A Decentralized Messaging App built on Blockchain Technologies

Mohammed Jamal

Bachelor's thesis
2023

Tomas Bata University in Zlín
Faculty of Applied Informatics

Tomas Bata University in Zlín
Faculty of Applied Informatics
Department of Informatics and Artificial Intelligence

Academic year: 2022/2023

ASSIGNMENT OF BACHELOR THESIS
(project, art work, art performance)

Name and surname:	**Mohammed Khalid Jamal**
Personal number:	**A19900**
Study programme:	**B0613A140021 Software Engineering**
Type of Study:	**Full-time**
Work topic:	**Aplikace pro odesílání zpráv postavená na technologiích blockchain**
Work topic in English:	**A Decentralized Messaging App built on Blockchain Technologies**

Theses guidelines

1. Study and describe terminology of blockchain technologies including actual state according to the topic.
2. Choose appropriate technologies.
3. Design the messaging application built on blockchain technologies.
4. Implement your solution of messaging application using the blockchain technologies including smart contracts.
5. Test the application and appropriately present the results.

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

Recommended resources:

1. Secretum. Revolutionary Secure Messaging & Trading Dapp powered by Solana [online]. [cit. 2022-09-26]. Dostupné z: https://secretum.io/market-comparison/
2. Solana Cookbook. (2022). Retrieved September 26, 2022, from https://solanacookbook.com/#contributing
3. Hoover, D. H., & Solorio, K. (2019). Hands-On Smart Contract Development with Solidity and Ethereum. O'Reilly Media.
4. RAZO-ZAPATA, Iván S. a Mihail MIHAYLOV. Blockchain as messaging infrastructure for smart grids. In: Local Electricity Markets [online]. Elsevier, 2021, 2021, s. 199-213 [cit. 2022-09-26]. ISBN 9780128200742. Dostupné z: doi:10.1016/B978-0-12-820074-2.00013-7
5. ELLEWALA, U. P., W.D.H.U AMARASENA, H.V Sachini LAKMALI, L.M.K SENANAYAKA a A.N. SENARATHNE. Secure Messaging Platform Based on Blockchain. In: 2020 2nd International Conference on Advancements in Computing (ICAC) [online]. IEEE, 2020, 2020-12-10, s. 317-322 [cit. 2022-09-26]. ISBN 978-1-7281-8412-8. Dostupné z: doi:10.1109/ICAC51239.2020.9357306
6. 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.

Supervisors of bachelor thesis:	**Ing. Petr Žáček, Ph.D.** Department of Informatics and Artificial Intelligence
Consultant of bachelor thesis:	**prof. 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**

L.S.

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

ABSTRAKT

Cílem této práce je prozkoumat možnosti využití blockchain a smart contract technologie při vývoji aplikace pro zasílání zpráv, která eliminuje závislost na centrálním serveru. Teoretická část práce poskytuje komplexní přehled současného stavu jak decentralizovaných, tak centralizovaných aplikací pro zasílání zpráv, a současně zkoumá terminologii a architekturu blockchain technologie a smart contract. Také popisuje reálné případy použití blockchain technologií a předkládá analýzu výhod a nevýhod spojených s blockchain technologií. Teoretická část navíc rozebírá technologie použité při vývoji aplikace pro zasílání zpráv. Praktická část práce se zaměřuje na implementaci blockchain technologií do aplikace pro zasílání zpráv. Hlavním cílem integrování blockchainu je zvýšení bezpečnosti aplikace, důraz na ochranu soukromí uživatelů a revoluce v řízení uživatelských informací a služeb. Decentralizovaná aplikace pro zasílání zpráv funguje bez centrálního serveru a je postavena na blockchainu Polygon.

Klíčová slova: Blockchain, Chytré smlouvy, Decentralizované aplikace, Solidity, Polygon Blockchain, Ethereum, Aplikace pro zasílání zpráv.

ABSTRACT

The objective of this thesis is to explore the potential application of blockchain technology and smart contracts in the development of a decentralized messaging app that eliminates reliance on a central server. The theoretical part of the thesis provides a comprehensive overview of the current state of both decentralized and centralized messaging applications, along with an exploration of the terminology and architecture of blockchain technology and smart contracts. It also describes the real-world use cases of blockchain technologies and presents an analysis of the advantages and disadvantages associated with blockchain technology. Additionally, the theoretical section delves into the technologies utilized in the development of the messaging app. The practical part of the thesis focuses on the implementation of blockchain technologies within the messaging app. The primary objective of integrating blockchain is to enhance the app's security, prioritize user privacy, and revolutionize the management of user information and services. The decentralized messaging app operates without a central server and is built on the Polygon blockchain.

Keywords: Blockchain, Smart Contracts, Decentralized Apps, Solidity, Polygon Blockchain, Ethereum. Messaging App.

ACKNOWLEDGEMENTS

I would like to express my deepest and most heartfelt gratitude to everyone who has been an unwavering pillar of support throughout my academic journey. Your constant love, encouragement, and unwavering belief in my abilities have been instrumental in my achievements and success.

My beloved family and friends, I am sincerely grateful for your unending support, understanding, and encouragement. Your belief in my abilities, even during the most challenging times, has pushed me to overcome obstacles and reach new heights. I am forever blessed to have such an incredible support system in my life.

I am deeply grateful to my esteemed supervisor, Ing. Petr Žáček, Ph.D., for his invaluable guidance, unwavering commitment, and mentorship. His trust, wisdom, and passion have shaped my academic journey profoundly. Additionally, I deeply appreciate Prof. Ing. Roman Šenkeřík, Ph.D., for his invaluable contributions as my consultant. His insights, expertise, and unwavering support greatly enriched my research.

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.

INTRODUCTION

The rise of smartphones and the internet in today's world has made communication much easier than it was previously. Messaging apps are among the most popular tools for routine communication, and most individuals prefer communication apps over face-to-face engagement. Every day, they create and deploy new messaging app features. Large technological corporations host most of today's most popular chat platforms. It means that they rely on a central server for user communication and keep user data on this server, over which they have complete control of the apps' functioning. The main issue with this method is that if the central server fails, the entire network collapses.

The objective behind the implementation of blockchain technology was to enhance transaction security. In addition to their application in cryptocurrencies, blockchain technologies have the potential to revolutionize decentralized systems. They can be employed to facilitate secure communication, maintain data integrity, and establish peer-to-peer (P2P) networks. This project aims to build a messaging application that replicates the functionality of traditional messaging apps while eliminating the need for a central authority. The messaging app is built on the Polygon blockchain and utilizes the Solidity programming language. The thesis is structured as follows: firstly, it presents an overview of the current state of centralized and decentralized systems, along with an explanation of the key terminology and architecture of blockchain and smart contracts. It also provides an overview of various blockchain types and programming languages commonly used for smart contracts. Secondly, it explores the technologies and libraries employed in the development of messaging applications. It showcases the process of developing the messaging application using these technologies, including the web presentation of the final result. The thesis further discusses the achievements of the project and encompasses the testing of the main functionalities of the smart contract.

I. THEORY

1 ACTUAL STATE

This section analyzes centralized and decentralized messaging applications and assesses their current state.

1.1 Centralized Messaging App

A messaging application that uses a central server is called a centralized messaging application. In centralized messaging applications, all data and communication transfers are routed through a central server managed by the messaging service provider. To perform operations, application users connect to this central server. If the central server goes down, users cannot take any action.

WhatsApp is one of the world's most widely used centralized messaging applications. Erlang is a highly regarded programming language due to its efficacy, speed, and scalability. It allows users to send text messages, make voice and video calls, and share photos, videos, and documents, among other media types. End-to-end encryption is a feature of the application, which means that only the sender and receiver can see any encrypted messages or phone calls [78]. WhatsApp requires a central server to perform its operation and it does not have open-source code.

1.2 Decentralized Messaging App

A decentralized messaging application utilizes a decentralized network instead of a centralized server. Decentralized messaging applications, it uses peer-to-peer networks or distributed ledgers to facilitate message exchange without relying on a central authority. In decentralized messaging applications, data is typically stored across the network in a distributed fashion. Instead of being stored on a single server, the data is distributed and replicated across multiple network nodes and devices.

Status is a blockchain-based chat app, cryptocurrency wallet, and Web3 browser. It is built on the Ethereum blockchain, which makes it secure to send messages. The app does not have a central authority, and end-to-end security protects users' privacy. The users are the only ones who can see the messages. The app uses the Waku peer-to-peer messaging protocol, which lets messages be sent to the whole network to be processed and encrypted. Users do not have to give Status any personal information, and the app uses a crypto wallet [77].

2 BLOCKCHAIN TECHNOLOGY

This chapter provides a comprehensive overview of blockchain technology. It defines what blockchain is and how it functions, including its decentralized and distributed characteristics. It analyzes various blockchain platforms and systems, as well as their functions. In addition, the chapter explores the advantages and disadvantages of blockchain technology, highlighting its potential to improve security, transparency, and efficiency as well as obstacles such as scalability and regulatory ambiguity.

2.1 Definition of Blockchain Technology

A blockchain is a public, immutable, and decentralized ledger that is shared across a network. It is one of the current tech trends. Blockchain utilizes an innovative method for storing and managing data via the internet and other computer nodes. It allows for the recording of transactions and tracking of assets within a business network. These assets may include money, copyrights, a vehicle, a home, or other property. This allows for an infinite number of value transfer possibilities between peers. The technologies based on blockchain are currently making a significant impact and assisting businesses in a variety of ways, including the secure interchange of business data, the tracking of consumer behavior, and the processing of faster payments, among others. The complexity of technology does not deter corporations from investing in it and investigating its capabilities [1].

Blockchain databases are decentralized public ledgers that are not subject to centralized control. This differs from conventional databases situated on a single, centralized server.

Rather than being centrally managed, these systems are sustained by a collective of users who utilize software installed on their respective personal devices. The blockchain is a decentralized public ledger that is distributed among multiple nodes as shown in

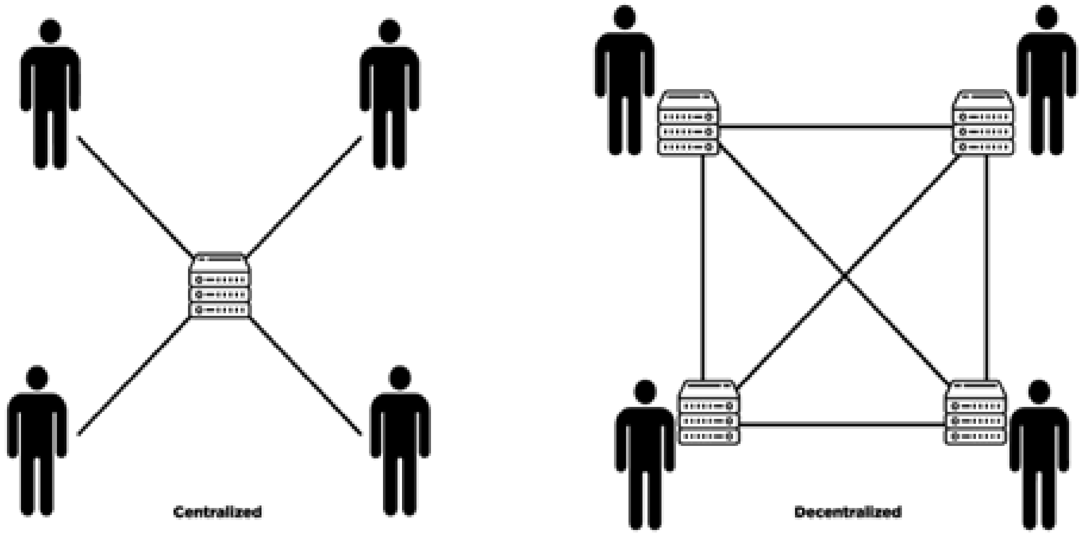


Figure 1 - Centralized Vs Decentralized [64].. Each node functions autonomously as a central server, performing actions independently. From a technical perspective, this is the fundamental nature of blockchain. In centralized networks, a single node serves as the central server and handles most of the network processing, and all the communications are mediated by a central system [2].

In the meantime, the central authority is a disadvantage of the traditional network system for example, WhatsApp and Facebook are examples of that. Having this central power has the flaw of being a single point of failure. If the central authority is compromised, the database can be compromised as well [3].

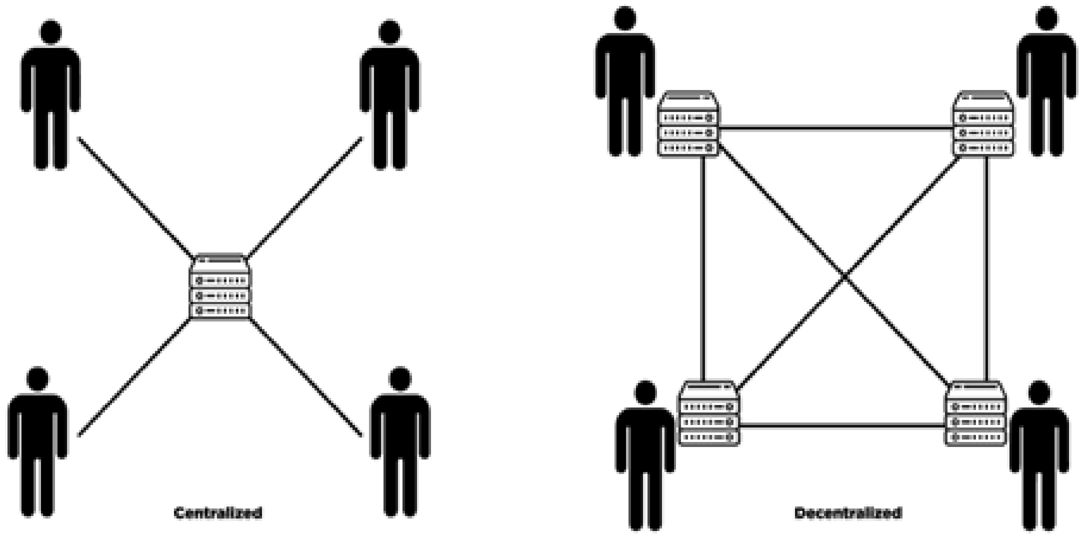


Figure 1 - Centralized Vs Decentralized [64].

1.1.1 Understanding the Inner Workings of Blockchain Technology

A new transaction joins the blockchain network with all its data, indicating the movement of assets, and it is recorded in blocks. The transaction is subsequently communicated to the network through every node, after which the nodes must validate and concur on the transaction's legitimacy. Once they agreed and the transaction was validated, the deal was finalized. After the newly created block has been assigned its hash value and verified, it is ready to be added to the blockchain as it is shown in Figure 2.

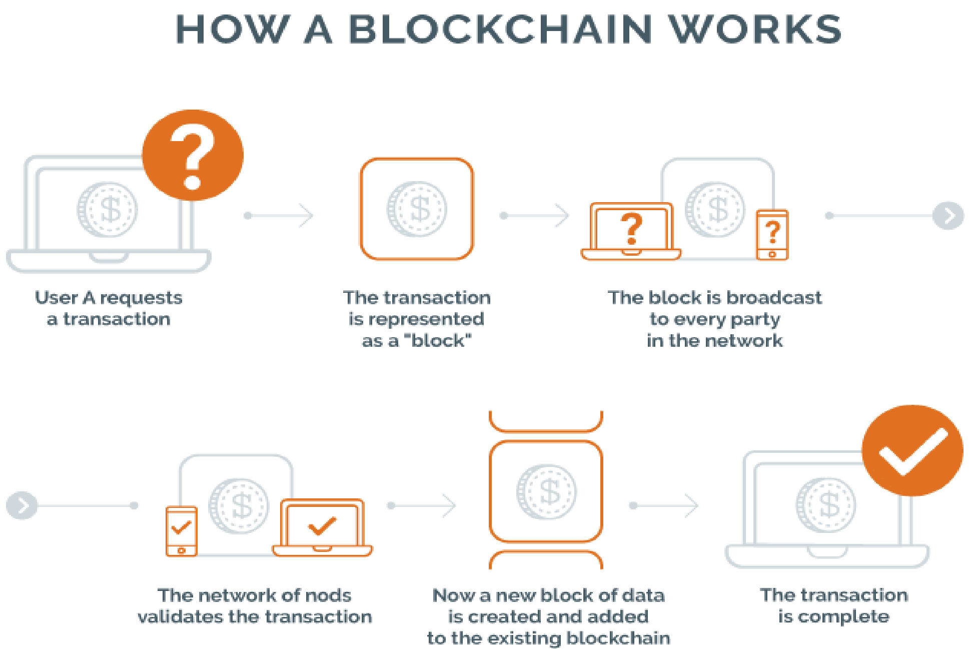

HOW A BLOCKCHAIN WORKS
User A requests a transaction
The transaction is represented as a "block"
The block is broadcast to every party in the network
The network of nods validates the transaction
Now a new block of data is created and added to the existing blockchain
The transaction is complete

Figure 2 - How a blockchain works [65].

Each successive block increases the verification of the prior block using the hash, and hence the entire blockchain will be secure. After adding the block to the blockchain. The system provides the most up-to-date copy of the central ledger to all participants, and anyone can retrieve transaction data and confirm them as [4].

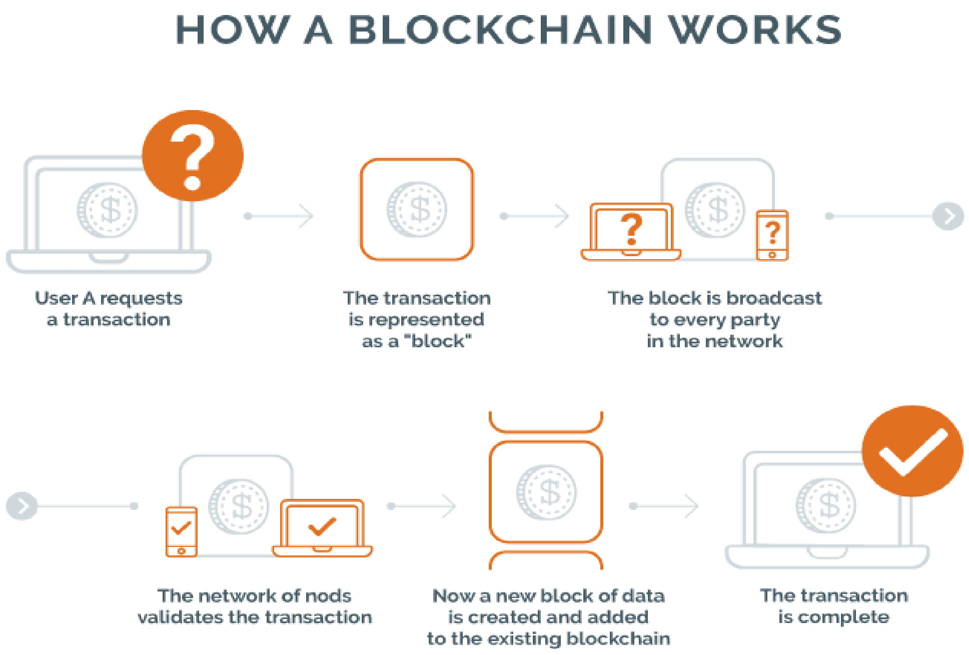

HOW A BLOCKCHAIN WORKS
User A requests a transaction
The transaction is represented as a "block"
The block is broadcast to every party in the network
The network of nods validates the transaction
Now a new block of data is created and added to the existing blockchain
The transaction is complete

Figure 2 - How a blockchain works [65].

1.1.2 Use Case of Blockchain in Real World

Blockchain technology is widely utilized across modern business sectors due to its decentralized nature, secure features, and transparency to the public, resulting in a diverse range of applications. The blockchain technology was first developed to facilitate transactions within cryptocurrencies. However, there has been a growing interest in its potential application across various industries and business domains [5].

Blockchain technology can be applied to the healthcare system as a potential use case, facilitating safe communication of patient medical records. Furthermore, it enhances the security of healthcare data and effectively oversees the supply chain of pharmaceuticals [5].

The distributed public ledger is ideal for tasks like monitoring inventory in real time as it is transferred between warehouses and retailers. It is extremely challenging to track and trace from beginning to end, especially considering the number of distinct and independent actors engaged in moving even a single cargo to its destination [6].

The real estate market, which is notoriously conservative and has significant obstacles to entry, stands to gain a great deal from blockchain technology. Blockchain technology has recently been used to remove third parties. Blockchain would hasten the sale of homes by rapidly confirming financial information, reducing fraud with its encryption, and providing full transparency throughout the transaction [6].

Blockchain will be a huge part of how identities are managed. By using blockchain to manage identity, people can take control of their identity by making a global ID that can be used for many things. Blockchain technology gives users the peace of mind that no third party will be able to access their personally identifiable information without their permission. Self-sovereign identity, data monetization, and data portability are examples of decentralized and digital identification use cases [7].

2.2 The Origin of Blockchain

Stuart Haber and W. Scott Stornetta proposed blockchain technology for the first time in 1991 [8]. They were the first to implement the idea of cryptographically protecting a chain of blocks to prevent tampering with or backdating digital data. Satoshi Nakamoto published a white paper in 2008 outlining the basic ideas behind the distributed ledger system we use today. A few months later, in 2009, the first blockchain was used as the public ledger for bitcoin transactions. He altered the Merkle Tree concept to produce a more secure system that also records the secure history of data exchange. His system uses a peer-to-peer time stamping network. Because transactions are irreversible, and Nakamoto believes that irreversibility will protect merchants from fraud, his technology has proven so valuable that it has become the foundation of cryptography. Attackers and hackers cannot access the system while most of the control is held by trustworthy participant nodes [8].

The basic idea of Bitcoin was to get rid of the necessity of relying on an intermediary entity. The verification of transactions can be performed instead. The payment system exhibits robustness due to the irreversibility of transactions, the precise tracking of transaction orders by blocks, and the decentralized nature of the computers involved. Satoshi proposed a decentralized system that utilized a peer-to-peer (P2P) network architecture.

2.3 Overview of Different Types of Blockchain

There are numerous varieties of blockchain networks, and each has its own set of advantages and disadvantages. Before selecting a blockchain type, it must be thoroughly evaluated, as each type is determined by a unique set of factors. The extent of decentralization, network size, and network security are some factors that must be considered. The most well-known blockchains are public, private, consortium, and hybrid, and they are divided into two categories [9], more detail is in Figure 3.

- Permissionless
- Permissioned

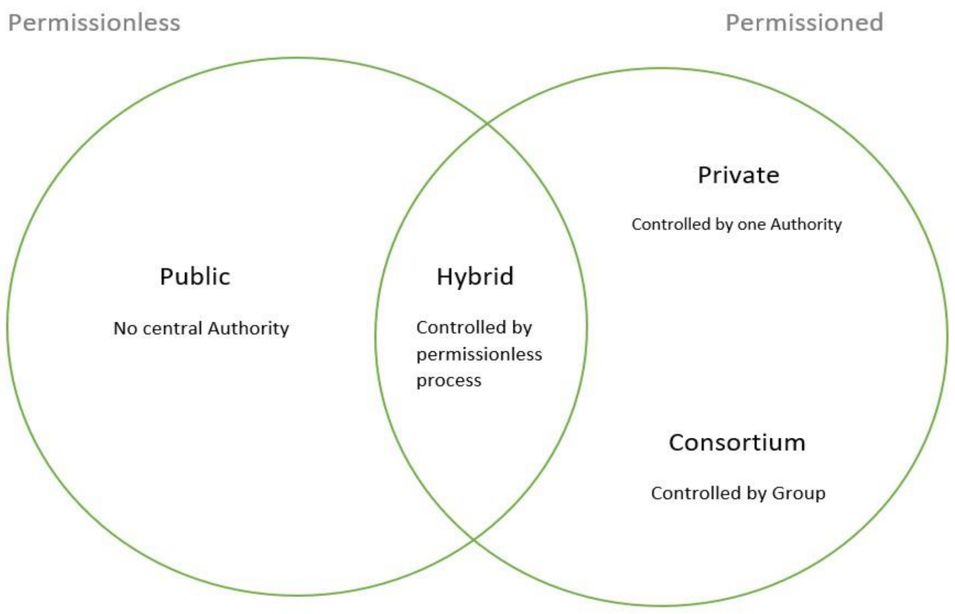


Figure 3 - Different types of blockchain [66].

The public blockchain, which is also the most accessible option, is the most well-known blockchain network among these four varieties. A Public blockchain enables every node on the internet to read, write, and join the distributed ledger system. This contributes to the attainment of autonomy. When the required transaction enters the network system, the nodes should validate it; once validated, the transaction can be sent and stored within each block without delay. In addition, public blockchains are decentralized and transparent because they lack central authority [9].

Private blockchains, also known as enterprise blockchains, are another form of blockchain. Private blockchains are maintained and accessible only to a specific organization or participant group. A private organization uses this type of blockchain for internal purposes. For instance, sharing data via a network between various organizational departments or collaborators Private blockchains differ from public blockchains in that nodes cannot join the system unless invited by the central authority [10].

Consortium blockchains, or federated blockchains, are not private or owned by a single organization because they are managed by a group of organizations. Anyone can join and participate in the network, but it is only accessible to a limited group of participants who maintain the blockchain network and oversee the agreement-making process. Financial services typically use consortium blockchains, which share both public and private functionality and features [9].

Hybrid blockchains are typically more decentralized than consortium blockchains because the public side of hybrid blockchains is typically more accessible and open to a larger group of participants. Although hybrid blockchains and consortium blockchains both combine private and public blockchain features, the private network side of a hybrid is used for storing sensitive data and regulating access to the network, while the public network side is used for verifying and providing transparency information about the network itself. A user must acquire authorization prior to joining a hybrid blockchain [11].

Table 1 - Different blockchain platforms [67].

Criteria	Ethereum	Hyperledger Fabric	R3 Corda	Ripple
Ledger type	Permissionless	Permissioned	Permissioned	Permissioned
Cryptocurrency	Ether	None	None	Ripple (XRP)
Industry focus	Cross-industry	Cross-industry	Financial service	Financial service
Data access	Public with private fork	Private	Only to relevant parties	Only to relevant parties
Smart contract	Yes	Yes	Yes	No

2.4 Advantages and Disadvantages

Blockchain's growing popularity and increasing demand can be attributed to the technology's beneficial features and functions. The structure of a blockchain network and the logic of its protocol provide a firm foundation on which to build innovative software. However, just like every other digital technology, blockchain offers both benefits and drawbacks [12].

2.4.1 Advantages of Blockchain Technology

One advantage of blockchain technology is that it facilitates decentralized trust. With blockchain technology, there is no need for an intermediary to perform transactions. Instead, data will be exchanged through a decentralized approach, with many ledgers containing identical information for each member of the network to ensure that no single point of failure is vulnerable to cyberattack. Corruption and fraud may be avoided as a result as well [13].

Transparency is a feature of blockchain, where transactions on a blockchain are recorded in a public ledger, which means that anybody can view the information. For instance, everyone can see how much money is in the wallet, but nobody can determine who the owner of the wallet is. As every transaction is recorded on the blockchain, it is much simpler to determine who started a transaction and who the recipient of the funds transferred was [14].

Traceability in blockchain could ensure that user is allowed easily to track any transaction or assets exchange, which are stored in the blockchain network. It gives the participants of the network both safety and decentralization. Unlike traditional databases, which are not transparent. In supply chain management and other scenarios where it is vital to track the transit of items from the point of origin to the destination, blockchain's usage of smart contracts can enable an unprecedented level of transparency. Using the blockchain, the complete product lifecycle may be recorded and tracked back to its very beginnings [14].

Blockchain technology is renowned for its security features, particularly in protecting private and sensitive data. By distributing data across a network of computers instead of a centralized server, blockchain can significantly transform the perception of critical information. This approach makes it more challenging for unauthorized parties to access and view the data, thereby enhancing its security. The recording of transactions occurs across multiple nodes, thereby rendering it arduous for malevolent entities to change or delete data. The distribution of information blocks throughout the network guarantees the absence of a singular point of failure and precludes the control of the system by any individual or group [15].

2.4.2 Disadvantages of Blockchain Technology

One of the most serious concerns confronting blockchain is scalability. There are some factors that impact scalability. One of them is the size of the blocks, it can be scaled. As It is known, blocks are where the data in the blockchain is stored. Each block can store some of the transactions, and each block is around 1MB in size. The number of transactions that can be processed simultaneously is affected by a block's size. As the number of transactions grows, so must the block size to accommodate the increased volume of work. However, larger blocks might result in longer transaction times and an increase in network traffic, which can cause congestion [16].

And another one is bandwidth of the network. A blockchain network with limited bandwidth may be incapable of handling a huge volume of transactions, because, when a block is mined, it is retransmitted to all nodes, resulting in delays and hefty transaction fees [16].

Data modification, in its technical nature, blockchain is an immutable database, which means the ledgers of blockchain can retain a permanent history of transactions. Data in blockchain cannot be manipulated, if any transaction is modified, it will be detected as invalid and will be canceled by the network ledgers. Another concern is once it is written it cannot be deleted by anyone [17].

Performance is another con of blockchain, because transaction on the network can take longer process, for example on traditional payment, the transaction is faster, because it takes time, whenever each transaction happens, it requires confirmation from multiple nodes in the network, therefore it will be valid to send. The block size and the condensation mechanism influence the performance of blockchain [18].

Energy consumption is needed for keeping a real-time ledger. It is one of the main problems when it comes to blockchain technology. The nature of consensus mechanism makes the blockchain use energy. For that developers need proper planning and execution to integrate into their process [18].

3 TECHNICAL SPECIFICATIONS OF BLOCKCHAIN ARCHITECTURE

Every character of blockchain technology can be identified by its constituent elements, specifically blocks, nodes, and miners. As stated, a blockchain's main aim is to facilitate the storage and dissemination of electronic data. Blockchain architecture refers to the design of a distributed ledger technology or peer-to-peer (P2P) network that is utilized for recording transactions across multiple nodes [19].

3.1 Blocks

A blockchain block refers to an established storage unit where data is stored. A blockchain is a distributed ledger technology that comprises a series of interconnected blocks, each containing discrete units of data. The concatenation of the previous block's hash to the header of the subsequent block serves as the linkage mechanism for these blocks. The preservation of block order and the immutability of data within them are key features of the system. The structure of a blockchain is characterized by its distinctiveness, comprising two fundamental components, namely the header and the body [20].

- Header: The header of a block has information about the block and the miner. It is split into two parts: the hash of the previous block and other fields. The hash of the previous block will link blocks together and make the information in the previous blocks unchangeable. The other field has the Nonce, the timestamp, the complexity, the Merkle root hash, and the height of the block. In blockchain, a nonce is a whole number that is made up at random when a block is made. This amount is then used to make a block header hash. The time stamp shows what time the block was made. The difficulty target changes how hard it is for mining. The block's height is a unique identification, and it shows the difference between the first block and the current one. It lets to know where the block is in the chain. And Merkel root hashes, which are pairs of transactions that are taken apart until only one hash is left. It acts as a summary or fingerprint of all the transactions in block [20].
- Body: This section contains all the information recorded in the block, such as a list of transactions. Transaction data may comprise sender, recipient, and amount of cryptocurrency transmitted information [20].

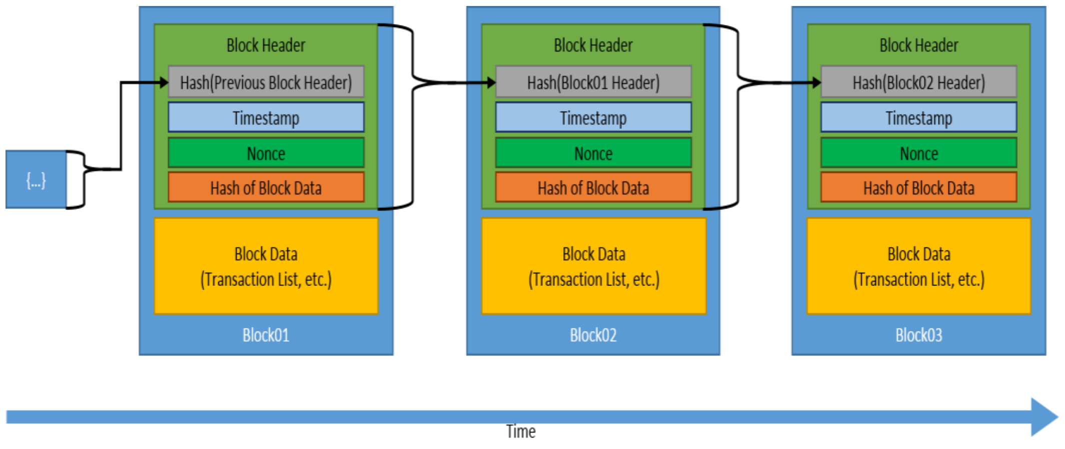


Figure 4 - Structure of blocks [68].

After transactions are validated and grouped together, a nonce generates the cryptographic hash, which is put to the block header, and a new block is produced on a blockchain. After the first block is created, it is linked to the preceding block. As a result of connecting each block to the previous blocks, a chain of blocks is formed [21].

3.2 Nodes

Nodes are one of the most significant elements in blockchain; they are responsible for building the infrastructure of a decentralized network and acting as connection centers for different network operations. Their major purpose is to operate the blockchain protocol's program, which allows it to help validate transactions and secure the network. After a transaction is approved, a new block is added to the blockchain in a process known as mining. The Bitcoin blockchain utilizes a PoW consensus mechanism, which involves miners who oversee the process, although not every blockchain uses mining or has miners. Various consensus systems employ various methods to determine who validates transactions [22].

A blockchain node's primary functions are to publish and validate transactions. A node receives a transaction sent by a user and broadcasts it to the remainder of the network. The transaction is checked by all network nodes to ensure the sender has the funds available and can transmit them. After the nodes validate new transactions, they are bundled into blocks. Each new block is added to the blockchain in accordance with the rules of its consensus mechanism, which are enforced by a limited group of nodes known as full nodes [23].

3.3 Transaction Workflow

A transaction is defined as a contract, arrangement, transaction, or exchange of goods or services between more than one party. Typically, the asset is cash or property. Similarly, a blockchain transaction is nothing more than data transmission through a blockchain system's network of computers. To transact on the blockchain, one needs a wallet, which is a program linked to the blockchain that grants exclusive access to the user. It maintains track of the cryptocurrency held and facilitates trading with it. Each wallet is protected by a unique cryptographic mechanism that employs a distinct but linked pair of keys: a private and a public key. A user whoever owns the private key can utilize their wallet to approve or sign operations, transferring value to a new owner [24].

It is needed to inform the payee that the previous owners did not sign any previous transactions. There is no point worrying about further attempts to double-spend because the first transaction is what matters. To establish the absence of a transaction, the user must be aware of all transactions. The mint was aware of all transactions and decided which arrived first in the mint-based paradigm. To accomplish this without the help of a trusted third, transactions must be made public, and we need a mechanism that allows participants to agree on a single history of the order in which they were received. The payer must demonstrate that at the time of each transaction, most nodes agreed it was the first received [25].

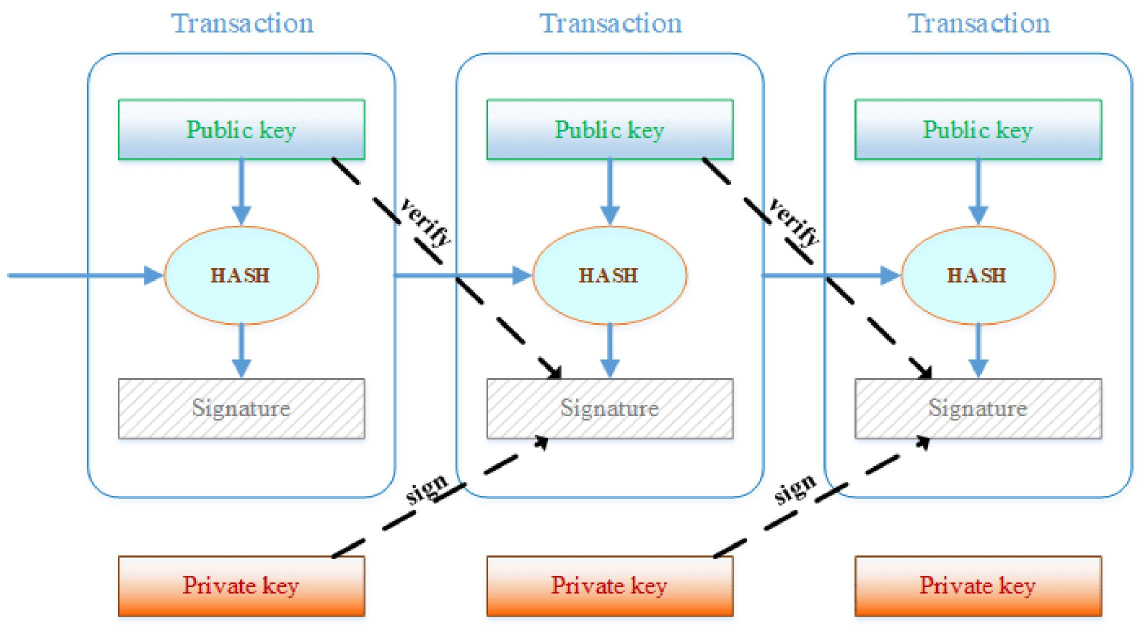


Figure 5 - Transaction workflow [69].

3.4 Timestamp Server

The hash of a block of items is time stamped by the Timestamp server. It will widely disseminate the hash. The timestamp is a method of verifying that the data existed at the time. The timestamp proves that the data had to exist at the time to be included in the hash. Each timestamp hash includes the previous timestamp. This will result in the formation of a chain. The next timestamp will reinforce the previous ones [26], as it is shown in Figure 6.

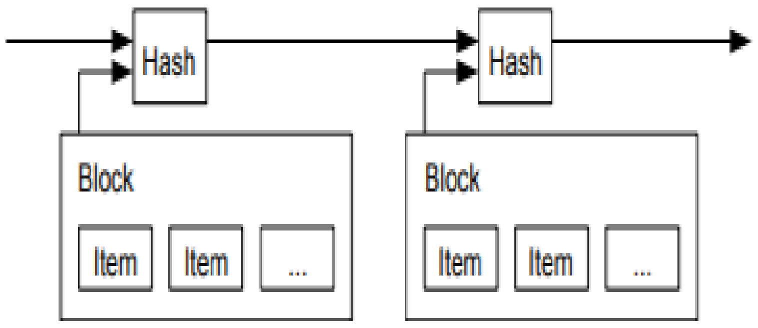


Figure 6 - Timestamping each block in its hash [70].

3.5 Consensus Mechanism

Consensus mechanisms are the protocols, algorithms, or other computer systems that establish transaction legitimacy and blockchain governance. Consensus refers to a universal agreement reached by all network participants. Consider a bunch of individuals heading to the movies. A consensus is reached when there is no dispute over a proposed film choice. Whenever there is dispute, the group must be able to choose which film to watch. In extreme instances, the group will break up. Consensus mechanisms are the entire protocol stack that allows any approach to attain trust and security across a decentralized computer network [27].

The consensus algorithms utilized by various blockchain platforms are mostly motivated by the sort of applications the platform hopes to offer and the threats to the chain's integrity. Proof-of-work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS), Proof-of-Capacity (PoC), Proof of Authority (PoA), and Practical Byzantine Fault Tolerance are some of the consensus techniques and protocols often utilized in blockchain networks PBFT (Practical Byzantine Fault Tolerance). These are just a few of the various consensus methods and protocols utilized in blockchain technology. Several factors are evaluated while selecting the best consensus model for a certain network, including the intended network, participant relationships, and both functional and non-functional elements. Each mechanism has advantages and disadvantages, and the choice of mechanism is frequently determined by the blockchain network's specific demands and aims [28].

3.5.1 Proof of Work

PoW is the Bitcoin blockchain's consensus protocol, and it was also employed by the Ethereum blockchain. is the mechanism that once allowed the decentralized Ethereum network to reach agreement on topics like account balances and transaction order [28].

To add a new block, participants must solve computationally challenging "cryptographic puzzles" in a process called mining. As the number of participants grows, the calculation becomes much more sophisticated and necessitates more powerful gear. As a result, electricity consumption is high. As in the case of Bitcoin, the miner node gets rewarded with a set quantity of Bitcoin [29].

3.5.2 Proof of Stake

Some consensus procedures employed by blockchains to reach distributed consensus are based on proof-of-stake. Miners use proof-of-work to demonstrate that they have capital at stake by consuming energy. Ethereum employs proof-of-stake, in which validators expressly stake capital in the form of ETH into an Ethereum smart contract. This staked ETH is subsequently used as collateral, which can be destroyed if the validator is dishonest or lazy. The validator is thus in charge of ensuring that new blocks propagated across the network are valid, as well as periodically producing and propagating new blocks. PoS enables faster transactions because blocks are approved faster without the need to solve complex mathematical equations. There is greater scalability because no physical devices or mining farms needing enormous amounts of energy are required to generate consensus [30].

Proof of stake is the most used substitute for proof of work. Instead of investing in processing equipment and energy, validators (stakeholders) invest in currencies. On day one, all the system's currencies are there, and no mining is necessary. Yet, as compared to PoW, it may give greater scalability and energy efficiency. The Proof-of-Stake algorithm's popularity has increased lately. This is reinforced by the fact that Ethereum, one of the most popular blockchain systems, is migrating from the Proof-of-Work algorithm to the Proof-of-Stake algorithm [31].

3.6 Cryptography

Cryptography is a method for protecting data from unauthorized access. In blockchain, cryptography plays a significant role. Blockchain, as a decentralized and transparent ledger, relies on cryptographic algorithms to secure transactions, verify data integrity, and provide privacy to its participants. It helps secure various transactions in the blockchain network. It makes sure that the only people who can access, read, and process transactions are the intended recipients of the transaction data. Cryptography is employed to encrypt messages in the P2P network, while hashing is utilized to fortify block data and interconnect blocks within the blockchain. The two main concepts behind cryptography are encryption and decryption.

Types of Cryptography:

- **Symmetric-key** cryptography is a cryptographic technique that relies on the same key for both encryption and decryption processes. Importantly, the symmetric key encryption technique is likewise suitable for protecting website connections or data encryption. It facilitates a secure mode of communication between users on the blockchain, enabling them to exchange information while maintaining the privacy of their private keys.
- **Asymmetric-key** encryption and decryption employ separate keys. This encryption employs public and private keys. The private key decrypts and verifies digital signatures. The hashes are utilized to authenticate the integrity of blocks and transactions, thereby guaranteeing that data has not been exposed to unauthorized alterations [74].

3.6.1 Public Key

Public keys are accessible to all individuals and enable communication and the reception of transactions. It serves as a unique identifier for each network user and is used to validate their identity on the blockchain. The process of generating a public key from a private key is straightforward, whereas the inverse operation is more intricate. The process of inferring the private key from the public key is a challenging task. Upon transmission, an electronic communication incorporates its public key and is authenticated via a private key. The address, which is a condensed form of the public key, is commonly utilized as the recipient's public key for transactional purposes. Consequently, it is possible to share the public key without any concern [32].

3.6.2 Private Key

The private key enables a user to assert ownership over the assets at a particular address. It is an encrypted cryptographic key that is generated alongside the wallet's public address and instantly retained by the wallet. When sending from a wallet, the program marks the transaction with the private key (without showing it), indicating to the network that the user are authorized to transmit the assets to the receiving address. It is also used to generate digital signatures that validate the authenticity of the user's transmission [33].

4 SMART CONTRACT

This chapter provides a detailed introduction to smart contracts, including smart contract definition and historical context. It covers the mechanisms of blockchain transactions, the interaction between decentralized chat applications and smart contracts, the execution of smart contracts, and the concept and calculation of gas fees. Furthermore, the chapter explores the actual benefits and real-world applications of blockchain technology, highlighting its potential global benefits.

4.1 Definition and Origin

The term "smart contract" appears to have originated with the introduction of blockchain, yet the concept of smart contracts existed long before the first blockchain. Nick Szabo first presented it in a study in the 1990s. A smart contract, according to him, is a computerized transaction mechanism for carrying out contractual conditions and establishing secure connections through computer networks [34].

A smart contract is a piece of code or an automated mechanism that manages a transaction between two parties without involving a third party and is stored on a blockchain. Once the company's smart contract is deployed on the blockchain network, it becomes independent of its developers and cannot be controlled by any other entity [35].

4.2 Understanding the Execution of Smart Contracts

A smart contract is a piece of code or a program that operates on the blockchain network. When the smart contract is deployed on a blockchain, it becomes a permanent part of the network. When many parties specify their agreement and desired outcome, the contract code provides the rules and conditions that must be followed for the contract to be carried out automatically and without the need for a third party. Then when the conditional parameters are met, a computer program is built that will execute automatically. Following that, anytime the parties achieve an agreement on authentication and verification, the code is executed, and the results are documented for compliance and verification. Eventually, all network nodes update their ledgers to reflect the new state. Once an item has been published and verified on a blockchain network, it cannot be modified; only append mode exists [36].

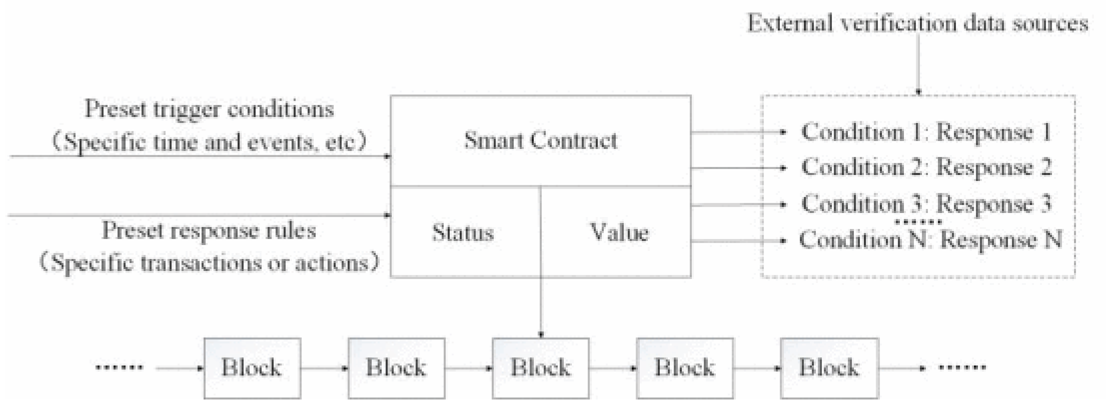

External verification data sources
Preset trigger conditions (Specific time and events, etc)
Preset response rules (Specific transactions or actions)
Smart Contract
Status
Value
Condition 1: Response 1
Condition 2: Response 2
Condition 3: Response 3
Condition N: Response N
Block

Figure 7 - Operating mechanism of smart contract [71].

4.3 Decentralized applications (DApps) and smart contracts

DApps are blockchain-based applications. The blockchain allows apps to run without the need for a central location. Smart contracts are required to introduce dApps. They are built with smart contracts and run on a decentralized peer-to-peer network. dApps must be carefully designed and rigorously tested because they are regulated by the logic expressed in the contract [37].

Here are some pros of developing dApps [38].

- Immutability of data
- Zero downtime
- Anonymity
- No censors
- Trustless computation

Decentralized applications are client-side user interfaces that let users engage with a blockchain smart contract. Creating a user interface is a necessary step in the development of a decentralized application. Implementing a smart contract with a high-level programming language; compiling the smart contract with a language-specific compiler to generate a contract binary; deploying the generated contract binary code on the blockchain with any blockchain platform based on the platform for which the contract was written;

and obtaining the address of the smart contract that will be used by the web application to interact with the smart contract on the blockchain [39].

4.4 Programming Languages for Smart Contracts

Programming languages play a particularly important part in the creation of smart contracts. To construct and manage this system, specialized programming languages that can communicate with blockchain technology must exist. There are a variety of programming languages that can be used to create smart contracts, each with its unique capabilities and characteristics.

- Solidty: It is an object-oriented, high-level programming language used to create smart contracts that implement business logic and generate a chain of blockchain transaction information. Smart contracts are computer programs that manage the behavior of Ethereum wallets. Solidity is a curly-bracket programming language that serves as a compiler for machine-level code on the Ethereum Virtual Machine (EVM). It is related to Python, JavaScript, and C++ and is straightforward to learn and comprehend [40].
- Rust: It is a prominent programming language in the blockchain and smart contract development fields. Rust is utilized in programming for three critical purposes: performance, safety, and memory management. Rust has emerged as a popular programming language for creating decentralized applications, smart contracts, and blockchain protocols. The blockchains that use Rust as a programming language for writing smart contracts are Solana, Hyperledger, Elrond, Polkadot, and Near Protocol [41].

4.5 Smart Contract Execution and Gas Fees

The term gas fee refers to the quantity of money certain blockchain protocol users pay to network validators whenever they wish to execute a function on the blockchain. To run on a blockchain network, a smart contract needs a certain number of processing resources. Gas is the unit of computing labor needed to run a smart contract. These computational resources are measured in gas. Gas fees are important because they move money from the people who need a blockchain network service to the people who provide the computing power needed to run it. Gas costs motivate validators to execute transactions correctly and keep the blockchain ledger secure. End users pay gas fees to miners on proof-of-work (PoW) blockchains such as Ethereum, but in Ethereum 2.0, which uses proof-of-stake, end users pay no gas fees (PoS). Gas fees are paid to validators who first commit a particular amount of cryptocurrency to the network to be chosen to verify new transactions on these blockchains [42].

4.6 Benefits and Use Case

Since smart contracts are built on blockchain, they ensure the immutability of data, enabling contracts and agreements to be created without having to know each other and preventing potential conditions violations or errors in the management and execution of the contract [43].

Some benefits of smart contract are:

- Security
- Disintermediation
- Near real-time execution
- Transparency

There are various sectors in which widespread application of intelligent contracts would have the biggest benefits. For instance, smart contracts are primarily utilized in the finance industry. The financial sector is continually searching for technology that can deliver a transparent and trustworthy system [44].

Use cases of smart contracts in real world industries [45]:

- Education
- Insurance
- Warehouse
- Supply Chain
- Transport & Logistics
- Charities
- Government systems

5 POLYGON BLOCKCHAIN

Polygon, which used to be termed Matic network, is a blockchain platform that intends on developing a multi-chain blockchain system as an Ethereum Layer 2 scaling option. MATIC is the name of Polygon's own currency. Polygon uses a proof-of-stake consensus method to produce new MATIC and keep the network safe. This means that one way to generate income from owned MATICs is by staking them. Polygon blockchain lets developers make their own blockchain networks with their own rules and settings that can be changed for different use cases [46].

Polygon is a scaling option for a sidechain that works with the Ethereum blockchain. Its goal is to help Ethereum scale up and make trade faster and cheaper.

5.1 A Comparative Analysis of Ethereum and Polygon

Ethereum and Polygon are both blockchain platforms that let developers create dApps. But there are some significant distinctions between the two blockchains, basic differentiation of both is shown in Table 2.

Ethereum is widely recognized as the pioneering blockchain platform that facilitated the implementation of smart contracts. Blockchains experience a slow and excessive cost during periods of high traffic. Ethereum serves as a prime illustration of how blockchain systems that prioritize autonomy and security measures may encounter limitations in scaling to accommodate real-world usage. The platform boasts a substantial and firmly established group of developers, resulting in the growth of numerous widely used dApps [47].

Polygon is a recently developed blockchain that aims to address the scaling issues encountered by Ethereum. As a layer 2 solution, it was constructed on the Ethereum platform. The implementation of this technology enhances the efficiency and cost-effectiveness of Ethereum; however, it has a broader scope beyond its function as a mere scaling solution. Over time, Polygon has expanded its developer community and garnered several popular decentralized applications. The scaling solution for Ethereum applications has gained significant popularity, providing blockchain developers with a plethora of valuable tools and software development kits [47].

Table 2 - Difference between Ethereum and polygon [48].

Criteria	Ethereum	Polygon
Native Token	ETH	MATIC
Year of Foundation	2013	2017
Transaction Speed	27-30	65,000
Consensus Mechanism	Proof of Stake	Proof of Stake Plasma-based sidechain
Architecture	Stateful architecture	Multichain architecture
Scalability	Limited Scalability	Better scalability
Transaction Fees and Gas Prices	High	Low
Smart Contract Languages	Solidity	Golang, Solidity, Vyper
Developer Community	Highest number of developers	Comparatively lower

Polygon and Ethereum are two of the most well-known Proof-of-Stake networks. On-chain users can deal with DeFi and develop dApps on these networks.

5.2 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is a software system that runs smart contracts, calculates the state of the Ethereum network after each new block is added to the chain, and compiles various kinds of smart contract code into an understandable format called Bytecode. The client program required to run a node on Ethereum includes the built-in Ethereum virtual machine. Nodes on Ethereum keep copies of transaction data, which the EVM uses to update the global ledger. Smart contracts are programs that run on their own and automatically execute the conditions of a contract among multiple parties. The EVM sits on top of Ethereum's hardware and node network layer, which is where smart contracts are run. Its major job is to figure out the state of the network and run these contracts, making sure they are safe and predictable. Figure 8 shows the architecture of EVM.

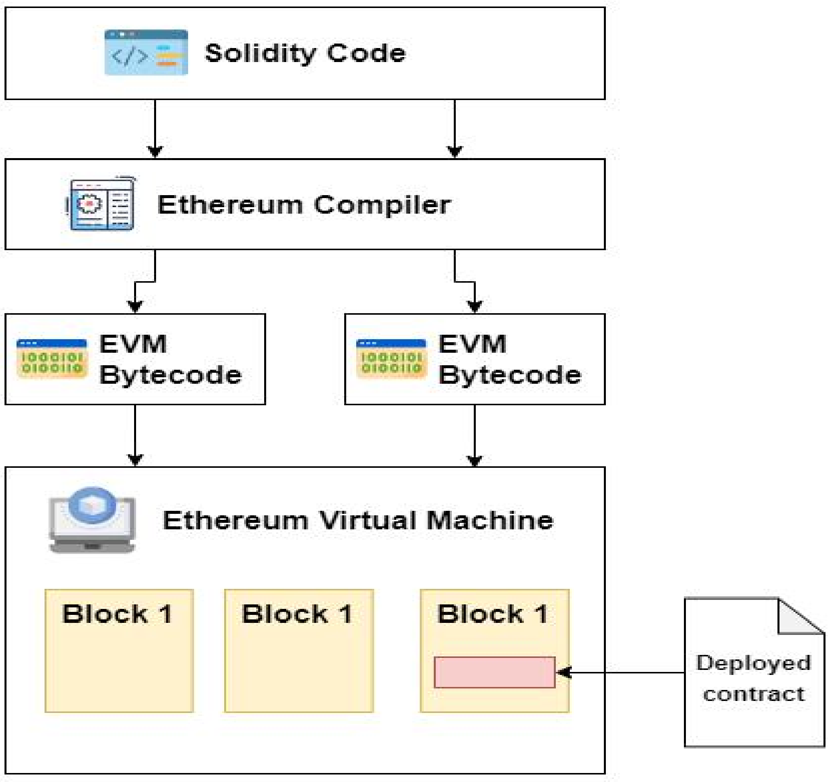


Figure 8 - EVM architecture [72].

5.3 Transaction Process

Polygon becomes available to all nodes within the network. Subsequently, the authenticated entities proceed to verify the signature and nonce of the transaction, along with the sender's account balance, and ascertain whether the transaction adheres to the regulations and prerequisites of the network, which encompass the minimum transaction fee. Upon validation, the transaction is appended to the subsequent block of the blockchain, and the account balance of the sender is duly modified. The estimated rate of polygon transactions per second is 65,000, which is a remarkable figure [50].

5.4 The Impact of Gas on Payment Transactions

Since the Polygon network is a decentralized blockchain and is not run by a single body or authority, the Matic tokens operate as native tokens on the Polygon chain. To use Polygon, the user first needs to buy some MATIC tokens, which can do on many of the most cryptocurrency trading platforms. Therefore, a person can use MATIC to pay for gas and send MATIC to other accounts, just like they can use ETH to pay for gas on Ethereum and send ETH to other accounts [51].

Fees are charged for any programmable work in Polygon or any other blockchain. The list of fees is written in terms of units of gas. So, any part of a programmable computation, like "deploying a contract," "sending a message," "adding a contact," or "getting to the account storage," needs gas fees to be carried out. How much MATIC is needed for a transaction depends on how complicated it is and how busy the network is at the time. Most of the time, Polygon's gas costs are less than Ethereum's. To determine the gas fee, it is necessary to multiply the gas price by the quantity of gas that can be obtained. The gas price is how much a user is willing to pay for one unit of gas for a transaction. The gas limit is how much gas a user is willing to use for the transaction. Formula: Gas Fee = Gas Price x Gas Limit [52].

6 TECHNOLOGIES FOR IMPLEMENTING ON BLOCKCHAIN

This chapter provides a comprehensive analysis of the technologies, libraries, and dependencies used to develop the messaging application. It explains the architectural composition of a smart contract written in Solidity. It emphasizes the benefits of these technologies for the development of decentralized applications.

6.1 Structure of Smart contract

Smart contracts are an innovative way of creating software that differs from the standard method. We use programming languages to create smart contracts. Some of these were created specifically for smart contract writing, while others were derived from extant languages. Most smart contracts are written in the Solidity programming language, but it is also possible to use Vyper, Rust, Yul, C++, and JavaScript [53].

The following are some fundamental concepts that characterize smart contracts, as well as the fundamental structure of the Solidity programming language [54].

- Pragma will tell the compiler which version of Solidity to use to build the contract. The Pragma alignment is at the beginning of the source file and turns the source code into bytecode.
- A contract is the top level of the code and is like a class. A contract consists of state variables, functions, function modifiers, events, structures, and enums.
- Data storage is represented by state variables and is referred to as storage. Depending on the situation, data values can be saved as storage, memory, or call data. When compiling, the contract can keep track of how much storage on the blockchain it requires.
- Functions are the codes that conduct the contract's actions. Functions can be invoked as many times as needed. In reaction to incoming transactions, functions can obtain or set information.
- Construct functions are only run once the contract is deployed for the first time. Function modifiers
- Events are like logging in. When a contract is signed, it is sent out. Smart contracts can send out a signal when a deal is mined.

- View functions are a function which only reads data, but it does not change the state variable.

Solidity is designed to work with the Ethereum virtual machine (EVM), and it is typed statically. Solidity can be made on a Mac, a Windows computer, or a Linux computer. It has a lot of plugins and tools for IDEs and editors, with VSCode being the most well-known.

6.2 React

React is a widely used JavaScript library. It is an open-source JavaScript library designed to facilitate the rapid development of user interfaces or UI components with significantly fewer lines of code compared to using plain JavaScript. The company that made it was Facebook. A lot of front-end and full-stack workers need to know how to work with React. It makes parts that can be used more than once, and these parts show data as it changes over time. These components are separate parts of the final interface that, when put together, make up the whole user interface for the program. When a person wants to get a page, they type a URL into their browser, which then sends a request to the page's server. The computer sends the page back to the browser, which displays it. If a user wants to go to a different page on the same website, the browser makes a new request for the page the user wants [55].

React.js is important for Web3 developers because it lets them make interfaces for decentralized apps that are fast, scalable, and easy to use. It has a wide range of tools and resources that make development easier, and its flexible design makes it easy to combine it with other frameworks. Overall, React.js is a key tool for building Web3 applications.

6.3 Node package manager

The Node package manager (npm) is the biggest program library in the world. It is both a library and a registry for software packages that use JavaScript. NPM also has tools for the command line that help to install packages and control how they depend on each other. The Node package manager is open source, which means that writers can use and share code with each other. It is a method based on modules. A package is a folder that contains one or more files and a file called "package.json" that has information about the package. JSON must be used to write the text of package.json [56].

With NPM, Web 3 developers can easily install packages with just one command. They also have access to a large ecosystem of open-source packages and can handle dependencies for smart contract development on blockchain platforms.

6.4 Ethereum.js

Ethers.js is a JavaScript library used to interact with the Ethereum blockchain, and it is the blockchain that is a side chain on the Ethereum network, such as Polygon. It is commonly used to build dApps, deploy smart contracts, and create basic scripts that require reading and writing to the blockchain. Web3.js is the library upon which Ether.js is built [57].

6.5 Web3.js

Web3.js makes up a collection of JavaScript libraries that enable the front-end application to set up communication with the blockchain, obtain information from smart contracts, and engage with dApps through HTTP, IPC, or WebSocket. Web3 refers to an open-source standard initiative that eases the interaction between client-side applications and the Ethereum network. Web3.eth is a crucial instrument that helps direct user engagement with the blockchain [58]. Web3.js is an indispensable tool for developers who are creating Ethereum-based decentralized applications. Web3 is used to:

- Creating and managing wallets
- Sending and receiving Ethereum based token or ether.
- Querying data from the Ethereum network, such as account balances and transaction histories

- Listening for and reacting to blockchain events in real-time
- Signing and verifying messages and transactions using public-key cryptography
- Integrating Ethereum-based functionality into web and mobile applications.

6.6 ABI

The Contract ABI (Application Binary Interface) is a JSON-based format that supplies a means of communication between software components. It is a widely accepted industry standard. In the environment of Web 3.0, ABI plays a crucial role in easing the interaction between contracts in the Ethereum ecosystem beyond the blockchain. This enables seamless communication between decentralized applications, user interfaces, and contracts, allowing them to exchange information without any hindrances. The nature of the data employed for compression. This serves as an intermediary connecting the blockchain layer and the application layer [59].

6.7 Hardhat

The Hardhat platform is a software development environment designed for Ethereum that eases the development, testing, and execution of smart contracts by blockchain developers with greater efficiency. The Hardhat platform is open-source software that offers support for multiple programming languages such as Solidity, Vyper, and various others [60].

Prior to their deployment on the Ethereum network, smart contracts are documented in written form. A computer can interpret bytecode, which is the low-level representation of a contract, by converting Solidity code into it. After the process of activating the contract, the bytecode is transmitted to the blockchain [61]. It is imperative to acknowledge that developers commonly use a compiler to compile it. The Hardhat tool is a practical possibility for developers looking to compile and deploy Solidity code.

Here are some uses of Hardhat:

- Local blockchain network
- Contract testing
- Debugging
- Integration with popular tools
- Support for multiple languages

6.8 MetaMask

MetaMask is a well-known cryptocurrency wallet that includes a key vault, secure login, token wallet, and asset exchange. MetaMask is available as a browser extension as shown in Figure 9 and a mobile app for connecting to the blockchain with a personal account and interacting with smart contracts. It is used to confirm who owns a blockchain address that can receive and send messages. It creates passwords and keys on the device, allowing users to access only their accounts and data. User can decide what to share and what want to keep secret [62].

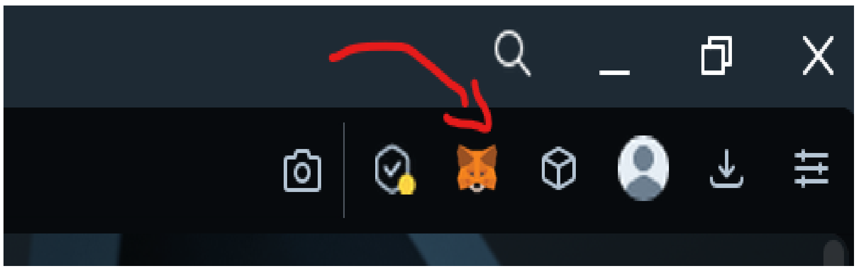


Figure 9 - MetaMask extension in Opera crypto browser.

One of MetaMask's primary features is its capacity to serve as a link between the standard internet and the decentralized web, as it illustrated in Figure 10. With MetaMask, users may use their ordinary web browsers to access dApps and engage with smart contracts without the need for specialist software or hardware.

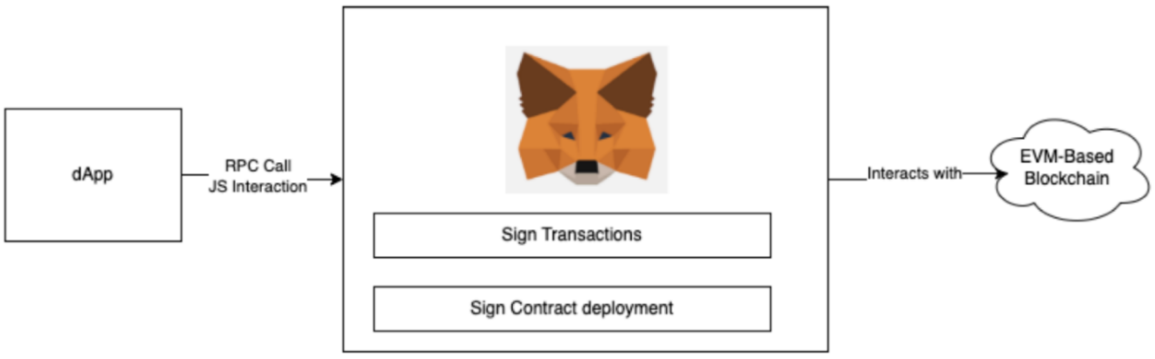


Figure 10 - MetaMask interaction [81].

6.9 Polygon Mumbai test network

The Polygon network's test net is called the Polygon Mumbai test net. It is a fully working network that looks like the main Polygon network but uses fake coins that have no value in the real world. It gives developers a lab where they can test their apps on the Polygon network. This makes it easy for them to see how dApps would work in the real world. The PoA consensus method is used on the Mumbai Test net, which is a test network that works with Ethereum. This is fast and efficient for testing. The Polygon Mumbai test net has a chain ID of 80001 [63].

Overall, the Polygon Mumbai test net is a vital resource for developers creating on the Polygon network since it provides a secure and controlled environment in which to test their applications before making them available to the public.

6.10 Alchemy

Alchemy is a platform and API for the development of dApps. It provides numerous management tools for developing decentralized applications on various blockchains, including Ethereum, Polygon, and Solana. It gives the applications scalability, dependability, and data accuracy. Alchemy's primary purpose is to assist developers in building, monitoring, and testing their applications [75].

Key offers:

- API Access
- Developer Tools
- Infrastructure Management
- Analytics and Monitoring
- Scalability Solutions

6.11 Mocha and Chai

Mocha is a JavaScript testing framework that is open-source and used for executing tests on diverse web applications, backends, and smart contracts. Mocha and Chai are often used alongside each other to conduct unit testing. Chai is an assertion library that provides a fluent and expressive syntax for making assertions, allowing developers to write clear and concise tests [76].

6.12 CryptoJS

CryptoJS is a JavaScript library that provides secure cryptographic algorithms. It helps developers mask user data by translating it into meaningless information. It supports a wide variety of cryptographic algorithms, including symmetric-key algorithms and hash functions. CryptoJS helps developers execute diverse operations, including encryption, decryption, hashing, and message authentication [79].

II. PRACTICAL

7 DEVELOPMENT AND IMPLEMENTATION

This chapter provides an in-depth analysis of the business model employed by a messaging application that utilizes blockchain technologies. It discusses the key features of the application and outlines the user journey flow, highlighting the interaction between the user interface and smart contracts. Additionally, it explores the process of deploying smart contracts and guides users on setting up the Metamask wallet. The chapter also includes a graphical representation of the application's user interface and concludes with a discussion on the future development prospects of the application. The messaging application is publicly accessible on GitHub at https://github.com/hamakhalid99-erbil/bloxiety.git.

7.1 Business Model

The business model of this application is to provide a messaging app that is built on blockchain technologies. The app has been developed on the base of last chapter’s technologies and dependencies. The messaging app consists of a Smart Contract and a web interface. The smart contract is where the data is stored. The application is decentralized and can work without any centralized server, because it is dependent on Polygon network and the smart contract deployed there.

To register and contact other users, the user must share their public key, but they must keep their private key private. All the transactions are available on the blockchain, and the messages are encrypted and only the receiver can decrypt.

Primary features of the messaging app:

- Users can join the chat app by creating an account on the blockchain with the use of MetaMask wallet, which they can access with their private key. This operation involves the payment of a transaction fee.
- Users can add friends to their accounts after successfully registering by entering the registered user’s public key. This operation involves the payment of a transaction fee.
- They can send messages to each other once the friend is added to the account. This operation involves the payment of a transaction fee.
- Receiving the messages. This process does not involve the payment of a transaction fee.

Figure 11, illustrates the journey of user, from creating an account to adding friend and lastly sending a message to the friend.

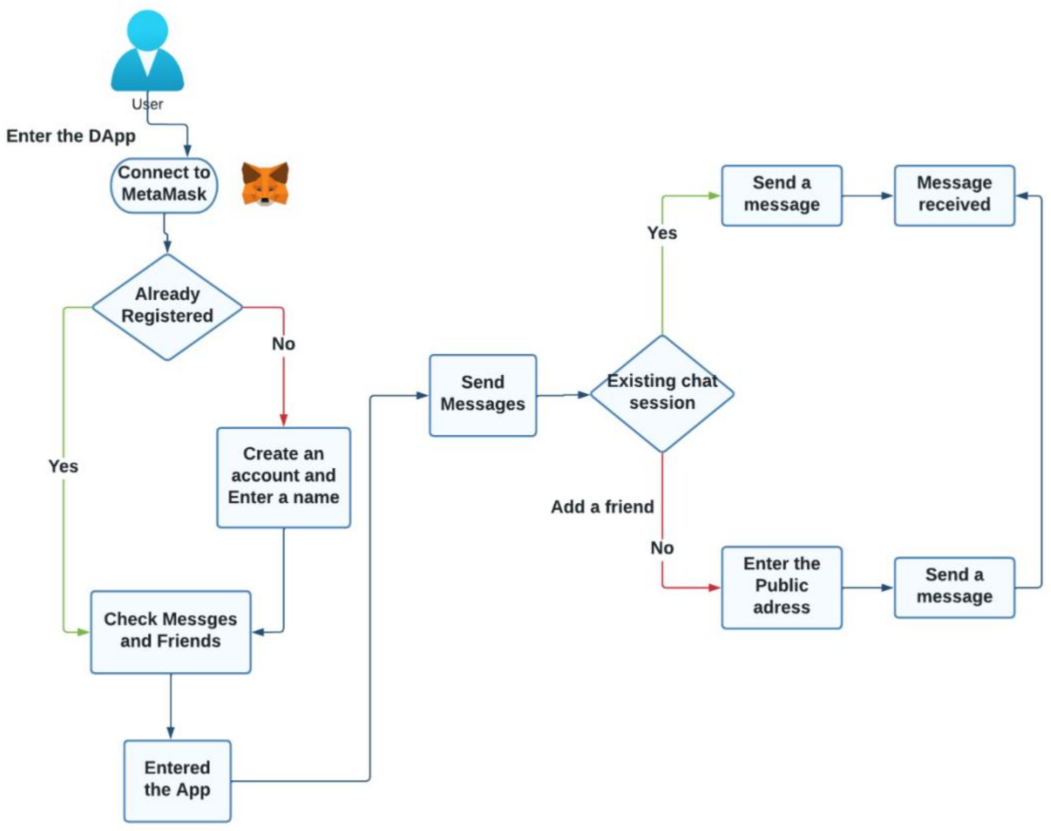
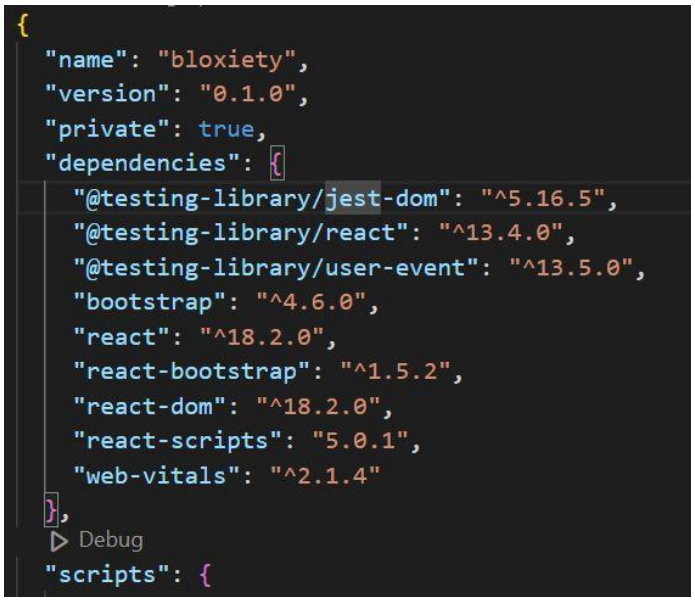


Figure 11 - User Journey Flow.

7.2 JSON File

The `*package.json*` file lists the project's dependencies. It includes the name, the version, the script, the test, and the dependencies. The process of developing a package.json is commonly the initial stage of a node project and is necessary for the installation of dependencies in npm. JSON is a basis for JavaScript and Node applications. Node.js can be downloaded from nodejs.org; select the appropriate package for the operating system. The `*npm init*` command is utilized for the purpose of configuring a new or existing npm packages [80]. Using npm also helps to install all the necessary React packages and modules; it can quickly create a React app and setup the environment by writing `*npm install -g create-react-app*` and `*create-react-app bloxeity*` in the terminal.



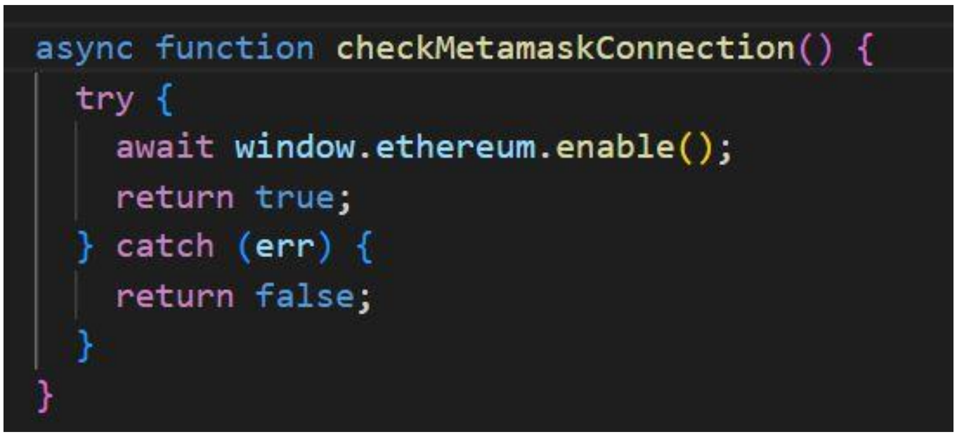
```
{
  "name": "bloxiety",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^4.6.0",
    "react": "^18.2.0",
    "react-bootstrap": "^1.5.2",
    "react-dom": "^18.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  ▷ Debug
  "scripts": {
```

Figure 12 - Structure of *package.json*.

7.3 Interacting with the Smart Contract

For interacting with smart contracts, web3.js has been used. The first step is to install the web3 library. The initialization of the contract address and ABI in React code is required to get the functionality of the smart contract. The ABI is used for defining the methods and arguments of the smart contract. Then web3.js converts high-level function calls into low-level ABI calls, which are subsequently transmitted to the Polygon network.

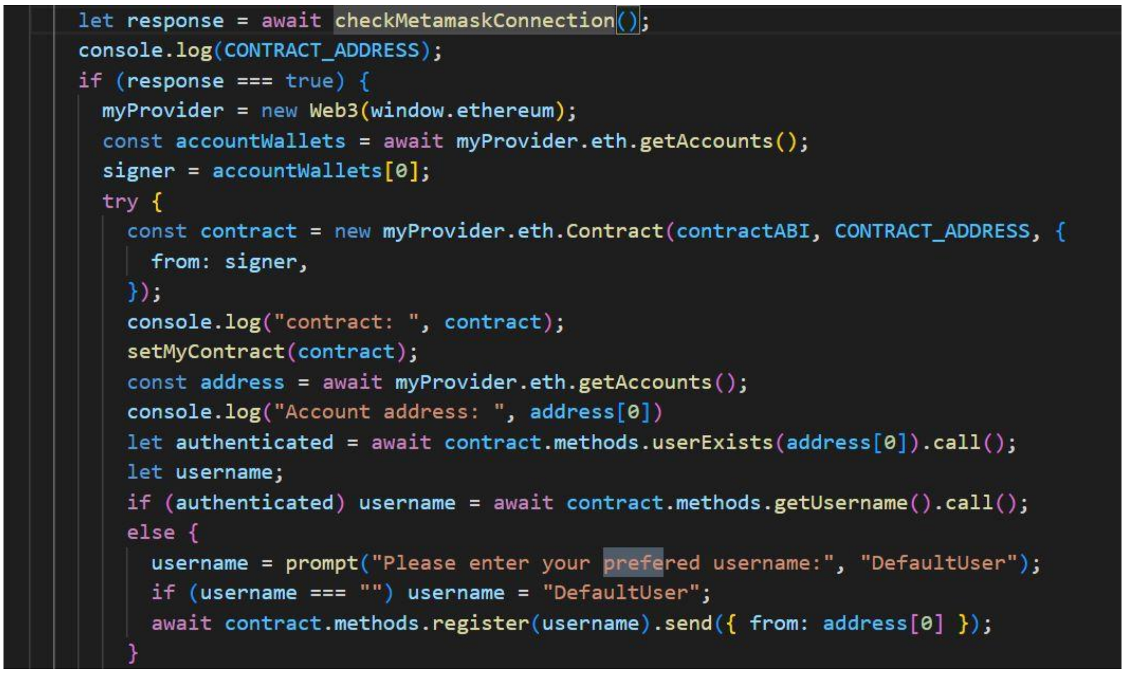
The application tries to connect the smart contract with the use of MetaMask, which manages the operations. It will check if the MetaMask extension is installed or not with the method `*await window.ethereum.enable()*`.



```
async function checkMetamaskConnection() {
  try {
    await window.ethereum.enable();
    return true;
  } catch (err) {
    return false;
  }
}
```

Figure 13 - Checking the MetaMask.

When the MetaMask is found, `*myProvider*` is created using the Web3 constructor and initialized with the ` *window.ethereum*` object. The `*window.ethereum* `object is provided by the Metamask extension and represents the user's Ethereum provider. `*myProvider*` has been used to interact with the Polygon network by creating instances of smart contracts, calling contract methods, and sending transactions. The `signer` is used as the sender address when creating an instance of the smart contract, which is a variable that represents the Polygon account selected by the user in Metamask. It ensures that the user's account is used to sign and send transactions related to the contract. It is obtained by calling `*myProvider.eth.getAccounts()* `.



```
let response = await checkMetamaskConnection();
console.log(CONTRACT_ADDRESS);
if (response === true) {
  myProvider = new Web3(window.ethereum);
  const accountWallets = await myProvider.eth.getAccounts();
  signer = accountWallets[0];
  try {
    const contract = new myProvider.eth.Contract(contractABI, CONTRACT_ADDRESS, {
      from: signer,
    });
    console.log("contract: ", contract);
    setMyContract(contract);
    const address = await myProvider.eth.getAccounts();
    console.log("Account address: ", address[0])
    let authenticated = await contract.methods.userExists(address[0]).call();
    let username;
    if (authenticated) username = await contract.methods.getUsername().call();
    else {
      username = prompt("Please enter your prefered username:", "DefaultUser");
      if (username === "") username = "DefaultUser";
      await contract.methods.register(username).send({ from: address[0] });
    }
```

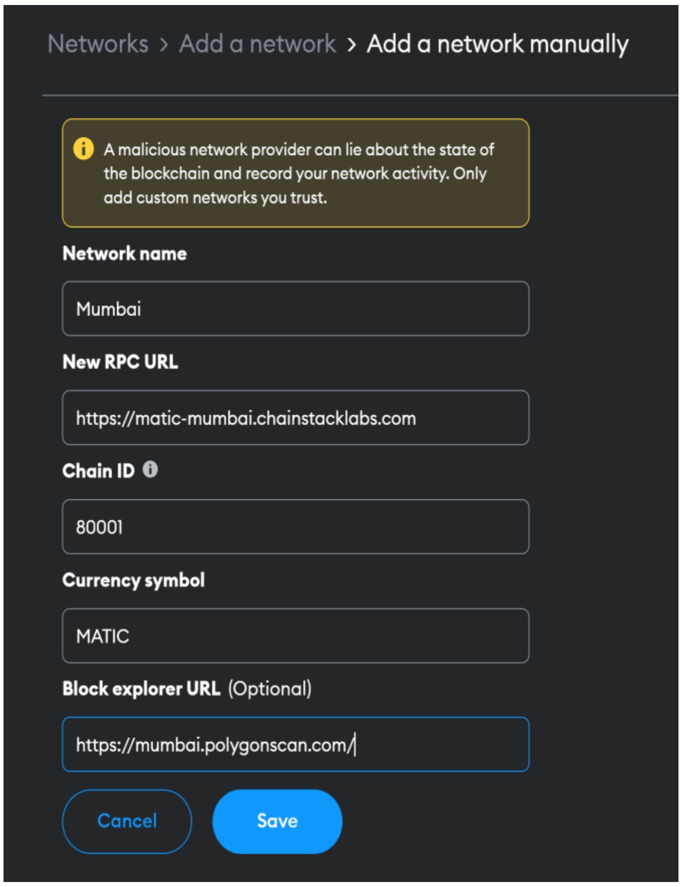
Figure 14 Connecting and registering to Blockchain.

7.4 MetaMask Wallet Setup

To use Polygon's Mumbai-Test network, users are required to integrate the Polygon network into their wallet. To configure the network, the user must provide specific details by accessing the network section of the MetaMask wallet and selecting the option to "Add network."

The user is asked to input a designated name for the network (Mumbai) in the Network Name section. To configure the RPC settings, input the following information: https://rpc-mumbai.maticvigil.com/ in the New RPC URL field, 80001 in the Chain ID field, MATIC in the Currency Symbol field, and https://mumbai.polygonscan.com/ in the Block Explorer URL field, the form for adding a network manually is shown in Figure 15.

Upon successful completion of all required fields, the wallet undergoes a conversion process to Polygon's Mumbai-Test network, as it is shown in Figure 16.


Networks > Add a network > Add a network manually

A malicious network provider can lie about the state of the blockchain and record your network activity. Only add custom networks you trust.

Network name: Mumbai

New RPC URL: https://matic-mumbai.chainstacklabs.com

Chain ID: 80001

Currency symbol: MATIC

Block explorer URL (Optional): https://mumbai.polygonscan.com/

Figure 15 - Adding Polygon Mumbai Testnet.

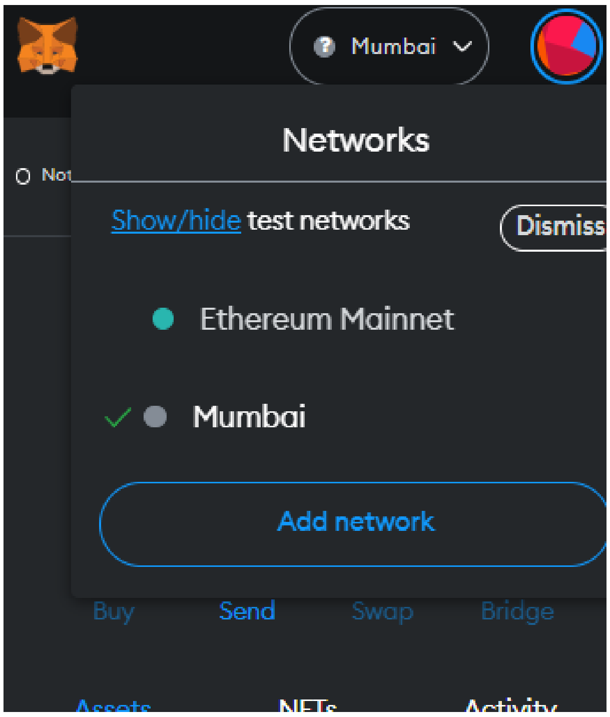


Figure 16 - Mumbai network.

The wallet does not have any MATIC. To fill the wallet with currency, the user should get some fake currency to perform the actions. It can be gotten from the Polygon faucet, which gives a small amount for testing purposes, the balance is funded as it is illustrated in **Error! Reference source not found.**.

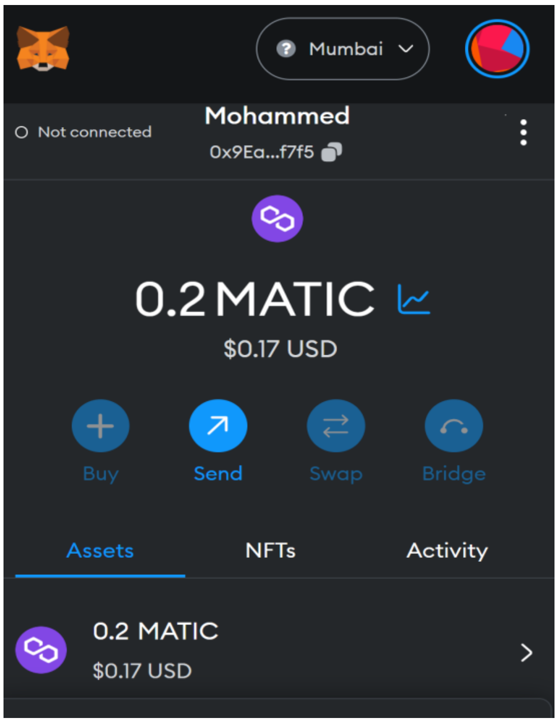
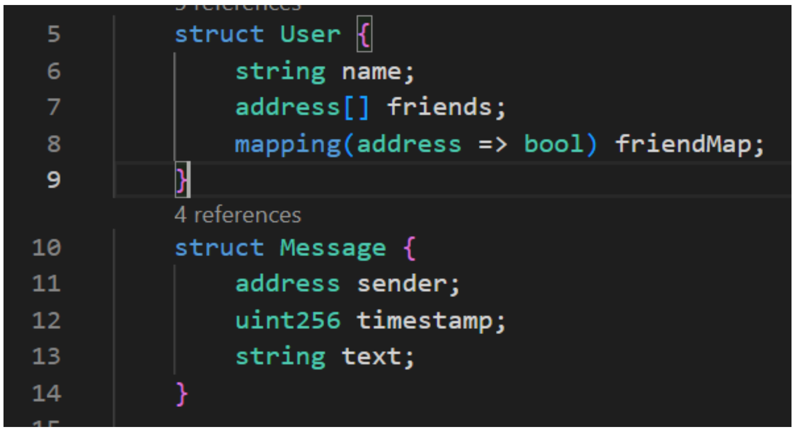


Figure 17- Funded wallet.

7.5 Smart Contract Functionalities

The smart contract is used in the messaging application to enable secure and decentralized chats between users, verify their identities, and provide an encrypted, secure message. It consists of one smart contract that handles all the operations of the messaging app because having more than one smart contract and communication between them takes more gas than having a single contract. The smart contract consists of two main structs as shown in Figure 18:

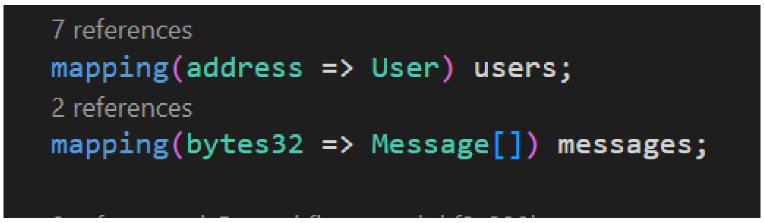
- User struct consists of the user's name, an array of addresses of their friends, and a map that associates each friend's address with a Boolean value indicating the friendship status of the user with that friend.
- Message struct contains the address of the sender, a timestamp, and the text.



```
struct User {
    string name;
    address[] friends;
    mapping(address => bool) friendMap;
}
4 references
struct Message {
    address sender;
    uint256 timestamp;
    string text;
}
```

Figure 18 - User struct & message struct.

There are two mappings for the structs: the first one is `users", which associates the user's address with the User struct, and the second one is a mapping of "message," which associates a byte32 hashed message with an array of the `message` struct, which can be used to store and retrieve the messages of the users that are friends with each other.

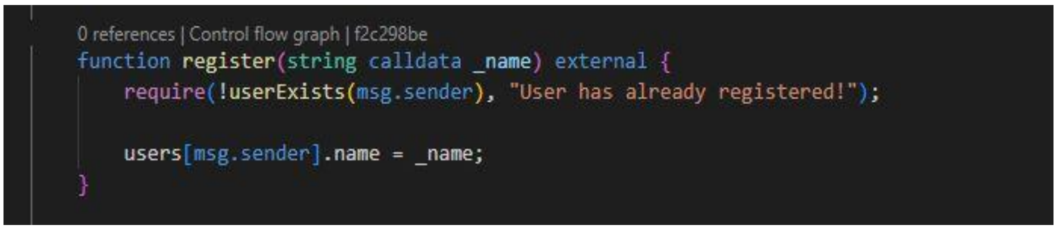


```
7 references
mapping(address => User) users;
2 references
mapping(bytes32 => Message[]) messages;
```

Figure 19 - Users & messages mapping.

The functionalities of the smart contract:

- The **register()** function takes a string as an input, enabling users to sign up for the Polygon blockchain. The function requires whether the user has an account already. The function used `calldata` to just read the names of the users from the blockchain.

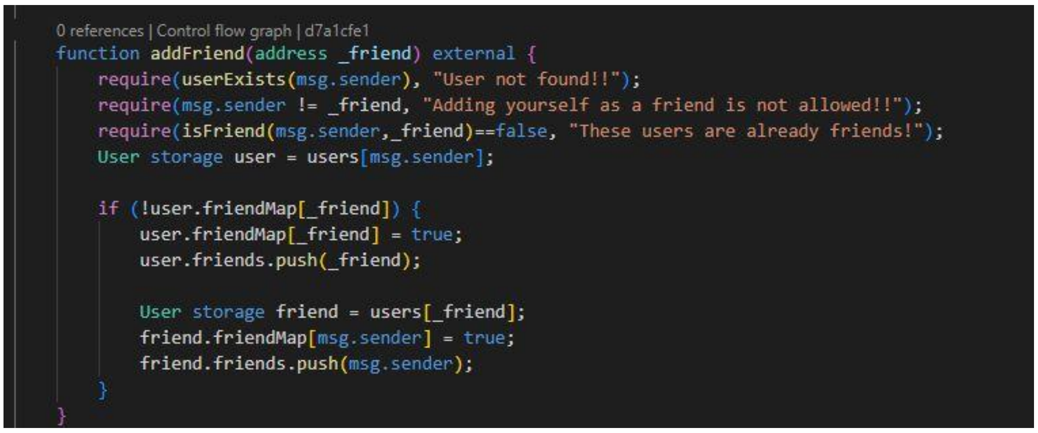


```
0 references | Control flow graph | f2c298be
function register(string calldata _name) external {
    require(!userExists(msg.sender), "User has already registered!");

    users[msg.sender].name = _name;
}
```

Figure 20 - Register function.

- The **addFriend()** function allows the user to add friends by accepting the friend's address. The function prevents the user from adding themselves, it checks if the user is available or not, and it reverts when the user tries to add a friend where it is in the friend list.



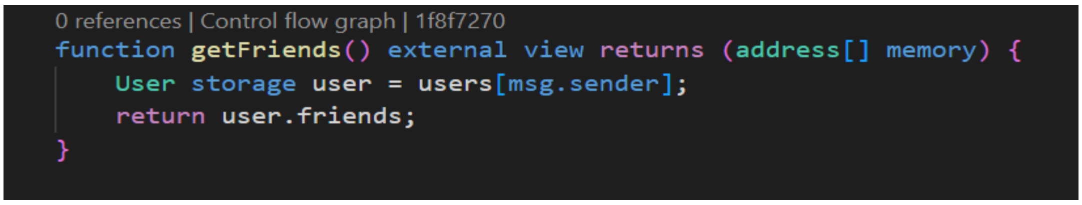
```
0 references | Control flow graph | d7a1cfe1
function addFriend(address _friend) external {
    require(userExists(msg.sender), "User not found!!");
    require(msg.sender != _friend, "Adding yourself as a friend is not allowed!!");
    require(isFriend(msg.sender,_friend)==false, "These users are already friends!");
    User storage user = users[msg.sender];

    if (!user.friendMap[_friend]) {
        user.friendMap[_friend] = true;
        user.friends.push(_friend);

        User storage friend = users[_friend];
        friend.friendMap[msg.sender] = true;
        friend.friends.push(msg.sender);
    }
}
```

Figure 21 - Add friend function.

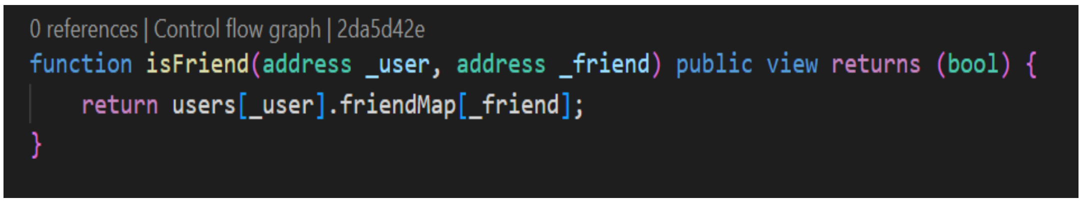
- The **getFriend()** function returns an array of the user’s friend.



```
0 references | Control flow graph | 1f8f7270
function getFriends() external view returns (address[] memory) {
    User storage user = users[msg.sender];
    return user.friends;
}
```

Figure 22 - Get friend function.

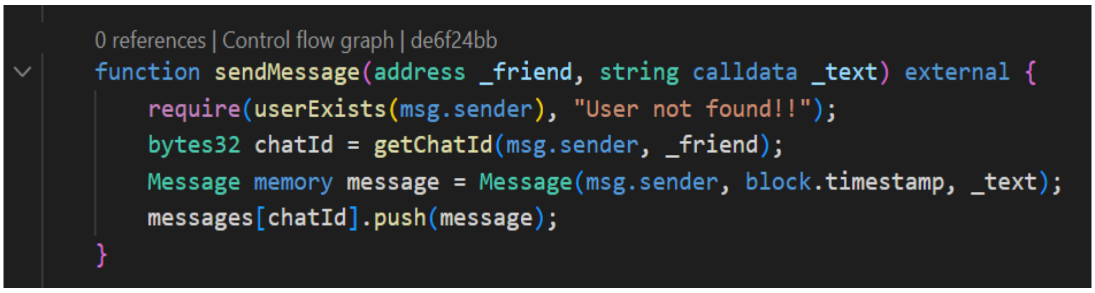
- The **isFriend()** function accepts two addresses of the users and checks if the users are friends.



```
0 references | Control flow graph | 2da5d42e
function isFriend(address _user, address _friend) public view returns (bool) {
    return users[_user].friendMap[_friend];
}
```

Figure 23 - Is friend function.

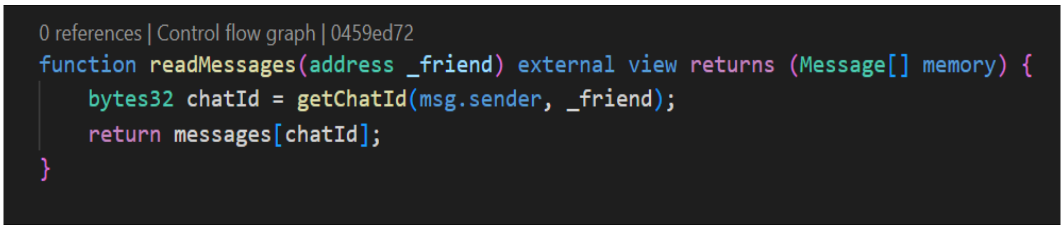
- The **sendMessage()** function accepts friend's address and text message. It allows the friends to send messages. It creates a Chat Id for the friends and adds the text message to the Chat Id. The functions check if there is a friend or not.



```
0 references | Control flow graph | de6f24bb
function sendMessage(address _friend, string calldata _text) external {
    require(userExists(msg.sender), "User not found!!");
    bytes32 chatId = getChatId(msg.sender, _friend);
    Message memory message = Message(msg.sender, block.timestamp, _text);
    messages[chatId].push(message);
}
```

Figure 24 - Send message function.

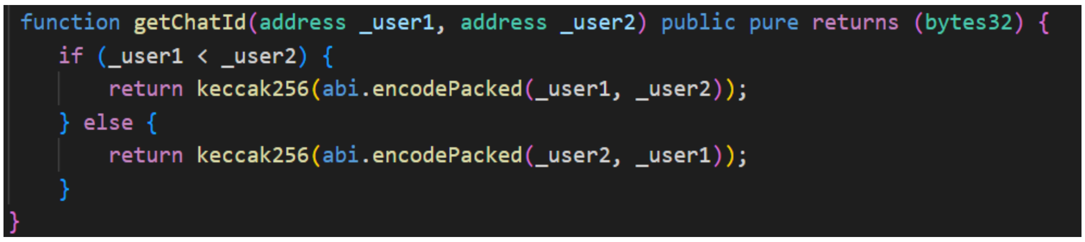
- The **readMessages()** function accepts the address of the friend to read the message that retrieved from the Chat Id.



```
0 references | Control flow graph | 0459ed72
function readMessages(address _friend) external view returns (Message[] memory) {
    bytes32 chatId = getChatId(msg.sender, _friend);
    return messages[chatId];
}
```

Figure 25 - Read messages function.

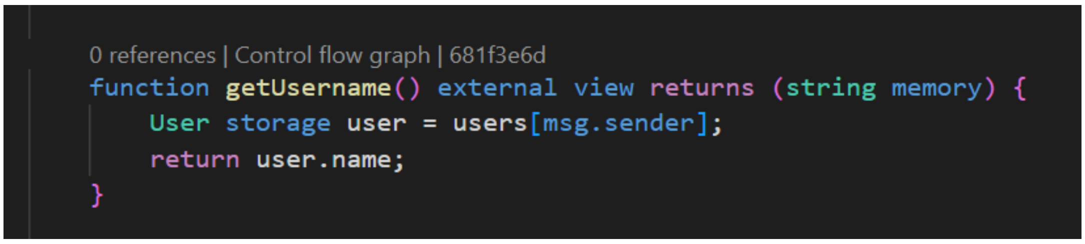
- The **getChatId()** function generates a Chat Id of two users based on their addresses.



```
function getChatId(address _user1, address _user2) public pure returns (bytes32) {
    if (_user1 < _user2) {
        return keccak256(abi.encodePacked(_user1, _user2));
    } else {
        return keccak256(abi.encodePacked(_user2, _user1));
    }
}
```

Figure 26 - Get ChatId function.

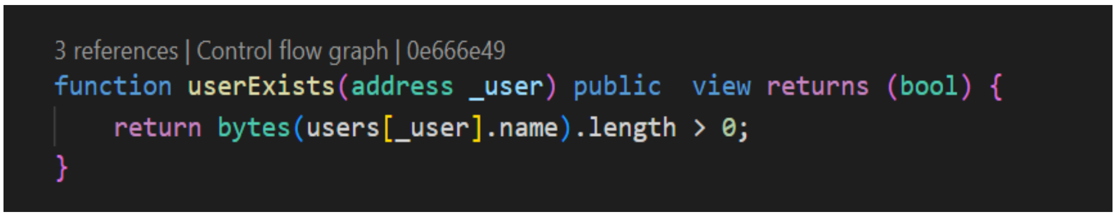
- The **getUsername()** function returns the username of the user.



```
0 references | Control flow graph | 681f3e6d
function getUsername() external view returns (string memory) {
    User storage user = users[msg.sender];
    return user.name;
}
```

Figure 27 - Get username function.

- The **userExists()** function checks if the user is available in the blockchain.

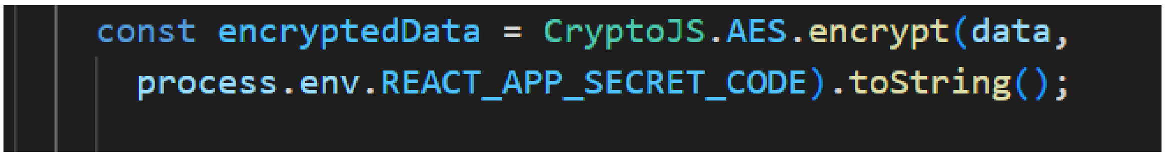


```
3 references | Control flow graph | 0e666e49
function userExists(address _user) public  view returns (bool) {
    return bytes(users[_user].name).length > 0;
}
```

Figure 28 - User exists function.

7.6 Encryption and Decryption Techniques for Secure Message Communication

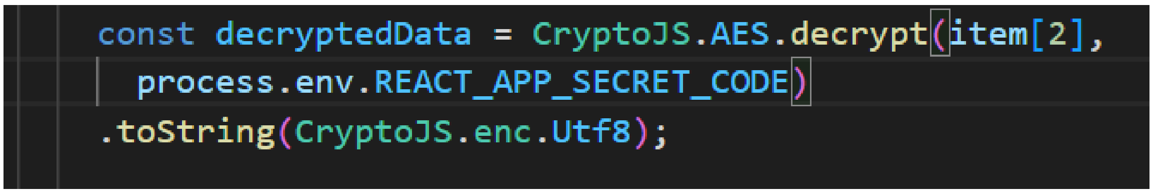
The communication system ensures that all messages are encrypted and can only be decrypted by the intended recipient. The encryption and decryption processes are facilitated using CryptoJS. When a user sends a text message, it undergoes encryption via the AES (Advanced Encryption Standard) algorithm, which transforms the content into an incomprehensible format. This encryption operation relies on a confidential key acquired from an environment variable.



```
const encryptedData = CryptoJS.AES.encrypt(data,
  process.env.REACT_APP_SECRET_CODE).toString();
```

Figure 29- Encrypt message.

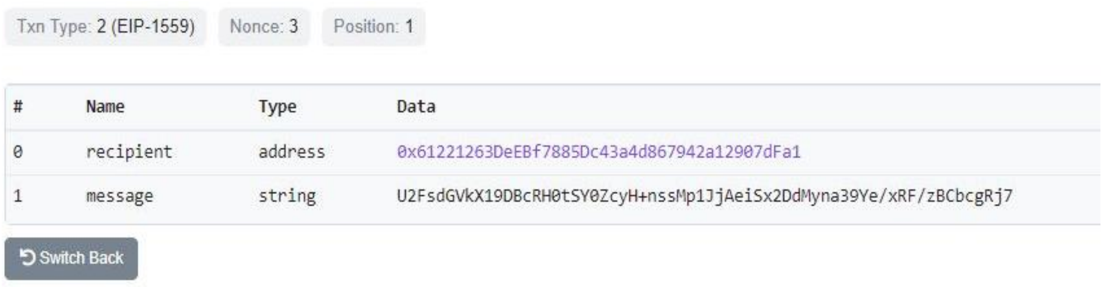
The communications are of a public nature and are accessible through the blockchain platform. The decryption of these messages involves the utilization of the AES algorithm implemented through CryptoJS. This process involves employing the encrypted text messages in conjunction with the corresponding encryption key. It is essential to emphasize that only the intended recipient possesses the authority to decrypt the encrypted message.



```
const decryptedData = CryptoJS.AES.decrypt(item[2],
  process.env.REACT_APP_SECRET_CODE)
.toString(CryptoJS.enc.Utf8);
```

Figure 30 - Decrypt message.

As result of encryption the message will be publicly shown on the Polygon Mumbai network in encoded format as shown in Figure 31.



Txn Type: 2 (EIP-1559) Nonce: 3 Position: 1

#	Name	Type	Data
0	recipient	address	0x61221263DeEBf7885Dc43a4d867942a12907dFa1
1	message	string	U2FsdGVkX19DBcRH0tSY0ZcyH+nssMp1JjAeiSx2DdMyna39Ye/xRF/zBCbcgRj7

Figure 31 - Message on Blockchain

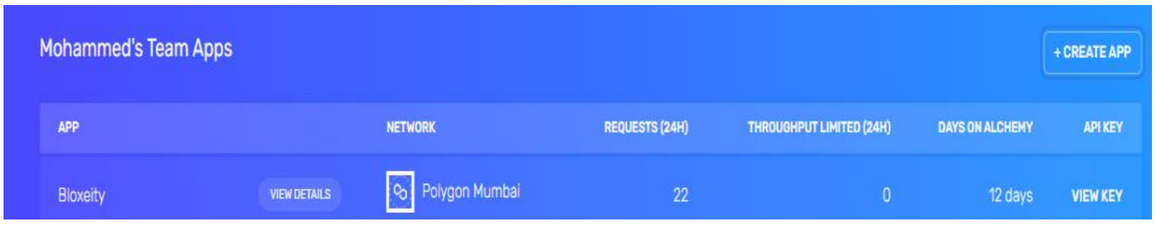
7.7 Deploying the Smart Contract

For deploying the smart contract, Hardhat has been used, which is an environment for deploying the smart contract locally. The dependency is installed by installing in the terminal ` *npm install --save-dev hardhat*`, After Hardhat has been installed, a new project can be created by running `*npx hardhat*` in the terminal, which installs Hardhat in the project's directory and creates node modules.

Once every folder has been added to the project. The deployment network should be configured. When we run 'npx hardhat' and established a JavaScript project, Hardhat built a boilerplate project for us. In the project directory, a contract folder will be added, and the smart contract code should be written there.

To deploy the smart contracts, the Polygon wallet and Alchemy should be added to the application. For connecting to these, ethers.js and Dotenv should be installed. Ether.js is used for interacting with the Polygon blockchain, and Dotenv is a package used for protecting secrets such as our account private key and other sensitive information by loading environment variables from a `. *env*` file into process.env that can be safely referenced in our code. To install these packages, the npm can be used to install them in the terminal.

After installing the required dependencies, Alchemy and Polygon accounts should be added to the application environment. There should be a unique private key to our wallet address to send transactions. To connect to the Alchemy API key, an HTTPS URL should be added to it. The HTTPS URL can be obtained by creating an app in the Alchemy account.

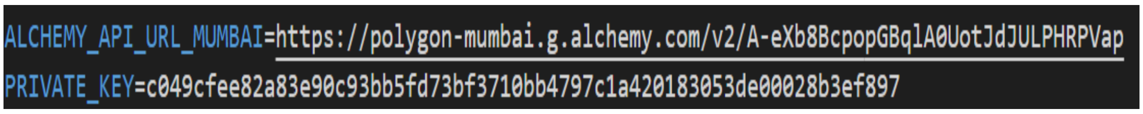


Mohammed's Team Apps

APP		NETWORK	REQUESTS (24H)	THROUGHPUT LIMITED (24H)	DAYS ON ALCHEMY	API KEY
Bloxeity	VIEW DETAILS	Polygon Mumbai	22	0	12 days	VIEW KEY

Figure 32 - Project in Alchemy.

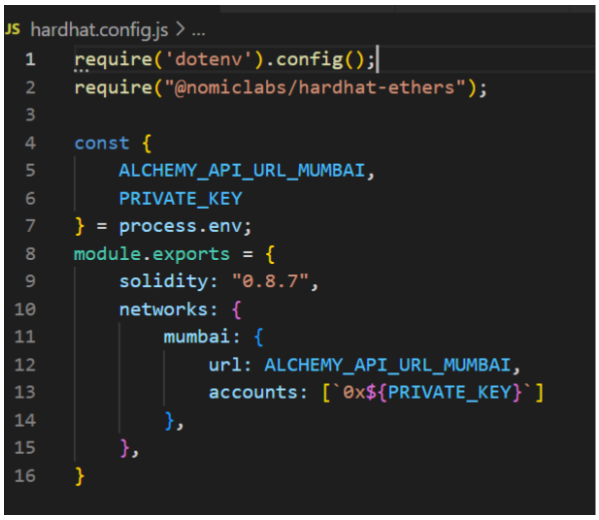
After getting the private key from the Polygon wallet and the Alchemy API, they have to be put inside the `.*env*` file as shown in Figure 33.



```
ALCHEMY_API_URL_MUMBAI=https://polygon-mumbai.g.alchemy.com/v2/A-eXb8BcpopGBqlA0UotJdJULPHRPVap
PRIVATE_KEY=c049cfee82a83e90c93bb5fd73bf3710bb4797c1a420183053de00028b3ef897
```

Figure 33 - Constant values.

Before compiling, hardhat should be aware of the dependencies and the private key that have been added to the project. The "hardhat.config.js" file should configure a network for the Mumbai test net using the Alchemy API and a private key for account access and specify which version of Solidity is used.

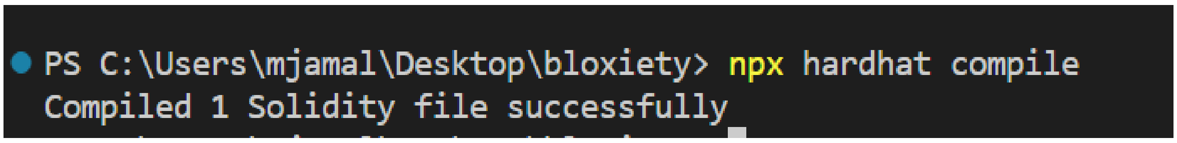


```javascript
require('dotenv').config();
require("@nomiclabs/hardhat-ethers");

const {
    ALCHEMY_API_URL_MUMBAI,
    PRIVATE_KEY
} = process.env;
module.exports = {
    solidity: "0.8.7",
    networks: {
        mumbai: {
            url: ALCHEMY_API_URL_MUMBAI,
            accounts: [`0x${PRIVATE_KEY}`]
        },
    },
}
```

Figure 34 - hardhat.config.js file.

For compiling the smart contract `*npx hardhat compile*` has been used in the terminal, which automatically detects the Solidity file in the `contracts` directory and compiles them into bytecode and ABI, as shown in Figure 35 the smart contract compiled successfully.

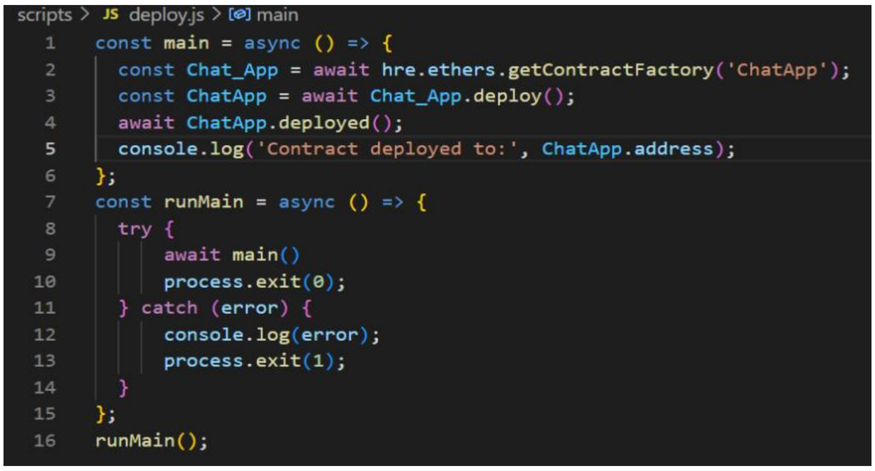


```
PS C:\Users\mjamal\Desktop\bloxiety> npx hardhat compile
Compiled 1 Solidity file successfully
```

Figure 35 - Compiled smart contract in the terminal log.

After the smart contract is compiled, it is ready for deployment to the Polygon Mumbai Test network. But before that, there should be a script code for deployment. The contract factory for the "ChatApp` contract is obtained by using the "ethers.js` library in the

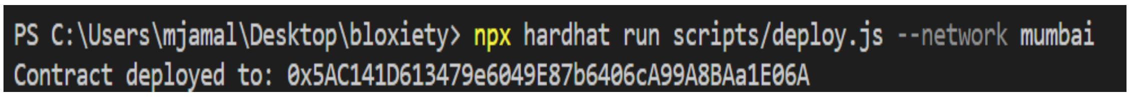
Hardhat run environment. To deploy the contract, a ` *npx hardhat run scripts/deploy.js --network mumbai*` is used in the terminal, which runs the `deploy.js` using the Hardhat environment with the `mumbai` network specified as the target network. After that, the deployed contract address is logged to the console, as it is illustrated in Figure 37.



scripts > JS deploy.js > main

```javascript
const main = async () => {
  const Chat_App = await hre.ethers.getContractFactory('ChatApp');
  const ChatApp = await Chat_App.deploy();
  await ChatApp.deployed();
  console.log('Contract deployed to:', ChatApp.address);
};
const runMain = async () => {
  try {
      await main()
      process.exit(0);
  } catch (error) {
      console.log(error);
      process.exit(1);
  }
};
runMain();
```

Figure 36 - deploy.js file.



```
PS C:\Users\mjamal\Desktop\bloxiety> npx hardhat run scripts/deploy.js --network mumbai
Contract deployed to: 0x5AC141D613479e6049E87b6406cA99A8BAa1E06A
```

Figure 37 - The deployed contract address in terminal log.

7.8 Application Presentation

The homepage of the application provides comprehensive information about its features and functionalities. It comprises various elements that facilitate user interaction and navigation. Positioned prominently on the page are several essential buttons, including the option to connect the user to the Metamask wallet, add friends, and reload messages.

The left side of the application interface accommodates the friend list, allowing users to access and manage their connections conveniently. This section enables users to view, add, and remove friends as per their preferences and requirements.

On the right side of the application interface, the message card area is located. This dedicated space serves as the primary platform for users to engage in message exchanges. Users can access and interact with their messages within this designated section. The message card area is designed to provide an organized and user-friendly environment for users to communicate effectively.

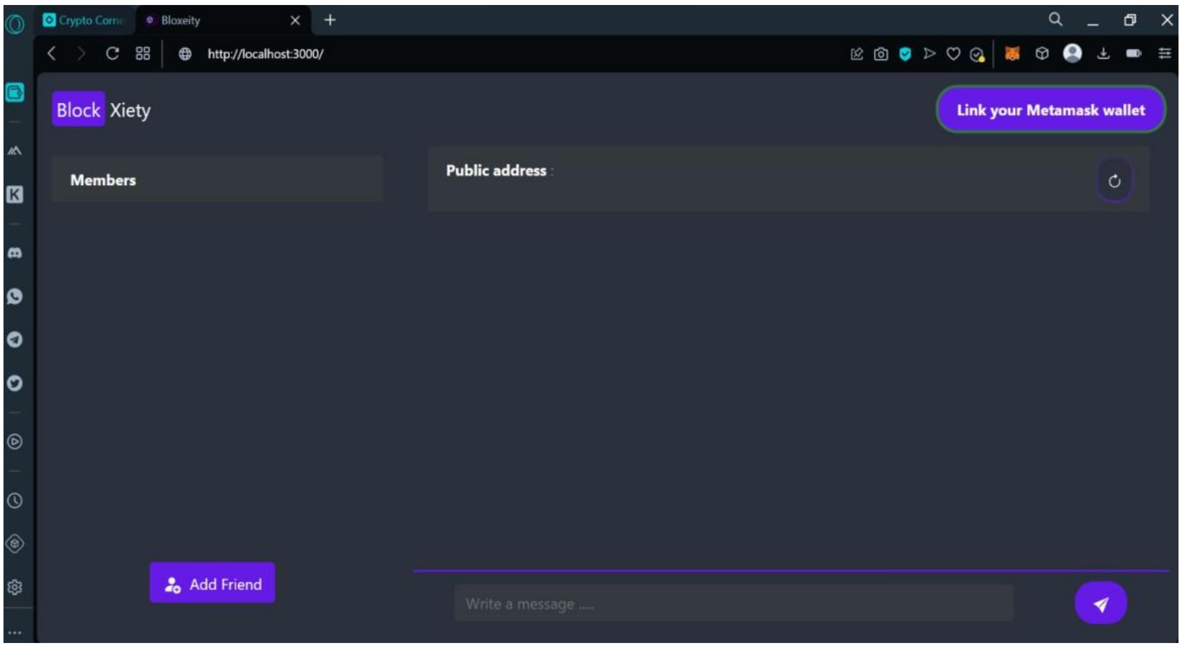
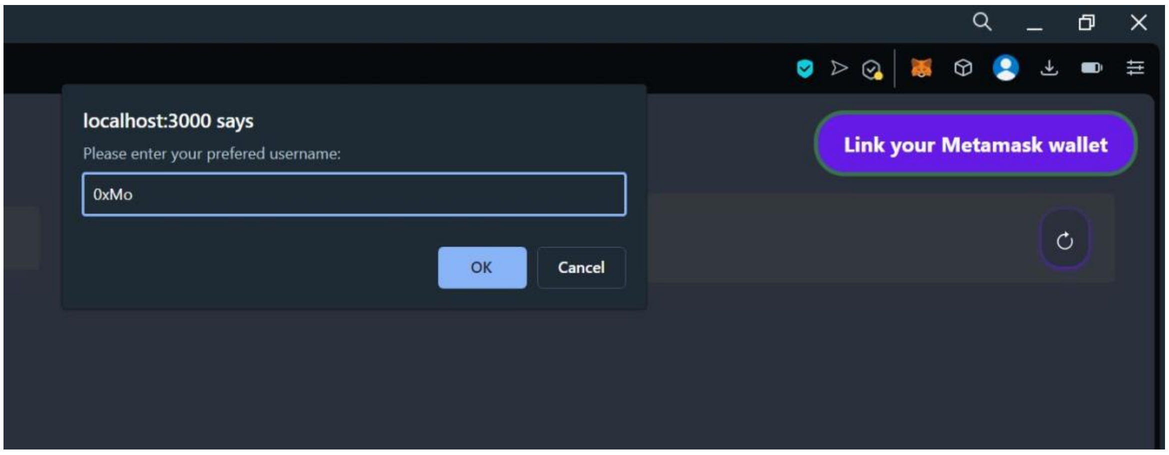


Figure 38 - Home page.

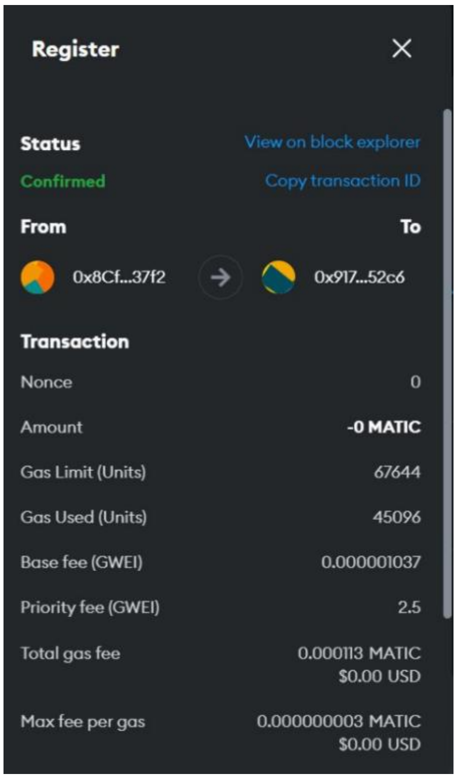
Upon clicking the "Link your Metamask wallet" button, the application prompts the user to provide their preferred name as part of the registration process for creating an account on the Polygon blockchain, as it is shown in Figure 39, it ask the user to enter the preferred name for registering on the blockchain .


localhost:3000 says

Please enter your prefered username:

0xMo

Figure 39 - Pop message.


Register

Status | View on block explorer
Confirmed | Copy transaction ID
From | To
0x8Cf...37f2 → 0x917...52c6

Transaction

Nonce	0
Amount	-0 MATIC
Gas Limit (Units)	67644
Gas Used (Units)	45096
Base fee (GWEI)	0.000001037
Priority fee (GWEI)	2.5
Total gas fee	0.000113 MATIC $0.00 USD
Max fee per gas	0.000000003 MATIC $0.00 USD

Figure 40 - Register transaction on MetaMask.

Once the user has successfully registered with their preferred name, a user profile will be generated, displaying the registered account's username. This user profile serves as a representation of the user within the application's ecosystem, providing the username for the user.

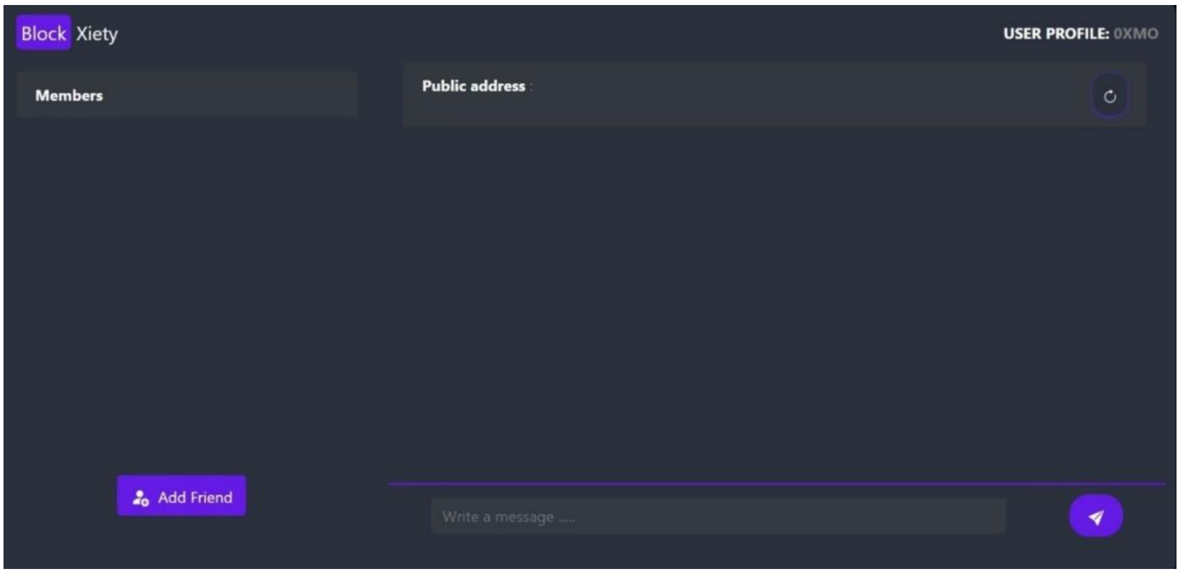


Figure 41 - User profile.

To add a friend, a registered user can click on the "Add Friend" button, triggering the display of a form. This form prompts the user to enter the public address of the desired user, as it is illustrated in Figure 42, who must already be registered on the blockchain.

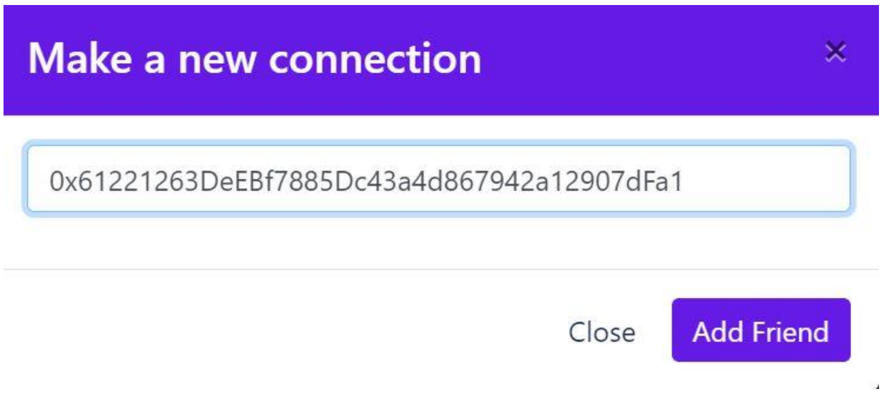
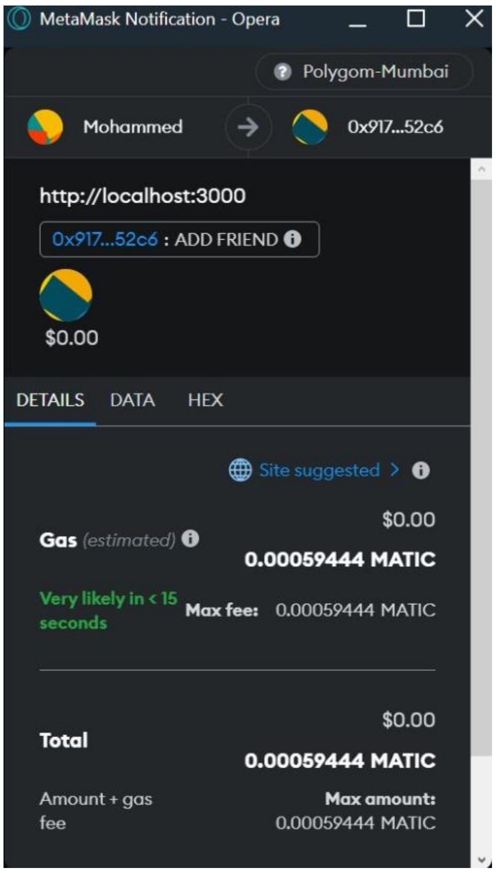


Figure 42 - Add friend form.



Gas (estimated) $0.00 0.00059444 MATIC

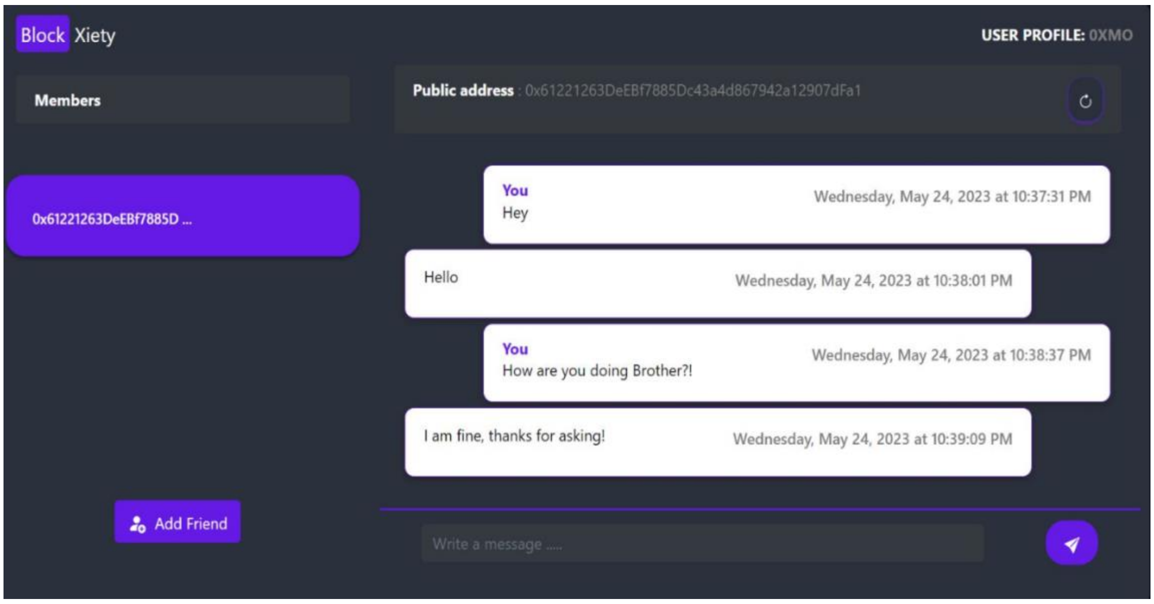
Very likely in < 15 seconds Max fee: 0.00059444 MATIC

Total $0.00 0.00059444 MATIC

Amount + gas fee Max amount: 0.00059444 MATIC

Figure 43 - Add friend transaction on MetaMask.

Following the successful addition of a friend, users gain the ability to engage in message exchanges with each other within the application, as it is shown in Figure 44, the users are engaging the messages.



Public address : 0x61221263DeEBf7885Dc43a4d867942a12907dFa1

0x61221263DeEBf7885D ...

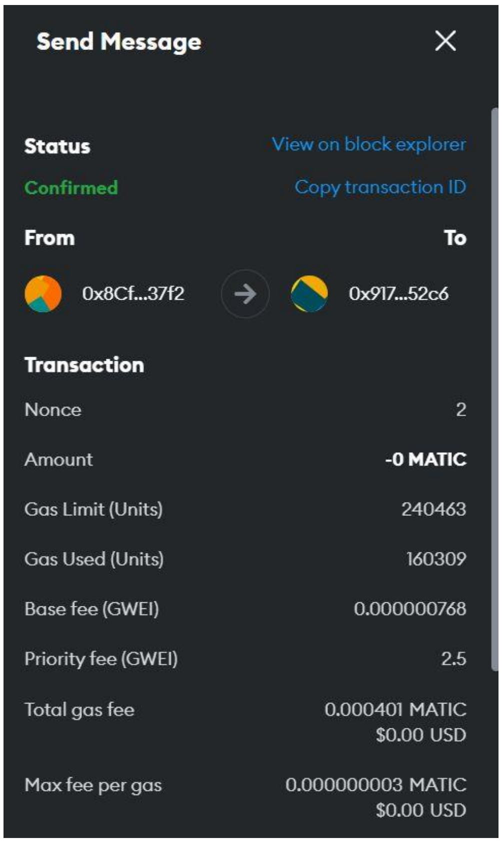
You
Hey
Wednesday, May 24, 2023 at 10:37:31 PM

Hello
Wednesday, May 24, 2023 at 10:38:01 PM

You
How are you doing Brother?!
Wednesday, May 24, 2023 at 10:38:37 PM

I am fine, thanks for asking!
Wednesday, May 24, 2023 at 10:39:09 PM

Figure 44 - Message panel.



Status
Confirmed

From 0x8Cf...37f2
To 0x917...52c6

Transaction

Nonce	2
Amount	-0 MATIC
Gas Limit (Units)	240463
Gas Used (Units)	160309
Base fee (GWEI)	0.000000768
Priority fee (GWEI)	2.5
Total gas fee	0.000401 MATIC $0.00 USD
Max fee per gas	0.000000003 MATIC $0.00 USD

Figure 45 - Send message transaction on MetaMask.

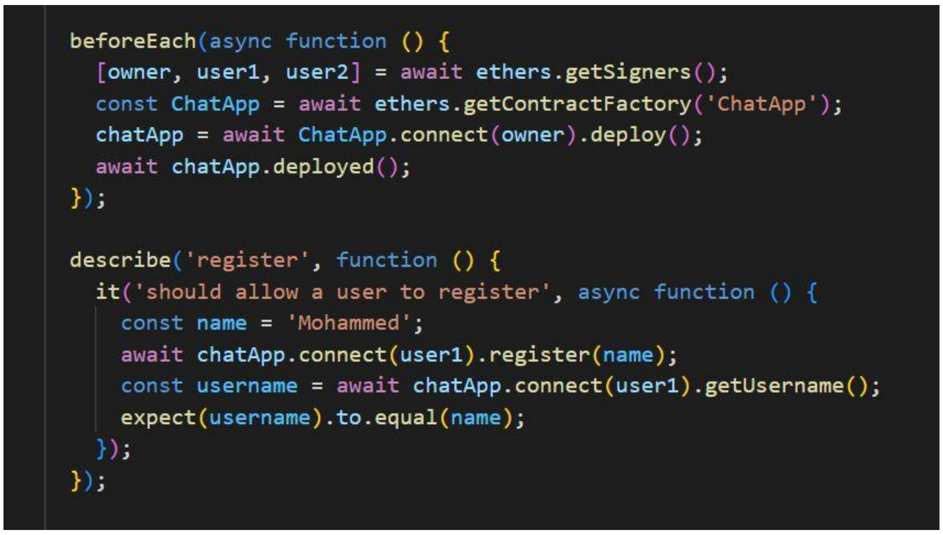
8 TESTING SMART CONTRACT

The functionality of the smart contract should be tested before deployment. There are a lot of libraries and ways to test smart contracts. Mocha and Chai have been used to test the necessary functions of the smart contract, such as registering, adding a friend, and sending and receiving messages.

The first step is to install the necessary dependencies with the npm. Once the successful installation of dependencies has taken place, the script and use cases are to be incorporated into the test directory, which is generated during the creation of a Hardhat project. The function can be tested in the terminal with `*npx hardhat test*`. The logs of the tests should be available in the terminal.

8.1 Register Use Case

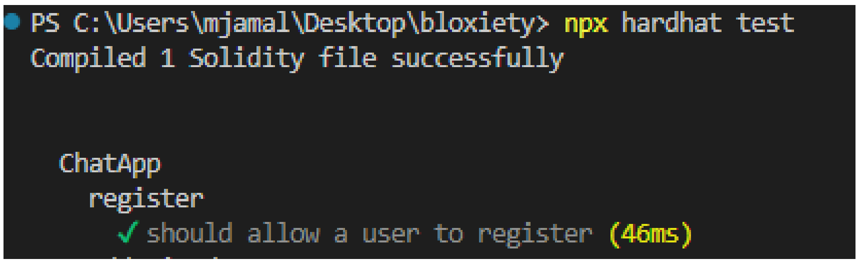
The register case verifies that the user should register successfully; it checks that the retrieved name is equal to the expected name.



```
beforeEach(async function () {
  [owner, user1, user2] = await ethers.getSigners();
  const ChatApp = await ethers.getContractFactory('ChatApp');
  chatApp = await ChatApp.connect(owner).deploy();
  await chatApp.deployed();
});

describe('register', function () {
  it('should allow a user to register', async function () {
    const name = 'Mohammed';
    await chatApp.connect(user1).register(name);
    const username = await chatApp.connect(user1).getUsername();
    expect(username).to.equal(name);
  });
});
```

Figure 46 - Register test case.



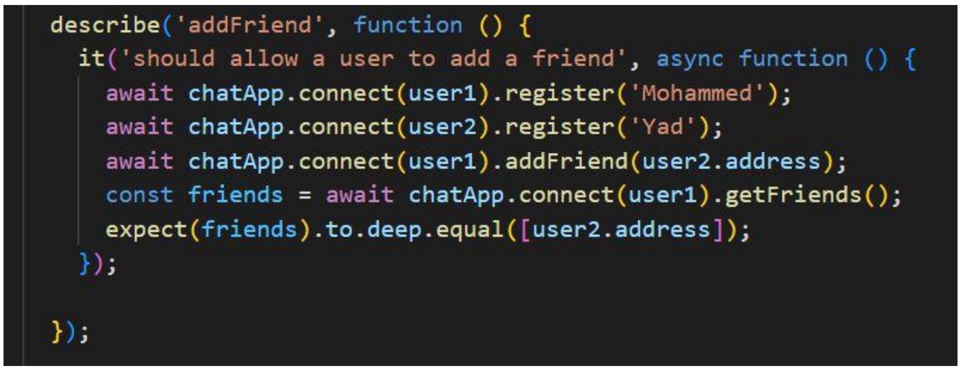
```
PS C:\Users\mjamal\Desktop\bloxiety> npx hardhat test
Compiled 1 Solidity file successfully


  ChatApp
    register
      ✓ should allow a user to register (46ms)
```

Figure 47 - Register test case result.

8.2 Add Friend Use Case

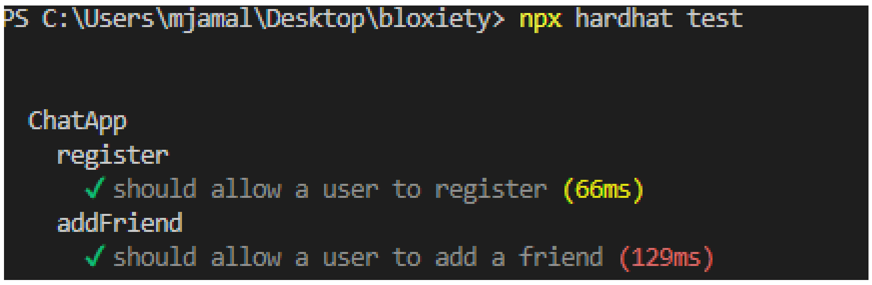
The Add Friend case verifies that the user can add a friend. It connects to the smart contract of the messaging app. It registers two users. And adds the second user as a friend. Finally, we retrieve the list of friends of user 1 and find that user 2 is in the list of friends.



```
describe('addFriend', function () {
  it('should allow a user to add a friend', async function () {
    await chatApp.connect(user1).register('Mohammed');
    await chatApp.connect(user2).register('Yad');
    await chatApp.connect(user1).addFriend(user2.address);
    const friends = await chatApp.connect(user1).getFriends();
    expect(friends).to.deep.equal([user2.address]);
  });

});
```

Figure 48 - Add friend test case.



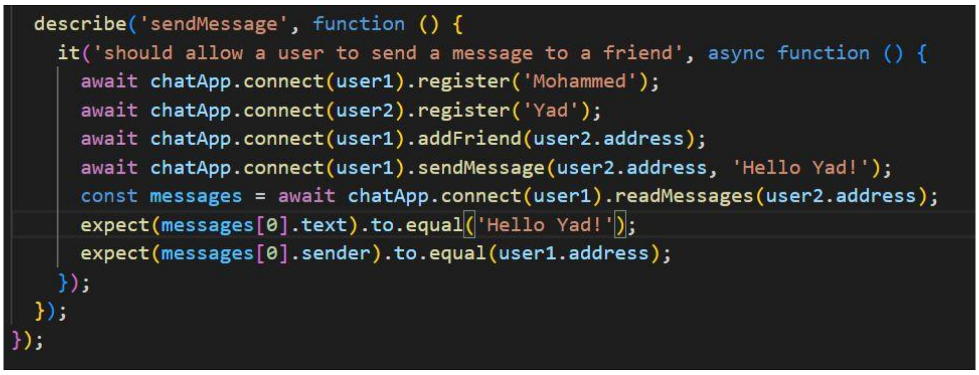
```
PS C:\Users\mjamal\Desktop\bloxiety> npx hardhat test


  ChatApp
    register
      ✓ should allow a user to register (66ms)
    addFriend
      ✓ should allow a user to add a friend (129ms)
```

Figure 49 - Add friend test case result.

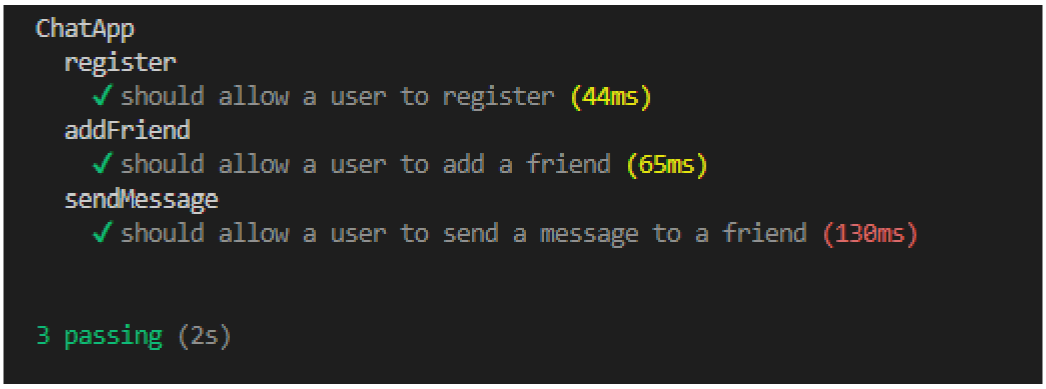
8.3 Message Use Case

The message case tests if the user can send a message to a friend. It tries to send a message from the 1st user to the 2nd user and expects the message to be sent by the 1st user and read by the 2nd user.



```
  describe('sendMessage', function () {
    it('should allow a user to send a message to a friend', async function () {
      await chatApp.connect(user1).register('Mohammed');
      await chatApp.connect(user2).register('Yad');
      await chatApp.connect(user1).addFriend(user2.address);
      await chatApp.connect(user1).sendMessage(user2.address, 'Hello Yad!');
      const messages = await chatApp.connect(user1).readMessages(user2.address);
      expect(messages[0].text).to.equal('Hello Yad!');
      expect(messages[0].sender).to.equal(user1.address);
    });
  });
});
```

Figure 50 - Send message test case.



```
ChatApp
  register
    ✓ should allow a user to register (44ms)
  addFriend
    ✓ should allow a user to add a friend (65ms)
  sendMessage
    ✓ should allow a user to send a message to a friend (130ms)


3 passing (2s)
```

Figure 51 - Send message test case result.

CONCLUSION

The aim of this project was to develop a decentralized messaging application on the blockchain that ensures data integrity, enhances security, and eliminates the need for central authority. The messaging app achieves decentralization by utilizing a smart contract deployed on the blockchain to store and process data. While all transactions are public and accessible, message encryption ensures that only the intended recipient can decrypt the messages. The theoretical section of the project covered the current state of the messaging app, blockchain and smart contract terminology, and provided an overview of different blockchain platforms and technologies used for its development. In the practical part, a business model and user flow were described, along with the implementation details of the messaging app, encryption/decryption mechanisms, user interface, and basic smart contract functionality testing.

Despite the advantages offered by blockchain technology, it is important to recognize certain limitations such as transaction throughput and scalability. However, these challenges were addressed by building the messaging app on the Polygon blockchain, a Layer 2 solution built on top of Ethereum. Polygon offers faster transaction speeds, improved scalability, and lower gas fees compared to the Ethereum blockchain. Nevertheless, the network still requires additional validators to achieve full decentralization.

Looking ahead, future developments for the messaging app include the incorporation of group chat functionality, the ability to accept friends, and media sharing features. Additionally, the integration of machine learning algorithms for detecting text message toxicity is envisioned, aiming to enhance the user experience and foster a safer communication environment. These advancements will further contribute to the evolution and adoption of decentralized messaging applications on the blockchain.

BIBLIOGRAPHY

[1] "What Is Blockchain Technology - IBM Blockchain | IBM." What Is Blockchain Technology? - IBM Blockchain | IBM, www.ibm.com/topics/blockchain.

[2] "What Is Blockchain?" McKinsey & Company, www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-blockchain.

[3] "What Are Centralized, Decentralized, and Distributed Systems?" Educative: Interactive Courses for Software Developers, www.educative.io/answers/what-are-centralized-decentralized-and-distributed-systems.

[4] "What Is Blockchain Technology? - Blockchaining Explained - AWS." Amazon Web Services, Inc., aws.amazon.com/what-is/blockchain.

[5] Gupta, Sahil, et al. "Emergence of Blockchain Technology: Fundamentals, Working and Its Various Implementations." SSRN Electronic Journal, Elsevier BV, 2020. Crossref, https://doi.org/10.2139/ssrn.3569577.

[6]Panel®, Expert. "Council Post: 15 Industries That Could Significantly Benefit From Blockchain Technology." Forbes, 10 June 2022, www.forbes.com/sites/forbestechcouncil/2022/06/10/15-industries-that-could-significantly-benefit-from-blockchain-technology.

[7] "Blockchain for Digital Identity | Real World Blockchain Use Cases | ConsenSys." ConsenSys, consensys.net/blockchain-use-cases/digital-identity.

[8] "History of Blockchain - GeeksforGeeks." GeeksforGeeks, 10 Mar. 2021, www.geeksforgeeks.org/history-of-blockchain

[9] "Types of Blockchain: Public, Private, or Something in Between | Foley and Lardner LLP." Types of Blockchain: Public, Private, or Something Else | Foley & Lardner LLP, 19 Aug. 2021,

[10] Himanshi. "An Introduction to Private Blockchain - Shiksha Online." *Shiksha Online*, 6 Mar. 2023, www.shiksha.com/online-courses/articles/private-blockchain.

[11] "What Are the 4 Different Types of Blockchain Technology? | TechTarget." CIO, www.techtarget.com/searchcio/feature/What-are-the-4-different-types-of-blockchain-technology.

[12] Iredale, Gwyneth. "Ultimate Guide to Pros and Cons of Blockchain." 101 Blockchains, 2 Feb. 2021, 101blockchains.com/pros-and-cons-of-blockchain

[13] Team, Shardeum Content, et al. "Pros and Cons of Blockchain Technology - an Overview | Shardeum." EVM-based Sharded Layer 1 Blockchain, 8 Aug. 2022, shardeum.org/blog/pros-and-cons-of-blockchain.

[14] "Building a Transparent Supply Chain." Harvard Business Review, 1 May 2020, hbr.org/2020/05/building-a-transparent-supply-chain.

[15] "What Is Blockchain Security? | IBM." What Is Blockchain Security? | IBM, www.ibm.com/topics/blockchain-security.

[16] Geroni, Diego. "Blockchain Scalability Problem - Why Is It Difficult to Scale Blockchain." 101 Blockchains, 30 Sept. 2021, 101blockchains.com/blockchain-scalability-challenges.

[17] "What Are the Disadvantages of Blockchain | the Financial Express." What Are the Disadvantages of Blockchain | the Financial Express, 18 Sept. 2022, www.financialexpress.com/blockchain/what-are-the-disadvantages-of-blockchain/2672273/#:~:text=Immutability%20can%20only%20exist%20if,can't%20erase%20its%20record.

[18] "Blockchain Technology: Pros and Cons [2023]." Blockchain Technology: Pros and Cons [2023], www.knowledgehut.com/blog/blockchain/blockchain-technology-pros-cons.

[19] "What Is Blockchain Technology? How Does It Work? | Built In." What Is Blockchain Technology? How Does It Work? | Built In, 1 Sept. 2022, builtin.com/blockchain.

[20] "What Are Blocks in Blockchain Technology?" Educative: Interactive Courses for Software Developers, www.educative.io/answers/what-are-blocks-in-blockchain-technology.

[21] Agbo, Cornelius, et al. "Blockchain Technology in Healthcare: A Systematic Review." Healthcare, vol. 7, no. 2, MDPI AG, Apr. 2019, p. 56. Crossref, https://doi.org/10.3390/healthcare7020056.

[22] "What Are Blockchain Nodes? Detailed Guide - Blockchain Council." What Are Blockchain Nodes? Detailed Guide - Blockchain Council, 22 Feb. 2022, www.blockchain-council.org/blockchain/blockchain-nodes.

[23] Daly, Lyle. “What Is a Node in Blockchain? | the Motley Fool.” The Motley Fool, www.fool.com/investing/stock-market/market-sectors/financials/blockchain-stocks/blockchain-node.

[24] “How Does a Blockchain Transaction Work? | Ledger.” Ledger, www.ledger.com/academy/how-does-a-blockchain-transaction-work.

[25] Nakamoto, Satoshi. October 31, 2008. Bitcoin: A peer-to-peer electronic cash system

[26] “Bitcoin: A Peer-to-Peer Electronic Cash System: 3. Timestamp Server | Saylor Academy.” Saylor Academy, learn.saylor.org/mod/book/view.php?id=36306&amp;chapterid=18885.

[27] “Consensus Mechanisms | ethereum.org.” ethereum.org, ethereum.org.

[28] “Proof-of-work (PoW) | ethereum.org.” ethereum.org, ethereum.org.

[29] Daly, Lyle. “What Is Proof of Work (PoW)? | the Motley Fool.” The Motley Fool, www.fool.com/investing/stock-market/market-sectors/financials/cryptocurrency-stocks/proof-of-work.

[30] “Proof-of-stake (PoS) | ethereum.org.” ethereum.org, ethereum.org.

[31] “Ethereum Could Turn on Proof of Stake Sooner Than We Anticipate | ConsenSys.” ConsenSys, 11 Mar. 2021, consensys.net/blog/ethereum-2-0/proof-of-stake-is-coming-to-ethereum-sooner-than-we-think.

[32] “Public and Private Keys: What Are They? | Gemini.” Gemini, www.gemini.com/cryptopedia/public-private-keys-cryptography.

[33] “Private Key: What It Is, How It Works, Best Ways to Store.” Investopedia, 17 Feb. 2023, www.investopedia.com/terms/p/private-key.asp.

[34] “The History of Smart Contracts.” The History of Smart Contracts, 1 Oct. 2001, pontem.network/posts/the-history-of-smart-contracts.

[35] “What Are Smart Contracts on Blockchain? | IBM.” What Are Smart Contracts on Blockchain? | IBM, www.ibm.com/topics/smart-contracts.

[36] “Smart Contracts in Blockchain - GeeksforGeeks.” GeeksforGeeks, 7 Jan. 2019, www.geeksforgeeks.org/smart-contracts-in-blockchain.

[37] “Introduction to Dapps | ethereum.org.” ethereum.org, ethereum.org.

[38] Hacken. “Dapps Vs Smart Contracts - Hacken.” Hacken, 19 Aug. 2022, hacken.io/discover/dapps-vs-smart-contracts.

[39] “Decentralised Applications (DApps) - Blockchain Patterns.” Blockchain Patterns, research.csiro.au/blockchainpatterns/general-patterns/deployment-patterns/dapp.

[40] “Solidity &Mdash; Solidity 0.8.19 Documentation.” Solidity &Mdash; Solidity 0.8.19 Documentation, docs.soliditylang.org/en/v0.8.19.

[41] Weston, Georgia. “List of Top Blockchains Using the Rust Programming Language.” 101 Blockchains, 21 Dec. 2022, 101blockchains.com/top-blockchains-using-rust-programming-language.

[42] “What Is a Blockchain Gas Fee? | NFT Gas Fees Explained | Kraken.” What Is a Blockchain Gas Fee?, https://www.kraken.com/learn/what-is-a-blockchain-gas-fee.

[43] “Role of Smart Contracts on Blockchain Explained.” Role of Smart Contracts on Blockchain Explained - Insights | Infosys, www.infosys.com/insights/digital-future/smart-contracts.html.

[44] Team, LCX. “Most Popular Use-Cases of Smart Contracts - LCX.” LCX, 2 Aug. 2022, www.lcx.com/most-popular-use-cases-of-smart-contracts.

[45] Benefits of smart contracts. Newburn Law, P.C. (2022, February 28). Retrieved March 27, 2023, from https://www.newburnlaw.com/benefits-of-smart-contracts

[46] “What Is Polygon (MATIC)?” Coinbase, www.coinbase.com/learn/crypto-basics/what-is-polygon.

[47] Angell, Phoenix. “What Is the Difference Between Ethereum and Polygon?” ScreenRant, Sept. 2022, screenrant.com/difference-between-ethereum-polygon-what.

[48] Abrol, Ayushi. “Solana Vs. Polygon Vs. Ethereum – the Ultimate Comparison.” Blockchain Council, Mar. 2023, www.blockchain-council.org/blockchain/solana-vs-polygon-vs-ethereum.

[49] Ethereum Virtual Machine | Coinbase Help. help.coinbase.com/en/coinbase/getting-started/crypto-education/glossary/ethereum-virtual-machine.

[50] Weston, Georgia. “Transactions in Polygon – Know Everything.” 101 Blockchains, Sept. 2022, 101blockchains.com/transactions-in-polygon.

[51] MATIC | Polygon Wiki. 22 Mar. 2023, wiki.polygon.technology/docs/operate/gas-token.

[52]Z,Afiq.“What IsGas Fee?.” Polygonscan Support Center, support.polygonscan.com/support/solutions/articles/69000793833-what-is-gas-fee-%E2%9B%BD#:~:text=This%20mechanism%20charges%20senders%20of,validators%20and%20smart%20contract%20executions.

[53] Axen, Douglas. “Top Smart Contract Programming Languages for Blockchain Developers.” Moralis Web3 | Enterprise-Grade Web3 APIs, 3 Jan. 2023, moralis.io/top-smart-contract-programming-languages-for-blockchain-developers.

[54] “Anatomy of Smart Contracts | ethereum.org.” ethereum.org, ethereum.org.

[55] “What Is React.js? (Uses, Examples, and More).” What Is React.js? (Uses, Examples, & More), 27 June 2022, blog.hubspot.com/website/react-js.

[56] “About Npm | Npm Docs.” About Npm | Npm Docs, docs.npmjs.com/about-npm.

[57] Documentation. docs.ethers.org/v6.

[58] “web3.js - Ethereum JavaScript API &Mdash; web3.js 1.0.0 Documentation.” web3.js - Ethereum JavaScript API &Mdash; web3.js 1.0.0 Documentation, web3js.readthedocs.io/en/v1.8.2.

[59] “Contract ABI Specification &Mdash; Solidity 0.8.19 Documentation.” Contract ABI Specification &Mdash; Solidity 0.8.19 Documentation, docs.soliditylang.org/en/v0.8.19/abi-spec.html.

[60] “Getting Started With Hardhat | Ethereum Development Environment for Professionals by Nomic Foundation.” Getting Started With Hardhat | Ethereum Development Environment for Professionals by Nomic Foundation, hardhat.org.

[61] “Compiling Smart Contracts | ethereum.org.” ethereum.org, ethereum.org.

[62] “The Crypto Wallet for Defi, Web3 Dapps and NFTs | MetaMask.” The Crypto Wallet for Defi, Web3 Dapps and NFTs | MetaMask, metamask.io.

[63] “How to Add Polygon Mumbai to MetaMask (2023).” DataWallet, www.datawallet.com/crypto/add-polygon-mumbai-to-metamask#:~:text=%E2%80%8D-,What%20is%20Polygon%20Mumbai%3F,applications%20on%20the%20Polygon%20network.

[64] Blockchain Transition, Web 3. 0. “What Is the Blockchain and How Does It Work? | W3BT.” W3BT, 24 Sept. 2019, w3bt.io/what-is-the-blockchain-and-how-does-it-work.

[65] Hassan, Nurul & Jain, Nishchay & Chandna, Vinay. (2018). BLOCKCHAIN, CRYPTOCURRENCY AND BITCOIN.

[66] “Types of Blockchain - GeeksforGeeks.” GeeksforGeeks, 27 Apr. 2022, www.geeksforgeeks.org/types-of-blockchain.

[67] Clincy, Victor A., and H. Shahriar. “Blockchain Development Platform Comparison | Semantic Scholar.” Blockchain Development Platform Comparison | Semantic Scholar, 1 Jan. 2021, www.semanticscholar.org/paper/Blockchain-Development-Platform-Comparison-Clincy-Shahriar/9d1eb2a08236256456fb1f334baadd6226ef7880#citing-papers.

[68] “Blockchain.” NIST, 25 Sept. 2019, www.nist.gov/blockchain.

[69] Vujičić, Dejan & Jagodic, Dijana & Ranđić, Siniša. (2018). Blockchain technology, bitcoin, and Ethereum: A brief overview. 1-6. 10.1109/INFOTEH.2018.8345547.

[70] Ahmad, Mohammad A. Al et al. “Comparison between PoW and PoS Systems Of Cryptocurrency.” Indonesian Journal of Electrical Engineering and Computer Science (2018): n. pag.

[71] “An Overview of Smart Contract: Architecture, Applications, and Future Trends.” An Overview of Smart Contract: Architecture, Applications, and Future Trends | IEEE Conference Publication | IEEE Xplore, ieeexplore.ieee.org/document/8500488.

[72] Sivan, Vishnu. “Create and Deploy Your First Smart Contract With Solidity.” Medium, 9 May 2022, coinsbench.com/create-and-deploy-your-first-smart-contract-with-solidity-92c39987655e.

[73] Petcu, Adrian, et al. “A Secure and Decentralized Authentication Mechanism Based on Web 3.0 and Ethereum Blockchain Technology.” MDPI, 9 Feb. 2023, https://doi.org/10.3390/app13042231.

[74] “Cryptography in Blockchain - GeeksforGeeks.” GeeksforGeeks, 7 May 2022, www.geeksforgeeks.org/cryptography-in-blockchain.

[75] “Alchemy - the Web3 Development Platform.” Alchemy - the Web3 Development Platform, www.alchemy.com.

[76] “Unit Testing for NodeJS Using Mocha and Chai | BrowserStack.” BrowserStack, 28 Apr. 2022, browserstack.wpengine.com/guide/unit-testing-for-nodejs-using-mocha-and-chai.

[77] “Exploring Blockchain-Based Messaging Apps - Moralis Academy.” Moralis Academy, 6 Dec. 2022, academy.moralis.io/blog/exploring-blockchain-based-messaging-apps.

[78] “How to Build Your Own Real-time Chat App Like WhatsApp? | HackerNoon.” How to Build Your Own Real-time Chat App Like WhatsApp? | HackerNoon, hackernoon.com/how-to-build-your-own-real-time-chat-app-like-whatsapp-9d1d058afd5b.

[79] “CryptoJS - CryptoJS.” CryptoJS - CryptoJS, cryptojs.gitbook.io/docs.

[80] “NPM-Init.” Npm Docs, docs.npmjs.com/cli/v9/commands/npm-init. Accessed 24 May 2023.

[81] Petcu, Adrian, et al. “A Secure and Decentralized Authentication Mechanism Based on Web 3.0 and Ethereum Blockchain Technology.” Applied Sciences, vol. 13, no. 4, Feb. 2023, p. 2231. Crossref, https://doi.org/10.3390/app13042231.

LIST OF ABBREVIATIONS

ABI	Application Binary Interface
AES	Advanced Encryption Standard
dApp	Decentralized Application
dApps	Decentralized Applications
DPoS	Delegated Proof of Stake
JSON	JavaScript Object Notation
npm	Node Package Manager
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
P2P	Peer-to-Peer
UI	User Interface
Vscode	Visual Studio Code

LIST OF FIGURES

LIST OF TABLES

APPENDICES

APPENDIX P I: APPENDIX TITLE