|  |  |
| --- | --- |
| Team ID | NM2023TMID00434 |
| Project Name | Ethereum Decentralised Identity Smart Contarct |

Project Report Format

# INTRODUCTION

* 1. **Project Overview**

A Farmer Insurance Chain Project typically refers to an initiative or system aimed at providing insurance coverage to farmers. The primary goals of such projects are to mitigate the financial risks that farmers face due to various factors like adverse weather conditions, crop failures, pests, and other unpredictable events.

Farmer Insurance Chain Projects may have unique features and objectives based on their location, stakeholders, and the particular challenges faced by farmers in that region. If you have a specific project in mind, I would recommend researching or reaching out to the organization or agency responsible for it to get more detailed and up-to-date information.

**Purpose**

The purpose of a farmer insurance chain, also known as an agricultural insurance system or program, is to provide financial protection and risk mitigation for farmers. This type of insurance is designed to address the specific challenges and uncertainties that farmers face in their agricultural activities

# LITERATURE SURVEY

* 1. **Existing problem**

Farmer insurance programs, also known as agricultural insurance chains or systems, exist in many countries around the world to provide protection to farmers and mitigate the risks associated with agricultural activities. These programs may vary in terms of structure, coverage, and administration, but they generally serve the common purpose of offering insurance to farmers.

* 1. **References**

Agricultural Insurance Organizations: Organizations like the World Bank, the International Fund for Agricultural Development (IFAD), and the International Association of Agricultural Production Insurers (AIAG) often publish reports and research on agricultural insurance.

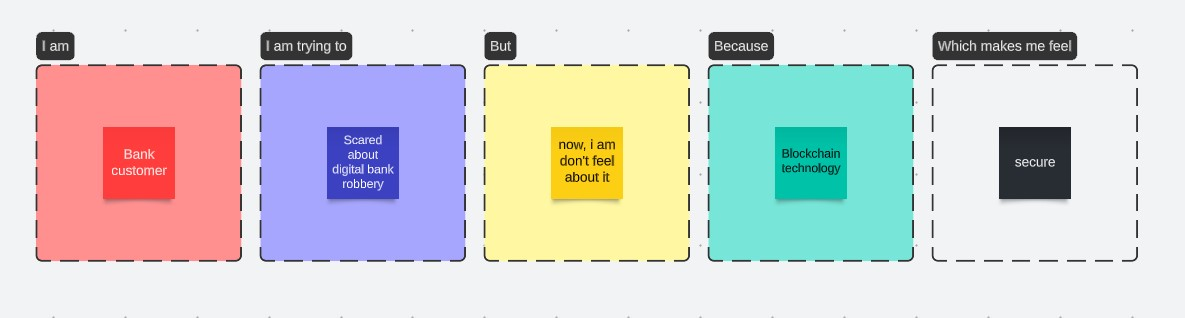
Academic Journals: Scholarly journals related to agriculture, economics, and insurance may contain research articles and studies on farmer insurance. You can search databases like JSTOR, Google Scholar, or your university's library resources.

Industry Publications: Trade magazines and publications related to agriculture and insurance often feature articles and reports on insurance programs for farmers. Look for publications in the agriculture and insurance sectors.

Research Institutions: Universities and research institutions may conduct studies and publish reports on various aspects of farmer insurance. Check the websites of well-known agricultural economics or insurance research centers.

* 1. **problem Statement Definition**

Problem statement 1:

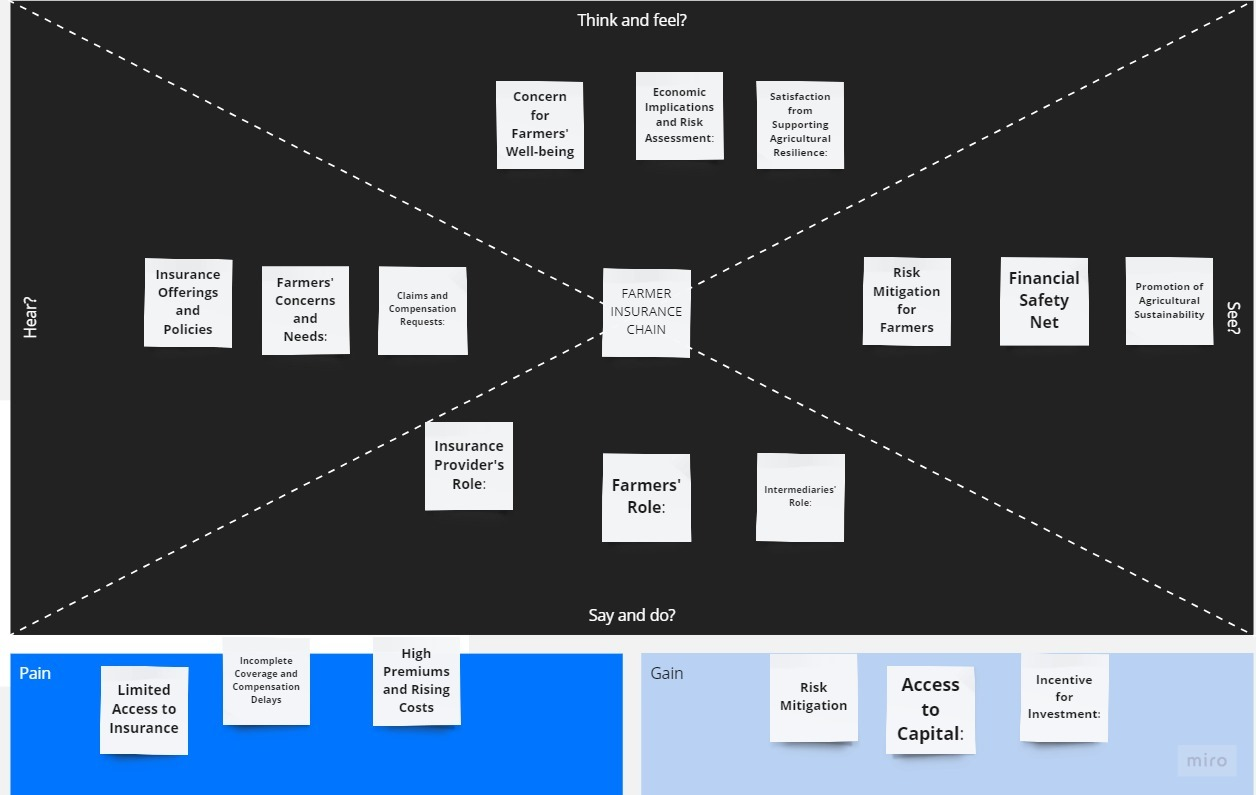


Problem statement 2:

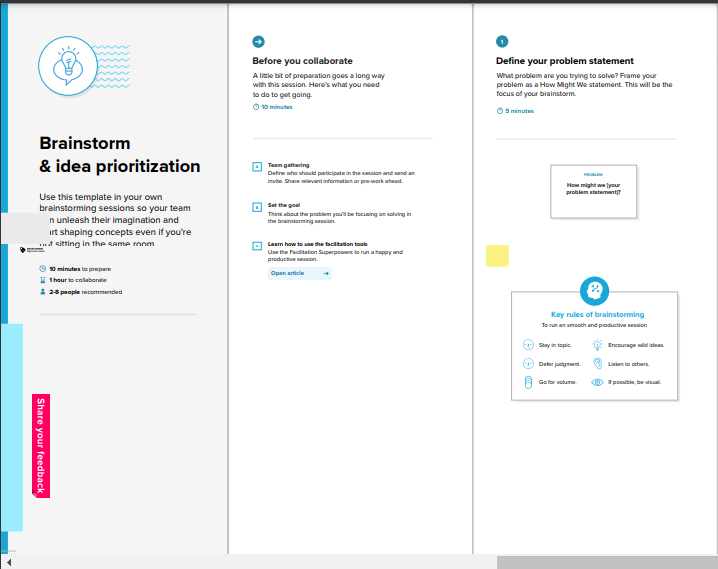


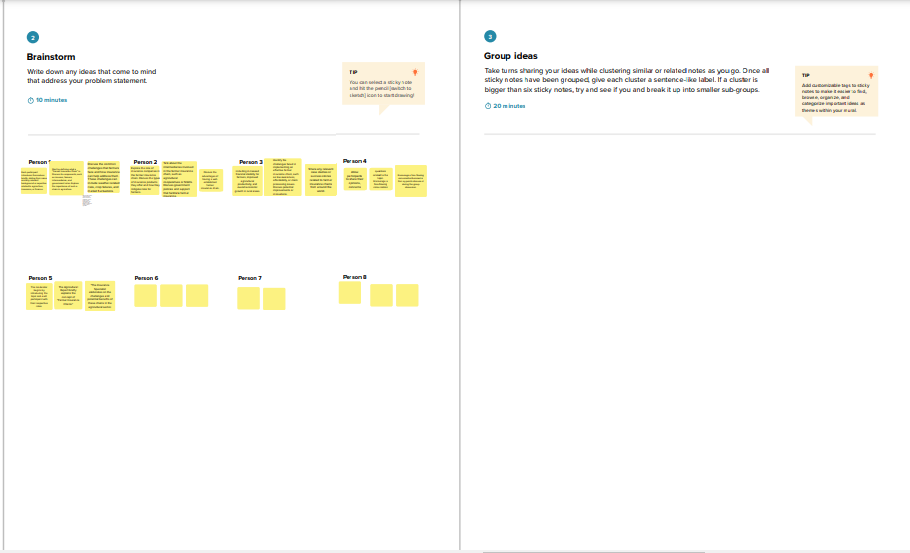
# IDEATION & PROPOSED SOLUTION

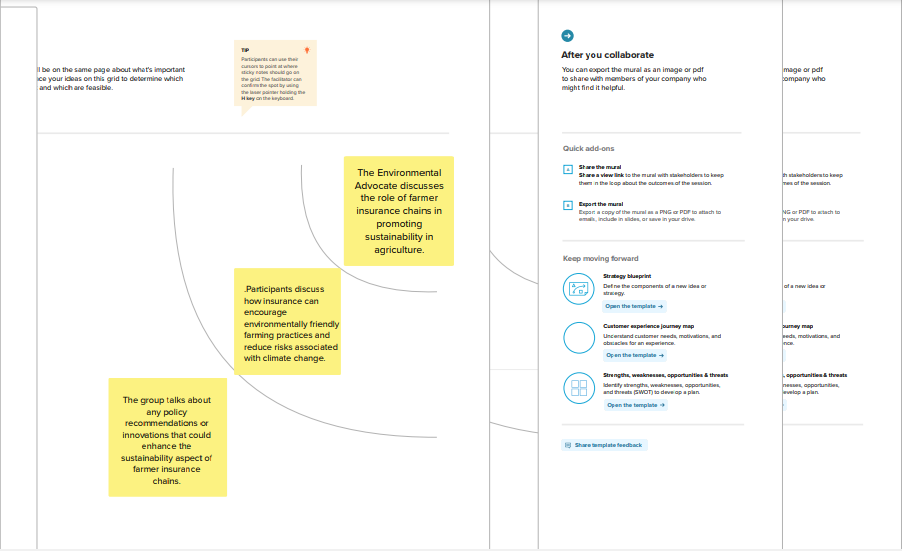
* 1. **Empathy Map Canvas**



* 1. **Ideation & Brainstorming**

****

****

****

# REQUIREMENT ANALYSIS

* 1. **Functional requirement**

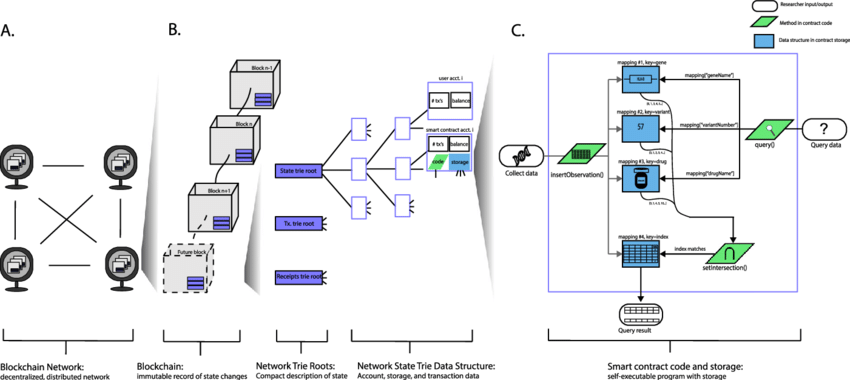
Functional requirements for Ethereum decentralized identity smart contracts typically specify the capabilities and features that the smart contract should possess to effectively manage decentralized identities on the Ethereum blockchain.

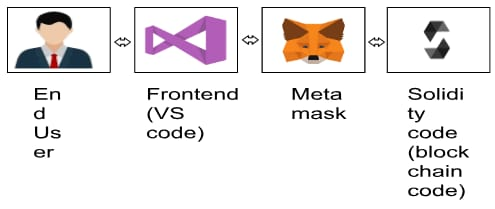
1. Cross-Platform Compatibility: Ensure that the decentralized identity system is compatible with various platforms and devices, making it accessible and functional for users on different devices and environments.
2. Identity Creation: Users should be able to create a new decentralized identity, including defining attributes like name, contact information, and other relevant personal data.
3. Key Pair Management: The smart contract must handle the creation, storage, and management of cryptographic key pairs for the user. This includes generating and securing public and private keys.
4. Identity Verification: Users should be able to request and receive verifiable credentials from trusted sources, such as educational institutions, government agencies, or banks
   1. **Non-Functional requirement**

Non-functional requirements for Ethereum decentralized identity smart contracts encompass qualities or characteristics of the system beyond its core functionality. These requirements focus on aspects such as performance, security, usability, and reliability.

# PROJECT DESIGN

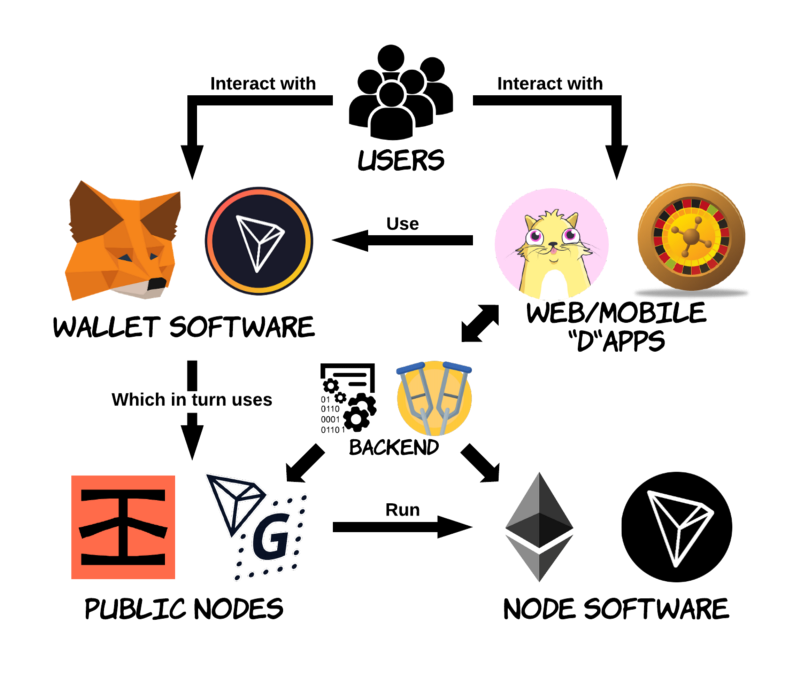
* 1. **Data Flow Diagrams & User Stories**





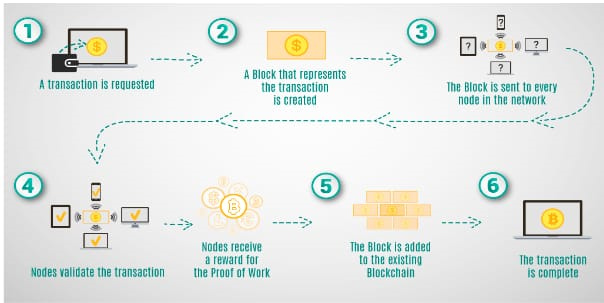
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User Type** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Acceptance Criteria** | **Priority** | **Team Member** |
| New User | User Registration | US01 | As a new user, I want to create a decentralized identity on Ethereum to control my personal information securely.  . | * The user can complete the registration process. * The user's decentralized identity is stored on the Ethereum blockchain. * The user receives their unique Decentralized Identifier (DID). | High | G.GOKULRAJ |
| Identity Owner | Identity Update | US02 | As a user, I want to update my decentralized identity with new information, such as a change of address or a new email. | * The user can access and edit their identity information. * Changes are securely recorded on the blockchain. * The user's private key is required to initiate updates. | Medium | P. Kumaravel |
| Relying Party (Third Party) | Credential Validation | US03 | As a relying party, I want to verify the authenticity of verifiable credentials presented by a user to ensure the integrity of identity-related transactions. | * The relying party can access the user's presented verifiable credentials. * The credentials are securely verified. * The relying party can trust the validity of the credentials for the transaction. | High | K.SIVASAKTHI |
| Identity Owner | Privacy Preferences | US04 | As a user, I want to set privacy preferences for my identity to control how much information is shared during transactions and interactions. | * The user can access and configure privacy settings. * Privacy settings are clearly defined and user-friendly. * The user can restrict or allow the sharing of specific attributes. | Medium | P. Sneha |

* 1. **Solution Architecture**



# PROJECT PLANNING & SCHEDULING

**Technical Architecture**



* 1. **Sprint Planning & Estimation**

1. Define the Scope:

Clearly define the features and functionality you want in your DID smart contract. For Ethereum, this typically includes managing identity claims, public keys, and the associated data in a decentralized and secure manner.

2. User Stories and Epics:

Break down the project into user stories and epics. For example, user stories may include user registration, claim issuance, claim verification, and claim revocation.

3. Story Estimation:

Estimate the effort required for each user story or epic. You can use story points or time-based estimates (e.g., in hours) for this purpose.

4. Prioritization:

Prioritize the user stories based on their importance and dependencies. Identify the minimum viable product (MVP) features that should be implemented first.

5. Sprint Planning:

Determine the sprint length (e.g., 2 weeks) and select a set of user stories to be completed during the sprint. These stories should align with your project's objectives and priorities.

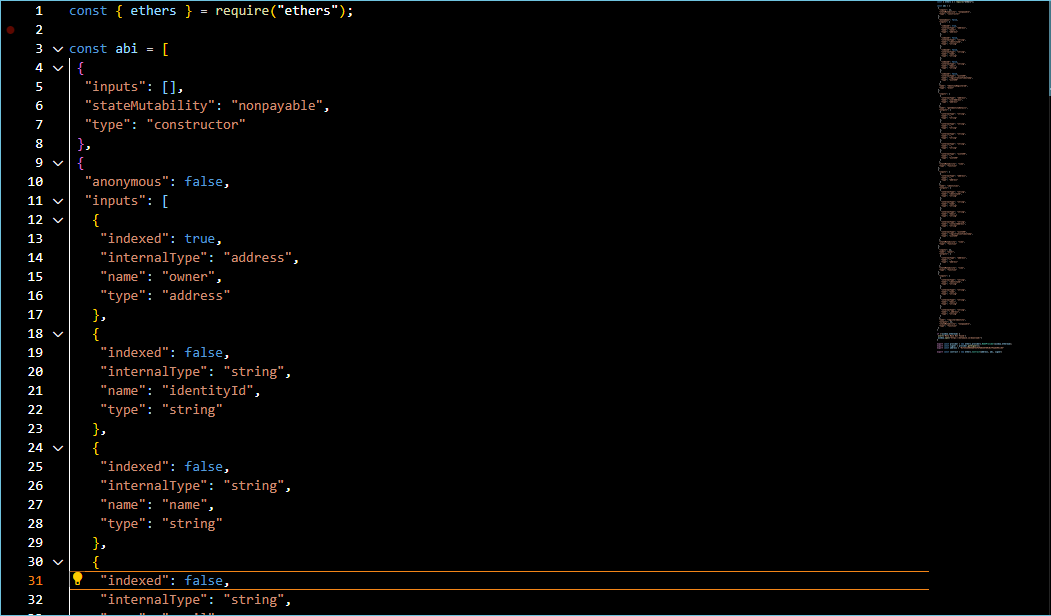
* 1. **Sprint Delivery Schedule**

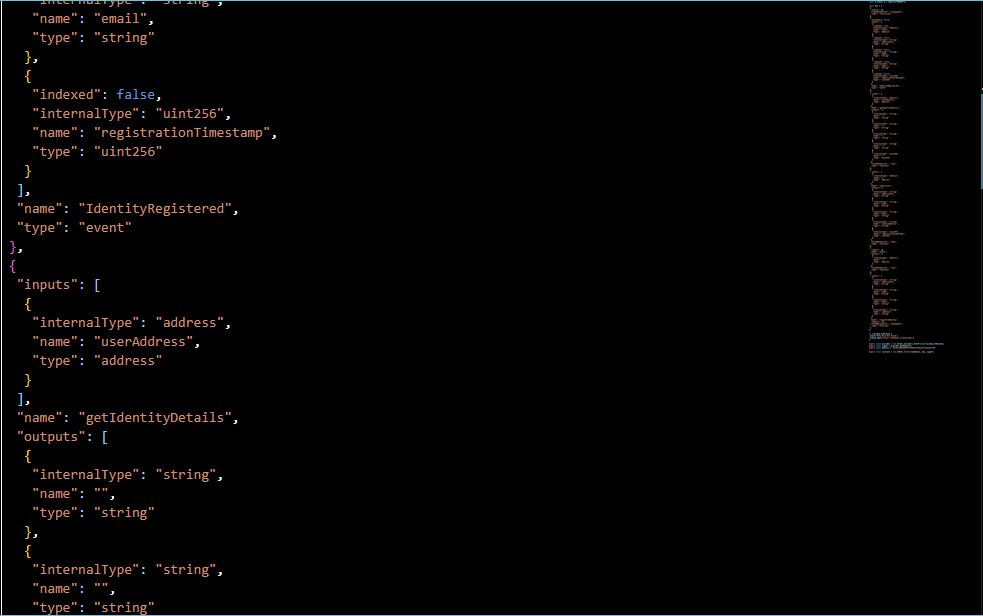
Creating a sprint delivery schedule for an Ethereum decentralized identity (DID) smart contract project involves breaking down the project into manageable tasks and allocating them to specific sprints. A typical sprint duration is two weeks, but it can vary based on your team's preferences and project complexity.

# CODING & SOLUTIONING (Explain the features added in the project along with code)

* 1. **Feature 1**

Ethereum decentralized identity (DID) smart contracts are designed to provide users with control over their digital identities and personal data while ensuring security and privacy.

****

****

* 1. **Feature 2**

Ethereum DID smart contracts are a fundamental component of decentralized identity systems in the Web3 ecosystem, enabling users to have greater control over their digital identities, data, and privacy. The specific features and design of these contracts can vary depending on the project's goals and requirements.

// 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

        );

    }

}

# PERFORMANCE TESTING

* 1. **Performace Metrics**

**[vm]**

**from:** 0x5B3...eddC4

**to:** Identification.(constructor)

**value:** 0 wei

**data:** 0x608...20033

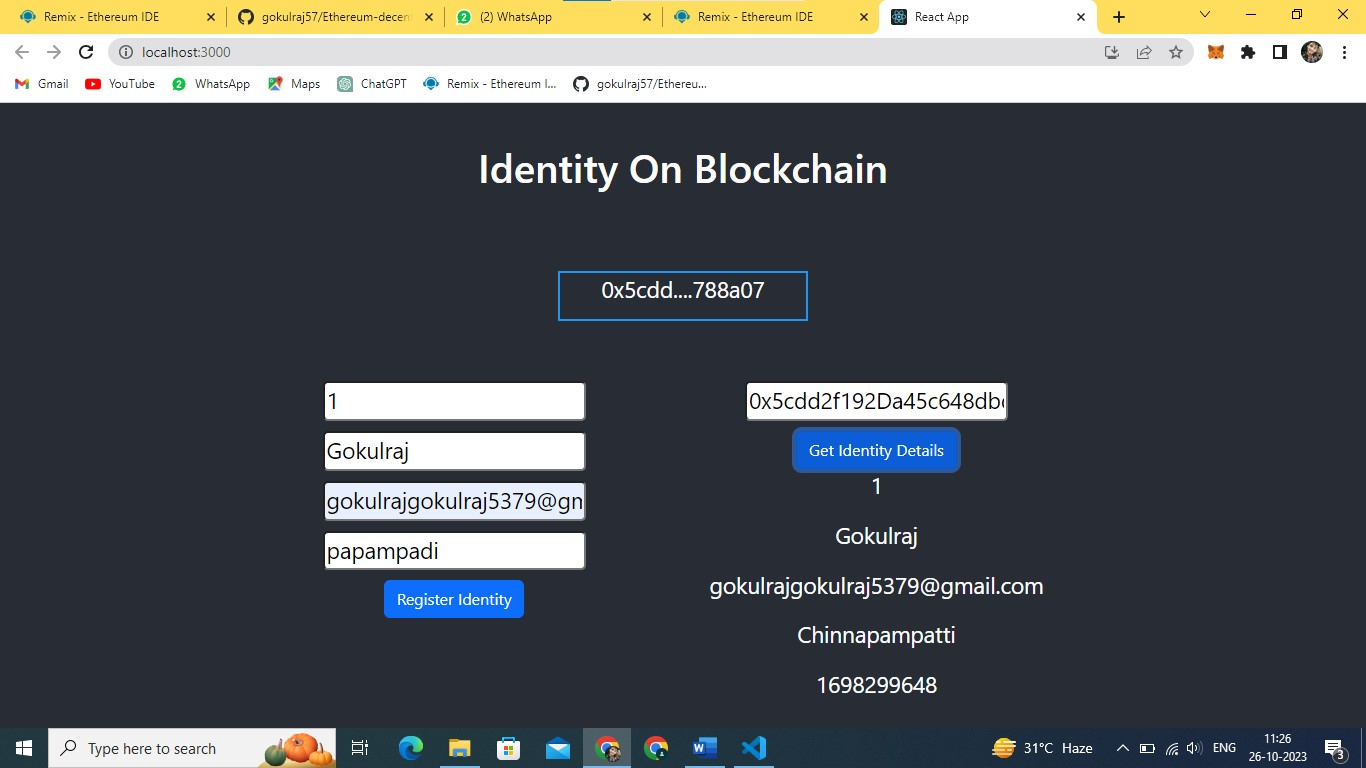
**logs:** 0

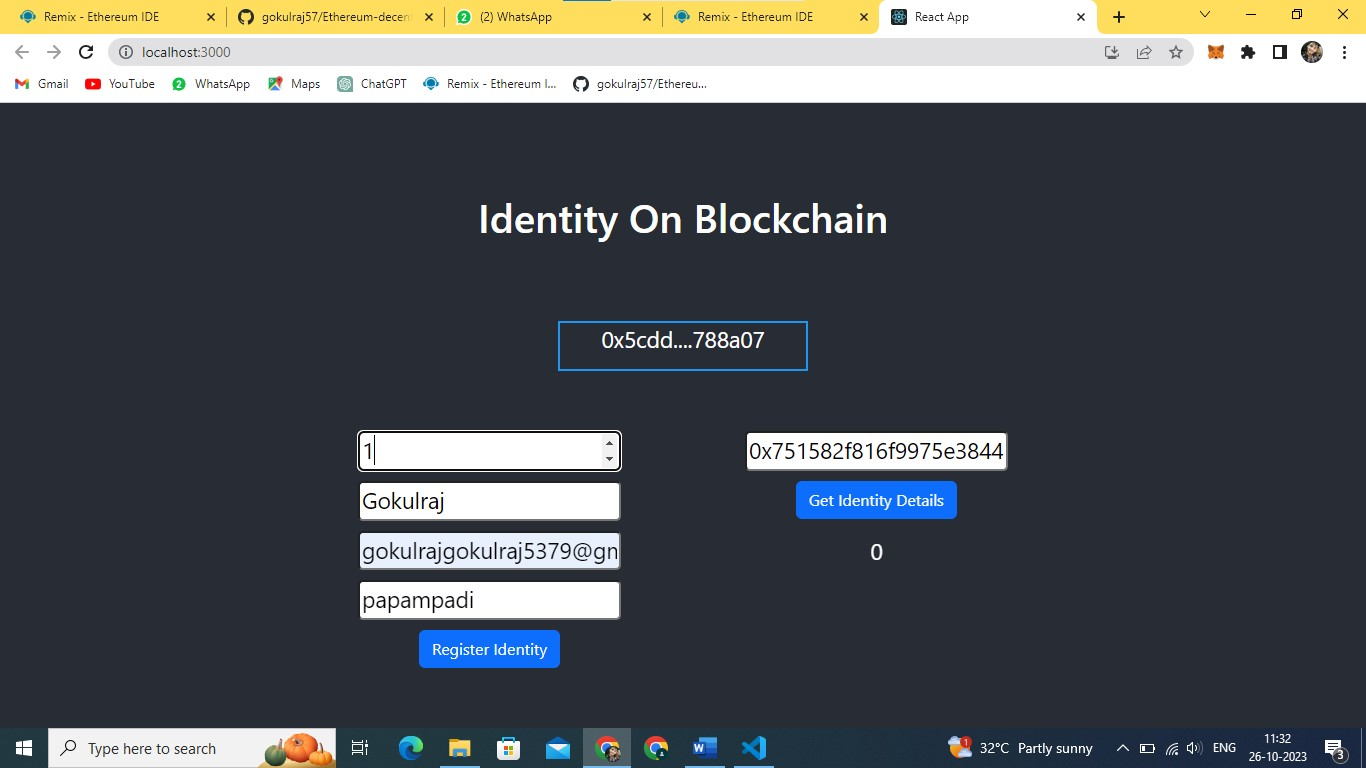
**hash:** 0xf3d...ebf93

|  |  |  |
| --- | --- | --- |
| **status** | true Transaction mined and execution succeed | |
| **transaction hash** | | 0xf3d9cff106d10a96f7713949679bdef034ba98fdd1fcd819698a346204cebf93 | |
| **block hash** | | 0x6b3068a1bcace3a1b3f6148e282a3d5db2491915ee54d16635c82cba52f3606e | |
| **block number** | | 1 | |
| **contract address** | | 0xd9145CCE52D386f254917e481eB44e9943F39138 | |
| **from** | | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 | |
| **to** | | Identification.(constructor) | |
| **gas** | | 1216358 gas | |
| **transaction cost** | | 1057994 gas | |
| **execution cost** | | 934714 gas | |
| **input** | | 0x608...20033 | |
| **decoded input** | | {} | |
| **decoded output** | | - | |
| **logs** | | [] | |
| **val** | | 0 wei | |

# RESULTS

**Output Screenshots**





# ADVANTAGES & DISADVANTAGES

# ADVANTAGES:

* User Control:

Users have complete control over their digital identities. They can create, update, and manage their decentralized identities and associated claims.

* Privacy:

Users can selectively disclose information, enhancing privacy. They only share the information they choose, reducing the risk of overexposing personal data.

* Security:

Ethereum DID smart contracts utilize strong cryptographic methods, making it difficult for malicious actors to tamper with or forge identities and claims.

* Immutability:

Once recorded on the Ethereum blockchain, DIDs and claims are immutable and tamper-resistant, providing a high level of trust in the data.

**DISADVANTAGES:**

* Complexity: Implementing and managing Ethereum DID smart contracts can be complex, requiring a deep understanding of blockchain technology, cryptography, and identity standards.
* Scalability: The Ethereum blockchain has limitations in terms of scalability. As more users adopt DID solutions, network congestion and high gas fees can become issues.
* Smart Contract Vulnerabilities: Smart contracts can have vulnerabilities that might be exploited by malicious actors if not properly audited and secured.

1. **CONCLUSION**

In conclusion, Ethereum decentralized identity (DID) smart contracts represent a promising and innovative approach to digital identity management, providing individuals with greater control, security, and privacy over their personal data. While there are advantages to using Ethereum DID smart contracts, including user empowerment, enhanced security, and privacy, there are also challenges and considerations that need to be addressed.

# FUTURE SCOPE

The future of Ethereum decentralized identity smart contracts is closely tied to the broader evolution of blockchain technology, digital identity, and the growing emphasis on user sovereignty and data privacy. As these trends continue to develop, decentralized identity solutions are poised to play a significant role in shaping the digital landscape.

1. **APPENDIX**

**Source Code**

**Solidity 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

);

}

}

**VS Code:**

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 = "0xf241e6055b8f4CF2Fb8C35769cBc7F2aAC95ccE3"

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

**GitHub & Project Demo Link**

**GitHub:**

[**https://github.com/gokulraj57/Ethereum-decentralised-identity-smart-contarct**](https://github.com/gokulraj57/Ethereum-decentralised-identity-smart-contarct)