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: ETHEREUM DECENTRALIZED IDENTITY SMART CONTRACT

TEAM MEMBERS

REGISTER NUMBER	NAME
731620104001	ABISHEK ANAND
731620104003	ARJUN K
731620104046	SANJAY M
731620104048	SANJITH V

TABLE OF CONTENT

1.INTRODUCTION

1.1 PROJECT OVERVIEW

The Ethereum Decentralized Identity Smart Contract project is designed to revolutionize digital identity management by leveraging the capabilities of Ethereum's blockchain technology. Through the implementation of smart contracts, the project aims to establish a secure and transparent framework for identity creation, authentication, and authorization processes. With a focus on ensuring user data privacy and control, the project seeks to provide users with immutable and tamper-proof identity records, empowering them to selectively share specific information with authorized parties. By fostering interoperability between different decentralized identity systems, the project endeavors to create a seamless and integrated digital identity ecosystem. Additionally, the project will prioritize the development of robust security measures and risk mitigation strategies to ensure the integrity and confidentiality of user data. The successful implementation of this project is expected to significantly enhance the trustworthiness and security of digital identity management within the Ethereum network and beyond.

1.2 PURPOSE

The purpose of the Ethereum Decentralized Identity Smart Contract lies in revolutionizing the management of digital identities within the Ethereum blockchain ecosystem. By utilizing smart contracts, the project aims to establish a secure and transparent platform for users to create, control, and safeguard their digital identities. This initiative seeks to prioritize user privacy and data security, allowing individuals to authenticate and authorize access to their identity information while retaining full ownership and authority over their personal data. Moreover, the project aspires to facilitate seamless communication and data exchange between different decentralized identity systems, fostering a cohesive and interconnected digital identity landscape. By implementing stringent security protocols and risk management strategies, the project aims to cultivate a trustworthy and dependable environment for managing digital identities, ultimately contributing to the advancement of secure and decentralized online interactions within the Ethereum network and beyond .

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.

2.1 Existing Problem :

Ethereum's decentralized identity smart contract projects have encountered a series of challenges, with scalability concerns being a prominent issue. The platform's limitations in handling large volumes of data efficiently, resulting in delays and high transaction costs, have hindered the smooth operation of these identity solutions. Balancing the need for privacy with the transparency and security requirements has also proved to be complex, with maintaining user privacy while ensuring the system's integrity being a delicate task. Furthermore, user adoption has been hindered by complicated interfaces and suboptimal user experiences, impeding the widespread acceptance of these solutions. Ensuring interoperability with other blockchain networks and platforms has been a struggle, as has fortifying the security of smart contracts against potential vulnerabilities and breaches. Additionally, adhering to diverse regulatory frameworks across jurisdictions has posed a significant challenge, requiring careful navigation of legal complexities and data protection requirements to establish long-term viability for Ethereum's decentralized identity projects.

2.2REFERENCES

[1]. Lee, Sherman Embracing Blockchain Could Completely Change The Way Artists Sell Music And Interact With Fans. Forbes (25 Apr 2018).

[2]. Dredge, Stuart What Could Blockchain Do for Music? Medium. (24 Jan 2018).

[3]. Kavinsky, Marc. How Blockchain Might Shake Up The IoT. IoT Business News (22 December 2020).

[4].6 Reasons Why Employers Need to Join the Blockchain Revolution and Consider Smart Contracts. JD SUPRA(14 July 2021).

2.2 Problem Statement Definition :

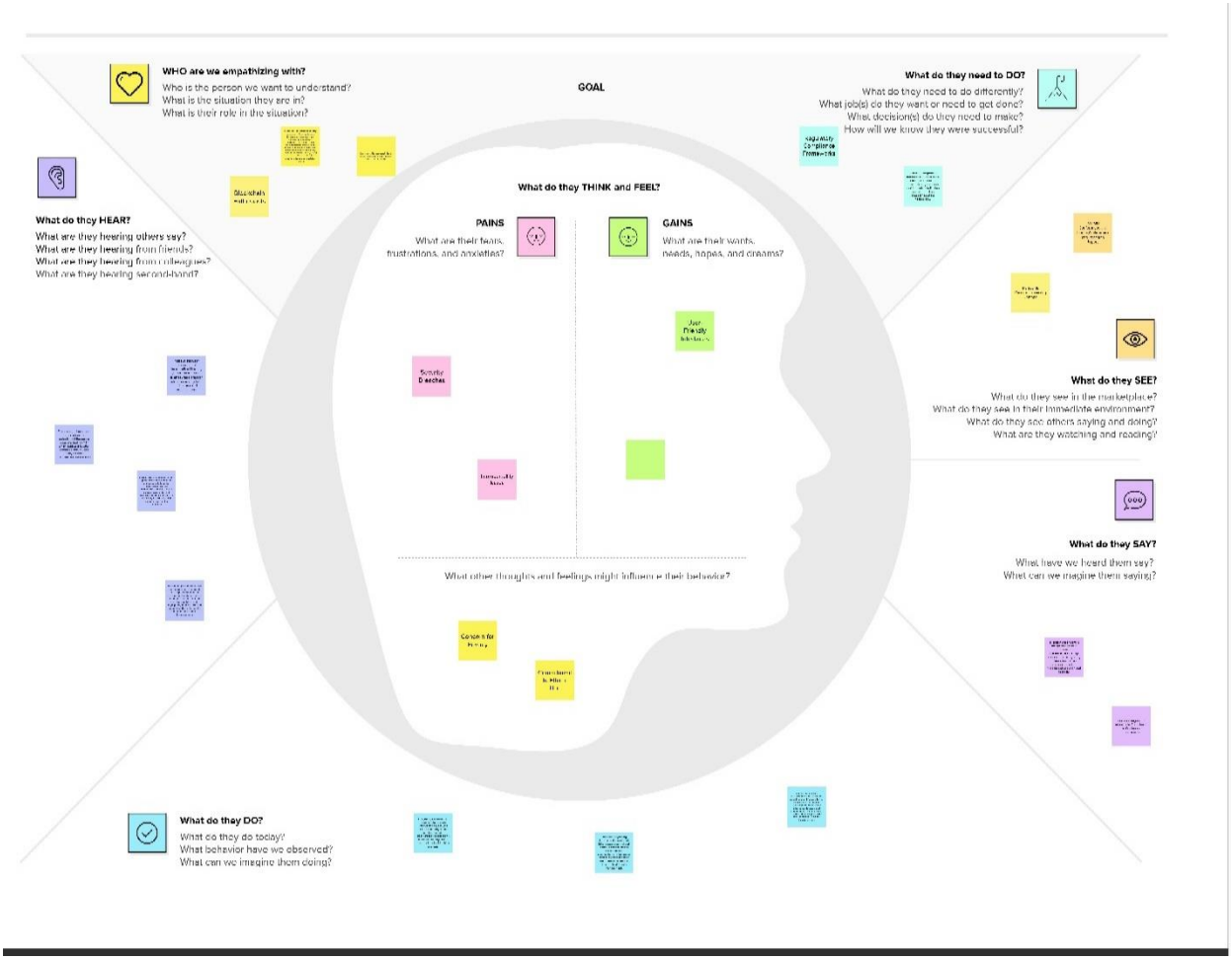
The existing traditional methods of identity management often lack transparency, leaving room for potential security breaches and unauthorized access. Additionally, the centralized nature of these systems poses the risk of single points of failure, making them vulnerable to data manipulation and fraud. To address these challenges, there is a need for a more secure, transparent, and decentralized solution. Therefore, the problem at hand is to design a smart contract using the Ethereum blockchain that can securely store identity details while ensuring data transparency and integrity. The smart contract should also enable authorized parties to query the stored identity information from the blockchain. This solution aims to leverage the decentralized and tamper-proof nature of blockchain technology, providing end-to-end verification advantages and ensuring that identity information remains immutable and transparent. By implementing this smart contract, the goal is to establish a robust and secure identity management system that mitigates the risks associated with traditional centralized approaches, offering enhanced data protection, nonrepudiation, and distributed security characteristics.

3.IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas

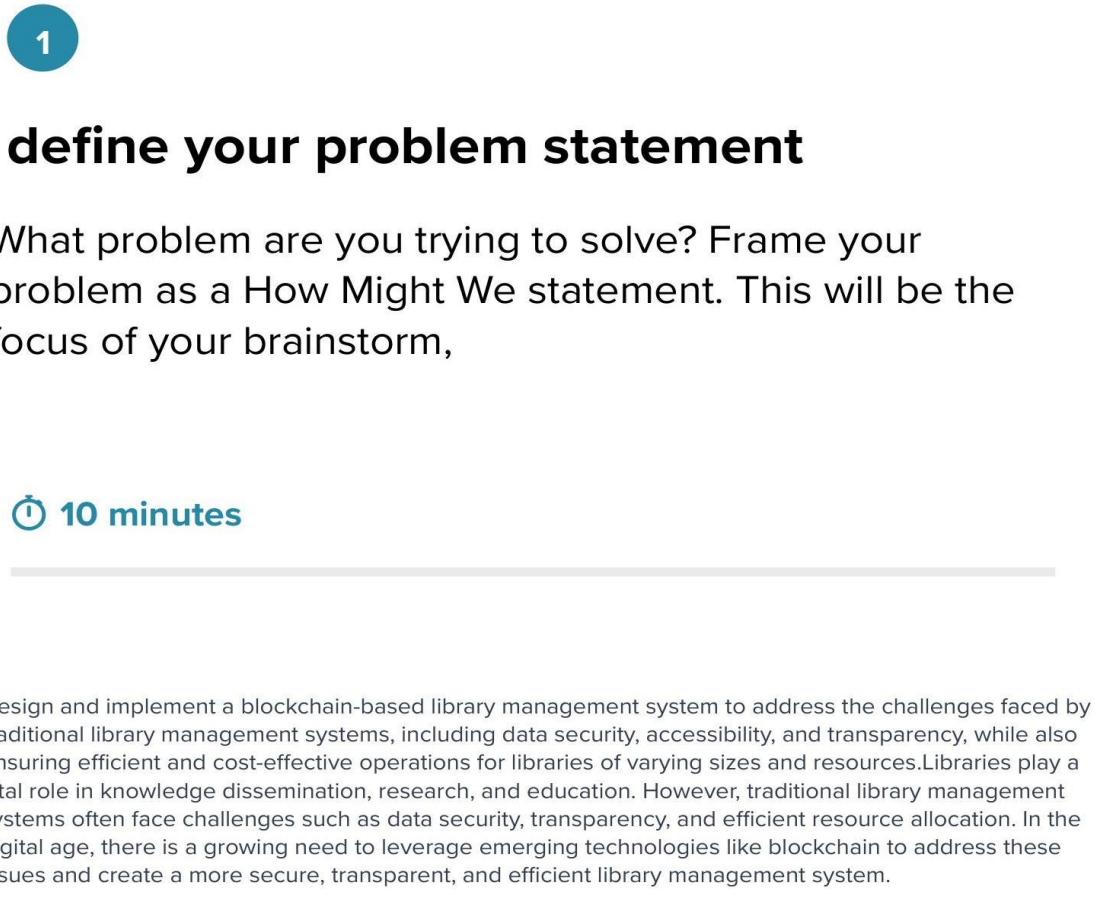
Creating an empathy map for an Ethereum decentralized identity smart contract is crucial in understanding the diverse needs and concerns of the involved stakeholders. In this context, the key stakeholders include the users, developers, and

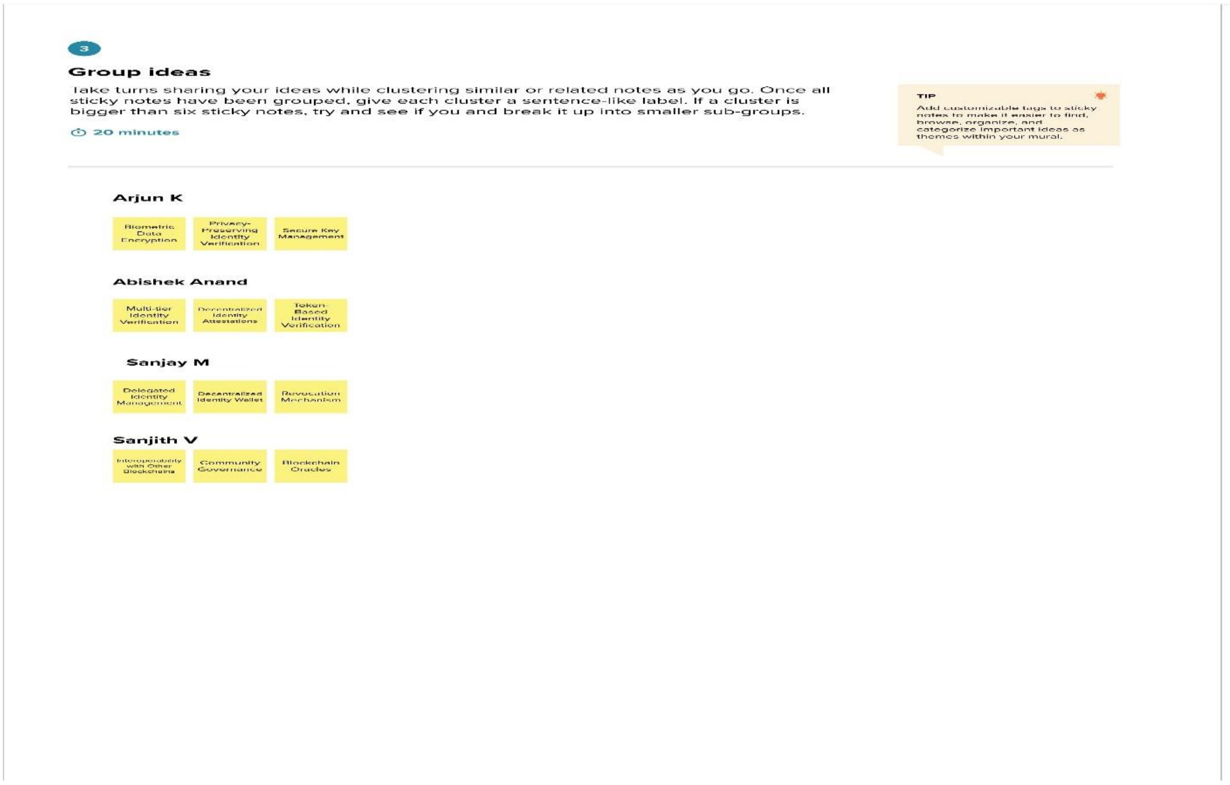
validators, each with specific requirements and considerations. Users seek secure and easy-to-use interfaces that offer them control over their personal data. Developers prioritize the robustness and scalability of the smart contract, ensuring its compatibility with other applications. Validators focus on the authentication and verification of user data, adhering to relevant compliance standards. Identifying pain points such as security vulnerabilities and usability challenges, alongside potential gains like enhanced data security and improved user experience, can inform the development process for a more effective and user-friendly decentralized identity system on the Ethereum blockchain.

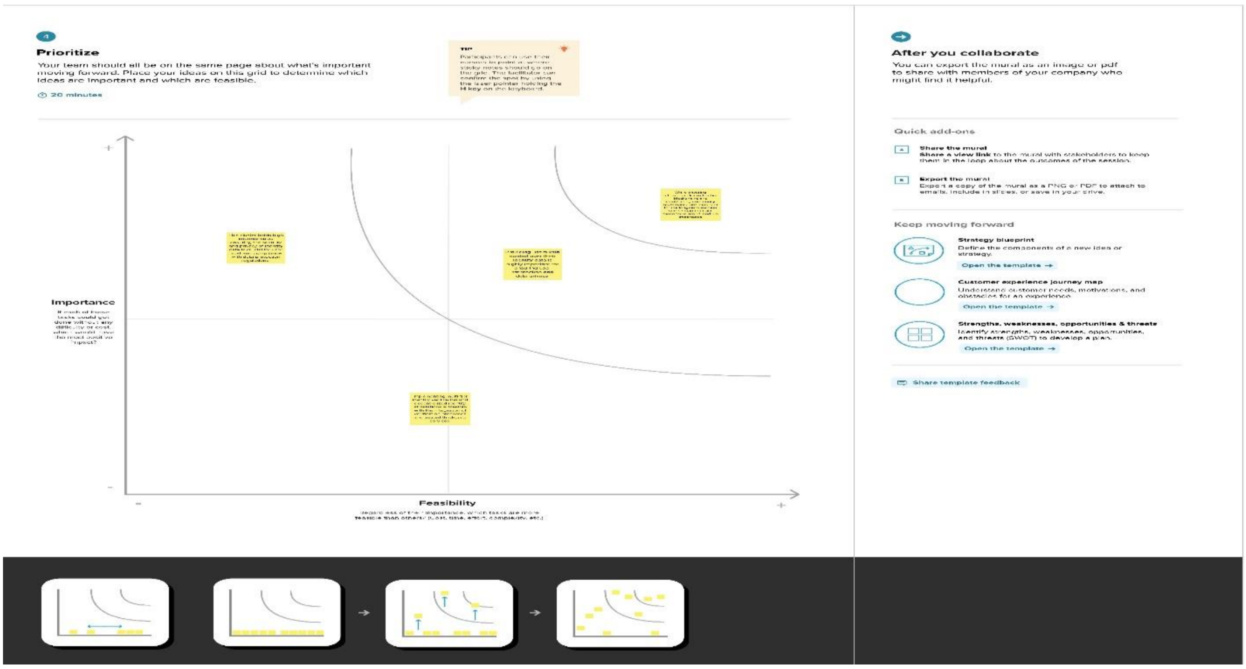


3.2 Ideation & Brainstorming :

When brainstorming for an Ethereum decentralized identity 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 Ethereum's 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.




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.
Arjun K
Biometric Data Encryption
Privacy-Preserving Identity Verification
Secure Key Management
Abishek Anand
Multi-tier Identity Verification
Decentralized Identity Attestations
Token Based Identity Verification
Sanjay M
Delegated Identity Management
Decentralized Identity Wallet
Revocation Mechanism
Sanjith V
Interoperability with Other Blockchains
Community Governance
Blockchain Oracles


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
Importance
Feasibility
After you collaborate
You can export the mural as an image or pdf to share with members of your company who might find it helpful.
Quick add-ons
Share the mural
Export the mural
Keep moving forward
Strategy blueprint
Customer experience journey map
Strengths, weaknesses, opportunities & threats
Open the template
Share template feedback

4.REQUIREMENT ANALYSIS:

4.1 Functional Requirements

i.User Authentication

The smart contract should facilitate a secure and reliable method for users to authenticate their identities on the Ethereum blockchain. This might involve the use of cryptographic protocols or other secure methods to verify the identity of users.

Identity Verification and Management

The smart contract should enable the verification and management of user identities in a decentralized manner. This could involve mechanisms for identity attestation, verification of credentials, and the management of identity-related data.

ii.Privacy Controls

The smart contract should incorporate features to ensure that users have control over their personal data and the level of privacy they wish to maintain. This might involve implementing privacy-preserving techniques such as zero-knowledge proofs or other cryptographic methods to protect sensitive information.

iii.Data Integrity and Security:

The smart contract should ensure the integrity and security of user data stored on the blockchain. This might involve implementing encryption, access control mechanisms, and other security measures to protect against unauthorized access and data tampering.

iv.Interoperability with Other Systems

The smart contract should be designed to facilitate interoperability with other decentralized identity systems, as well as with traditional identity management systems. This could involve the use of standardized protocols and formats to enable seamless data exchange and interoperability between different platforms.

V.Revocation and Recovery Mechanisms

The smart contract should incorporate mechanisms for the revocation and recovery of identities in case of loss or theft. This might include implementing procedures for identity recovery, as well as mechanisms for revoking compromised or lost identity credentials.

Vii. User Experience and Interface

The smart contract should provide a user-friendly interface that makes it easy for users to manage their identities and interact with the decentralized identity system. This might involve designing intuitive user interfaces and providing clear instructions for users to follow.

Viii.Compliance with Regulatory Standards

The smart contract should comply with relevant regulatory standards and data protection laws to ensure that the decentralized identity system adheres to legal requirements. This might involve implementing data protection measures and ensuring compliance with regulations related to identity management and user data protection.

4.2 Non-Functionl Requirements

Security:

Ensuring the security of the smart contract and the associated decentralized identity system is paramount. This includes safeguarding against potential cyber attacks, data breaches, and unauthorized access to sensitive user information.

Performance:

The smart contract should be designed to operate efficiently and reliably, with minimal latency and processing delays. This includes optimizing the execution speed of transactions and minimizing the time taken for identity verification and authentication processes.

Scalability:

The decentralized identity system should be scalable to accommodate a growing number of users and increasing volumes of identity-related data. This involves designing the smart contract architecture to handle a large number of concurrent transactions and users without compromising performance or security.

Reliability and Availability:

Ensuring the reliability and availability of the decentralized identity system is crucial. This includes minimizing downtime and service interruptions, as well as implementing robust backup and recovery mechanisms to prevent data loss in the event of system failures or disruptions.

Interoperability:

The smart contract should be interoperable with other blockchain networks and decentralized applications, enabling seamless integration with external systems and platforms. This includes adhering to industry standards and protocols to facilitate data exchange and communication between different blockchain ecosystems.

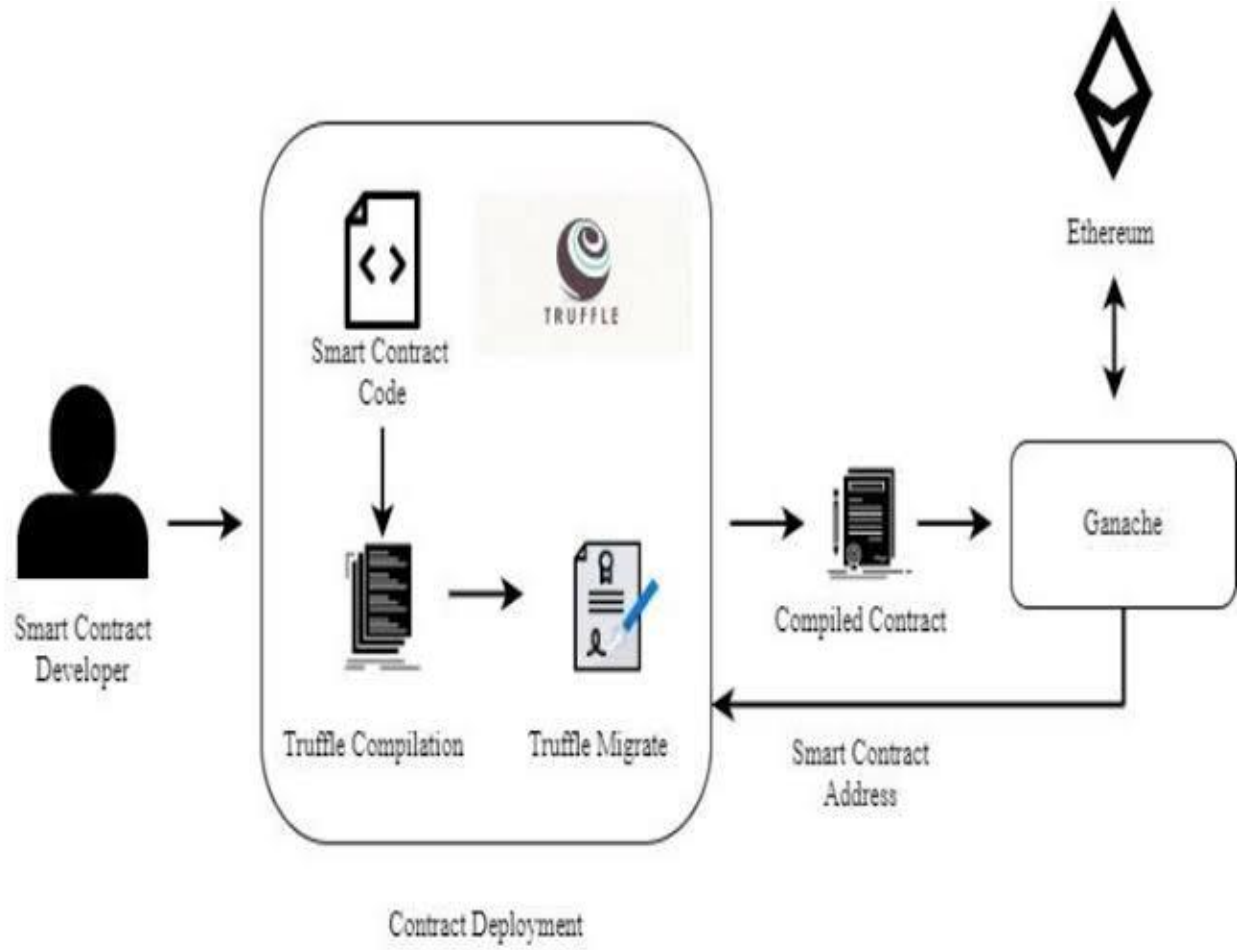
Usability:

The user interface and overall user experience should be intuitive and user-friendly, enabling users to easily navigate the decentralized identity system and perform identity-related tasks with minimal effort. This includes providing clear instructions, informative error messages, and helpful prompts to guide users through the identity management processes.

Compliance and Governance:

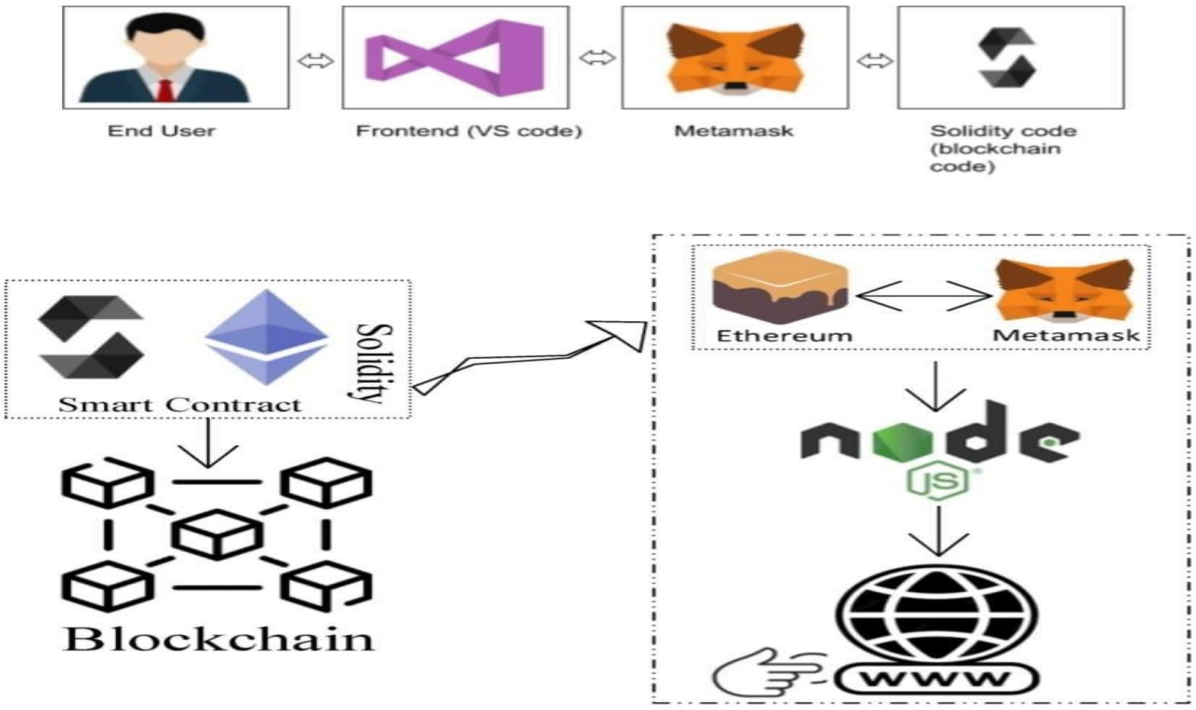
Ensuring compliance with regulatory requirements and industry standards is essential. This includes adhering to data protection regulations, privacy laws, and other legal frameworks governing the storage and management of user identity information.

5.PROJECT DESIGN
5.1 Data Flow Diagram & User Stories



Ethereum's decentralized identity system involves user input of personal information, including identity attributes and access permissions, which is then verified through cryptographic methods. The verified data is registered on the Ethereum blockchain, creating a decentralized record that enables secure identity authentication and access control. On the other hand, Ethereum smart contracts are deployed on the blockchain, defining specific terms and conditions for automated execution. Users interact with these contracts, triggering predefined actions or transactions that are validated and executed on the Ethereum network. The results of these interactions, such as value transfers or state changes, are recorded on the blockchain, ensuring transparency and immutability.

5.2 Solution Architecture



The architecture of an Ethereum decentralized identity smart contract solution typically comprises distinct layers that collectively facilitate secure and efficient identity management on the blockchain. At the forefront is the User Interface Layer, housing the user-facing components like web applications or mobile interfaces enabling users to interact with the system, authenticate themselves, and conduct identity-related tasks. Below this, the Identity Management Layer governs the verification, authentication, and storage of identity data through specialized smart contracts, incorporating features for credential verification and data integrity checks. Serving as the foundation is the Blockchain Layer, constituting the Ethereum network and the Ethereum Virtual Machine (EVM), responsible for executing transactions and recording identity data securely. The Security and Privacy Layer employs encryption and access controls to fortify the system against unauthorized access and potential breaches. Ensuring seamless integration with external systems, the Interoperability Layer incorporates protocols facilitating data exchange between various blockchain platforms management.

6.PROJECT PLNNING & SCHEDULING

6.1 Technical Architecture

In an Ethereum decentralized identity smart contract system, the technical architecture is meticulously structured to ensure the secure and efficient management of identities on the blockchain. At its core, the Smart Contract Layer comprises a series of Ethereum smart contracts tailored for identity management, handling critical functions such as user authentication, identity verification, and the secure storage of identity-related data within the Ethereum blockchain. Supporting this layer is the Blockchain Layer, providing the decentralized ledger where all identity transactions and data are recorded, ensuring transparency and data immutability. To bolster security and privacy, the architecture incorporates various cryptographic protocols and algorithms, including digital signatures, cryptographic hashing functions, and encryption mechanisms, safeguarding sensitive data and validating identity transactions. Furthermore, the system integrates interoperability standards and protocols to facilitate seamless communication and data exchange with other blockchain networks and decentralized applications, enhancing the system's functionality and usability. Off-chain storage solutions are also implemented to efficiently handle substantial identity-related data volumes, optimizing system performance and scalability. Identity providers and verifiers may participate in the identity verification process, bolstering trust and reliability within the decentralized identity system. Together, these components establish a comprehensive technical architecture that effectively manages identities on the Ethereum blockchain while ensuring robust security, privacy, and interoperability.

6.2 SPRINT PLANNING AND ESTIMATION

Backlog Refinement:

The team reviews and refines the product backlog, ensuring that the user stories and tasks are well-defined and prioritized based on their importance and feasibility.

Feature Selection:

The team selects a set of features and tasks from the product backlog to be included in the upcoming sprint, taking into account the66 project's objectives and the current development status.

Task Breakdown:

The selected features and tasks are further broken down into smaller,

manageable tasks that can be completed within the sprint duration. Each task is assigned to specific team members based on their skills and expertise.

Estimation:
The team estimates the effort required to complete each task, considering factors such as the complexity of the task, potential challenges, and dependencies. Techniques like story points or task hours are commonly used for estimation.

Capacity Planning:
Based on the team's capacity and availability, the team determines how many tasks can be taken on during the sprint, ensuring that the workload is manageable and realistic.

Sprint Goal Definition:
The team defines a clear sprint goal that outlines the specific objectives and deliverables to be achieved by the end of the sprint, providing a shared understanding of what the team aims to accomplish.

Task Assignment and Commitment:
Tasks are assigned to team members, and the team collectively commits to completing the selected tasks within the sprint duration, fostering a sense of ownership and accountability.

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)

```
pragma solidity ^0.8.0;

contract DecentralizedIdentity {
    struct User {
        uint id;
        string name;
        address userAddress;
    }
```

```
    mapping(address => User) public users;
    uint public userCount;

    function registerUser(string memory _name) public {
        userCount++;
        users[msg.sender] = User(userCount, _name, msg.sender);
    }
}

pragma solidity ^0.8.0;

contract DecentralizedIdentity {
    mapping(address => bool) public verifiedIdentities;

    function verifyIdentity(address _userAddress) public {
        require(verifiedIdentities[_userAddress] == false, "Identity already
verified");
        verifiedIdentities[_userAddress] = true;
    }
}
```

7.2 Feature 2

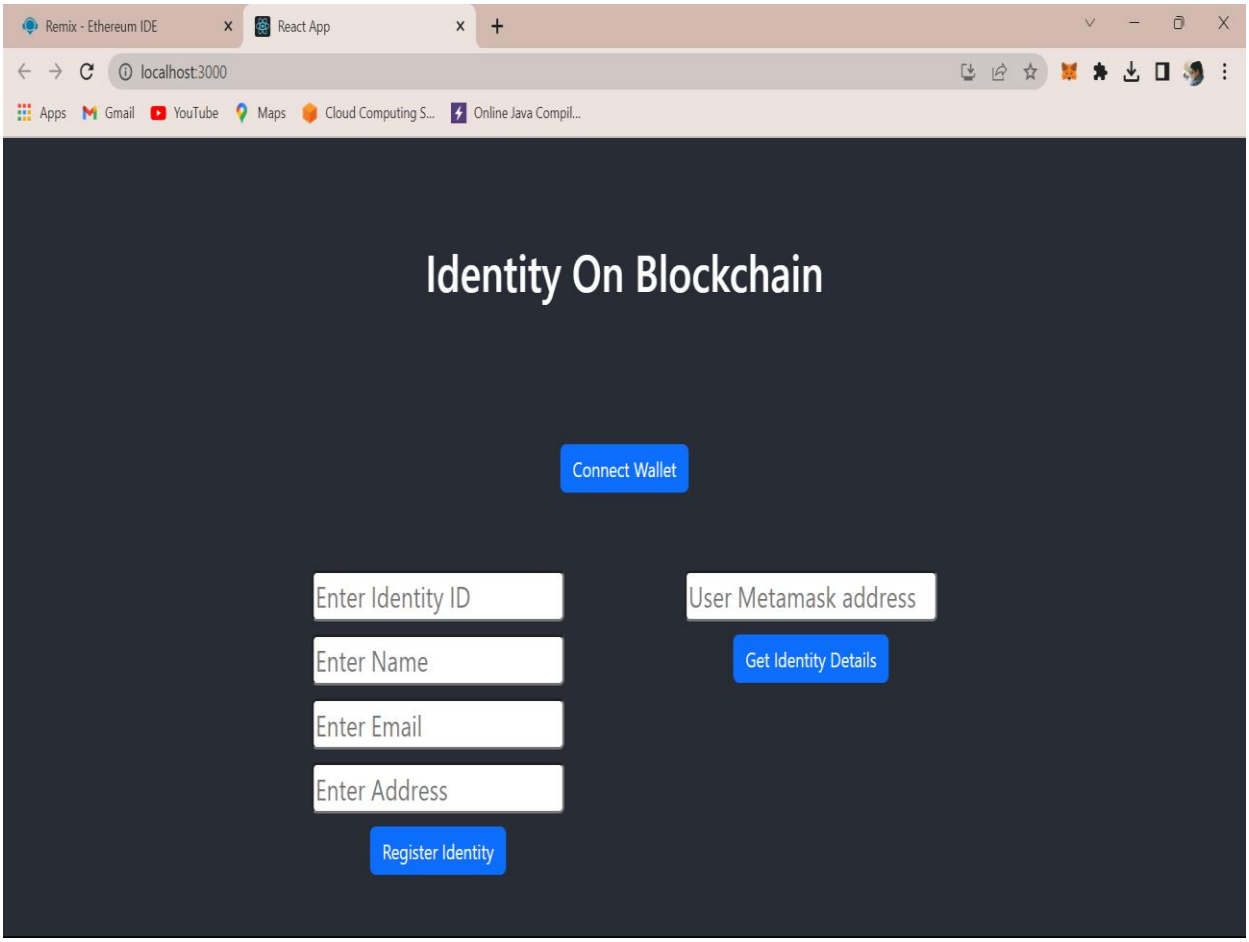
User registration feature

Ethereum decentralized identity smart contract, the user registration feature plays a crucial role in establishing user identities securely on the blockchain. Typically implemented using Solidity, the contract includes a structured representation of a user, containing unique attributes such as an identification number, the user's name, and their Ethereum address. Managed through a mapping structure, the contract maintains a record of each user's details, ensuring efficient storage and retrieval. The registration process involves a function that allows users to submit their name, triggering the assignment of a unique identifier and the creation of their user profile within the decentralized system.

Identity verification feature

Ethereum decentralized identity smart contract, the identity verification feature is instrumental in establishing the authenticity and trustworthiness of

user identities on the blockchain. Typically integrated using Solidity, the contract incorporates mechanisms for verifying the identity of users, ensuring that each user's information is validated and confirmed. This process is often facilitated through a function that checks whether a user's identity has already been verified and, if not, marks the identity as verified upon successful validation. By leveraging cryptographic protocols and secure verification procedures, the smart contract promotes a robust and secure identity verification process, enhancing the overall reliability and credibility of the decentralized identity system.



8.PEFORMANCE TESTING

8.1 performance Metrics

Transaction Throughput:

This metric measures the number of identity-related transactions that the smart contract system can process within a specific time frame. It helps evaluate the system's capacity to handle a high volume of identity verification and registration requests efficiently.

Transaction Latency:

Transaction latency refers to the time it takes for a transaction to be processed within the smart contract system. Monitoring transaction latency helps identify any delays or bottlenecks in the identity verification and registration processes, allowing for timely optimizations and improvements.

Scalability:

Scalability assesses the system's ability to accommodate a growing number of users and identity-related data without compromising performance or security. Evaluating scalability metrics helps ensure that the smart contract system can handle an increasing workload and user base effectively.

Resource Utilization:

Resource utilization metrics track the utilization of computing resources, such as CPU, memory, and storage, within the Ethereum network. Efficient resource management is crucial for maintaining optimal system performance and preventing resource exhaustion or inefficiencies.

Security Audits and Compliance:

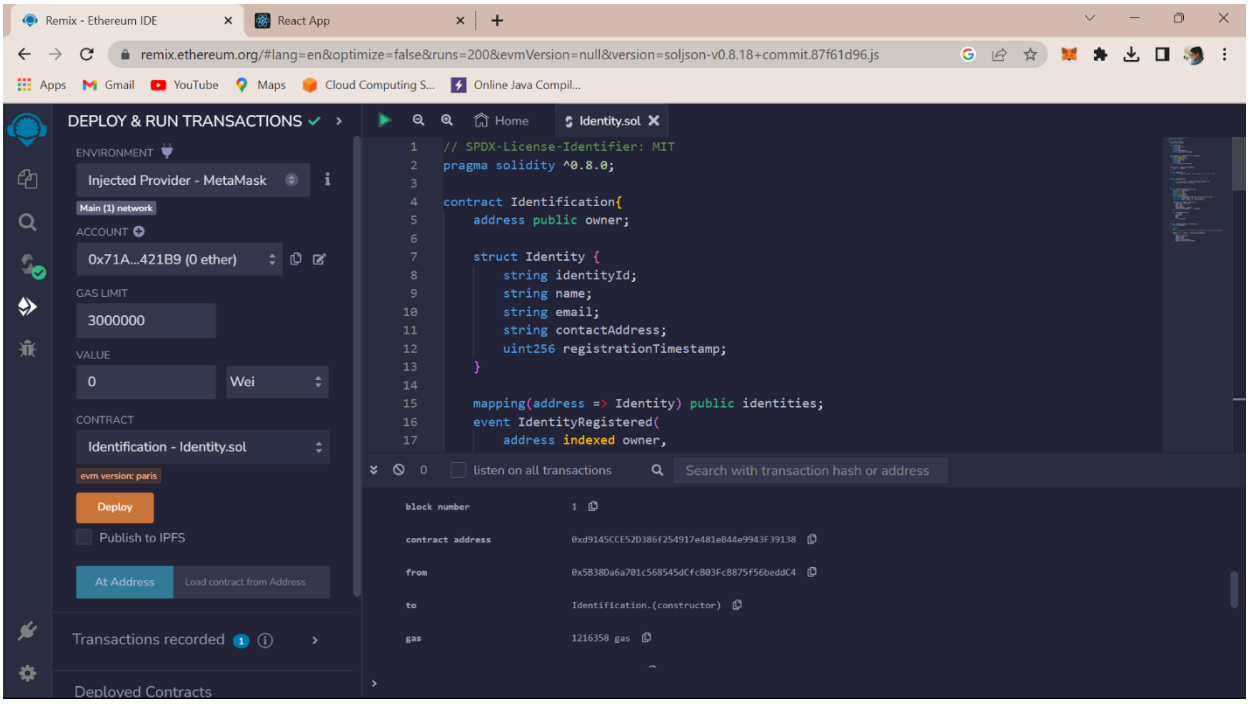
Regular security audits and compliance checks help assess the system's adherence to security best practices and regulatory requirements. These audits ensure that the decentralized identity smart contract system meets industry standards for data protection, privacy, and security.

Response Time:

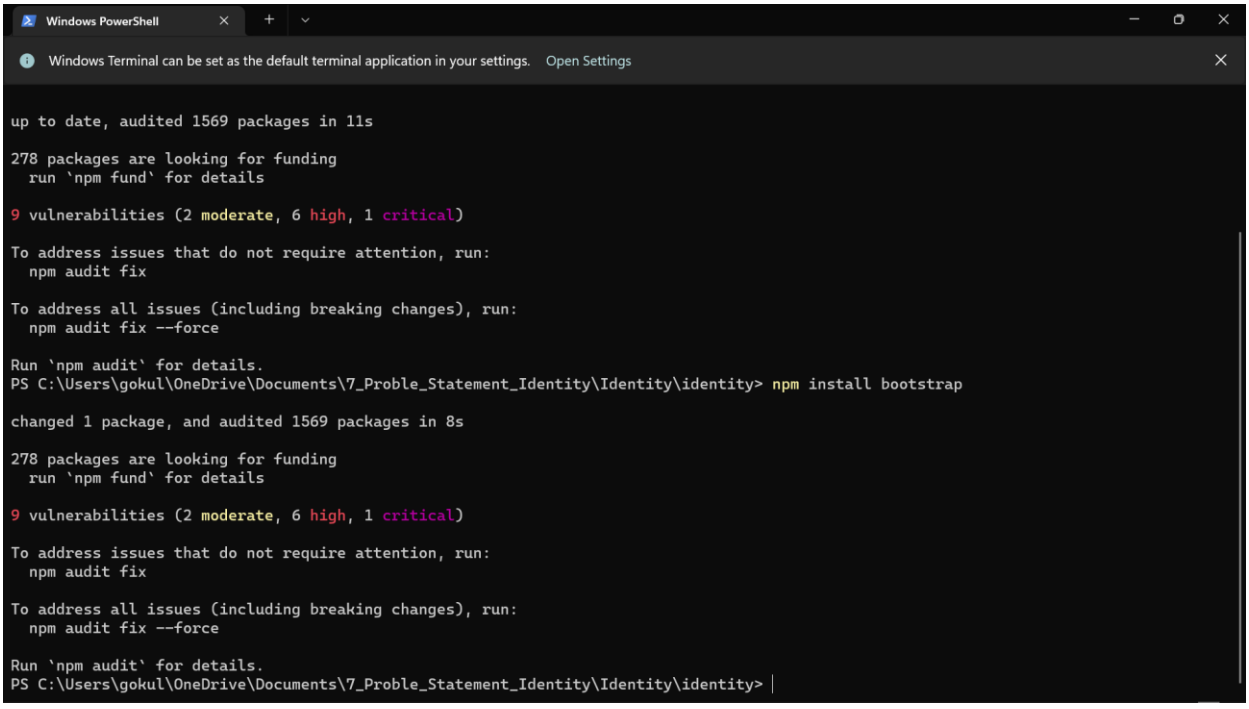
Response time measures the time taken for the system to respond to identity verification and registration requests. Monitoring response time helps identify potential performance issues and optimize system components to enhance overall responsiveness.

9.RESULTS

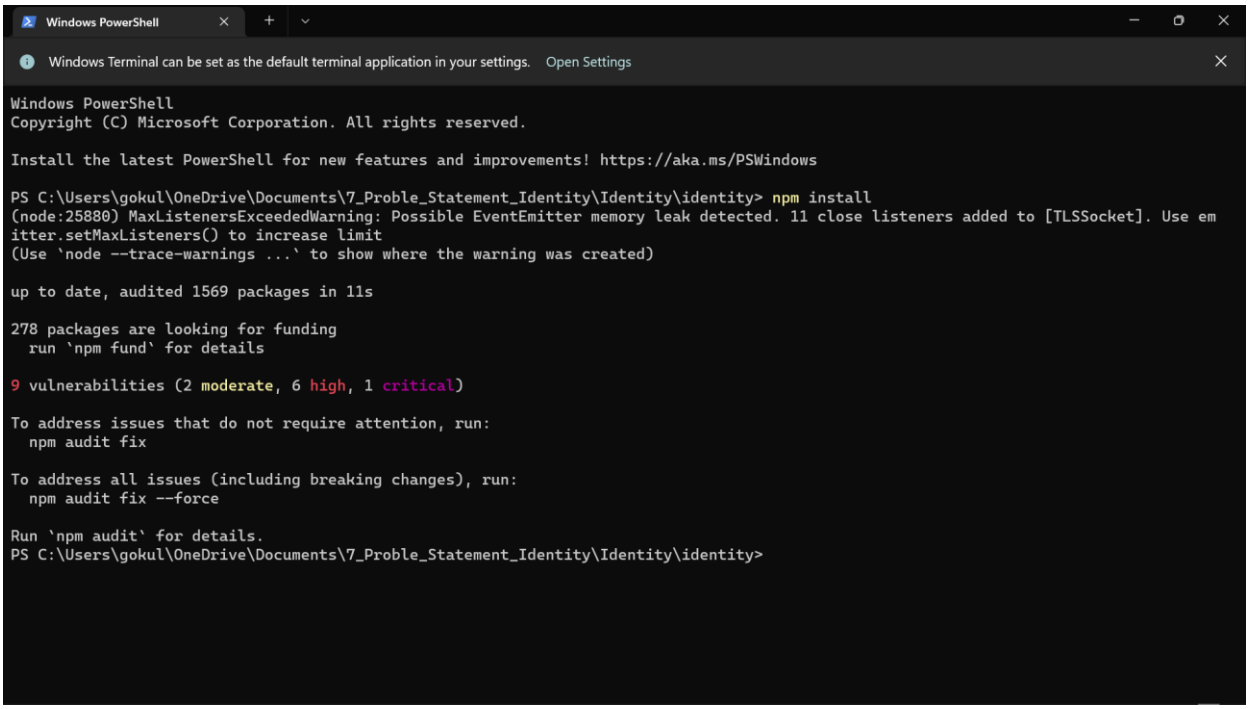
9.1 Output Screenshots



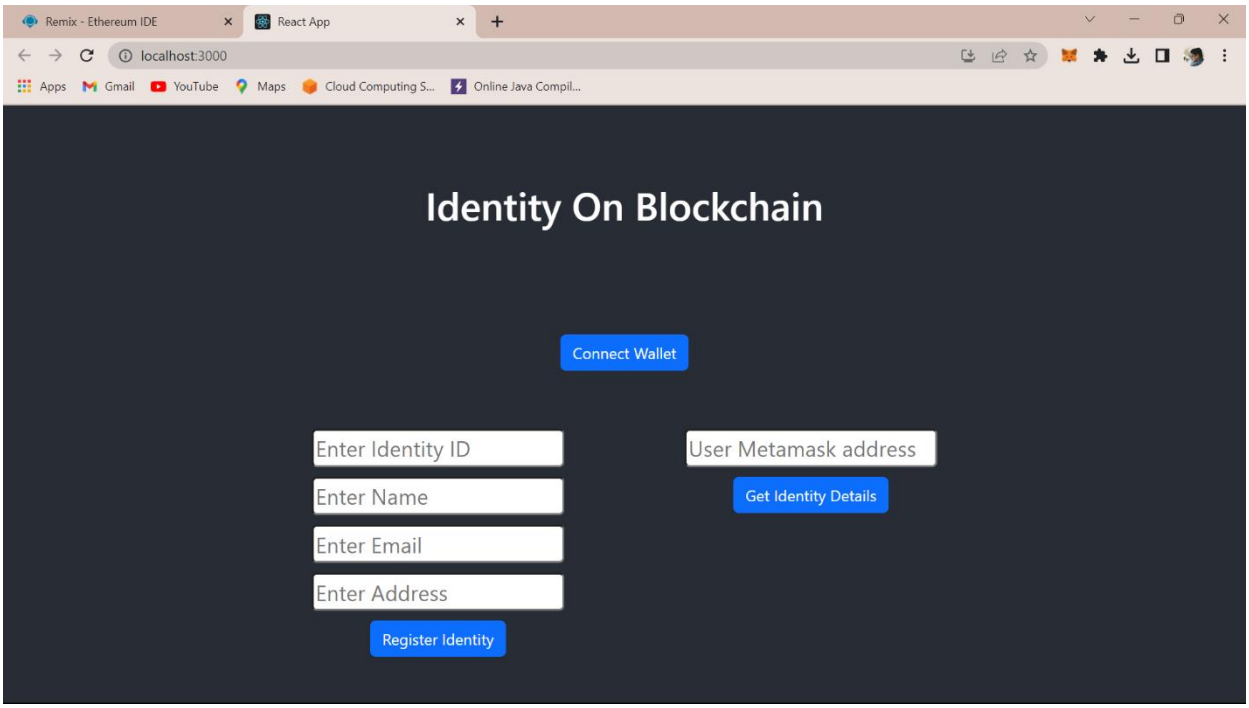
CREATING SMART CONTRACT



INSTALLING DEPENDENCIES



HOSTING THE SITE LOCALLY



OUTPUT SCREEN

10. ADVANTAGES AND DISADVANTAGES

10.1Advantages

Transaction Throughput: This metric measures the number of identity-related transactions that the smart contract system can process within a specific time frame. It helps evaluate the system's capacity to handle a high volume of identity verification and registration requests efficiently.

Transaction Latency: Transaction latency refers to the time it takes for a transaction to be processed within the smart contract system. Monitoring transaction latency helps identify any delays or bottlenecks in the identity verification and registration processes, allowing for timely optimizations and improvements.

Scalability: Scalability assesses the system's ability to accommodate a growing number of users and identity-related data without compromising performance or security. Evaluating scalability metrics helps ensure that the smart contract system can handle an increasing workload and user base effectively.

Resource Utilization: Resource utilization metrics track the utilization of computing resources, such as CPU, memory, and storage, within the Ethereum network. Efficient resource management is crucial for maintaining optimal system performance and preventing resource exhaustion oSecurity: Decentralized identity systems provide a higher level of security and privacy compared to traditional centralized systems. Users have more control over their personal data, reducing the risk of data breaches.

User Control: Users have greater control over their identity information and can choose what information to share with whom. They can manage their digital identities without relying on intermediaries, enhancing user autonomy.

10.2 Disadvantages

Complexity: Implementing decentralized identity solutions can be complex, requiring a good understanding of cryptography and blockchain technology. This complexity may deter some users and developers from adopting these solutions.

Regulatory Challenges: The regulatory environment for decentralized identity is still evolving, and compliance with existing regulations can be challenging. Adherence to privacy laws and other regulatory requirements may pose obstacles to widespread adoption.

User Experience: User experience issues, such as the management of multiple keys, authentication processes, and recovery mechanisms, can pose usability challenges for mainstream adoption.

Privacy Concerns: While decentralized identity systems aim to enhance user privacy, they may also introduce new privacy concerns related to the storage and management of personal data on the blockchain.

Immutability Risks: Smart contracts, once deployed, are immutable, which means that any bugs or vulnerabilities in the code cannot be easily rectified without deploying a new contract. This lack of flexibility can lead to significant issues if not properly addressed during the development phase.

Security Vulnerabilities: Despite being touted as secure, smart contracts are still susceptible to security vulnerabilities and exploits. Flaws in the code or unexpected interactions with other contracts can lead to significant financial losses.

11.Conclusion

Decentralized identity solutions on the Ethereum blockchain provide enhanced security, user control, and privacy. However, complexities in implementation, regulatory challenges, and potential privacy concerns may impede widespread adoption.On the other hand, Ethereum's smart contracts automate processes, ensure transparency, and reduce costs, fostering trust and efficiency. Yet, immutability risks, security vulnerabilities, scalability challenges, and energy consumption concerns remain critical areas for improvement.Despite these challenges, Ethereum's decentralized identity and smart contracts hold immense potential to transform various sectors, such as finance, supply chain, healthcare, and governance. As the ecosystem evolves, addressing these drawbacks through technological advancements and regulatory developments will be crucial for realizing the full promise of Ethereum's decentralized identity and smart contract capabilities.

12.FUTURE SCOPE

Enhanced Privacy Solutions:

Continued advancements in decentralized identity systems may focus on further enhancing user privacy through advanced cryptographic techniques and zero-knowledge proofs, allowing for secure and private transactions and interactions.

Interoperability Across Blockchains:

Efforts to improve interoperability between different blockchain networks can enable seamless data and identity management across various decentralized applications, fostering a more connected and accessible digital ecosystem.

Regulatory Integration and Compliance:

Collaboration with regulatory bodies and the integration of compliance standards within decentralized identity and smart contract solutions can facilitate regulatory acceptance and streamline adoption in regulated industries.

Scalability and Energy Efficiency:

Ethereum's ongoing transition to a more energy-efficient consensus mechanism, such as Proof of Stake (PoS), along with the development of layer 2 scaling solutions, can enhance the scalability and energy efficiency of decentralized identity and smart contract applications.

Enterprise Adoption:

Increased enterprise adoption of Ethereum's decentralized identity and smart contract solutions can revolutionize supply chain management, digital identity verification, and automated contract execution, leading to improved operational efficiency and cost savings.

Cross-Industry Applications:

Integration of decentralized identity and smart contracts in various sectors, including finance, healthcare, real estate, and governance, can streamline processes, reduce fraud, and enhance transparency, leading to a more secure and trustworthy digital environment.

User-Friendly Interfaces:

Simplified user interfaces and enhanced user experiences for managing decentralized identities and interacting with smart contracts can drive mainstream adoption and make blockchain technology more accessible to the general public.

12.APPENDIX

Source Code

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

contract Identification{
    address public owner;

    struct Identity {
        string identityId;
        string name;
        string email;
        string contactAddress;
        uint256 registrationTimestamp;
    }

    mapping(address => Identity) public identities;
    event IdentityRegistered(
        address indexed owner,
        string identityId,
        string name,
        string email,
        uint256 registrationTimestamp
    );

    constructor() {
        owner = msg.sender;
    }

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

```
    _;
}

modifier notRegistered() {
    require(
        bytes(identities[msg.sender].identityId).length == 0,
        "Identity already registered"
    );
    _;
}

function registerIdentity(
    string memory identityId,
    string memory name,
    string memory email,
    string memory _address
) external notRegistered {
    require(bytes(identityId).length > 0, "Invalid identity ID");
    require(bytes(name).length > 0, "Invalid name");
    require(bytes(email).length > 0, "Invalid email");

    identities[msg.sender] = Identity({
        identityId: identityId,
        name: name,
        email: email,
        contactAddress : _address,
        registrationTimestamp: block.timestamp
    });

    emit IdentityRegistered(
        msg.sender,
        identityId,
        name,
        email,
        block.timestamp
    );
}
```

```
    function getIdentityDetails(
        address userAddress
    )
        external
        view
        returns (string memory, string memory, string memory, string memory,uint256)
    {
        Identity memory identity = identities[userAddress];
        return (
            identity.identityId,
            identity.name,
            identity.email,
            identity.contactAddress,
            identity.registrationTimestamp
        );
    }
}
```

Connector.js

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

const abi = [
 {
  "inputs": [],
  "stateMutability": "nonpayable",
  "type": "constructor"
 },
 {
  "anonymous": false,
  "inputs": [
   {
    "indexed": true,
    "internalType": "address",
    "name": "owner",
    "type": "address"
   },
   {
```

```
   "indexed": false,
   "internalType": "string",
   "name": "identityId",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "name",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "email",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "uint256",
   "name": "registrationTimestamp",
   "type": "uint256"
  }
 ],
 "name": "IdentityRegistered",
 "type": "event"
},
{
 "inputs": [
  {
   "internalType": "address",
   "name": "userAddress",
   "type": "address"
  }
 ],
 "name": "getIdentityDetails",
 "outputs": [
  {
```

```
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "uint256",
   "name": "",
   "type": "uint256"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  }
 ],
 "name": "identities",
 "outputs": [
```

```
 {
  "internalType": "string",
  "name": "identityId",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "name",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "email",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "contactAddress",
  "type": "string"
 },
 {
  "internalType": "uint256",
  "name": "registrationTimestamp",
  "type": "uint256"
 }
],
"stateMutability": "view",
"type": "function"
},
{
"inputs": [],
"name": "owner",
"outputs": [
 {
  "internalType": "address",
  "name": "",
  "type": "address"
 }
```

```
  ],
  "stateMutability": "view",
  "type": "function"
 },
 {
  "inputs": [
   {
    "internalType": "string",
    "name": "identityId",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "name",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "email",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "_address",
    "type": "string"
   }
  ],
  "name": "registerIdentity",
  "outputs": [],
  "stateMutability": "nonpayable",
  "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 = "0x418e6373e103F7bf5D73f51b6A43a0cA2EC9d9a5"

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

Home.js

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

const abi = [
 {
  "inputs": [],
  "stateMutability": "nonpayable",
  "type": "constructor"
 },
 {
  "anonymous": false,
  "inputs": [
   {
    "indexed": true,
    "internalType": "address",
    "name": "owner",
    "type": "address"
   },
   {
    "indexed": false,
    "internalType": "string",
    "name": "identityId",
    "type": "string"
   },
   {
    "indexed": false,
    "internalType": "string",
    "name": "name",
    "type": "string"
   },
   {
```

```
   "indexed": false,
   "internalType": "string",
   "name": "email",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "uint256",
   "name": "registrationTimestamp",
   "type": "uint256"
  }
 ],
 "name": "IdentityRegistered",
 "type": "event"
},
{
 "inputs": [
  {
   "internalType": "address",
   "name": "userAddress",
   "type": "address"
  }
 ],
 "name": "getIdentityDetails",
 "outputs": [
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
```

```
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "uint256",
   "name": "",
   "type": "uint256"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  }
 ],
 "name": "identities",
 "outputs": [
  {
   "internalType": "string",
   "name": "identityId",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "name",
   "type": "string"
  },
  {
   "internalType": "string",
```

```
   "name": "email",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "contactAddress",
   "type": "string"
  },
  {
   "internalType": "uint256",
   "name": "registrationTimestamp",
   "type": "uint256"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [],
 "name": "owner",
 "outputs": [
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "string",
   "name": "identityId",
   "type": "string"
  },
  {
```

```
    "internalType": "string",
    "name": "name",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "email",
    "type": "string"
   },
   {
    "internalType": "string",
    "name": "_address",
    "type": "string"
   }
  ],
  "name": "registerIdentity",
  "outputs": [],
  "stateMutability": "nonpayable",
  "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 = "0x418e6373e103F7bf5D73f51b6A43a0cA2EC9d9a5"

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

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.csss

```
.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": "identity",
  "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",
    "bootstrap": "^5.3.1",
    "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"
    ]
  }
}
```

GithHub & Project Demo Link

GithHub : https://github.com/Sanjith52/NM-BLOCKCHAIN-DEVELOPMENT-ETHEREUM-DECENTRALIZED-IDENTITY-SMART-CONTRACT.git

Demo Link : https://drive.google.com/drive/folders/1-10QEsRfXpuV_2ExSwmFLb1tPMyJqJWj