ETHEREUM-BASED KYC FRAMEWORK FOR IMPROVED CUSTOMER VERIFICATION IN BANKING

A MINI PROJECT REPORT

Submitted by

RIYAZ AHAMED J	**810421104141**
SIVASAKTHI S	**810421104165**
THANIGASALAM V	**810421104177**
YUVARAJ K	**810421104192**

In partial fulfillment for the award of degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE

(AUTONOMOUS)

PERAMBALUR-621 212

ANNA UNIVERSITY : CHENNAI 600 025

NOVEMBER 2024

DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE (AUTONOMOUS) PERAMBALUR – 621 212

BONAFIDE CERTIFICATE

Certified that this mini project report **"ETHEREUM-BASED KYC FRAMEWORK FOR IMPROVED CUSTOMER VERIFICATION IN BANKING"** is the bonafide work of **"RIYAZ AHAMED J (810421104141), SIVASAKTHI S (810421104165), THANIGASALAM V (810421104177), YUVARAJ K (810421104192)"** who carried out the project work under my supervision.

SIGNATURE

Dr. R. GOPI, M.Tech., Ph.D.,

HEAD OF THE DEPARTMENT,
Department of Computer Science and Engineering,
Dhanalakshmi Srinivasan Engineering College (Autonomous),
Perambalur – 621 212.

SIGNATURE

Mrs. S. FRANCIS SHAMILI M.E.,

SUPERVISOR,
Department of Computer Science and Engineering,
Dhanalakshmi Srinivasan Engineering College (Autonomous),
Perambalur – 621 212.

Submitted for Mini-Project Viva-Voce Examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our gratitude and thanks to **Our Parents** first for giving health and a sound mind for completing this mini project. We give all the glory and thanks to our almighty **GOD** for showering upon, the necessary wisdom and grace for accomplishing this project.

It is our pleasant duty to express a deep sense of gratitude to our honorable Chancellor, **Shri. A. Srinivasan**, for his kind encouragement. We sincerely thank our principal **Dr. D. Shanmugasundram, M.E., Ph.D., F.I.E., C.Eng.,** our **Dr. K. Anbarasan M.E., Ph.D.**, and our COE, **Dr. K. Velmurugan, M.E., Ph.D.,** for their unflinching devotion, which enabled us to complete this project.

We express our faithful and sincere gratitude to our Head of the Department **Dr. R. Gopi, M.Tech., Ph.D.,** for his valuable guidance and support that he gave us during the project time.

We express our faithful and sincere gratitude to our Project Coordinator **Mr. V. Gokulakrishnan, M.E., M.B.A., Ph.D.,** of Department of Computer Science and Engineering for giving support throughout our project.

We also thankful to our internal guide **Mrs. S. Francis Shamili, M.E.,** of Department of Computer Science and Engineering for her valuable guidance and precious suggestion to complete this project work successfully.

We render our thanks to all **Faculty members** and **Programmers** of Department of **Computer Science and Engineering** for their timely assistance.

DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE (AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision and Mission of the Department:

Vision

To produce globally competent, socially responsible professionals in the field of Computer Science and Engineering.

Mission

M1: Impart high quality experiential learning to get expertise in modern software tools

M2: Inculcate industry exposure and build inter disciplinary research skills.

M3: Mould the students to become Software Professionals, Researchers and Entrepreneurs by providing advanced laboratories.

M4: Acquire Innovative skills and promote lifelong learning with a sense of societal and ethical responsibilities

Program Educational Objectives (PEOs)

PEO 1: Graduates of the programme will develop proficiency in identifying, formulating, and resolving complex computing problems.

PEO 2: Graduates of the programme will achieve successful careers in the field of computer science and engineering, pursue advanced degrees, or demonstrate entrepreneurial success.

PEO 3: Graduates of the programme will cultivate effective communication skills, teamwork abilities, ethical values, and leadership qualities for professional engagement in industry and research organizations.

ABSTRACT

This project aims to enhance the existing Ethereum blockchain framework for Know Your Customer (KYC) by implementing a decentralized and streamlined KYC verification system. Traditional centralized approaches to KYC often involve redundant processes and lack transparency, leading to inefficiencies and increased operational costs. In contrast, this innovative solution utilizes smart contracts on a private Ethereum network, established using Geth, to facilitate secure and efficient customer identity verification by authorized banks. The proposed system introduces automated role-based access control, ensuring that only registered and verified institutions can perform KYC checks. This feature not only enhances security but also minimizes the risk of unauthorized access to sensitive customer information. Additionally, customer data is stored securely and immutably on the blockchain, significantly reducing redundancy and mitigating the risks of fraud and data breaches. By eliminating the need for repeated verifications across multiple banks, this decentralized solution lowers operational costs and improves efficiency for financial institutions. Furthermore, it ensures compliance with regulatory standards, as the transparency provided by smart contracts makes the KYC process tamper-resistant and auditable in real time. This project offers a robust, scalable, and efficient solution to the challenges faced by the financial sector in customer identity verification. Moreover, the implementation of this system can lead to improved customer experience, as individuals will only need to complete the KYC process once, avoiding the hassle of repetitive submissions. The platform can also support a wider range of financial services by providing a standardized and trustworthy method for identity verification, facilitating quicker onboarding for customers. Ultimately, this solution positions financial institutions to better leverage blockchain technology, fostering innovation and enhancing their competitive edge in a rapidly evolving market.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

KYC	KNOW YOUR CUSTOMER
POA	PROOF OF AUTHORITY
POS	PROOF OF STAKES
POW	PROOF OF WORK
GDPR	GENERAL DATA PROTECTION REGULATION
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
OS	OPERATING SYSTEM
CPU	CENTAL PROCESSING UNIT
SSD	SOLID STATE DISK
GB	GIGA BYTES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In today's digital landscape, Know Your Customer (KYC) verification has become a critical process for financial institutions to verify the identity of their clients, reduce fraud, and ensure compliance with regulatory standards. Traditional KYC processes are often centralized, costly, and prone to delays, making them inefficient for both institutions and customers. Blockchain technology offers a decentralized, secure, and transparent solution to streamline KYC verification.This project presents a blockchain-based KYC verification system built on a private Ethereum network. Utilizing smart contracts, this system allows multiple banks to verify customer identities in collaboration with a central admin, reducing the risk of identity fraud while enhancing privacy and data integrity. Each bank node in the network can securely register and verify clients by interacting with the admin node, ensuring that only authorized banks have the right to perform KYC registration.The project employs Proof of Authority (PoA) consensus to optimize the network's performance, with a Dockerized setup deployed on Minikube for containerized node management. Monitoring through Prometheus allows for real-time tracking of network health and performance, ensuring the robustness and reliability of the system. This approach provides a modern, secure, and efficient solution for KYC verification, addressing the limitations of traditional centralized systems. Under this system, customers experience faster onboarding while banks benefit from reduced KYC costs and streamlined verification processes. The decentralized nature of blockchain enhances security and privacy, as sensitive data is not stored in a central repository. This approach also reduces redundant KYC checks, enabling a more efficient and user-friendly experience.

1.2 OBJECTIVES

To design and implement a blockchain-based KYC verification system on a private Ethereum network, facilitating secure and efficient identity verification among banks. The project aims to establish a decentralized framework where multiple banks can interact with an admin node to perform KYC verification securely, ensuring only authorized entities can register users. Utilizing Proof of Authority (PoA) consensus, Dockerized Geth nodes, and Minikube for infrastructure management, this system provides a scalable, reliable, and transparent solution for KYC verification, monitored through Prometheus for network health and performance metrics. The system aims to enhance data integrity and privacy by leveraging blockchain's decentralized nature, thereby eliminating reliance on a central authority for KYC verification. By implementing smart contracts, the project ensures that each verification is tamper-proof and accessible only to authorized bank nodes. This setup reduces the risk of identity fraud, as customer information cannot be altered or duplicated across the network. The proposed system also aims to streamline the KYC process by allowing banks to share verified identities, thus reducing redundant verifications and saving time for both institutions and customers. By storing encrypted customer data on a decentralized ledger, the system enhances security and mitigates the risk of data breaches associated with traditional centralized storage. The PoA consensus mechanism further ensures that only trusted and pre-approved nodes participate in the network, adding an extra layer of security and control. Smart contracts enable automated validation of KYC data, minimizing manual intervention and reducing operational overhead. The project's use of containerization with Docker and Minikube allows for seamless scalability and deployment across diverse environments, promoting long-term adaptability in the face of evolving regulatory demands.

CHAPTER 2

LITERATURE SURVEY

2.1 A SURVEY ON BLOCKCHAIN TECHNOLOGY: EVOLUTION, ARCHITECTURE AND SECURITY.

AUTHOR: M. N. M. Bhutta, A. A. Khwaja, A. Nadeem, H. F. Ahmad, M. K. Khan, M. A. Hanif, H. Song, M. Alshamari and Y. Cao,

PUBLISHER: IEEE Access

PUBLISHED YEAR: 2021

DESCRIPTION:

This article surveys Blockchain technology, tracing its evolution from cryptocurrencies like Bitcoin to broader applications across industries. It covers Blockchain's architecture, development frameworks, security issues, and types—public, private, and consortium—highlighting their characteristics and use cases. The paper also explores consensus algorithms, cryptographic primitives, future directions, and open research challenges, emphasizing Blockchain's transparency, decentralization, and security.

TECHNIQUES USED: Distributed Ledger Technology (DLT), Smart Contracts, Cryptographic Hashing, Internet of Things (IoT) Integration, Tokenization

ADVANTAGES:

- **Transparency:** Real-time tracking ensures Blockchain transparency.
- **Accountability:** Immutable records enhance accountability across the supply chain.

DISADVANTAGES:

- **Integration Challenges:** Existing SCM systems may face difficulties in adopting blockchain without significant adjustments

2.2 FORTIFYING THE BLOCKCHAIN: A SYSTEMATIC REVIEW AND CLASSIFICATION OF POST-QUANTUM CONSENSUS SOLUTIONS FOR ENHANCED SECURITY AND RESILIENCE

AUTHOR: J. Gomes, S. Khan, and D. Svetinovic

PUBLISHER: IEEE Access

PUBLISHED YEAR: 2023

DESCRIPTION:

This paper reviews Post-Quantum Blockchain Consensus (PQBC), analyzing 29 key studies on the impact of quantum computing on blockchain security and consensus. It highlights post-quantum cryptographic methods for improved privacy, security, speed, and resistance to quantum attacks, while identifying research gaps and future directions to enhance blockchain scalability, trust, and privacy (SSTP).

TECHNIQUES USED: Post-Quantum Cryptography, Quantum-resistant Consensus Algorithms, Quantum Key Distribution (QKD), Hash-Based Signatures and Lattice-Based Cryptography

ADVANTAGES:

- **Quantum Resistance:** Shields blockchains from quantum attacks.
- **Improved Security:** Secures transactions with post-quantum cryptography.

DISADVANTAGES:

- **Quantum Computing Development**: Shields blockchains from quantum attacks.
- **Compatibility Issues:** Secures transactions with post-quantum cryptography.

2.3 AN EXAMINATION OF DISTRIBUTED AND DECENTRALIZED SYSTEMS FOR TRUSTWORTHY CONTROL OF SUPPLY CHAINS

AUTHOR: V. Kumar C and P. Selvaprabhu

PUBLISHER: IEEE Access

PUBLISHED YEAR: 2023

DESCRIPTION:

This paper examines the role of effective Supply Chain Management (SCM) systems during crises like pandemics, disasters, and conflicts, emphasizing the need for efficient coordination to avoid risks such as counterfeit products and transportation bottlenecks. It highlights blockchain technology's potential to enhance transparency, accountability, and data integrity in SCM through its decentralized, secure architecture. The paper also explores blockchain's applications across sectors, focusing on its practical use in improving SCM.

TECHNIQUES USED: Consensus Algorithms, Cryptographic Primitives, Smart Contracts, Distributed Ledger Technology (DLT) and Public, Private, and Consortium Blockchains.

ADVANTAGES:

- **Transparency:** Ensures clear, auditable records of transactions.
- **Decentralization:** Eliminates the need for centralized authorities, reducing risks of corruption and manipulation.

DISADVANTAGES:

- **Scalability Issues:** Blockchains can face performance bottlenecks as data grows.
- **Energy Consumption:** Some consensus algorithms (like Proof of Work) require significant energy resources.

2.4 DEMO: BLOCKCHAIN FOR THE SIMPLIFICATION AND AUTOMATION OF KYC RESULT SHARING

AUTHOR: R. Norvill, M. Steichen, W. M. Shbair, and R. State

PUBLISHER: IEEE Access

PUBLISHED YEAR: 2023

DESCRIPTION:

This paper proposes a blockchain-based system to improve the Know Your Customer (KYC) process for banks. KYC requires banks to collect personal data from customers to verify their identity and comply with regulations. The current process is costly, redundant, and time-consuming. The proposed system automates tasks and enables secure sharing of customer data between banks. It uses blockchain to store hashes of customer data (off-chain), ensuring GDPR compliance while providing an immutable audit trail of actions and permissions. Access control is used to manage data access levels based on permissions, enhancing security and privacy.

TECHNIQUES USED: Blockchain, Access Control, Private Blockchain, Off-chain Storage and GDPR Compliance.

ADVANTAGES:

- **Cost Reduction**: Automates repetitive KYC tasks, reducing operational costs for banks.
- **Efficiency:** Speeds up customer onboarding and data verification.

DISADVANTAGES:

- **Storage Inefficiency**: Storing hashes on-chain can still incur costs and complexity.
- **Complex Setup**: Integration with existing banking systems can be time-consuming and resource-intensive.

2.5 BLOCKCHAIN-BASED KYC MODEL FOR CREDIT ALLOCATION IN BANKING

AUTHOR: B. Karadag, A. Halim Zaim, and A. Akbulut

PUBLISHER: IEEE Access

PUBLISHED YEAR: 2023

DESCRIPTION:

This paper proposes a blockchain-based Know Your Customer (KYC) model for the banking sector to improve credit allocation and risk management. The system, built on Ethereum with smart contracts, decentralizes data storage and sharing, enhancing transparency, security, and efficiency in KYC and credit assessment. By using a private blockchain, it eliminates the need for intermediaries, reducing costs, data breaches, and addressing Sybil attack concerns.

TECHNIQUES USED: Blockchain Technology, Ethereum, Smart Contracts (Solidity) and Private Blockchain Network

ADVANTAGES:

- **Faster KYC:** The blockchain system speeds up the KYC process by enabling real-time data sharing between banks.
- **Cost Savings:** Reduces the need for intermediaries, lowering transaction and data handling costs.

DISADVANTAGES:

- **Complex Setup:** Implementing the blockchain system in existing banking infrastructure may be difficult and costly.
- **Scalability Issues:** The system may slow down as more banks and transactions are added, especially with high data volumes.

2.6 A REVIEW OF BLOCKCHAIN APPROACHES FOR KYC

AUTHOR: Mohamed S. Hassan.

PUBLISHER: IEEE Access

PUBLISHED YEAR: 2023.

DESCRIPTION:

The paper explores blockchain's potential to improve the traditional Know Your Customer (KYC) process in banking by offering a decentralized, secure, and transparent system for real-time data verification. Blockchain eliminates intermediaries, reducing costs, fraud, and redundant document submissions, while enhancing customer experience. The paper also reviews previous studies on blockchain's role in KYC, highlighting challenges and opportunities in the banking and finance sectors.

TECHNIQUES USED: Blockchain, Access Control, Private Blockchain, Off-chain Storage and GDPR Compliance.

ADVANTAGES:

- **Enhanced Security:** Blockchain's cryptographic nature ensures that customer data is secure and immutable, reducing the risk of fraud and data breaches.
- **Increased Efficiency:** Blockchain automates the KYC process, making data verification faster and more accurate, reducing time and administrative costs.

DISADVANTAGES:

- **Implementation Complexity**: Integrating blockchain into existing banking systems can be complex and costly, requiring significant technological upgrades.
- **Scalability Issues**: As more institutions and customers participate, blockchain systems could face performance issues, especially with large volumes of data.

2.7 A BLOCKCHAIN BASED SOLUTION TO KNOW YOUR CUSTOMER (KYC) DILEMMA

AUTHOR: D. George, A. Wani, and A. Bhatia

PUBLISHER: IEEE Access.

PUBLISHED YEAR: 2023

DESCRIPTION:

This article discusses the traditional Know Your Customer (KYC) process in financial institutions, highlighting its manual intervention, centralized data storage, and vulnerability to attacks. It proposes a Blockchain-based solution using Ethereum, smart contracts, and digital signatures to create an immutable, tamper-proof system for managing KYC data. The decentralized approach enhances security, reduces redundancy, and ensures better auditability, addressing the errors and vulnerabilities of the current centralized system.

TECHNIQUES USED: Private Blockchain, Distributed Application (DApp), Proof-of-Work (PoW), Proof-of-Stake (PoS) and Proof of Authority (PoA).

ADVANTAGES:

- **Enhanced Security:** Blockchain ensures data is tamper-proof, with no single point of failure, making KYC data much more secure against malicious attacks.
- **Decentralization:** The use of a distributed ledger prevents data from being compromised through central authority vulnerabilities.

DISADVANTAGES:

- **Initial Setup Cost:** The implementation of a Blockchain-based system involves significant costs for development, infrastructure, and training.
- **Integration Challenges:** Integrating the Blockchain solution with existing banking systems can be complex and time-consuming.

2.8 TOWARDS IMPROVING PRIVACY AND SECURITY OF IDENTITY MANAGEMENT SYSTEMS USING BLOCKCHAIN TECHNOLOGY: A SYSTEMATIC REVIEW

AUTHOR: H. Alanzi and M. Alkhatib

PUBLISHER: Appl. Sci.

PUBLISHED YEAR: 2022

DESCRIPTION:

The paper explores how blockchain can improve Identity Management Systems (IDMS) by addressing privacy, confidentiality, and security in digital transactions. Traditional, centralized IDMSs rely on third-party entities, making them vulnerable to data breaches, user tracking, and single points of failure. Blockchain's decentralized, immutable nature offers solutions to these issues, enhancing privacy and security through features like distributed ledger technology (DLT) and smart contracts. The paper also identifies research gaps and best practices for implementing blockchain in IDMS.

TECHNIQUES USED: Blockchain Technology, Distributed Ledger Technology, Decentralization, Immutable Data Storage.

ADVANTAGES:

- **Decentralization:** No central authority, reducing risks of failure and improving data availability.
- **Immutability:** Data can't be altered once added, ensuring data integrity.

DISADVANTAGES:

- **Scalability:** Blockchain networks may slow down as they grow.

2.9 BLOCKCHAIN IN BANKING AND FINANCE: A BIBLIOMETRIC REVIEW

AUTHOR: R. Patel, M. Migliavacca, and M. E. Oriani

PUBLISHER: Res. Int. Bus. Finance

PUBLISHED YEAR: 2022

DESCRIPTION:

This paper presents a bibliometric review and content analysis of academic literature on the adoption and impact of blockchain technology in banking and finance. The study aims to map out the key themes, influential authors, and target journals within this rapidly growing research area. The authors analyze a final sample of 154 papers, published between 2009 and 2021, focusing on the antecedents, applications, and consequences of blockchain in the banking and financial sectors. The paper also provides insights into trends like co-authorship, co-citation, and coupling analysis, and highlights key literature streams and future research directions.

TECHNIQUES USED: Co-authorship Analysis, Co-citation Analysis, Coupling Analysis, Cartographic Analysis and Content Analysis

ADVANTAGES:

- **Decentralization:** Blockchain eliminates intermediaries, reducing transaction costs and improving efficiency.
- **Immutability:** Blockchain ensures transaction integrity and transparency by preventing alterations.

DISADVANTAGES:

- **Adoption Barriers:** Banks face technological, organizational challenges.
- **Regulatory Issues:** Blockchain's decentralization complicates compliance.

2.10 A SYSTEMATIC LITERATURE REVIEW OF BLOCKCHAIN-BASED E-KYC SYSTEMS

AUTHOR: M. A. Hannan, M. A. Shahriar, M. S. Ferdous, M. J. M. Chowdhury, and M.S.Rahman

PUBLISHER: Computing

PUBLISHED YEAR: 2023

DESCRIPTION:

Blockchain-based e-KYC integrates traditional KYC processes with blockchain's security, transparency, and efficiency, enabling remote customer onboarding via digital tools like biometrics or national ID authentication. Blockchain's decentralized, immutable ledger ensures secure, auditable, and tamper-proof identity data sharing between financial institutions, addressing issues of centralization, data duplication, and privacy. Despite its benefits, challenges such as lack of standardization, regulatory uncertainty, and fraud risks remain.

TECHNIQUES USED: Smart Contracts, Zero-Knowledge Proofs, Decentralized Identifiers (DIDs), Verifiable Credentials and Public Key Infrastructure(PKI)

ADVANTAGES:

- **Enhanced Security:** Blockchain's cryptographic features ensure that sensitive data is protected from unauthorized access and tampering.
- **Transparency and Immutability:** Transactions and identity verifications are recorded on a tamper-proof ledger, making it easier to track and audit the process.

DISADVANTAGES:

- **Scalability Issues:** As the volume of transactions and verifications grows, blockchain systems can face scalability challenges, impacting performance.

CHAPTER 3

SYSTEM DEVELOPMENT

3.1 EXISTING SYSTEM

The existing system in "Ethereum Blockchain Framework Enabling Banks to Know Their Customers" leverages Ethereum blockchain technology to modernize the KYC process, addressing traditional inefficiencies, high costs, and security risks. By decentralizing KYC management across a network of banks under a central regulatory authority, this system enables a single verified KYC record for each customer, accessible by all authorized banks, reducing redundant processes and enhancing security. A central bank manages the registry and oversees compliance, while Ethereum smart contracts automate KYC tasks like bank registration, identity verification, and privilege management, ensuring consistent, tamper-proof processes. This blockchain setup reduces operational costs by minimizing intermediaries and enables efficient data access, as customer verification by one bank is accepted across the network. Although highly secure, the system may still face vulnerabilities in smart contracts, such as overflow and re-entrance issues, and relies on centralized oversight, highlighting areas for future improvement.

3.1.1 Disadvantages:

- **Scalability**: The private network and PoA consensus can hinder scalability, causing performance issues as the number of banks or transaction volumes grow.
- **Centralized Control:** The admin node holds significant authority, creating a single point of control that conflicts with blockchain's decentralized nature.
- **Privacy Concerns:** Storing sensitive KYC data on-chain raises privacy risks, potentially violating data protection laws like GDPR, which require data modification or deletion options.
- **Regulatory Compliance:** Blockchain's immutability makes it difficult to ensure compliance with regulations that mandate data alteration or removal upon user request

3.2 PROPOSED SYSTEM

The proposed system is a blockchain-based KYC framework on a private Ethereum network using Proof of Authority (PoA) consensus. It includes an admin node, two bank nodes, and an observer node, with each playing specific roles in the KYC process. Bank.sol and Admin.sol smart contracts enable secure, authorized interactions: banks register KYC data, while the admin validates and approves it. Only whitelisted banks can submit KYC data, preventing unauthorized access, and MetaMask ensures that only verified entities interact with the blockchain.

The system benefits from blockchain's immutability (locking KYC data from alteration) and transparency (tracking all actions on the blockchain for a complete audit trail). PoA makes it efficient, eliminating the heavy computation of Proof of Work. However, scalability may be limited as adding banks requires more infrastructure, and access control remains centralized through the admin node. Compliance with privacy laws and external verification could also pose challenges.

3.2.1 Advantages:

- **High Security:** Only authorized banks can access and submit KYC data, preventing unauthorized use
- **Data Integrity:** KYC data is immutable once registered, ensuring accuracy and preventing tampering
- **Transparency:** Every action is logged, creating a clear audit trail for security and compliance
- **Efficiency:** Proof of Authority (PoA) consensus is cost-effective and suitable for private networks
- **Privacy:** Only authorized entities can access KYC data, protecting user information
- **Cost Savings:** Reduces administrative costs by streamlining the KYC process on a decentralized network

3.3 SYSTEM ARCHITECTURE

In this system architecture smart contracts is used for secure transactions and it is developed and tested using remix IDE and deployed using metamask in the PoA network where that network is orchestrated in minikube cluster. In minikube cluster 3 namespaces named as PoA-KYC-Network, PoA-KYC-Dapp and PoA-KYC-Monitoring is created to manage and store all the transactions. This PoA network can be accessed by customer,bank1,bank2 and authorized validator using frontend page by interacting using metamask.

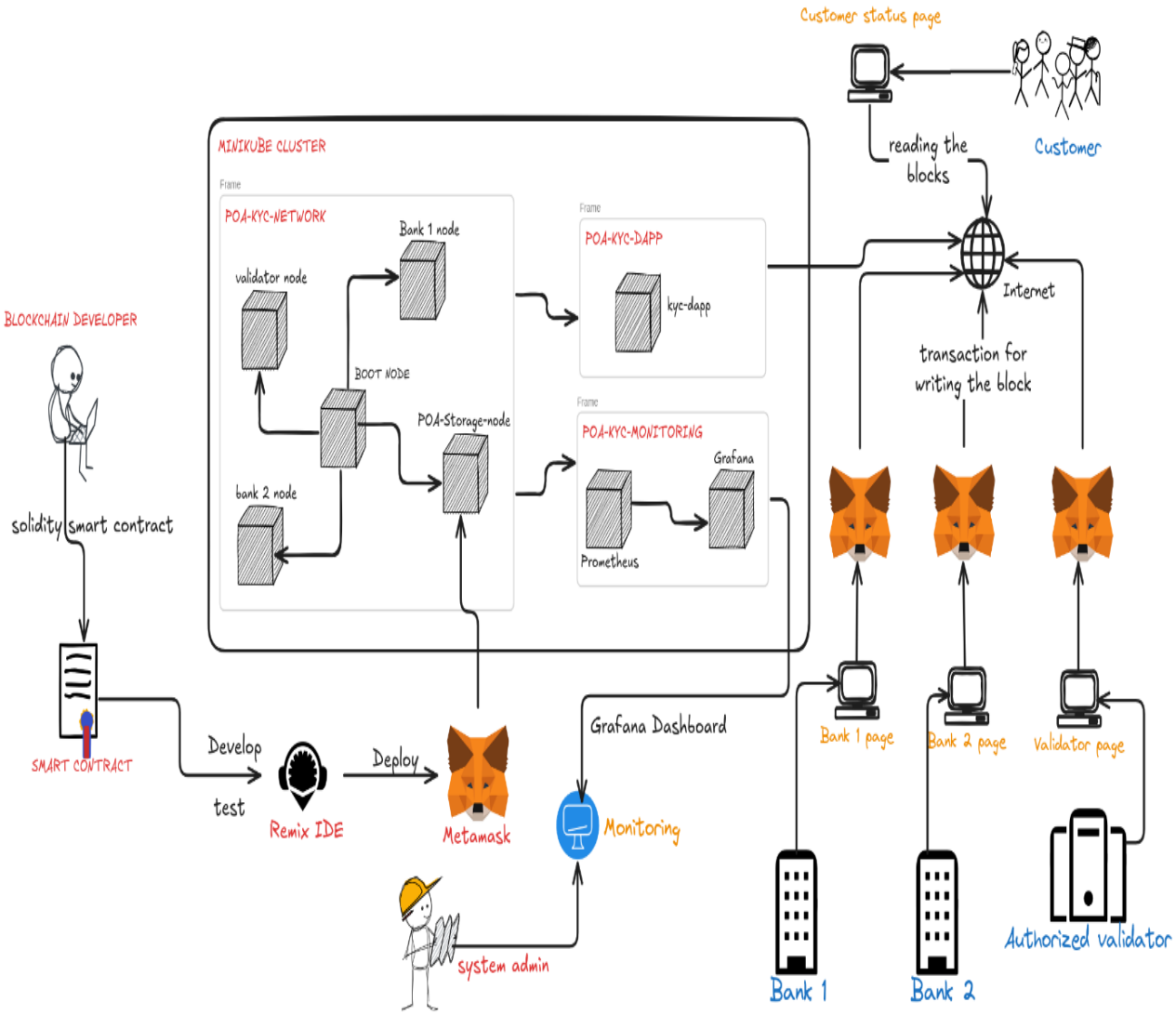

Fig 3.3: System Architecture

CHAPTER 4

MODULE DESCRIPTION

4.1 SET UP NODES FOR PRIVATE BLOCKCHAIN

Using Geth, you created a private Ethereum blockchain. Nodes were configured for specific roles, including a bootnode for coordination, a validator node for transaction validation, two bank nodes (bank1 and bank2), and a Proof of Authority (PoA) storage node for consensus. This setup forms the core of the private network infrastructure. Each node was assigned unique configurations to fulfill its role, ensuring seamless communication and efficient transaction processing. To enhance security, the nodes were connected using encrypted communication channels. A custom genesis block was configured to define the blockchain's initial state, including parameters like block time and difficulty. This setup provided a secure, scalable foundation for testing and deploying decentralized applications (DApps).

4.2 DEVELOP SMART CONTRACT

Developed smart contracts using Solidity with files like poa-contract.sol, validator.sol, bank1.sol, bank2.sol, and KYCList.sol. These contracts define the business logic, roles, and permissions for participants in the network, automating processes and interactions on the blockchain. Each smart contract was written with a focus on modularity and reusability to simplify updates and enhancements. Key functionalities included defining access controls, transaction limits, and data-sharing protocols for secure interactions between nodes. The KYCList.sol contract centralized customer data while maintaining privacy through cryptographic hashing. Rigorous testing ensured the contracts adhered to business requirements and industry standards for security and functionality.

4.3 IMPLEMENT, TEST, AND DEPLOY SMART CONTRACT

Remix IDE was utilized for testing, debugging, and deploying the smart contracts. Remix allowed you to simulate various scenarios, ensuring your contracts worked as intended before deploying them to the private blockchain. Test cases covered edge scenarios, such as invalid inputs and permission breaches, to ensure robustness. A local blockchain environment was used for iterative testing to minimize deployment errors. Deployment scripts were automated using Truffle and web3.js, streamlining the process and reducing manual intervention. Post-

deployment, contract interactions were verified using MetaMask, ensuring that all functionalities were accessible and functioning correctly within the network.

4.4 CONTAINERIZE THE BLOCKCHAIN NODES

Docker was employed to containerize each node, enabling isolated environments and consistent deployment. This approach automated the deployment process and simplified network management, making the blockchain more scalable and easy to replicate. Each container was preloaded with the necessary configurations and dependencies to ensure seamless operation. Docker Compose files were created to manage multi-container applications, facilitating quick setup and teardown. Resource limits were defined for each container to optimize performance and prevent resource contention. The use of containers also enabled easy migration of nodes across systems, enhancing portability and resilience in production environments.

4.5 HANDLE ACCOUNTS AND TRANSACTIONS

For secure account management and transaction handling we used MetaMask alongside ethers.js. These tools facilitated interaction with Ethereum accounts, enabling users to manage accounts, initiate transactions, and interact with deployed smart contracts within the blockchain ecosystem. MetaMask provided a user-friendly interface for managing private keys and signing transactions securely. Ethers.js was used for programmatically interacting with the Ethereum network, allowing for seamless integration of smart contracts with the frontend application. The combination of these tools ensured secure, transparent, and efficient handling of transactions within the blockchain, giving users control over their funds and enabling reliable communication with the deployed contracts. Ethers.js also helped with transaction monitoring, ensuring that users could track the status of their actions in real-time.

4.6 MONITOR NODES AND NETWORK PERFORMANCE:

Prometheus and Grafana were integrated for monitoring network health. Prometheus gathered metrics such as node performance and transaction rates, while Grafana provided data visualization and alerting, helping maintain stability and performance by identifying and addressing issues promptly. Prometheus was configured to collect detailed logs and performance metrics from each node, allowing for real-time monitoring and historical analysis. These metrics

were then visualized through Grafana dashboards, offering clear insights into network health, resource usage, and transaction throughput. Alerts were set up to notify administrators of potential issues, such as low node performance or transaction delays, allowing for quick intervention. This monitoring system ensured the blockchain network remained reliable, scalable, and efficient over time.

4.7 COORDINATE NODE DEPLOYMENT AND SCALING

Minikube was used to set up Kubernetes for managing node deployments. This allowed for efficient scaling, load balancing, and orchestration, supporting network growth and ensuring that resources were allocated as demand increased, maintaining a resilient and balanced network. Kubernetes automated the deployment, scaling, and management of containerized blockchain nodes, making it easy to add or remove nodes based on network requirements. Minikube provided a local environment for testing Kubernetes deployments before moving to production. By integrating Kubernetes, you were able to ensure that the blockchain network could scale dynamically, adapting to changing workloads and minimizing downtime. The orchestration framework allowed for seamless management of node updates, ensuring that the blockchain infrastructure remained secure and operational.

CHAPTER 5

SYSTEM DESIGN

5.1 OVERVIEW

The architecture of the private blockchain network comprises multiple nodes and tools that enable secure, decentralized data management. A bootnode serves as the central coordinator for node discovery and connections. The network includes validator nodes, which uphold transaction validation and block generation, and two bank nodes, which facilitate specific banking services. A dedicated PoA (Proof of Authority) storage node is also configured to store transactions securely. User access and interactions are managed via MetaMask, with monitoring tools such as Prometheus and Grafana for real-time tracking of node performance, making the system robust and manageable. Additionally, the architecture is designed to prioritize scalability, ensuring it can accommodate an increasing number of transactions and nodes. All components work cohesively to provide a secure, efficient, and transparent environment for banking operations.

5.2 NODE AND NETWORK SETUP

To establish the blockchain network, Geth was used to create and configure various types of nodes, each assigned a specific role. The bootnode coordinates all node connections, allowing efficient node discovery and communication. Validator nodes ensure transaction validation and blockchain security by implementing the PoA consensus. Bank nodes, named bank1 and bank2, are deployed to handle specific banking operations, while a PoA storage node stores transaction data securely. This setup, leveraging PoA consensus, ensures that the network is reliable and suited for private, permissioned environments with predefined participants. Moreover, each node is uniquely configured to ensure high availability, with backup mechanisms in place to prevent data loss during potential failures. The modular nature of this setup makes it adaptable to different use cases within the banking domain.

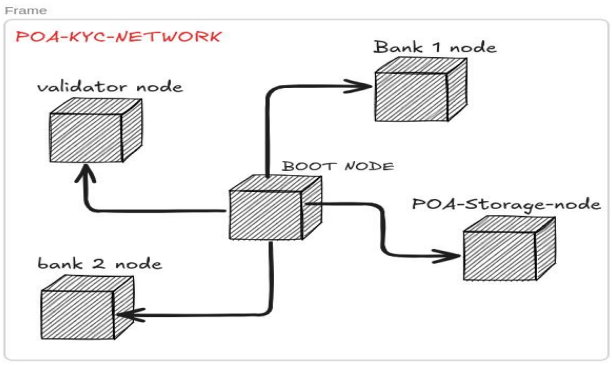

Fig 5.2 Poa Blockchain Architecture

5.3 SMART CONTRACT STRUCTURE

The smart contract structure is composed of several Solidity files, each serving a unique purpose within the network. poa-contract.sol defines rules for PoA consensus and manages validator selection. validator.sol establishes the role and responsibilities of validators within the network. The bank1.sol and bank2.sol contracts are dedicated to bank-specific operations, such as transaction processing and account queries. Additionally, KYCList.sol handles the storage and verification of KYC (Know Your Customer) data, enabling a compliant, secure banking system. Each smart contract functions autonomously yet contributes to the overall network integrity. Furthermore, these contracts are designed to be extensible, allowing future upgrades to incorporate additional functionality without compromising existing operations. Comprehensive testing ensured their resilience against vulnerabilities, bolstering the network's reliability.

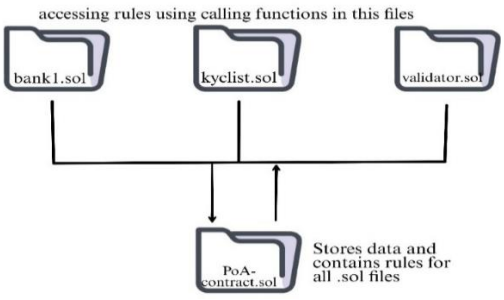

Fig 5.3: Solidity Files Structure

5.4 DOCKER CONTAINERIZATION DESIGN

Docker was utilized to containerize each blockchain node, creating an isolated environment for each to run consistently across different platforms. Containers for the bootnode, validator nodes, bank nodes, and PoA storage node ensure that the entire setup is portable and easily replicable. Docker Compose was also used to automate the deployment process, simplifying the management of dependencies and configurations across the containers. This containerized setup not only ensures system isolation but also facilitates scaling and maintenance by creating a reliable, repeatable environment for each node. By utilizing containerization, the blockchain network becomes more resilient to environment-specific issues, improving both testing and production efficiency. Regular updates to the containers can be deployed seamlessly, further enhancing system reliability.

Fig 5.4 Docker Workflow

5.5 USER ACCOUNT AND TRANSACTION MANAGEMENT

User account and transaction management are facilitated by integrating MetaMask and ethers.js. MetaMask serves as the primary wallet for user accounts, enabling secure storage and use of private keys. Ethers.js is used to interact with MetaMask, allowing users to create accounts, initiate transactions, and interact with the deployed smart contracts. This integration streamlines the user experience by securely handling account management and transaction processing, ensuring that only authenticated users can perform operations within the network. Additionally, it provides a user-friendly interface, allowing non-technical users to interact with the blockchain seamlessly. The combination of MetaMask and ethers.js enhances security while maintaining a decentralized approach to managing transactions and user identities.

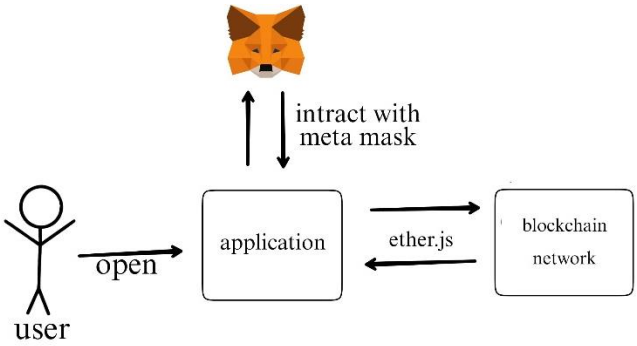

Fig 5.5 Transaction Workflow

5.6 MONITORING AND ALERTING SYSTEM

Prometheus and Grafana are employed to monitor and visualize network performance in real-time. Prometheus collects metrics from each node, including transaction volume, latency, and resource usage, which are then visualized in Grafana dashboards. Grafana's alerting system notifies administrators of potential issues, such as high latency or dropped transactions, ensuring prompt attention to maintain optimal network health. This monitoring setup provides comprehensive insights into network performance, enabling effective management and rapid troubleshooting. Furthermore, the system allows historical data analysis to identify trends and optimize network configurations. This ensures the blockchain network remains stable and well-maintained, even under high loads.

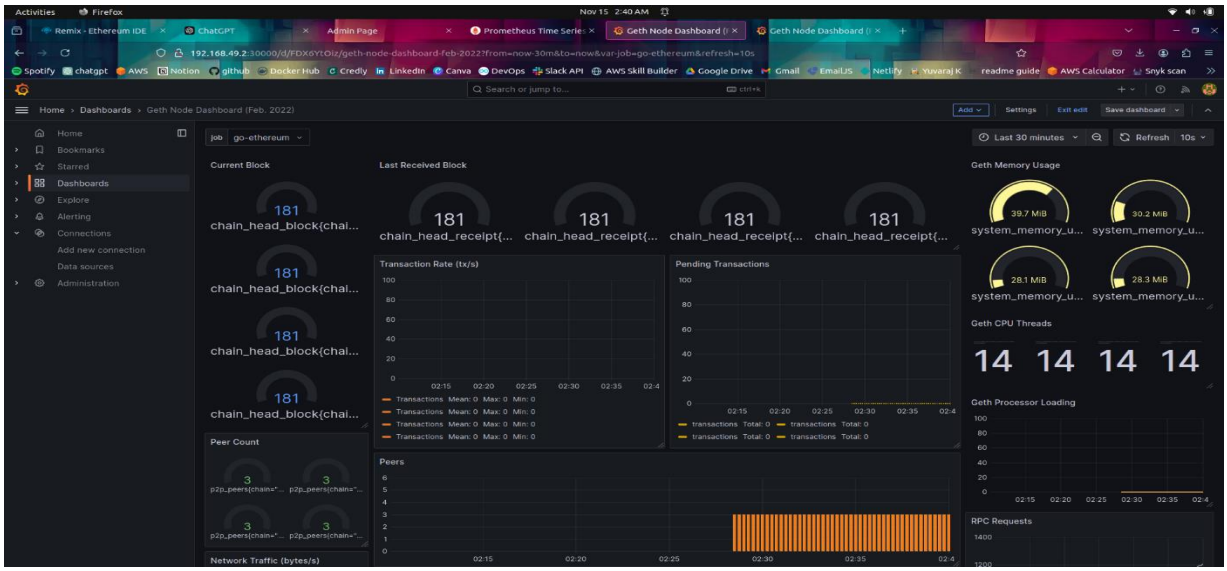

Fig 5.6: Grafana Dashboard

5.7 DEPLOYMENT AND SCALING STRATEGY WITH KUBERNETES

Kubernetes, managed with Minikube in a local development environment, is used for orchestrating node deployment and scaling within the network. Minikube allows testing of Kubernetes configurations, while Kubernetes itself handles load balancing, scaling, and automated deployment of nodes as demand fluctuates. The system is designed to dynamically add or remove nodes, ensuring that resources are efficiently allocated and that the network remains responsive to increases in transaction volume or user activity. Additionally, Kubernetes ensures high availability by redistributing workloads if nodes fail, maintaining network resilience. This deployment strategy supports a scalable, flexible infrastructure tailored to private blockchain environments.

5.8 SECURITY MEASURES AND DATA PRIVACY

The system incorporates various security measures to protect user data and ensure secure transaction processing. MetaMask is employed for secure wallet access and private key management, safeguarding user accounts and preventing unauthorized access. Transactions are signed and validated through MetaMask, ensuring authenticity. Additional security layers, including node access restrictions, are implemented to secure network data and operations, allowing only authenticated participants to interact with the blockchain. Role-based access control and encrypted communication channels further enhance security. These protocols establish a robust environment for managing sensitive data, ensuring user trust and regulatory compliance.

5.9 DATA FLOW AND WORKFLOW DIAGRAMS

Data flow within the network follows specific workflows that optimize security and efficiency. Users interact with MetaMask to initiate processes such as account creation, KYC verification, and transactions, which are then routed through ethers.js to the appropriate smart contracts on the blockchain. Each transaction follows a defined path from the user interface to MetaMask, which authenticates and forwards it to the relevant smart contract, ensuring compliance and integrity. Detailed workflow diagrams illustrate each process, clarifying the

interaction between users, MetaMask, ethers.js, and the smart contracts. These diagrams also highlight error-handling mechanisms, ensuring the system remains robust and reliable during various operations.

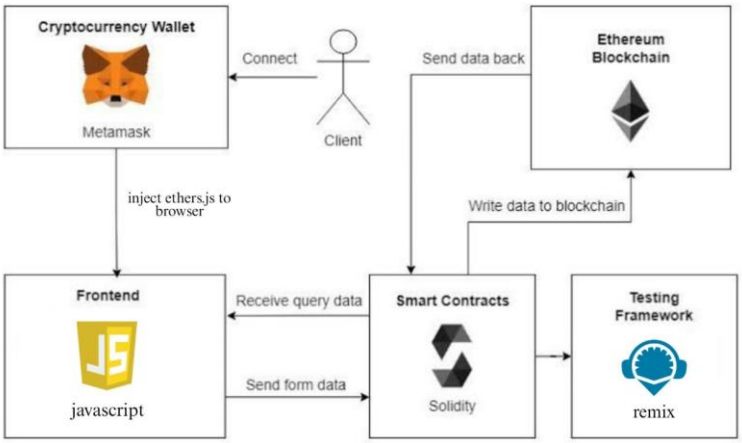

Fig 5.9: Data Flow Diagram

5.10 SYSTEM CONSTRAINTS AND ASSUMPTIONS

The system is designed for private networks using PoA consensus, with pre-approved, trusted validator nodes. It assumes sufficient computational resources for containerized nodes and Kubernetes orchestration, optimizing for private use rather than large-scale public deployment. These constraints shape the architecture to meet the intended purpose. Assumptions also include a stable network environment and participants adhering to pre-established roles and permissions. While the design is tailored for financial institutions, future scalability to public networks would require addressing additional regulatory and infrastructure challenges.

CHAPTER 6

SYSTEM REQUIREMENTS

6.1 HARDWARE REQUIREMENTS

1.Processor:

- Minimum Quad-Core CPU (Intel i5 or equivalent for local development; Intel Xeon or AMD EPYC for deployment)

2.Memory (RAM):

- Development: 8 GB minimum (16 GB recommended)
- Production/Deployment: 16-32 GB for better performance and scalability

3.Storage:

- Development: At least 25 GB SSD for storing blockchain data and Docker images
- Deployment: 50 GB SSD or higher, especially if multiple nodes are maintained on the same server

4.Graphics Processing Unit (GPU):

- Not required, as PoA consensus does not involve heavy computational processing.

5.Network Requirements:

- Stable internet connection for syncing with the private blockchain network and interacting with external monitoring tools like Prometheus.

6.Additional Hardware for Monitoring (optional):

- If using a dedicated server for monitoring, ensure it has sufficient CPU and RAM (e.g., 4-core CPU, 8 GB RAM).

6.2 SOFTWARE REQUIREMENTS

1.Operating System: Linux (Ubuntu recommended), Windows, or macOS

2.Blockchain Framework: Ethereum with Proof of Authority (PoA) consensus for private networks

3.Development Environment: Remix IDE for Solidity smart contract development, or Visual Studio Code with Solidity extensions

4.Programming Languages:

- **Solidity**: For writing smart contracts (Bank.sol and Admin.sol)
- **JavaScript**: For frontend and blockchain interaction (integrated with MetaMask)

5.Blockchain Tools:

- **MetaMask**: Browser extension for wallet management and transaction signing
- **Geth (Go-Ethereum)**: For setting up and running Ethereum nodes in the private network
- **Docker**: For containerizing the blockchain nodes
- **Minikube**: For managing and orchestrating Docker containers in a Kubernetes environment
- **Prometheus** : For monitoring network health and collecting performance metrics
- **Grafana** : Visualizing the metrics used in dashboards.

6.Web Libraries/Frameworks:

- **Ethers.js**: For interacting with the Ethereum blockchain through JavaScript
- **HTML/CSS/JavaScript**: For the user interface

7.Version Control: Git for managing code versions and collaboration

CHAPTER 7

7.1 CONCLUSION

The blockchain-based KYC verification system offers a significant improvement over traditional centralized KYC processes. By leveraging the Ethereum blockchain with a Proof of Authority (PoA) consensus mechanism, the system ensures a secure, decentralized, and transparent method of verifying customer identities across multiple banks. The use of smart contracts automates the KYC process, reducing manual intervention and ensuring data integrity. Moreover, the system's ability to maintain an immutable audit trail and streamline data verification enhances both security and efficiency. With the growing need for secure, efficient, and compliant KYC processes in the financial sector, this blockchain solution provides a modern approach that addresses existing challenges.

7.2 FUTURE ENHANCEMENTS

- Future enhancements for our blockchain-based KYC verification project could include expanding the Ethereum network for better scalability and decentralization, implementing additional security measures like multi-signature wallets or hardware security modules, and improving the user experience by integrating with government databases for streamlined KYC.
- Advanced monitoring using tools like the ELK Stack, cross-chain interoperability for broader use, and automated reporting for regulatory compliance are potential upgrades.
- Smart contract enhancements, including role-based access control and transaction fee management, as well as introducing user incentives for verified entities, would further improve the platform’s performance, security, and engagement.

CHAPTER 8

APPENDICES

APPENDIX I

SOURCE CODE

Sample Index code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Welcome to Our Bank</title>
<link rel="stylesheet" href="css/index.css">
</head>
<body>
<!-- Header section with logo and buttons -->
<header class="header">
<div class="logo">KYC-Bank</div>
<nav class="button-container">
<button onclick="location.href='bank1.html'">Bank 1</button>
<button onclick="location.href='bank2.html'">Bank 2</button>
<button onclick="location.href='admin.html'">Admin</button>
<button onclick="location.href='customer.html'">Customer</button></nav></header>
<!-- Main content section -->
<div class="container">
<div class="welcome-section">
<h1>Welcome to Online Banking</h1>
<p>Your trusted partner in financial services</p>
</div></div>
<script src="https://cdn.jsdelivr.net/npm/ethers@5.6.9/dist/ethers.umd.min.js"></script>
</body></html>
```

Sample JavaScript Code:

```
const BANK_CONTRACT_ADDRESS =
"0xF9FEA4e7213F81134c5979BA4c50f9149A77eaA2"; // Replace with your actual contract
address

const BANK_CONTRACT_ABI
=[{"inputs":[{"internalType":"uint256","name":"id","type":"uint256"}],"name":"moveToPendi
ngKYC","outputs":[],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internal
Type":"string","name":"name","type":"string"},{"internalType":"string","name":"dob","type":
"string"}],"name":"registerKYC","outputs":[{"internalType":"uint256","name":"","type":"uint
256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[{"internalType":"address
","name":"_kycStorage","type":"address"}],"stateMutability":"nonpayable","type":"constructo
r"},{"inputs":[],"name":"bankAddress","outputs":[{"internalType":"address","name":"","type"
:"address"}],"stateMutability":"view","type":"function"},{"inputs":[{"internalType":"string","
name":"name","type":"string"},{"internalType":"string","name":"dob","type":"string"}],"name
":"findKYCId","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMuta
bility":"view","type":"function"},{"inputs":[],"name":"kycStorage","outputs":[{"internalType"
:"contract
KYCStorage","name":"","type":"address"}],"stateMutability":"view","type":"function"}]

;  // Replace with the ABI of your contract

let contract;

// Manually input bank account address

const bankAccountAddress = '0x1f848B6B5FCf3D418285228830EA170f65B48612'; //
Replace this with the actual bank account address

async function connectToMetaMask() {

if (window.ethereum) {

console.log("MetaMask is available:", window.ethereum);

try {
```

```
// Request account access
await window.ethereum.request({ method: 'eth_requestAccounts' });
const provider = new ethers.providers.Web3Provider(window.ethereum);
// Manually set the bank account address as the signer
const signer = provider.getSigner(bankAccountAddress); // Use the provided bank account address
// Use the signer to create a contract instance
contract = new ethers.Contract(BANK_CONTRACT_ADDRESS, BANK_CONTRACT_ABI, signer);
// Print out the connected bank address
const signerAddress = await signer.getAddress();
console.log("Connected Bank Address:", signerAddress);
return contract;
} catch (error) {
console.error("Error connecting to MetaMask or the bank account:", error);
alert("Error connecting to MetaMask: " + error.message);}
} else {
console.error("MetaMask is not installed. Please install MetaMask to use this application.");
alert("MetaMask is not installed. Please install MetaMask.");}}
// Register a new user
async function registerUser() {
const name = document.getElementById("nameInput").value;
```

```
const dob = document.getElementById("dobInput").value;

try {

const contract = await connectToMetaMask();

const tx = await contract.registerKYC(name, dob);

await tx.wait();

console.log("KYC registered successfully with transaction:", tx);

alert("User registered successfully!");

} catch (error) {

console.error("Error registering KYC:", error);

alert("Failed to register user.");}}

// Find KYC ID by name and DOB

async function findKycId() {

const name = document.getElementById("findNameInput").value;

const dob = document.getElementById("findDobInput").value;

try {

const contract = await connectToMetaMask();

const userId = await contract.findKYCId(name, dob);

console.log("KYC ID:", userId.toString());

document.getElementById("kycIdDisplay").innerHTML = `KYC ID: ${userId.toString()}`;

} catch (error) {

console.error("Error finding KYC ID:", error);

alert("Failed to retrieve KYC ID.");}}
```

```
// Move user to pending KYC list

async function moveToPending() {

const userId = document.getElementById("userIdToPending").value;

try {

const contract = await connectToMetaMask();

const tx = await contract.moveToPendingKYC(userId);

await tx.wait();

console.log("User moved to pending successfully with transaction:", tx);

document.getElementById("moveStatusDisplay").innerHTML = `User ID ${userId} moved to pending.`;

} catch (error) {

console.error("Error moving user to pending:", error);

alert("Failed to move user to pending.");}}

// Event listeners for buttons

document.getElementById("registerButton").addEventListener("click", registerUser);

document.getElementById("findKycIdButton").addEventListener("click", findKycId);

document.getElementById("moveToPendingButton").addEventListener("click",
moveToPending);
```

Sample CSS Code:

```
/* General Styles */
body, html {
    margin: 0;
    padding: 0;
    width: 100%;
    height: 100%;
    overflow-x: hidden;
    background-color: #f4f4f4;
    font-family: Arial, sans-serif;
    box-sizing: border-box;
    display: flex;
    flex-direction: column;
    align-items: center;
}

/* Header section */
.header {
    width: 100%;
    background-color: #db981d; /* Gold background color */
    color: #000000;
    padding: 15px 20px;
    box-sizing: border-box;
    display: flex;
    justify-content: space-between;
    align-items: center;
    position: fixed;
    top: 0;
```

```
    z-index: 1;
    border-bottom: 2px solid #333; /* Separator line */
    opacity: 0;
    animation: slideIn 1s forwards; /* Slide in animation */
}

/* Animation for the header */
@keyframes slideIn {
    from {
        transform: translateY(-100%);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}

/* Logo styling */
.logo {
    font-size: 1.5em;
    font-weight: bold;
    animation: slideIn 1s ease-out 0.5s forwards; /* Animation delay for logo */
}

/* Button container within header */
.button-container {
    display: flex;
```

```
    gap: 10px;
}

/* Updated Button styling */
.button-container button {
    background-color: transparent; /* Removed background color */
    color: #000000; /* Button text color */
    border: none; /* Removed border */
    padding: 10px 15px;
    cursor: pointer;
    border-radius: 5px;
    text-align: center;
    text-transform: uppercase; /* Make text uppercase */
    font-weight: bold; /* Optional for added professionalism */
    opacity: 0;
    animation: slideIn 1s ease-out forwards;
}

.button-container button:nth-child(1) {
    animation-delay: 0.6s;
}
.button-container button:nth-child(2) {
    animation-delay: 0.8s;
}
.button-container button:nth-child(3) {
    animation-delay: 1s;
}
.button-container button:nth-child(4) {
```

```
    animation-delay: 1.2s;
}

.button-container button:hover {
    background-color: #db981d; /* Hover background color */
    color: white; /* Change text color on hover */
}

/* Main content container */
.container {
    width: 100%;
    height: 100vh; /* Ensure the container takes up the full height */
    display: flex;
    justify-content: center;
    align-items: center;
    padding-top: 100px; /* Space for the header */
    box-sizing: border-box;
    background-image:
url('https://i.pinimg.com/1200x/fc/58/36/fc583694de50374e536d84564579919e.jpg'); /*
Background image for main content */
    background-size: cover; /* Ensure the image covers the entire container */
    background-position: center; /* Center the image */
    background-attachment: fixed; /* Keep the background fixed when scrolling */
    animation: fadeIn 1.5s forwards; /* Animation for main content */
}

/* Animation for fading in the main content */
@keyframes fadeIn {
    from {
```

```
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}

/* Welcome section styling */
.welcome-section {
    text-align: center;
    color: white;
    text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.8);
    opacity: 0;
    animation: slideInText 1.5s ease-out forwards;
}

/* Animation for sliding in welcome text */
@keyframes slideInText {
    from {
        transform: translateY(50px);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}
```

```
/* Welcome text styles */
.welcome-section h1 {
    font-size: 3em;
    margin: 0;
}

.welcome-section p {
    font-size: 1.2em;
    margin-top: 10px;
    animation: slideInText 1.5s ease-out 0.5s forwards; /* Animation for the welcome message
*/
}
```

Sample Solidity Code:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract KYCStorage {

struct KYCInfo {

uint id;

string name;

string dob;  // Date of birth in YYYY-MM-DD format

address registeredBy;  // Address of the bank that registered the KYC}

// Mappings for each KYC status

mapping(uint => KYCInfo) public registeredKYC;

mapping(uint => KYCInfo) public pendingKYC;

mapping(uint => KYCInfo) public approvedKYC;

mapping(uint => KYCInfo) public rejectedKYC;

// Track existence in each mapping

mapping(uint => bool) public isRegistered;

mapping(uint => bool) public isPending;

mapping(uint => bool) public isApproved;

mapping(uint => bool) public isRejected;

uint private nextId = 1;

event Notify(string message);

// Add to registered KYC
```

```
function addRegisteredKYC(string memory name, string memory dob, address bankAddress)
external returns (uint) {

registeredKYC[nextId] = KYCInfo(nextId, name, dob, bankAddress);

isRegistered[nextId] = true;

emit Notify("KYC is registered Successfully.");

return nextId++;}

// Move to pending KYC with checks for existing approved KYC using name and dob

function moveToPendingKYC(uint id) external {

require(isRegistered[id], "KYC must be registered before moving to pending");

// Check if KYC is already approved by name and date of birth

for (uint i = 1; i < nextId; i++) {

if (isApproved[i] &&

keccak256(abi.encodePacked(approvedKYC[i].name)) ==
keccak256(abi.encodePacked(registeredKYC[id].name)) &&

keccak256(abi.encodePacked(approvedKYC[i].dob)) ==
keccak256(abi.encodePacked(registeredKYC[id].dob))) {

// Remove from registered KYC

delete registeredKYC[id];

isRegistered[id] = false;

emit Notify("KYC already approved with this name and date of birth");

return;}

if (isPending[i] &&
```

```
keccak256(abi.encodePacked(pendingKYC[i].name)) ==
keccak256(abi.encodePacked(registeredKYC[id].name)) &&

keccak256(abi.encodePacked(pendingKYC[i].dob)) ==
keccak256(abi.encodePacked(registeredKYC[id].dob))) {

// Remove from registered KYC

delete registeredKYC[id];

isRegistered[id] = false;

emit Notify("KYC already pending with this name and date of birth");

return;}}

// Move to pending

pendingKYC[id] = registeredKYC[id];

isPending[id] = true;

// Remove from registered KYC

delete registeredKYC[id];

isRegistered[id] = false;

emit Notify("KYC is move to Verification Status Successfully.");}

// Approve KYC

function approveKYC(uint id) external {

require(isPending[id], "KYC must be in pending status to approve");

approvedKYC[id] = pendingKYC[id];

isApproved[id] = true;

// Remove from pending KYC
```

```
delete pendingKYC[id];

isPending[id] = false;

emit Notify("KYC is Approve Successfully.");}

// Reject KYC

function rejectKYC(uint id) external {

require(isPending[id], "KYC must be in pending status to reject");

rejectedKYC[id] = pendingKYC[id];

isRejected[id] = true;

// Remove from pending KYC

delete pendingKYC[id];

isPending[id] = false;

emit Notify("KYC is Rejected.");}

// Get KYC Status

function getKYCStatus(uint id) external view returns (string memory) {

if (isRegistered[id]) {

return "Registered";

} else if (isPending[id]) {

return "Pending";

} else if (isApproved[id]) {

return "Approved";

} else if (isRejected[id]) {

return "Rejected";
```

```
} else {
return "KYC ID does not exist";}}
// Function to find KYC ID by name and date of birth
function findKYCId(string memory name, string memory dob) external view returns (uint) {
for (uint i = 1; i < nextId; i++) { // Start from 1 assuming IDs start from 1
if (isRegistered[i] &&
keccak256(abi.encodePacked(registeredKYC[i].name)) ==
keccak256(abi.encodePacked(name)) &&
keccak256(abi.encodePacked(registeredKYC[i].dob)) == keccak256(abi.encodePacked(dob)))
{
return registeredKYC[i].id; // Return the ID if found}}
revert("KYC not found for the provided name and date of birth");}
// List all registered KYC data
function listRegisteredKYC() external view returns (KYCInfo[] memory) {
KYCInfo[] memory result = new KYCInfo[](nextId);
uint count = 0;
for (uint i = 1; i < nextId; i++) { // Start from 1 assuming IDs start from 1
if (isRegistered[i]) {
result[count] = registeredKYC[i];
count++;}}
// Resize the array to the actual number of elements
assembly { mstore(result, count) }
```

```
return result;}

// List all pending KYC data

function listPendingKYC() external view returns (KYCInfo[] memory) {

KYCInfo[] memory result = new KYCInfo[](nextId);

uint count = 0;

for (uint i = 1; i < nextId; i++) { // Start from 1

if (isPending[i]) {

result[count] = pendingKYC[i];

count++;}}

assembly { mstore(result, count) }

return result;}

// List all approved KYC data

function listApprovedKYC() external view returns (KYCInfo[] memory) {

KYCInfo[] memory result = new KYCInfo[](nextId);

uint count = 0;

for (uint i = 1; i < nextId; i++) { // Start from 1

if (isApproved[i]) {

result[count] = approvedKYC[i];

count++;}}

assembly { mstore(result, count) }

return result;}

// List all rejected KYC data
```

```
function listRejectedKYC() external view returns (KYCInfo[] memory) {

KYCInfo[] memory result = new KYCInfo[](nextId);

uint count = 0;

for (uint i = 1; i < nextId; i++) { // Start from 1

if (isRejected[i]) {

result[count] = rejectedKYC[i];

count++;}}

assembly { mstore(result, count) }

return result;}
```

APPENDIX II

OUTPUT WITH SCREENSHOTS

- **Bootnode:** Allows other nodes to connect and join the network.

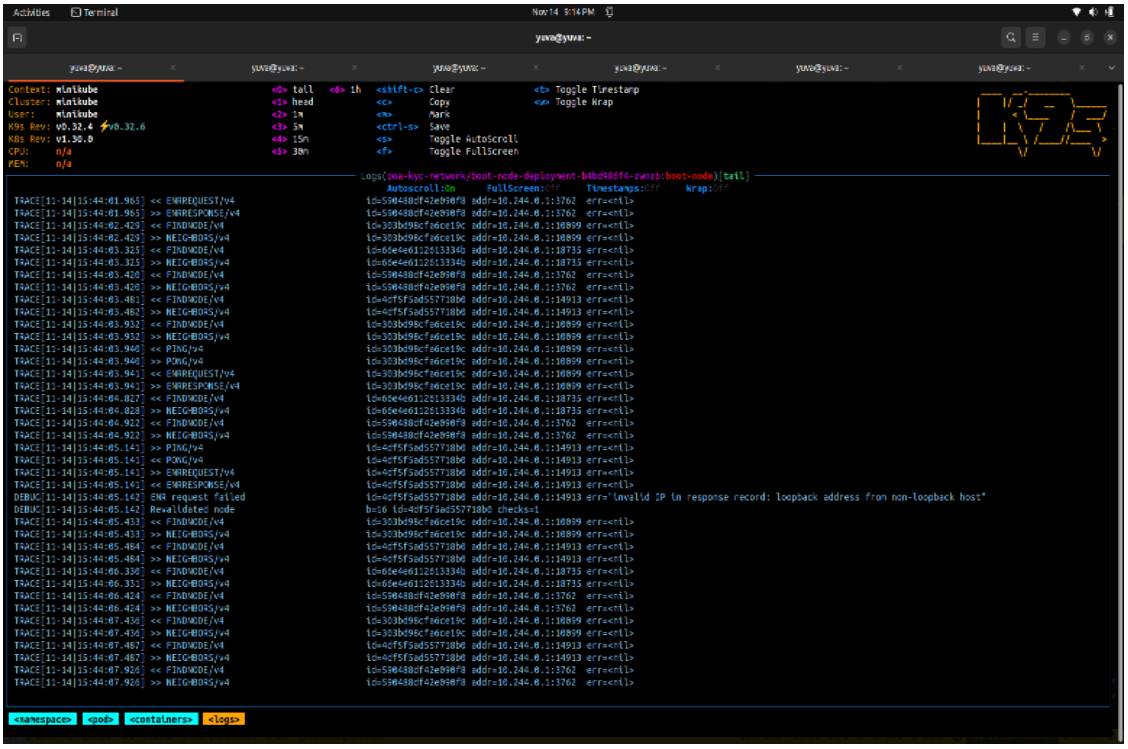

Fig 8.1: Boot Node

- **Validator Node:** Validates transactions and secures the blockchain.

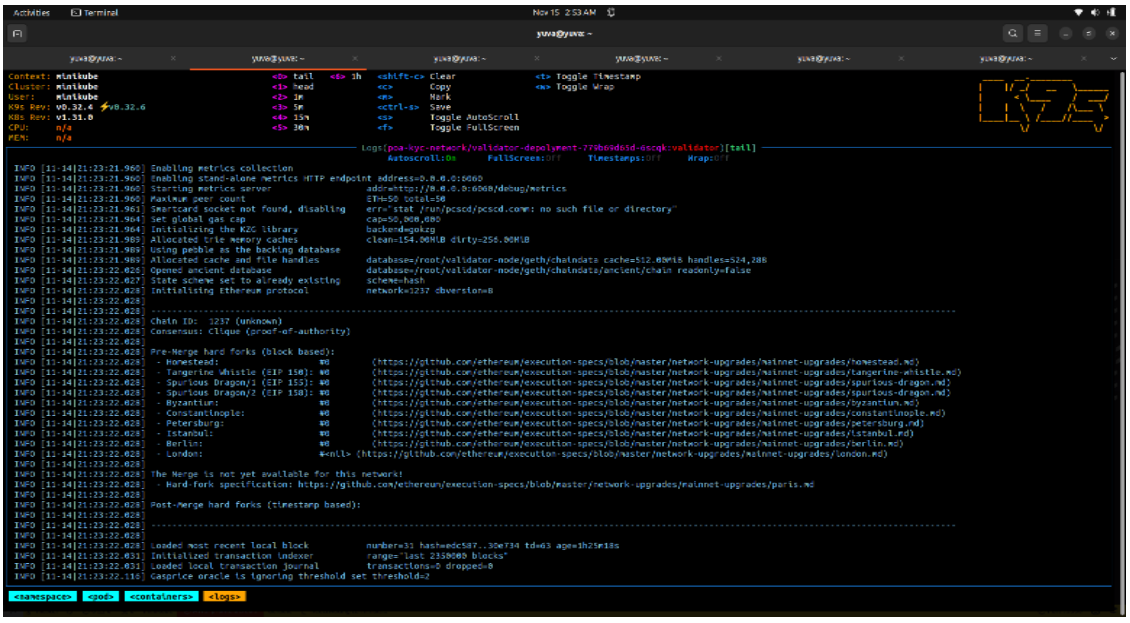

Fig 8.2: Validator Node

- **Bank1 Node and Bank2 Node:** Represent two banks that handle transactions and interact with smart contracts.

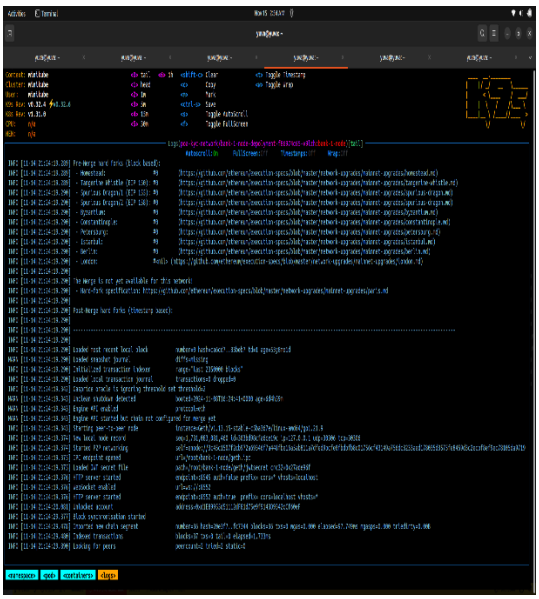

Fig 8.3: Bank1 Node

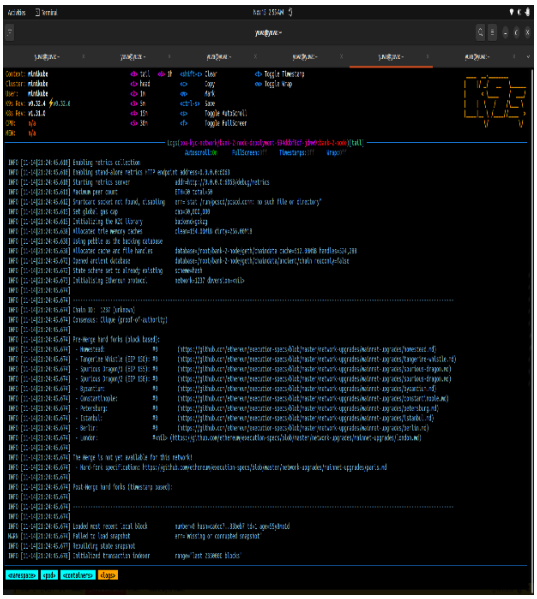

Fig 8.3: Bank2 Node

- **PoA Storage Node:** Stores data under the Proof of Authority (PoA) consensus for fast, secure storage.

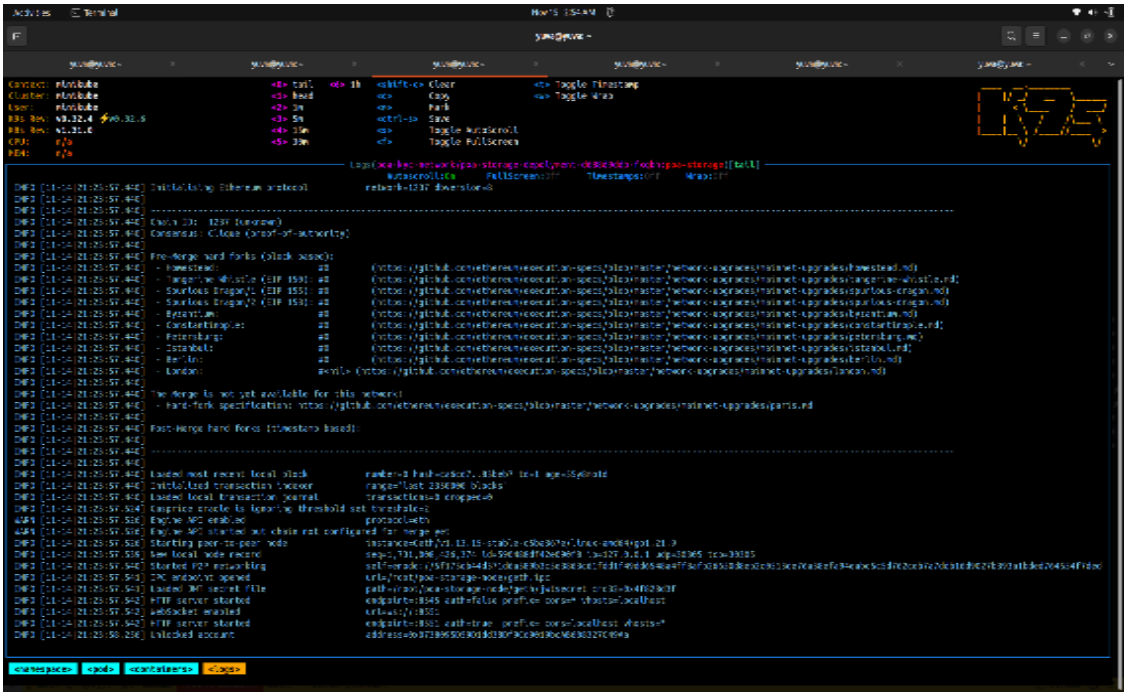

Fig 8.4: Poa Storage Node

- **Prometheus:** A monitoring system that collects metrics from your Ethereum nodes (e.g., Admin, Bank1, Bank2) such as CPU usage, memory usage, and transaction count.
- **Grafana:** A visualization tool used to display the collected metrics in real-time through dashboards.

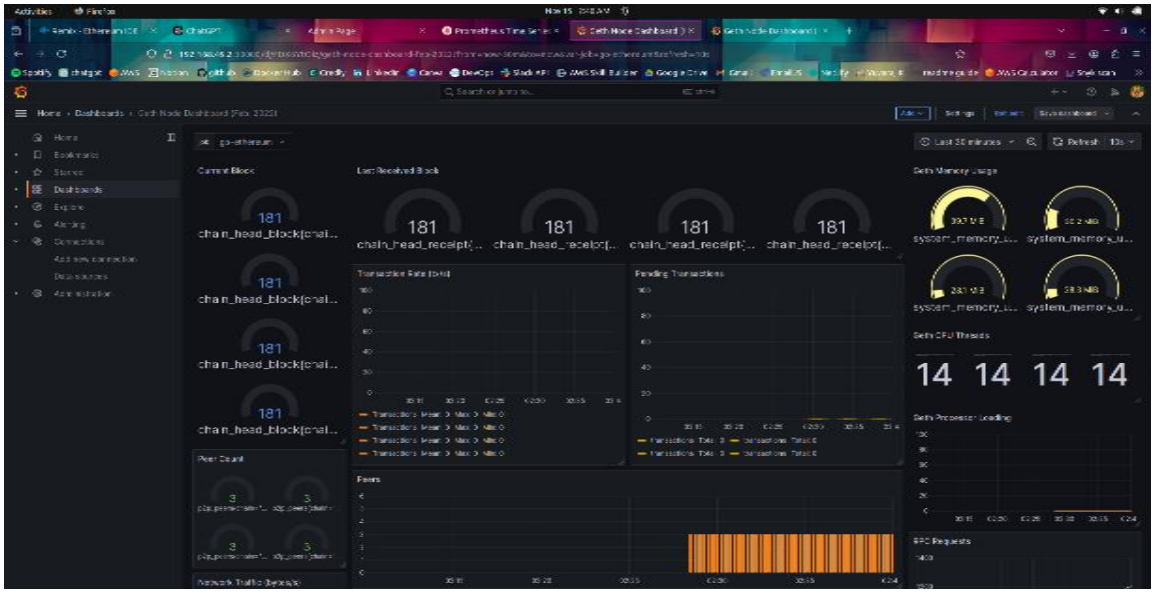

Fig 8.5: Prometheus & Grafana Dashboards

- **Bank page:** It is used for user registeration, Find KYC ID and Move to Pending process.

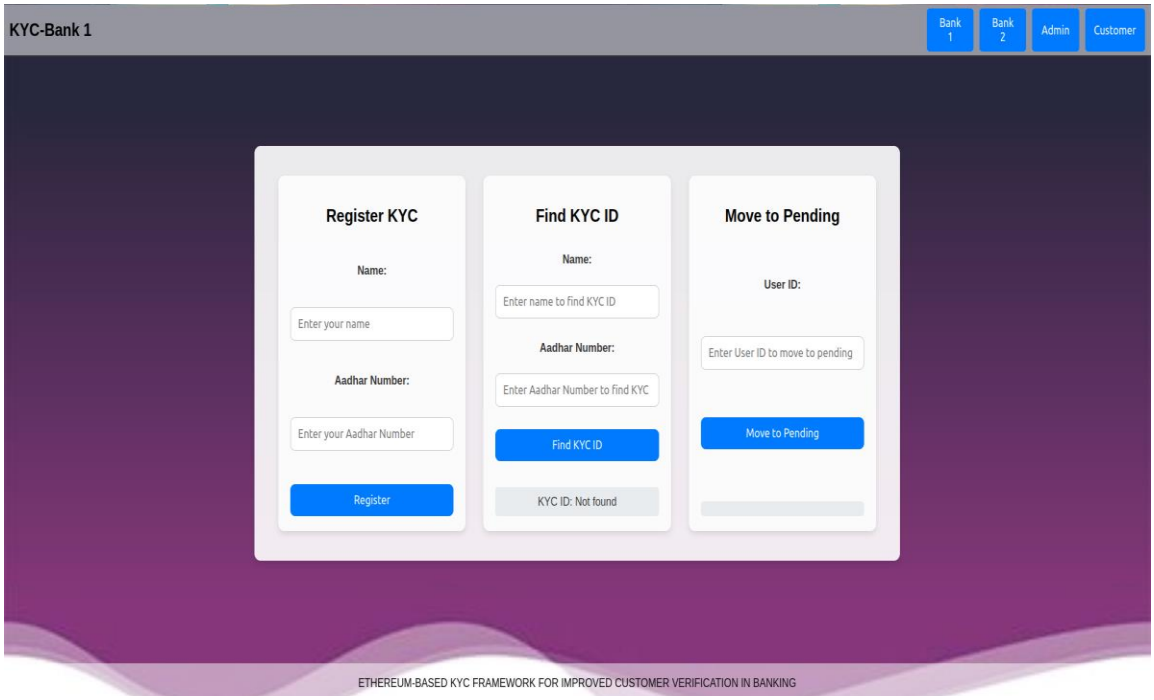

Fig 8.6: Bank Page

- **Admin page:** In this page administrator can do a process like approve or rejection process and get KYC status.

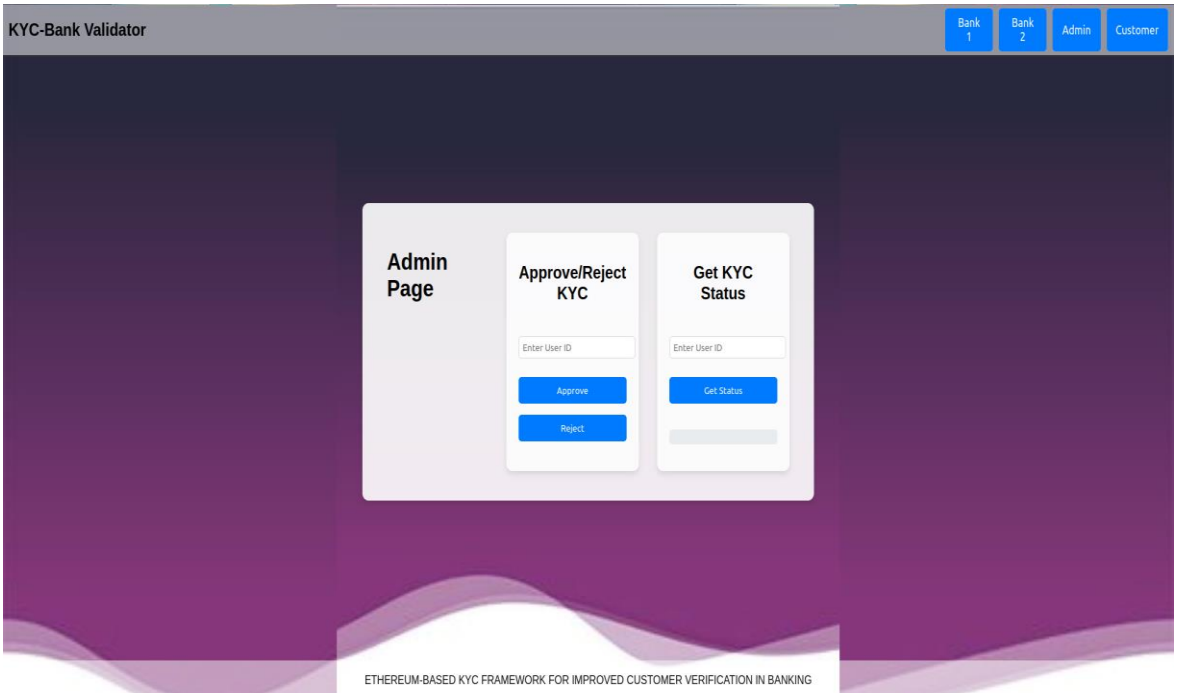

Fig 8.7: Admin Page

- **Customer page:** In this page admins can check who are all registered and their status.

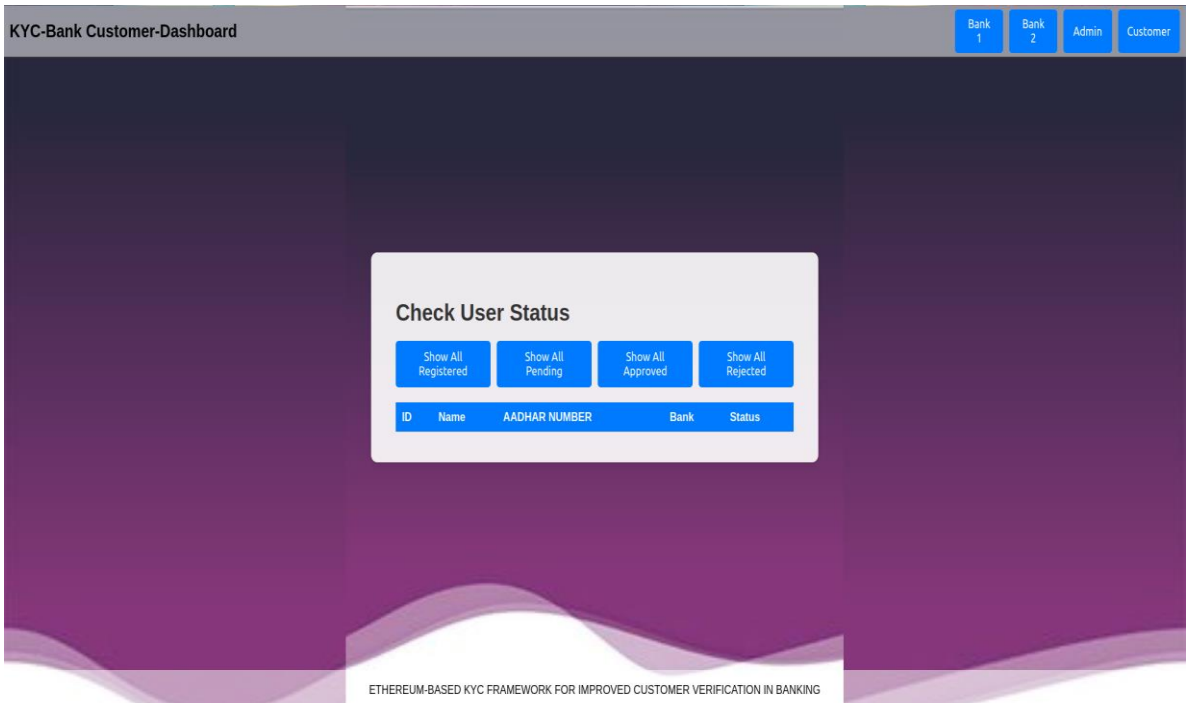

Fig 8.8: Customer Page

REFERENCES

1. M. N. M. Bhutta, A. A. Khwaja, A. Nadeem, H. F. Ahmad, M. K. Khan, M.A.Hanif, H.Song,M. Alshamari and Y.Cao ,‘‘A survey on blockchain technology: Evolution, architecture and security,’’ IEEE Access, vol. 9, pp. 61048–61073, 2021.

2. J. Gomes, S. Khan, and D. Svetinovic, ‘‘Fortifying the blockchain: A systematic review and classification of post-quantum consensus solutions for enhanced security and resilience,’’ IEEE Access, vol. 11, pp. 74088–74100, 2023.

3. V. Kumar C and P. Selvaprabhu, ‘‘An examination of distributed and decentralized systems for trustworthy control of supply chains,’’ IEEE Access, vol. 11, pp. 137025–137052, 2023.

4. R. Norvill, M. Steichen, W. M. Shbair, and R. State, ‘‘Demo: Blockchain for the simplification and automation of KYC result sharing,’’ in Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC), May 2023, pp. 9–10.

5. B. Karadag, A. Halim Zaim, and A. Akbulut, ‘‘Blockchain-based KYC model for credit allocation in banking,’’ IEEE Access, vol. 12, pp. 80176–80182, 2024.

6. N. Mansoor, K. F. Antora, P. Deb, T. A. Arman, A. A. Manaf, and M. Zareei, ‘‘A review of blockchain approaches for KYC,’’ IEEE Access, vol. 11, pp. 121013–121042, 2023.

7. D. George, A. Wani, and A. Bhatia, ‘‘A blockchain based solution to know your customer (KYC) dilemma,’’ in Proc. IEEE Int. Conf. Adv. Netw. Telecommun. Syst. (ANTS), Dec. 2023, pp. 1–6.

8. H. Alanzi and M. Alkhatib, ‘‘Towards improving privacy and security of identity management systems using blockchain technology: A systematic review,’’ Appl. Sci., vol. 12, no. 23, p. 12415, Dec. 2022.

9.R. Patel, M. Migliavacca, and M. E. Oriani, ‘‘Blockchain in banking and finance : A bibliometric review,’’ Res. Int. Bus. Finance, vol. 62, Dec. 2022, Art. no. 101718.

10. M. A. Hannan, M. A. Shahriar, M. S. Ferdous, M. J. M. Chowdhury, and M.S.Rahman,‘‘A systematic literature review of blockchain-based e-KYC systems,’’ Computing, vol. 105, no. 10, pp. 2089–2118, Oct. 2023.