure 66. Contract deployment result

## 8 IMPLEMENTATION AND EVALUATION

Now it is important to describe how the application internally operates, from the side of blockchain most important and from client and back-end sides. Mainly trying to look at the flow of the application functionality, like how data flows between the layers for specific functionality.

### 8.1 Application's smart contract

The application's smart contract is the heart or the core of the application, it must be explained in detail since the focus is building a DApp on the Ethereum blockchain or related networks. The smart contract will have the name "SocialChain" same as the whole application name.

#### 8.1.1 Contract's states and variables

States and variables are used to remember something important in the contract, they are persistent data and are stored in the contract storage, that will help with contract's functionalities. In the current application the contract has the following variables or states as shown in the *Figure 67*.



```
1 address payable public owner;
2 uint256 public totalUsers = 0;
3 uint256 public totalPosts = 0;
4 uint256 public totalComments = 0;
5 uint256[] private postIds;
```

*Figure 67. Contract's states and variables*

1. **owner:** Will be used to identify the owner of the contract, the value of it will be populated when the contract is first deployed but assign it to the one who deployed the contract (we) in the contract's constructor.
2. **totalUsers:** This variable will be used to get info about how many users are currently registered on the platform, when user is registered to the platform successfully this variable will be increased by one. But also, this variable will be used as an ID generator, after the

user successfully registered to the platform his ID will be obtained from the totalUsers after the increment.

- totalPosts** and **totalComments**: The total posts variable and totalComments have the same idea as the totalUsers has.
- postIds**: This is an array that's contain all the Ids of the posts, it will be used to retrieve posts in the front-end and then call a function to retrieve a post by its id. Every time a post is created, a postid will be pushed to this array.

### 8.1.2 Contract's mappings

Mappings are a data structure of (key, value) where a key points to a value. These will be acted as the database of the application where users, posts, comments data and more will be stored using these mappings. These mappings will be populated with the values mostly when the functions are called. The current implementation of the contract has the mappings as shown in the *Figure 68*.

```
1 mapping(address => User) private users;
2 mapping(string => address) private userAddressFromUserName;
3 mapping(string => bool) private usernames;
4
5 mapping(uint256 => Post) private posts;
6 mapping(address => uint256[]) private userPosts;
7 mapping(uint256 => mapping(address => bool)) private postLikers;
8
9 mapping(uint256 => Comment) private comments;
10 mapping(address => uint256[]) userComments;
11 mapping(uint256 => uint256[]) private postComments;
```

*Figure 68. Contract's mapping*

- users**: This mapping will be used to store a user object that instantiated from user struct with the specific corresponding address, the mapping will be also used to retrieve a user object from the corresponding key (address).
- userAddressFromUserName**: The mapping will be used to store a user address by his username, so to retrieve user address by username.
- usernames**: The purpose of this mapping is to indicate which username is reserved and can't be taken, so it will be a list of all usernames in the platform with a Boolean value.

4. **posts:** Used to store/retrieve a post by postId.
5. **userPosts:** This mapping has all the postIds that user owns, each user's address has an array of integers that's represents the postids that the user owns. When a registered user creates a post, the postId will be appended to the array of posts that correspond to the author of the post (his account address).
6. **postLikers:** This mapping has a kind of a complex structure, the key to this mapping is the postId and the value will be another mapping of address to Boolean, in other words each post will have a list of likers (users who liked the posts).
7. **comments:** Have the same concept as **users** mapping.
8. **userComments:** Have the same concepts as **userPosts** mapping.
9. **postComments:** This mapping basically indicated all the comments ids on a specific id.

### 8.1.3 Contract's events

Event as explained in 5.5.3 can be used to log data after a successful execution of an operation in the contract. Events can also be useful to return data to the client-side as in normal APIs there are responses and requests data objects. The current implementation of the application has the events as shown in the *Figure 69*.

```
1  event logRegisterUser(address userAddress, uint256 userId);
2  event logPostCreated(address _author, uint256 _userId, uint256 _postId);
3  event logCommentCreated(
4      address _author,
5      uint256 _commentId,
6      uint256 _postId,
7      uint256 _likeCount,
8      uint256 _reportCount,
9      uint256 _timeStamp,
10     string _content
11 );
```

*Figure 69. Contract's events*

1. **logRegisterUser:** This event will be emitted when a user successfully register to the contract storage or the platform, the user address and userId will be emitted from the event.
2. **logPostCreated:** Same as **logRegisterUser** the event will be emitted after successful creation of post, but more parameters will be emitted and most importantly is the **\_author**.
3. **logCommentCreated:** Also, the same applies to this event, with the difference of the emitted parameters.

The parameters that are emitted from the events can be modified according to the need of the owner or the need of the client-side application (web app/API).

### 8.1.4 Contract's functions

Functions are the most important code in the contract, they determine how data will be changed and how the contract operates, functions should be handled and coded carefully, they can be the main danger of the contract, or the gates to bugs. They can be used by hackers to steal ETH of the owner, or even stop the whole contract from operating. In the current implementation we have the following functions:

1. **userNameAvailable:** Since the main thing that makes a user unique from other users is the username and public address. This function will check that a chosen Username is available before successful registration of the new user.

The function will return a Boolean value that will indicate if a username is available or not by checking the mapping that contains a Boolean value for each username as shown in the *Figure 70* That shows whether a username is already registered or not.

```
1  function userNameAvailable(string memory _username)
2      public
3      view
4      returns (bool status)
5  {
6      return !usernames[_username];
7  }
```

*Figure 70. Username availability smart contract's function*

2. **registerUser**: As the function's name indicates, the purpose of this function is to register the user to the contract storage (to the platform) as shown in the *Figure 71*, the user can register with username, name, profile image hash (IPFS CID), cover image hash (IPFS CID), with a bio and birth date, also he can choose if he wants to show his username on the platform or no (all of this done on the client-side). The function has two custom modifiers, and these are “**checkUserNameTaken**” and second is the “**checkUserNotRegisteredByAddress**” both are explained in the section 8.1.5.

```
1  function registerUser(  
2      string memory _username,  
3      string memory _name,  
4      string memory _lmgHash,  
5      string memory _coverHash,  
6      string memory _bio,  
7      uint256 birthDate,  
8      bool showUserName  
9  )  
10 public  
11     checkUserNotRegisteredByAddress(msg.sender)  
12     checkUserNameTaken(_username)  
13 {  
14     //Attack prevented  
15     //[1]- reserve the user name  
16     usernames[_username] = true;  
17     //[2]- increase counter of total users  
18     totalUsers = totalUsers + 1;  
19     //[3]- set id of new user to the counter  
20     uint256 id = totalUsers;  
21     //[4]- add User object to our mapping of (address => user) for the specific registered user  
22     users[msg.sender] = User(  
23         id,  
24         msg.sender,  
25         _username,  
26         _name,  
27         _bio,  
28         birthDate,  
29         showUserName,  
30         _lmgHash,  
31         _coverHash,  
32         accountStatus.Active  
33     );  
34     //[5]- set add user address [calling of the function] to our mapping of (username => address) to get user address  
35     by username  
36     userAddressFromUserName[_username] = msg.sender;  
37     //[6]- emit the event  
38     emit logRegisterUser(msg.sender, id);  
39 }
```

Figure 71. Register user smart contract's function

3. **getUser:** The function is used to retrieve a user object from the mapping **users** by providing the Ethereum public address of the user as shown in the *Figure 72* below.

```
1  function getUser(address accountAddress)
2      public
3      view
4      returns (
5          uint256 id,
6          string memory userName,
7          string memory name,
8          string memory bio,
9          uint256 birthDate,
10         bool showUsername,
11         string memory imageHash,
12         string memory coverHash
13     )
14     {
15         User memory u = users[accountAddress];
16
17         return (
18             u.id,
19             u.userName,
20             u.name,
21             u.userBio,
22             u.birthDate,
23             u.showUserName,
24             u.profileImgHash,
25             u.profileCoverImgHash
26         );
27     }
```

*Figure 72. Get user contract's function*

4. **createPost:** This function is used to create a post in the contract (store new post by a user), this function is called by the user. The function has the custom modifier called “**onlyAllowedUser**” as shown in the *Figure 73*. Means only users with the Enum status active will be able to create a new post, a new post request should have (the author address, the post content in text, the profile image hash if exists), also instead of sending “**\_accountAddress**” with the function’s parameters we can simply use “**msg.sender**” which is the user that called the function. The function will also emit the event “**logPostCreated**” to log the created post and the owner of it.



```
1  function createPost(string memory _postdescription, string memory _imghash)
2      public
3      onlyAllowedUser(msg.sender)
4  {
5      totalPosts = totalPosts + 1;
6      uint256 postId = totalPosts;
7      posts[postId] = Post(
8          postId,
9          msg.sender,
10         _postdescription,
11         _imghash,
12         block.timestamp,
13         0,
14         0,
15         postStatus.Active
16     );
17     //each user will have an array of postId that he posted
18     userPosts[msg.sender].push(postId);
19     postIds.push(postId);
20     emit logPostCreated(msg.sender, users[msg.sender].id, postId);
21 }
```

Figure 73. Create post contract's function

- 5. `getPostById`:** The function is used to retrieve a post object that is initiated from the post structure by its id. The function has two custom modifiers that indicates the post should be active nor deleted, nor banned, also a modifier that indicates only registered and active users in the contract are allowed to call this function as shown in the *Figure 74* below.

```
1  function getPostById(uint256 _postId)
2      public
3      view
4      onlyAllowedUser(msg.sender)
5      onlyActivePost(_postId)
6      returns (Post memory)
7  {
8      return posts[_postId];
9  }
```

Figure 74. Get post by id contract's function

- 6. `getUserPosts`:** The function accepts a parameter of type address which is the user address and returns the list of all his posts' objects that are initiated from the post struct. The function is also restricted by the modifier "**onlyAllowedUser**" as shown in the *Figure 75* below.

```
1  function getUserPosts()
2      public
3      view
4      onlyAllowedUser(msg.sender)
5      returns (Post[] memory)
6  {
7      uint256[] memory userPostIds = userPosts[msg.sender];
8      Post[] memory userPostsTemp = new Post[](userPostIds.length);
9      for (uint256 i = 0; i < userPostIds.length; i++) {
10         userPostsTemp[i] = posts[userPostIds[i]];
11     }
12     return userPostsTemp;
13 }
```

Figure 75. Get user posts contract's function

- 7. `getPostIds`:** This function is one of the most important functions in the contract, it is responsible for retrieving the posts of the platform that are showing in the feed page of the users. When trying to create this function, retrieving all the posts in one request is a bad idea, if we imagine we have ten thousand posts in the platform and user login and we forward him to feed, and feed page tries to make a request to load all posts, this operation is expensive from side of user experience because he has to wait for a really long time specially when interacting with a blockchain and not a normal server, and also it is an expensive operation from the side of memory consumption on client-side. The function uses the idea of **pagination** as shown in the *Figure 76*. Which is a way to retrieve data by chunks, for example the feed page will be divided into pages and each page will have a specific number of posts loaded. For instance, on page one we load 20 posts by calling the function and sending (`_page = 1`, `_perPage = 20`) and so on. The algorithm is quietly used in such platforms or any blogging platforms. The function also has custom modifiers to make sure it is called only by allowed/registered users. To make the function lighter and have faster response the function returns an array of posts ids instead of a whole array of objects, and then for each post id on the client side the **`getPostById`** function is called.

```
1 function getPostIds(uint256 _page, uint256 _perPage)
2     public
3     view
4     onlyAllowedUser(msg.sender)
5     returns (uint256[] memory)
6 {
7     uint256 start = (_page - 1) * _perPage;
8     uint256 end = start + _perPage;
9     if (end > postIds.length) {
10        end = postIds.length;
11    }
12    uint256[] memory result = new uint256[](end - start);
13    for (uint256 i = start; i < end; i++) {
14        result[i - start] = postIds[i];
15    }
16    return result;
17 }
```

Figure 76. Get platform posts - pagination

8. **likePost**: This function can be called only if the post is active and if the user calling the function is a valid registered user, the function as the name indicates and as shown in the Figure 77. Is used to like a post in the platform and it receives the post id as a parameter, the user that likes the post is the one who call the function and it is represented as “**msg.sender**”.

```
1 function likePost(uint256 _postId)
2     public
3     onlyAllowedUser(msg.sender)
4     onlyActivePost(_postId)
5 {
6     // [1]- The post should not be liked already by the specific user (should return false)
7     require(!postLikers[_postId][msg.sender]);
8     // [2]- increase number of likes for the specified post:
9     posts[_postId].likeCount = posts[_postId].likeCount + 1;
10    // [3]- set that the specified user liked the post
11    postLikers[_postId][msg.sender] = true;
12 }
```

Figure 77. Likes post contract's function

9. **unLikePost**: Un liking a post works same as liking a post, but instead of increasing number of likes for the post we decrease it by one as illustrated in the Figure 78.

```
1 function unLikePost(uint256 _postId)
2     public
3     onlyAllowedUser(msg.sender)
4     onlyActivePost(_postId)
5     {
6         //[1]- Post should be like already by the specific user
7         require(postLikers[_postId][msg.sender]);
8         //[2]- decrease number of likes for the specified post:
9         posts[_postId].likeCount = posts[_postId].likeCount - 1;
10        //[3]- set that the specified user liked the post
11        postLikers[_postId][msg.sender] = false;
12    }
```

Figure 78. Unlike post contract's function

- 10. isLikedByAddress:** The function is used to check if a post is already liked by a user, and of course the modifiers are used as shown in the *Figure 79*, to make sure the post already exists, and the function is called by registered/valid user. The purpose of this function is when a user in the client-side try to like a post that he already liked, we call this function and upon the Boolean result we call the function “**unLikePost**” otherwise we call “**likePost**”.

```
1 function isLikedByAddress(uint256 _postId, address _userAddress)
2     public
3     view
4     onlyActivePost(_postId)
5     onlyAllowedUser(_userAddress)
6     returns (bool)
7     {
8         return postLikers[_postId][_userAddress];
9     }
```

Figure 79. is liked by address contract's function

- 11. createComment:** Create comment function has the same behavior as the “**createPost**” function as illustrated in the *Figure 80*.

```
1 function createComment(uint256 _postId, string memory _comment)
2     public
3     onlyAllowedUser(msg.sender)
4     onlyActivePost(_postId)
5 {
6     //[1]- make sure comment is not empty
7     bytes memory tempStringValidation = bytes(_comment);
8     require(tempStringValidation.length != 0, "Comment can not be empty !");
9     //[2]- Id incremental
10    totalComments = totalComments + 1;
11    uint256 commentId = totalComments;
12    //[3]- Adding the post to the mapping
13    comments[commentId] = Comment(
14        commentId,
15        msg.sender,
16        _postId,
17        _comment,
18        0,
19        0,
20        block.timestamp,
21        commentStatus.Active
22    );
23    //[4]- Adding the comment to the post mapping (Each post can have many comments)
24    postComments[_postId].push(commentId);
25    //[5]- Log the created comment
26    // event logCommentCreated(address _author,uint _commentId,uint _postId, uint
27    _likeCount, uint _reportCount, uint _timeStamp, string _content);
28    emit logCommentCreated(
29        msg.sender,
30        commentId,
31        _postId,
32        0,
33        0,
34        block.timestamp,
35        _comment
36    );
37 }
```

Figure 80. Create comment contract's function

12. **getCommentById**: This function also has the same behavior as the “getPostById” function.

```
1 function getCommentById(uint256 _commentId)
2     public
3     view
4     onlyAllowedUser(msg.sender)
5     onlyActiveComment(_commentId)
6     returns (Comment memory)
7 {
8     return comments[_commentId];
9 }
```

Figure 81. Get comment by Id contract's function.

**13. getPostComments:** Same goes for this function the idea of pagination is implemented. In the front-end side or client-side there will be something like “show more comments” and every time user clicks on it, more comments will be loaded, each click is represented as call of this function with different list number request as shown in the parameters of the function in the *Figure 82*.

```
1  function getPostComments(  
2      uint256 _listNumber,  
3      uint256 _commentPerList,  
4      uint256 _postId  
5  )  
6      public  
7      view  
8      onlyAllowedUser(msg.sender)  
9      onlyActivePost(_postId)  
10     returns (Comment[] memory)  
11  {  
12     uint256 startIndex = (_listNumber - 1) * _commentPerList;  
13     uint256 endIndex = startIndex + _commentPerList;  
14     //to avoid IndexOutOfRangeException  
15     if (endIndex > postComments[_postId].length) {  
16         endIndex = postComments[_postId].length;  
17     }  
18     Comment[] memory commentsResponse = new Comment[](  
19         endIndex - startIndex  
20     );  
21     uint256 tempCommentId = 0;  
22     for (uint256 i = startIndex; i < endIndex; i++) {  
23         tempCommentId = postComments[_postId][i];  
24         commentsResponse[i - startIndex] = comments[tempCommentId];  
25     }  
26     return commentsResponse;  
27 }
```

*Figure 82. Get post's comments contract's function*

### 8.1.5 Contract's modifiers

Modifiers, as explained before, are a way to restrict a code from execution before it gets executed. It acts as an if condition before a function continues, it is execution. Modifiers are a useful way to protect the contract from getting attacked or letting anyone find potential vulnerabilities. Modifiers are a good feature because they can utilize the “**require**” keyword, this keyword can throw an error with a message of type string that can be specified by the developer. In the current implementation the following modifiers are used as shown in the *Figure 83*. However, more could be added in the future, to make the flow of execution more precise and more secure.

1. **checkUserNotRegisteredByAddress:** To prevent an already registered user from registering again.
2. **checkUserNameTaken:** Each user must have a unique username and this modifier is used for the purpose of that limitation.
3. **onlyAllowedUser:** This is the most important modifier for the contract, almost all the functions should have this modifier applied to them, the modifier restrict that only registered users can interact with the smart contract functionality, the modifier acts as the authorization of the smart contract.
4. **onlyActivePost:** Only not deleted, and not banned post can be interacted with.
5. **onlyActiveComment:** Only not deleted, and not banned comment can be interacted with.

```
1  modifier checkUserNotRegisteredByAddress(address userAddress) {
2      require(
3          users[userAddress].status == accountStatus.NP,
4          "User already registered"
5      );
6      _;
7  }
8  modifier checkUserNameTaken(string memory userName) {
9      require(!usernames[userName], "Username already taken");
10     _;
11 }
12
13 modifier onlyAllowedUser(address userAddress) {
14     require(
15         users[userAddress].status == accountStatus.Active,
16         "Not a Registered User!"
17     );
18     _;
19 }
20
21 modifier onlyActivePost(uint256 postId) {
22     require(
23         posts[postId].status == postStatus.Active,
24         "Not an active post"
25     );
26     _;
27 }
28
29 modifier onlyActiveComment(uint256 commentId) {
30     require(
31         comments[commentId].status == commentStatus.Active,
32         "Not an active comment"
33     );
34     _;
35 }
```

Figure 83. Contract's modifiers

## 8.2 Demonstration of the main functionalities

The main and the most important part of the research is the real demonstration of it and showing it in real how it operates. In this part the focus will be mostly on how the user is registered to the platform and how his data and routes of the website are protected. The good thing about the current implementation is adding the feature of combining the Digital wallet and the JWT authorization pattern.

### 8.2.1 Registering of new users to the platform

It is better to write the algorithm first before the real code or the implementation is written. Since the idea of combining the JWT with the web3 is kind of a complicated process. The flow of data when the user tries to register to the platform will be in order as below:

1. **Client-side:** User enters his details and the info provided should pass the form validation as shown in the *Figure 87*.
2. **Digital wallet:** As illustrated in *Figure 87*. User should connect his wallet and confirm the transaction to register him and store him in the contract storage.
3. **Blockchain contract:** registering the user to the contract storage, only if the provided username is not taken.
4. **Back-end:** Generating JWT tokens for the user by sending a message from back-end to front-end that he has to sign it through his digital wallets as presented in *Figure 89*, only if he signed the message and if the message is verified that has been signed by the same public address after decryption of the message, user will obtain access and refresh tokens that they will be appended as HTTP Only cookies, and then user will be able to access the website's protected routes.
5. **Response back to client-side:** After successful generation of the JWT token from the client-side the user will be forwarded to the /home route as in the *Figure 90*, this process for routing or redirecting will be handled by the middleware.js of the Next.js application.

It is better to demonstrate the data flow as a flow chart other than text however the algorithm that shown in *Figure 84* is also available as PDF<sup>23</sup> for higher quality. And available as a text file<sup>24</sup> for full demonstration of the algorithm.

---

<sup>23</sup> User registration workflow PDF file: <https://shorturl.at/euyXZ>

<sup>24</sup> User registration workflow text file: <https://shorturl.at/dvzCF>



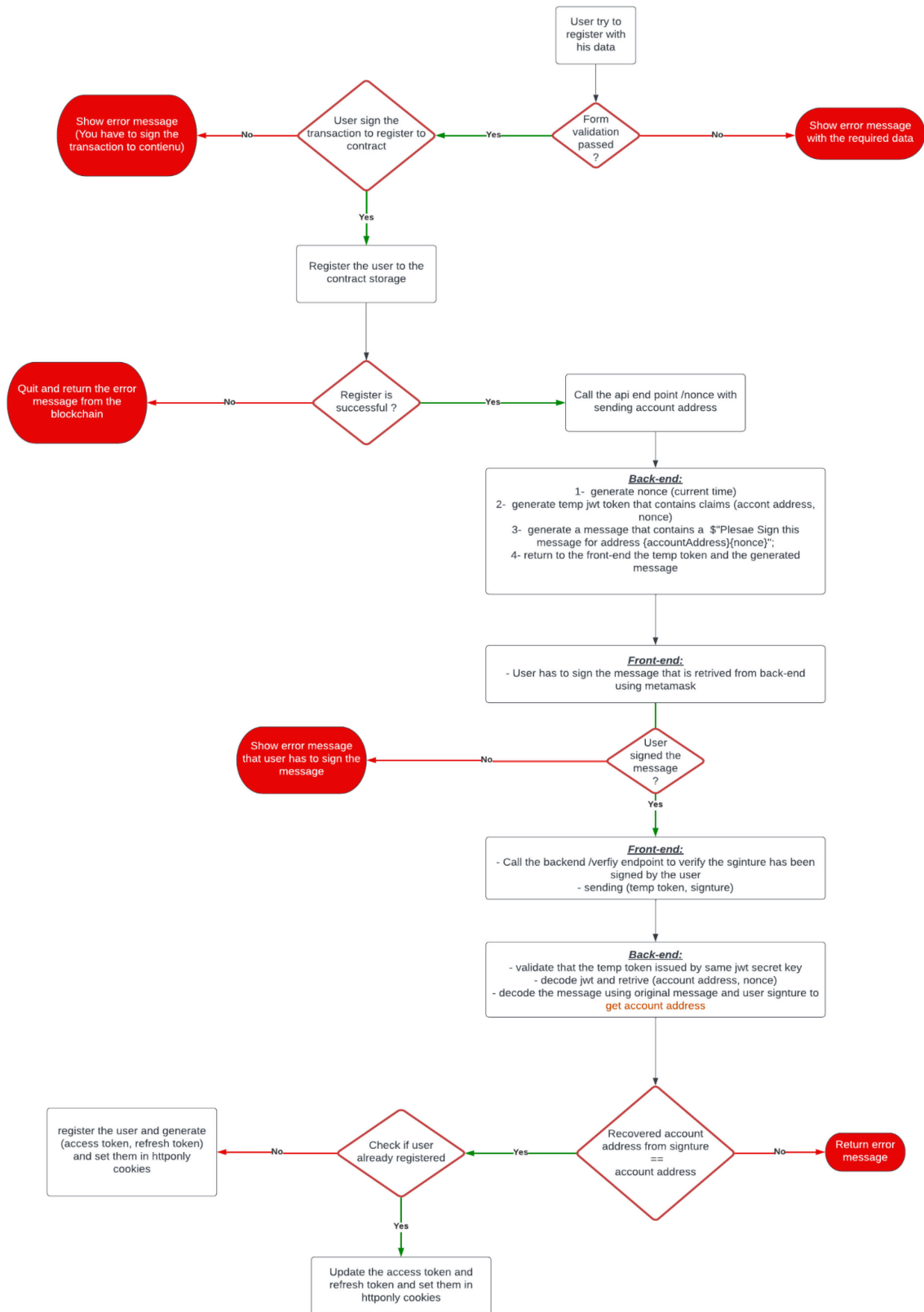


Figure 84. User registration flowchart

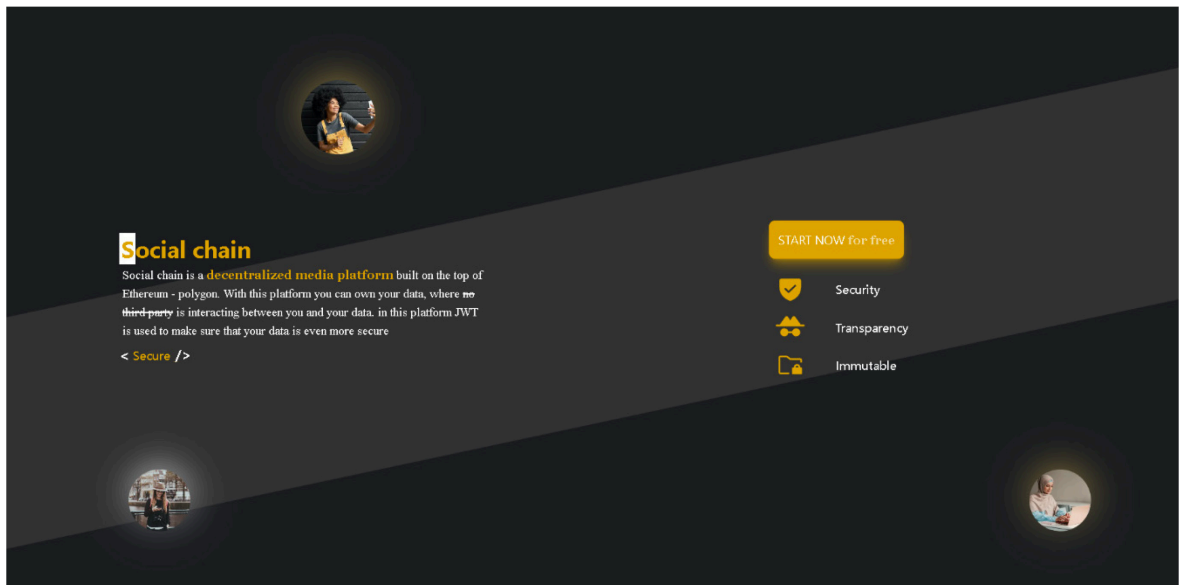


Figure 85. User enters the website.

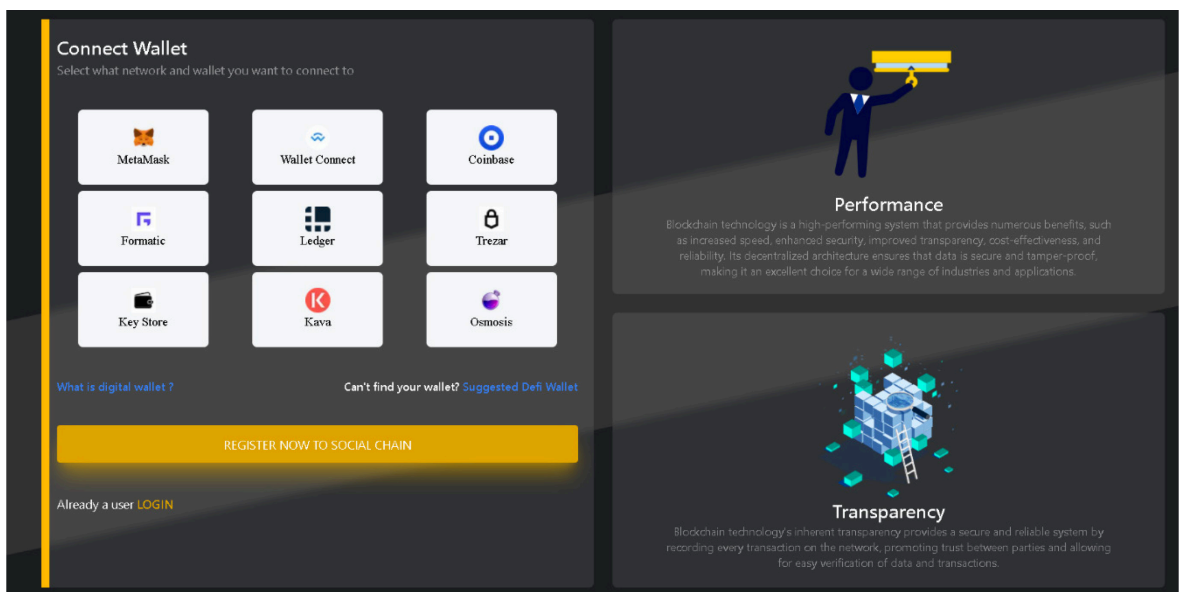


Figure 86. User Request sign page

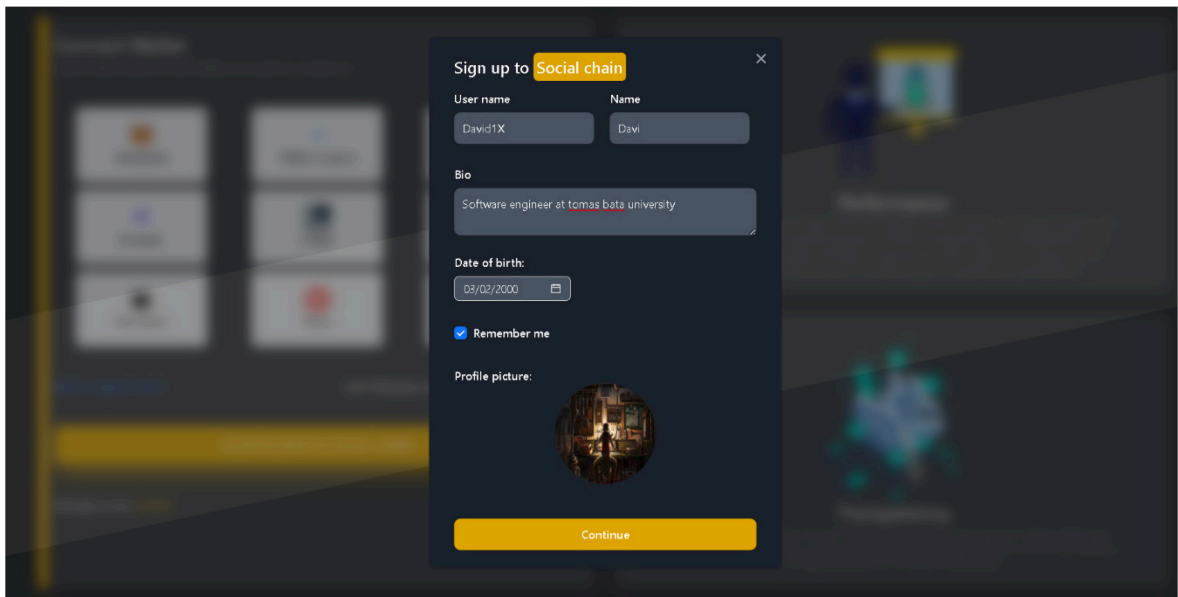


Figure 87. User registration details

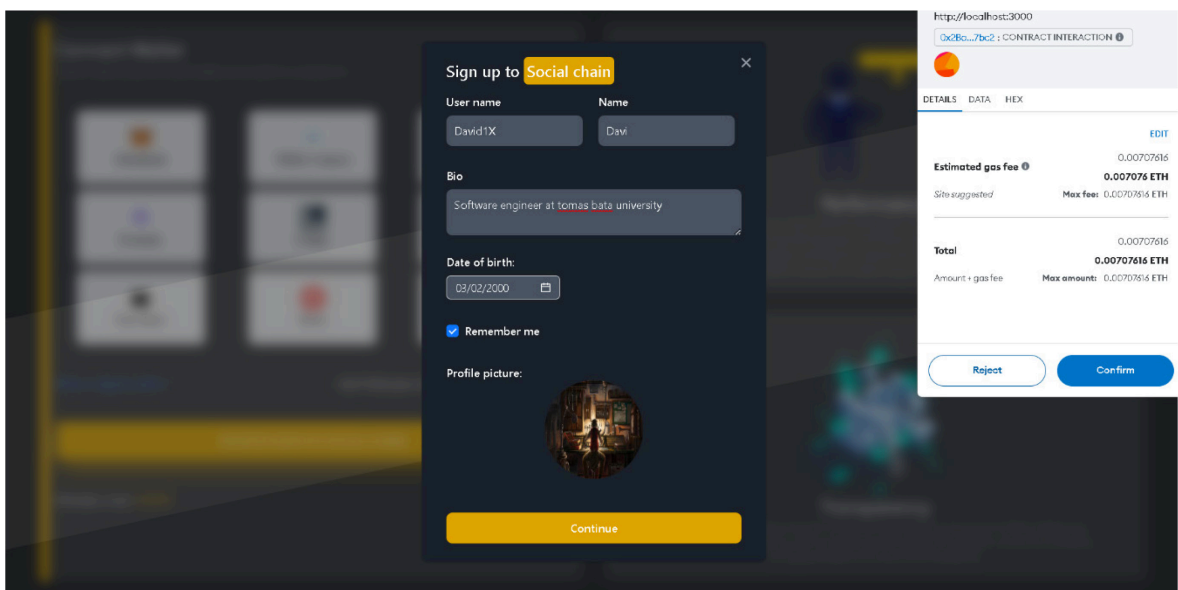


Figure 88. User sign the transaction to register

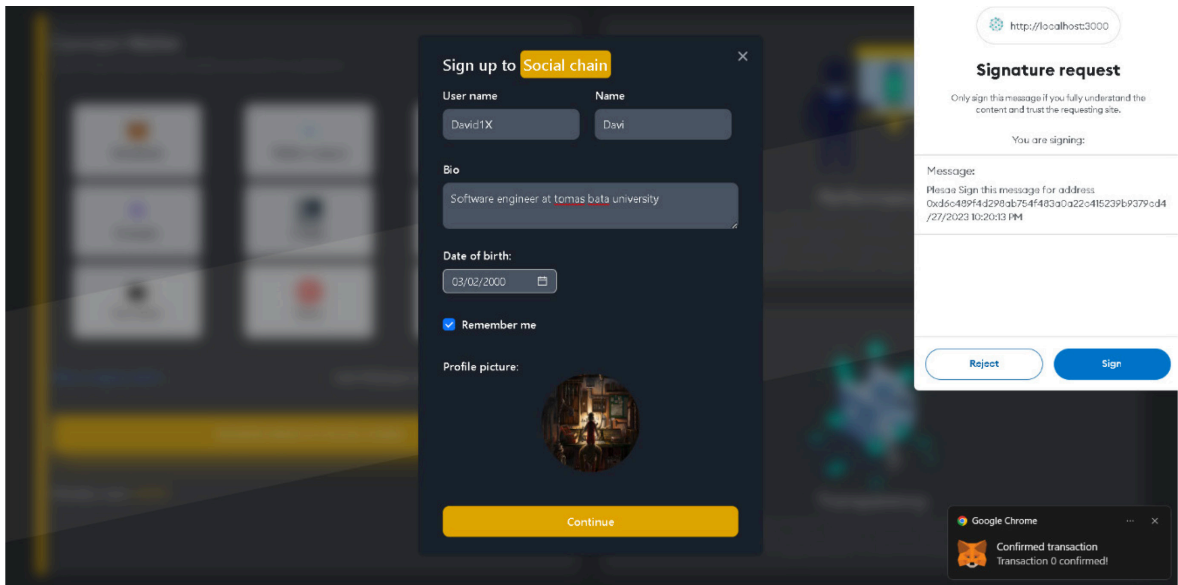


Figure 89. User sign the message to get JWT token

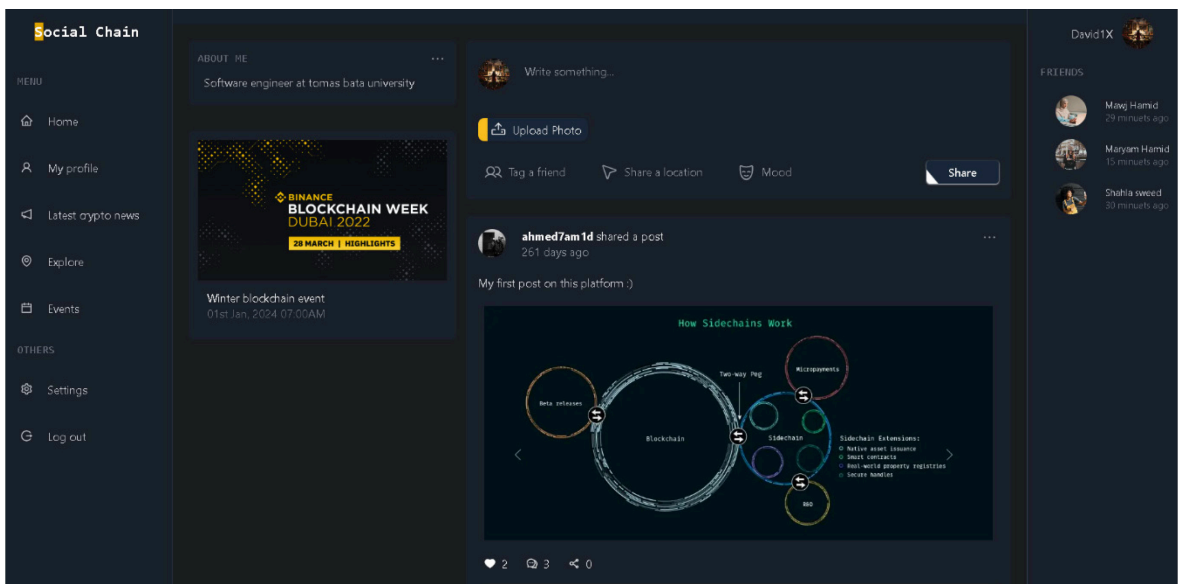


Figure 90. Successful registration

## 8.2.2 Login of existing user to the platform

The login process is almost the same algorithm as the register process it only differs in two cases:

1. Users will not be asked to provide anything other than signing a message as shown in the *Figure 93*.
2. Since the user is already registered to the platform the login process will update his access token and refresh token of the JWT, only if he is a registered user, otherwise we will notify the client-side that a user with the public address is not a valid registered user.

For the login process, one of the nice features that implemented in the login, is that user has ability to choose (remember me) when he registers. all of us have seen this feature on almost every website that provides login or in other word authentication process. Basically, what this function or process does is that user chooses that he wants to stay logged in when he refreshes the page or closes the page and visits the website again another time with no need to login. The process is done by storing the user preference in the browser's local storage that user choose this option, and whenever the page is refreshed or any page's routes are requested, in the "middleware.js" we check the local storage to find his preference about the 'remember me' if it is true we check that his JWT token is valid and if it is no expired we forward him to requested route, if his JWT is expired we try to use his refresh token, to update access token, if refresh token does not match the one in the database then he is not a valid user. For better full view of the algorithm a flow char can be used to describe the algorithm better as shown in the *Figure 91*. The flow chart is also available in the form of PDF<sup>25</sup> for higher quality and as full text<sup>26</sup> description for better understanding.

---

<sup>25</sup> User login workflow PDF file: <https://shorturl.at/buEKY>

<sup>26</sup> User login workflow text file: <https://shorturl.at/vCHN9>

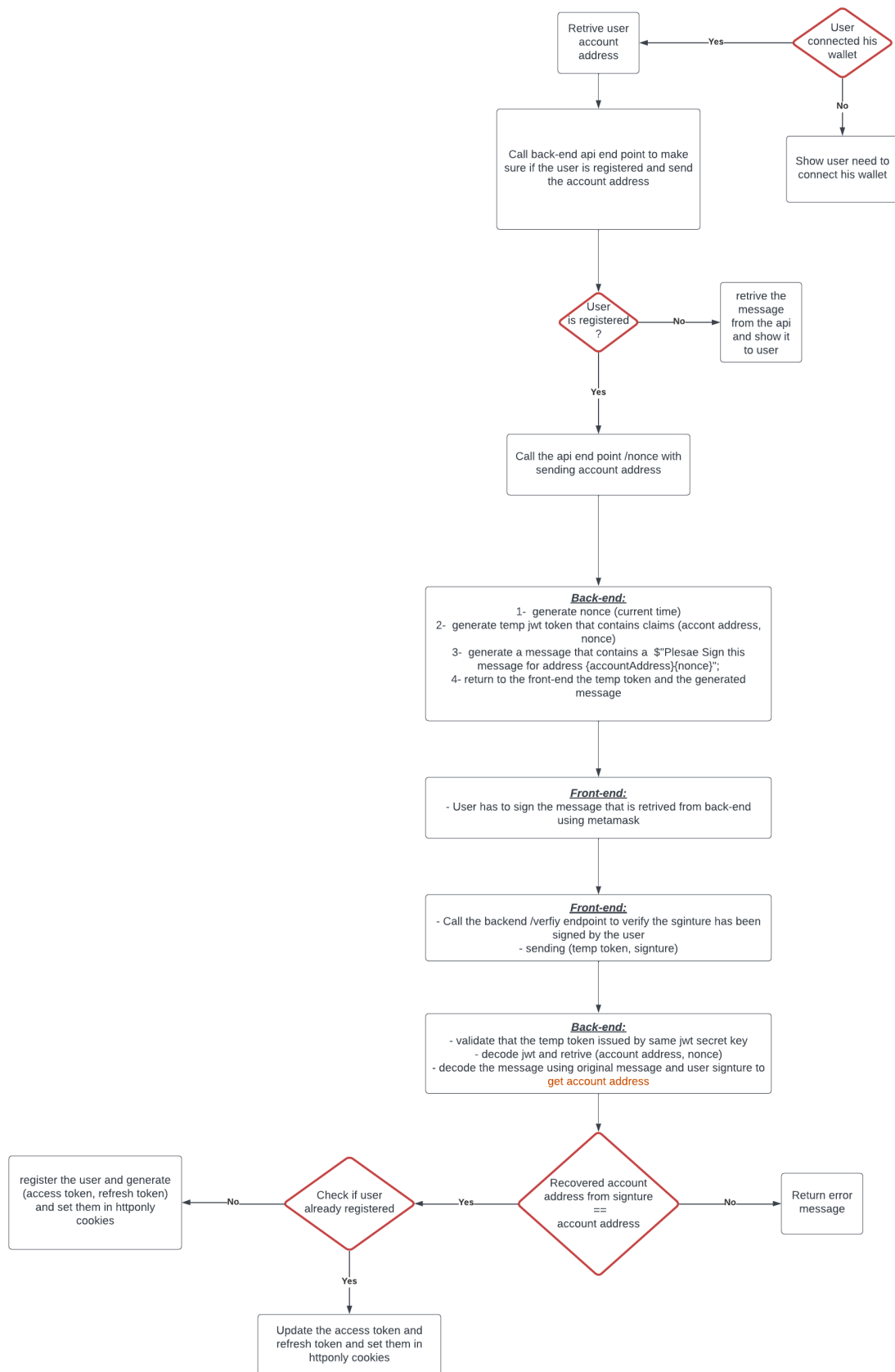


Figure 91. User login flowchart.

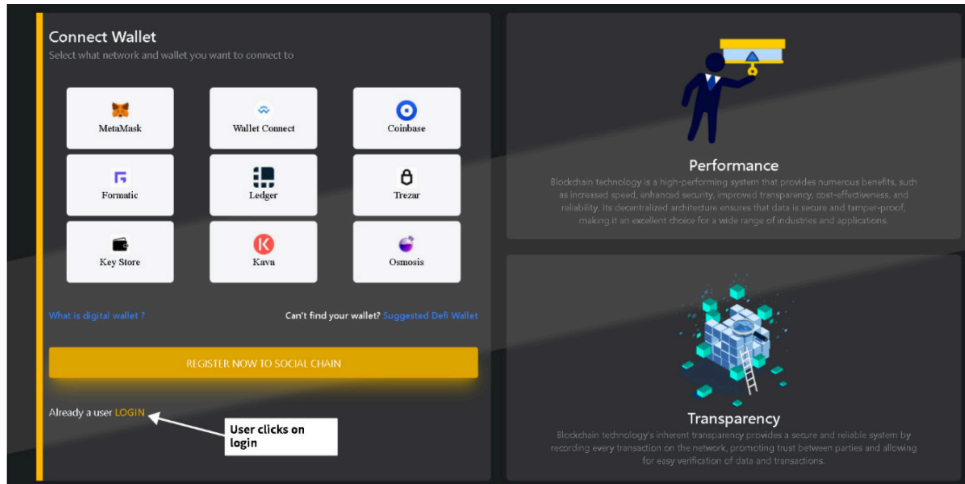


Figure 92. User login

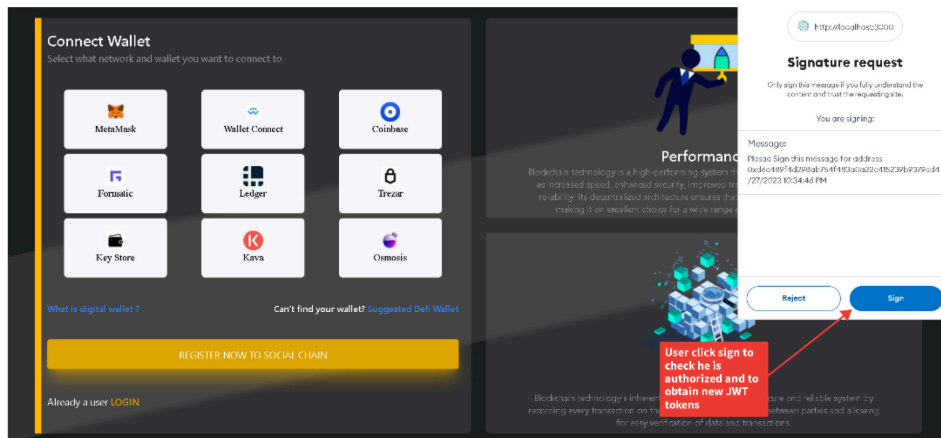


Figure 93. User sign message to obtain JWT

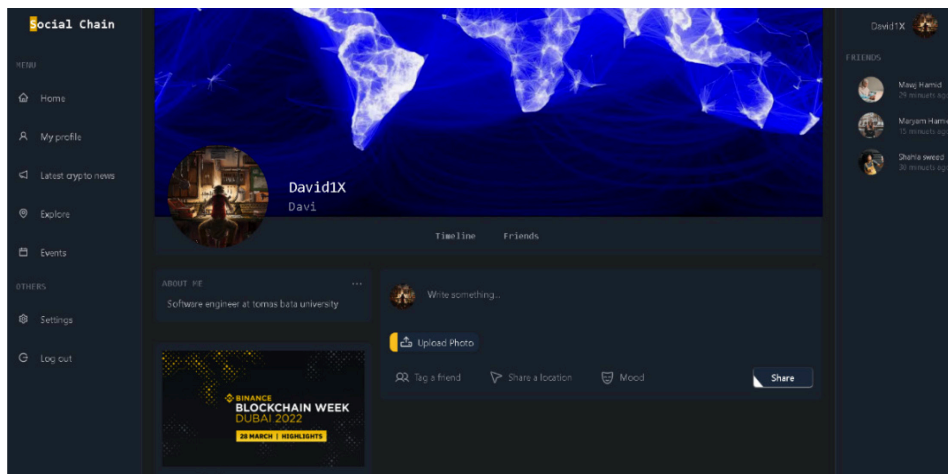


Figure 94. Forward to profile after successful login

### 8.2.3 Creation of new post

Creation of a new post on the platform requires uploading an Image to the IPFS platform (If user added image to his post). Creating a post in the platform includes these steps:

1. Validation that provided post’s data are not empty.
2. Users must sign the transaction through MetaMask and pay for gas fees as shown in the *Figure 96*.
3. Image should be uploaded to IPFS and retrieve the hash of the image back from the IPFS.
4. The new post should be added dynamically to the front-end with no need for the user to refresh the page as illustrated in the *Figure 97*.

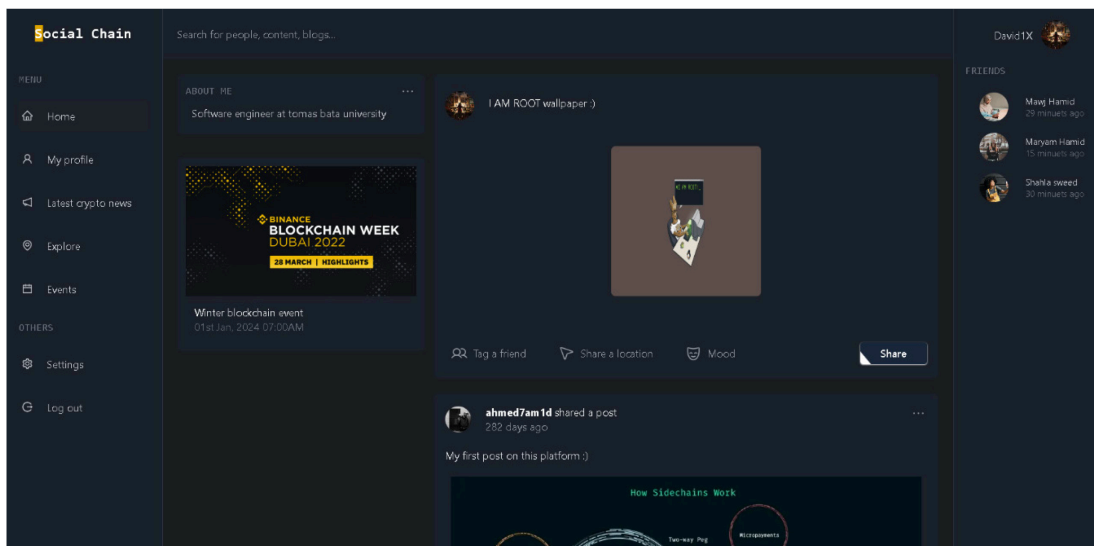


Figure 95. Filling new post data

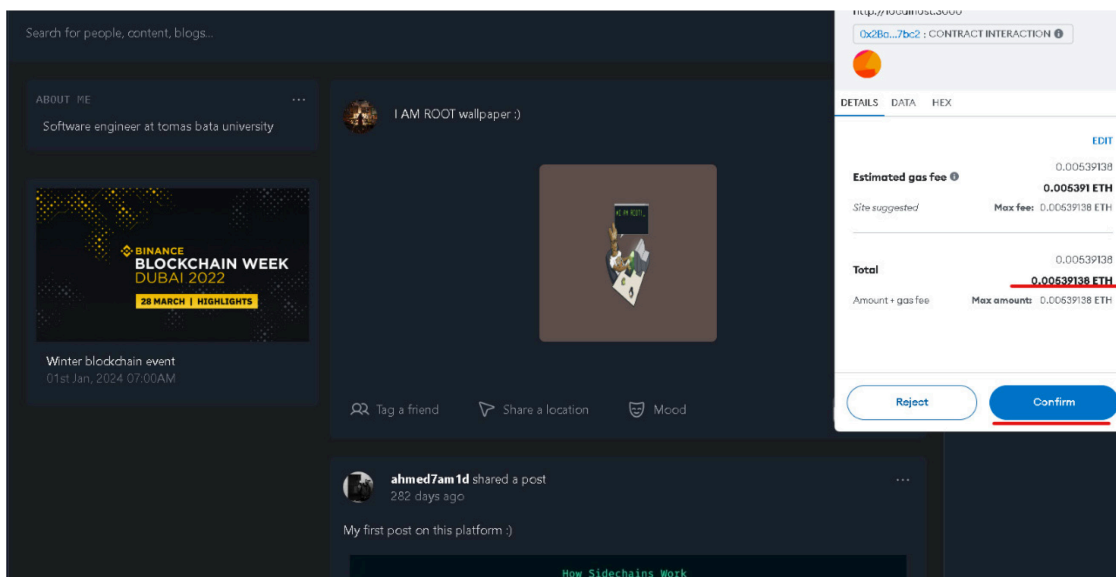


Figure 96. Submission of new post



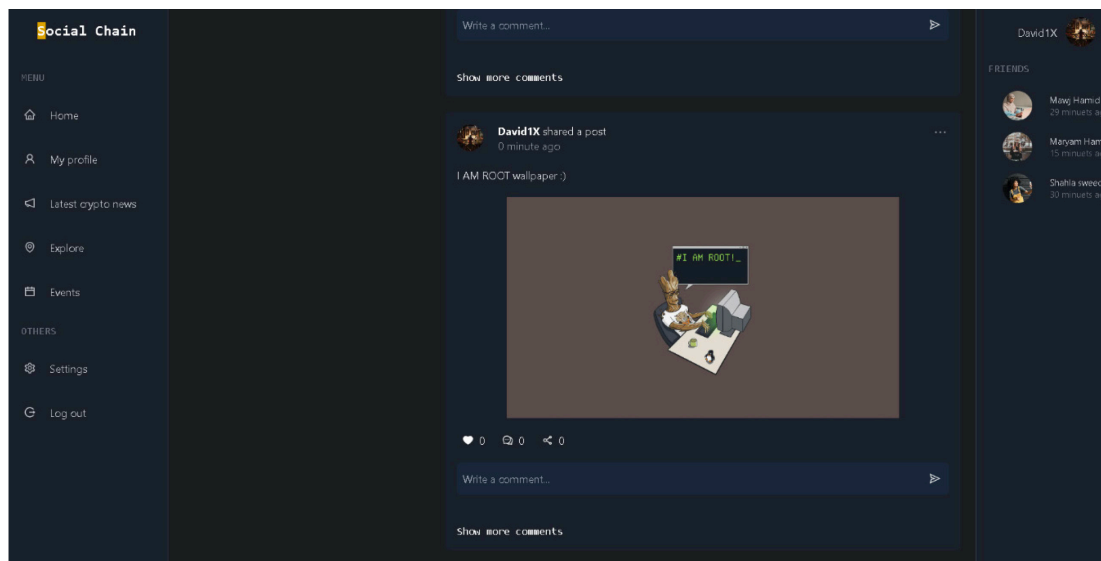


Figure 97. Post creation result

### 8.2.4 Commenting on a post

Commenting on a post in the platform does not require any back end, or any web3 storage provider, since the content of the comment is only text as illustrated in *Figure 98*. Creating a comment on a post requires only data to be sent to the contract's function. In the front-end. The creation of an empty comment is not possible because we are validating the submitted data, so if user tries to create an empty comment he will be warned. For even more data validation, the smart contract's function has also a require keyword to check whether the content of the comment sent to the contract is empty or not, if so the execution of the function will be stopped as showed in the *Figure 80*. A user needs to sign and confirm a transaction that has a specific amount of gas fee to be able to create a new comment on a post as shown in the *Figure 99*.

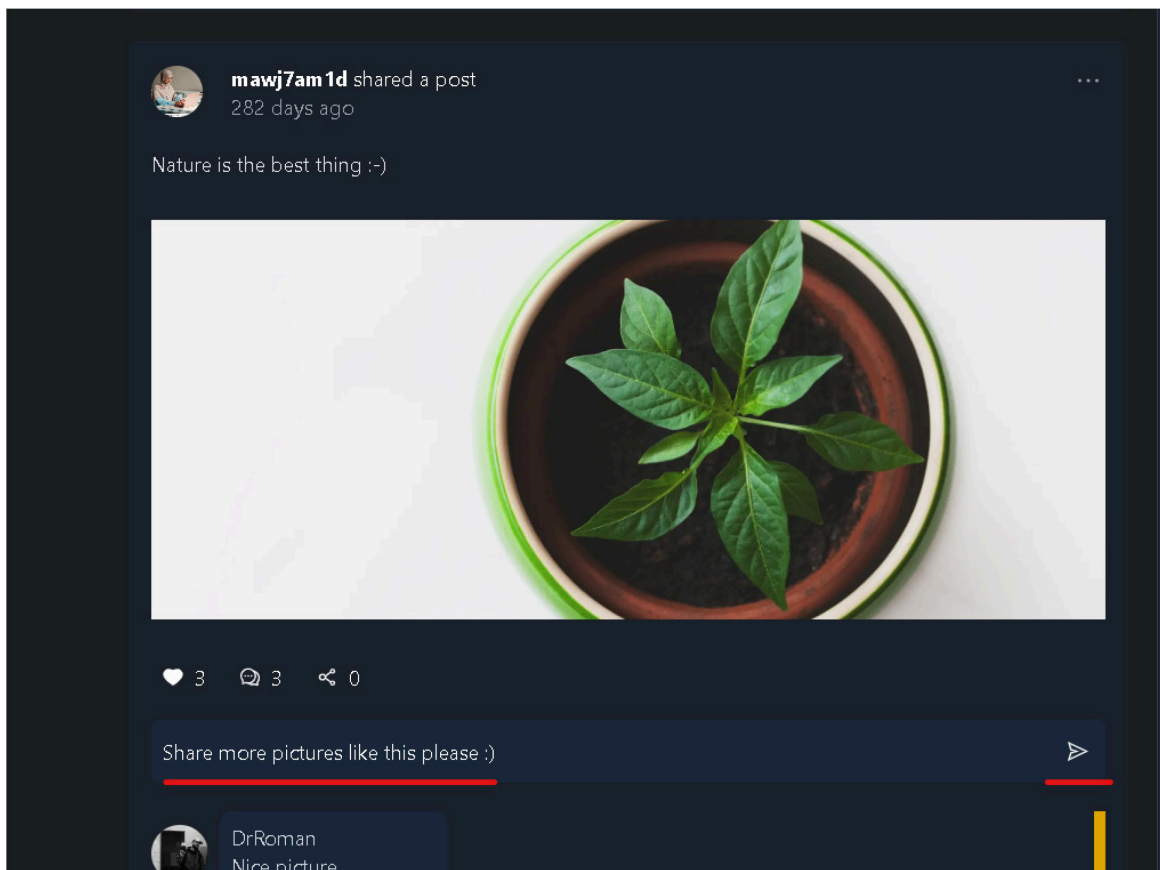


Figure 98. Filling in new comment data

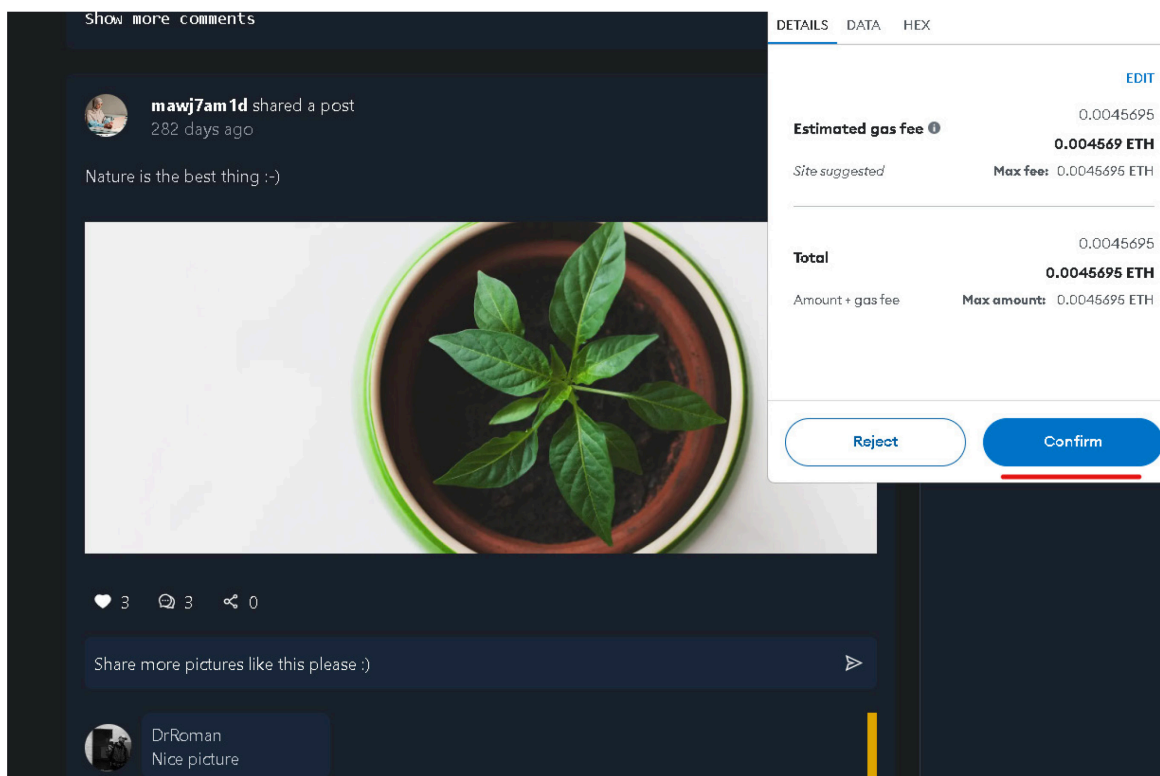
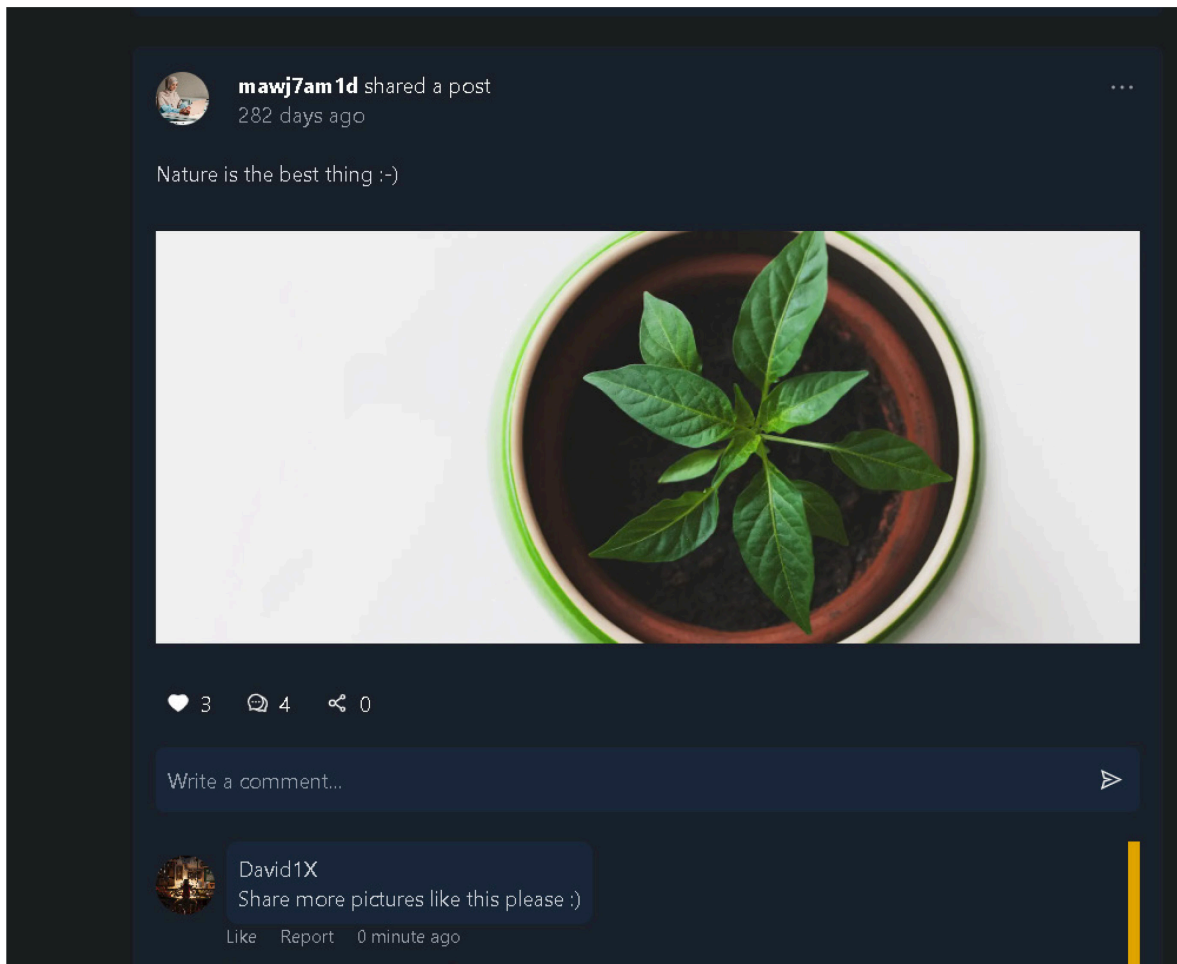


Figure 99. Submission of new comment

And of course, after successful submission of new comment, the new comment should be added dynamically to the front-end without the need to refresh the page, and as mentioned before here comes the strength of client-side rendering, client-side rendering gives better user experience and more user-friendly experience.



*Figure 100. Comment creation result*

### 8.2.5 Liking a post

Liking a post is also an operation that requires a gas fee and transaction confirmation as shown in the *Figure 101*, or blockchain in general because changing in data in the contract or blockchain requires a gas fee to be paid. Liking a post has two cases:

1. **Post is already liked:** If the post is already liked means the user want to unlike the post, for that purpose in the front-end there is a function for handling (like, unlike), and in the smart contract also there is a function called „**isLikedByAddress**“ as mentioned previously it is main purpose to check if a post is already liked by the one who calls to

like/unlike post. If the post is already liked, it is necessary to decrease number of likes to that post with the specific post id.

- Post is not liked:** The function „likePost“ in the smart contract is called to like the specific post that was clicked by user. When user clicks like post, the post id should be sent to the smart contract’s function and the function should be responsible to increase the number of likes to that specific post.

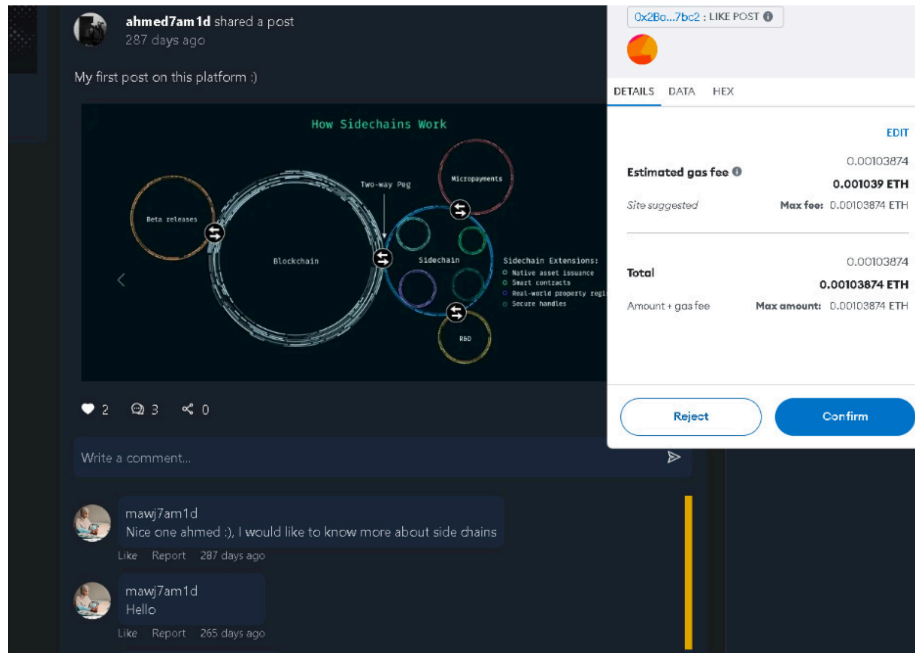


Figure 101. Like a post

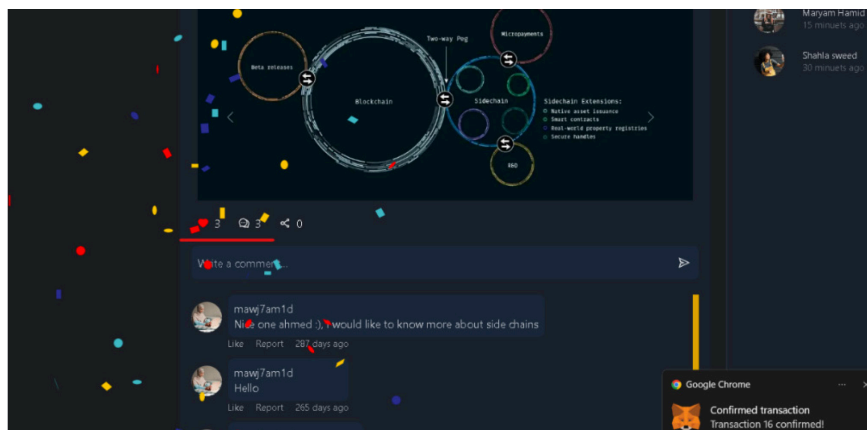
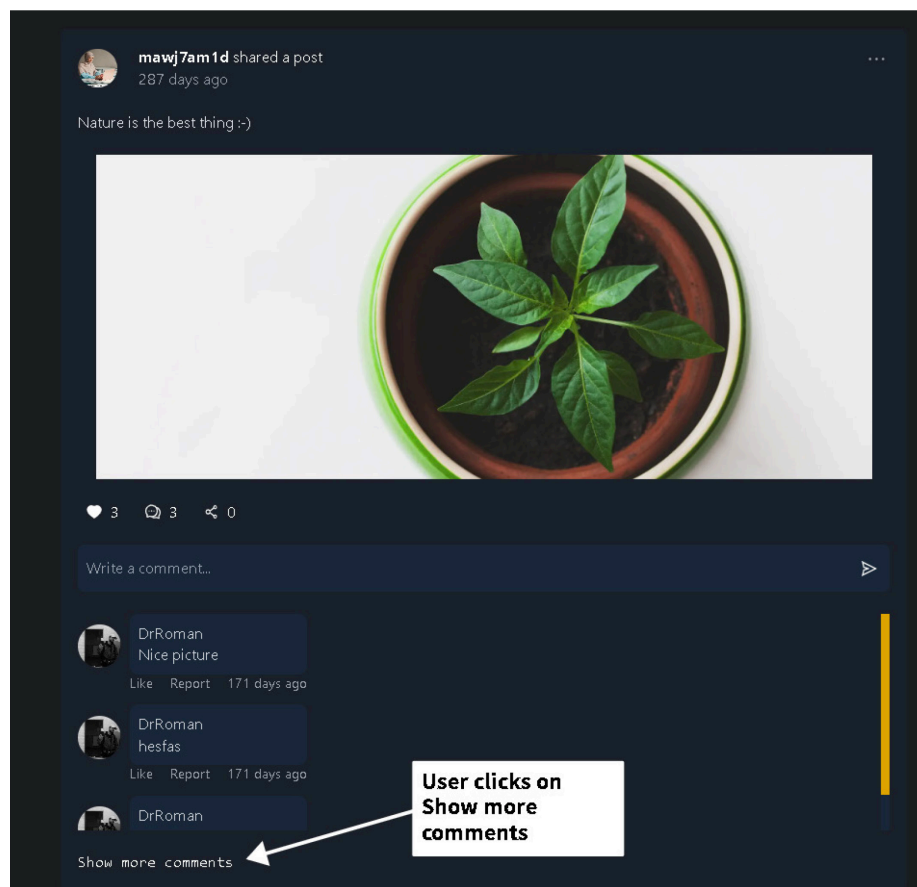


Figure 102. Like's result.

### 8.2.6 Comments pagination and post modal

As describe in 8.1.4 in the part where the concept of pagination was mentioned, the function “**getPostComments**” is used mainly here, the concept in the front-end is that each post has a link button called “show more comments” as shown in the *Figure 103*. When clicking on it, it will open a post modal that will contains all details about the requested post’s comments, and an option for user to load more options on each click three more comments are loaded by calling the function in the smart contract.



*Figure 103. Post's comments link*

In the *Figure 103* as shown there is only 3 comments that are loaded per post and when clicking on show more comments, a post modal should open like in the *Figure 104. Post modal*. And when scrolling down in the post modal by clicking on show more comments another three comments is fetched from the smart contract’s storage dynamically to the front-end as illustrated in the *Figure 105*.

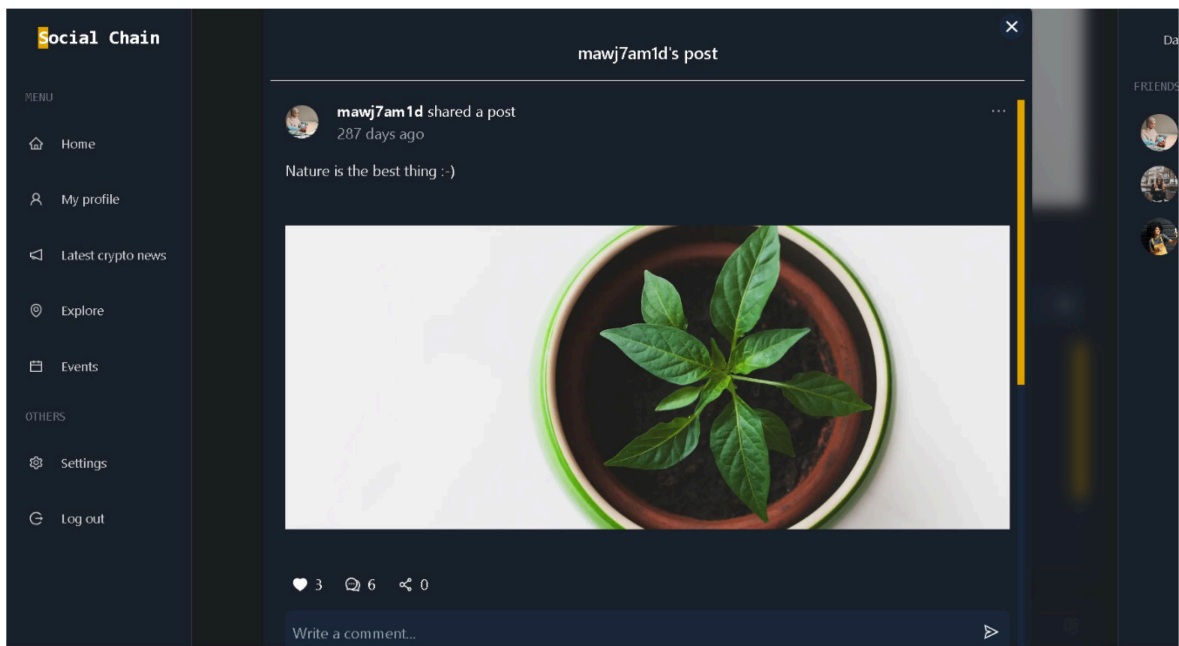


Figure 104. Post modal

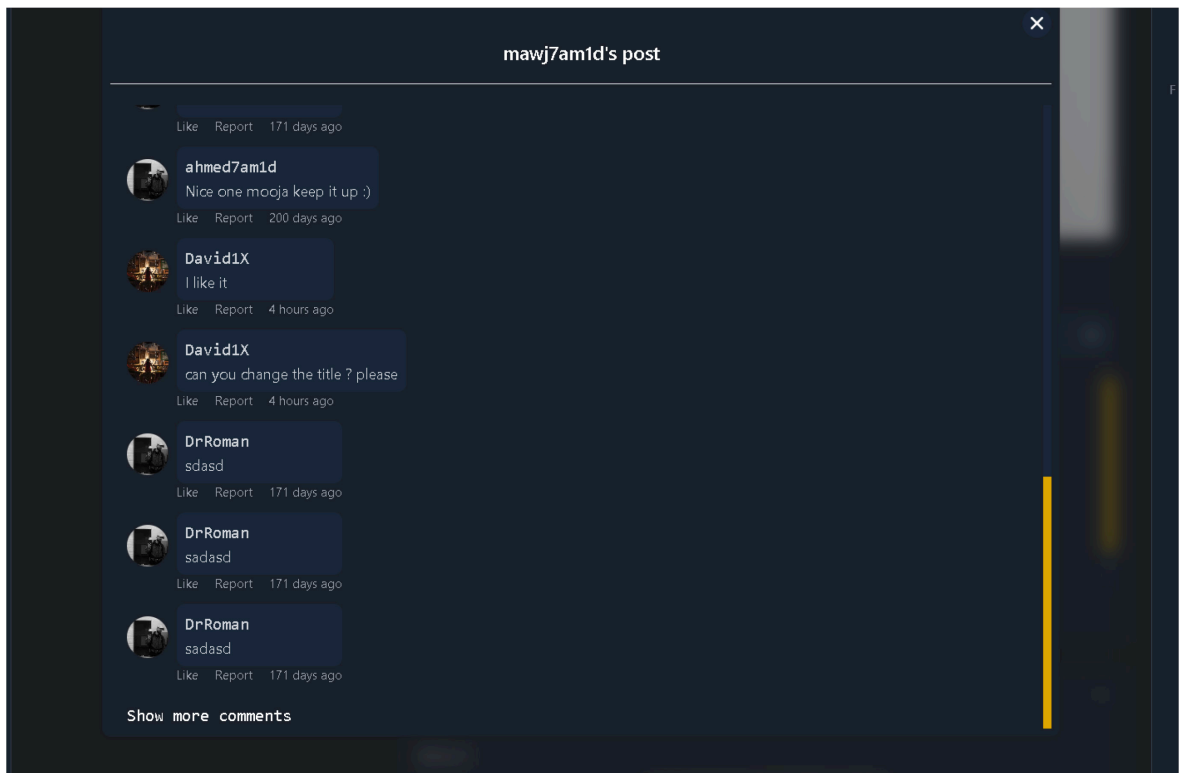


Figure 105. Comment's pagination result

## 9 TESTING OF THE SMART CONTRACT

Testing of the smart contract is a crucial part of the application. Since once a smart contract is deployed on the blockchain, it is no longer can be altered or deleted. To ensure a safe, error less smart contract and free of security threats, writing of automated tests and using of special analyzer frameworks must be used.

### 9.1 Automated testing

Automated testing is a way to run test cases easily without the need to call and run each one separately/manually. Since the applications uses hardhat as the development environment for the web3 layer, hardhat already comes with installed libraries for running automated tests, such as **chai** and **Mocha**, **chai** is a popular assertion library used with the combination of testing frameworks such as **Mocha**, for testing JavaScript codes, including smart contracts on blockchain such as Ethereum or other networks. **Mocha** will be used to run the test cases in a serial automated way. **Mocha** maps uncaught exceptions to the correct test cases. **Mocha** is running on Node.js and in the browser. Several types of testing can be done, mainly the unit testing will be done in the current implementation, since the testing is done on the functions individually. [66][67][68]

#### 9.1.1 User test suites

In this subheading point the demonstration of all test suites and test cases that are related to the User will be shown, such as (registration, reading of user data, checking of username availability). **First (“registerUser”)**, it is important to start with the registration of new user to the platform as illustrated in the *Figure 106*. Whenever a user register to the platform it is necessary to make sure that the user is registered successfully, means the user’s data is stored as expected, so for that case, the try to register new user to the contract, will be done by calling the function “*registerUser*” then by making sure that the user data are actually stored by retrieve it after registration using the function “*getUser*” that accepts a user address, we compare the retrieved results with the sent one, after that the need to make sure also that the user name is now reserved and no other user can register with it.

```
1 describe('registerUser', function () {
2     const username = 'Milad07'
3     const name = 'Milad'
4     const imgHash = 'imgHash'
5     const coverHash = 'coverHash'
6     const bio = 'Software engineering student'
7     const birthDate = 953108387
8     const showUserName = true
9     it('Should register a new user successfully', async function () {
10        // [A]- Connect to the contract using the user's address
11        const connectedContract = contract.connect(user2)
12
13        // [B]- Register the new user
14        await connectedContract.registerUser(
15            username,
16            name,
17            imgHash,
18            coverHash,
19            bio,
20            birthDate,
21            showUserName
22        )
23
24        // [C]- Verify the user registration
25        const registeredUser = await contract.getUser(user2.address)
26        const registeredUserObject = Object.assign({}, registeredUser)
27        expect(parseInt(registeredUserObject.id._hex, 16)).to.equal(2)
28        expect(registeredUserObject.userName).to.equal(username)
29        expect(registeredUserObject.name).to.equal(name)
30        expect(registeredUserObject.bio).to.equal(bio)
31        expect(parseInt(registeredUserObject.birthDate._hex, 16)).to.equal(
32            birthDate
33        )
34        expect(registeredUserObject.showUsername).to.equal(showUserName)
35        expect(registeredUserObject.imageHash).to.equal(imgHash)
36        expect(registeredUserObject.coverHash).to.equal(coverHash)
37    })
38    it('Should shows that user name is reserved after successful registration',
39    async function () {
40        // [D]- Verify the username is reserved
41        const isUsernameTaken = await contract.userNameAvailable(username)
42        expect(isUsernameTaken).to.be.true
43    })
44 })
```

Figure 106. Register user test suite

Two test cases should pass the test as expected and as shown in the Figure 107, the test cases passed successfully.



```
RUNS tests/SocialChain.test.js...
SocialChain
registerUser
  ✓ Should register a new user successfully (223 ms)
  ✓ Should shows that user name is reserved after successful registration (23 ms)
Passed: 2
Failed: 0
Time Taken: 750 ms
```

Figure 107. Register user test suite result

Second (“getUser”), this test suite has one test case as shown in the code in the *Figure 108*, and its purpose is to validate that the retrieved user information using the function “getUser” from the contract received as expected, after running the test the following result as shown in *Figure 109* is retrieved.

```
1 describe('getUser', function () {
2   it('Should return the correct user information', async function () {
3     const connectedContract = contract.connect(user2)
4     const username = 'Milad07'
5     const name = 'Milad'
6     const imgHash = 'imgHash'
7     const coverHash = 'coverHash'
8     const bio = 'Software engineering student'
9     const birthDate = 953108387
10    const showUserName = true
11
12    // [A]- Register the user
13    await connectedContract.registerUser(
14      username,
15      name,
16      imgHash,
17      coverHash,
18      bio,
19      birthDate,
20      showUserName
21    )
22
23    // [B]- Get the user information
24    const registeredUser = await contract.getUser(user2.address)
25    const registeredUserObject = Object.assign({}, registeredUser)
26    expect(parseInt(registeredUserObject.id._hex, 16)).toEqual(2)
27    expect(registeredUserObject.userName).toEqual(username)
28    expect(registeredUserObject.name).toEqual(name)
29    expect(registeredUserObject.bio).toEqual(bio)
30    expect(parseInt(registeredUserObject.birthDate._hex, 16)).toEqual(
31      birthDate
32    )
33    expect(registeredUserObject.showUserName).toEqual(showUserName)
34    expect(registeredUserObject.imageHash).toEqual(imgHash)
35    expect(registeredUserObject.coverHash).toEqual(coverHash)
36  })
37 })
```

Figure 108. Get user test suite

```
RUNS tests/SocialChain.test.js...
SocialChain
  getUser
    ✓ Should return the correct user information (253 ms)
Passed: 1
Failed: 0
Time Taken: 599 ms
```

Figure 109. Get user test suite result

**Third (“userNameAvailable”)**, the purpose of this test suite is to check that when an already registered user has a username, this username should not be used by any other user in the platform, as it makes each user unique from the other one. The test suite has two test cases, one to check for availability, that function should return true if the username is available, second test case is that the function should return false if the username is not available by trying to register a new user to the contract and then check availability of the username after the registration as illustrated in the *Figure 110* and in the *Figure 111*.

```
1 describe('userNameAvailable', function () {
2   it('Should return true if username is available', async function () {
3     const username = 'Ahmed07'
4     const status = await contract.userNameAvailable(username)
5     expect(status).toBe(true)
6   })
7
8   it('Should return false if the username is taken', async function () {
9     // [A]- Connect to the contract using the user's address (so it is
10    different from deployer)
11    const connectedContract = contract.connect(user1)
12
13    // [B]- Register new user to the contract
14    const username = 'Ahmed07'
15    await connectedContract.registerUser(
16      username,
17      'Ahmed',
18      'DASF3420942F',
19      'DDF211XC',
20      'Software engineering student',
21      953108387,
22      true
23    )
24    // [C]- Check the registered user name is not available anymore
25    const status = await contract.userNameAvailable(username)
26    expect(status).toBe(false)
27  })
28 })
```

Figure 110. Username availability test suite

```
RUNS tests/SocialChain.test.js...
SocialChain
  userNameAvailable
    ✓ Should return true if username is available (71 ms)
    ✓ Should return false if the username is taken (322 ms)
Passed: 2
Failed: 0
Time Taken: 1190 ms
```

Figure 111. Username availability test suite result

### 9.1.2 Post test suites

Since social networking is all about sharing activities and blogs, it is important to make sure that creating a post, retrieving a post, showing post to other users on the feed all working as expected. **First (“createPost”)** test suite, the purpose of this test suite as name indicates is to make sure that the post created successfully, and the contract storage is populated with no problems. The test suite will try to create a new post as shown in *Figure 112*. And then by calling the mapping and sending the post id, the same post should be retrieved, that was sent for creation, in other words the test suite should pass the test as show in the *Figure 113*.

```
1 describe('createPost', function () {
2   it("Should create a new post and append it to the user's array of posts", async
3   function () {
4     // [A]- Creation of post info
5     const postDescription = 'This is a new post'
6     const imgHash = 'postImgHash'
7
8     // [B]- Call create post function
9     await contract.createPost(postDescription, imgHash)
10
11    // [C]- Get the post information
12    const postId = 1
13    const post = await contract.posts(postId)
14    const postObject = Object.assign({}, post)
15
16    expect(parseInt(postObject.postId._hex, 16)).to.equal(postId)
17    expect(post.author).to.equal(owner.address)
18    expect(post.postDescription).to.equal(postDescription)
19    expect(post.imgHash).to.equal(imgHash)
20    expect(parseInt(post.timeStamp._hex, 16)).to.not.equal(0)
21    expect(parseInt(post.likeCount._hex, 16)).to.equal(0)
22    expect(parseInt(post.reportCount._hex, 16)).to.equal(0)
23    expect(post.status, 16).to.equal(1)
24  })
25 }
```

Figure 112. Create post test suite

```
RUNS tests/SocialChain.test.js...
SocialChain
createPost
  ✓ Should create a new post and append it to the user's array of posts (324 ms)
Passed: 1
Failed: 0
Time Taken: 675 ms
```

Figure 113. Create post test suite result

**Second (“getPostById”)** test suite. Tries to validate the correct retrieved post’s data, by creating new post and then retrieve it and compare both results as illustrated in the *Figure 114*. It also has another test case that tells when trying to retrieve an unactive post or none exists post, should throws an revert error from the contract with the specified error message that was provided in the contract.

```
1 describe('getPostById', function () {
2   it('Should return the correct post by ID', async function () {
3     const postDescription = 'My first post'
4     const imgHash = 'imgHash'
5
6     // [A]- Create a new post
7     await contract.createPost(postDescription, imgHash)
8
9     // [B]- Retrieve the post using getPostById
10    const retrievedPost = await contract.getPostById(
11      await contract.totalPosts()
12    )
13
14    // [C]- Verify the retrieved post data
15    expect(parseInt(retrievedPost.postId._hex, 16)).to.equal(
16      await contract.totalPosts()
17    )
18    expect(retrievedPost.postDescription).to.equal(postDescription)
19    expect(retrievedPost.imgHash).to.equal(imgHash)
20  })
21
22  it('Should throw an error if the post is not active', async function () {
23    const nonExistetPostId = 12345
24
25    // [A]- Call getPostById with a non exists post ID
26    await expect(
27      contract.getPostById(nonExistetPostId)
28    ).to.be.revertedWith('Not an active post')
29  })
30 })
```

Figure 114. Get post by id test suite

```

RUMS tests/SocialChain.test.js...
SocialChain
  getPostById
    ✓ Should return the correct post by ID (479 ms)
    ✓ Should throw an error if the post is not active (30 ms)
Passed: 2
Failed: 0
Time Taken: 1129 ms

```

Figure 115. Get post by id test suite result

**Third (“getUserPosts”)**, the test suite has two test cases that are used to verify that the posts that are created by a user are the same as the retrieved one. And has the same values and that they belong to the user by checking the author of the post to the caller of the function as shown in the *Figure 116* below with the test result in the *Figure 117*.

```

1  describe('getUserPosts', function () {
2    it('Should returns the posts created by the user', async function () {
3      const post1Description = 'My first post - Hello World'
4      const post1ImgHash = 'imgHash1'
5      const post2Description = 'My new wallpaper'
6      const post2ImgHash = 'imgHash2'
7      // [A]- Create two posts by the user
8      await contract.createPost(post1Description, post1ImgHash)
9      await contract.createPost(post2Description, post2ImgHash)
10     // [B]- Retrieve the user posts by calling getUserPosts
11     const retrievedPosts = await contract.getUserPosts()
12
13     // [C]- Verify the retrieved posts
14     expect(retrievedPosts.length).toEqual(2)
15
16     // [D]- Validate the properties of the first post
17     expect(parseInt(retrievedPosts[0].postId._hex, 16)).toEqual(1)
18     expect(retrievedPosts[0].postDescription).toEqual(post1Description)
19     expect(retrievedPosts[0].imgHash).toEqual(post1ImgHash)
20
21     // [E]- Validate the properties of the second post
22     expect(parseInt(retrievedPosts[1].postId._hex, 16)).toEqual(2)
23     expect(retrievedPosts[1].postDescription).toEqual(post2Description)
24     expect(retrievedPosts[1].imgHash).toEqual(post2ImgHash)
25   })
26
27   it('Should return that the user created the post is the author of the posts', async function () {
28     const post1Description = 'My first post - Hello World'
29     const post1ImgHash = 'imgHash1'
30     const post2Description = 'My new wallpaper'
31     const post2ImgHash = 'imgHash2'
32     // [A]- Create two posts by the user
33     await contract.createPost(post1Description, post1ImgHash)
34     await contract.createPost(post2Description, post2ImgHash)
35     // [B]- Retrieve the user posts by calling getUserPosts
36     const retrievedPosts = await contract.getUserPosts()
37     // [C]- Verify the authorship of the posts
38     expect(retrievedPosts[0].author).toEqual(owner.address)
39     expect(retrievedPosts[1].author).toEqual(owner.address)
40   })
41 })

```

Figure 116. Get user's posts test suite

```
RUNS tests/storage.test.js....
SocialChain
getUserPosts
  ✓ Should returns the posts created by the user (419 ms)
  ✓ Should return that the user created the post is the author of the posts (436 ms)
Passed: 2
Failed: 0
Time Taken: 1848 ms
```

Figure 117. Get user's post test suite result

**Fourth (“getPostIds”)**, this test suite needs to be handled carefully since the idea of pagination is implemented here, and it must be tested well, the function “getPostIds” can affect the user experience in the client-side if it is not handled and tested well. The test suit should pass the test as shown in *Figure 119*. Where an attempt to create three new post and retrieve them using the function “getPostIds” is done. The comparison is mainly done by comparing the length of items retrieved from the post “getPostIds” should be equal to the request list of items, but also by Id as shown in the *Figure 118*.

```
1 describe('getPostIds', function () {
2   it('Should return the correct post IDs based on pagination', async function () {
3     const postOneDescription = 'My first post'
4     const postOneImgHash = 'imgHashOne'
5     const postTwoDescription = 'My second post'
6     const postTwoImgHash = 'imgHashTwo'
7     const postThreeDescription = 'My third post'
8     const postThreeImgHash = 'imgHashThree'
9
10    //[[A]- Create three posts
11    await contract.createPost(postOneDescription, postOneImgHash)
12    await contract.createPost(postTwoDescription, postTwoImgHash)
13    await contract.createPost(postThreeDescription, postThreeImgHash)
14
15    //[[B]- Retrieve the post IDs using getPostIds with pagination
16    const page = 1
17    const perPage = 3
18    const retrievedPostIds = await contract.getPostIds(page, perPage)
19
20    //[[C]- Verify the retrieved post IDs
21    expect(retrievedPostIds.length).to.equal(perPage)
22    expect(parseInt(retrievedPostIds[0]._hex,16)).to.equal(1) // ID of the first post
23    expect(parseInt(retrievedPostIds[1]._hex,16)).to.equal(2) // ID of the second post
24    expect(parseInt(retrievedPostIds[2]._hex,16)).to.equal(3) // ID of the third post
25  })
26 })
```

Figure 118. Get posts' ids test suite

```
RUNS tests/storage.test.js...
SocialChain
getPostIds
  ✓ Should return the correct post IDs based on pagination (533 ms)
Passed: 1
Failed: 0
Time Taken: 1075 ms
```

*Figure 119. Get posts' ids test suite result*

**Fifth (“likePost”)**, the test suite has three test cases as illustrated in the *Figure 120*, one checks that a post is liked successfully by checking the total number of likes on the post and by checking the user is become a liker of the post. Second test case making sure if post already is liked by the user, if so, it should throw an error with the message “*Post already liked by the user*”. Third test case is about an unactive post cannot be liked, and an error message with the string “*Not an active post*” must be thrown.

```

1 describe('likePost', function () {
2   it('Should like a post successfully', async function () {
3     // [A]- Creating of new post
4     const postDescription = 'My new post'
5     const imgHash = 'imgHash'
6     await contract.createPost(postDescription, imgHash)
7     // [B]- Liking of a post
8     await contract.likePost(await contract.totalPosts())
9     // [C]- Retrieve the post and check the Like count
10    const retrievedPost = await contract.getPostById(
11      await contract.totalPosts()
12    )
13    expect(parseInt(retrievedPost.likeCount._hex, 16)).to.equal(1)
14
15    // [D]- Making sure number the user actually is a liker of the post
16    const isPostLikedByUser = await contract.postLikers(
17      await contract.totalPosts(),
18      owner.address
19    )
20    expect(isPostLikedByUser).to.be.true
21  })
22
23  it('Should revert if the post is not active', async function () {
24    const nonExistentPostId = 12345
25
26    // [A]- Call LikePost with a non-existent post ID
27    await expect(
28      contract.likePost(nonExistentPostId)
29    ).to.be.revertedWith('Not an active post')
30  })
31
32  it('Should revert if the post is already liked by the user', async function () {
33    const postDescription = 'My first post'
34    const imgHash = 'imgHash'
35
36    // [A]- Create a new post
37    await contract.createPost(postDescription, imgHash)
38
39    // [B]- Like the post
40    const postId = await contract.totalPosts()
41    await contract.likePost(postId)
42
43    // [C]- Attempt to Like the post again
44    await expect(contract.likePost(postId)).to.be.revertedWith(
45      'Post already liked by the user'
46    )
47  })
48 })

```

Figure 120. Like post test suite

```

RUNS tests/storage.test.js...
SocialChain
likePost
  ✓ Should like a post successfully (432 ms)
  ✓ Should revert if the post is not active (28 ms)
  ✓ Should revert if the post is already liked by the user (259 ms)
Passed: 3
Failed: 0
Time Taken: 1723 ms

```

Figure 121. Like post test suite result

Sixth (“unLikePost”), this test suite is the inverse of the “likePost” test suite, covering the functionality of unliking a post instead of liking it, as shown in the Figure 122.



```

1 describe.only('unLikePost', function () {
2   it('Should unlike a post successfully', async function () {
3     // [A]- Creating a new post
4     const postDescription = 'My new post'
5     const imgHash = 'imgHash'
6     await contract.createPost(postDescription, imgHash)
7
8     // [B]- Liking the post
9     const postId = await contract.totalPosts()
10    await contract.likePost(postId)
11
12    // [C]- Unliking the post
13    await contract.unLikePost(postId)
14
15    // [D]- Retrieve the post and check the like count
16    const retrievedPost = await contract.getPostById(postId)
17    expect(parseInt(retrievedPost.likeCount._hex, 16)).to.equal(0)
18
19    // [E]- Making sure the user is no longer a liker of the post
20    const isPostLikedByUser = await contract.postLikers(
21      postId,
22      owner.address
23    )
24    expect(isPostLikedByUser).to.be.false
25  })
26
27  it('Should revert if the post is not active', async function () {
28    const nonExistentPostId = 12345
29
30    // [A]- Call unlikePost with a non-existent post ID
31    await expect(
32      contract.unLikePost(nonExistentPostId)
33    ).to.be.revertedWith('Not an active post')
34  })
35
36  it('Should revert if the post is not liked by the user', async function () {
37    const postDescription = 'My first post'
38    const imgHash = 'imgHash'
39
40    // [A]- Create a new post
41    await contract.createPost(postDescription, imgHash)
42
43    // [B]- Attempt to unlike the post without liking it first
44    const postId = await contract.totalPosts()
45    await expect(contract.unLikePost(postId)).to.be.revertedWith(
46      'Post not liked by the user'
47    )
48  })
49 })

```

Figure 122. Un like post test suite

```

RUNS tests/storage.test.js...
SocialChain
unLikePost
  ✓ Should unlike a post successfully (470 ms)
  ✓ Should revert if the post is not active (34 ms)
  ✓ Should revert if the post is not liked by the user (206 ms)
Passed: 3
Failed: 0
Time Taken: 1730 ms

```

Figure 123. Un like post test suite result

**Seventh (“isLikedByAddress”)**, the test suite has two test cases, checking that in both cases if post is not liked should return false, otherwise true, as shown in the *Figure 124*.

```
1 describe('isLikedByAddress', function () {
2   it('Should return true if the post is liked by the user', async function () {
3     // [A]- Creating a new post
4     const postDescription = 'My new post'
5     const imgHash = 'imgHash'
6     await contract.createPost(postDescription, imgHash)
7
8     // [B]- Liking the post
9     const postId = await contract.totalPosts()
10    await contract.likePost(postId)
11
12    // [C]- Checking if the post is liked by the user
13    const isLiked = await contract.isLikedByAddress(
14      postId,
15      owner.address
16    )
17    expect(isLiked).to.be.true
18  })
19
20  it('Should return false if the post is not liked by the user', async function () {
21    // [A]- Creating a new post
22    const postDescription = 'My new post'
23    const imgHash = 'imgHash'
24    await contract.createPost(postDescription, imgHash)
25
26    // [B]- Checking if the post is liked by the user (who has not liked it)
27    const postId = await contract.totalPosts()
28    const isLiked = await contract.isLikedByAddress(
29      postId,
30      owner.address
31    )
32    expect(isLiked).to.be.false
33  })
34 })
```

Figure 124. Is liked by address test suite

```
RUNS tests/storage.test.js...
SocialChain
isLikedByAddress
  ✓ Should return true if the post is liked by the user (371 ms)
  ✓ Should return false if the post is not liked by the user (255 ms)
Passed: 2
Failed: 0
Time Taken: 1290 ms
```

Figure 125. Is liked by address test suite result

### 9.1.3 Comment test suites

Comments represent the core of the post, comments add more interactivity and user experience to the platform, and they should be handled carefully.

First (“createComment and getCommentById”), the test suite has the same concept and flow as “createPost” and “getPostById” test suites. The test suite has two main test cases one for the creation and retrieving of a post and one for checking that a post cannot be retrieved if it is not an active post as shown in the *Figure 126*.

```
1 describe('createComment and getCommentById', function () {
2   it('Should create a comment and retrieve it by ID', async function () {
3     // [A]- Creating a new post
4     const postDescription = 'My new post'
5     const imgHash = 'imgHash'
6     await contract.createPost(postDescription, imgHash)
7
8     // [B]- Creating a comment for the post
9     const postId = await contract.totalPosts()
10    const comment = 'This is my comment'
11    await contract.createComment(postId, comment)
12
13    // [C]- Retrieving the comment by ID
14    const commentId = await contract.totalComments()
15    const retrievedComment = await contract.getCommentById(commentId)
16
17    // [D]- Verifying the retrieved comment
18    expect(retrievedComment.content).toEqual(comment)
19    expect(retrievedComment.postId).toEqual(postId)
20  })
21
22  it('Should handle retrieving a non-existent comment', async function () {
23    // [A]- Creating a new post
24    const postDescription = 'My new post'
25    const imgHash = 'imgHash'
26    await contract.createPost(postDescription, imgHash)
27
28    // [B]- Attempting to retrieve a non-existent comment
29    const nonExistentCommentId = 12345
30    await expect(
31      contract.getCommentById(nonExistentCommentId)
32    ).toBe.revertedWith('Not an active comment')
33  })
34 })
```

Figure 126. Create comment and get comment by id test suite

```
RUNS tests/storage.test.js...
SocialChain
createComment and getCommentById
  ✓ Should create a comment and retrieve it by ID (568 ms)
  ✓ Should handle retrieving a non-existent comment (269 ms)
Passed: 2
Failed: 0
Time Taken: 1549 ms
```

Figure 127. Create comment and get comment by id test suite result

Second (“getPostComments”), focuses on validating the functionality of the pagination algorithm. Its purpose is to ensure the proper retrieval of post comments based on pagination criteria. Additionally, the test suite verifies the expected behavior of displaying an error message when attempting to retrieve comments from a non-existent post. As shown in the *Figure 128*, containing two distinct test cases.

```
1 describe.only('getPostComments', function () {
2   it('Should retrieve comments for a post based on pagination', async function () {
3     // [A]- Creating a new post
4     const postDescription = 'My new post'
5     const imgHash = 'imgHash'
6     await contract.createPost(postDescription, imgHash)
7
8     // [B]- Creating comments for the post
9     const postId = await contract.totalPosts()
10    const comment1 = 'This is comment 1'
11    const comment2 = 'This is comment 2'
12    await contract.createComment(postId, comment1)
13    await contract.createComment(postId, comment2)
14
15    // [C]- Retrieving comments for the post
16    const listNumber = 1
17    const commentPerList = 2
18    const retrievedComments = await contract.getPostComments(
19      listNumber,
20      commentPerList,
21      postId
22    )
23
24    // [D]- Verifying the retrieved comments
25    expect(retrievedComments.length).toEqual(2)
26    expect(retrievedComments[0].content).toEqual(comment1)
27    expect(retrievedComments[1].content).toEqual(comment2)
28  })
29
30  it('Should handle retrieving comments for a non-existent post', async function () {
31    // [A]- Creating a new post
32    const postDescription = 'My new post'
33    const imgHash = 'imgHash'
34    await contract.createPost(postDescription, imgHash)
35
36    // [B]- Trying to retrieve comments for a non-existent post
37    const nonExistentPostId = 12345
38    const listNumber = 1
39    const commentPerList = 2
40    await expect(
41      contract.getPostComments(
42        listNumber,
43        commentPerList,
44        nonExistentPostId
45      )
46    ).toBe.revertedWith('Not an active post')
47  })
48 })
```

Figure 128. Get post's comments test suite

```
RUNS tests/storage.test.js...
SocialChain
getPostComments
  ✓ Should retrieve comments for a post based on pagination (698 ms)
  ✓ Should handle retrieving comments for a non-existent post (293 ms)
Passed: 2
Failed: 0
Time Taken: 1682 ms
```

Figure 129. Get post's comments test suite result

#### 9.1.4 Automated testing total testing result

The primary objective of automated testing is to execute a comprehensive run of all test suites and assess the results. The *Figure 130*, demonstrates the execution of all tests, which have resulted in success.

```
✓ Should show that user name is reserved after successful registration (33 ms)
getUser
  ✓ Should return the correct user information (1195 ms)
createPost
  ✓ Should create a new post and append it to the user's array of posts (484 ms)
getPostById
  ✓ Should return the correct post by ID (728 ms)
  ✓ Should throw an error if the post is not active (34 ms)
getUserPosts
  ✓ Should returns the posts created by the user (723 ms)
  ✓ Should return that the user created the post is the author of the posts (689 ms)
getPostIds
  ✓ Should return the correct post IDs based on pagination (849 ms)
likePost
  ✓ Should like a post successfully (850 ms)
  ✓ Should revert if the post is not active (36 ms)
  ✓ Should revert if the post is already liked by the user (658 ms)
unlikePost
  ✓ Should unlike a post successfully (501 ms)
  ✓ Should revert if the post is not active (35 ms)
  ✓ Should revert if the post is not liked by the user (580 ms)
isLikedByAddress
  ✓ Should return true if the post is liked by the user (996 ms)
  ✓ Should return false if the post is not liked by the user (580 ms)
createComment and getCommentById
  ✓ Should create a comment and retrieve it by ID (1744 ms)
  ✓ Should handle retrieving a non-existent comment (979 ms)
getPostComments
  ✓ Should retrieve comments for a post based on pagination (1010 ms)
  ✓ Should handle retrieving comments for a non-existent post (618 ms)
Passed: 23
Failed: 0
```

Figure 130. Total test suites results.

## 10 FUTURE ENHACMENT AND IMPROVEMENTS

The application, as it stands now, demonstrates a commendable level of functionality. However, it presents an opportunity for numerous enhancements and additions that can elevate it to new heights in terms of security, reliability, user-friendliness, performance, and overall user experience. By incorporating these improvements, we can ensure a more robust and efficient application that exceeds user expectations and delivers a seamless and satisfying experience.

### 10.1 Smart contract improvements

The smart contract is the core of the application, and it should be the most accurate component in the application, because once a smart contract is deployed to a main or test net network it is no more editable, because of the blockchain nature (immutability). And if there was any security vulnerabilities or bugs, the hackers will try to take the opportunity and take down the contract, and make a huge damage to the platform, owner of contract and the platform's funds. Hence the following should be considered to improve the smart contract code:

- 1. More restrictions:** It is important to use Solidity modifiers, require keywords/behaviors to ensure that no unwanted behavior is executed. Almost every store operation of the data in the contract should be checked before the real execution of the commands.
- 2. Writing an effective code:** Writing an effective code that is clean and short, can lead to less gas fees, because every operation on the blockchain (specifically Ethereum blockchain) is translated to low-level machine code (memory instruction), and every memory instruction in the Ethereum virtual machine require specific gas fee. More clean and effective code means less gas fees. Writing an effective code is an endless goal, and from my point of view I feel there is always a space to improve the code.
- 3. Usage of analyzers frameworks:** When building smart contract, security is the most curtail topic to talk about, there are some frameworks that runs a suite of vulnerability detectors, prints visual information about contract details, such as “**Slither**”<sup>27</sup>, such tool can help to make the contract more secure and fault tolerance.
- 4. Possibility to add a friend:** Almost every social media platform provides this functionality, having and connecting with friends in social media platforms make the social media attracts more users.

---

<sup>27</sup> Slither framework, smart contract analyzer: <https://github.com/crytic/slither>

5. **Possibility to edit/delete content:** Users should be able to modify their personal information and modify/delete their comments and posts, with the help of the Enums we have in the contract, such that when user chooses to delete a post for example the post can be hidden from the front-end by checking its status on the blockchain.

## 10.2 Gas fees improvements and choosing of the suitable blockchain network.

Since user satisfaction is hugely related to gas fees when interacting with a DApp. When a platform does not let users feel that they are interacting with a new technology specially when talking about decentralized social medias, a user will feel so much comfortable, in other words when the platform does not ask users for a transaction fee to be accepted by a user on every specific operation, is much better than asking the user for the agreement of every action/transaction the user does. Even though if the platform asks for a direct fee transaction that should be paid by user, the transaction fee should be reduced as much as possible using specific approaches, there are several approaches a contract/platform owner can take to reduce or eliminate gas fees from the user side:

1. **Deployment on a suitable blockchain network:** Layer 2 scaling blockchain was created for the purpose of fulfilling the requirement to make a DApp better to deal with, and more user friendly. Layer 2 and side chains blockchains were created for the purpose to address the limitation of layer 1 blockchain specially in the world of Web3.0. Most of the layer 2 blockchains have a low gas fee and faster transaction per second. Such blockchain is Polygon that is built on top of Ethereum. Or even the usage and deployment on a better standalone blockchain network such as “**Avalanche**”<sup>28</sup> when comparing Avalanche to Ethereum or Polygon, also faster transaction speed per second.
2. **Usage of so-called meta-transactions.**

### 10.2.1 Usage of so-called meta-transactions

Meta transactions is the process where users do not pay for transaction fees. This technique allows users the choice to sign a transaction for free and have a third party securely execute it, with the third party paying the gas to do so. Expecting the typical user to purchase cryptocurrency and pay for gas to use a DApp is impractical. To solve this problem, it is possible to separate the sender of a transaction from the role of the payer, opening the

---

<sup>28</sup> Avalanche blockchain: <https://www.investopedia.com/avalanche-avax-definition-5217374>

possibility of scaling transaction execution and starting a seamless transacting experience. Meta transactions allow anyone to interact with the blockchain. They do not require users to have tokens to pay for the network's services through transaction fees. This is done by decoupling the sender of a transaction and the payer of gas. Usually, meta transactions occur with the help of third party relayers that are responsible for this process. In conclusion, meta transactions are a structure or specific design pattern where:

1. A user (sender) transmits a request to a relayer after signing it with their private key.
2. The relayer will be responsible for wrapping the request into a transaction and sending it to a contract.
3. The contract unwraps the transaction and executes it.

There are some existing relayers that can help to implement the idea of meta transactions, such as **Infura**<sup>29</sup>, **Biconomy**<sup>30</sup>. [65]

### 10.3 Front-end and client-side improvements

The front end is the gate of the application to the users, it should give a smooth experience to the users, and an easy way to navigate between components and platform's functionalities. Also, color consistency is an important topic when talking about taking care of the UI. For now, several possible improvements can be added to the platform such as:

1. **Switching between dark and light theme.**
2. **Apply pagination to the feed page.**
3. **Ability to use different networks accounts on the platform:** User should be able to register for example with his account from different network that the platform is deployed on.
4. **Support of different types of wallets.**

---

<sup>29</sup> Infura ITX meta-transactions: <https://docs.infura.io/infura/features/itx-transactions/itx-meta-transactions>

<sup>30</sup> Biconomy gasless transaction: <https://docs.biconomy.io/build-with-biconomy-sdk/gasless-transactions>



## CONCLUSION

The comprehensive exploration of blockchain technology, its various types, and its applications have provided valuable insights into its potential. We have examined popular blockchains such as Bitcoin and Ethereum, focusing mainly on Ethereum blockchain, delving into their functionalities and limitations. Side chains have also been discussed, highlighting their benefits and drawbacks. Additionally, the advent of Web 3.0 has ushered in a new era of decentralized social networks, addressing the shortcomings of centralized platforms.

Furthermore, this exploration delved into real-life examples of decentralized social networks, providing a deeper understanding of their unique characteristics and advantages. Platforms like Steemit, Minds, and Lens Protocol showcased different approaches to decentralization, content monetization, and user governance. These examples shed light on the diverse possibilities that decentralized social networks offer, including rewarding users for their contributions, ensuring censorship-resistant content, and fostering community-driven decision-making.

In the practical realm, we have analyzed an application's architecture, utilizing front-end and back-end technologies to meet functional and non-functional requirements. The integration of Web3 technologies has facilitated interaction with smart contracts and the blockchain. Emphasis has been placed on security architecture, incorporating measures such as JSON Web Tokens for enhanced protection.

Setting up the development environment and implementing the application's features, including user registration, post creation, and commenting, has demonstrated the potential of blockchain-based platforms. Automated testing has been conducted to ensure the robustness of the smart contract.

Looking to the future, there are opportunities for further enhancements, particularly in smart contract optimization and gas fee reduction. Improving the user interface and client-side experience will contribute to a more seamless and enjoyable application.

In conclusion, this exploration of blockchain technology, its practical applications, and the development of a blockchain-based decentralized social network has provided valuable insights into the transformative potential of this innovative approach. The ability to create a secure, transparent, and user-centric social networking experience has been demonstrated.

**BIBLIOGRAPHY**

- [1] What is a blockchain? *Lisk* [online]. [Accessed 13 August 2022]. Available from:  
<https://lisk.com/learn/about-web3/what-is-a-blockchain>
- [2] PANDEY, Rudramani. What is a block in blockchain? SAP Blogs [online]. 14 January 2019. [Accessed 11 March 2023]. Available from:  
<https://blogs.sap.com/2019/01/14/what-is-a-block-in-blockchain/>
- [3] TECH, Blockchain. Structure of the block inside a Blockchain Network. Medium [online]. 22 March 2022. [Accessed 11 March 2023]. Available from:  
<https://medium.com/coinmonks/structure-of-the-block-inside-a-blockchain-network-7ad66ea5bea>
- [4] FRANKENFIELD, Jake. Nonce: What it means and how it's used in Blockchain. Investopedia [online]. 24 October 2022. [Accessed 15 March 2023]. Available from:  
<https://www.investopedia.com/terms/n/nonce.asp>
- [5] COINTELEGRAPH. A beginner's guide to the different types of blockchain networks. *Cointelegraph* [online]. 27 July 2022. [Accessed 14 August 2022]. Available from:  
<https://cointelegraph.com/blockchain-for-beginners/a-beginners-guide-to-the-different-types-of-blockchain-networks>
- [6] PARIZO, Christine. What are the 4 different types of blockchain technology? *SearchCIO* [online]. 28 May 2021. [Accessed 14 August 2022]. Available from:  
<https://www.techtarget.com/searchcio/feature/What-are-the-4-different-types-of-blockchain-technology>
- [7] GERONI, Diego. Hybrid blockchain: The best of both worlds. 101 Blockchains [online]. 10 November 2021. [Accessed 14 August 2022]. Available from:  
<https://101blockchains.com/hybrid-blockchain/>
- [8] SHARMA, Toshendra Kumar. Types of blockchains explained- public vs. private VS. consortium. Web3 & Blockchain Certifications [online]. 10 August 2020. [Accessed 14

- August 2022]. Available from: <https://www.blockchain-council.org/blockchain/types-of-blockchains-explained-public-vs-private-vs-consortium/>
- [9] AFREEN, Sana. Why is blockchain important and why does it matters? [2022]: Simplilearn. Simplilearn.com [online]. 12 August 2022. [Accessed 14 August 2022].
- [10] FRANKENFIELD, Jake. Distributed Ledger Technology. Investopedia [online]. 8 February 2022. [Accessed 17 August 2022]. Available from: <https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp>
- [11] 19 blockchain application use cases that will surprise you. The Entire Supply Chain. Reimagined [online]. [Accessed 15 August 2022]. Available from: <https://supplain.io/news/blockchain-applications-use-cases>
- [12] Blockchain explained and its application to payments. Paiementor [online]. 1 March 2020. [Accessed 17 August 2022]. Available from: <https://www.paiementor.com/blockchain-explained-application-payments/>
- [13] ANWAR, Hasib. Blockchain for Digital Identity: The decentralized and self-sovereign identity (SSI). 101 Blockchains [online]. 15 August 2022. [Accessed 29 January 2023]. Available from: <https://101blockchains.com/digital-identity/>
- [14] Blockchain in Supply Chain: Benefits top use cases in 2023. AIMultiple [online]. [Accessed 2 March 2023]. Available from: <https://research.aimultiple.com/blockchain-supply-chain/>
- [15] What are the benefits of Blockchain for government services? Appinventiv [online]. 4 July 2022. [Accessed 21 February 2023]. Available from: <https://appinventiv.com/blog/role-of-blockchain-in-government/>
- [16] What are smart contracts? CryptoNinjas [online]. 17 March 2021. [Accessed 16 March 2023]. Available from: <https://www.cryptoninjas.net/what-are-smart-contracts/>
- [17] SIMPLILEARN. What is blockchain wallet and how does it work? [updated]. Simplilearn.com [online]. 20 February 2023. [Accessed 17 March 2023]. Available from: <https://www.simplilearn.com/tutorials/blockchain-tutorial/blockchain-wallet>

- [18] What is proof of work (POW) | explained for Beginners. YouTube [online]. 22 October 2018. [Accessed 18 August 2022]. Available from:  
<https://www.youtube.com/watch?v=3EUAcxhuoU4>
- [19] COMIDOR LOW-CODE AUTOMATION PLATFORM. Blockchain technology definition and Fundamentas: Comidor. Comidor Low-code Automation Platform [online]. 8 July 2022. [Accessed 11 March 2023]. Available from:  
<https://www.comidor.com/knowledge-base/blockchain-technology-knowledge-base/blockchain-fundamentals/>
- [20] Proof-of-stake (vs proof-of-work). YouTube [online]. 21 March 2018. [Accessed 18 August 2022]. Available from: [https://www.youtube.com/watch?v=M3EFi\\_POhps](https://www.youtube.com/watch?v=M3EFi_POhps)
- [21] KASPERSKY. What is cryptocurrency and how does it work? *www.kaspersky.com* [online]. 9 February 2022. [Accessed 13 November 2022]. Available from:  
<https://www.kaspersky.com/resource-center/definitions/what-is-cryptocurrency>
- [22] ROSIC, Ameer, BLOCKGEEKS, ZAPOTOCHNY, Andrew and BAGGETTA, Matthew. What is cryptocurrency? [everything you need to know!]. *Blockgeeks*

- [online]. 19 October 2022. [Accessed 13 November 2022]. Available from:  
<https://blockgeeks.com/guides/what-is-cryptocurrency/>
- [23] HUSSEY, Matt. What was Digicash?: The beginner's guide. *Decrypt* [online]. 10 July 2019. [Accessed 13 November 2022]. Available from:  
<https://decrypt.co/resources/digicash-what-is-cryptocurrency-explainer>
- [24] The dao: What was the dao hack? Gemini [online]. [Accessed 22 March 2023].  
Available from: <https://www.gemini.com/cryptopedia/the-dao-hack-makerdao>
- [25] FRANKENFIELD, Jake. What is bitcoin? how to mine, buy, and use it. Investopedia [online]. 23 November 2022. [Accessed 8 March 2023]. Available from:  
<https://www.investopedia.com/terms/b/bitcoin.asp>
- [26] FRANKENFIELD, Jake. What is ethereum and how does it work? Investopedia [online]. 20 January 2023. [Accessed 8 March 2023]. Available from:  
<https://www.investopedia.com/terms/e/ethereum.asp>
- [27] JERGA, Eincode by Filip and JERGA, Filip. Solidity & Ethereum in react (NEXT JS): The Complete Guide. Udemy [online]. [Accessed 11 March 2023]. Available from: <https://www.udemy.com/course/solidity-ethereum-in-react-next-js-the-complete-guide/>
- [28] HORVAT, Matija. Matija Horvat. Be on the Right Side of Change [online]. [Accessed 7 April 2023]. Available from: <https://blog.finxter.com/ethereum-virtual-machine-memory-and-instruction-set-solidity-smart-contracts/>
- [29] Yellow paper - github pages. [online]. [Accessed 7 April 2023]. Available from:  
<https://ethereum.github.io/yellowpaper/paper.pdf>
- [30] Introduction to Ethereum | Mastering Blockchain Programming with Solidity. Packt subscription [online]. [Accessed 15 March 2023]. Available from:  
<https://subscription.packtpub.com/book/data/9781839218262/2/ch02lv11sec04/introduction-to-ethereum>
- [31] KLEMENS, Sam. Ethereum Review: Ethereum use cases, Advantages & Disadvantages. Exodus Crypto News & Insights [online]. 5 March 2021.

- [Accessed 13 April 2023]. Available from: <https://www.exodus.com/news/ethereum-review/#head3>
- [32] What are sidechains? How do bitcoin and crypto work?: Get started with Bitcoin.com. Buy Bitcoin & cryptocurrency [online]. [Accessed 10 April 2023]. Available from: <https://www.bitcoin.com/get-started/what-are-sidechains/#2/>
- [33] ROTH, Stephan. An introduction to sidechains. CoinDesk Latest Headlines RSS [online]. 7 March 2022. [Accessed 10 April 2023]. Available from: <https://www.coindesk.com/learn/an-introduction-to-sidechains/>
- [34] APIS, Team Crypto. Get to know polygon: Ethereum's layer 2 scaling solution. Crypto APIs blockchain infrastructure suite [online]. 7 October 2022. [Accessed 10 April 2023]. Available from: <https://cryptoapis.io/blog/111-get-to-know-polygon-ethereums-layer-2-scaling-solution>
- [35] KHITROV, Sergei. Pros and cons of development on Polygon. HackerNoon [online]. 3 August 2022. [Accessed 11 April 2023]. Available from: <https://hackernoon.com/pros-and-cons-of-development-on-polygon>
- [36] DEMERS, Vincent. The added value of web 2.0 and social networking sites. Entrepreneur Web [online]. [Accessed 25 March 2023]. Available from: <https://www.entrepreneurweb.com/added-value-web-20-and-social-networking-sites>
- [37] STAFF, History Computer. Web 2.0 explained: Everything you need to know. History [online]. 1 May 2022. [Accessed 15 August 2022]. Available from: <https://history-computer.com/web-2-0/>
- [38] ARPIT, KHATIWARA, Bibek, CHHAPERIA, Vishal, VIJAYD, VENUGOPAL, Vimal, KSHITIJA, BEENA, SRINATH, R., MAHESH, JOEL, SAMEER, SUBHAS, QASTATION, CHANDRU, MAHENDRA, VINOTH, TIGER2K, SAMEERA, ANILKUMAR, APARNA, SHYAM, KARIM, SAMEERA, SAM, R, DHINESH.N, TAMIL, SELVAM, RUKMAL, VIPIN, JAYARAJ, karthik, SANJUKTA, CHINNI, SANJUKTA, SAM, SHRINVIAS, SUDARSHAN, AUTHOR, ragu STH, ANGEL, PRASAD, Hari, RANI, GURMEET, HEMANG and MAHESH. Client server testing, web testing & desktop testing guide. Software Testing Help [online]. 24 February

2023. [Accessed 25 March 2023]. Available from:  
<https://www.softwaretestinghelp.com/what-is-client-server-and-web-based-testing-and-how-to-test-these-applications/comment-page-1/>
- [39] Usage statistics of JavaScript as client-side programming language on websites. W3Techs [online]. [Accessed 17 August 2022]. Available from:  
<https://w3techs.com/technologies/details/cp-javascript>
- [40] KERNER, Sean Michael and GILLIS, Alexander S. What is web 3.0? - definition from techtarget.com. WhatIs.com [online]. 10 June 2022. [Accessed 17 August 2022]. Available from: [https://www.techtarget.com/whatis/definition/Web-30#:~:text=Web%203.0%20\(Web3\)%20is%20the,providing%20website%20and%20a%20application%20services](https://www.techtarget.com/whatis/definition/Web-30#:~:text=Web%203.0%20(Web3)%20is%20the,providing%20website%20and%20a%20application%20services).
- [41] SAMUR, Alexandra. The history of social media in 33 key moments. Social Media Marketing & Management Dashboard [online]. 6 April 2023. [Accessed 17 April 2023]. Available from: <https://blog.hootsuite.com/history-social-media/#:~:text=The%20first%20social%20media%20site,only%20to%20shutter%20in%202000>.
- [42] R, Rahul A. Decentralized Social Network: Breaking the boundaries of Social Media. Accubits Blog [online]. 11 March 2022. [Accessed 20 April 2023]. Available from: <https://blog.accubits.com/decentralized-social-network-breaking-the-boundaries-of-social-media/>
- [43] Social Media and freedom of speech and expression. Legal Service India - Law, Lawyers and Legal Resources [online]. [Accessed 18 April 2023]. Available from:

<https://www.legalserviceindia.com/legal/article-426-social-media-and-freedom-of-speech-and-expression.html>

- [44] What are decentralized social networks? - cointelegraph.com. [online]. [Accessed 20 April 2023]. Available from: <https://cointelegraph.com/explained/what-are-decentralized-social-networks>
- [45] BEAVERS, Jack. What is Steemit and the smart media token (SMT)? Moralis Academy [online]. 23 December 2022. [Accessed 25 April 2023]. Available from: <https://academy.moralis.io/blog/what-is-steemit-and-the-smart-media-token-smt>
- [46] STEEMIT. Steemit. SourceForge [online]. 15 April 2023. [Accessed 25 April 2023]. Available from: <https://sourceforge.net/software/product/Steemit/>
- [47] Steemit [online]. 31 December 1969. [Accessed 25 April 2023]. Available from: <https://steemit.com/>
- [48] ADMIN. What is minds? blockchain social network (minds token) pays users. Oh I Will [online]. 7 October 2020. [Accessed 25 April 2023]. Available from: <https://ohiwill.com/what-is-minds-blockchain-social-network-minds-token-pays-users/>
- [49] What is minds? – minds. [online]. [Accessed 24 April 2023]. Available from: <https://support.minds.com/hc/en-us/articles/4402687840916-What-is-Minds->
- [50] PRODUCTS, CryptoSlate. Minds. CryptoSlate [online]. 4 February 2020. [Accessed 25 April 2023]. Available from: <https://cryptoslate.com/products/minds/>
- [51] Lenster.xyz [online]. [Accessed 25 April 2023]. Available from: <https://lenster.xyz/>
- [52] Lens protocol. Lens Protocol [online]. [Accessed 25 April 2023]. Available from: <https://www.lens.xyz/apps>
- [53] ZAINAB. Lens Protocol - Dapps. IQ.Wiki [online]. 18 April 2023. [Accessed 25 April 2023]. Available from: <https://iq.wiki/wiki/lens-protocol>
- [54] SAGAR, Paresh. What is a single-page application? meaning, pitfalls & benefits. Excellent Webworld [online]. 11 April 2023. [Accessed 14 April 2023]. Available from: <https://www.excellentwebworld.com/what-is-a-single-page-application/>



- [55] TOMAR, Pari. Hardhat or truffle? as a beginner blockchain developer, which one do I pick. Medium [online]. 31 May 2022. [Accessed 19 April 2023]. Available from: <https://medium.com/buildbear/hardhat-or-truffle-as-a-beginner-blockchain-developer-which-one-do-i-pick-34a6924a6983>
- [56] concepts. IPFS Docs [online]. [Accessed 20 April 2023]. Available from: <https://docs.ipfs.tech/concepts/>
- [57] The IPFS protocol explained with examples - welcome to the decentralized web. YouTube [online]. 31 January 2021. [Accessed 20 April 2023]. Available from: <https://www.youtube.com/watch?v=PlvMGpQnqOM>
- [58] Solidity tutorial. Tutorials Point [online]. [Accessed 20 April 2023]. Available from: <https://www.tutorialspoint.com/solidity/index.htm>
- [59] Solidity. Solidity [online]. [Accessed 20 April 2023]. Available from: <https://docs.soliditylang.org/en/v0.8.19/>
- [60] AXEN, Douglas. WEB3 libraries - list of WEB3 libraries for DEVS in 2023. Moralis Web3 | Enterprise-Grade Web3 APIs [online]. 25 January 2023. [Accessed 21 April

- 2023]. Available from: <https://moralis.io/web3-libraries-list-of-web3-libraries-for-devs-in-2023/>
- [61] AUTH0.COM. JSON web tokens introduction. JSON Web Token Introduction [online]. [Accessed 19 April 2023]. Available from: <https://jwt.io/introduction>
- [62] What is JWT?: Akana by perforce. Akana [online]. [Accessed April 2023]. Available from: <https://www.akana.com/blog/what-is-jwt>
- [63] SAHU, Ankit. 5 types of software architecture patterns. Turing Blog [online]. 8 February 2023. [Accessed 18 April 2023]. Available from: <https://www.turing.com/blog/software-architecture-patterns-types/>
- [64] The Clean Code Blog. Clean Coder Blog [online]. [Accessed 18 April 2023]. Available from: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- [65] Meta transactions. Polygon Wiki [online]. [Accessed 28 April 2023]. Available from: <https://wiki.polygon.technology/docs/develop/meta-transactions/meta-transactions/>
- [66] The fun, simple, flexible JavaScript test framework. Mocha [online]. [Accessed 18 May 2023]. Available from: <https://mochajs.org/>
- [67] Chai assertion library. Chai [online]. [Accessed 18 May 2023]. Available from: <https://www.chaijs.com/>
- [68] Testing contracts: Ethereum development environment for professionals by Nomic Foundation. 5. Testing contracts | Ethereum development environment for professionals by Nomic Foundation [online]. [Accessed 18 May 2023]. Available from: <https://hardhat.org/tutorial/testing-contracts>

**LIST OF ABBREVIATIONS**

DApp	Decentralized application
SSI	Self-sovereign identity
EVM	Ethereum virtual machine
CSS	Cascading style sheet
HTML	Hypertext markup language
PHP	Hypertext preprocessor
DOM	Document object model
ECMA	European computer manufacturers association
DLT	Distributed ledger technology
NONCE	Number used once
GPU	Graphic processing unit
POW	Proof-of-work
POS	Proof-of-stake
DPOS	Delegated Proof-of-stake
JWT	JSON web token
LIFO	Last in first out
BMM	Blind merged mining
SSR	Server side rendering
CSR	Client side rendering
SEO	Search engine optimization
UML	Unified modeling language
API	Application programming interface
IPFS	Interplanetary file system
ORM	Object relational mapper
EF	Entity framework

JSON	JavaScript object notation
ABI	Application binary interface
UI	User interface
NFT	Non-fungible token
CID	Content identifier

**LIST OF FIGURES**

<i>Figure 1. Blocks of blockchain [2]</i> .....	16
<i>Figure 2. Block structure [2]</i> .....	16
<i>Figure 3. Markle tree [2]</i> .....	17
<i>Figure 4. Money transfer blockchain [12]</i> .....	22
<i>Figure 5. Blockchain wallet [17]</i> .....	27
<i>Figure 6. Proof of work Process [19]</i> .....	29
<i>Figure 7. Solidity smart contract example</i> .....	37
<i>Figure 8. The Ethereum world state and nodes</i> .....	39
<i>Figure 9. Ethereum gas price</i> .....	41
<i>Figure 10. Ethereum yellow paper [29]</i> .....	42
<i>Figure 11. Contract based account in the Ethereum world state</i> .....	43
<i>Figure 12. Ethereum externally owned account [30]</i> .....	44
<i>Figure 13. Contract account on the Ethereum network [30]</i> .....	45
<i>Figure 14. Side chains architecture [33]</i> .....	46
<i>Figure 15. Polygon architecture [35]</i> .....	49
<i>Figure 16. Web 2.0 features [36]</i> .....	51
<i>Figure 17. General Communication Architecture[38]</i> .....	52
<i>Figure 18. HTML simple page</i> .....	52
<i>Figure 19. HTML code result</i> .....	53
<i>Figure 20. HTML with inline CSS</i> .....	54
<i>Figure 21. CSS code result</i> .....	54
<i>Figure 22. HTML with JavaScript</i> .....	55
<i>Figure 23. JavaScript Result Code</i> .....	56
<i>Figure 24. Centralized Social Network [42]</i> .....	59
<i>Figure 25. Decentralized social media architecture</i> .....	63
<i>Figure 26. Steemit user interface</i> .....	65
<i>Figure 27. Steemit platform's keys</i> .....	65
<i>Figure 28. Minds' user interface</i> .....	66
<i>Figure 29. DApps on Lens protocol</i> .....	69
<i>Figure 30. Lenster user interface</i> .....	70
<i>Figure 31. Registered user use cases</i> .....	78
<i>Figure 32. Unregistered user use cases</i> .....	78

<i>Figure 33. UML Class model</i> .....	81
<i>Figure 34. SPA vs multi page app [54]</i> .....	82
<i>Figure 35. NextJS server-side rendering example</i> .....	83
<i>Figure 36. NextJS middleware</i> .....	85
<i>Figure 37. HTML flex box</i> .....	85
<i>Figure 38. CSS flex box</i> .....	86
<i>Figure 39. Tailwind CSS example</i> .....	86
<i>Figure 40. Tailwind CSS and normal styling result</i> .....	86
<i>Figure 41. Social chain index page</i> .....	87
<i>Figure 42. Social chain login page</i> .....	88
<i>Figure 43. Social chain register modal</i> .....	88
<i>Figure 44. Social chain feed/home page</i> .....	89
<i>Figure 45. Social chain user profile page</i> .....	89
<i>Figure 46. Social chain post modal</i> .....	90
<i>Figure 47. IPFS overview[57]</i> .....	93
<i>Figure 48. Solidity modifier and function</i> .....	95
<i>Figure 49. Solidity event and emit.</i> .....	96
<i>Figure 50. JWT hashing</i> .....	99
<i>Figure 51. JWT encoder and decoder [61]</i> .....	100
<i>Figure 52. Application's architecture</i> .....	104
<i>Figure 53. Ganache user interface and workspace</i> .....	106
<i>Figure 54. Ganache blockchain</i> .....	106
<i>Figure 55. MetaMask Digital wallet</i> .....	107
<i>Figure 56. MetaMask network configs</i> .....	108
<i>Figure 57. Adding Ethereum account to digital wallet</i> .....	108
<i>Figure 58. C# User entity/class</i> .....	109
<i>Figure 59. Application's database context</i> .....	110
<i>Figure 60. EF migration result</i> .....	111
<i>Figure 61. Updating database from migration result</i> .....	111
<i>Figure 62. Hardhat network configs</i> .....	112
<i>Figure 63. Hardhat compilation configs</i> .....	113
<i>Figure 64. Hardhat compilation result</i> .....	113
<i>Figure 65. Hardhat deploying script.</i> .....	114

<i>Figure 66. Contract deployment result</i> .....	114
<i>Figure 67. Contract's states and variables</i> .....	115
<i>Figure 68. Contract's mapping</i> .....	116
<i>Figure 69. Contract's events</i> .....	117
<i>Figure 70. Username availability smart contract's function</i> .....	118
<i>Figure 71. Register user smart contract's function</i> .....	119
<i>Figure 72. Get user contract's function.</i> .....	120
<i>Figure 73. Create post contract's function.</i> .....	121
<i>Figure 74. Get post by id contract's function.</i> .....	121
<i>Figure 75. Get user posts contract's function.</i> .....	122
<i>Figure 76. Get platform posts - pagination.</i> .....	123
<i>Figure 77. Likes post contract's function.</i> .....	123
<i>Figure 78. Unlike post contract's function.</i> .....	124
<i>Figure 79. is liked by address contract's function.</i> .....	124
<i>Figure 80. Create comment contract's function.</i> .....	125
<i>Figure 81. Get comment by Id contract's function.</i> .....	125
<i>Figure 82. Get post's comments contract's function.</i> .....	126
<i>Figure 83. Contract's modifiers.</i> .....	127
<i>Figure 84. User registration flowchart.</i> .....	129
<i>Figure 85. User enters the website.</i> .....	130
<i>Figure 86. User Request sign page</i> .....	130
<i>Figure 87. User registration details.</i> .....	131
<i>Figure 88. User sign the transaction to register.</i> .....	131
<i>Figure 89. User sign the message to get JWT token.</i> .....	132
<i>Figure 90. Successful registration.</i> .....	132
<i>Figure 91. User login flowchart.</i> .....	134
<i>Figure 92. User login</i> .....	135
<i>Figure 93. User sign message to obtain JWT</i> .....	135
<i>Figure 94. Forward to profile after successful login</i> .....	135
<i>Figure 95. Filling new post data.</i> .....	136
<i>Figure 96. Submission of new post.</i> .....	136
<i>Figure 97. Post creation result.</i> .....	137
<i>Figure 98. Filling in new comment data.</i> .....	138

<i>Figure 99. Submission of new comment.</i> .....	138
<i>Figure 100. Comment creation result.</i> .....	139
<i>Figure 101. Like a post.</i> .....	140
<i>Figure 102. Like's result.</i> .....	140
<i>Figure 103. Post's comments link.</i> .....	141
<i>Figure 104. Post modal.</i> .....	142
<i>Figure 105. Comment's pagination result.</i> .....	142
<i>Figure 106. Register user test suite.</i> .....	144
<i>Figure 107. Register user test suite result.</i> .....	145
<i>Figure 108. Get user test suite.</i> .....	145
<i>Figure 109. Get user test suite result.</i> .....	146
<i>Figure 110. Username availability test suite.</i> .....	146
<i>Figure 111. Username availability test suite result.</i> .....	147
<i>Figure 112. Create post test suite.</i> .....	147
<i>Figure 113. Create post test suite result.</i> .....	148
<i>Figure 114. Get post by id test suite.</i> .....	148
<i>Figure 115. Get post by id test suite result.</i> .....	149
<i>Figure 116. Get user's posts test suite.</i> .....	149
<i>Figure 117. Get user's post test suite result.</i> .....	150
<i>Figure 118. Get posts' ids test suite.</i> .....	150
<i>Figure 119. Get posts' ids test suite result.</i> .....	151
<i>Figure 120. Like post test suite.</i> .....	152
<i>Figure 121. Like post test suite result.</i> .....	152
<i>Figure 122. Un like post test suite.</i> .....	153
<i>Figure 123. Un like post test suite result.</i> .....	153
<i>Figure 124. Is liked by address test suite.</i> .....	154
<i>Figure 125. Is liked by address test suite result.</i> .....	154
<i>Figure 126. Create comment and get comment by id test suite.</i> .....	155
<i>Figure 127. Create comment and get comment by id test suite result.</i> .....	155
<i>Figure 128. Get post's comments test suite.</i> .....	156
<i>Figure 129. Get post's comments test suite result.</i> .....	156
<i>Figure 130. Total test suites results.</i> .....	157



**LIST OF TABLES**

<i>Table 1. Types of blockchain overview .....</i>	<i>20</i>
<i>Table 2. Decentralized social network comparison (functionality) .....</i>	<i>71</i>
<i>Table 3. Decentralized social network comparison (client-side perspective) .....</i>	<i>72</i>

## **APPENDICES**

**APPENDIX P I: APPENDIX TITLE**