

# ETHEREUM DECENTRALISED IDENTITY SMART CONTRACT

## 1.INTRODUCTION

### 1.1 Project Overview

The Ethereum Decentralized Identity Smart Contract is a groundbreaking project that aims to leverage blockchain technology, specifically the Ethereum platform, to create a secure and decentralized identity management system. This project seeks to address the issues of identity theft, data breaches, and centralized control over personal information by providing individuals with full ownership and control of their digital identities. It establishes a decentralized identity management system using Ethereum's blockchain capabilities. This project provides users with self-sovereign identity, enabling them to control and share their personal information at their discretion. Ensure the privacy and security of personal data through cryptographic techniques and smart contract functionality. The system enables secure and frictionless identity verification for various online services and applications. Users have full control over their digital identities, allowing them to grant or revoke access to their information as needed. This centralized system that enables users to have full control over their digital identities, allowing them to grant or revoke access to their information as needed. It implements strong encryption and privacy measures to protect users' data from unauthorized access. This blockchain ensures compatibility with established identity standards such as W3C's Decentralized Identifiers (DIDs) and Verifiable Credentials (VCs). Utilize decentralized storage solutions like IPFS (InterPlanetary File System) to store sensitive user data securely. The Ethereum Decentralized Identity Smart Contract project is a pioneering effort to provide individuals with control over their digital identities, enhancing security and privacy while offering a seamless and secure identity verification process for various online services and applications. Through the use of Ethereum's blockchain technology, this project aims to revolutionize the way identities are managed in the digital world, empowering users to take ownership of their personal information.

### 1.2 Purpose

Ethereum Decentralized Identity (DID) smart contract is to provide a secure, self-sovereign, and decentralized system for managing digital identities. Here are some key purposes and benefits of Ethereum DID smart contracts. Ethereum DID smart contracts enable individuals to have full control over their digital identities. Users can create, update, and manage their identity data without relying on centralized identity providers. DID smart contracts allow users to share only the specific information they choose, enhancing privacy and reducing the risk of identity theft and data breaches. Ethereum DID smart contracts are designed to work with various blockchain platforms and decentralized technologies, fostering interoperability in the decentralized identity ecosystem. DIDs provide a trust framework for verifying the authenticity of individuals and entities, reducing the need for intermediaries and centralized identity verification services. These smart contracts can be used in various applications, including access control, document signing, authentication, and more, making it a versatile solution for various decentralized identity use cases. Leveraging Ethereum's blockchain infrastructure ensures data immutability, transparency, and resilience, making Ethereum DID smart contracts suitable for applications requiring a high level of security. By distributing identity management across the Ethereum network, Ethereum DID smart contracts reduce the risk of a single point of failure that centralized identity providers can pose.

## 2.EXISTING PROBLEM

### Existing Problem

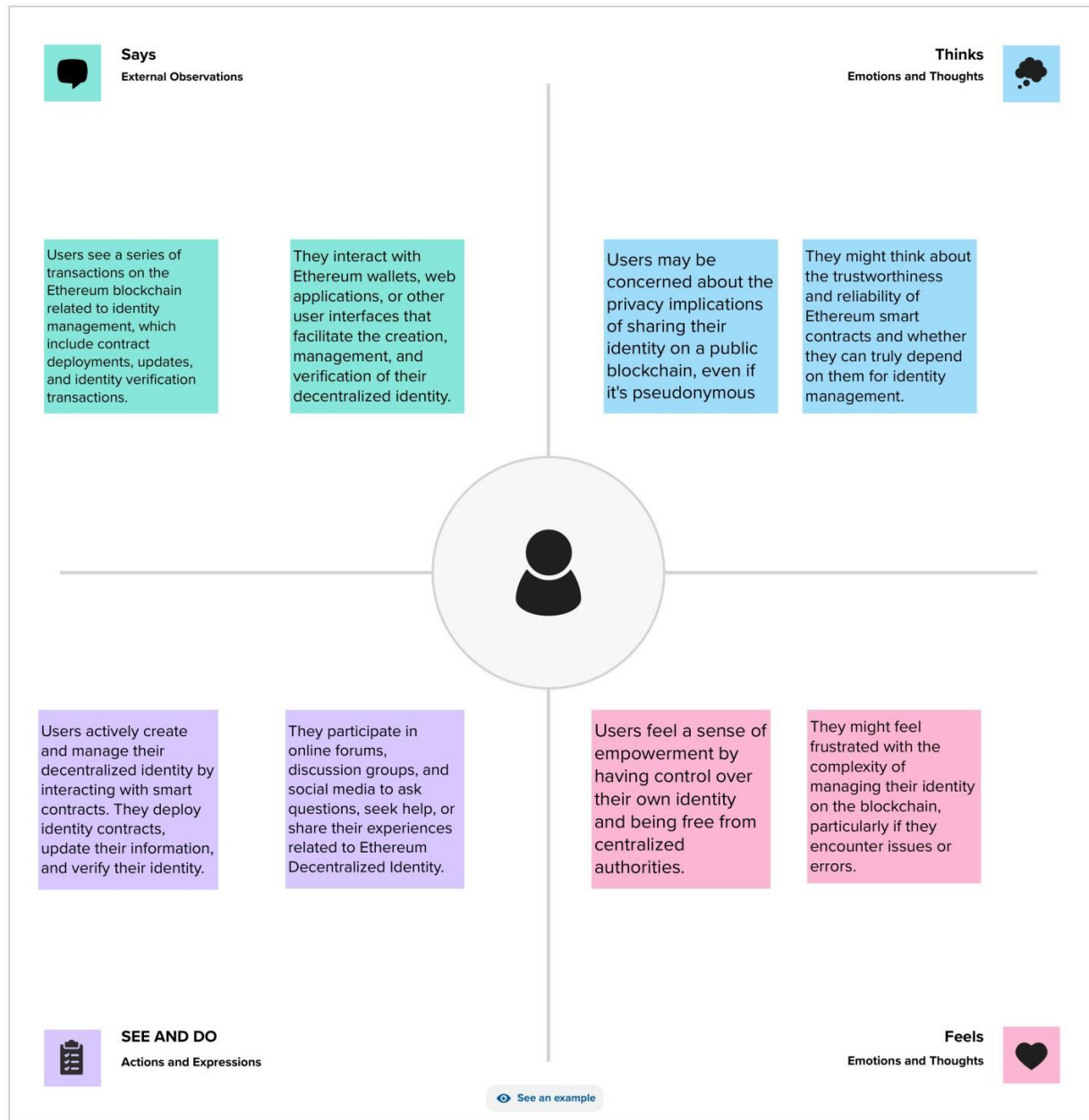
Ethereum decentralized identity (DID) smart contract space is the issue of scalability. Ethereum, while a pioneering blockchain platform, has faced challenges in terms of scalability due to its consensus mechanism and the limited transaction processing capacity of its network. Decentralized identity systems on Ethereum often rely on smart contracts to manage and control user identities and associated data. As more users and applications adopt decentralized identity solutions, the demand for smart contract execution increases, which can lead to several issues. Executing smart contracts on Ethereum requires users to pay gas fees. As the network becomes congested, gas fees can become prohibitively expensive, making it less feasible for individuals and organizations to manage their identities on the platform. Ethereum's block confirmation times can be relatively slow, especially during periods of high demand. This results in delays in executing smart contracts, which can impact the user experience and the real-time nature of identity verification. Ethereum's throughput is limited, meaning it can handle only a certain number of transactions per second. This limitation becomes a bottleneck as more identity transactions are conducted on the network. To address these issues, Ethereum has been working on scaling solutions such as Ethereum 2.0, which is expected to introduce a more efficient consensus mechanism and increased network capacity. However, transitioning to these solutions takes time and introduces complexities for decentralized identity providers that are already using Ethereum's smart contracts. Decentralized identity systems often aim to work across various blockchains and ecosystems. Ensuring interoperability and data consistency between different blockchain networks is a challenge that complicates the implementation of decentralized identity solutions.

### Problem Statement Definition

Decentralized Identity (DID) systems offer a promising solution for providing secure, user-centric, and privacy-preserving digital identity management. These systems enable individuals to have control over their digital identities and personal data while reducing reliance on centralized identity providers. However, the current landscape of DID management is fragmented and lacks a unified standard for interoperability and user-friendly management. The problem is to establish a standardized and secure Ethereum-based Decentralized Identity Smart Contract that facilitates the creation, management, and verification of DIDs while ensuring compatibility with various blockchain and web-based applications. The Ethereum Decentralized Identity Smart Contract is a blockchain-based solution designed to address the fragmented and non-interoperable nature of decentralized identity management. This smart contract is built on the Ethereum blockchain and provides a standardized framework for creating, managing, and verifying Decentralized Identifiers (DIDs) and associated verifiable credentials. The smart contract allows users to create unique DIDs associated with their Ethereum addresses. These DIDs are designed to be self-sovereign, enabling users to have full control over their identity. Users can issue, revoke, and manage verifiable credentials linked to their DIDs. These credentials can include personal information, attributes, or claims, and are cryptographically signed for authenticity. The smart contract adheres to W3C DID and Verifiable Credential standards, ensuring interoperability with a wide range of decentralized identity solutions and web-based applications. Users have fine-grained control over the information shared through their DIDs and associated credentials. They can choose what data to disclose and to whom, enhancing privacy and security. The Ethereum Decentralized Identity Smart Contract aims to provide a robust, standardized, and secure foundation for decentralized identity management, fostering user empowerment, privacy, and interoperability across a wide range of applications and services while leveraging the Ethereum network's security and decentralization features.

### 3.IDEATION AND PROPOSED SOLUTION

#### Empathy Map Canvas



### 3.2 Ideation and Brainstroming






#### After you collaborate

A brainstorm like this typically results in a handful of promising ideas that you can carry forward and act upon.

#### Quick add-ons

- A Cluster related ideas**  
Look for patterns or similarities in the standout ideas. Could any be combined together to form a stronger concept? Cluster similar ideas and label each cluster with a theme.
- B Vote on the most promising ideas**  
Narrow your focus to only the strongest few ideas by holding a **Voting Session**. Give each person 2 votes

#### Keep moving forward

- **2x2 Prioritization matrix**  
Build shared understanding and make collective decisions for moving ideas forward.  
[Open the template →](#)
- **Storyboarding**  
Show existing and/or future consumer experiences through the act of sketching.  
[Open the template →](#)
- **Pre-mortem**  
Harness the collective experience and wisdom of the team, before the project even starts.  
[Open the template →](#)

[Share template feedback](#)



#### 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 →](#)



#### 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](#)

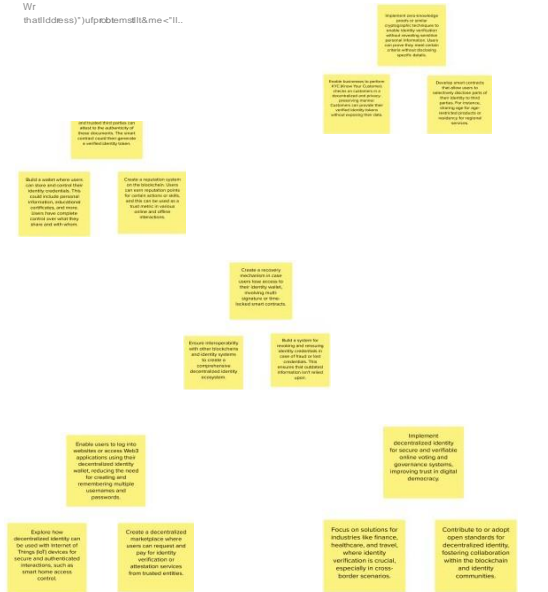
#### Ethereum Decentralised Identity Smart Contract

Develop a decentralized identity (DID) smart contract on the Ethereum blockchain to address the growing concerns of online identity theft and the lack of user control over personal information. The smart contract should provide a secure and self-sovereign identity management system that enables users to have full control over their personal data, allow them to selectively share their information with trusted parties, while maintaining privacy and security in a decentralized and trustless environment. The solution should also be interoperable with existing standards and compatible with various Ethereum-based applications providing a robust and user-centric identity solution for the decentralized web.

it down any ideas that come to mind

## Brästorm

Wr  
thatIddress)\*ufproblemsIt&me<"ll..

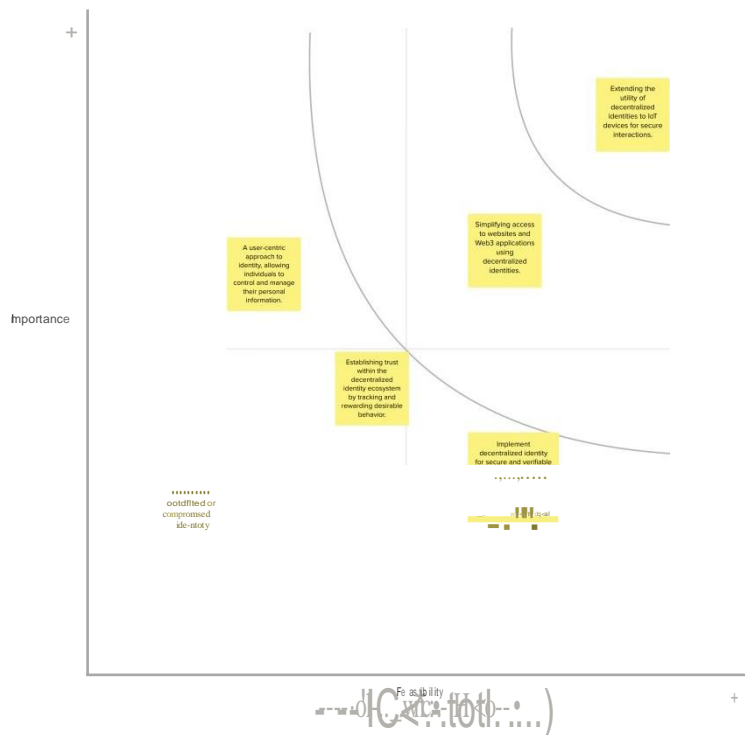


Take turns sharing your ideas while clustering similar or related notes as you go. Once 111



### Group Ideas

1) If given the following notes, try and see if you find it hard to put it up into smaller sub-groups



Quick add-ons

On the other hand, the fact that the model is not a perfect fit to the data is a limitation. The model is based on a number of assumptions, such as the assumption that the system is in a steady state, which may not be valid in all cases. Additionally, the model does not account for the effects of external factors, such as changes in the environment or the presence of other species, which could also influence the results.

[!] Exportthemur...  
ExPO"IIICQP"l oft n, -IMePNGOtPOFtoettaott>to  
emails.IncludeWI lodes. Of SIIYeWI)"UU'drive

Keep moving forward

Stret-stY  
Deofnet tsollroewkleeOf  
Sir&t@9Y.  
thetem Nte-+

 Customer experience journey INP  
Indemancly/SIOMer nMds, motNitions, lild  
obstacles for an experience.

 lengths...\_oppor1.1nids&...\_  
ldotlt W"erogtlll.-&knti opportunit>H.  
andth-eats (SWOT)ioMvelop-pt.n.  
[Open the template →](#)

[Share template feedback](#)

## 4. REQUIREMENT ANALYSIS

### Functional Requirement

**Identity Creation:** The smart contract should allow users to create a new decentralized identity by generating a unique DID and associating it with their Ethereum address.

**DID Resolution:** The smart contract should provide a way to resolve DIDs to their associated Ethereum addresses.

**Authentication:** Users should be able to prove ownership of their decentralized identity by signing messages or transactions using their private key.

**DID Methods:** Support for various DID methods such as "ethr" (Ethereum) or others, allowing users to choose the method that best suits their needs.

**DID Document Management:** Users should be able to update their DID documents to include information such as public keys, service endpoints, and other relevant metadata.

**Revocation:** The ability to revoke and recover control of a DID in case of compromise or loss of keys.

**Access Control:** Implement access control mechanisms that allow users to specify who can update their DID document or perform other actions related to their identity.

**Privacy:** Ensure that sensitive information within the DID document is appropriately protected and accessible only to authorized parties.

**Interoperability:** Comply with emerging DID standards (W3C DID specification) and ensure compatibility with other DID systems and implementations.

**Event Logging:** Record key events and changes to the DID smart contract in an immutable event log for transparency and auditing.

**Recovery Mechanism:** Implement a secure method for users to recover their decentralized identity in case they lose access to their private key or wallet.

**Compatibility with DID Methods:** Ensure compatibility with existing and emerging DID methods and standards to support a wide range of use cases.

**Gas Efficiency:** Optimize gas consumption to minimize transaction costs for users when interacting with the smart contract.

**Metadata and Context:** Allow for the inclusion of additional context and metadata in the DID document for specific use cases, such as organizational affiliations or document notarization.

**Decentralization and Security:** Implement mechanisms to ensure the security and decentralization of the DID system, avoiding single points of failure.

## 4.1 Non Functional Requirements

**Security:**The smart contract must be immutable once deployed to prevent unauthorized changes.The contract should provide transparency and auditability for all identity-related transactions and changes.Ensure robust methods for verifying users' identity and authorization to access or modify their identity data.

**Scalability and Privacy:** The contract should support a high number of identity transactions per second to accommodate a growing user base.Optimize gas consumption to minimize transaction costs for users. Personal data should be stored securely and encrypted to protect user privacy. Implement privacy-preserving techniques like zero-knowledge proofs to enable selective disclosure of identity information.

**Availability:**Ensure high availability by deploying the smart contract on multiple Ethereum nodes or networks to prevent downtime.Disaster Recovery have a plan in place for disaster recovery to minimize service interruptions.

**Interoperability:**Ensure that the smart contract can interact with other Ethereum-based decentralized applications (dApps) and identity solutions.Standardization follow industry standards (e.g., DID, Verifiable Credentials) for better interoperability with other identity systems.

**Compliance:** Ensure the smart contract complies with relevant data protection and identity verification regulations (e.g., GDPR, KYC/AML).Audit Trails that maintain comprehensive audit logs to demonstrate compliance with legal requirements.

**Performance:**Response time to specify acceptable response times for identity verification and management operations.Concurrent Users to define the maximum number of concurrent users the smart contract should support without significant performance degradation.

**User Experience:**Usability to ensure that users can easily interact with the smart contract through user-friendly interfaces, such as dApps or wallets.Documentation that Provide clear and comprehensive documentation for developers and end-users.

**Economic Sustainability:** Analyze and manage the long-term cost of running the smart contract on the Ethereum network.Sustainability Models to explore sustainable funding models, like token-based incentives or subscription-based services.

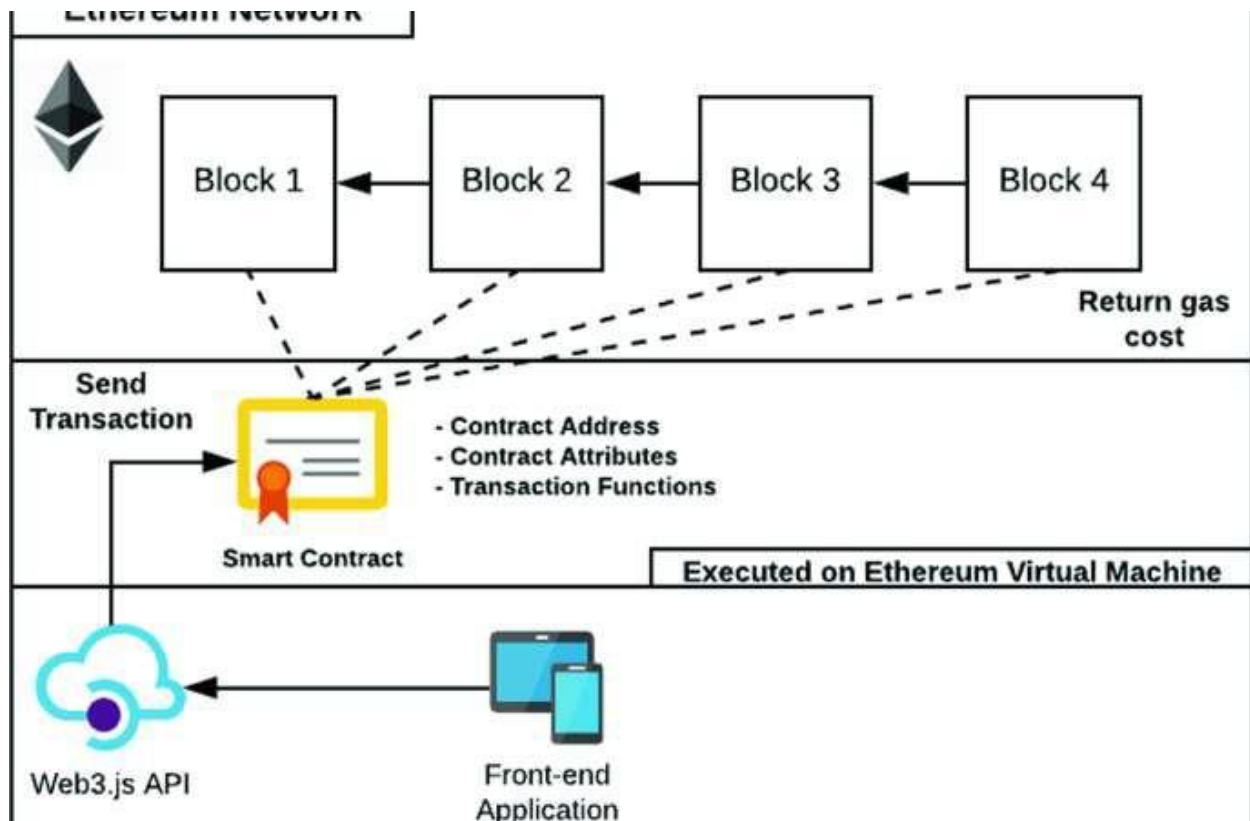
**Consensus Mechanism:** Consider the Ethereum network version (e.g., Ethereum 2.0, Ethereum Classic) and consensus mechanism (e.g., Proof of Stake) that best align with the project's goals.

**Monitoring and Alerting:** Implement continuous monitoring and alerting systems to promptly detect and respond to security incidents or performance issues.

**Network Reliability:** Ensure reliable access to Ethereum nodes to maintain smooth operation of the smart contract. Clarify data ownership and user rights concerning their decentralized identities. Establish guidelines for ethical use of the identity system and prevent misuse..

## 5.PROJECT DESIGN

### Data Flow Daigram and User Stories



#### Story 1

As a user, I want to create a decentralized identity on the Ethereum blockchain so that I can have full control over my personal information and digital assets. want to link my Ethereum wallet address to my decentralized identity so that I can use it for authentication and authorization purposes. I want to store my personal information (such as name, email, and other relevant details) securely within my decentralized identity smart contract.

#### Story 2

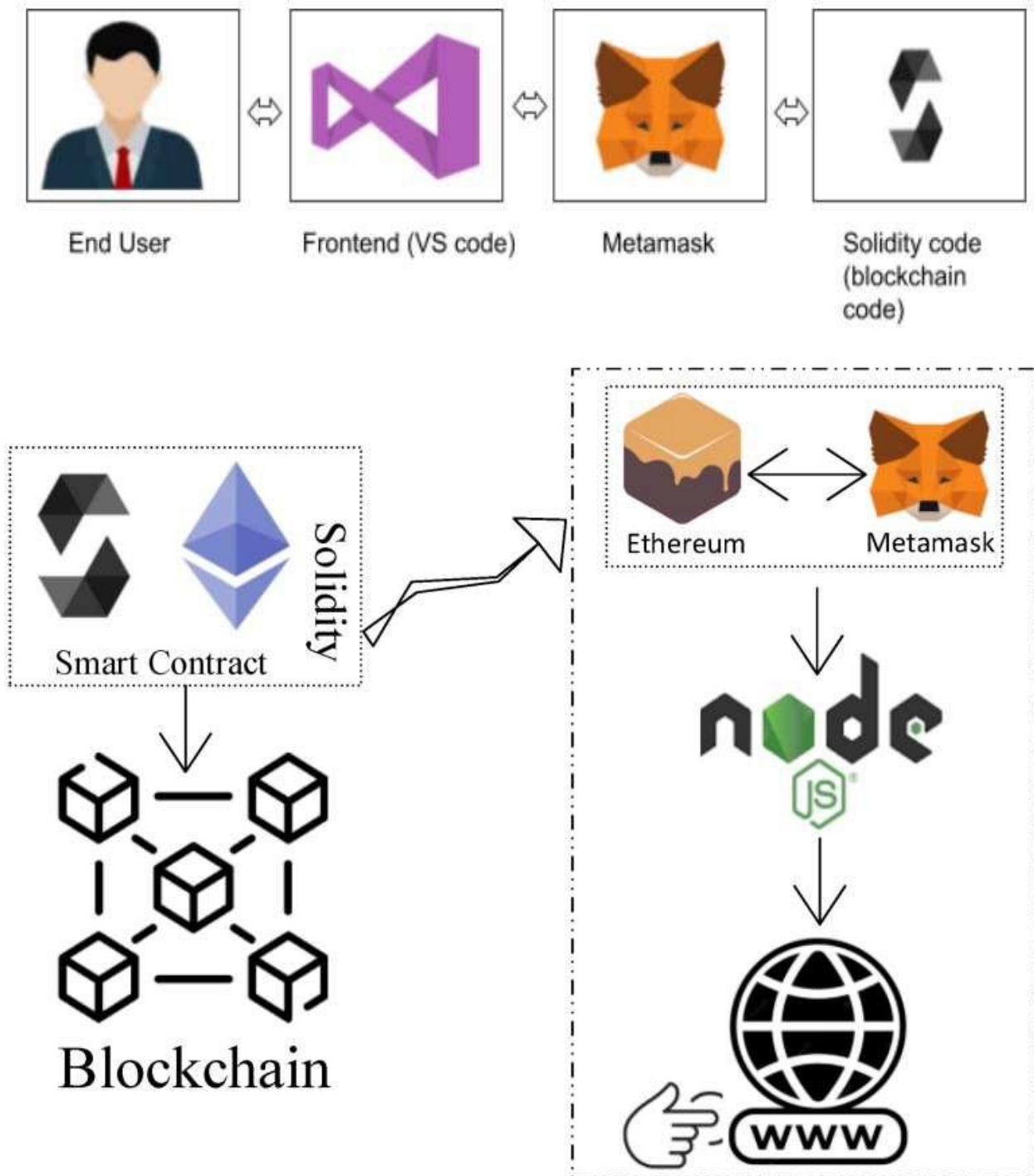
As a user, I want to share specific elements of my decentralized identity, such as my email address, with a third party in a secure and privacy-preserving manner. As a user, I want to have the ability to update and revoke access to my personal information from third parties I've shared it with through the decentralized identity smart contract. As a service provider, I want to verify the authenticity of a user's decentralized identity to ensure the user is who they claim to be.

#### Story 3

As a developer, I want to easily integrate the Ethereum DID smart contract into my applications and services for secure user authentication and data access.



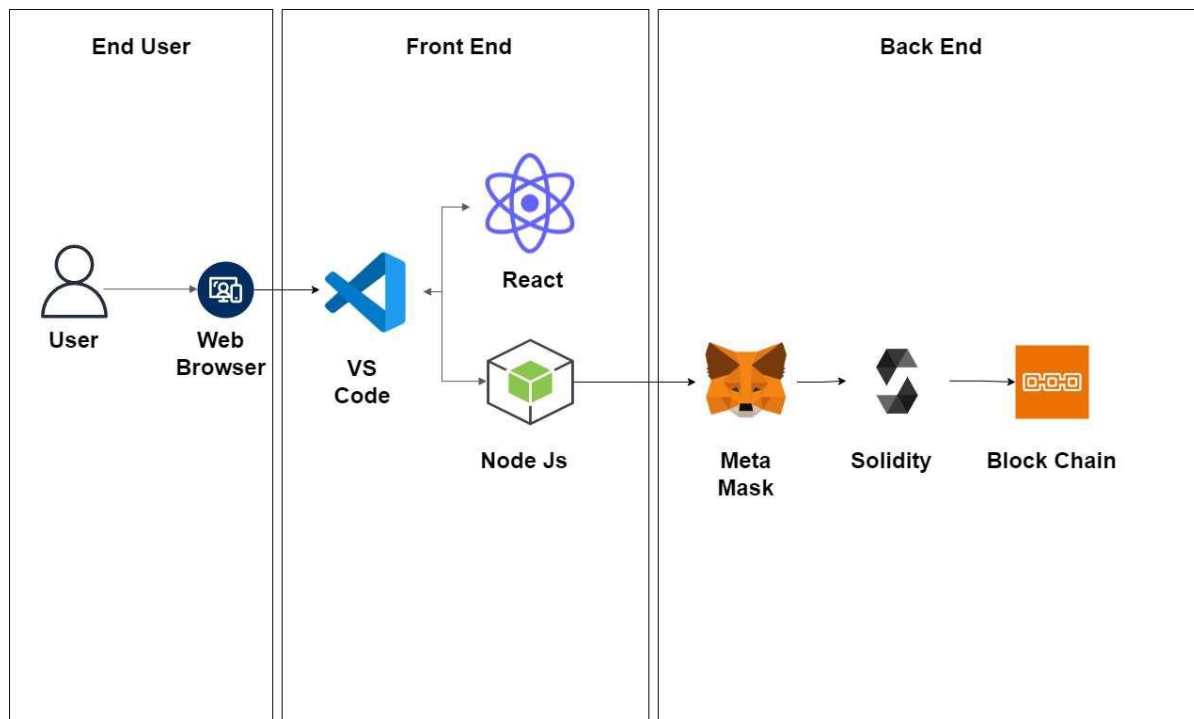
## Solution Architecture



INTERACTION BETWEEN WEB AND THE CONTRACT

## 6.PROJECT PLANNING & SCHEDULING

### Technical Architecture



### GENERAL ARCHITECTURE

#### 6.2 Sprint Planning and Estimation

Sprint planning and estimation for Ethereum Decentralized Identity (DID) smart contracts, like any software development project, involve breaking down tasks, setting objectives, and estimating the time and effort required to complete them. Here's a high-level outline of how you might plan and estimate work for a sprint focused on Ethereum Decentralized Identity:

##### Sprint Planning:

**Define Sprint Goals:** Clearly define the objectives of the sprint. This could be implementing specific features, improving existing ones, or fixing issues related to the Ethereum DID smart contract.

**Task Identification:** Create a list of tasks required to achieve the sprint goals. Tasks may include writing and testing smart contract code, updating documentation, and testing.

**Prioritization:** Prioritize tasks based on their importance and dependencies. Ensure critical features are completed first.

**Assign Ownership:** Assign tasks to team members based on their expertise and availability. Make sure each task has a responsible owner.

**Timeframe:** Determine the sprint duration (e.g., 2 weeks) and schedule regular stand-up meetings to track progress.

## Sprint Delivering Schedule:

### Week 1: Project Initiation and Planning

- ▮ Define project objectives and scope.
- ▮ Gather project requirements and conduct initial research.
- ▮ Set up the development environment.
- ▮ Create a high-level project plan.

### Week 2: Smart Contract Architecture

- ▮ Design the architecture of the Decentralized Identity Smart Contract.
- ▮ Define the data structures and functions.
- ▮ Establish the contract's ownership and access control mechanisms.

### Week 3: Decentralized Identifier and User Interface Development

- ▮ Integrate a DID resolver for Ethereum.
- ▮ Test DID resolution to fetch user information.
- ▮ Handle DID document updates and revocation.
- ▮ Develop a user-friendly frontend for interacting with the smart contract.
- ▮ Enable users to create, manage, and verify their decentralized identities.
- ▮ Ensure a seamless user experience.

## 7. CODING AND SOLUTING

### Feature1

#### Smart Contract(Solidity)

// 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
    );
}
}

```

This Solidity smart contract, named "Identification," seems to be a basic identity registration system on the Ethereum blockchain. Here are some features you can consider adding or enhancing to make this contract more functional and secure. Introduce role-based access control to allow for different levels of permissions (e.g., admin, manager, user). Create a mechanism for administrators to manage and modify user identities. Add functions to allow users to update their identity details or delete their identity when needed. Implement a Know Your Customer (KYC) process for identity verification. Integrate with an oracle or external service for identity verification, such as government-issued IDs or documents. Enhance data privacy by encrypting sensitive information, such as email addresses or contact addresses. Implement encryption and decryption mechanisms for stored data, allowing only authorized parties to access it. Additionally, security should always be a top priority when dealing with identity data and user information on the blockchain.

## 7.2Feature 2

Front end(Java Script)

```
import React, { useState } from "react";

import { Button, Container, Row, Col } from 'react-bootstrap';

import './../node_modules/bootstrap/dist/css/bootstrap.min.css';

import { contract } from "../connector";

function Home() {

  const [Id, setId] = useState("");

  const [name, setName] = useState("");

  const [email, setEmail] = useState("");

  const [addr, setAddr] = useState("");

  const [mAddr, setmAddr] = useState("");

  const [data, setdata] = useState("");

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


  const handleId = (e) => {

    setId(e.target.value)

  }

  const handleName = (e) => {

    setName(e.target.value)

  }

  const handleEmail = (e) => {
```

```
    setEmail(e.target.value)
  }
```

```
const handleAddr = (e) => {
  setAddr(e.target.value)
}
```

```
const handleRegisterIdentity = async () => {
  try {
    let tx = await contract.registerIdentity(Id.toString(), name, email, addr)
```

```
    let wait = await tx.wait()
    alert(wait.transactionHash)
    console.log(wait);
  } catch (error) {
    alert(error)
  }
}
```

```
const handleMetamaskAddr = (e) => {
  setmAddr(e.target.value)
}
```

```
const handleAddrDetails = async () => {
  try {
    let tx = await contract.getIdentityDetails(mAddr)

    let arr = []
```



```
tx.map(e => arr.push(e))

setdata(arr)

//alert(tx)

console.log(tx);

} catch (error) {

  alert(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" }}>Identity On 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={handleId} type="number"
placeholder="Enter Identity ID" value={Id} /> <br />
```

```
<input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleName} type="string"
placeholder="Enter Name" value={name} /> <br />
```

```
<input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleEmail} type="string"
placeholder="Enter Email" value={email} /><br />
```

```
<input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleAddr} type="string"
placeholder="Enter Address" value={addr} /><br />
```

```
<Button onClick={handleRegisterIdentity} style={{ marginTop: "10px" }} variant="primary">Register
Identity</Button>
```

```
</div>
```

```
</Col>
```

```
<Col>
```

```
<div>
```

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

```
<Button onClick={handleAddrDetails} style={{ marginTop: "10px" }} variant="primary">Get Identity
Details</Button>
```

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

```
}) : <p></p>}
```

```
</div>
```

```
</Col>
```

```
</Row>
```

```
</Container>
```

```
</div>
```

```
)
```

```
}
```

```
export default Home;
```

Contract ABI (Application Binary Interface):

The abi variable holds the ABI of an Ethereum smart contract. ABIs are essential for encoding and decoding function calls and data when interacting with the Ethereum blockchain.

MetaMask Check:

The code first checks whether the MetaMask wallet extension is installed in the user's browser. If MetaMask is not detected, it displays an alert notifying the user that MetaMask is not found and provides a link to download it.

Ethers.js Configuration:

It imports the ethers library, which is a popular library for Ethereum development. It creates a provider using Web3Provider, which connects to the user's MetaMask wallet and provides access to Ethereum. It creates a signer to interact with the Ethereum blockchain on behalf of the user. It defines an Ethereum contract address and sets up the contract object using ethers.Contract, allowing the JavaScript code to

interact with the contract's functions. In summary, this code is used for interacting with an Ethereum smart contract through MetaMask and ethers.js. It configures the necessary Ethereum provider and signer for communication with the blockchain and sets up a contract object for executing functions and fetching data from the specified contract address using the provided ABI.

## 8. PERFORMANCE TESTING

### Performance Matrix

A performance matrix for an Ethereum Decentralized Identity (DID) smart contract would assess various aspects of the contract's functionality and performance. Here is a sample performance matrix for such a smart contract.

#### Scalability:

- ▮ Number of DIDs supported concurrently.
- ▮ Transactions per second (TPS) for creating, updating, and resolving DIDs.
- ▮ Gas consumption per operation.

#### Security:

- ▮ Smart contract audit results.
- ▮ Vulnerability assessments (e.g., OWASP Top Ten).
- ▮ Compliance with Ethereum best practices and standards.

#### Privacy and Confidentiality:

- ▮ Protection of user data.
- ▮ Encryption and decryption efficiency.
- ▮ Privacy-preserving techniques employed (e.g., zero-knowledge proofs).

#### Interoperability:

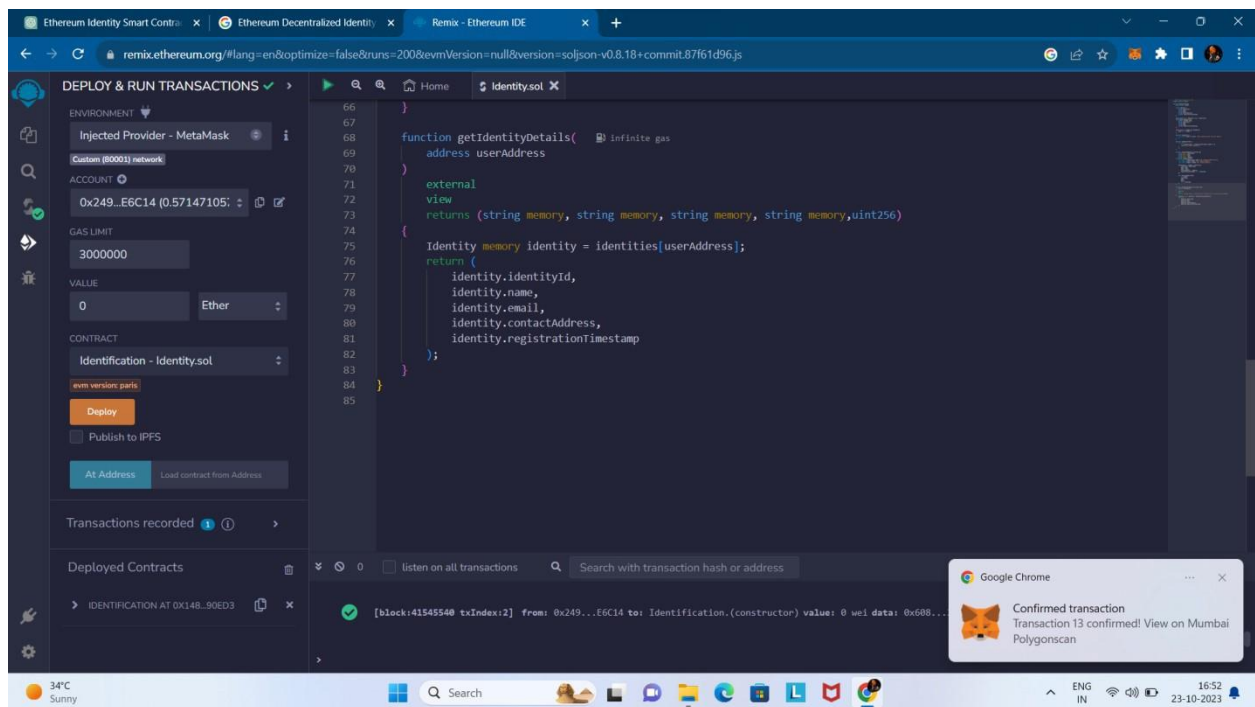
- ▮ Compatibility with popular identity standards (e.g., W3C DID, Verifiable Credentials).
- ▮ Integration with other Ethereum-based smart contracts and platforms.
- ▮ Support for cross-platform compatibility.

#### Ease of Use:

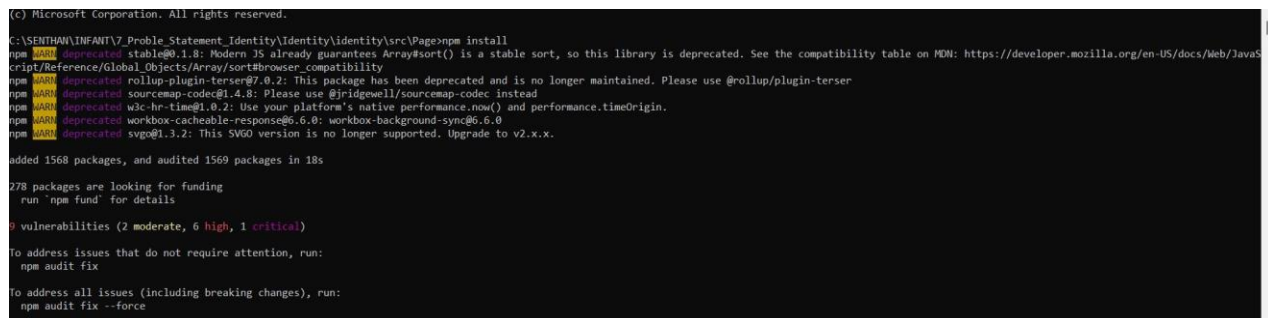
- ▮ User-friendly API and documentation.
- ▮ Accessibility and usability for non-technical users.
- ▮ Availability of SDKs and developer resources.

## 9.RESULTS

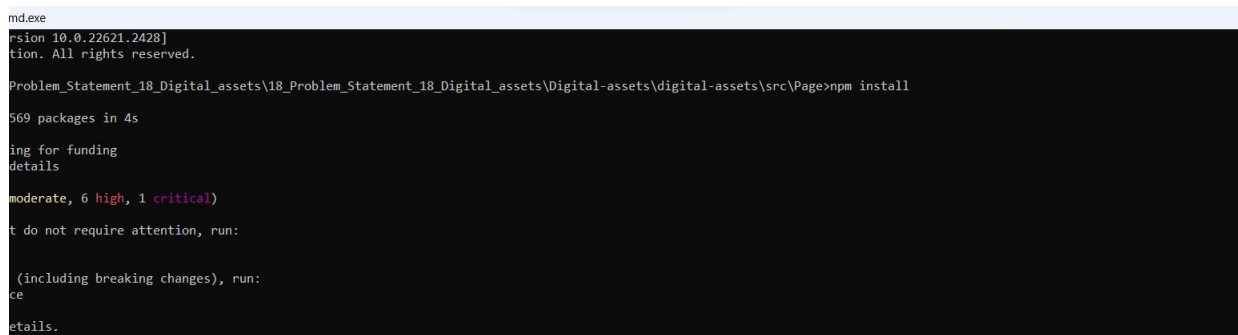
### 9.1 OUTPUT SCREENSHOTS



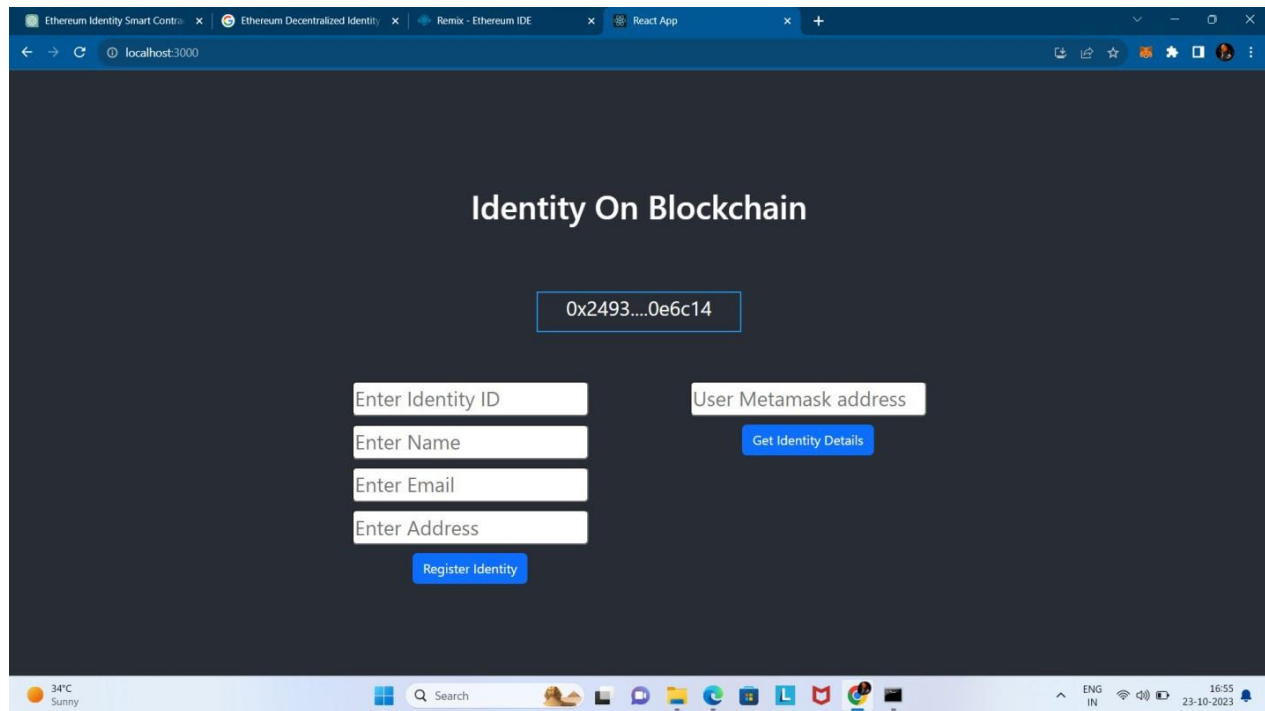
### CREATING A SMART CONTACT



### INSTALLING DEPENDENCIES 1



### INSTALLING DEPENDENCIES 2



WEB PAGE OUTPUT

## 10.ADVANTAGES AND DISADVANTAGES

### ADVANTAGES

Users have full control over their identity and personal data. They can choose what information to share, reducing the risk of data breaches and unauthorized access. Data stored on the Ethereum blockchain is highly secure and tamper-resistant, ensuring the integrity of identity information. Ethereum DID smart contracts can be used across various applications and services, making it easier for users to manage their identity across different platforms. Ethereum DID smart contracts can adhere to standards such as DID and Verifiable Credentials, enhancing interoperability. There is no central authority or intermediary responsible for managing identities. Users are not reliant on a single entity for identity verification. It's harder for any one entity to censor or control an individual's identity. Ethereum DID smart contracts can simplify the authentication process. Users can easily prove their identity to access services or resources without repetitive registrations or logins. Reduces the need for multiple, redundant identity verification processes, potentially reducing costs for both users and service providers. Smart contracts, being self-executing and transparent, enhance trust in identity verification processes. Users can provide verifiable claims and credentials, making it easier for others to trust their identity claims. Ethereum DIDs are not tied to any specific geographic region, enabling individuals to access services and establish their identity globally. This technology can help individuals without traditional forms of identification, such as refugees or those without official government IDs, participate in the digital economy. The Ethereum blockchain maintains an immutable history of identity-related transactions, which can be useful for auditing and dispute resolution. Ethereum DIDs put users in control of their identity, empowering them to manage and share their information as they see fit. The security features of blockchain and smart contracts can significantly reduce the risk of identity theft and fraudulent activities.

## DISADVANTAGES

Ethereum, like many other blockchain platforms, can face scalability issues, especially when handling a large number of transactions and smart contracts. This can lead to slow confirmation times and higher transaction costs. These limitations can impact the efficiency and responsiveness of decentralized identity systems. Executing smart contracts on the Ethereum network requires the payment of gas fees. These fees can vary widely depending on network congestion and the complexity of the contract. For decentralized identity systems that frequently interact with the blockchain, these gas costs can become a significant financial burden. Ethereum, like many other blockchain platforms, relies on proof-of-work (PoW) consensus mechanisms, which are energy-intensive. The environmental impact of Ethereum and the high energy consumption associated with PoW blockchains can be a concern, especially in an era where sustainability is a priority. Achieving full interoperability between different decentralized identity systems and across various blockchains is a complex task. Smart contracts on Ethereum might not seamlessly integrate with other blockchain-based identity solutions, leading to potential compatibility issues. Managing cryptographic keys is a fundamental part of decentralized identity systems. While users have more control over their keys, they also bear the responsibility of securely managing them. Loss of keys or compromised key security can result in a complete loss of access to one's identity, which can be a significant drawback. Blockchain-based decentralized identity systems, including those on Ethereum, are still relatively new and not widely adopted. Usability challenges and a lack of user-friendly interfaces can deter mainstream users from fully embracing these solutions. The regulatory landscape for blockchain-based identity solutions is still evolving. Decentralized identity systems might encounter legal and compliance challenges as governments and authorities work to establish guidelines and regulations in this area.

## 11 CONCLUSION

In conclusion, Ethereum's Decentralized Identity smart contracts represent a significant step towards enhancing privacy, security, and user control in the digital realm. These smart contracts enable individuals to take ownership of their identity data, reducing the need for centralized identity providers and mitigating the risk of data breaches. Through blockchain technology, users can manage their identity attributes while maintaining anonymity and control. As the field of decentralized identity continues to evolve, Ethereum's smart contracts are a promising tool for reshaping how we interact online, placing individuals in charge of their personal information and fostering a more secure and user-centric digital ecosystem.

## 12. FUTURESCOPE

The future for Ethereum Decentralized Identity (DID) smart contracts is promising, as the technology continues to evolve and mature. Ethereum is one of the leading platforms for building decentralized identity solutions, and it's likely to play a significant role in shaping the future of digital identity management. Here are some potential developments and trends for Ethereum Decentralized Identity smart contracts.

**Interoperability:** Ethereum DID smart contracts may become more interoperable with other blockchains and decentralized identity platforms. This would allow for seamless identity management across different ecosystems, improving user experience and utility.

**Standardization:** The development of industry standards for Ethereum DID smart contracts can help establish a common framework for identity management. This will make it easier for different applications and services to integrate and trust Ethereum-based identity solutions.

**Privacy and Security:** As concerns over data privacy and security grow, Ethereum DID smart contracts may incorporate advanced cryptographic techniques and zero-knowledge proofs to enhance user privacy while maintaining security.

**Scalability:** Ethereum is working on scaling solutions like Ethereum 2.0, which should increase the network's capacity and reduce transaction costs. This would make Ethereum a more viable platform for decentralized identity applications, which can involve a high volume of interactions.

**Self-Sovereign Identity:** Ethereum DID smart contracts will continue to empower individuals to own and control their digital identities. Users will be able to manage their personal information and choose which aspects they share with different services, reducing the reliance on centralized identity providers.

**Tokenization and NFTs:** Non-fungible tokens (NFTs) and other token standards may play a role in decentralized identity. This could involve representing aspects of an individual's identity as NFTs, which can be easily transferred and verified.

**Use Cases:** Ethereum DID smart contracts can be applied to various sectors beyond just personal identity, such as access control, supply chain management, healthcare, and more. As adoption grows, the range of applications for decentralized identity will expand.

**Integration with DeFi:** Integration with decentralized finance (DeFi) platforms may allow users to leverage their decentralized identity for financial services. This could lead to easier and more secure onboarding processes for DeFi applications.

**Regulatory Compliance:** Ethereum DID smart contracts may incorporate features that help meet regulatory compliance requirements, such as identity verification for Know Your Customer (KYC) and Anti-Money Laundering (AML) purposes.

**User-Friendly Interfaces:** Improvements in user interfaces and user experiences will be essential to make Ethereum DID smart contracts accessible to a broader audience. Mobile apps and easy-to-use tools will likely become more prevalent.

**Cross-Platform Integration:** Ethereum DID smart contracts may integrate with web browsers, mobile devices, and other applications to offer a seamless and secure digital identity experience.

**Community and Ecosystem Growth:** The Ethereum community will continue to develop and innovate in the field of decentralized identity, fostering the growth of the ecosystem.

It's important to note that the development of Ethereum DID smart contracts will depend on the collaboration of developers, organizations, and regulators to ensure that these solutions meet the needs of users while addressing concerns related to privacy, security, and compliance.



### 13.APPENDIX

#### SOURCE CODE

Digitalassest.sol

// 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": "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 = "0x14845D1Dd4A964ECa48b017d3A37F4Eb38c90ed3"
```

```
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 './../node_modules/bootstrap/dist/css/bootstrap.min.css';  
import { contract } from "../connector";
```

```
function Home() {  
  const [Id, setId] = useState("");  
  const [name, setName] = useState("");  
  const [email, setEmail] = useState("");  
  const [addr, setAddr] = useState("");  
  const [mAddr, setmAddr] = useState("");  
  const [data, setdata] = useState("");
```

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

```
const handleId = (e) => {  
  setId(e.target.value)  
}
```

```
const handleName = (e) => {  
  setName(e.target.value)  
}
```

```
const handleEmail = (e) => {  
  setEmail(e.target.value)  
}
```

```
const handleAddr = (e) => {  
  setAddr(e.target.value)  
}
```

```
const handleRegisterIdentity = async () => {  
  try {  
    let tx = await contract.registerIdentity(Id.toString(), name, email, addr)  
  
    let wait = await tx.wait()  
    alert(wait.transactionHash)  
    console.log(wait);  
  } catch (error) {
```

```
    alert(error)
  }
}
```

```
const handleMetamaskAddr = (e) => {
  setmAddr(e.target.value)
}
```

```
const handleAddrDetails = async () => {
  try {
    let tx = await contract.getIdentityDetails(mAddr)
    let arr = []
    tx.map(e => arr.push(e))
    setdata(arr)
    //alert(tx)
    console.log(tx);
  } catch (error) {
    alert(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" }}>Identity On 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={handleId} type="number"
    placeholder="Enter Identity ID" value={Id} /> <br />

    <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleName} type="string"
    placeholder="Enter Name" value={name} /> <br />

```

```
    <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleEmail} type="string"
placeholder="Enter Email" value={email} /><br />
```

```
    <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleAddr} type="string"
placeholder="Enter Address" value={addr} /><br />
```

```
    <Button onClick={handleRegisterIdentity} style={{ marginTop: "10px" }} variant="primary">Register
Identity</Button>
```

```
  </div>
```

```
</Col>
```

```
<Col>
```

```
  <div>
```

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

```
    <Button onClick={handleAddrDetails} style={{ marginTop: "10px" }} variant="primary">Get Identity
Details</Button>
```

```
    {data ? data?.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;
```

GITHUB LINK : <https://github.com/vinor01/NM2023TMID06099>

DEMO VIDEO LINK :

[https://drive.google.com/drive/folders/1bXkcYHiIEUIyZn50VQ5qdnvo4Dra\\_iPC?usp=sharing](https://drive.google.com/drive/folders/1bXkcYHiIEUIyZn50VQ5qdnvo4Dra_iPC?usp=sharing)