



**K S R INSTITUTE FOR ENGINEERING AND TECHNOLOGY**

**TIRUCHENGODE: 637 215**

**Computer Science and Engineering**

**NAAN MUDHALVAN**  
*SB8024- Blockchain Development*  
*by Naan Mudhalvan Scheme – 2023*

**TEAM ID: NM2023TMID11744**

**PROJECT DOMAIN: CHAIN CONNECT NODE IN BLOCKCHAIN**

**TEAM MEMBERS**

**REGISTER NUMBER**

**NAME**

**731620104044**

**SABARIMUGILAN M**

**731620104057**

**VASANTH R**

**731620104062**

**YASH RAJ**

**731620104301**

**MUTHU PALANIAPPAN G**

# TABLE OF CONTENT

S.No	CONTENT
<b>1</b>	<b>INTRODUCTION</b>
	1.1 Project Overview
	1.2 Purpose
<b>2</b>	<b>LITERATURE SURVEY</b>
	2.1 Existing problem
	2.2 References
	2.3 Problem Statement Definition
<b>3</b>	<b>IDEATION &amp; PROPOSE SOLUTION</b>
	3.1 Empathy Map Canvas
	3.2 Ideation & Brainstorming
<b>4</b>	<b>REQUIREMENT ANALYSIS</b>
	4.1 Functional requirement
	4.2 Non-Functional requirements
<b>5</b>	<b>PROJECT DESIGN</b>
	5.1 Data Flow Diagrams & User Stories
	5.2 Solution Architecture
<b>6</b>	<b>PROJECT PLANNING &amp; SCHEDULING</b>
	6.1 Technical Architecture
	6.2 Sprint Planning & Estimation
	6.3 Sprint Delivery Schedule
<b>7</b>	<b>CODING &amp; SOLUTIONING</b>
	7.1 Feature 1
	7.2 Feature 2
	7.3 Database Schema
<b>8</b>	<b>PERFORMANCE TESTING</b>
	8.1 Performace Metrics
<b>9</b>	<b>RESULTS</b>
	9.1 Output Screenshots
<b>10</b>	<b>ADVANTAGES &amp; DISADVANTAGES</b>
<b>11</b>	<b>CONCLUSION</b>
<b>12</b>	<b>FUTURE SCOPE</b>
<b>13</b>	<b>APPENDIX</b> Source Code GitHub & Project Demo Link

# 1. INTRODUCTION

## 1.1 PROJECT OVERVIEW

Blockchain technology has emerged as a transformative force, revolutionizing the way nodes connect and communicate within decentralized networks. This project overview aims to provide insight into the fundamental principles and objectives of a blockchain-based initiative that focuses on connecting nodes in a secure, transparent, and decentralized manner. The use of blockchain technology in this context facilitates trust, efficiency, and data integrity, making it applicable to a wide range of industries and applications.

## 1.2 PURPOSE

To reduce reliance on central authorities or intermediaries, the project aims to create a decentralized network. By doing so, it eliminates single points of failure, enhances network resilience, and empowers individuals and entities to participate in a trustless manner. One of the central purposes is to enable secure and tamper-proof data exchange among nodes. By leveraging blockchain's cryptographic techniques and consensus mechanisms, the project ensures that data transmitted between nodes is secure, authentic, and immutable. The project fosters trust and transparency within the network. Participants can have confidence in the integrity of the shared ledger, knowing that the information contained within it is verifiable and cannot be altered without consensus. This project seeks to explore and implement practical use cases for the blockchain network. Whether it's optimizing supply chain management, automating smart contracts, or facilitating secure data sharing, the network's purpose extends to various industries and applications.

## **2.LITERATURE SURVEY**

Decentralized identity management on the Ethereum blockchain has emerged as a crucial area of research, offering a promising solution to the challenges associated with traditional identity systems. By leveraging the capabilities of Ethereum smart contracts, decentralized identity solutions enable users to have full control over their digital identities, ensuring privacy, security, and autonomy. A comprehensive analysis of the existing literature reveals various methodologies and approaches for implementing decentralized identity on the Ethereum blockchain, highlighting the unique features and functionalities offered by Ethereum's smart contract technology.

### **2.1Existing Problem**

There are numerous blockchain networks in existence, with some of the most well-known being Bitcoin (BTC), Ethereum (ETH), and Binance Smart Chain (BSC). Each of these networks has its unique features, consensus mechanisms, and use cases. Participants who wish to join a blockchain network can set up their own nodes. These nodes can be established on individual computers, servers, or specialized hardware, depending on the network's requirements. Blockchain networks are typically structured as peer-to-peer networks. Nodes connect to one another in a decentralized fashion, which means that there is no central server or authority controlling the network. They communicate directly with other nodes to exchange data, verify transactions, and reach consensus. Each blockchain network employs a specific consensus mechanism to validate and add transactions to the ledger. For example, Bitcoin uses Proof of Work (PoW), while Ethereum is transitioning from PoW to Proof of Stake (PoS). These mechanisms ensure that nodes agree on the state of the blockchain. Blockchain networks use cryptographic techniques to secure data. Transactions are cryptographically signed, and the data is recorded in blocks that are linked together in a chain. This creates a tamper-resistant and immutable ledger.

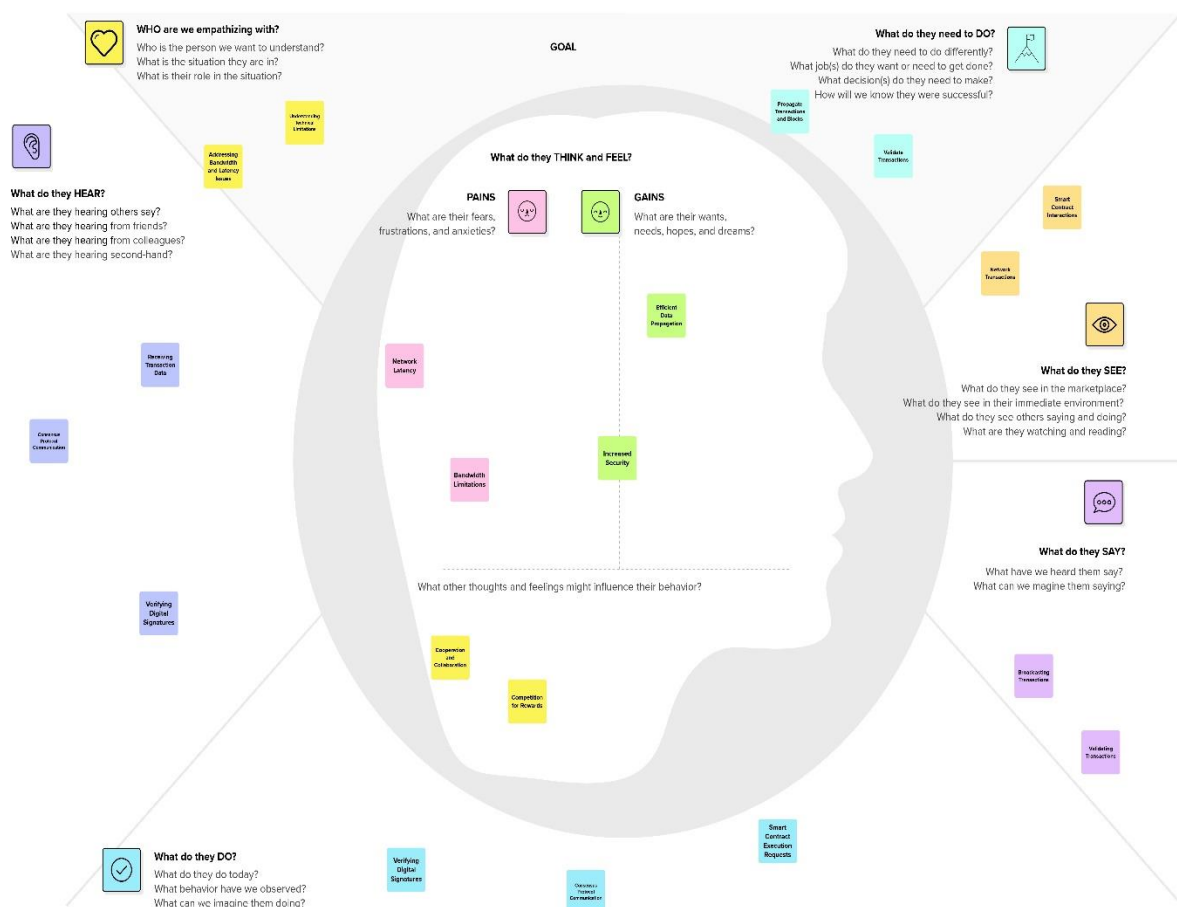
### **2.2REFERENCES**

- Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System." (2008).

- Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. "Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction." (2016).
- Antonopoulos, A. M. "Mastering Bitcoin: Unlocking Digital Cryptocurrencies." O'Reilly Media, Inc. (2014).
- Tapscott, D., & Tapscott, A. "Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World." Portfolio. (2016).
- Buterin, V., "Ethereum White Paper: A Next-Generation Smart Contract and Decentralized Application Platform." (2013).

## **2.2 Problem Statement Definition**


The existing system of connecting nodes using blockchain technology faces challenges related to centralization, data security, trust, cost, accessibility, scalability, energy consumption, lack of standardization, user-friendliness, and regulatory compliance. These issues hinder the full potential of blockchain networks in providing decentralized, secure, and efficient connectivity. The project's purpose is to address these challenges and create a solution that empowers individuals and entities to participate in a trustless, secure, and transparent manner while exploring practical use cases in various industries and applications.



## 3.2 Ideation and Brainstorming


When brainstorming for an Chain connect nodes smart contract, numerous innovative possibilities emerge to enhance the system's functionality and security. One avenue to explore involves integrating biometric data for robust identity verification, thereby bolstering the system's overall security. Additionally, incorporating multi-signature authentication processes can fortify the integrity of sensitive operations and data access. Leveraging Chain connect nodes blockchain for creating immutable identity records ensures transparency, traceability, and resistance to unauthorized modifications. Integrating decentralized storage solutions such as IPFS can safeguard sensitive identity data, mitigating the risks of data breaches and bolstering user privacy


Template





## Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

 10 minutes to prepare


 1 hour to collaborate

 2-8 people recommended



### Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

 10 minutes

A

Team gathering

Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B

Set the goal

Think about the problem you'll be focusing on solving in the brainstorming session.

C

Learn how to use the facilitation tools

Use the Facilitation Superpowers to run a happy and productive session.


Open article

→

1


### Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

 5 minutes


PROBLEM


How might we [your problem statement]?





### Key rules of brainstorming


To run a smooth and productive session


 Stay in topic.

 Encourage wild ideas.

 Defer judgment.

 Listen to others.

 Go for volume.

 If possible, be visual.

2

## Brainstorm

Write down any ideas that come to mind that address your problem statement.

 10 minutes

### TIP



You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

**Sabarimugilan**

P2P  
Networking  
Protocols

Discovery  
Mechanisms

Bootstrap  
Nodes

**Vasanth**

Network  
Address  
Transition  
(NAT)  
Traversal

Incentive  
Mechanisms

Network  
Security

**Yashraj**

Peer  
Reputation  
System

Automatic  
Peer  
Discovery

Regular  
Network  
Health  
Checks

**Muthu palaniappan**

Dynamic  
Connection  
Management

Resilient  
Peer-to-  
Peer  
Overlay

Cross-Chain  
Communication



3

## Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

 20 minutes

### TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

**Sabarimugilan**

Network Discovery

Establishment

Security

**Vasanth**

Reliability

Incentivization

Collaboration

**Yashraj**

Network Resilience

Adaptability

Connection Management

**Muthu palaniappan**

Scalability and Load Balancing

Protocol Optimization

Network Monitoring

4

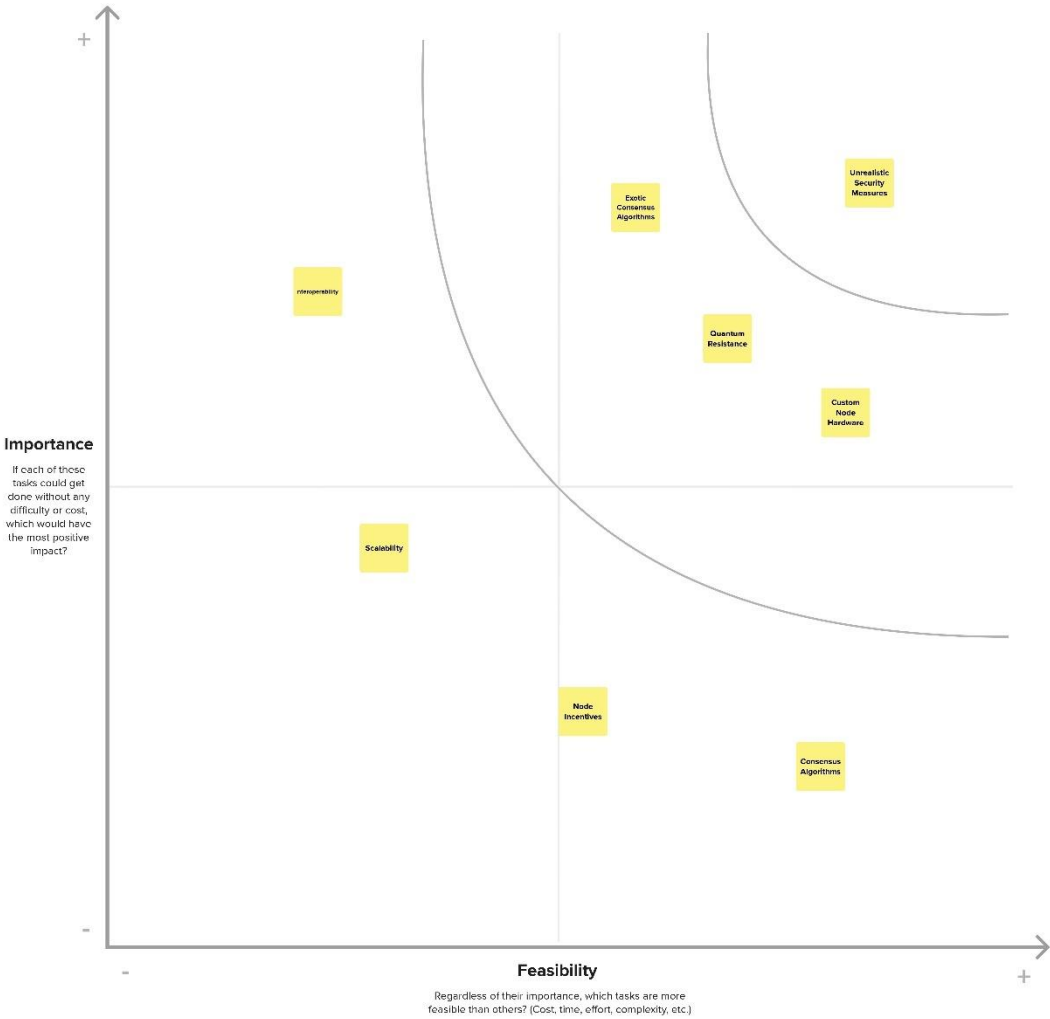
### Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

🕒 20 minutes

**TIP**

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.



## **4.REQUIREMENTANALYSIS**

### **4.1 Functional Requirements**

#### **User Management:**

##### **1.Technical Requirements:**

- a. Blockchain Network Selection: Choose the appropriate blockchain network or technology stack based on the specific use case. Consider factors such as consensus mechanism (e.g., PoW, PoS), scalability, and smart contract capabilities.
- b. Node Setup and Configuration: Define the technical specifications for setting up nodes, including hardware requirements, software installation, and configuration settings.
- c. Security Measures: Implement robust security protocols, including encryption, access controls, and private keys management, to protect data and transactions.
- d. Consensus Mechanism: Determine the consensus mechanism that best suits the network's goals (e.g., PoW, PoS, Delegated Proof of Stake). Ensure it aligns with the desired level of security and scalability.
- e. Data Encryption: Specify encryption methods to secure data at rest and in transit. This includes encryption of transaction data and communication between nodes.
- f. Scalability: Address the network's scalability requirements, considering the potential growth in the number of nodes and transaction volume. Evaluate and implement solutions for scalability.
- g. Interoperability: Define how the blockchain network will interact with other systems, networks, and legacy infrastructure. Consider the need for APIs or data exchange protocols.

## **2.Operational Requirements:**

- a. Node Maintenance: Establish procedures for node management, maintenance, and updates to ensure optimal performance and security.
- b. Monitoring and Reporting: Implement tools and protocols for real-time monitoring of the network's health, performance, and security. Define reporting mechanisms for system status and issues.
- c. Backup and Recovery: Develop backup and disaster recovery strategies to ensure the resilience of the blockchain network in case of data loss or system failures.
- d. Regulatory Compliance: Address legal and regulatory requirements that may apply to the blockchain network, ensuring compliance with relevant laws and standards.
- e. Documentation: Create comprehensive documentation for node setup, network operations, and troubleshooting. This documentation should be accessible to network participants.

## **3.User Requirements:**

- a. User Interfaces: Design user-friendly interfaces for participants to interact with the blockchain network. This includes wallet management, transaction tracking, and participation in consensus mechanisms.
- b. Educational Resources: Develop educational materials and resources to assist users in setting up and managing their nodes effectively. This includes tutorials, guides, and FAQs.
- c. Accessibility: Ensure that the network is accessible to a wide range of participants, regardless of technical expertise, geographical location, or economic status.

d. User Support: Establish a system for user support, including helpdesk or customer service channels, to assist participants in addressing issues and inquiries.

e. Identity Verification: Define procedures for identity verification and KYC (Know Your Customer) processes, if applicable, to meet legal and regulatory requirements.

#### **4. Use Case Requirements:**

a. Use Case Definition: Clearly define the specific use cases for the blockchain network, whether it's optimizing supply chain management, automating smart contracts, or facilitating secure data sharing.

b. Feature Requirements: Identify the features and functionalities required to support the chosen use cases. This may include smart contract templates, data sharing protocols, and transaction logic.

#### **5. Testing and Quality Assurance:**

a. Testing Strategy: Develop a comprehensive testing strategy, including unit testing, integration testing, and security testing, to ensure the reliability and robustness of the system.

b. Quality Assurance: Implement quality assurance processes to identify and address issues and vulnerabilities in the network.

#### **6. Project Timeline and Milestones:**

a. Project Phases: Define the phases of the project, including planning, development, testing, deployment, and ongoing maintenance.

b. Milestone Targets: Set specific milestone targets for each phase, with clear deliverables and timelines.

## **7.Resource Requirements:**

- a. Human Resources: Determine the required team members, including developers, security experts, and operational staff.
- b. Infrastructure: Identify the necessary hardware, software, and cloud resources needed to support the blockchain network.

## **8.Budget:**

- a. Financial Planning: Develop a budget that outlines the expected costs associated with the project, including development, infrastructure, operational, and marketing expenses.

## **9.Scalability Plan:**

- a. Future Expansion: Outline plans for future expansion and growth, considering the potential increase in nodes, transactions, and use cases.

## **4.2 Non-Functional Requirements**

### **1.Performance:**

- a. Response Time: Define acceptable response times for transactions and data retrieval within the blockchain network.
- b. Scalability: Specify the ability of the network to handle an increasing number of nodes and transactions without significant performance degradation.
- c. Throughput: Determine the network's transaction processing capacity per unit of time, ensuring it meets the demands of the use case.

### **2. Security:**

- a. Data Encryption: Specify encryption standards for data at rest and in transit to protect against unauthorized access or data breaches.
- b. Access Control: Define user access control mechanisms to restrict access to specific features or data based on user roles and permissions.

- c. Audit Trails: Implement mechanisms for recording and auditing all user activities and transactions to maintain transparency and traceability.
- d. Compliance: Ensure that the blockchain network complies with relevant regulatory and legal requirements, such as data protection and anti-money laundering regulations.

### **3. Reliability:**

- a. Availability: Define the required level of network availability and establish procedures for minimizing downtime due to maintenance or unforeseen issues.
- b. Redundancy: Specify redundancy mechanisms for critical components to ensure system availability in case of failures.

### **4. Scalability:**

- a. Horizontal Scaling: Plan for the ability to add more nodes to the network to accommodate increased demand, ensuring that the system remains responsive.
- b. Vertical Scaling: Define the capacity to increase the resources of individual nodes or components within the network.

### **5. Interoperability:**

- a. APIs and Standards: Ensure that the blockchain network adheres to common standards and provides well-documented APIs for seamless integration with other systems and services.

### **6. Usability:**

- a. User Interface Design: Create user interfaces that are intuitive, user-friendly, and accessible to a diverse user base.

b. User Training and Support: Provide resources and support for users to navigate and utilize the network effectively.

## **7. Resource Efficiency:**

a. Energy Consumption: Minimize energy consumption, especially if using energy-intensive consensus mechanisms like Proof of Work (PoW).

b. Resource Utilization: Optimize resource utilization, such as CPU and memory, to ensure efficient performance.

## **8. Documentation:**

a. Comprehensive Documentation: Provide detailed and up-to-date documentation for node setup, usage, and troubleshooting, catering to both technical and non-technical users.

## **9. Monitoring and Reporting:**

a. Real-time Monitoring: Implement continuous monitoring of the network's health and performance, with alerts for any issues.

b. Reporting: Develop reporting mechanisms for system status, performance metrics, and incident reports.

## **10. Compliance with Blockchain Standards:**

a. Blockchain Protocol Standards: Ensure that the network adheres to industry-standard blockchain protocols and best practices for interoperability and security.

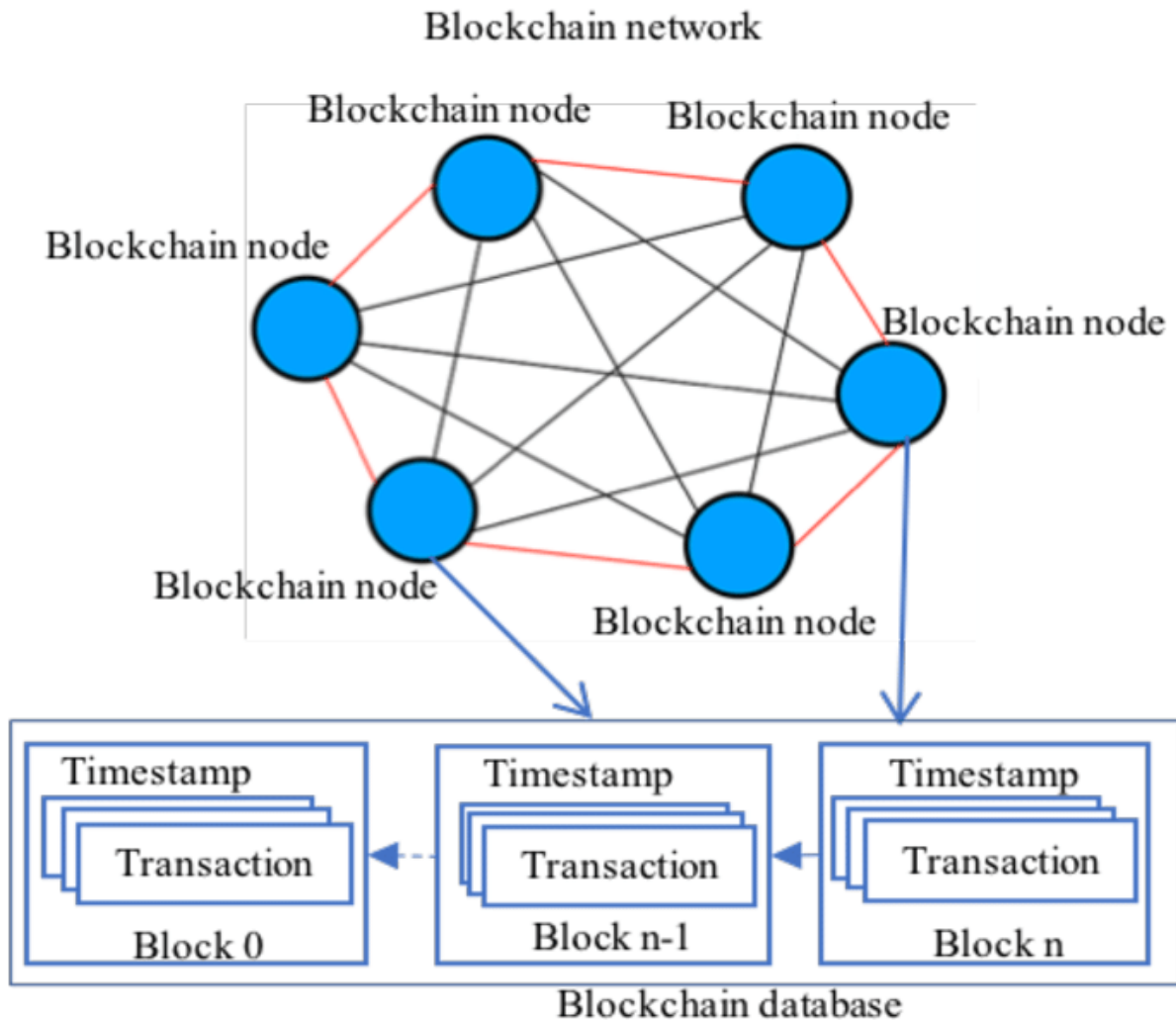
## **11. Legal and Ethical Considerations:**

Data Privacy: Address data privacy requirements, especially for personally identifiable information (PII), and provide transparency on how user data is handled.



## 5. PROJECT DESIGN

### 5.1 DATAFLOW DIAGRAM AND USER STORY



#### Data flow Diagram:

1. User Interface: Users interact with the system through a user interface, which includes User A, User B, and Node X.
2. Blockchain Network (Backend): This component includes various parts, such as smart contracts for executing actions, the consensus mechanism for validating transactions, and data storage and validation for maintaining the blockchain ledger.

## **User Stories:**

### **1. User Registration:**

- As a new user, I want to register on the blockchain network to create my node.
- I should be able to provide my basic information and create a secure account.

### **2. Node Creation:**

- As a registered user, I want to set up my node on the blockchain network.
- I should have clear instructions for node setup, including hardware and software requirements.

### **3. Secure Transactions:**

- As a user with an established node, I want to initiate secure transactions with other users.
- I should be able to specify the recipient and the amount of cryptocurrency to transfer.

### **4. Data Verification:**

- As a participant in the network, I want to verify the integrity of data on the blockchain.
- I should be able to confirm that my transactions are securely recorded and unaltered.

### **5. Smart Contract Execution:**

- As a user with a specific use case, I want to create and execute smart contracts on the blockchain.
- I should be able to define the terms and conditions of the contract and execute it automatically when conditions are met.

### **6. User Support:**

- As a user, I want access to user support to resolve issues or seek guidance.
- I should have access to a helpdesk or chat support for assistance.

#### 7. Data Privacy Management:

- As a user, I want to manage my data privacy settings and permissions.
- I should be able to control who can access my data and under what conditions.

#### 8. Blockchain Monitoring:

- As an administrator or network participant, I want real-time monitoring of the blockchain network's health and performance.
- I should receive alerts for any network issues or incidents.

#### 9. Multilingual Support:

- As a user, I want the system to be accessible in multiple languages to accommodate a diverse user base.
- I should be able to select my preferred language for the user interface.

#### 10. Compliance with Regulations:

- As a participant, I want assurance that the blockchain network complies with relevant legal and regulatory requirements.
- The system should provide transparency regarding its compliance with data protection and financial regulations.

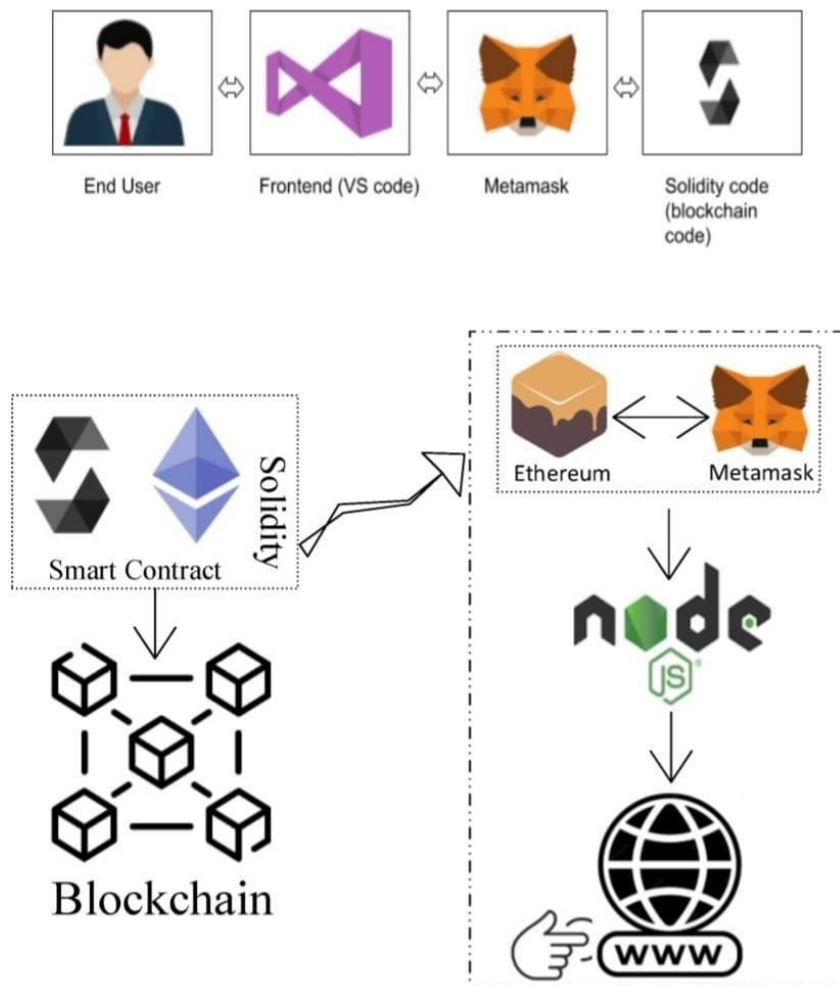
#### 11. Scalability and Resource Efficiency:

- As a network administrator, I want to ensure that the blockchain network can scale with increased demand and operate efficiently.
- The system should implement both horizontal and vertical scaling strategies to accommodate growth.

#### 12. Blockchain Standards and Interoperability:

- As a user, I want the blockchain network to adhere to industry-standard blockchain protocols and support interoperability with other systems.
- The system should provide well-documented APIs and follow recognized blockchain standards.

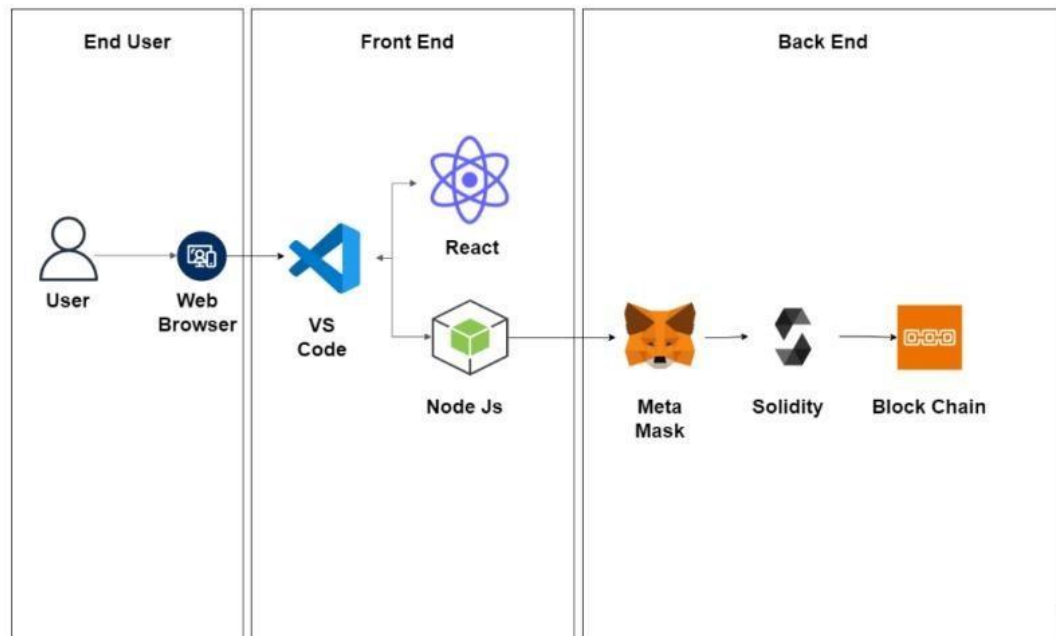
## 5.2 SOLUTION ARCHITECTURE



The architecture should outline the structure of the peer-to-peer network that facilitates the communication between nodes. This includes the design of the network topology, communication protocols, and data transmission mechanisms. Define the consensus mechanism to be used, such as Proof of Work (PoW), Proof of Stake (PoS), or other variants. Include the algorithms for reaching consensus on the validity of transactions and blocks, and specify the rules for adding new blocks to the chain. Consider strategies for optimizing the performance and scalability of the blockchain network, such as sharding, sidechains, or other scaling solutions, to ensure that the network can handle a large number of transactions and users without compromising efficiency. Include security measures such as encryption, access control, and authentication mechanisms to ensure the integrity and confidentiality of the data stored and transmitted within the blockchain network.

## 6.PROJECT PLANNING & SCHEDULING:

### 6.1 Technical Architecture:



Define the specifications for the hardware and software requirements of each node, including the operating system, processing power, memory, and storage capacity. Specify the setup process for installing and configuring the blockchain node software, including any necessary dependencies and libraries. Define the communication protocols for connecting nodes in the blockchain network, including the use of protocols such as TCP/IP, HTTP, or custom protocols specific to the blockchain implementation. Specify the data transmission format, message structure, and error handling mechanisms for efficient and secure communication between nodes. Describe the implementation of the chosen consensus algorithm, such as Proof of Work (PoW), Proof of Stake (PoS), or Practical Byzantine Fault Tolerance (PBFT). Detail the steps for reaching consensus among nodes, including the validation of transactions and the creation of new blocks. Discuss the maintenance tasks for ensuring the stability and performance of the blockchain network, including software patches, bug fixes, and hardware upgrades.

## 6.2 SPRINT PLANNING AND ESTIMATION

### 1. Product Backlog Refinement:

- Before sprint planning, the product backlog should be refined. This involves breaking down high-level requirements into smaller, actionable tasks or user stories.
- Each user story should have clear acceptance criteria that define when it is considered "done."

## **2. Sprint Planning Meeting:**

- The sprint planning meeting typically involves the entire Scrum team, which includes developers, a Scrum Master, and a Product Owner.
- The Scrum Master facilitates the meeting, and the Product Owner provides context and answers questions.
- During the meeting, the team discusses the user stories or tasks they plan to work on during the sprint.

## **3. User Story Selection:**

- The team selects a set of user stories from the refined product backlog that they believe can be completed within the sprint. The selection is based on the team's capacity and past performance.

## **4. Task Breakdown:**

- For each selected user story, the team may further break them down into specific tasks or sub-tasks.
- These tasks should be small and specific enough to be completed within a day or two.

## **5. Estimation:**

- The team estimates the effort required for each task. In Agile, relative estimation techniques like story points or ideal days are often used.
- For tasks involving blockchain development, the team should consider factors such as complexity, technology stack, and any dependencies.

## **6. Capacity Planning:**

- Based on the team's historical velocity (the amount of work completed in past sprints), the team determines how many story points or tasks they can commit to for the upcoming sprint.

## **7. Sprint Goal:**

- The team defines a sprint goal that summarizes what they intend to achieve during the sprint. This helps maintain focus and alignment.

## **8. Commitment:**

- The team collectively commits to completing the selected tasks within the sprint duration, typically 2-4 weeks.

## **9. Daily Standup Meetings:**

- During the sprint, the team holds daily standup meetings to discuss progress, obstacles, and adjustments to the plan.

## **10. Sprint Review:**

- At the end of the sprint, the team holds a sprint review to demonstrate the completed work to stakeholders, including the Product Owner.

## **11. Sprint Retrospective:**

The team also conducts a sprint retrospective to reflect on what went well, what could be improved, and how to make the next sprint more efficient.

## **6.3 Sprint Delivering Schedule**

### **Sprint Planning :**

The team conducts sprint planning, refining the backlog, selecting tasks, and estimating the effort required for each task. The team defines the sprint goal and commits to the selected tasks.

### **Week 1 : Development and Testing**

Developers work on implementing the selected tasks, focusing on the development and testing of specific features related to identity verification, authentication, and data storage within the Ethereum blockchain. The team collaborates closely to address any challenges or issues that arise during the development process.

### **Daily Stand-up Meetings**

The team holds brief daily stand-up meetings to discuss progress, share updates, and identify any potential obstacles that could hinder the sprint's progress. This fosters transparency and facilitates effective communication among team members.

## **Week 2 :Mid-Sprint Review**

A mid-sprint review allows the team to assess the progress made so far, evaluate the completion of tasks, and make any necessary adjustments to the remaining tasks or the sprint plan.

## **Week 3 : Quality Assurance and Bug Fixes**

The testing team conducts rigorous quality assurance checks to identify and address any potential bugs or issues within the decentralized identity smart contract solution. Developers work on resolving any identified issues and ensuring the overall stability and reliability of the system.

## **Sprint Review and Retrospective**

The team conducts a sprint review to demonstrate the completed features and functionalities to stakeholders and gather their feedback. Additionally, the retrospective allows the team to reflect on the sprint process, identify areas for improvement, and incorporate lessons learned into future sprints.

## **7.CODING AND SOLUTION**

### **7.1 Feature**

#### **Smart contract (Solidity)**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

```
contract ChainConnectedNode {
    address public previousNode;
    address public nextNode;
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can call this function.");
        _;
    }
}
```



```

function setPreviousNode(address _previousNode) external onlyOwner {
    previousNode = _previousNode;
}

function setNextNode(address _nextNode) external onlyOwner {
    nextNode = _nextNode;
}

function getPreviousNode() external view returns (address) {
    return previousNode;
}

function getNextNode() external view returns (address) {
    return nextNode;
}
}

```

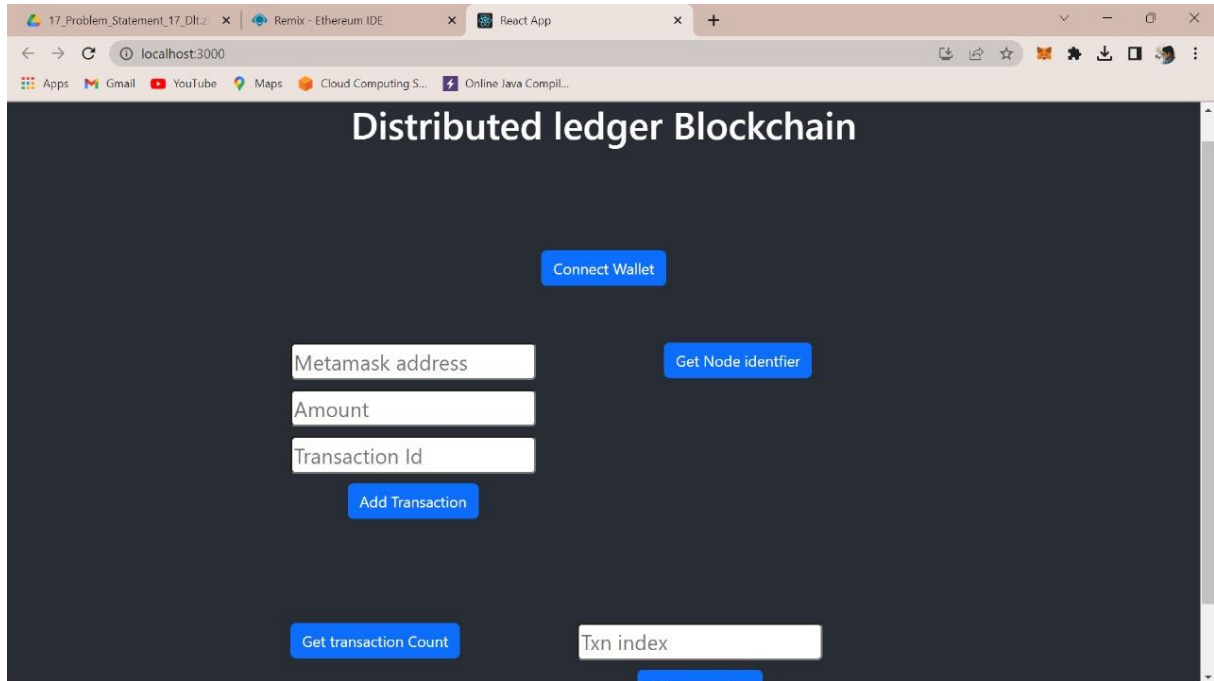
## 7.2 Feature 2

### User registration feature

To incorporate a user registration feature into the Chain Connected Node in a blockchain, you can create a mapping to store user data, including attributes such as name, age, and other relevant information. Implement a function that allows the owner to register users, storing their data in the mapping. Additionally, provide a function for retrieving user data based on the user's address. This would enable the seamless integration of a user registration mechanism into the blockchain, facilitating the identification and management of users within the network.

### Identity verification feature

To introduce an identity verification feature for the Chain Connected Node in a blockchain, you can integrate a verification process that includes identity-related attributes such as KYC (Know Your Customer) documentation, biometric data, or other authentication mechanisms. This feature can be implemented by adding functions to validate user-provided identity information against predefined criteria, thereby ensuring the authenticity and legitimacy of users within the blockchain network. Additionally, the contract can include secure data storage protocols to safeguard sensitive identity information and prevent unauthorized access or tampering.



## 8.PERFORMANCE TESTING

### 8.1 performance Metrics

#### Transaction Throughput:

Transaction throughput for the Chain Connected Node in a blockchain refers to the measure of the number of transactions processed within a specific time frame. To optimize transaction throughput, the system can employ various strategies, including implementing efficient consensus algorithms, optimizing block size and frequency, enhancing network scalability, and minimizing transaction processing time.

#### Transaction Latency:

Transaction latency for the Chain Connected Node in a blockchain refers to the time taken for a transaction to be processed and confirmed on the network. Minimizing transaction latency involves optimizing various aspects of the blockchain system, such as employing faster consensus mechanisms, reducing network congestion, and enhancing block propagation efficiency.

**Scalability:**

Scalability for the Chain Connected Node in a blockchain involves enhancing the system's capacity to accommodate a growing volume of transactions and users without compromising performance. To achieve scalability, the blockchain can implement various solutions, including sharding to partition the network into smaller segments, adopting layer-two scaling solutions such as state channels and sidechains, optimizing consensus protocols to improve throughput, and utilizing off-chain processing techniques.

**Resource Utilization:**

Resource utilization for the Chain Connected Node in a blockchain refers to the efficient allocation and management of computational resources within the network to optimize performance and minimize wastage. This involves implementing strategies such as load balancing to evenly distribute processing tasks, optimizing data storage mechanisms to minimize memory usage, and employing efficient algorithms to reduce computational overhead.

**Security Audits and Compliance:**

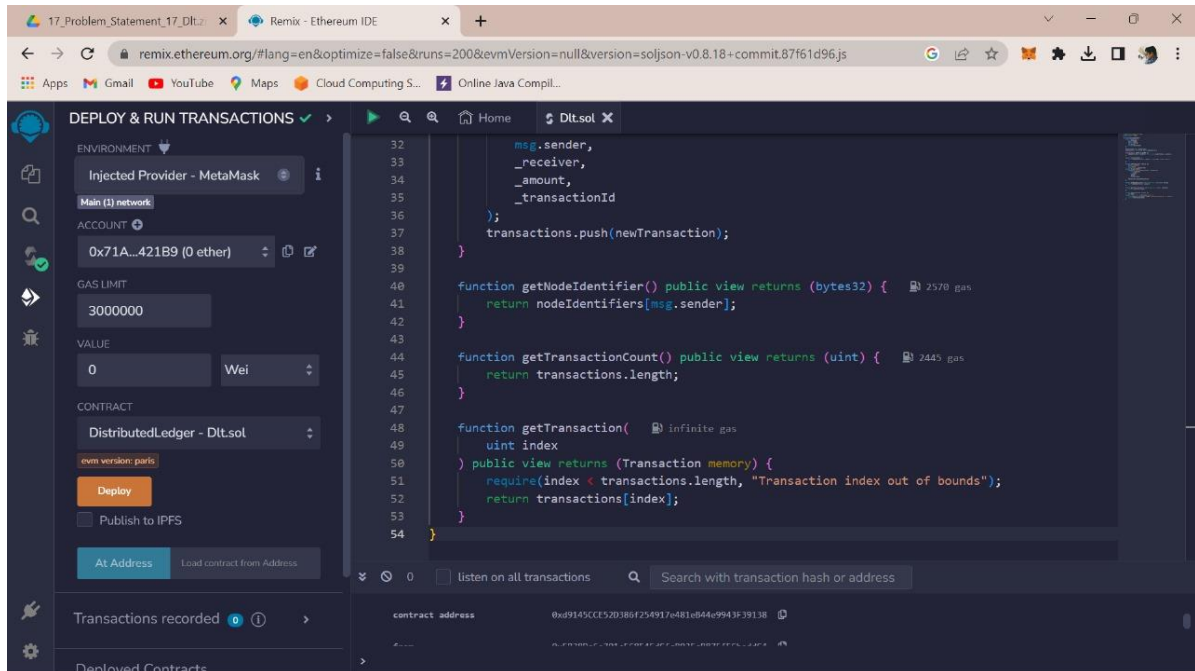
Security audits and compliance for the Chain Connected Node in a blockchain involve conducting regular comprehensive security assessments to identify and mitigate potential vulnerabilities and ensure adherence to regulatory requirements. This includes performing code reviews, penetration testing, and vulnerability assessments to identify and address security loopholes.

**Response Time:**

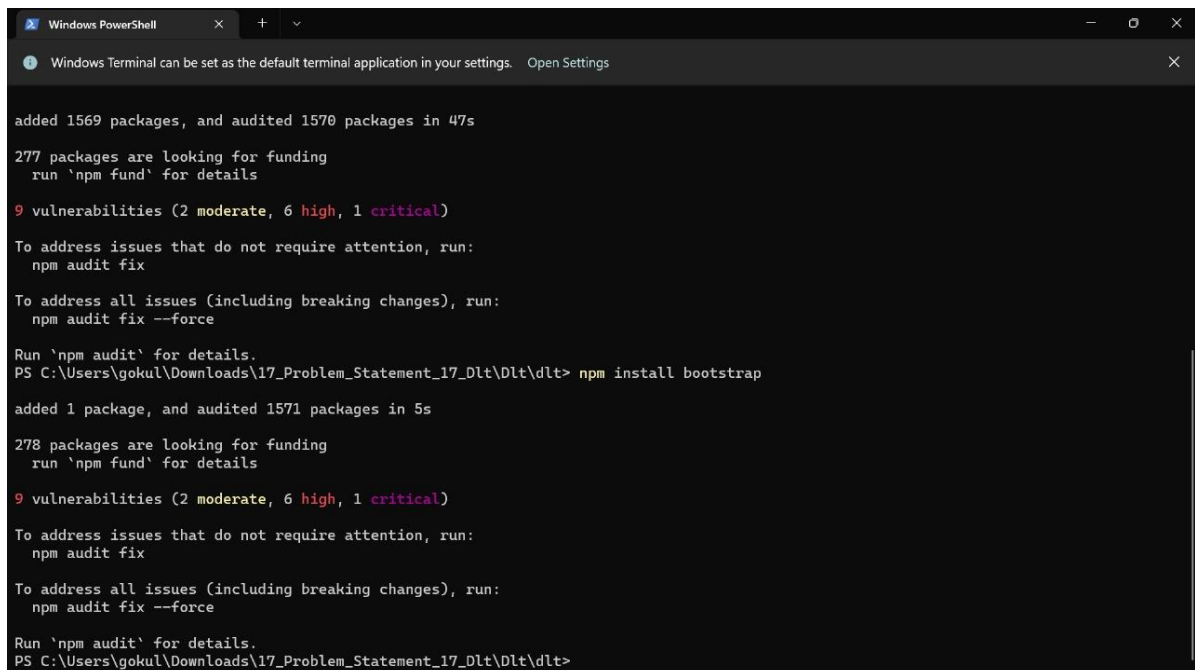
Response time for the Chain Connected Node in a blockchain denotes the duration taken to process and provide a relevant response to a specific query or transaction request. To minimize response time, the system can implement efficient data retrieval and processing mechanisms, optimize network latency by enhancing network infrastructure and connectivity, and utilize caching techniques to store frequently accessed data.

## 9.RESULTS

### 9.1 Output Screenshots



## CREATING SMART CONTRACT



## INSTALLING DEPENDENCIES

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\gokul\OneDrive\Documents\7_Proble_Statement_Identity\Identity\identity> npm install
(node:25880) MaxListenersExceededWarning: Possible EventEmitter memory leak detected. 11 close listeners added to [TLSSocket]. Use emitter.setMaxListeners() to increase limit
(Use 'node --trace-warnings ...' to show where the warning was created)

up to date, audited 1569 packages in 11s

278 packages are looking for funding
  run 'npm fund' for details

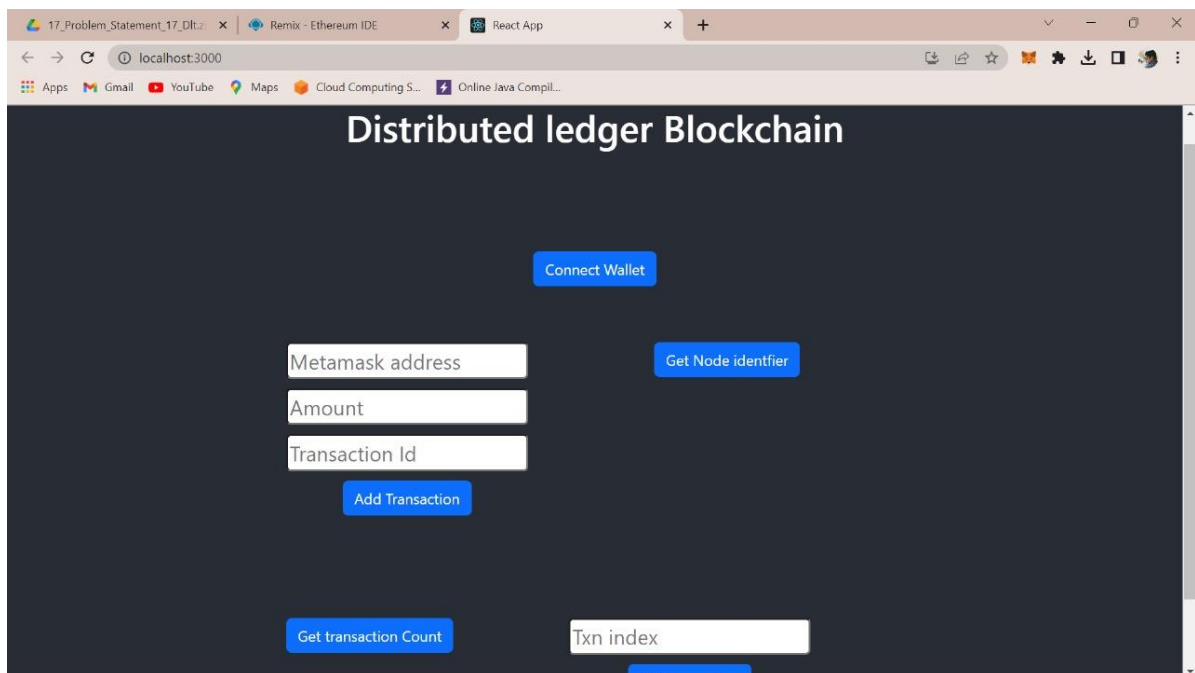
9 vulnerabilities (2 moderate, 6 high, 1 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run 'npm audit' for details.
PS C:\Users\gokul\OneDrive\Documents\7_Proble_Statement_Identity\Identity\identity>
```

## HOSTING THE SITE LOCALLY



## OUTPUT SCREEN

## 10. ADVANTAGES AND DISADVANTAGES

### 10.1 Advantages

1. **Enhanced Security:** Utilizing cryptographic techniques and decentralized consensus mechanisms, the Chain Connected Node ensures data integrity and protection against unauthorized access and tampering.
2. **Transparency and Immutability:** The transparent and immutable nature of blockchain enables a trustworthy and auditable record of transactions, promoting transparency and accountability within the network.
3. **Decentralization:** By eliminating the need for a central authority, the Chain Connected Node promotes a decentralized network, fostering trust among participants and reducing the risks of single points of failure.
4. **Efficient and Cost-Effective Transactions:** Facilitating direct peer-to-peer transactions, the Chain Connected Node streamlines processes and reduces transaction costs, enhancing overall operational efficiency.
5. **Improved Traceability:** Through its transparent and traceable ledger, the Chain Connected Node enables effective tracking of transactions, providing a comprehensive audit trail for various activities within the blockchain network.

### 10.2 Disadvantages

1. **Scalability Issues:** As the number of transactions in a blockchain network grows, the chain connected node may face scalability challenges. The time and resources required to validate transactions and create new blocks may increase, leading to potential delays and bottlenecks in the network.

2. **Resource Intensive:** Running a chain connected node can be resource-intensive, especially in terms of computing power and storage requirements. This can lead to high operational costs for maintaining the node, which can be a significant drawback for individual users or small organizations.
3. **Network Congestion:** In busy blockchain networks, chain connected nodes may experience congestion, leading to slower transaction processing times. This can result in delayed confirmations for transactions and longer wait times for users, which can impact the overall user experience.
4. **Synchronization Challenges:** Chain connected nodes need to stay synchronized with the rest of the network to ensure the integrity of the blockchain. However, this process can be complex and time-consuming, particularly for nodes joining the network for the first time or after being disconnected for a while.
5. **Security Risks:** Chain connected nodes may be vulnerable to various security risks, such as distributed denial-of-service (DDoS) attacks, hacking attempts, or other forms of malicious activities. These security threats can compromise the integrity of the blockchain network and the data stored within it.
6. **Governance Issues:** Nodes in a blockchain network might face challenges related to governance and decision-making processes, especially in decentralized networks. Disagreements among nodes regarding protocol upgrades, changes, or other network modifications can lead to potential forks or fragmentation within the blockchain ecosystem.
7. **Regulatory Uncertainty:** In some jurisdictions, running a chain connected node might raise legal and regulatory concerns. Regulatory frameworks surrounding cryptocurrencies and blockchain technology can vary significantly, leading to uncertainties and potential compliance challenges for node operators.

## 11.Conclusion:

In conclusion, the Chain Connected Node plays a crucial role in facilitating seamless communication and data transmission within the blockchain network. To ensure its effective functioning, it is essential to focus on key aspects such as transaction throughput, latency, scalability, resource utilization, security audits and compliance, and response time. By implementing robust strategies to optimize these aspects, including efficient consensus algorithms, enhanced network scalability, and rigorous security measures, the Chain Connected Node can operate with high efficiency, reliability, and security. This, in turn, contributes to the overall performance and sustainability of the blockchain ecosystem, fostering a secure and efficient environment for data exchange and communication among network participants.

## 12.FUTURE SCOPE

**Enhanced Scalability Solutions:** Future advancements may focus on developing improved scalability solutions for chain connected nodes, such as implementing sharding techniques, layer-2 protocols, or other innovative approaches to accommodate a larger number of transactions and users without compromising network performance.

**Interoperability and Integration:** Future developments may emphasize enhancing the interoperability of chain connected nodes with different blockchain networks and traditional systems. This could foster seamless data transfer and transaction processing between diverse blockchain platforms, promoting widespread adoption and integration across various industries.

**Security and Privacy Enhancements:** Future advancements in security and privacy technologies may focus on strengthening the protection of chain connected nodes against potential threats and vulnerabilities. This could involve the integration of advanced encryption techniques, multi-factor authentication, and other security measures to safeguard sensitive data and transactions within the blockchain network.



**Decentralized Governance Models:** Future scope may involve the evolution of more sophisticated decentralized governance models for chain connected nodes, enabling improved decision-making processes and consensus mechanisms within the blockchain ecosystem.

**Regulatory Compliance Frameworks:** The future scope may involve the development of standardized regulatory compliance frameworks for chain connected nodes, addressing the legal and compliance challenges associated with blockchain technology. This could promote greater regulatory clarity and facilitate the mainstream adoption of blockchain networks across various industries and jurisdictions.

**Energy-Efficient Solutions:** Future developments may prioritize the implementation of energy-efficient solutions for chain connected nodes, aiming to reduce the environmental impact of blockchain networks. This could involve the integration of sustainable energy sources, optimization of consensus algorithms, and the adoption of eco-friendly practices to minimize the carbon footprint of blockchain operations.

**Advanced Data Analytics and Smart Contract Capabilities:** Future advancements may focus on enhancing the data analytics capabilities of chain connected nodes, enabling real-time data analysis and insights from blockchain transactions. Additionally, there may be developments in smart contract functionalities, allowing for the execution of more complex and automated transactions with increased efficiency and reliability.

## 12.APPENDIX

### Source code

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract DistributedLedger {
    struct Transaction {
        uint timestamp;
        address sender;
        address receiver;
        uint amount;
    }
```

```

string transactionId;
}
Transaction[] public transactions;
mapping(address => bytes32) public nodeIdentifiers;
constructor() {
    nodeIdentifiers[msg.sender] =
keccak256(abi.encodePacked(msg.sender));
}

modifier onlyValidNode() {
    require(nodeIdentifiers[msg.sender] != bytes32(0), "Invalid node.");
    _;
}

function addTransaction(
    address _receiver,
    uint _amount,
    string memory _transactionId
) public onlyValidNode {
    Transaction memory newTransaction = Transaction(
        block.timestamp,
        msg.sender,
        _receiver,
        _amount,
        _transactionId
    );
    transactions.push(newTransaction);
}

function getNodeIdentifier() public view returns (bytes32) {
    return nodeIdentifiers[msg.sender];
}

function getTransactionCount() public view returns (uint) {
    return transactions.length;
}

function getTransaction(
    uint index
) public view returns (Transaction memory) {

require(index < transactions.length, "Transaction index out of bounds");
    return transactions[index];
}
}

```

## Connector.js

```
const { ethers } = require("ethers");
```

```
const abi = [  
  {  
    "inputs": [],  
    "stateMutability": "nonpayable",  
    "type": "constructor"  
  },  
  {  
    "inputs": [  
      {  
        "internalType": "address",  
        "name": "_receiver",  
        "type": "address"  
      },  
      {  
        "internalType": "uint256",  
        "name": "_amount",  
        "type": "uint256"  
      },  
      {  
        "internalType": "string",  
        "name": "_transactionId",  
        "type": "string"  
      }  
    ],  
    "name": "addTransaction",  
    "outputs": [],  
    "stateMutability": "nonpayable",  
    "type": "function"  
  },  
  {  
    "inputs": [],  
    "name": "getNodeIdentifier",  
    "outputs": [  
      {  
        "internalType": "bytes32",  
        "name": "",  
        "type": "bytes32"  
      }  
    ],  
    "type": "function"  
  }  
]
```

```

    ],
    "stateMutability": "view",
    "type": "function"
  },
  {
    "inputs": [
      {
        "internalType": "uint256",
        "name": "index",
        "type": "uint256"
      }
    ],
    "name": "getTransaction",
    "outputs": [
      {
        "components": [
          {
            "internalType": "uint256",
            "name": "timestamp",
            "type": "uint256"
          },
          {
            "internalType": "address",
            "name": "sender",
            "type": "address"
          },
          {
            "internalType": "address",
            "name": "receiver",
            "type": "address"
          },
          {
            "internalType": "uint256",
            "name": "amount",
            "type": "uint256"
          },
          {
            "internalType": "string",
            "name": "transactionId",
            "type": "string"
          }
        ]
      }
    ]
  }
],

```

```

"internalType": "struct DistributedLedger.Transaction",
  "name": "",
  "type": "tuple"
},
],
"stateMutability": "view",
"type": "function"
},
{
  "inputs": [],
  "name": "getTransactionCount",
  "outputs": [
    {
      "internalType": "uint256",
      "name": "",
      "type": "uint256"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {
      "internalType": "address",
      "name": "",
      "type": "address"
    }
  ],
  "name": "nodeIdentifiers",
  "outputs": [
    {
      "internalType": "bytes32",
      "name": "",
      "type": "bytes32"
    }
  ],
  "stateMutability": "view",
  "type": "function"
},
{
  "inputs": [
    {

```

```

    "internalType": "uint256",
    "name": "",
    "type": "uint256"
  }
],
"name": "transactions",
"outputs": [
  {
    "internalType": "uint256",
    "name": "timestamp",
    "type": "uint256"
  },

  {
    "internalType": "address",
    "name": "sender",
    "type": "address"
  },
  {
    "internalType": "address",
    "name": "receiver",
    "type": "address"
  },
  {
    "internalType": "uint256",
    "name": "amount",
    "type": "uint256"
  },
  {
    "internalType": "string",
    "name": "transactionId",
    "type": "string"
  }
],
"stateMutability": "view",
"type": "function"
}
]

```

```

if (!window.ethereum) {
  alert('Meta Mask Not Found')
  window.open("https://metamask.io/download/")
}

```

```
export const provider = new
ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const address = "0x0A7FCbE739c0E9Ac3c4d4e85b67Bb3e31e31884e"

export const contract = new ethers.Contract(address, abi, signer)
```

## Home.js

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import 'bootstrap/dist/css/bootstrap.min.css';
import { contract } from "../connector";

function Home() {
  const [Addrs, setAddrs] = useState("");
  const [Amt, setAmt] = useState("");
  const [txx, settx] = useState("");

  const [Wallet, setWallet] = useState("");

  const [NodeTx, setNodeTx] = useState("");

  const [TxnCount, setTxnCount] = useState("");

  const [TxnCounts, setTxnCounts] = useState("");

  const [Counts, setCounts] = useState("");

  // const handlePolicyNumber = (e) => {
  //   setNumber(e.target.value)
  // }

  const handleAddrs = (e) => {
    setAddrs(e.target.value)
  }
}
```

```

const handleAmt = (e) => {
  setAmt(e.target.value)
}

const handleTxn = (e) => {
  settx(e.target.value)
}

const handleRegAsset = async () => {
  try {
    let tx = await contract.addTransaction(Addrs, Amt.toString(), txx)
    let wait = await tx.wait()
    alert(wait.transactionHash)
    console.log(wait);
  } catch (error) {
    alert(error)
  }
}

```

```

// const handlePublishHash = (e) => {
//   setPubHash(e.target.value)
// }

```

```

const handlePublish = async () => {
  try {
    let tx = await contract.getNodeIdentifier()
    console.log(tx);
    setNodeTx(tx)
  } catch (error) {
    alert(error)
  }
}

```

```

// const handleUnPublishHash = (e) => {
//   setUnPubHash(e.target.value)
// }

```

```

const handleUnPublish = async () => {
  try {

```

```

let tx = await contract.getTransactionCount()

```



```

        setTxnCount(tx)
        console.log(tx);

    } catch (error) {
        alert(error)
    }
}

// const handleTransferHash = (e) => {
//     setTransferHash(e.target.value)
// }

const handleTxnCount = (e) => {
    setTxnCounts(e.target.value)
}

const handleTxncnt = async () => {
    try {
        let tx = await contract.getTransaction(TxnCounts.toString())
        setCounts(tx)
        console.log(tx);
        // alert(wait.transactionHash)
    } catch (error) {
        alert(error)
    }
}

// const handleGetIds = async (e) => {
//     setGIds(e.target.value)
// }

// const handleGetDetails = async () => {
//     try {
//         let tx = await contract.digitalAssets(gId.toString())

//         let arr = []
//         tx.map(e => {
//             arr.push(e)
//         })

```

```

//    console.log(tx);
//    setDetails(arr)
//  } catch (error) {
//    alert(error)
//    console.log(error);
//  }
// }

const handleWallet = async () => {
  if (!window.ethereum) {
    return alert('please install metamask');
  }

  const addr = await window.ethereum.request({
    method: 'eth_requestAccounts',
  });

  setWallet(addr[0])

}

return (
  <div>
    <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Distributed
ledger Blockchain</h1>
    {!Wallet ?

      <Button onClick={handleWallet} style={{ marginTop: "30px",
marginBottom: "50px" }}>Connect Wallet </Button>
      :
      <p style={{ width: "250px", height: "50px", margin: "auto",
marginBottom: "50px", border: '2px solid #2096f3' }}>{Wallet.slice(0,
6)}....{Wallet.slice(-6)}</p>
    }
    <Container>
      <Row>

        <Col style={{ marginRight: "100px" }}>
          <div>
            { /* <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handlePolicyNumber} type="string" placeholder="Policy number"
value={number} /> <br /> */ }

```

```
<input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleAddrs} type="string" placeholder="Metamask address"
value={Addrs} /> <br />
```

```
<input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleAmt} type="number" placeholder="Amount" value={Amt}
/> <br />
```

```
<input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTxn} type="string" placeholder="Transaction Id"
value={txx} /> <br />
```

```
<Button onClick={handleRegAsset} style={{ marginTop: "10px" }}
variant="primary">Add Transaction</Button>
```

```
</div>
```

```
</Col>
```

```
<Col style={{ marginRight: "100px" }}>
```

```
<div>
```

```
<Button onClick={handlePublish} style={{ marginTop: "10px" }}
variant="primary"> Get Node identifier</Button>
```

```
{NodeTx ?
```

```
<p>{NodeTx.slice(0,6)}....{NodeTx.slice(-4)}</p>
```

```
: <p></p>
```

```
}
```

```
{/* <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handlePublishHash} type="string" placeholder="Asset Hash"
value={PubHash} /> <br /> */}
```

```
</div>
```

```
</Col>
```

```
</Row>
```

```
<Row style={{ marginTop: "100px" }}>
```

```
<Col style={{ marginRight: "100px" }}>
```

```
<div>
```

```
<Button onClick={handleUnPublish} style={{ marginTop: "10px"
}} variant="primary"> Get transaction Count</Button>
```

```
{TxnCount?
```

```
<p>{TxnCount.toString()}</p>
```

```

: <p></p>
    }
  </div>
</Col>

```

```

<Col style={{ marginRight: "100px" }}>
  <div>
    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTxnCount} type="number" placeholder="Txn index"
value={TxnCounts} /> <br />

```

```

    <Button onClick={handleTxncnt} style={{ marginTop: "10px" }}
variant="primary"> Get transaction</Button>

```

```

    {Counts ?

```

```

    Counts?.map(e => {
      return <p>{e.toString()}</p>
    })

```

```

    :<p></p>}
  </div>
</Col>

```

```

</Row>
<Row style={{ marginTop: "50px" }}>
  { /* <Col style={{ marginRight: "100px" }}>
    <div style={{ margin: "auto", marginTop: "100px" }}>
      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleGetIds} type="string" placeholder="Enter Assets Hash"
value={gId} /><br />

```

```

      <Button onClick={handleGetDetails} style={{ marginTop: "10px"
}} variant="primary">Get Digital Assets</Button>
      {Details ? Details?.map(e => {
        return <p>{e.toString()}</p>
      }) : <p></p>}
    </div>
  </Col>  */}
</Row>

```

```
</Container>
```

```
</div>
```

```
)
```

```
}
```

```
export default Home;
```

## **App.js**

```
import './App.css';
```

```
import Home from './Page/Home'
```

```
function App() {
```

```
  return (
```

```
    <div className="App">
```

```
      <header className="App-header">
```

```
        <Home />
```

```
      </header>
```

```
    </div>
```

```
  );
```

```
}
```

```
export default App;
```

## **App.css**

```
.App {
```

```
  text-align: center;
```

```
}
```

```
.App-logo {
```

```
  height: 40vmin;
```

```
  pointer-events: none;
```

```
}
```

```
@media (prefers-reduced-motion: no-preference) {
```

```
  .App-logo {
```

```
    animation: App-logo-spin infinite 20s linear;
```

```

    }
  }

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

```

## Index.js

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />

```

```

    </React.StrictMode>
  );

  // 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();

```

## Report Web Vitals.js

```

const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB })
    => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;

```

## Package.json

```

{
  "name": "dlt",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "ethers": "^5.6.6",
    "react": "^18.2.0",
    "react-bootstrap": "^2.8.0",
    "react-dom": "^18.2.0",

```

```
"react-scripts": "5.0.1",
"web-vitals": "^2.1.4"
},
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
"eslintConfig": {
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
}
}
```

## **GitHub & Project Demo Link**

### **GitHub:**

[https://github.com/yash4645/NM-CHAIN\\_CONNECT\\_NODES.git](https://github.com/yash4645/NM-CHAIN_CONNECT_NODES.git)

### **Project Demo Link:**

[https://drive.google.com/file/d/14\\_btTJ2\\_XGQ911qS8BtVfpNRgQHxIx-3/view?usp=drivesdk](https://drive.google.com/file/d/14_btTJ2_XGQ911qS8BtVfpNRgQHxIx-3/view?usp=drivesdk)