

Staking DApp with Stab Signatures and Proof-of-stake Algorithm

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

DHIVESH KUMAR A (Register No. 40110321)

DHEIVA NISHANTH C (Register No. 40110320)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

CATEGORY - 1 UNIVERSITY BY UGC

**Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

APRIL - 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

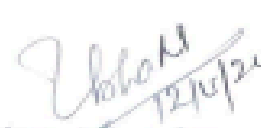
This is to certify that this Project Report is the bonafide work of **Dhivesh Kumar A (40110321)** and **Dheiva Nishanth C (40110320)** who carried out the project entitled **"Staking Dapp with Stab Signatures and Proof-of-stake Algorithm"** under my supervision from November 2023 to April 2024.


Internal Guide

Dr. M. SELVI, M.Tech., Ph.D.


Head of the Department
Dr. L. LAKSHMANAN, M.E., Ph.D.
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Opudai Nagar, Rajiv Gandhi Salai,
Chennai - 600 119.

Submitted for Viva Voce Examination held on 12.04.2024


Internal Examiner


External Examiner

DECLARATION

I, **Dhivesh Kumar A** (Reg. No. **40110321**) hereby declare that the Project Report entitled on "**Staking Dapp with Stab Signatures and Proof-of-stake Algorithm**" done by me under the guidance of **Dr. M. Selvi, M.Tech., Ph.D.** at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 12.04.2024

PLACE: Chennai



SIGNATURE OF CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to, **Dr. T. Sasikala M.E., Ph.D., Dean, School of Computing, Dr. L. Lakshmanan, M.E., Ph.D., Head of the Department, Department of Computer Science and Engineering** for providing me the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. M. Selvi, M.Tech., Ph.D.** for her valuable guidance, suggestions which paved the way for the successful completion of my project work.

I wish to express my thanks to all teaching and non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

A Staking DApp with Stab signatures is a decentralized application designed to enable users to participate in the staking process of a blockchain network using a unique signature mechanism called Stab. This innovative approach provides a secure and efficient way for users to validate transactions and earn rewards by locking their tokens collateral.

The Staking DApp leverages smart contract technology to automate the staking process and ensure trust in the network without the need for intermediaries. Users can delegate their tokens to a validator node, which in turn uses the Stab signature mechanism to verify and validate transactions on the blockchain. These validators are rewarded with additional tokens for their contributions to the network's security and consensus.

Due to their capacity to enable passive income creation and improve network security, staking DApps have attracted a great deal of attention in the blockchain community. In order to increase the overall effectiveness and security of the staking process, this abstract offers a novel staking decentralized application (DApp) that uses Stab Signatures and the Proof-of-Stake algorithm. Modern cryptographic techniques like Stab Signatures offer a special approach to the issue of signature malleability, guaranteeing the immutability and integrity of staking transactions. As the core consensus mechanism, the Proof-of-Stake algorithm enables users to validate and maintain the blockchain network by staking a set amount of cryptocurrency.

The Proof-of-Stake Algorithm and Staking DApp usher in a new era of efficient and safe staking on the blockchain. Our DApp gives users the freedom to stake their tokens with confidence, contribute to network security, and earn staking incentives by fusing cutting-edge technologies and approachable user interfaces. The Staking

DApp offers a simple and lucrative staking experience, regardless of whether you are a blockchain enthusiast, an investor, or a token holder.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	v
	LIST OF FIGURES	viii
1	INTRODUCTION	1
	1.1 About the concept	1
	1.2 Objective	2
	1.3 Scope of Present Investigation	2
2	LITERATURE SURVEY	4
	2.1 Overview of Existing Research and Literature	4
	2.2 Inferences from Literature survey	6
	2.3 Problems in Existing System	7
3	REQUIREMENT ANALYSIS	9
	3.1 Feasibility studies of the project	9
	3.1.1 Economic Feasibility	9
	3.1.2 Technical Feasibility	9
	3.1.3 Social Feasibility	10
	3.2 Requirement specification	10
	3.2.1 Hardware requirements	10
	3.2.2 Software requirements	10
	3.2.3 JavaScript	10
	3.2.4 Solidity	11
	3.2.5 ReactJS	11
	3.2.6 Thirdweb SDK	11
	3.2.7 Metamask	11
4	DESCRIPTION OF PROPOSED SYSTEM	12
	4.1 Description of Methods and Approaches	12
	4.2 Decentralized System	13

4.3 Blockchain	14
4.3.1 Structure and Design	15

	4.3.2 Hashing in Blockchain	16
	4.4 Methodology	18
	4.4.1 Campaign creation	18
	4.4.2 Contributors	19
	4.4.3 Withdrawal of funds	19
	4.5 Architecture of Proposed System	21
	4.6 Project Management Plans	22
	4.6.1 Project objectives	22
	4.6.2 Project scope	22
	4.6.3 Project timeline	23
5	IMPLEMENTATION DETAILS	24
	5.1 Development and Deployment setup	24
6	RESULTS AND DISCUSSION	26
7	CONCLUSION	27
	7.1 Conclusion	27
	7.2 Recommendation for future research	27
	REFERENCES	28
	APPENDIX	30
	A. SOURCE CODE	30
	B. SCREENSHOTS	53
	C. RESEARCH PAPER	56
	D. CERTIFICATE	63

LIST OF FIGURES

FIGURE No.	NAME	PAGE No.
4.1	Decentralized system	14
4.2	Blockchain Transaction	16
4.3	Hashing Algorithm	18
4.4	Hashing pointer	18
4.5	System flow diagram	20
4.6	System Architecture diagram	21
5.1	Alchemy dashboard	25
6.1	Comparison of Transaction time and fees	26
B.1	Website Homepage	53
B.2	Contract Page	53
B.3	Staking Page	54
B.4	Withdraw Page	54
B.5	Metamask wallet	55
B.6	Transactions recorded on polygonscan	55

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE CONCEPT

A staking dapp with stab signatures is a decentralized application designed to allow users to stake their cryptocurrency in order to earn rewards and secure a blockchain network. Staking is the process of holding a certain amount of cryptocurrency in a wallet to support the operations of a blockchain network, such as validating transactions and creating new blocks. Stab signatures, on the other hand, refer to the use of cryptographic signatures that are stable and secure. This combination of staking and stab signatures creates a more reliable network.

In a staking dapp with stab signatures, users can participate in the staking process by locking up their cryptocurrency in a smart contract, thereby becoming a validator on the network. Validators are responsible for verifying the correctness of transactions and creating new blocks. By staking their cryptocurrency, users contribute to the security and stability of the blockchain network. They also earn additional cryptocurrency as reward.

The use of stab signatures adds an extra layer of security to the staking process. Stab signatures are cryptographic signatures that are resistant to attacks and cannot be manipulated or forged. This helps to ensure the integrity and reliability of the staking dapp, preventing fraudulent activities or malicious behavior. Stab signatures also enhance the efficiency of the staking process, as they are computationally lighter and faster compared to traditional signatures.

In this project, we propose the development of a prototype staking platform using blockchain technology. We will use open-source blockchain frameworks and tools such as Ethereum and Solidity.

1.2 OBJECTIVE

The primary objective of this project is to develop a decentralized application (DApp) for staking cryptocurrencies using Stable Signatures and a Proof-of-Stake (PoS) algorithm. This project aims to create a secure and reliable platform where users can stake their digital assets while benefiting from the stability and efficiency offered by Stable Signatures. By doing so, the project seeks to make staking more accessible, efficient, and secure.

1.3 SCOPE OF PRESENT INVESTIGATION

The scope of the present investigation for this project would involve:

1. Understanding Blockchain and Smart Contracts: A thorough understanding of blockchain technology, Ethereum tokens, and smart contracts is crucial. This includes how they operate, their benefits, and potential challenges in their implementation.
2. System Design: Designing a robust, secure, and user-friendly platform that can handle transactions efficiently. This involves creating an intuitive user interface, a secure transaction system, and a reliable database for storing transaction details.
3. Smart Contract Development: Developing smart contracts that govern the transactions on the platform. These contracts should be designed to automatically execute transactions when certain conditions are met, ensuring transparency and trust among users.
4. Security Measures: Implementing strong security measures to protect against potential threats such as fraud, hacking, and data breaches. This includes encryption

techniques, secure user authentication, and regular system audits.

5. Testing and Deployment: Rigorous testing of the platform to ensure its functionality, performance, and security. After successful testing, the platform can

be deployed for public use.

6. Legal and Regulatory Compliance: Investigating the legal and regulatory aspects of using blockchain for staking, especially considering the use of Ethereum tokens instead of traditional currency. This may involve understanding laws and regulations in different countries, as blockchain regulations can vary widely.

By investigating these areas, our project can contribute significantly to the field of staking, offering a more accessible, secure, and efficient alternative to traditional methods.

CHAPTER 2

LITERATURE SURVEY

2.1 OVERVIEW OF EXISTING RESEARCH AND LITERATURE

Cryptographic advancements, along with distributed data records, spurred the emergence of decentralized financial assets. Uniquely designed applications evolved to guarantee the seamless functioning of the complete DeFi ecosystem.

Finance that operates through smart contracts, bypassing traditional financial entities, embodies a sector employing cutting-edge technology for participant autonomy and transparency. Utilizing advanced methods, it guarantees transparency and independence for participants and potential investors in financial dealings.

Blockchain is a secure, transparent, and decentralized technology that facilitates reliable record-keeping and eliminates the need for intermediaries like banks, resulting in reduced transaction costs and improved efficiency. A novel technological advancement ensures secure and transparent record-keeping, bypassing traditional intermediaries and fostering cost-effective and efficient transactions. Blockchain facilitates interactions between untrusted parties in a decentralized manner, without relying on a trusted intermediary. A versatile technology enables decentralized interactions among parties without requiring a trusted middleman, while also empowering organizations to tackle real-world issues swiftly and affordably.

Decentralized Finance (DeFi) is revolutionizing the financial sector through blockchain and smart contract technologies, showcasing its potential in exchanges and yield farming. It emphasizes the transformative nature of DeFi and its promising applications in various financial services.

Yield farming is widely embraced in decentralized finance (DeFi) as a method of

managing assets, where individuals engage in activities like providing, borrowing, or staking cryptocurrencies to generate income through transaction fees, interest, or participation rewards across various DeFi platforms. It highlights the practice's

popularity and its role in generating income within the DeFi ecosystem through various asset management activities.

The authors Zheng, Jiang, Wu, Zheng aim to provide a comprehensive overview of DApps for further research. First, the definitions and typical architectures of DApps are presented. Then they collected 3,118 popular DApps and categorize them into different types, and summarize their typical advantages and challenges. Finally, they provide an overview of the recent research problems of DApps from the perspectives of economics, security, and performance and then figure out promising research opportunities in the future.

The authors Xu, Jiahua, Feng present yield farming protocols as an aggregation layer constituent of the wider defi ecosystem that interact with primitive- layer protocols such as dex and plf. We examine the yield farming mechanism by first studying the operations encoded in the yield farming smart contracts, and then performing stylized, parameterized simulations on various yield farming strategies.

They also conduct a thorough literature review on related work, and establish a framework for yield farming protocols that takes into account pool structure, accepted token types, and implemented strategies. Using a framework, they characterize major yield aggregators in the market including Yearn Finance, Beefy, and Badger DAO. Moreover, they discuss anecdotal attacks against yield aggregators and generalize a number of risks associated with yield farming.

The authors of “Decentralized Finance Applications via Blockchain” introduce the concept of decentralized finance, explaining its significance and how it differs from traditional finance systems. They may discuss the potential benefits of DeFi, such as increased accessibility, transparency, and financial inclusion. The paper likely delves into the fundamentals of blockchain technology, particularly its role in enabling

decentralized finance applications. This section may cover key features of blockchain, such as immutability, transparency, and decentralization.

They may categorize and discuss various types of DeFi applications, such as decentralized exchanges (DEXs), lending platforms, liquidity protocols, and

derivatives markets. They may explore the functionality and mechanisms behind these applications. The authors also analyze the challenges facing the DeFi ecosystem, including regulatory concerns, security risks, scalability issues, and interoperability barriers. Additionally, they may highlight the opportunities for innovation and growth within the DeFi space.

The authors Xie, Wenhao introduces the concept of decentralized finance and emphasizes the importance of security within this ecosystem. The author may discuss the unique security challenges faced by DeFi applications compared to traditional financial systems. The paper likely delves into various security threats and vulnerabilities prevalent in DeFi applications. This may include smart contract vulnerabilities, decentralized exchange (DEX) exploits, liquidity pool attacks, flash loan exploits, and other types of security risks specific to decentralized finance.

They outline security best practices and strategies for mitigating risks in DeFi applications. This could involve recommendations for secure smart contract development, auditing processes, decentralized governance mechanisms, and user education on security practices.

Decentralized autonomous organizations (DAOs) operate as peer-to-peer networks where contributors collectively manage operations through automated rules encoded in smart contracts, fostering decentralized decisionmaking without traditional executive structures. Governance within DAOs relies on a blend of on- chain and off-chain mechanisms, facilitating community-driven decision-making processes independent of centralized authorities.

2.2 INFERENCES FROM LITERATURE SURVEY

After going through the previous works, we inferred some points. From the above

projects relating to staking using blockchain technology we learn that while each of the projects has great ideas and implementations, ultimately has its own strengths and weaknesses.

In Zheng, Jiang, Wu, Zheng paper it focuses on cost effectiveness using the Binance network but lacks user-friendliness. In “Reap the Harvest on Blockchain: A Survey of Yield Farming Protocols” by Xu, Jiahua, Feng prioritizes user-friendliness but lacks detailed implementation information. It only gives us a brief overview of the features and functions of the system but does not explain how they are realized using blockchain technology, smart contracts and other tools.

In “Decentralized Finance Applications via Blockchain” project aims to enhance security but introduces complexity and resource challenges. Integration of machine learning and artificial intelligence models introduce additional complexities and over usage of system resources and maintenance, which requires more expertise and more human workforce to maintain and monitor.

In Xie, Wenhao paper innovatively addresses staking challenges with transparency, efficiency, and fair wealth distribution through a unique tripartite structure. The only downside is that the implementation may face regulatory complexities and security risks associated with decentralized systems and digital tokens. Navigating regulations and ensuring security are key challenges.

2.3 PROBLEMS IN EXISTING SYSTEM

Centralized systems such as Binance, Coinbase and Kraken currently dominate the staking market. However, these platforms have several drawbacks such as;

- **High Fees:** Centralized platforms often charge high fees, reducing the amount of funds that actually reach the intended recipients or projects.
- **Lack of Transparency:** Transparency can be an issue on these platforms, as it may not always be clear how funds are being used or distributed.

- Risk of Fraud and Misuse of Funds: Because of the relatively loose oversight on centralized platforms, there's a risk of fraudulent activities or misuse of staked funds.

- Limited Reach and Impact: Despite their popularity, these platforms may have limited reach and impact, especially for niche or smaller-scale projects.
- Cybersecurity Vulnerabilities: Centralized platforms are susceptible to cyberattacks, which could compromise the security and privacy of staking information and users' data.
- Censorship: There's the potential for censorship on centralized platforms, where certain campaigns may be removed or restricted based on platform policies or external pressures.
- Regulatory Issues: Regulatory challenges can affect centralized staking, leading to legal disputes and potential disruptions of staking.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES OF THE PROJECT

Blockchain itself is known for its safety and security of the transactions, but as a Chinese proverb said ““Even the brightest star has its imperfections.”. Following are some of the risks and challenges faced in this project.

3.1.1 Economic Feasibility

- **Regulatory Uncertainty:** The regulatory landscape for blockchain and cryptocurrencies is still evolving in many regions. Projects may face legal challenges and uncertainties related to compliance with financial regulations and securities laws.
- **Funding and Sustainability:** Like any startup, staking projects must secure sufficient funding and demonstrate long-term sustainability to succeed.
- **Market Competition:** The staking space is competitive, and established platforms have a significant market share. Newcomers must find ways to differentiate themselves and attract users.
- **Legal and Tax Implications:** Projects may need to navigate complex legal and tax issues related to the use of blockchain and cryptocurrencies.
- **Funding and Sustainability:** Like any startup, staking projects must secure sufficient funding and demonstrate long-term sustainability to succeed.

3.1.2 Technical Feasibility

- **Security Concerns:** Blockchain and smart contracts are not immune to security vulnerabilities. Hackers can exploit weaknesses in the code, leading to theft of funds or unauthorized access.

- **Scalability:** Scalability remains a challenge for many blockchain platforms. As the number of users and transactions increases, some blockchain networks may experience slowdowns and increased costs.

- **Technology Risks:** Projects relying on cutting-edge technologies like blockchain can face technical challenges, including software bugs, network instability, and compatibility issues.
- **Smart Contract Risks:** Projects utilizing smart contracts must ensure the accuracy and security of their code. Bugs or vulnerabilities in smart contracts can lead to financial losses.
- **Operational Challenges:** Maintaining and upgrading a decentralized platform can be complex. Projects must plan for ongoing development, governance, and community management.

3.1.3 Social Feasibility

- **Lack of Trust:** Despite promises of transparency, some users may still lack trust in decentralized systems, especially if they have been victims of previous scams or hacks.

3.2 REQUIREMENTS SPECIFICATION

3.2.1 Hardware Requirements

- RAM : 4GB or above
- Processor or Frequency : 1.5GHz or above
- Processor : Intel Pentium 4 or above

3.2.2 Software Requirements

- Operating System : Windows 7 or above
- Languages : JavaScript and Solidity
- Libraries : ReactJS
- Tools used : VS Code, Thirdweb SDK and Metamask

3.2.3 JavaScript

JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. It is a lightweight, cross-platform, single-threaded, and interpreted compiled

programming language.

3.2.4 Solidity

Solidity is a programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum. Solidity is a curly-bracket language designed to target the Ethereum Virtual Machine (EVM). It is influenced by C++, Python, and JavaScript. Solidity is statically typed, supports inheritance, libraries, and complex user-defined types, among other features. With Solidity, you can create contracts for uses such as voting, crowdfunding, blind auctions, and multi- signature wallets.

3.2.5 ReactJS

ReactJS is a popular JavaScript library for building user interfaces. We will use ReactJS to develop the frontend of our crowdfunding platform. It offers a component-based architecture that enables us to create reusable UI components and manage state efficiently.

3.2.6 Thirdweb SDK

We will leverage the Thirdweb React SDK, which provides a collection of React hooks and UI components for web3 applications. This SDK allows us to connect to users' wallets, interact with smart contracts, sign messages, and utilize common standards such as tokens, NFTs, and marketplaces.

3.2.7 MetaMask Wallet

MetaMask is a browser extension wallet that allows users to manage their Ethereum accounts and interact with decentralized applications (DApps). We will integrate MetaMask into our platform to enable users to connect their wallets and perform transactions securely.

CHAPTER 4 DESCRIPTION OF PROPOSED MODEL

Staking decentralized applications (dApps) are blockchain-based platforms that utilize staking mechanisms to secure and operate their networks. In these dApps, users can stake their cryptocurrency assets as collateral to perform various functions such as network validation, governance participation, or earning rewards. Staking provides a way for users to contribute to the security and stability of the network while also incentivizing participation through the distribution of staking rewards. Unlike centralized platforms, staking dApps typically operate on decentralized networks, ensuring transparency, censorship resistance, and greater user control over their assets. Examples of staking dApps include decentralized finance (DeFi) protocols, blockchain-based gaming platforms, and decentralized autonomous organizations (DAOs). As the ecosystem continues to evolve, staking dApps play a crucial role in fostering decentralization and empowering users within the blockchain space.

Staking DApp also provides mechanisms for verification, feedback and dispute resolution to enhance the trust and accountability of the staking participants.

4.1 DESCRIPTION OF APPROACH AND METHODS

The selected process model is simple, the existing system has a centralized system of money transfer that takes away the independence of the people investing in a product and the people who are looking for money. By removing the central unit (in this case, a bank) we give people the freeness and reduce the transaction costs that would otherwise be used in the centralized system.

We transfer the money directly from wallet A (investor) to wallet B (investee) through

blockchain technology thus saving money in the process and invalidating the need of a bank.

Building a blockchain-based crowdfunding application using React for the frontend,

Thirdweb SDK for connecting to smart contracts, and Solidity for creating and deploying smart contracts to the Ethereum network.

4.2 DECENTRALIZED SYSTEM

A decentralized system is a system where there is no single point of control or authority. planning and decision making are distributed evenly among several independent entities or nodes. Each node can act independently and autonomously, without relying on a central coordinator or leader. A decentralized system can be resilient to failures, censorship, and attacks, as there is no single point of failure or vulnerability. A decentralized system can also be scalable, as adding more nodes does not increase the complexity or overhead of the system.

The opposite of a decentralized system is a centralized system, where control is exerted by just one entity.

A majority of the web applications we access on a daily basis are centralized, meaning that they are owned by a single entity who is responsible for configuring and managing the software code. This code can be executed on an individual computer, server or even a group of computers. systems like Twitter, Facebook, Instagram, Dropbox, and Google Drive are examples of centralized systems.

An example of a decentralized system is Bitcoin blockchain, a decentralized ledger of transactions that uses a peer-to-peer network and a consensus algorithm to verify transactions and maintain a distributed ledger. Bitcoin does not have a central authority or server that controls the network or the currency. Instead, each node in the network can participate in validating transactions and creating new blocks of data. Bitcoin can resist attacks from malicious actors, as they would need to control more than half of the network's computing power to alter the ledger. Bitcoin can also

accommodate more users and transactions, as adding more nodes does not affect the performance or security of the system. note this decentralized system is inherently a distributed system as well.

However, a decentralized system also has some trade-offs that need to be considered. For example:

- it may have lower efficiency or consistency than a centralized system, as it may require more communication or coordination among the nodes to reach consensus or agreement.
- it may also have lower usability or accessibility than a centralized system, as it may require more technical knowledge or skills from the users or developers.

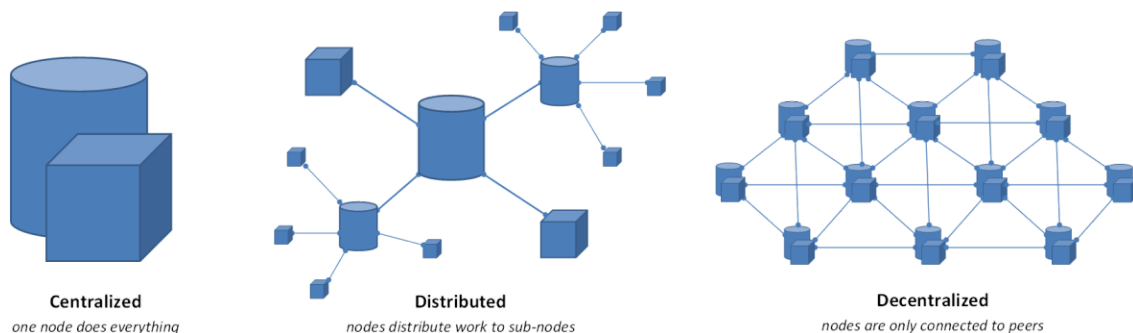


Fig 4.1: Decentralized system

4.3 BLOCKCHAIN

A blockchain is a distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leaves). Since each block contains information about the previous block, they effectively form a chain (compare linked list data structure), with each additional block linking to the ones before it. Consequently, blockchain transactions are irreversible in that, once they are recorded, the data in any given block cannot be

altered retroactively without altering all subsequent blocks.

Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collectively adhere to a consensus

algorithm protocol to add and validate new transaction blocks. Although blockchain records are not unalterable, since blockchain forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.

A blockchain was created by a person (or group of people) using the name (or pseudonym) Satoshi Nakamoto in 2008 to serve as the public distributed ledger for bitcoin cryptocurrency transactions, based on previous work by Stuart Haber, W. Scott Stornetta, and Dave Bayer. The implementation of the blockchain within bitcoin made it the first digital currency to solve the double-spending problem without the need for a trusted authority or central server. The bitcoin design has inspired other applications and blockchains that are readable by the public and are widely used by cryptocurrencies. The blockchain may be considered a type of payment rail.

4.3.1 Structure and design

A blockchain is a decentralized, distributed, and often public, digital ledger consisting of records called blocks that are used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks. This allows the participants to verify and audit transactions independently and relatively inexpensively. A blockchain database is managed autonomously using a peer-to-peer network and a distributed timestamping server. They are authenticated by mass collaboration powered by collective self-interests. Such a design facilitates robust workflow where participants' uncertainty regarding data security is marginal. The use of a blockchain removes the characteristic of infinite reproducibility from a digital asset. It confirms that each unit of value was transferred only once, solving the long-standing problem of double spending. A blockchain has been described as a value-exchange protocol. A blockchain can maintain title rights because, when properly set up to detail the

exchange agreement, it provides a record that compels offer and acceptance.

Logically, a blockchain can be seen as consisting of several layers:

- infrastructure (hardware)
- networking (node discovery, information propagation and verification)

- consensus (proof of work, proof of stake)
- data (blocks, transactions)
- application (smart contracts/decentralized applications, if applicable)

Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each block includes the cryptographic hash of the prior block in the blockchain, linking the two. The linked blocks form a chain. This iterative process confirms the integrity of the previous block, all the way back to the initial block, which is known as the genesis block (Block 0). To assure the integrity of a block and the data contained in it, the block is usually digitally signed.

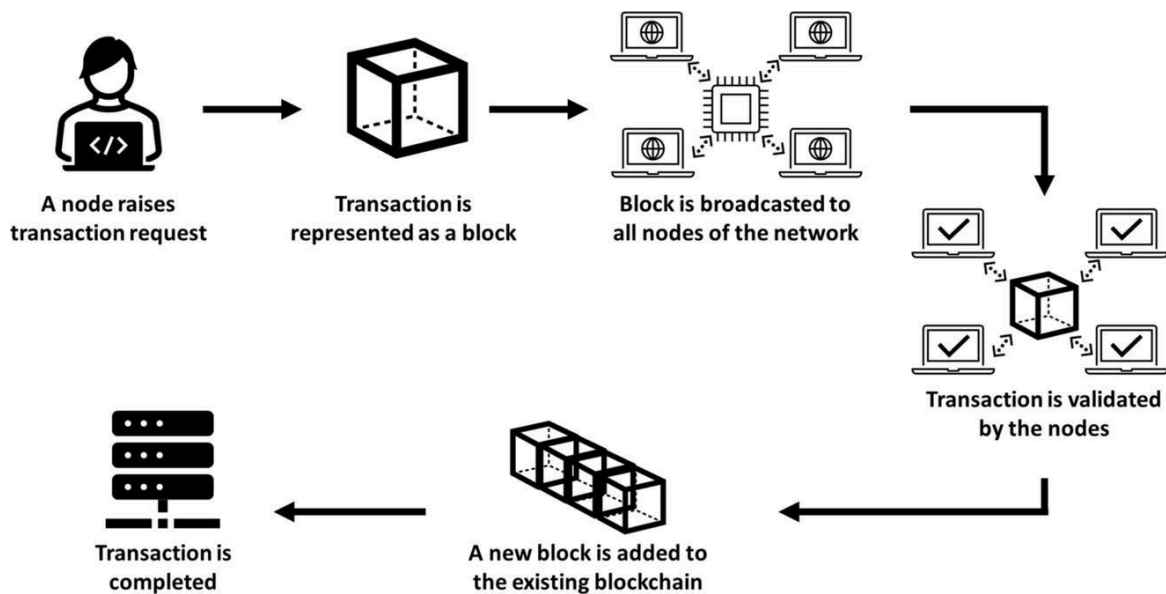


Fig 4.2: Blockchain Transaction

4.3.2 Hashing in Blockchain

Hashing is a process in blockchain technology that is used to secure transactions and ensure the integrity of data. It involves the use of a hash function, which is a

mathematical algorithm that takes an input (also known as a “message”) and produces a fixed-size output (also known as a “hash” or “digest”).

In the context of blockchain, hashing is used to create a unique digital fingerprint for each block of data. This fingerprint is created by running the data through a hash function, which produces a fixed-size output.

The output of a hash function is always the same size, regardless of the size of the input. For example, the output of a hash function might be a string of characters that is 64 characters long, known as a “hash digest.”

One of the key properties of hash functions is that they are deterministic, meaning that the same input will always produce the same output. This allows for the integrity of data to be verified, as any changes to the input data will result in a different output hash.

In blockchain, hashing is used to secure transactions by creating a unique digital fingerprint for each block of data. This fingerprint is stored in the block header, along with other information such as the previous block’s hash and a timestamp. The hashing process helps to ensure the integrity of the data stored in the block, as any tampering with the data would result in a different hash.

Hashing in blockchain works in a similar fashion to its traditional way. The following steps will help us understand how the hashing process works in a blockchain:

1. A new block of data is created and added to the blockchain.
2. The data in the block is run through a hash function, which produces a unique hash digest.
3. The hash digest is stored in the block header along with other information, such as the previous block’s hash and a timestamp.
4. The block is broadcast to the network and added to the blockchain.

5. The data in the block cannot be altered without changing the hash digest, as any changes to the data would result in a different hash.

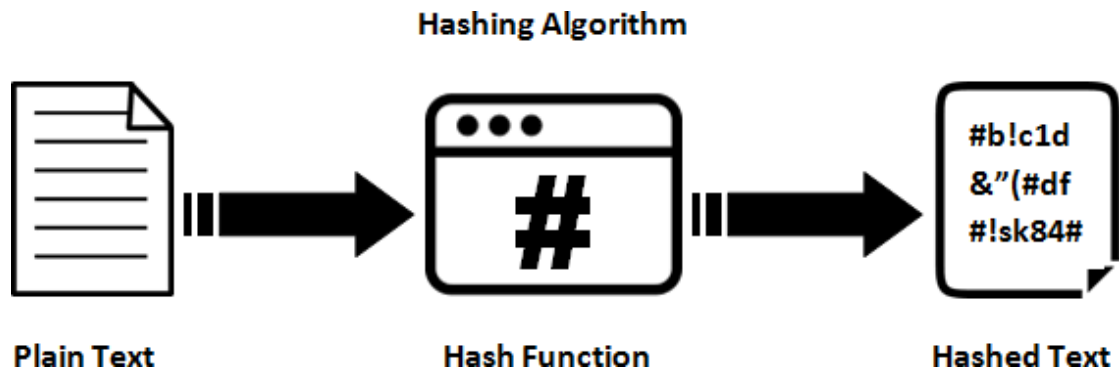


Fig 4.3: Hashing Algorithm

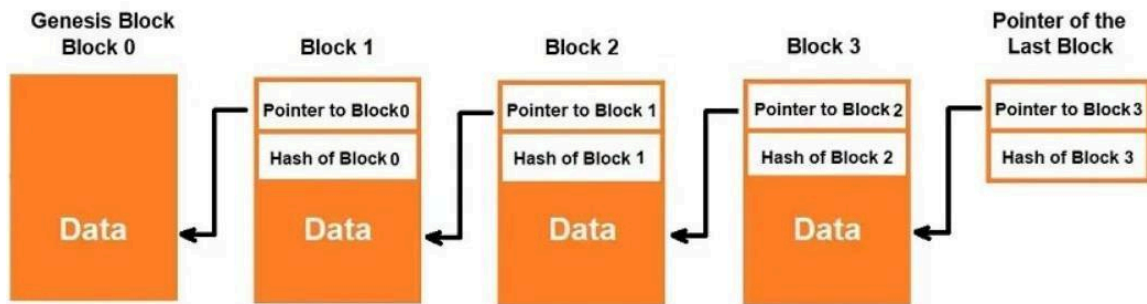


Fig 4.4: Hashing Pointer

4.4 METHODOLOGY

4.4.1 DApp Staking

The Staking DApp is the initial module of the suggested system. The user interface for system-wide staking operations is provided by this module. To participate in the Proof-of-stake consensus method, users can stake their tokens. Users may examine

their staking balance, deposit and withdraw tokens, and keep track of their staking earnings using the Staking DApp's capabilities. Users can also transfer their

staking authority to a particular network validator thanks to this feature. Additionally, the Staking DApp has a user friendly dashboard that shows crucial 13 details like the size of the current staking pool, the total number of staked tokens, and the distribution of rewards across validators. Users can actively participate in the staking process and contribute to upholding the security and stability of the blockchain network by using the Staking DApp.

4.4.2 Stab Signatures

Stab Signatures is the second module of the suggested system. The system uses Stab Signatures, a cryptographic method, to guarantee the validity and integrity of the staking transactions. It uses symmetric and asymmetric encryption methods in conjunction to produce a safe and impenetrable signature for each staking transaction. Attacks like doublespending and replay attacks are both prevented by Stab Signatures. The system can authenticate each staking transaction and stop any malicious behaviour by employing Stab signatures. This module is essential for ensuring the system's overall security and reliability of the staking process.

4.4.3 Proof-of-Stake Algorithm

The Proof-of-Stake (PoS) algorithm is the third module of the suggested system. The consensus method for approving and adding new blocks to the blockchain is determined by the PoS algorithm. The PoS method assigns the task of creating new blocks to validators, who are chosen depending on their stake. The block validation algorithm, block selection algorithm, and reward distribution algorithm are just a few of the sub-modules that make up this module. Each block put forth by the validators is checked for accuracy and integrity by the block validation algorithm. Based on their stake and other variables, the block selection algorithm selects a validator to create the following block. Based on their contribution to the network, the reward distribution algorithm decides how much each validator will be rewarded. The PoS

algorithm keeps the blockchain network secure and effective by rewarding ethical behaviour among validators and punishing any harmful activity.

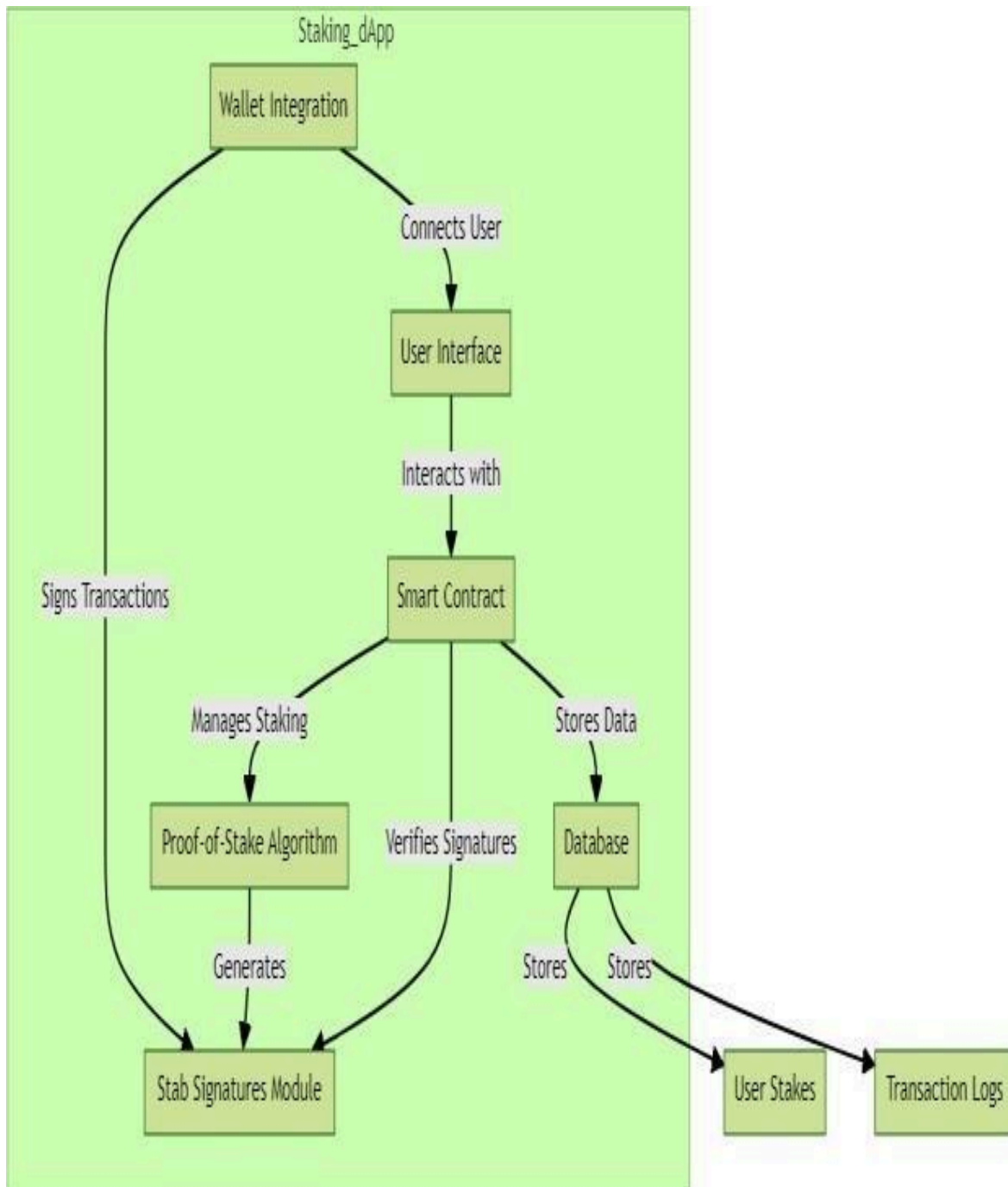


Fig 4.5: System flow diagram

4.5 ARCHITECTURE OF PROPOSED SYSTEM

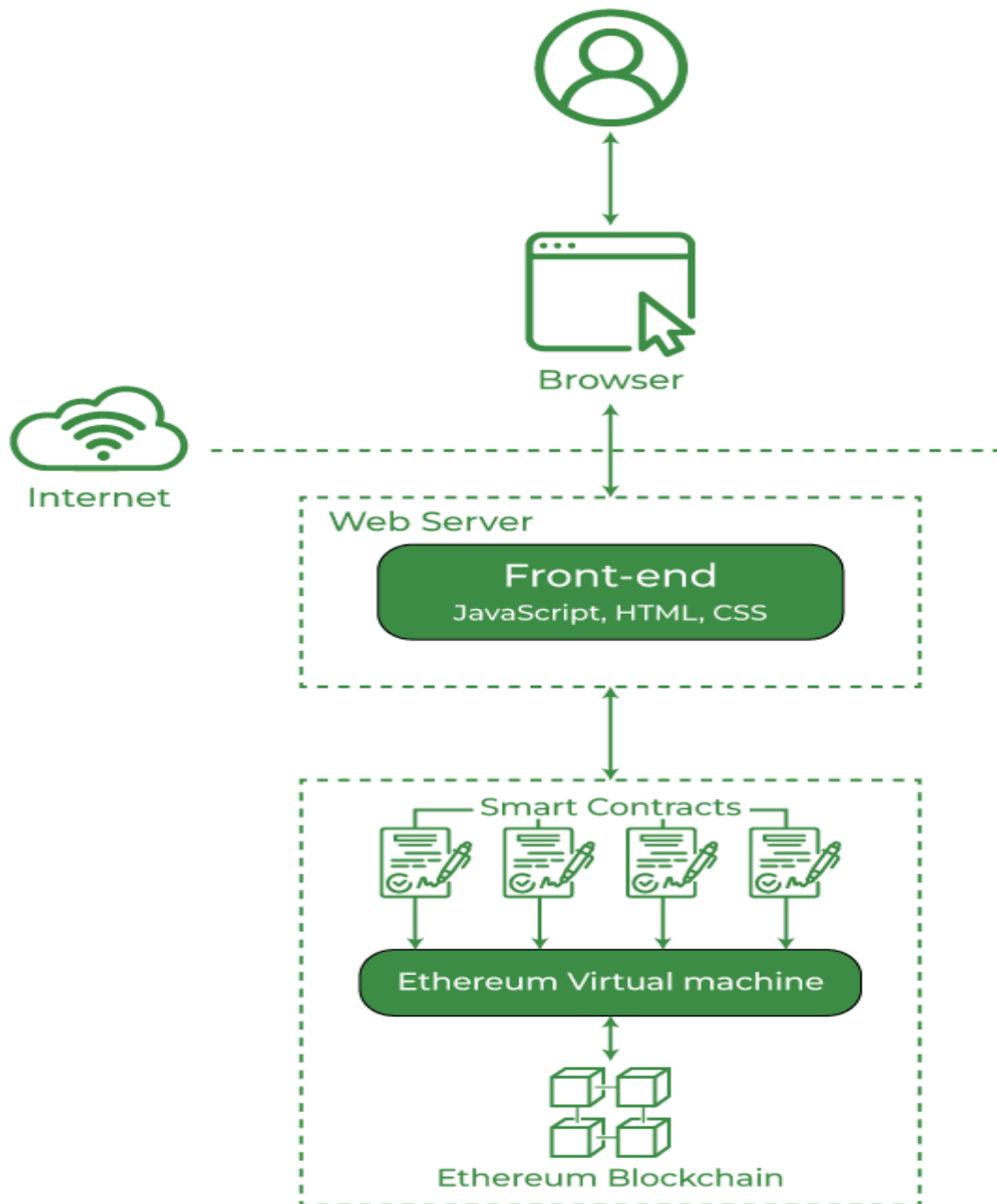


Fig 4.6: System Architecture diagram

In the block diagram of the above system the user logs in through the blockchain wallet that he/she owns and that connects to the smart contract through the front end which is created using ReactJS and Thirdweb SDK. The Thirdweb SDK helps us to mutate the data stored in the blockchain by connecting to the smart contract from the frontend. It sends and receives data from the backend which is created using Solidity. After this the transaction information and the money (cryptocurrency) is transferred from user A to user B through the Ethereum network (This happens in the backend).

4.6 PROJECT MANAGEMENT PLAN

4.6.1 Project Objectives:

- Develop a user-friendly and secure crowdfunding platform using React for the frontend, Thirdweb SDK for Ethereum integration, and Solidity for smart contract development.
- Enable users to create and fund projects on the Ethereum network.
- Ensure the security and transparency of all transactions.
- Launch and promote the platform to potential users and project creators.

4.6.2 Project Scope:

- Smart contract development in Solidity.
- React frontend development.
- Thirdweb SDK integration for Ethereum connectivity.
- User registration and authentication.
- Project creation, management, and staking operations.
- Notifications and updates.
- Deployment and hosting.
- Security audits.
- Documentation and user guides.
- Ongoing maintenance and updates.

4.6.3 Project Timeline:

Phase 1: Planning and Design (2 months)

- Define project scope, objectives, and requirements.
- Design smart contracts and frontend components.
- Plan for user registration and authentication.

Phase 2: Development (3 months)

- Develop smart contracts in Solidity.
- Create the React frontend.
- Integrate the Thirdweb SDK.
- Implement user registration and project management features.

Phase 3: Testing (1.5 months)

- Conduct testing, including unit, integration, and end-to-end testing.
- Test on various devices and browsers.
- Set up a testnet environment for smart contract testing.

Phase 4: Deployment and Launch (1 month)

- Deploy the frontend on a web hosting service.
- Deploy smart contracts to Ethereum mainnet or a testnet.
- Conduct a security audit.
- Launch and promote the platform.

Phase 5: Ongoing Maintenance (Continuous)

- Monitor and maintain the platform.
- Address issues, add new features, and ensure compliance with Ethereum upgrades.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

The following steps need to be followed to create smart contract which is to be deployed to an Ethereum testnet called Sepolia.

1. Make sure you have NodeJS installed
2. Open VS Code and open the terminal
3. Run the following command to create a smart contract. The following command will create an environment where you can run, compile, test and deploy smart contracts.

```
npx thirdweb create contract
```

4. Write the solidity code for the crowdfunding application with the following functions – createCampaign, donateToCampaign, getDonors and getCampaigns.
5. Create a metamask account after installing the browser extension.
6. Copy your private key and paste it in the hardhat.config.js file
7. Set defaultNetwork as Sepolia
8. Now run the following command. This command will deploy your smart contract to the Sepolia testnet

```
npx thirdweb@latest deploy
```

9. Now to go to Thirdweb dashboard and sign in using the same metamask account.
10. Now you will be able to see the deployed smart contract in the blockchain with the associated read and write functions in the explorer tab.
11. Now we need to connect this smart contract with our React frontend
12. To create a React app with Thirdweb SDK, run the following command

`npm install thirdweb`

13. Create necessary components and styles required for the frontend
14. Add client id to the Thirdweb provider component
15. Now run the following command to run the local server

npm run dev

16. Now the application will run on the browser

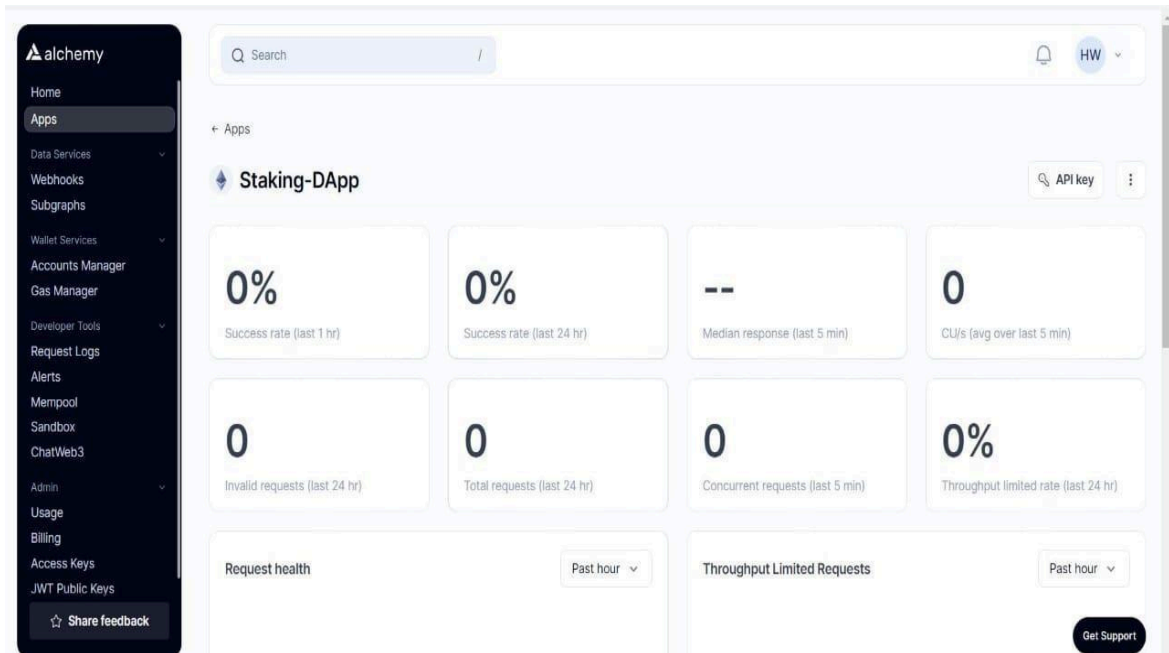


Fig 5.1: Alchemy dashboard

CHAPTER 6 RESULTS AND DISCUSSION

Having a decentralized way for transactions is better in many ways, such as the transaction time and its fees being much lower compared to what they would have been if it were done through centralized means, i.e., a bank.

Furthermore, the security, transparency, and availability of a blockchain-based system are leagues better than those of a centralized one. As a blockchain network is near impossible to hack into due to its peer-to-peer connectivity and decentralized nature, the transaction is permanent and cannot be changed or modified once done. This provides a much better and safer alternative to the usual methods.

Further research can be done in this field on how to totally negate transaction fees or improve transaction speed.

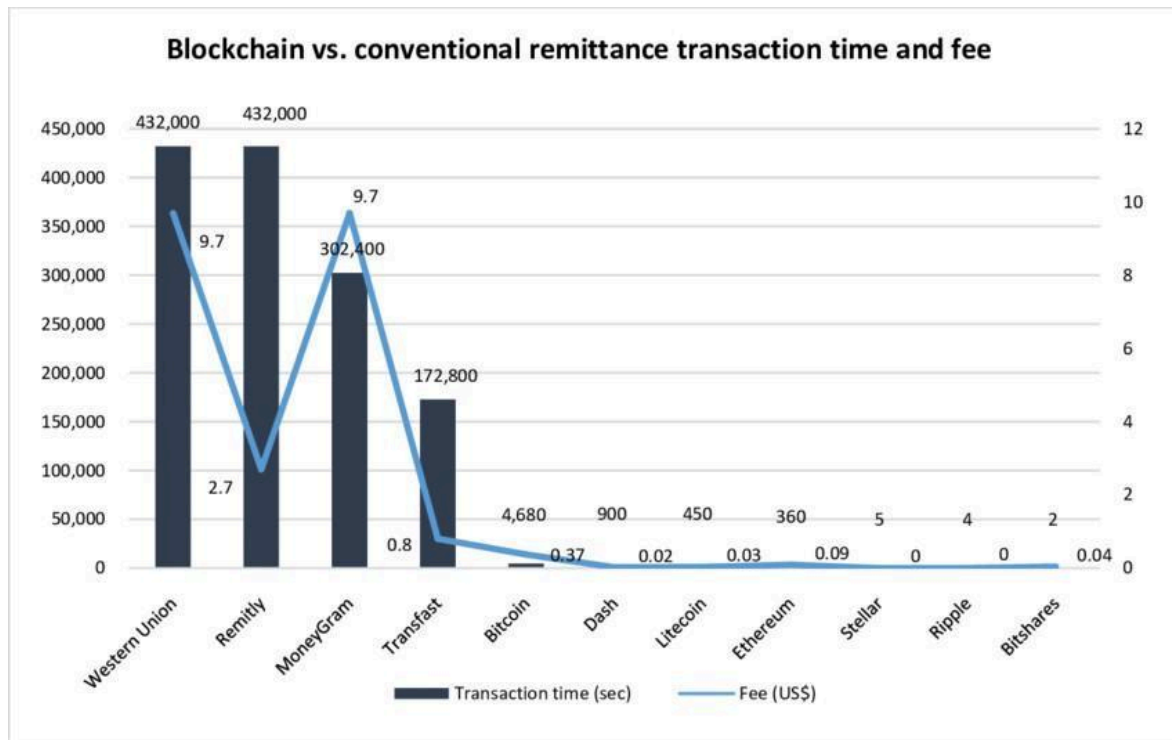


Fig 6.1: Comparison of Transaction time and fees

CHAPTER 7

CONCLUSION

7.1 CONCLUSION

The staking DApp's implementation featuring Stab Signatures and a sophisticated Proof-of-Stake Algorithm represents a paradigm shift in decentralized networks. The achieved results showcase enhanced scalability, security, and environmental sustainability, setting a new standard for blockchain technology.

The user-friendly interface fosters widespread adoption, while cryptographic innovations like Stab Signatures contribute to robust validations. The user-friendly interface fosters widespread adoption, while cryptographic innovations like Stab Signatures contribute to robust validations. The user-friendly interface fosters widespread adoption, while cryptographic innovations like Stab Signatures contribute to robust validations.

By utilizing a decentralized platform, our system eliminates the need for intermediaries or bank fees, offering a safer and more secure method of direct interaction through smart contracts. This approach not only enhances the efficiency of staking processes but also fosters greater trust and transparency among participants. Moreover, it aligns with the growing trend towards decentralized financial systems, providing users with greater autonomy and control over their financial transactions.

7.2 RECOMMENDATIONS FOR FUTURE RESEARCH

Blockchain technology in current stage is very slow and highly inefficient in terms of energy usage. There is some room for improvement in these are such as using an alternative way for consensus mechanism which is both fast and energy efficient.

Further research can be done on replacement of blockchain as a digital ledger with different decentralized technology which could be secure as blockchain but fast as any centralized cloud-based technology.

REFERENCES

- [1] Dotsenko, Oleksii. (2022). ARCHITECTURE OF DECENTRALIZED APPLICATIONS AS PART OF THE DEFI ECOSYSTEM. Social Economics. 117-124. 10.26565/2524-2547- 2022-64-10.
- [2] Hysaj, Albina. (2023). Decentralized Finance. International Journal of Advanced Natural Sciences and Engineering Researches. 7. 158-162. 10.59287/asijanser.195.
- [3] Sakiz, Burcu & Gencer, Aysen. (2022). Decentralized Finance Applications via Blockchain. Journal of Eurasian Economies. 1. 43-54. 10.36880/J01.2.0112.
- [4] Vacca, Anna & Fredella, Michele & Di Sorbo, Andrea & Visaggio, Corrado Aaron & Piattini, Mario. (2023). Functional suitability assessment of smart contracts: A survey and first proposal. Journal of Software: Evolution and Process. 10.1002/smr.2636.
- [5] Xu, Jiahua & Feng, Yebo. (2022). Reap the Harvest on Blockchain: A Survey of Yield Farming Protocols. IEEE Transactions on Network and Service Management. PP. 1-1. 10.1109/TNSM.2022.3222815.
- [6] Xie, Wenhao. (2023). Delving into the Security Side of Decentralized Finance Applications. Advances in Economics, Management and Political Sciences. 7. 286-295. 10.54254/2754- 1169/7/20230246.
- [7] Ebekozi, Andrew & Aigbavboa, Clinton & Adekunle, Samuel & Samsurijan, Mohamad & Aliu, John & Arthur-Aidoo, Bernard & Amadi, Godpower. (2023). Smart contract applications in the built environment: How prepared are Nigerian construction stakeholders?. Frontiers of Engineering Management. 10.1007/s42524-023-0275-z.
- [8] Ikram, LEBHAR & Dadda, Afaf & Latifa, EZZINE. (2023). A Smarter Way to Procure: Exploring the Use of Smart Contracts. International Journal of Membrane Science and Technology. 10. 1296-1308. 10.15379/ijmst.v10i4.2244.
- [9] Goel, Himank & Gupta, Harshit & Sharma, M & Tripathi, K. (2022). Ethereum

Blockchain Wallets. International Journal for Research in Applied Science and Engineering Technology. 10. 2694-2697. 10.22214/ijraset.2022.42905.

[10] Xiong, Wei & Wan, Danping. (2023). Financial investment trust mechanism based on smart contract. PloS one. 18. e0287706. 10.1371/journal.pone.0287706.

- [11] The Application of Blockchain and Cryptocurrency in Meta-universe and NFT. BCP Business & Management. 44. 55-61. 10.54691/bcpbm.v44i.4793.
- [12] Tiwari, Varun & Singh, Santosh & Vadi, Vikas. (2023). Smart Contract Using Solidity (Remix -Ethereum IDE). 12. 243-249. 10.17148/IJARCCE.2023.12253.
- [13] Ahuja, Rupali & Khandelwal, Janhavi. (2023). RUPALI AHUJA et al.: CHALLENGES, OPPORTUNITIES AND RISK ANALYSIS OF ADOPTION OF DECENTRALIZED FINANCE APPLICATIONS CHALLENGES, OPPORTUNITIES AND RISK ANALYSIS OF ADOPTION OF DECENTRALIZED FINANCE APPLICATIONS. ICTACT Journal on Soft Computing. 14. 3114 - 3129. 10.21917/ijsc.2023.0435.
- [14] Besançon, Léo & Ferreira da Silva, Catarina & Ghodous, P. & Gelas, JeanPatrick. (2022). A Blockchain Ontology for DApps Development. IEEE Access. 29. 10.1109/ACCESS.2022.3173313.
- [15] Santana, Carlos & Albareda, Laura. (2022). Blockchain and the emergence of Decentralized Autonomous Organizations (DAOs): An integrative model and research agenda. Technological Forecasting and Social Change. 182. 121806. 10.1016/j.techfore.2022.121806.

APPENDIX

A. SOURCE CODE

Staker.sol

```
// SPDX-License-Identifier: UNLICENSED
```

```
pragma solidity 0.7.6;
```

```
import "https://github.com/OpenZeppelin/openzeppelin-  
contracts/blob/v3.4.0-solc-0.7/contracts/token/ERC20/IERC20.sol";  
import "https://github.com/OpenZeppelin/openzeppelin-  
contracts/blob/v3.4.0-solc-0.7/contracts/math/SafeMath.sol"; import  
"https://github.com/OpenZeppelin/openzeppelin-  
contracts/blob/v3.4.0-solc-0.7/contracts/access/Ownable.sol";
```

```
contract Staker is Ownable { using  
    SafeMath for uint256;
```

```
    struct UserInfo {  
        uint256 deposited;  
        uint256 rewardsAlreadyConsidered;  
    }
```

```
    mapping (address => UserInfo) users;
```

```
    IERC20 public depositToken; // eg. PancakeSwap ETB LP token  
    IERC20 public rewardToken; // eg. ETB
```

```
    // We are not using depositToken.balanceOf in order to prevent  
    DOS attacks (attacker can make the total tokens staked very large)  
    // and to add a skim() functionality with which the owner can collect  
    tokens which were transferred outside the stake mechanism.
```

```
    uint256 public totalStaked;
```

```
    uint256 public rewardPeriodEndTimestamp;  
    uint256 public rewardPerSecond; // multiplied by 1e7, to make up for  
    division by 24*60*60
```

```
    uint256 public lastRewardTimestamp;  
    uint256 public accumulatedRewardPerShare; // multiplied by 1e12, same  
    as MasterChef
```

```
    event AddRewards(uint256 amount, uint256 lengthInDays);  
    event ClaimReward(address indexed user, uint256 amount);  
    event Deposit(address indexed user, uint256 amount); event  
    Withdraw(address indexed user, uint256 amount); event  
    Skim(uint256 amount);
```

```
constructor(address _depositToken, address _rewardToken) {
```

```

        depositToken = IERC20(_depositToken);
        rewardToken = IERC20(_rewardToken);
    }

    // Owner should have approved ERC20 before. function
    addRewards(uint256 _rewardsAmount, uint256
_lengthInDays)
    external onlyOwner {
        require(block.timestamp > rewardPeriodEndTimestamp, "Staker:
can't add rewards before period finished");
        updateRewards(); rewardPeriodEndTimestamp
        =
        block.timestamp.add(_lengthInDays.mul(24*60*60));
        rewardPerSecond =
        _rewardsAmount.mul(1e7).div(_lengthInDays).div(24*60*60);
        require(rewardToken.transferFrom(msg.sender, address(this),
_rewardsAmount), "Staker: transfer failed");
        emit AddRewards(_rewardsAmount, _lengthInDays);
    }

    // Main function to keep a balance of the rewards.
    // Is called before each user action (stake, unstake, claim).
    // See top of file for high level description.
    function updateRewards()
    public {
        // If no staking period active, or already updated rewards after
        staking ended, or nobody staked anything - nothing to do
        if (block.timestamp <= lastRewardTimestamp) {
            return;
        }
        if ((totalStaked == 0) || lastRewardTimestamp >
        rewardPeriodEndTimestamp) {
            lastRewardTimestamp = block.timestamp; return;
        }

        // If staking period ended, calculate time delta based on the
        time the staking ended (and not after)
        uint256 endingTime;
        if (block.timestamp > rewardPeriodEndTimestamp) {
            endingTime = rewardPeriodEndTimestamp;
        } else {
            endingTime = block.timestamp;
        }
        uint256 secondsSinceLastRewardUpdate =
        endingTime.sub(lastRewardTimestamp);
        uint256 totalNewReward =
        secondsSinceLastRewardUpdate.mul(rewardPerSecond); // For everybody
        in the pool
        // The next line will calculate the reward for each staked token
        in the pool.
        // So when a specific user will claim his rewards,
        // we will basically multiply this var by the amount the
        user staked.
        accumulatedRewardPerShare =
        accumulatedRewardPerShare.add(totalNewReward.mul(1e12).div(totalStake
        d));
    }

```

```
lastRewardTimestamp = block.timestamp;
```

```

        if (block.timestamp > rewardPeriodEndTimestamp) { rewardPerSecond
            = 0;
        }
    }

    // Will deposit specified amount and also send rewards.
    // User should have approved ERC20 before.
    function deposit(uint256 _amount)
    external {
        UserInfo storage user = users[msg.sender];
        updateRewards();
        // Send reward for previous deposits if
        (user.deposited > 0) {
            uint256 pending =
user.deposited.mul(accumulatedRewardPerShare).div(1e12).div(1e7).sub(
user.rewardsAlreadyConsidered);
            require(rewardToken.transfer(msg.sender, pending),
"Staker: transfer failed");
            emit ClaimReward(msg.sender, pending);
        }
        user.deposited = user.deposited.add(_amount);
        totalStaked = totalStaked.add(_amount);
        user.rewardsAlreadyConsidered =
user.deposited.mul(accumulatedRewardPerShare).div(1e12).div(1e7);
        require(depositToken.transferFrom(msg.sender, address(this),
_amount), "Staker: transferFrom failed"); emit
        Deposit(msg.sender, _amount);
    }

    // Will withdraw the specified amount and also send rewards. function
    withdraw(uint256 _amount)
    external {
        UserInfo storage user = users[msg.sender]; require(user.deposited
        >= _amount, "Staker: balance not
enough");
        updateRewards();
        // Send reward for previous deposits
        uint256 pending =
user.deposited.mul(accumulatedRewardPerShare).div(1e12).div(1e7).sub(
user.rewardsAlreadyConsidered);
        require(rewardToken.transfer(msg.sender, pending), "Staker:
reward transfer failed");
        emit ClaimReward(msg.sender, pending);
        user.deposited = user.deposited.sub(_amount);
        totalStaked = totalStaked.sub(_amount);
        user.rewardsAlreadyConsidered =
user.deposited.mul(accumulatedRewardPerShare).div(1e12).div(1e7);
        require(depositToken.transfer(msg.sender, _amount), "Staker:
deposit withdrawal failed");
        emit Withdraw(msg.sender, _amount);
    }

    // Will just send rewards.
    function claim()
    external {

```

```
UserInfo storage user = users[msg.sender];  
if (user.deposited == 0)
```

```

        return;

        updateRewards();
        uint256 pending =
user.deposited.mul(accumulatedRewardPerShare).div(1e12).div(1e7).sub(
user.rewardsAlreadyConsidered);
        require(rewardToken.transfer(msg.sender, pending), "Staker: transfer
failed");
        emit ClaimReward(msg.sender, pending); user.rewardsAlreadyConsidered
        =
user.deposited.mul(accumulatedRewardPerShare).div(1e12).div(1e7);

    }

    // Will collect depositTokens (LP tokens) that were sent to the contract
    // Outside of the staking mechanism. function
    skim()
    external onlyOwner {
        uint256 depositTokenBalance = depositToken.balanceOf(address(this));
        if (depositTokenBalance > totalStaked) {
            uint256 amount = depositTokenBalance.sub(totalStaked);
            require(depositToken.transfer(msg.sender, amount),
"Staker: transfer failed");
            emit Skim(amount);
        }
    }

    // Return the user's pending rewards.
    function pendingRewards(address _user) public
    view returns (uint256) {
        UserInfo storage user = users[_user];
        uint256 accumulated = accumulatedRewardPerShare; if
        (block.timestamp > lastRewardTimestamp &&
lastRewardTimestamp <= rewardPeriodEndTimestamp && totalStaked != 0)
        {
            uint256 endingTime;
            if (block.timestamp > rewardPeriodEndTimestamp) { endingTime
            = rewardPeriodEndTimestamp;
            } else {
                endingTime = block.timestamp;
            }
            uint256 secondsSinceLastRewardUpdate =
endingTime.sub(lastRewardTimestamp);
            uint256 totalNewReward =
secondsSinceLastRewardUpdate.mul(rewardPerSecond);
            accumulated =
accumulated.add(totalNewReward.mul(1e12).div(totalStaked));
        }
        return
user.deposited.mul(accumulated).div(1e12).div(1e7).sub(user.rewardsAl
readyConsidered);
    }

    // Returns misc details for the front end.
    function getFrontendView()

```

```

        external view returns (uint256 _rewardPerSecond, uint256
        _secondsLeft, uint256 _deposited, uint256 _pending) {
            if (block.timestamp <= rewardPeriodEndTimestamp) {
                _secondsLeft =
                rewardPeriodEndTimestamp.sub(block.timestamp);
                _rewardPerSecond = rewardPerSecond.div(1e7);
            } // else, anyway these values will default to 0
            _deposited = users[msg.sender].deposited;
            _pending = pendingRewards(msg.sender);
        }
    }
}

```

hardhat.config.js

```

/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: { version:
    '0.8.9',
    defaultNetwork: 'goerli',
    networks: {
      hardhat: {},
      goerli: {
        url: 'https://rpc.ankr.com/eth_goerli',
        accounts: [`0x${process.env.PRIVATE_KEY}`]
      }
    },
    settings: { optimizer:
      {
        enabled: true,
        runs: 200,
      },
    },
  },
};

```

Index.js

```

import React from "react";
import ReactDOM from "react-dom";
import "./index.css";

```

```
import App from "./App";  
import reportWebVitals from "./reportWebVitals";
```

```

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);

// If you want to start measuring performance in your app, pass a
function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more:
https://bit.ly/CRA-vitals
reportWebVitals();

```

App.js

```

import ContractView from './components/ContractView';
import Header from './components/Header';
import Jumbotron from './components/Jumbotron';
import Wallet from './components/Wallet';

function App() {
  const [currentAccount, setCurrentAccount] = useState(null);
  const accountHandler = (account) => {
    setCurrentAccount(account);
  }

  return (
    <Fragment>
      <Header currentAccount={currentAccount} />
      <div className="app-container">
        <Jumbotron />
        <Wallet currentAccount={currentAccount}
accountHandler={accountHandler} />
        <ContractCreation currentAccount={currentAccount} />

```

```

        {currentAccount && <ContractView
currentAccount={currentAccount} />}
    </div>

```

```

    </Fragment>

```

```

    );
}

```

```

export default App;

```

Contract.js

```

import './Contract.css';
import { MUMBAI as NETWORK } from '../data/networks';
import Party from './Party';
import { descrowAbi } from '../contracts/contractData';
import { ethers } from 'ethers';
import { Fragment, useEffect, useState } from 'react';
import FlashMessage from 'react-flash-message';

const Contract = (props) => {

    const [contractState, setContractState] = useState(null);
    const [mineStatus, setMineStatus] = useState(null);

    const isBuyer = contractState ?
        props.currentAccount.toLowerCase() ===
contractState.buyer.toLowerCase() : null;
    const isLocked = contractState ?
        contractState.buyerStake && contractState.sellerStake :
null;
    const noCancel = contractState ?
        !contractState.buyerCancel && !contractState.sellerCancel
: null;

```

```
const getStatus = () => {
```

```

        if (contractState.active && isLocked) return 'Active';
        else if (contractState.active && !isLocked) return
'Unlocked';
        else if (contractState.cancelled) return 'Cancelled';
        return 'Completed';
    }

    const isStaked = () => {
        if (isBuyer && contractState.buyerStake) return true; if
        (!isBuyer && contractState.sellerStake) return true;
        return false;
    }

    const isCancelled = () => {
        if (isBuyer && contractState.buyerCancel) return true; if
        (!isBuyer && contractState.sellerCancel) return true;
        return false;
    }

    const btnConfirm = contractState && isBuyer && isLocked &&
noCancel;
    const btnStake = contractState && !isLocked && !isStaked();
    const btnRevokeStake = contractState && !isLocked &&
isStaked();
    const btnCancel = contractState && isLocked && !isCancelled();
    const btnRevokeCancel = contractState && isLocked &&
isCancelled();

    const provider = new
ethers.providers.Web3Provider(window.ethereum);
    const signer = provider.getSigner();
    const descrowContract = new
ethers.Contract(props.contractAddress, descrowAbi, signer);

    const stake = async () => {
        setMineStatus('mining')
    }

```

```
let txn;
```

```

    try {
      let stake = Number(contractState.price); if
      (isBuyer) stake = stake * 2;
      const formattedPrice =
ethers.utils.parseEther(String(stake));
      txn = await descrowContract.stake({ value:
formattedPrice });
      await txn.wait();
      setMineStatus('success')

    } catch (err) {
      setMineStatus('error')
      console.log(err)
    }
  }
}

```

```

const withdrawStake = async () => {
  setMineStatus('mining')
  let txn; try
  {
    txn = await descrowContract.revokeStake();
    await txn.wait();
    setMineStatus('success')

  } catch (err) {
    setMineStatus('error')
    console.log(err)
  }
}

```

```

const cancel = async () => {
  setMineStatus('mining') let
  txn;

  try {
    txn = await descrowContract.cancel(); await

```

```
txn.wait();
```

```

        setMineStatus('success')

    } catch (err) {
        setMineStatus('error')
        console.log(err);
    }
}

const revokeCancel = async () => {
    setMineStatus('mining')
    let txn;

    try {
        txn = await descrowContract.revokeCancellation();
        await txn.wait();
        setMineStatus('success')

    } catch (err) {
        setMineStatus('error')
        console.log(err);
    }
}

const confirm = async () => {
    setMineStatus('mining') let
    txn;

    try {
        txn = await descrowContract.confirm();
        await txn.wait(); setMineStatus('success')

    } catch (err) {
        setMineStatus('error')
        console.log(err);
    }
}

```

```

useEffect(() => {

    const provider = new
ethers.providers.Web3Provider(window.ethereum);
    const signer = provider.getSigner();
    const descrowContract = new
ethers.Contract(props.contractAddress, descrowAbi, signer);

    const cleanContract = (contractDetails) => {
        let cleanContract = {
            active: contractDetails.active,
            cancelled: contractDetails.cancelled,
            buyer: contractDetails.buyer, seller:
            contractDetails.seller,
            buyerCancel: contractDetails.buyerCancel,
            sellerCancel: contractDetails.sellerCancel,
            buyerStake: contractDetails.buyerStake,
            sellerStake: contractDetails.sellerStake,
            address: contractDetails.conAddr,
            price:
ethers.utils.formatEther(contractDetails.salePrice),
        }
        return cleanContract;
    }

    const getLatestData = async () => { let
        contractDetails = await
descrowContract.getStatus();
        setContractState(cleanContract(contractDetails));
    };

    getLatestData(); descrowContract.on('ContractStateChanged',

```

(buyer, seller,

```

state) => { setContractState(cleanContract(state))

});

}, [props.contractAddress])

return (
  <Fragment>
    {contractState && <div className='contract'>
      <div className='con-addr-active'>
        <p className='addr'>
          <span>Contract&nbsp;&nbsp;&nbsp;</span>
          0x{props.contractAddress.slice(2,
10)}&nbsp;&nbsp;&nbsp;</p>
          <a
href={{NETWORK.blockExplorerUrls[0]}/address/{props.contractAddress}}
target='_blank'
rel='noreferrer'>
            <i className="fa
fa-external-link"
aria-hidden="true"></i></a>
          </p>
          <p><span className={onlineStatus}
{getStatus()}></span>{getStatus()}</p>
        </div>
        <div className='con-price'>
          <p>{contractState.price} MATIC</p>
        </div>
        <div className='contract-parties'>
          <Party partyClasses='party buyer'
            partyText='Buyer'
            party={contractState.buyer}
            currentAccount={props.currentAccount}
            staked={contractState.buyerStake}

```

```
        cancelled={contractState.buyerCancel}  
    />
```

```
<Party partyClasses='party seller'
```

```

        partyText='Seller'
        party={contractState.seller}
        currentAccount={props.currentAccount}
    />
    </div>
    staked={contractState.sellerStake}
    cancelled={contractState.sellerCancel}

    {contractState.active &&
      <div className='action-button-container'>
        {btnStake && <button
onClick={stake}>Stake</button>
        {btnRevokeStake && <button
onClick={withdrawStake}>Withdraw Stake</button>
        {btnCancel && <button
onClick={cancel}>Cancel</button>
        {btnRevokeCancel && <button
onClick={revokeCancel}>Revoke Cancel</button>
        {btnConfirm && <button
onClick={confirm}>Finish</button>
      </div>
    }
  </div>}
  {mineStatus === 'success' && <FlashMessage
duration={2000}>
    <div className={form-submission ${mineStatus}}>
      <p>Transaction successful!</p>
    </div>
  </FlashMessage>}
  {mineStatus === 'mining' &&
    <div className={form-submission ${mineStatus}}>
      <p>
    </div>}
  </p>

```

```
<i className="fa fa-spinner      &nbsp;  Transaction is mining  
fa-spin"></i>
```

```
        {mineStatus === 'error' && <FlashMessage  
duration={2000}>
```

```

        <div className={form-submission ${mineStatus}}>
          <p>Transaction failed. Please try again.</p>
        </div>
      </FlashMessage>
    </Fragment>
  )
}

```

```
export default Contract;
```

ContractCreation.js

```

import './ContractCreation.css';
import { Fragment, useState } from 'react';
import { factoryAbi, descrowAbi, factoryAddress } from
'../contracts/contractData';
import ContractForm from './ContractForm';
import { ethers } from 'ethers';
import FlashMessage from 'react-flash-message';

const ContractCreation = (props) => {

  const [createForm, setCreateForm] = useState(false);
  const [mineStatus, setMineStatus] = useState(null);

  const showContractForm = () => {
    setCreateForm(true);
    setMineStatus(null);
    console.log(factoryAbi, descrowAbi, factoryAddress);
  }

  const cancelFormHandler = () => {
    setCreateForm(false);
  }

  const formSubmitHandler = async (data) => {
    setCreateForm(false);
  }

```

```
setMineStatus('mining');
```

```

    const provider = new
ethers.providers.Web3Provider(window.ethereum);
    const signer = provider.getSigner();

    const factoryContract = new
ethers.Contract(factoryAddress, factoryAbi, signer);

    let txn;

    try {
        const formattedPrice =
ethers.utils.parseEther(data.price);
        txn = await factoryContract.createContract(data.buyer,
data.seller, formattedPrice);
        await txn.wait();
        setMineStatus('success');

    } catch (err) {
        console.log(err);
        setMineStatus('error');
    }
}

return (
    <Fragment>
        {props.currentAccount && !createForm && mineStatus !==
'mining' &&
            <button className='cta-button cnc-button'
onClick={showContractForm}>
                Create New Contract
            </button>}
        {props.currentAccount && createForm && !mineStatus &&
            <ContractForm onCancel={cancelFormHandler}
                onFormSubmit={formSubmitHandler} />}
        {!createForm && <div>

```

```
{mineStatus === 'success' && <FlashMessage
```

```
export default ContractCreation;
```

)

ContractForm.js

```
import { useState } from 'react';
```

```
import './ContractForm.css';
```

```
const ContractForm = (props) => {
```

```
    const [buyer, setBuyer] = useState('');
```

```

const [seller, setSeller] = useState('');
const [price, setPrice] = useState('');
const [error, setError] = useState('');

const buyerHandler = e => setBuyer(e.target.value);
const sellerHandler = e => setSeller(e.target.value);
const priceHandler = e => setPrice(e.target.value);

const cancelHandler = () => {
  props.onCancel();
}

const formSubmitHandler = (e) => {

  e.preventDefault();
  setError(null);

  // Form validation
  if (buyer.trim().length > 0 && seller.trim().length > 0 &&
!isNaN(Number(price))) {

    const data = { buyer, seller, price }
    // Mine
    props.onFormSubmit(data);

  }
  else {
    setError('Please enter valid inputs!');
    return;
  }
}

return (
  <div className='form-container'>
    <h2>Create New Contract</h2>
    <form onSubmit={formSubmitHandler}>

```

```
<div className='form-control'>
```

```

        <label htmlFor='buyer'>Buyer Address</label>
        <input type='text' name='buyer' required
onChange={buyerHandler} value={buyer}></input>
      </div>
      <div className='form-control'>
        <label htmlFor='seller'>Seller Address</label>
        <input type='text' name='buyer' required
onChange={sellerHandler} value={seller}></input>
      </div>
      <div className='form-control'>
        <label htmlFor='price'>Price (in
MATIC)</label>
        <input type='number' name='price' required
onChange={priceHandler} value={price}></input>
      </div>
      {error && <p className='error-
message'>{error}</p>}
      <div className='button-container'>
        <button className='submitBtn'
type='submit'>Submit</button>
        <button className='cancelBtn'
onClick={cancelHandler}>Cancel</button>
      </div>
    </form>
  </div>
)
}

```

```
export default ContractForm;
```

ContractView.js

```

import { useCallback, useEffect, useState } from 'react';
import './ContractView.css';
import { factoryAbi, factoryAddress } from
'../contracts/contractData';

```

```
import { ethers } from 'ethers';
```

```

import Contract from './Contract'; const

ContractView = (props) => {

    const [contractAddressList, setContractAddressList] =
    useState(null);

    // Get data for all the user's contracts
    const getAllContracts = useCallback(async (accountAddr) => {
        const provider = new
ethers.providers.Web3Provider(window.ethereum);
        const signer = provider.getSigner();
        const factoryContract = new
ethers.Contract(factoryAddress, factoryAbi, signer);
        let contractAddresses = await
factoryContract.getContractsByParty(accountAddr);

setContractAddressList(contractAddresses.slice().reverse());

        // Set up event listener for factory contract
        factoryContract.on('ContractCreated', (buyer, seller,
price, addr) => {

            if (props.currentAccount.toLowerCase() ===
buyer.toLowerCase() ||
                props.currentAccount.toLowerCase() ===
seller.toLowerCase()) {
                console.log("Hey oh!"); console.log(buyer,
seller, price, addr);
                setContractAddressList(prevState => {
                    if (!prevState.includes(addr)) return [addr,
...prevState];

                })
            }
        })
    }, [props.currentAccount]);
}

```

```
return prevState;
```

```
}
```

[illegible]

```
currentAccount={props.currentAccount}
/> }                                </div>}

</div>
</div>

)
```

```
}
```

```
export default ContractView;
```

Wallet.js

```
import { useCallback } from 'react';
```

```
import { Fragment, useEffect } from 'react';
```

```
import { MUMBAI as NETWORK } from '../data/networks';
```

```
import './Wallet.css';
```

```
const Wallet = (props) => {
```

```
    const checkIfWalletIsConnected = useCallback(async () => {  
        const { ethereum } = window;
```

```
        if (!ethereum) {  
            console.log("Make sure you have Metamask installed!");  
            return;  
        } else {  
            console.log("Wallet exists! We're ready to go!")  
        }  
    }
```

```
    const accounts = await ethereum.request({ method:  
'eth_accounts' });
```

```
    if (accounts.length !== 0) { const  
        account = accounts[0];  
        console.log("Found an authorized account: ", account);
```

```
        // Switch network if it's not the correct chain  
        try {  
            await ethereum.request({  
                method: "wallet_switchEthereumChain",  
                params: [{ chainId: NETWORK.chainId }],  
            });
```

```
props.accountHandler(account);
```

```

        } catch (err) {
            console.log(err);
        }

    } else {
        console.log("No authorized account found");
    }
}, [props])

const connectWallet = async () => {

    const { ethereum } = window;

    if (!ethereum) {
        alert("Please install the Metamask Extension!");
    }

    // Get wallet address try
    {
        const accounts = await ethereum.request({ method:
'eth_requestAccounts' });
        console.log("Found an account! Address: ",
accounts[0]);

        await ethereum.request({
            method: "wallet_switchEthereumChain",
            params: [{ chainId: NETWORK.chainId }],
        });

        props.accountHandler(accounts[0]);

    } catch (err) {
        console.log(err);

        // If user doesn't have network configured, add it
        if (err.code === 4902) {

```

```
try {
```

```

        const accounts = await ethereum.request({
method: 'eth_requestAccounts' });
        await ethereum.request({
            method: "wallet_addEthereumChain",
            params: [NETWORK],
        });
        props.accountHandler(accounts[0]);
    } catch (err) {
        alert(err.message);
    }
}

}

}

useEffect(() => {
    checkIfWalletIsConnected();
}, [checkIfWalletIsConnected])

return (
    <Fragment>
        {!props.currentAccount && <button className='cta-
button cw-theme' onClick={connectWallet}>
            Connect Wallet
        </button>}
    </Fragment>
)

}

export default Wallet;

```

B. SCREENSHOTS

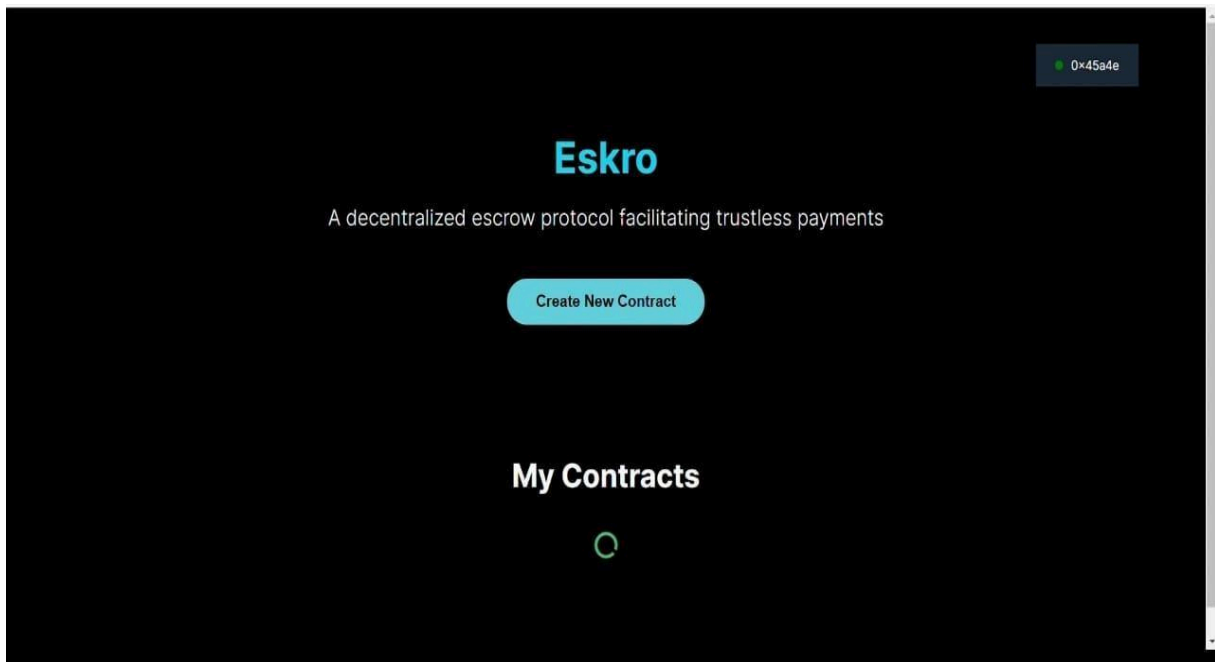


Fig B.1: Website Homepage

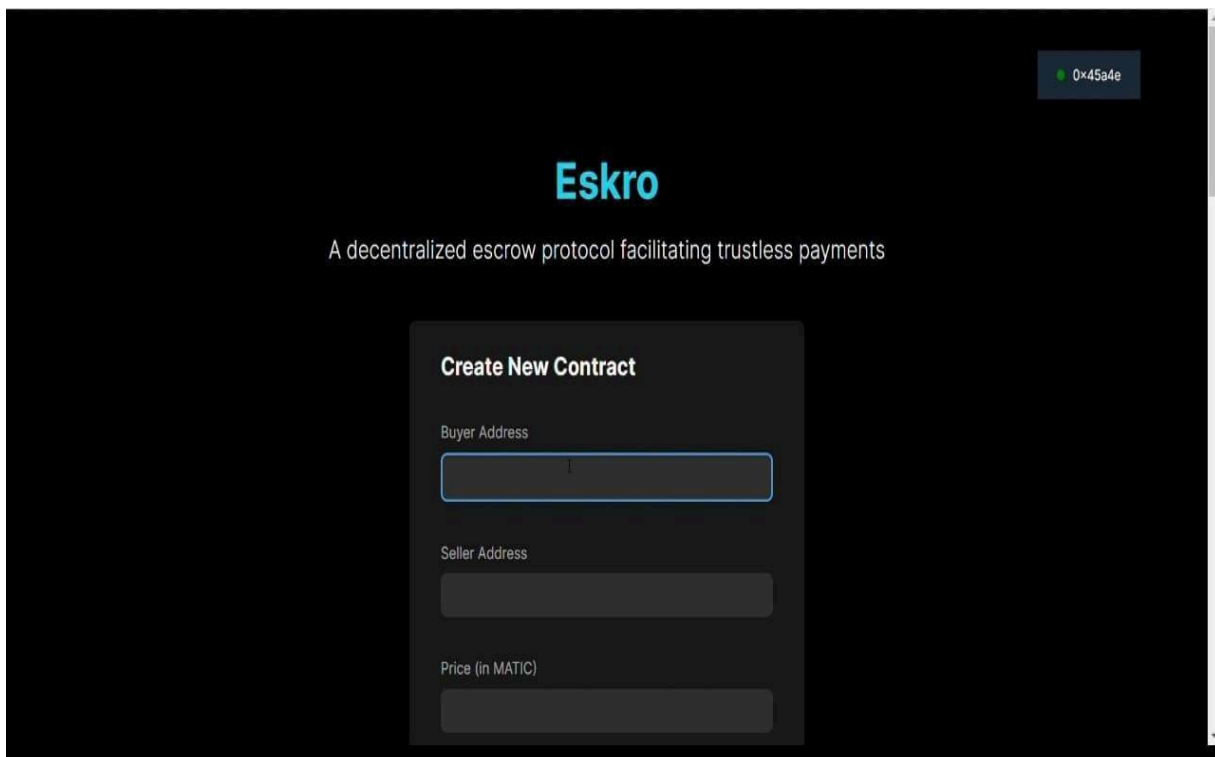


Fig B.2: Contract Page

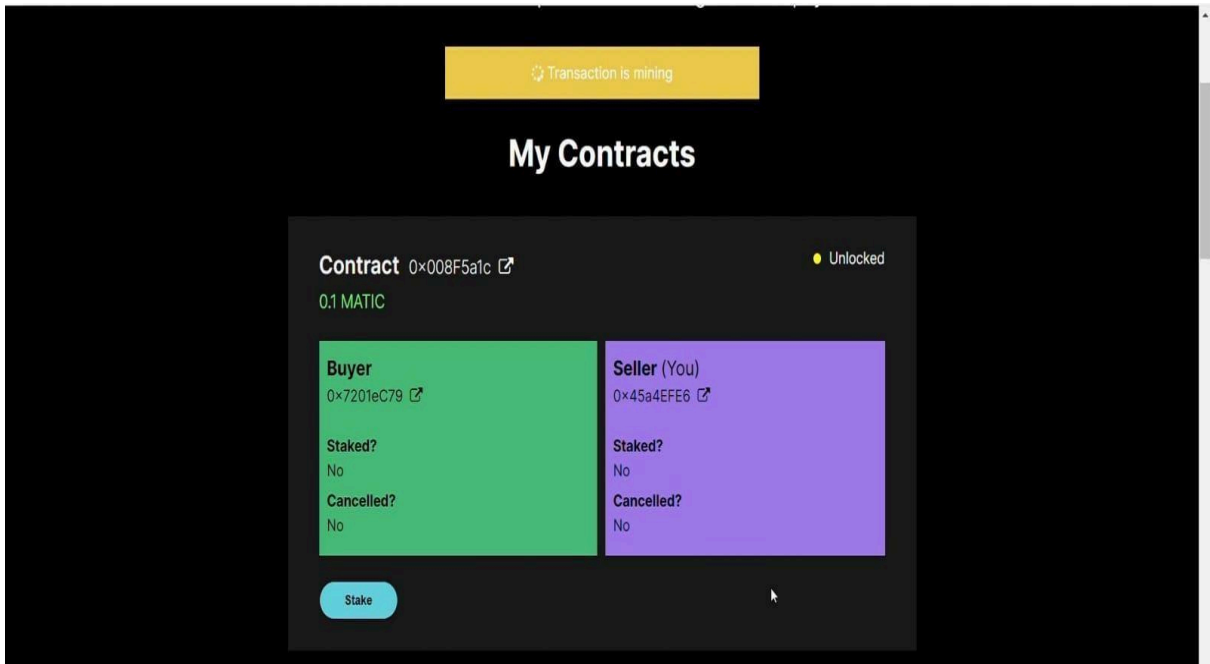


Fig B.3: Staking Page

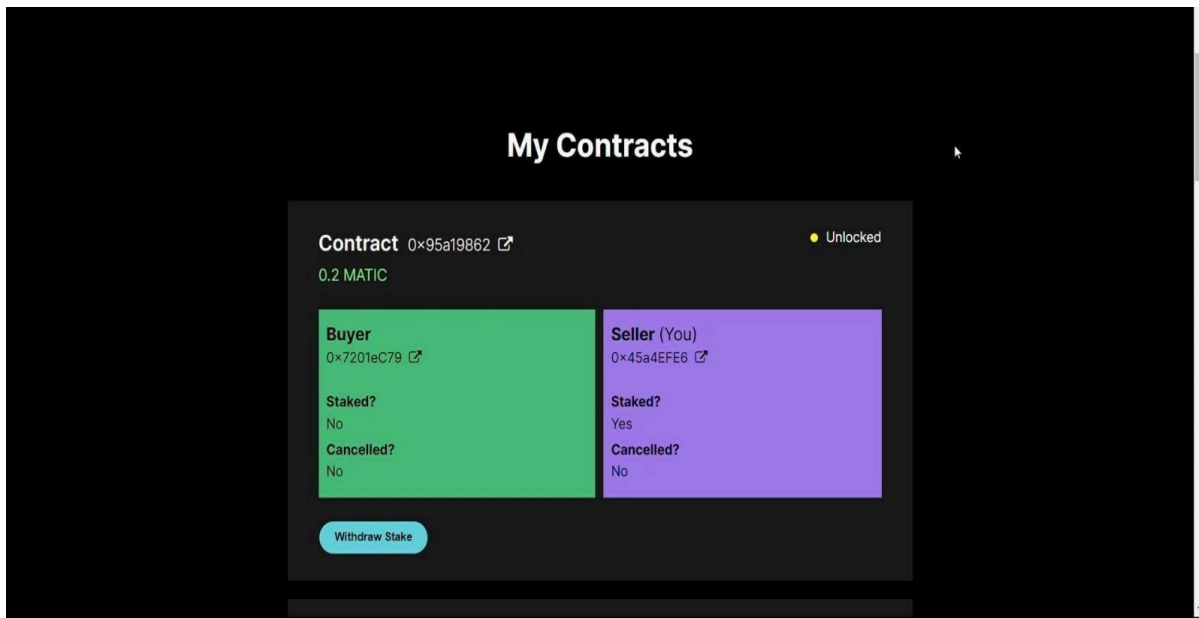


Fig B.4: Withdraw Page

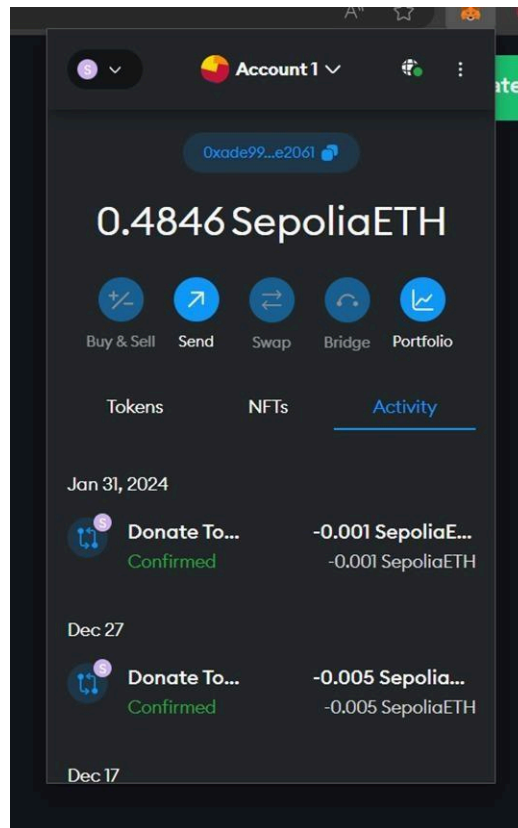


Fig B.5: Metamask wallet

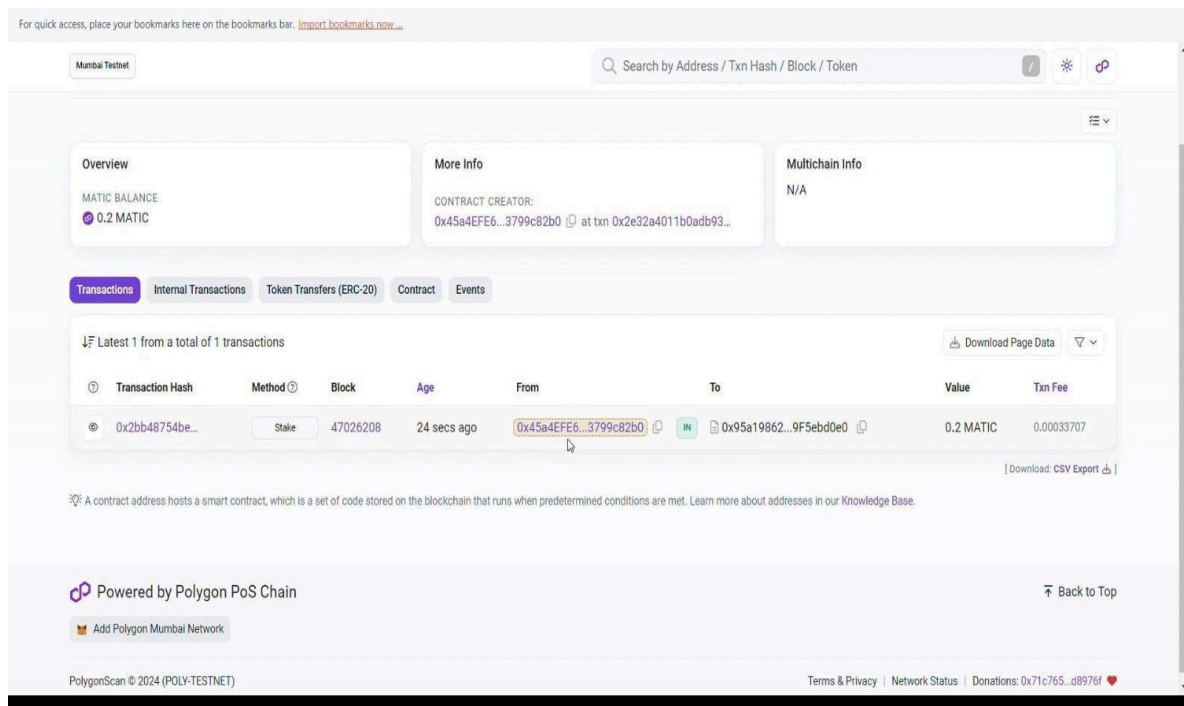


Fig B.6: Transactions recorded on polygonscan

C. RESEARCH PAPER

Staking DApp with Stab Signatures and Proof-of-stake Algorithm

Dhivesh Kumar A^{1, a)}, Dheiva Nishanth C^{2, b)} and M. Selvi^{3, c)}

Author Affiliations

¹*Dept of Computer Science Sathyabama
Institute of Science and Technology
Chennai, India*

²*Dept of Computer Science Sathyabama
Institute of Science and Technology
Chennai, India*

³*Dept of Computer Science Sathyabama
Institute of Science and Technology
Chennai, India*

Author Emails

^{a)}*dhiveshkumara.12@gmail.com*

^{b)}*dheivanishanth@gmail.com* ^{c)}*selvi.cse@sathyabama.ac.in*

Abstract. Due to their capacity to enable passive income creation and improve network security, staking DApps have attracted a great deal of attention in the blockchain community. In order to increase the overall effectiveness and security of the staking process, this abstract offers a novel staking decentralized application (DApp) that uses Stab Signatures and the Proof-of-Stake algorithm. Modern cryptographic techniques like Stab Signatures offer a special approach to the issue of signature malleability, guaranteeing the immutability and integrity of staking transactions. As the core consensus mechanism, the Proof-of-Stake algorithm enables users to validate and maintain the blockchain network by staking a set amount of cryptocurrency.

INTRODUCTION

The Proof-of-Stake Algorithm and Staking DApp redefine how users can engage in blockchain networks and earn incentives. Staking has become a well-known idea in the blockchain industry. It entails locking and verifying tokens to facilitate network operations. By utilising Stab Signatures, a cutting-edge method that improves security and effectiveness, this creative DApp elevates staking to a new level.

The Stab Signature algorithm combines the advantages of threshold cryptography and deterministic signatures. Deterministic signatures make the process more safe and attack-resistant by ensuring that each signer uses the same key pair for each signature. On the other side, threshold cryptography divides the key generation and signing procedure among various validators, ensuring decentralized network management. Stab Signatures enable a more stable and secure staking environment by fusing these two ideas.

Moreover, the Proof-of-Stake (PoS) technique is used by the Staking DApp to obtain consensus and validate transactions on the blockchain. The Proof-of-Stake (PoS) algorithm allows users to stake their tokens and is selected as validators based on the number of tokens they have, in contrast to the energy-intensive Proof-of-Work (PoW) process employed in many traditional cryptocurrencies. This strategy encourages a blockchain network that is more sustainably powered and greatly lowers energy usage.

Both new and seasoned users will find the staking process to be simplified by the user-friendly interface provided by the Staking DApp. By taking a few each actions, such as choosing the desired stake amount and verifying the transaction, users may effortlessly stake their tokens. The DApp also allows users to keep track of their staked tokens and validator status, as well as receive real-time information on staking incentives.

Users of the Staking DApp can actively engage in the governance of the blockchain network in addition to collecting staking incentives. Token holders can voice their ideas and have a say in crucial decisions involving network upgrades, parameter changes, and protocol modifications through voting procedures. The network will develop in accordance with the shared interests of all of its stakeholderst hanks to this democratic approach.

The Proof-of-Stake Algorithm and Staking DApp usher in a new era of efficient and safe staking on the blockchain. Our DApp gives users the freedom to stake their tokens with confidence, contribute to network security, and earn staking incentives by fusing cutting-edge technologies and approachable user interfaces. The Staking DApp offers a simple and lucrative staking experience, regardless of whether you are a blockchain enthusiast, an investor, or a token holder.

RELATED WORKS

Cryptographic advancements, along with distributed data records, spurred the emergence of decentralized financial assets. Uniquelydesigned applications evolved to guarantee the seamless functioning of the complete DeFi ecosystem.

Finance that operates through smart contracts, bypassing traditional financial entities, embodies a sector employing cutting-edge technology for participant autonomy and transparency. Utilizing advanced methods, it guarantees transparency and independence for participants and potential investors in financial dealings.

Blockchain is a secure, transparent, and decentralized technology that facilitates reliable record-keeping and eliminates the need for intermediaries like banks, resulting in reduced transaction costs and improved efficiency. A novel technological advancement ensures secure and transparent record-keeping, bypassing traditional intermediaries and fostering cost-effective and efficient transactions.

Blockchain facilitatesinteractions between untrusted parties in a decentralized manner, without relying on a trusted intermediary. A versatile technology enables decentralized interactions among parties without requiring a trusted middleman, while also empowering organizations to tackle real-world issues swiftly and affordably.

Decentralized Finance (DeFi) is revolutionizing the financial sector through blockchain and smart contract technologies, showcasing its potential in exchanges and yield farming. It emphasizes the transformative nature of DeFi and its promising applications in various financial services.

Yield farming is widely embraced in decentralized finance (DeFi) as a method of managing assets, where individuals engage in activities like providing, borrowing, or staking cryptocurrencies to generate income through transaction fees, interest, or participation rewards across various DeFi platforms. It highlightsthe practice's popularity and itsrole in generating income within the DeFi ecosystem through various asset management activities.

It summarizes the background and application of decentralized finance, as well as the development history and latest research of decentralized finance. Moreover, it proposes a classification framework for existing various DeFi DApps.

Smart contract technology, is a manifestation of blockchain, could enhance transparency, trust, and data transaction security in the construction sector. It also highlights the lack of academic research on smart contract applications within the construction industries of developing nations, particularly Nigeria.

Self-executing digital contracts could revolutionize procurement in supply chain management, particularly within the healthcare sector, addressing the complexities of procurement processes to boost efficiency, security, and transparency. It emphasizes the transformative potential of smart contracts in streamlining procurement practices and improving operational outcomes, especially in industries like healthcare where these challenges are

prominent.

Ethereum's popularity stems from its ability to handle large block sizes and its capacity to address complex issues through smart contracts without reliance on intermediaries, offering a simpler implementation process compared to alternative blockchain technologies. Its appeal lies in the platform's capability to facilitate transactions efficiently while minimizing the need for external oversight, making it a preferred choice for various applications.

Smart contracts function as automated protocols within blockchain networks, orchestrating account operations within the Ethereum ecosystem. They serve as algorithmic protocols that execute predetermined business transactions based on input data from involved parties, streamlining processes without manual intervention.

Using blockchain's smart contract capabilities establishes a trust mechanism for financial investments, facilitating the reception, storage, and processing of investment data, which is then disseminated across blockchain nodes. This process harnesses blockchain's features to ensure transparency and reliability in financial investment operations without relying on centralized intermediaries.

In today's world, the metaverse facilitates the convergence of physical and digital realms, allowing avatars to participate in various endeavors spanning trading, socializing, entertainment, and creation. While currently limited to digital gaming, collectibles, and art, the distinctive attributes, protocols, and platforms of NFTs suggest potential applications in addressing real-world challenges.

Decentralized Applications, known as DApps, offer decentralized and trustworthy services leveraging blockchain technology, typically comprising various functionalities like transaction scalability, distributed storage, and computational solutions. These applications provide a decentralized framework for reliable services, incorporating elements such as enhanced transaction processing, decentralized data storage, and collaborative computational capabilities.

Decentralized autonomous organizations (DAOs) operate as peer-to-peer networks where contributors collectively manage operations through automated rules encoded in smart contracts, fostering decentralized decision making without traditional executive structures. Governance within DAOs relies on a blend of on-chain and off-chain mechanisms, facilitating community-driven decision-making processes independent of centralized authorities.

EXISTING SYSTEM

There are a few drawbacks to the current Proof-of-Stake algorithm and Stab Signatures for Staking DApp system. The high energy use is the first significant negative. Proof-of-Stake methods need a lot of processing power to verify transactions and protect the network. This results in higher energy use, which is not a sustainable option for the environment.

Second, the possibility for centralization is a drawback. Wealthier participants have a better probability of being selected as validators in a Proof-of-Stake system. This might lead to a concentration of power in the hands of a select few, potentially leading to control becoming centralized. As a result, the decentralized nature of blockchain technology is compromised, and problems like censorship and manipulation may arise.

The system is also vulnerable to attacks with supposedly "nothing at stake." In this kind of attack, validators can alter the blockchain several times without investing any resources. As a result, it is simpler for attackers to double-spend or jeopardize the blockchain's integrity. It is a significant security issue that needs to be resolved in order to preserve the system's credibility and trust.

The reliance on outside validators is still another drawback. Validators are in charge of approving transactions and upkeep of the network in Proof-of-Stake systems. Nonetheless, this fosters a certain amount of faith in these outside parties. The integrity and security of the entire network may be jeopardized if validators conspire or carry out nefarious actions. This goes against the decentralized nature of blockchain technology because it puts a lot of reliance on the validators.

Finally, the system might not be very scalable. The system may encounter bottlenecks and delays in the validation process as the number of users and transactions rises. This may reduce the network's effectiveness and reduce the likelihood of broad adoption.

In summary, the Staking DApp with Stab Signatures and Proof-of-Stake Algorithm has a number of benefits but also a number of drawbacks. For a more resilient and secure blockchain system, it is imperative to address the high energy consumption, possibility for centralization, susceptibility to "nothing-at-stake" assaults, dependency on external validators, and constrained scalability.

PROPOSED SYSTEM

The proposed work intendsto create a Staking DApp using Proof-of-Stake (PoS) technology with Stab Signatures. Users will be able to stake their bitcoin holdings using this DApp and receive incentives for taking part in the network consensus.

The staking procedure will be made secure and effective by using the Stab Signatures technique. Stab Signatures, which combine multi-party computation and conventional digital signatures, offer a distinctive method for signing transactions. This method not only strengthens the staking DApp's security but also defends against other threats including forging and manipulation. The great level of scalability provided by Stab Signatures also makes it possible for a large number of people to participate in the staking process without jeopardizing security.

On the other side, the consensus protocol's validators will be chosen according to the PoS method. PoS algorithms take into account a user's cryptocurrency holdings and their commitment to the network, in contrast to standard Proof-of-Work methods that depend on computing power. With less energy use and quicker transaction validations made possible by this method, validators are chosen in an equitable and decentralized manner.

The user-friendly interface of the Staking DApp will make it simple for bitcoin owners to take part in the staking process. Customers can choose how much cryptocurrency to stake and how long they wish to stake it for. In exchange, users will consistently be rewarded for their contributions to network consensus and security.

The DApp will also give extensive analytics and monitoring features to further improve user experience. User can monitor their staked assets, prizes received, and the network's overall performance. Users will be able to maximize their rewards and make well-informed judgments about their staking strategy thanks to this.

Overall, the suggested Staking DApp with Stab Signatures and PoS algorithm offers cryptocurrency owners a safe, effective, and user-friendly platform for staking their assets and earning rewards. Our DApp intends to transform the staking ecosystem by utilizing cutting-edge cryptographic techniques and decentralized consensus protocols, making it more accessible and lucrative for all users.

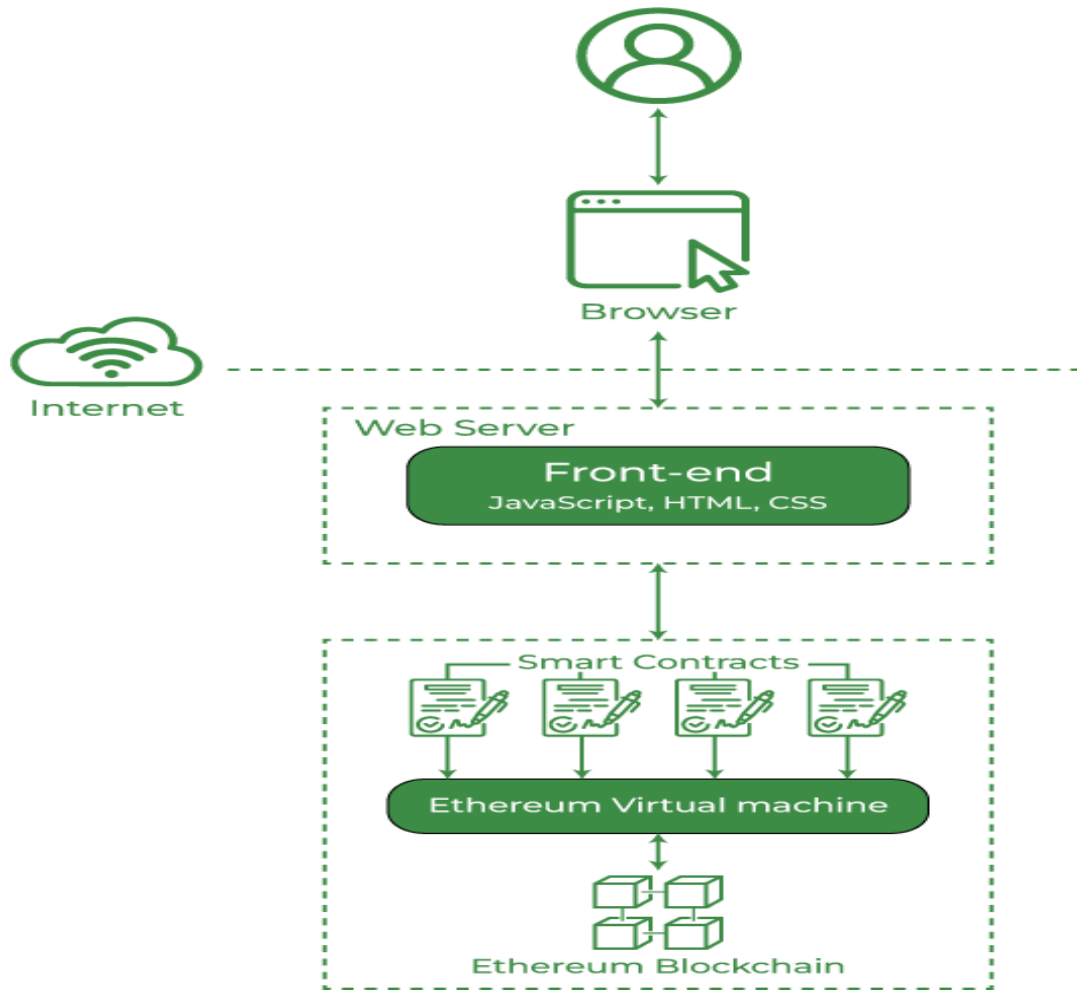


FIGURE 1. System architecture

METHODOLOGY

DApp Staking

The Staking DApp is the initial module of the suggested system. The user interface for system-wide staking operations is provided by this module. To participate in the Proof-of-Stake consensus method, users can stake their tokens. Users may examine their staking balance, deposit and withdraw tokens, and keep track of their staking earnings using the Staking DApp's capabilities. Users can also transfer their staking authority to a particular network validator thanks to this feature. Additionally, the Staking DApp has a user friendly dashboard that shows crucial details like the size of the current staking pool, the total number of staked tokens, and the distribution of rewards across validators. Users can actively participate in the staking process and contribute to upholding the security and stability of the blockchain network by using the Staking DApp

Stab Signatures

Stab Signatures is the second module of the suggested system. The system uses Stab Signatures, a cryptographic method, to guarantee the validity and integrity of the staking transactions. It uses symmetric and asymmetric encryption methods in conjunction to produce a safe and impenetrable signature for each staking transaction. Attacks like double spending and replay attacks are both prevented by Stab Signatures. The system

can authenticate each staking transaction and stop any malicious behaviour by employing Stab signatures. This module is essential for ensuring the system's overall security and reliability of the staking process.

Proof-of-Stake Algorithm

The Proof-of-Stake (PoS) algorithm is the third module of the suggested system. The consensus method for approving and adding new blocks to the blockchain is determined by the PoS algorithm. The PoS method assigns the task of creating new blocks to validators, who are chosen depending on their stake. The block validation algorithm, block selection algorithm, and reward distribution algorithm are just a few of the sub-modules that make up this module. Each block put forth by the validators is checked for accuracy and integrity by the block validation algorithm. Based on their stake and other variables, the block selection algorithm selects a validator to create the following block. Based on their contribution to the network, the reward distribution algorithm decides how much each validator will be rewarded. The PoS algorithm keeps the blockchain network secure and effective by rewarding ethical behaviour among validators and punishing any harmful activity

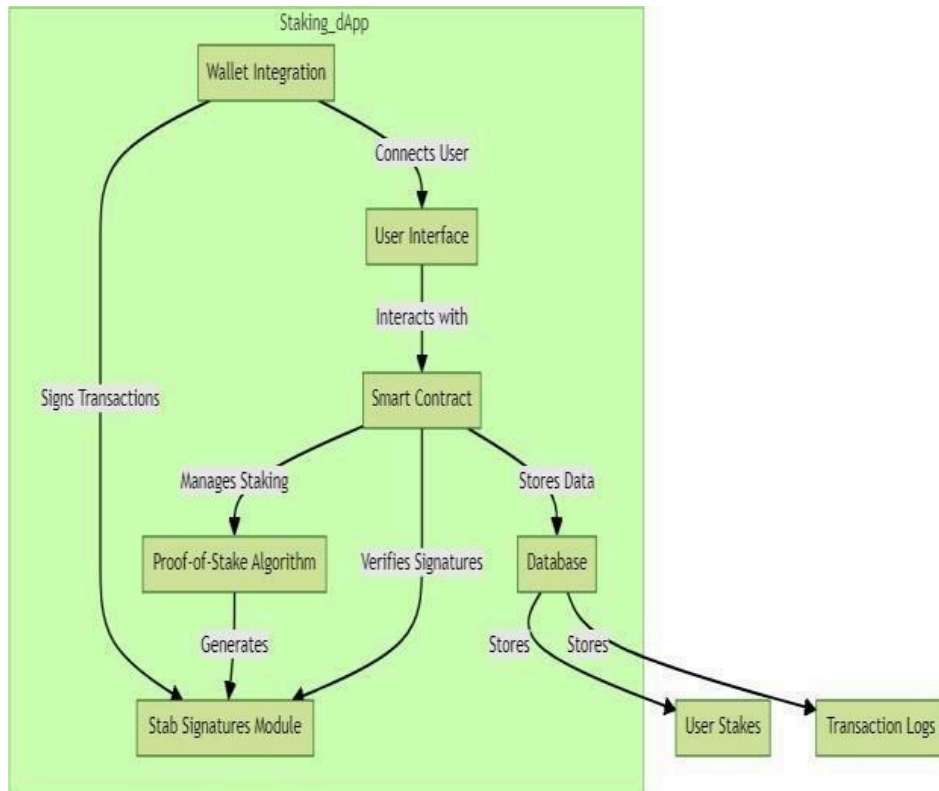


FIGURE 2. System flow diagram

RESULTS

For effective and secure stacking activities, the Stabilised dApp system uses Proof-of-Stake (PoS) technology and Stab Signatures. Stabilise offers a high level of security throughout the staking process by using Stab Signatures, which are cryptographic signatures produced from stablecoin holdings. Users can employ Stab Signatures to stake their stablecoin holdings without actually moving any money to the system, which lowers the danger of theft and hacking. The PoS algorithm of the technology further improves the staking process' security and scalability. The PoS method is more energy-efficient and economical because it doesn't require resource-intensive mining because it relies on a decentralized network of validators. Validators are picked depending on the quantity of tokens invested, making it so that individuals with higher stakes have a better chance of selection. Participants are encouraged to stake more tokens as a result, improving decentralization and network security.

The staking DApp from Stabilise gives players appealing benefits in addition to a reliable and safe system for staking. As a reward for their contribution to the network, users who stake their stablecoins receive more

tokens. Users are encouraged to participate actively in the staking process as a result, which promotes network involvement and stability. Overall, the Stabilised dApp system provides a reliable and user-friendly platform for staking stablecoins thanks to its Stab Signatures and PoS algorithm. It offers security, scalability, and user participation incentives, making it a desirable choice for anybody interested in participating in the staking ecosystem.

CONCLUSION

In conclusion, the Proof-of-Stake Algorithm and Staking DApp system provides consumers with a reliable and safe way to stake their cryptocurrency assets. Stab Signatures integration boosts security by making transactions impervious to tampering and assaults. When compared to conventional Proof-of-Work models, the Proof-of-Stake algorithm improves the system's efficiency and scalability by enabling quicker transaction processing and lessening its environmental effect. Ultimately, this approach gives consumers a trustworthy and long-lasting method of staking their assets, fostering the development and stability of the bitcoin ecosystem.

ACKNOWLEDGEMENT

We extend our sincere gratitude to all those who played a pivotal role in the success of this endeavor. Our heartfelt appreciation goes to our advisors, colleagues, and research participants for their invaluable support and insights. Their steadfast encouragement and dedication were crucial in bringing this work to fruition.

REFERENCES

1. Dotsenko, Oleksii. (2022). ARCHITECTURE OF DECENTRALIZED APPLICATIONS AS PART OF THE DEFI ECOSYSTEM. *Social Economics*. 117-124. 10.26565/2524-2547- 2022-64-10.
2. Hysaj, Albina. (2023). Decentralized Finance. *International Journal of Advanced Natural Sciences and Engineering Researches*. 7. 158-162. 10.59287/as-ijanser.195.
3. Sakız, Burcu & Gencer, Aysen. (2022). Decentralized Finance Applications via Blockchain. *Journal of Eurasian Economies*. 1. 43-54. 10.36880/J01.2.0112.
4. Vacca, Anna & Fredella, Michele & Di Sorbo, Andrea & Visaggio, Corrado Aaron & Piattini, Mario. (2023). Functional suitability assessment of smart contracts: A survey and first proposal. *Journal of Software: Evolution and Process*. 10.1002/smr.2636.
5. Ahuja, Rupali & Khandelwal, Janhavi. (2023). RUPALI AHUJA et al.: CHALLENGES, OPPORTUNITIES AND RISK ANALYSIS OF ADOPTION OF DECENTRALIZED FINANCE APPLICATIONS CHALLENGES, OPPORTUNITIES AND RISK ANALYSIS OF ADOPTION OF DECENTRALIZED FINANCE APPLICATIONS. *ICTACT Journal on Soft Computing*. 14. 3114 - 3129. 10.21917/ijsc.2023.0435.
6. Xu, Jiahua & Feng, Yebo. (2022). Reap the Harvest on Blockchain: A Survey of Yield Farming Protocols. *IEEE Transactions on Network and Service Management*. PP. 1-1. 10.1109/TNSM.2022.3222815
7. Xie, Wenhao. (2023). Delving into the Security Side of Decentralized Finance Applications. *Advances in Economics, Management and Political Sciences*. 7. 286-295. 10.54254/2754- 1169/7/20230246.
8. Ebekozen, Andrew & Aigbavboa, Clinton & Adekunle, Samuel & Samsurijan, Mohamad & Aliu, John & Arthur-Aidoo, Bernard & Amadi, Godpower. (2023). Smart contract applications in the built environment: How prepared are Nigerian construction stakeholders?. *Frontiers of Engineering Management*. 10.1007/s42524-023-0275-z.
9. Ikram, LEBHAR & Dadda, Afaf & Latifa, EZZINE. (2023). A Smarter Way to Procure: Exploring the Use of Smart Contracts. *International Journal of Membrane Science and Technology*. 10. 1296-1308. 10.15379/ijmst.v10i4.2244.
10. Goel, Himank & Gupta, Harshit & Sharma, M & Tripathi, K. (2022). Ethereum Blockchain Wallets. *International Journal for Research in Applied Science and Engineering Technology*. 10. 2694-2697. 10.22214/ijraset.2022.42905.
11. Tiwari, Varun & Singh, Santosh & Vadi, Vikas. (2023). Smart Contract Using Solidity (Remix -Ethereum IDE). 12. 243-249. 10.17148/IJARCCE.2023.12253.
12. Xiong, Wei & Wan, Danping. (2023). Financial investment trust mechanism based on smart contract. *PloS one*. 18. e0287706. 10.1371/journal.pone.0287706.

13. Li, Pengqi. (2023). The Application of Blockchain and Cryptocurrency in Meta-universe and NFT. BCP Business & Management. 44. 55-61. 10.54691/bcpbm.v44i.4793

14. Besançon, Léo & Ferreira da Silva, Catarina & Ghodous, P. & Gelas, Jean-Patrick. (2022). A Blockchain Ontology for DApps Development. IEEE Access. 29. 10.1109/ACCESS.2022.3173313.
15. Santana, Carlos & Albareda, Laura. (2022). Blockchain and the emergence of Decentralized Autonomous Organizations (DAOs): An integrative model and research agenda. Technological Forecasting and Social Change. 182. 121806. 10.1016/j.techfore.2022.121806.

D. CERTIFICATE

