DIGITAL ASSET MANAGEMENT ON THE ETHEREUM BLOCK CHAIN

1.INTRODUCTION

1.1 Project Overview

In an increasingly digitized world, the management and security of digital assets have become paramount. Digital assets encompass a wide range of items, including cryptocurrencies, NFTs (Non-Fungible Tokens), intellectual property, and more. To address the growing need for secure and efficient management of these assets, this project aims to develop a Digital Asset Management platform built on the Ethereum blockchain. It a robust system for securing digital assets, leveraging Ethereum's blockchain technology, which is renowned for its cryptographic security and decentralized nature.DAM Enable the management of various digital asset types, such as cryptocurrencies, NFTs, and other tokenized assets within a unified platform. It develop an intuitive and user-friendly interface for asset owners to interact with and manage their digital assets, making it accessible to both technical and non-technical users.DAM develop an intuitive and user-friendly interface for asset owners to interact with and manage their digital assets, making it accessible to both technical and non-technical users.It can Incorporate necessary compliance measures to adhere to legal and regulatory requirements in different jurisdictions, ensuring a secure and compliant environment for asset management.The block chain based management create a web applications to make asset management accessible across various devices and operating systems.It create a support system and a community around the platform, including forums, tutorials, and customer support to assist users in asset management. Explore partnerships and integration with other blockchain projects, DeFi protocols, and decentralized applications to expand the ecosystem and provide additional utility for digital assets.The main advantages in this system that can enable users to tokenize real-world assets, such as real estate or art, and facilitate fractional ownership, making it more accessible to a wider audience.

1.2 **Purpose**

Digital Asset Management (DAM) on the Ethereum blockchain serves as a sophisticated and secure system for the management and tracking of digital assets, including cryptocurrencies and non-fungible tokens (NFTs). Its primary purpose is to provide individuals and organizations with a reliable and transparent platform for the storage, transfer, and monitoring of digital assets. Here are the key purposes of DAM on the Ethereum blockchain. DAM on Ethereum ensures the secure storage and protection of digital assets through cryptographic methods and decentralized consensus mechanisms, reducing the risk of theft or unauthorized access.This system facilitates the verification of ownership for digital assets, using blockchain addresses and smart contracts to establish and confirm ownership rights, preventing fraudulent activities. Ethereum's transparent and immutable ledger ensures that all transactions and asset movements are recorded and publicly accessible. This transparency reduces the likelihood of disputes and fraud.This project on Ethereum allows for interoperability with other decentralized applications (dApps) and smart contracts, enabling assets to be utilized in various ways, such as lending, trading, and collateralization. Users can diversify their digital asset portfolios by easily managing a wide range of assets, including cryptocurrencies, NFTs, tokens, and more, all within a single platform. It enables seamless and instant transfer of digital assets to other Ethereum addresses, making it a convenient platform for trading, gifting, or lending assets to others. Ethereum's support for token standards that allows for the creation of digital assets that represent real-world assets, such as real estate, art, or stocks, thereby increasing liquidity and accessibility. The blockchain's transparency and auditability make it easier to track and verify the history of digital assets, which is crucial for regulatory compliance and asset validation.

2.EXISTING PROBLEM

2.1 Existing Problem

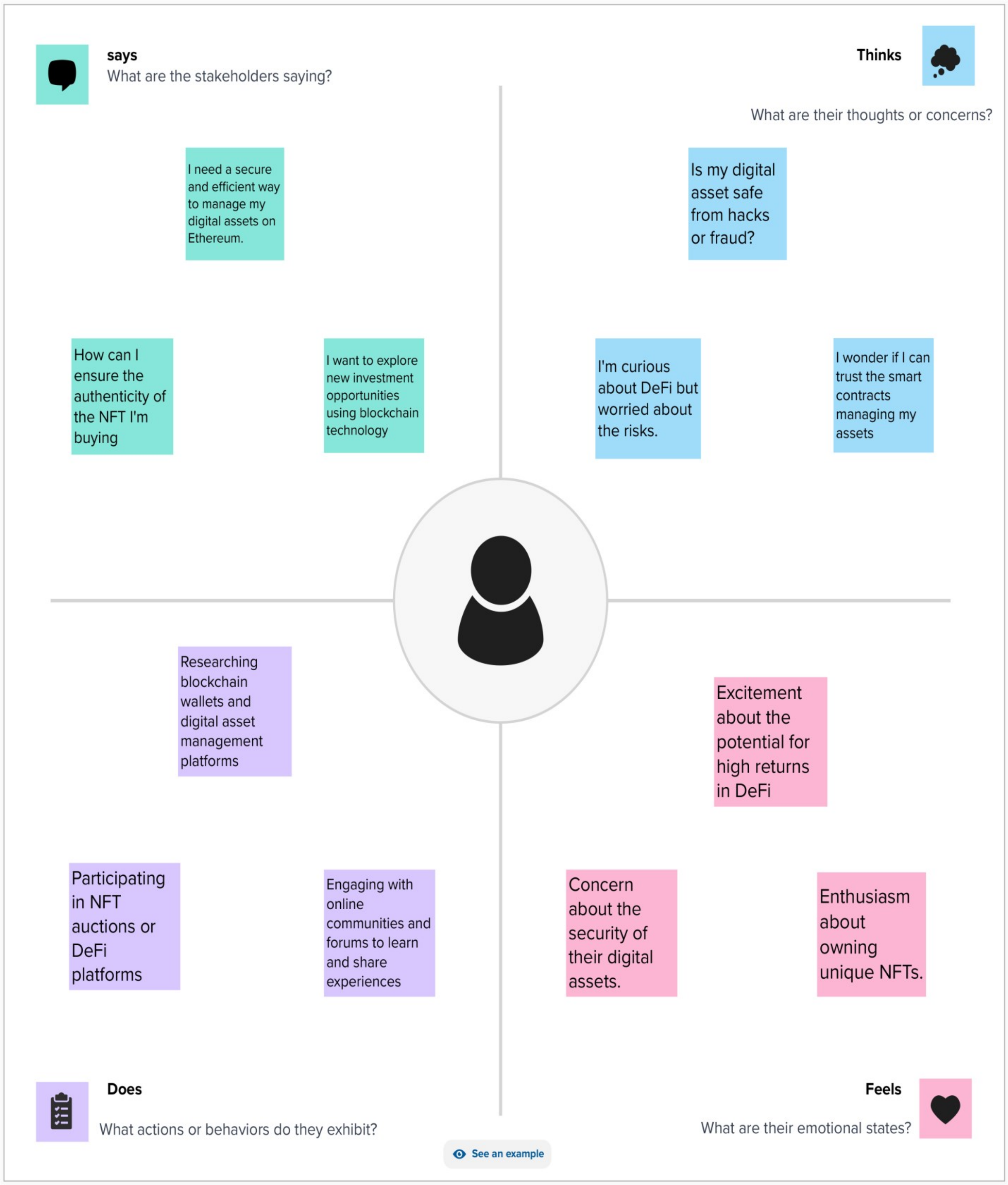
Digital Asset Management (DAM) is the challenge of Metadata Inconsistency.Metadata inconsistency occurs when the metadata associated with digital assets is not standardized or well-maintained, leading to difficulties in asset discovery, organization, and retrieval. This problem can manifest in various ways. Users may apply different tags or keywords to similar assets, making it difficult to find related or relevant content. For instance, one user might tag an image as beach,while another uses seashore. Users might provide inaccurate or incomplete metadata when uploading assets, which can lead to assets being mislabeled, making them hard to locate or identify. Assets may have filenames that are inconsistent or non-descriptive. For example, image files might have random strings of characters instead of meaningful titles. Duplicate or redundant metadata fields can confuse users and complicate the search process. For instance, having both "Author" and "Creator" fields for the same information. Without a proper governance framework, there may be no guidelines or policies in place to ensure metadata consistency across the organization.

2.2 Problem Statement Definition

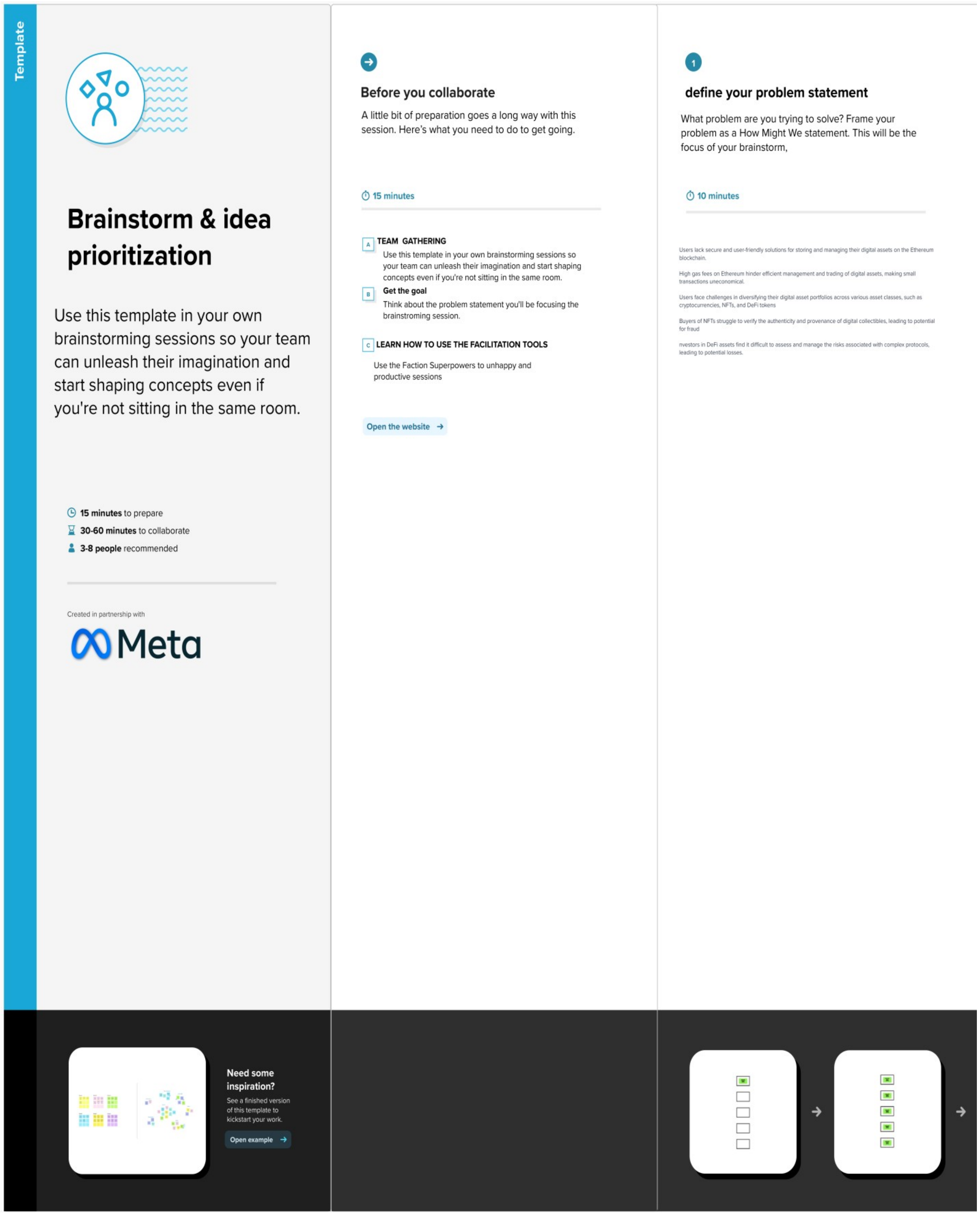
Digital Asset Management (DAM) on the Ethereum Blockchain is a complex and evolving field that aims to address the efficient and secure storage, access, and transfer of digital assets, such as tokens, non-fungible tokens (NFTs), and other blockchain-based assets. As the adoption of blockchain technology grows and the number of digital assets increases, managing these assets in a decentralized and secure manner has become a significant challenge. The problem at hand is to create a comprehensive and user-friendly Digital Asset Management system on the Ethereum blockchain that fulfills the following key requirements. DAM on the Ethereum Blockchain refers to the process of organizing, storing, tracking, and transferring digital assets in a decentralized and blockchain-based environment. It involves the use of smart contracts and decentralized applications (DApps) to facilitate the management of various types of digital assets, including cryptocurrencies, NFTs, digital documents, and more. The primary objectives of DAM on the Ethereum Blockchain.It can ensuring the secure storage and transfer of digital assets to prevent loss, theft, or unauthorized access.The blockchain system enabling compatibility and seamless transfer of assets between different platforms, wallets, and blockchains.this project Providing a transparent ledger of asset ownership and transaction history, allowing for easy auditing. It developing a user-friendly and accessible interface for users to manage their digital assets, even for those with limited technical knowledge.This system Addressing the challenges of scalability to support the growing number of digital assets and users on the Ethereum network. Implementing mechanisms for data recovery and backup to prevent data loss due to unforeseen circumstances. DAM minimizing transaction fees and storage costs associated with asset management on the Ethereum blockchain. The problem is to design and implement a DAM solution that meets these requirements while taking into account the unique characteristics and constraints of the Ethereum blockchain. Additionally, addressing the ever-evolving blockchain landscape and keeping pace with technological advancements in the field presents an ongoing challenge. Therefore, the problem statement revolves around developing an efficient, secure, and user-centric solution for Digital Asset Management on the Ethereum blockchain.

3.IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas


says
What are the stakeholders saying?
I need a secure and efficient way to manage my digital assets on Ethereum.
How can I ensure the authenticity of the NFT I'm buying
I want to explore new investment opportunities using blockchain technology
Thinks
What are their thoughts or concerns?
Is my digital asset safe from hacks or fraud?
I'm curious about DeFi but worried about the risks.
I wonder if I can trust the smart contracts managing my assets
Researching blockchain wallets and digital asset management platforms
Participating in NFT auctions or DeFi platforms
Engaging with online communities and forums to learn and share experiences
Excitement about the potential for high returns in DeFi
Concern about the security of their digital assets.
Enthusiasm about owning unique NFTs.
Does
What actions or behaviors do they exhibit?
See an example
Feels
What are their emotional states?

3.2 Ideation and Brainstroming



Template

Brainstorm & idea prioritization

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

15 minutes to prepare
30-60 minutes to collaborate
3-8 people recommended

Created in partnership with

Meta

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.

15 minutes

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

B **Get the goal**
Think about the problem statement you'll be focusing the brainstroming session.

C **LEARN HOW TO USE THE FACILITATION TOOLS**
Use the Faction Superpowers to unhappy and productive sessions

Open the website →

1 **define your problem statement**

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

10 minutes

Users lack secure and user-friendly solutions for storing and managing their digital assets on the Ethereum blockchain.

High gas fees on Ethereum hinder efficient management and trading of digital assets, making small transactions uneconomical.

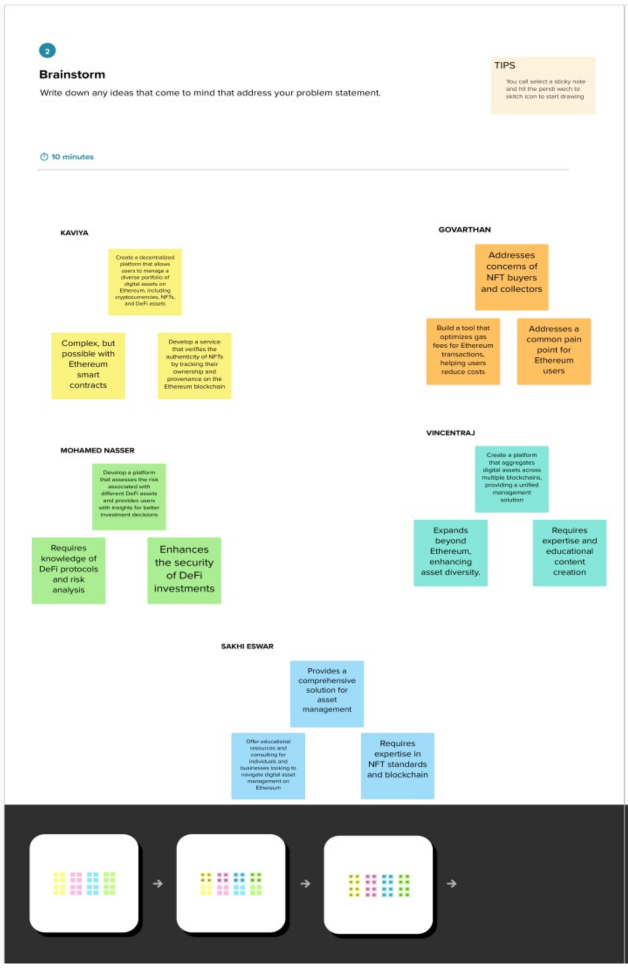
Users face challenges in diversifying their digital asset portfolios across various asset classes, such as cryptocurrencies, NFTs, and DeFi tokens

Buyers of NFTs struggle to verify the authenticity and provenance of digital collectibles, leading to potential for fraud

nvestors in DeFi assets find it difficult to assess and manage the risks associated with complex protocols, leading to potential losses.

Need some inspiration?
See a finished version of this template to kickstart your work.

Open example →



2

Brainstorm

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

TIPS

You call select a sticky note and hit the pendi wech to skitch icon to start drawing

10 minutes

KAVIYA

Create a decentralized platform that allows users to manage a diverse portfolio of digital assets on Ethereum, including cryptocurrencies, NFTs, and DeFi assets

Complex, but possible with Ethereum smart contracts

Develop a service that verifies the authenticity of NFTs by tracking their ownership and provenance on the Ethereum blockchain

GOVARTHAN

Addresses concerns of NFT buyers and collectors

Build a tool that optimizes gas fees for Ethereum transactions, helping users reduce costs

Addresses a common pain point for Ethereum users

MOHAMED NASSER

Develop a platform that assesses the risk associated with different DeFi assets and provides users with insights for better investment decisions

Requires knowledge of DeFi protocols and risk analysis

Enhances the security of DeFi investments

VINCENTRAJ

Create a platform that aggregates digital assets across multiple blockchains, providing a unified management solution

Expands beyond Ethereum, enhancing asset diversity.

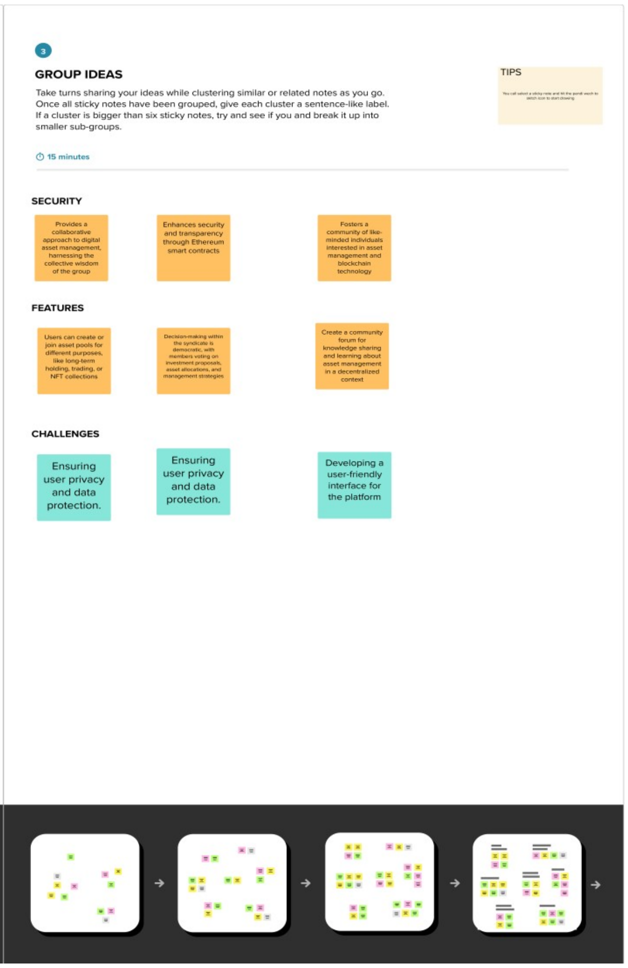
Requires expertise and educational content creation

SAKHI ESWAR

Provides a comprehensive solution for asset management

Offer educational resources and consulting for individuals and businesses looking to navigate digital asset management on Ethereum

Requires expertise in NFT standards and blockchain



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.

TIPS

You call select a sticky note and hit the pendi wech to sketch icon to start drawing

15 minutes

SECURITY

Provides a collaborative approach to digital asset management, harnessing the collective wisdom of the group

Enhances security and transparency through Ethereum smart contracts

Fosters a community of like-minded individuals interested in asset management and blockchain technology

FEATURES

Users can create or join asset pools for different purposes, like long-term holding, trading, or NFT collections

Decision-making within the syndicate is democratic, with members voting on investment proposals, asset allocations, and management strategies

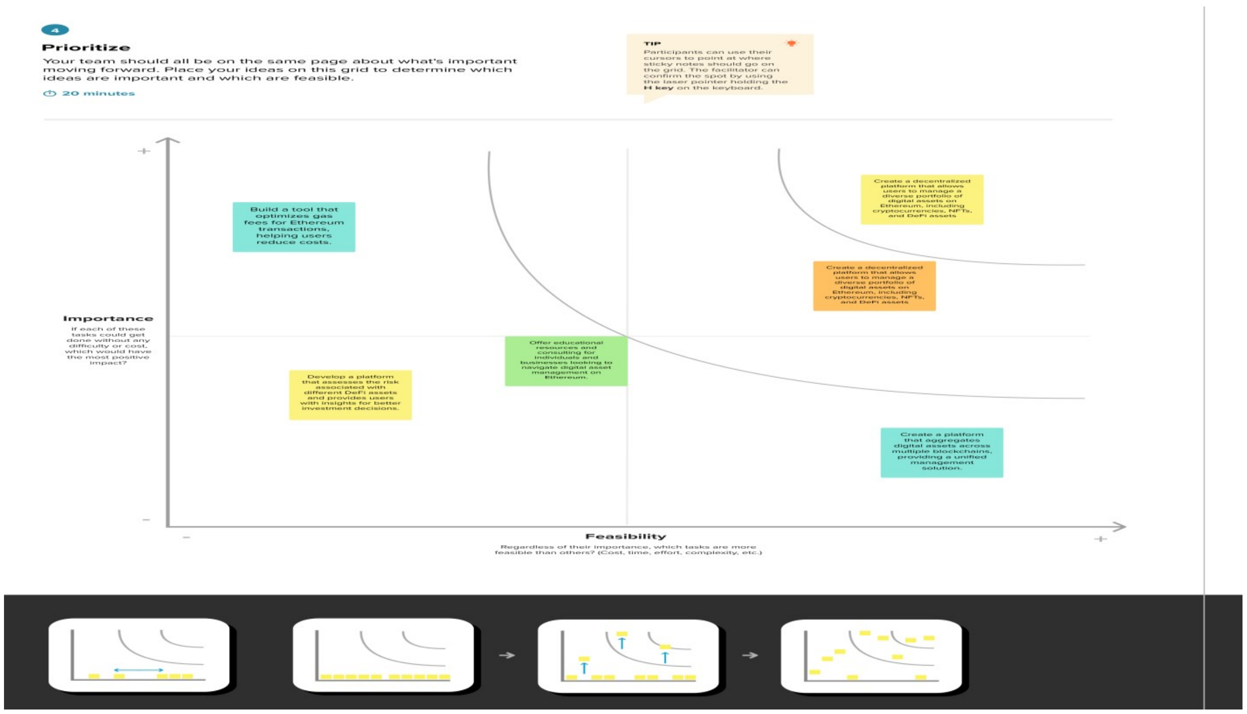
Create a community forum for knowledge sharing and learning about asset management in a decentralized context

CHALLENGES

Ensuring user privacy and data protection.

Ensuring user privacy and data protection.

Developing a user-friendly interface for the platform



4

Prioritize

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

20 minutes

TIP

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

Build a tool that optimizes gas fees for Ethereum transactions, helping users reduce costs.

Create a decentralized platform that allows users to manage a diverse portfolio of digital assets on Ethereum, including cryptocurrencies, NFTs, and DeFi assets

Create a decentralized platform that allows users to manage a diverse portfolio of digital assets on Ethereum, including cryptocurrencies, NFTs, and DeFi assets

Offer educational resources and consulting for individuals and businesses looking to navigate digital asset management on Ethereum.

Develop a platform that assesses the risk associated with different DeFi assets and provides users with insights for better investment decisions.

Create a platform that aggregates digital assets across multiple blockchains, providing a unified management solution.

+
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
−
−
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)
+

4.REQUIREMENT ANALYSIS

4.1 Functional Requirements

User Registration and Authentication:

- Users should be able to create accounts securely.
- Users must be able to log in and out of the system.

Asset Creation and Uploading:

- Users should be able to create digital assets (e.g., images, videos, documents) and upload them to the blockchain.
- Supported file formats and size limits should be specified.

Metadata Management:

- Users should be able to add, edit, and remove metadata associated with their digital assets.
- Metadata should include information such as title, description, date created, and author.

Asset Storage and Retrieval:

- The system should store digital assets securely on the Ethereum blockchain or an associated decentralized file storage system.
- Users must be able to retrieve their assets with a search and retrieval mechanism.

Asset Ownership and Permissions:

- Users should have control over the ownership of their assets.
- Permissions to view, edit, or transfer assets should be customizable.

Transfer and Sharing:

- Users should be able to transfer assets to other users.
- Sharing options, such as private or public sharing, should be available.

Transaction History:

- The system should maintain a record of all transactions, including asset transfers and updates.
- Users should be able to view their transaction history.

Smart Contracts:

- The system should utilize smart contracts for managing asset ownership, access control, and other functions.
- Smart contracts should be customizable and upgradable.

4.1 Non Functional Requirements

Security:

- Data and assets should be stored securely on the blockchain, with appropriate encryption and access control measures.
- Protection against vulnerabilities and attacks, including smart contract security, should be in place.

Scalability:

- The system should be able to handle a growing number of users and assets without compromising performance.
- Scalability solutions, such as layer 2 solutions, should be considered.

Speed and Performance:

- Asset retrieval and transaction processing should be efficient and responsive to provide a seamless user experience.
- Reliability and Availability:
- The system should be available 24/7, with minimal downtime.
- Redundancy and failover mechanisms should be in place.

Interoperability:

- The system should be designed to work with other blockchain platforms and external systems, enabling asset transfer and integration with external services.

Compliance:

- Ensure compliance with relevant legal and regulatory requirements, including copyright and intellectual property laws.

Privacy:

- Implement privacy features to protect sensitive metadata and user information.
- Comply with data protection regulations.

Cost Efficiency:

- Minimize gas fees and other transaction costs for users, making the system economically viable.

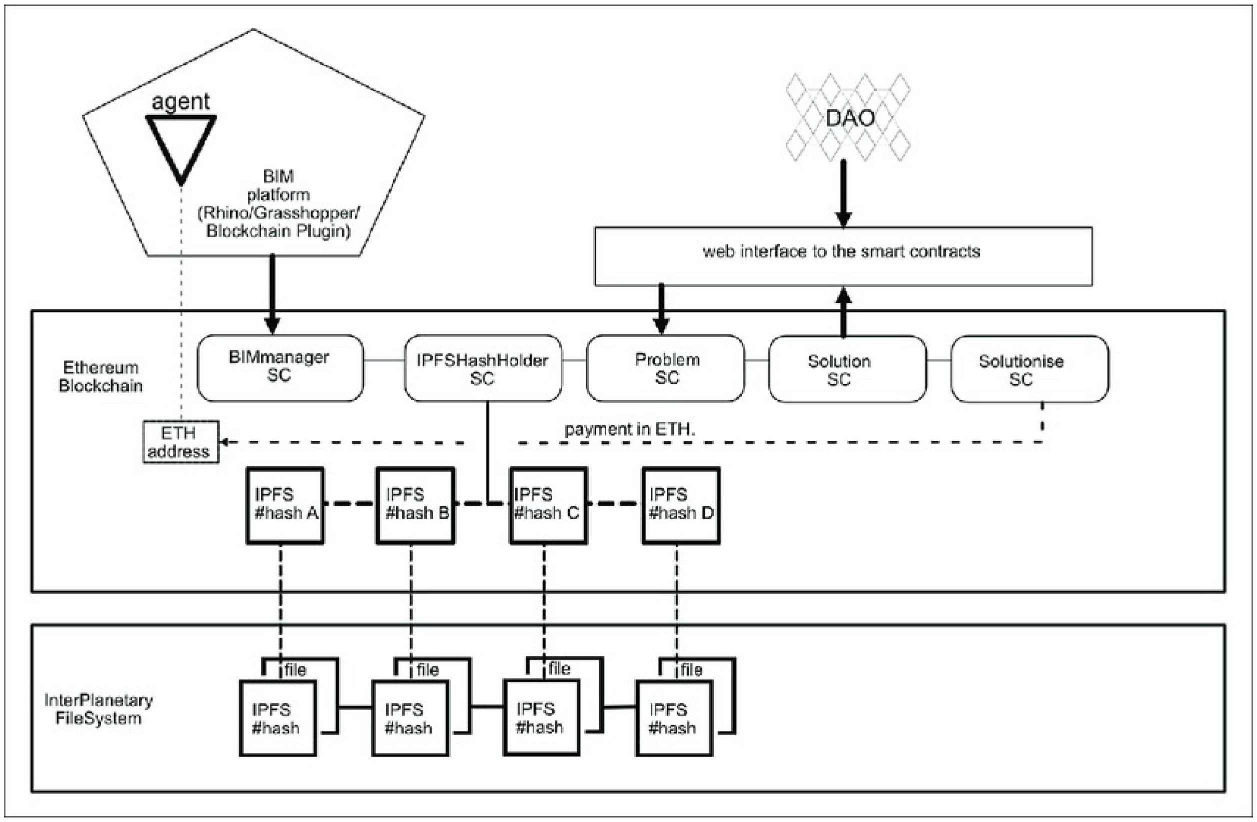
Monitoring and Analytics:

- Implement monitoring tools to track system performance and user behavior.
- Provide analytics to help users understand asset usage and trends.

These functional and non-functional requirements should serve as a solid foundation for the development of a Digital Asset Management system on the Ethereum blockchain. The specific details and priorities of these requirements may vary depending on the project's scope and goals.

5.PROJECT DESIGN

5.1 Data Flow Daigram and User Stories



TRANSACTION MANAGEMENT IN BLOCK CHAIN

Story 1

Kaviya as a new user, I want to create an account on the DAM system.I should be able to provide my email, username, and password.Upon registration, I want a unique Ethereum wallet address to be generated and associated with my account.

Story 2

Kaviya as a content creator, I want to upload my digital assets (images, videos) to the DAM system.I should be able to add metadata such as title, description, and tags.Upon upload, I want these assets to be securely stored on the Ethereum blockchain.

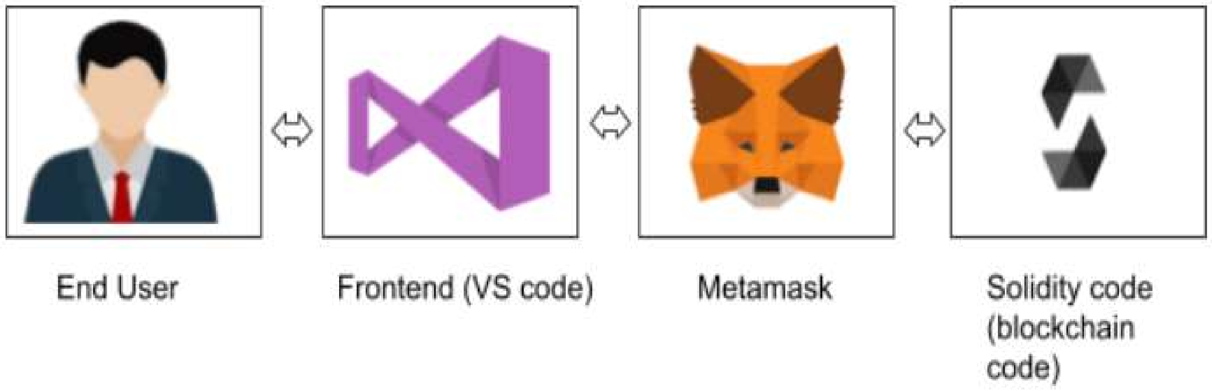
Story 3

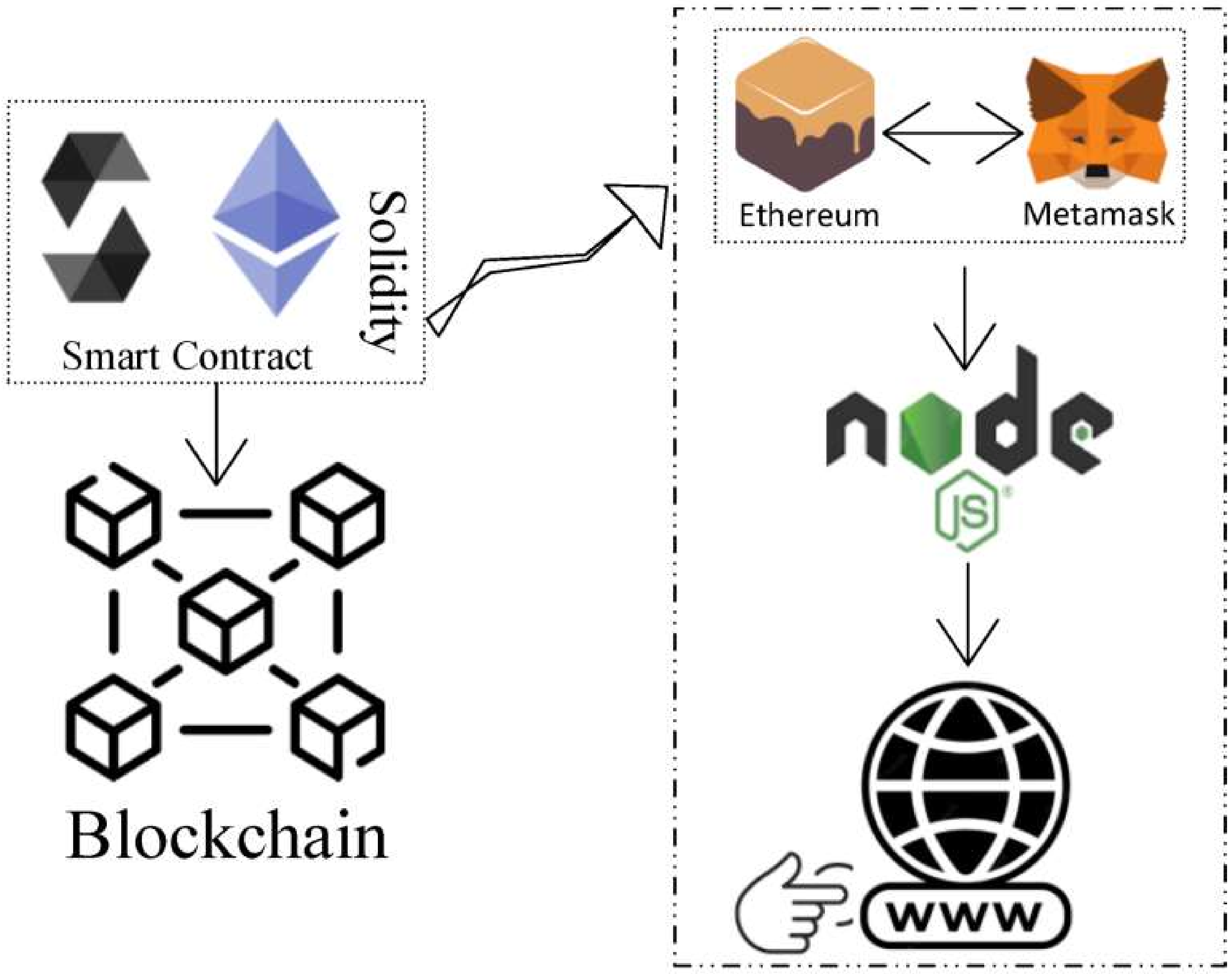
Kaviya as a content manager, I want to edit metadata for my digital assets.I should be able to update titles, descriptions, and tags.I also want the ability to delete assets if necessary.

Story 4

Kaviya as a user, I want to transfer ownership of my digital assets to another user.I should be able to specify the recipient's Ethereum wallet address.The transfer should be recorded on the blockchain.

5.2 Solution Architecture

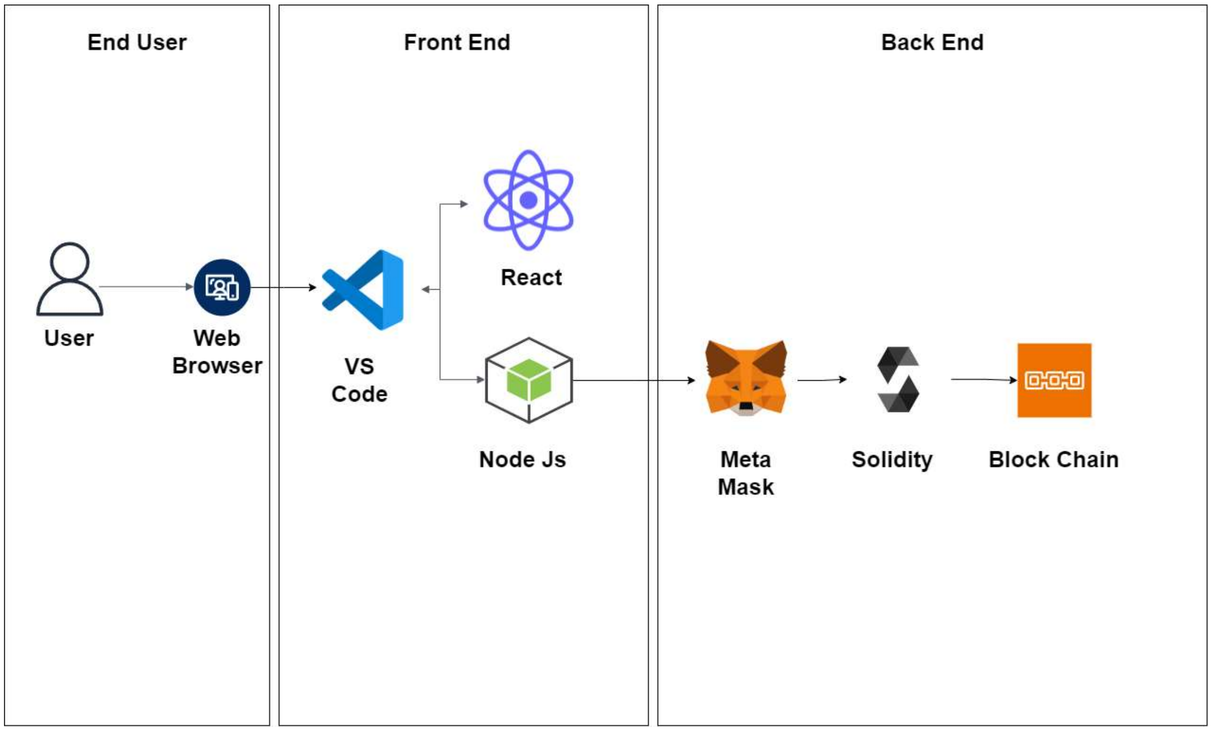




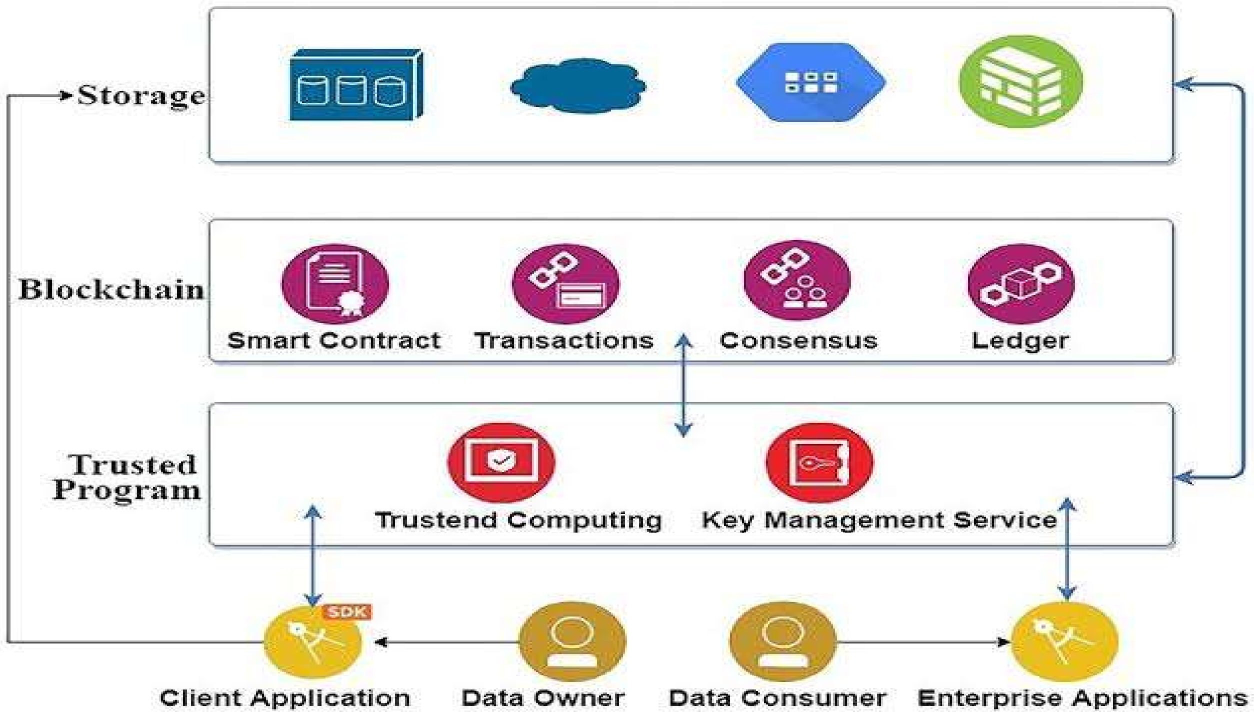
INTRACTION BETWEEN WEB AND THE CONTRACT

6.PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



GENERAL ARCHITECTURE



FLOW OF TRANSACTION

6.2 Sprint Planning and Estimation

Sprint planning is an essential part of Agile project management, helping teams organize their work for a specific time frame, usually two to four weeks. In the context of Digital Asset Management (DAM) on the Ethereum blockchain, the sprint planning would involve breaking down tasks related to the development, testing, and deployment of the DAM system.

Setting sprint goals: Develop a Minimum Viable Product (MVP) for a decentralized DAM system on the Ethereum blockchain.

Breaking Down User Stories: User stories and tasks are further decomposed into smaller, actionable sub-tasks. This detailed breakdown helps create a comprehensive plan for the sprint.

Estimation Team Work: Blockchain developers, smart contract developers, front-end developers, QA, product owner, and Scrum Master.

Sprint Backlog: The selected user stories and tasks, along with their estimates, constitute the sprint backlog. This forms the basis for what the team will work on during the sprint.

Sprint Asset Upload and Storage: Develop a smart contract for asset storage.,implement file upload functionality on the front-end.,ensure encryption and security measures for asset storage.

Sprint Asset Retrieval and Sharing: As a user, I want to retrieve and share my digital assets securely.

This sprint plan is a starting point for developing a decentralized Digital Asset Management system on the Ethereum blockchain. Adjust the tasks, user stories, and priorities based on your project's specific requirements and the feedback gathered during each sprint. Make sure to maintain regular communication within the team and stakeholders throughout the sprint to ensure a successful project delivery.

6.3 Sprint Delivering Schedule:

Week 1:**Create a Environment and Requirement**

- Conduct a project kickoff meeting to clarify goals and objectives.
- Create a detailed project scope document. Set up the Ethereum development environment.
- Develop a basic Ethereum smart contract for asset registration.
- Begin work on user authentication and access control.
- Conduct a sprint review and retrospective.

Week 2:**Enhance a Assest Management**

- Continue development of the Ethereum smart contract for asset management.
- Create a user interface for asset registration and management.
- Implement the ability to transfer assets between users.
- Conduct unit testing to ensure the smart contract's functionality.
- Conduct a sprint review and retrospective.

Week 3:Finalize and Prepare

- Develop a user-friendly dashboard for asset tracking.
- Enhance access control and security.
- Implement metadata support for assets.
- Begin integration testing.

7.CODING AND SOLUTING

7.1 Feature1

Smart Contract(Solidity)

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

pragma solidity ^0.8.0;


contract DigitalAssetRegistry {

  struct DigitalAsset {

    address owner;

    string assetHash;

    bool isPublished;

  }


  mapping(string => DigitalAsset) public digitalAssets;


  event AssetRegistered(address owner, string assetHash);

  event AssetPublished(string assetHash);

  event AssetUnpublished(string assetHash);

  event AssetOwnershipTransferred(string assetHash, address newOwner);


  modifier onlyOwner(string memory _assetHash) {

    require(digitalAssets[_assetHash].owner == msg.sender, "Only the owner can perform this action");

    _;

  }
  modifier onlyNotPublished(string memory _assetHash) {

    require(!digitalAssets[_assetHash].isPublished, "Asset is already published");

    _;

  }


  function registerAssetstring memory _assetHash) external {

   // require(digitalAssets[_assetHash].owner != address(0), "Asset already registered");

    digitalAssets[_assetHash] = DigitalAsset(msg.sender, _assetHash, false);

    emit AssetRegistered(msg.sender, _assetHash);

  }


  function publishAsset(string memory _assetHash) external onlyOwner(_assetHash)
onlyNotPublished(_assetHash) {

    digitalAssets[_assetHash].isPublished = true;

    emit AssetPublished(_assetHash);

  }


  function unpublishAsset(string memory _assetHash) external onlyOwner(_assetHash) {

    digitalAssets[_assetHash].isPublished = false;

    emit AssetUnpublished(_assetHash);

  }


  function transferOwnership(string memory _assetHash, address _newOwner) external
onlyOwner(_assetHash) {

    require(_newOwner != address(0), "Invalid new owner address");

    digitalAssets[_assetHash].owner = _newOwner;

    emit AssetOwnershipTransferred(_assetHash, _newOwner);

  }
```

```
}
```

The provided Solidity smart contract is named "DigitalAssetRegistry" and it allows users to register digital assets, publish or unpublish them, and transfer ownership of these assets. This is a struct that represents a digital asset. “owner” The address of the owner of the digital asset.“assetHash” A string representing the unique identifier or hash of the digital asset. A boolean flag indicating whether the asset is published or not. A public mapping that associates an asset hash (string) with a DigitalAsset struct. This mapping is used to store information about registered digital assets. Fired when a new digital asset is registered. It logs the owner's address and the asset hash. A custom modifier that restricts certain functions to be callable only by the owner of a specific digital asset. registerAsset: Allows any user to register a new digital asset by providing its unique asset hash. The asset is associated with the caller's address, and it is not published by default.

7.2**Feature 2**

Frond end(Java Script)

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

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

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

import { contract } from "./connector";


function Home() {
  const [RegHash, setRegHash] = useState("");


  const [PubHash, setPubHash] = useState("");




  const [UnPubHash, setUnPubHash] = useState("");




  const [TransferHash, setTransferHash] = useState("");

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

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

  const [Details, setDetails] = useState("");



  // const handlePolicyNumber = (e) => {

  //    setNumber(e.target.value)

  // }


  const handleRegHash = (e) => {

    setRegHash(e.target.value)

  }



  const handleRegAsset = async () => {

    try {

      let tx = await contract.registerAsset(RegHash)

      let wait = await tx.wait()

      alert(wait.transactionHash)

      console.log(wait);

    } catch (error) {

      alert(error)

    }

  }



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



  const handlePublish = async () => {
    try {
      let tx = await contract.publishAsset(PubHash)
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }

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



  const handleUnPublish = async () => {
    try {
      let tx = await contract.unpublishAsset(UnPubHash)
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }

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

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

  const handleTransfer = async () => {
    try {
      let tx = await contract.transferOwnership(TransferHash, Addr)
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }




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

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

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

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

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

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

  setWallet(addr[0])
  }


 return (

  <div>

   <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Digital Asset Registry 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={handlePolicyNumber}
type="string" placeholder="Policy number" value={number} /> <br /> */}

      <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleRegHash} type="string"
placeholder="Assets Hash" value={RegHash} /> <br />

       <Button    onClick={handleRegAsset}    style={{    marginTop:    "10px"    }}    variant="primary">Register
Asset</Button>

      </div>

     </Col>
```

```
      <Col style={{ marginRight: "100px" }}>
        <div>
          <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handlePublishHash}
type="string" placeholder="Asset Hash" value={PubHash} /> <br />


          <Button onClick={handlePublish} style={{ marginTop: "10px" }} variant="primary"> Publish
Assets</Button>
        </div>
      </Col>




    </Row>
    <Row style={{marginTop:"100px"}}>
            <Col style={{ marginRight: "100px" }}>
              <div>
                <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleUnPublishHash}
type="string" placeholder="Assets Hash" value={UnPubHash} /> <br />


                <Button onClick={handleUnPublish} style={{ marginTop: "10px" }} variant="primary">
Unpublish Assets</Button>
              </div>
            </Col>




            <Col style={{ marginRight: "100px" }}>
              <div>
                <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleTransferHash}
type="string" placeholder="Asset Hash" value={TransferHash} /> <br />
                <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleAddr}
type="string" placeholder="Owner metamask address" value={Addr} /> <br />
```

```
              <Button onClick={handleTransfer} style={{ marginTop: "10px" }} variant="primary"> Transfer
Ownership</Button>

            </div>

          </Col>



      </Row>

        <Row style={{ marginTop: "50px" }}>

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

            <div style={{ margin: "auto", marginTop: "100px" }}>

              <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleGetIds}
type="string" placeholder="Enter Assets Hash" value={gId} /><br />


              <Button onClick={handleGetDetails} style={{ marginTop: "10px" }} variant="primary">Get
Digital Assets</Button>

              {Details ? Details?.map(e => {

                return <p>{e.toString()}</p>

              }) : <p></p>}

            </div>

          </Col>

  </Row>

  </Container>


 </div>

 )

}
```

export default Home;

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

8.1 Performance Matrix

Security and Compliance: Smart Contract Audits number of smart contract security audits conducted.Vulnerabilities detected number of vulnerabilities identified and resolved.Compliance with regulatory standards evaluation of adherence to relevant regulations.

Asset Management: Total Assets Under Management (AUM) value of digital assets being managed.Asset Types Supported that variety of digital assets (e.g., tokens, NFTs) supported.Asset Lifecycle Management monitoring asset creation, transfer, and destruction.

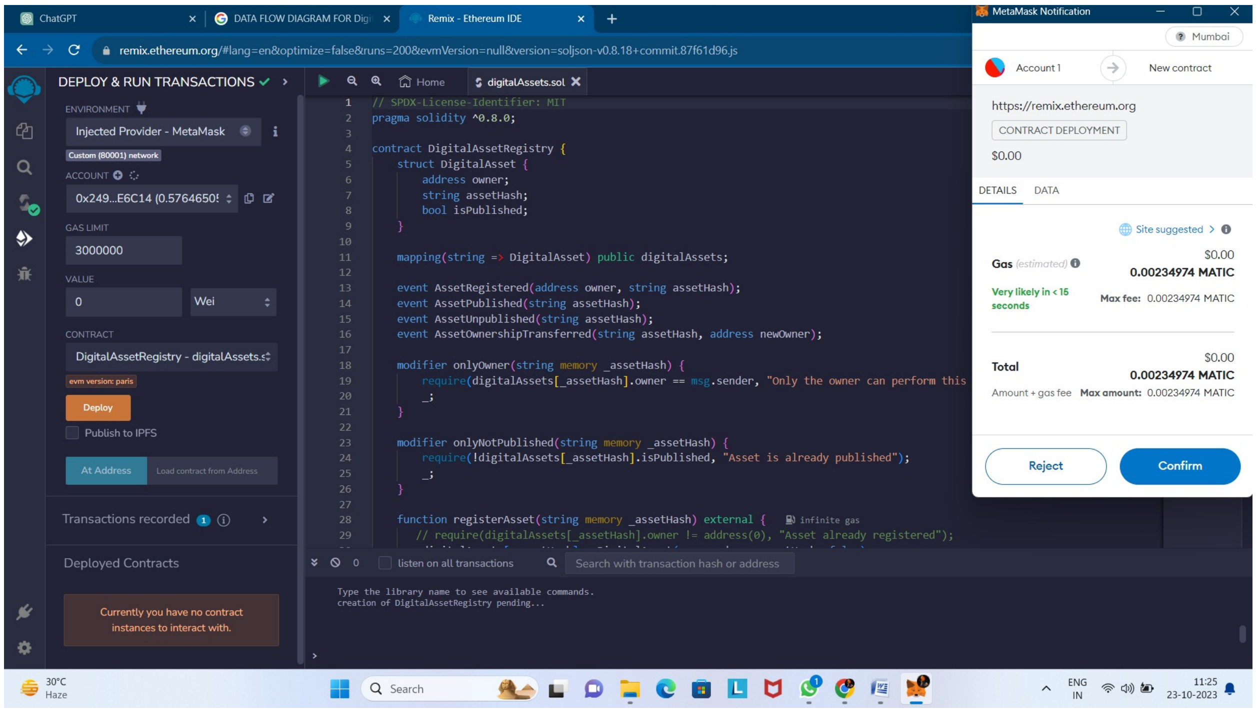
User Experience: User Interface (UI) that easy of use, design, and intuitiveness of the user interface.User Adoption number of users and their activity within the DAM system.Transaction speed is average confirmation time for asset transfers.

Scalability: Transactions per Second the system's capacity to handle transactions.Network Congestionof frequency and impact of network congestion on operations.Gas Costsis associated with Ethereum gas fees for transactions.

Interoperability: Cross-Chain Compatibilitythat Support for interoperability with other blockchains.Integration with External Systems Capability to connect with external services and applications.

9.RESULTS

9.1 OUTPUT SCREENSHOTS



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT: Injected Provider - MetaMask (Custom (80001) network)

ACCOUNT: 0x249...E6C14 (0.5764650!

GAS LIMIT: 3000000

VALUE: 0 Wei

CONTRACT: DigitalAssetRegistry - digitalAssets.sol (evm version: paris)

Currently you have no contract instances to interact with.

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

contract DigitalAssetRegistry {
    struct DigitalAsset {
        address owner;
        string assetHash;
        bool isPublished;
    }

    mapping(string => DigitalAsset) public digitalAssets;

    event AssetRegistered(address owner, string assetHash);
    event AssetPublished(string assetHash);
    event AssetUnpublished(string assetHash);
    event AssetOwnershipTransferred(string assetHash, address newOwner);

    modifier onlyOwner(string memory _assetHash) {
        require(digitalAssets[_assetHash].owner == msg.sender, "Only the owner can perform this
        _;
    }

    modifier onlyNotPublished(string memory _assetHash) {
        require(!digitalAssets[_assetHash].isPublished, "Asset is already published");
        _;
    }

    function registerAsset(string memory _assetHash) external {
      // require(digitalAssets[_assetHash].owner != address(0), "Asset already registered");
```

```
Type the library name to see available commands.
creation of DigitalAssetRegistry pending...
```

MetaMask Notification — Mumbai — Account 1 → New contract

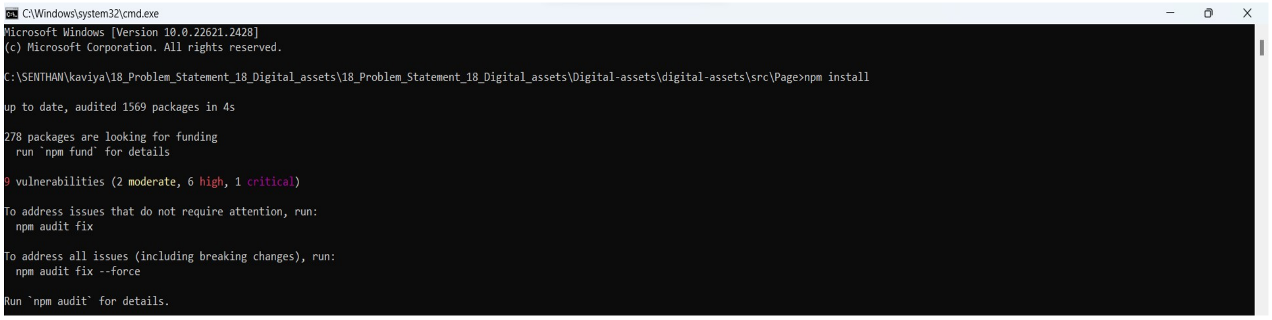
https://remix.ethereum.org — CONTRACT DEPLOYMENT — $0.00

Gas (estimated): $0.00, 0.00234974 MATIC — Very likely in < 15 seconds — Max fee: 0.00234974 MATIC

Total: $0.00, 0.00234974 MATIC — Amount + gas fee — Max amount: 0.00234974 MATIC

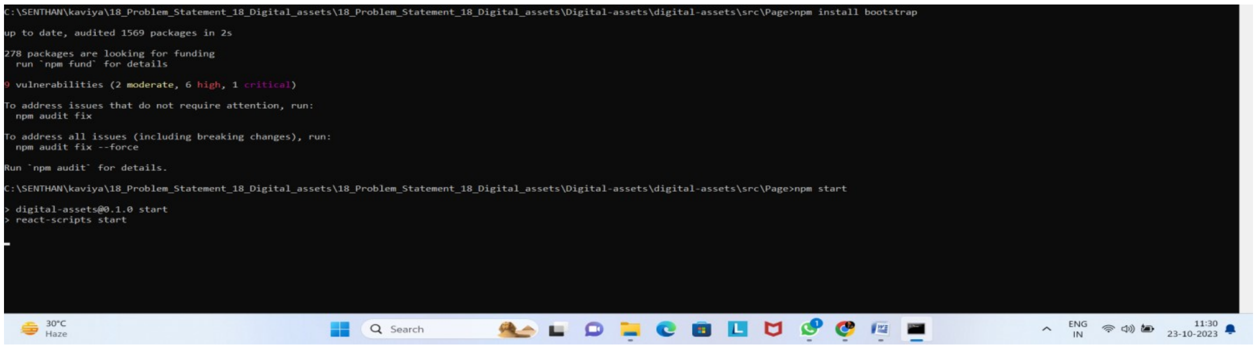
ChatGPT
DATA FLOW DIAGRAM FOR Digi
Remix - Ethereum IDE
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
Home
digitalAssets.sol
Deploy
Publish to IPFS
At Address
Load contract from Address
Transactions recorded 1
Deployed Contracts
listen on all transactions
Search with transaction hash or address
Site suggested
DETAILS
DATA
Reject
Confirm
30°C Haze
Search
ENG IN
11:25 23-10-2023

CREATING A SMART CONTACT



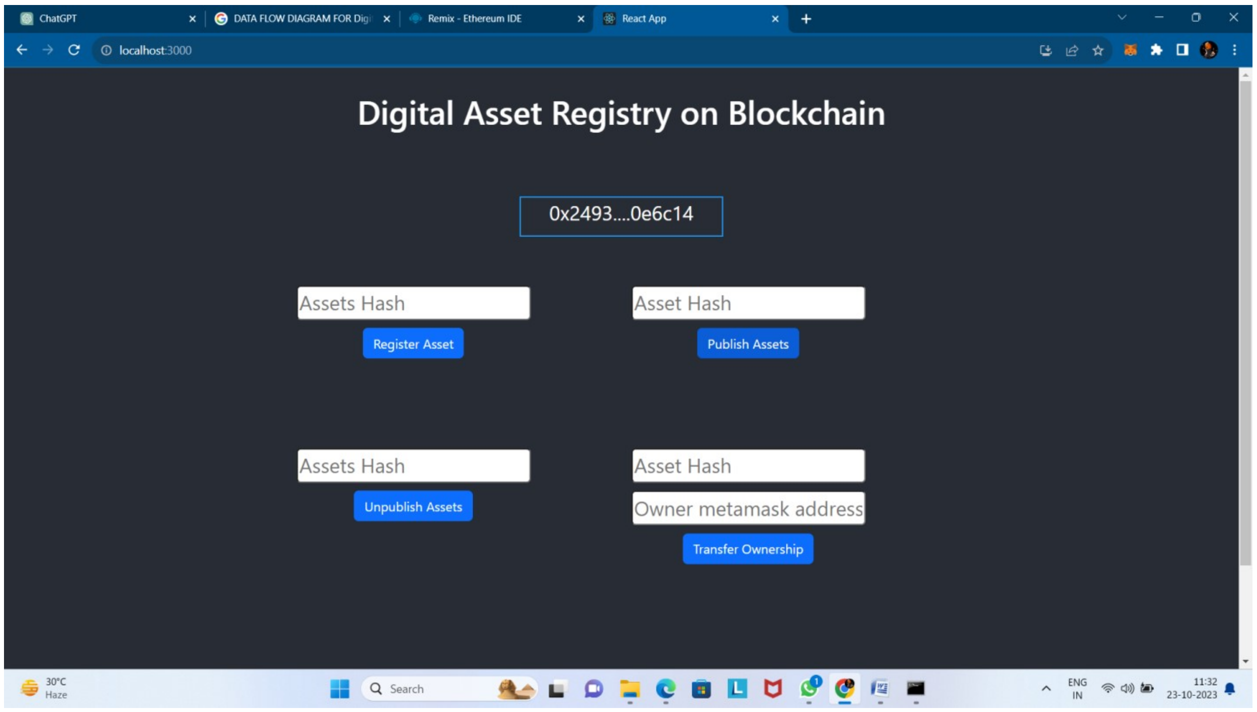
```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.22621.2428]
(c) Microsoft Corporation. All rights reserved.

C:\SENTHAN\kaviya\18_Problem_Statement_18_Digital_assets\18_Problem_Statement_18_Digital_assets\Digital-assets\digital-assets\src\Page>npm install

up to date, audited 1569 packages in 4s

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

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

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

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

Run `npm audit` for details.
```

INSTALLING DEPENDENCIES 1



```
C:\SENTHAN\kaviya\18_Problem_Statement_18_Digital_assets\18_Problem_Statement_18_Digital_assets\Digital-assets\digital-assets\src\Page>npm install bootstrap

up to date, audited 1569 packages in 2s

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

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

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

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

Run `npm audit` for details.

C:\SENTHAN\kaviya\18_Problem_Statement_18_Digital_assets\18_Problem_Statement_18_Digital_assets\Digital-assets\digital-assets\src\Page>npm start

> digital-assets@0.1.0 start
> react-scripts start
```

INSTALLING DEPENDENCIES 2



WEB PAGE OUTPUT

10.ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

Ethereum's blockchain is known for its robust security features. Assets are stored in a decentralized manner, making it extremely difficult for malicious actors to compromise the integrity of your digital assets. Ethereum's smart contracts also enable automated and trustless transactions, reducing the risk of fraud. Once data is recorded on the Ethereum blockchain, it becomes nearly impossible to alter or delete it. This immutability ensures the integrity and permanence of your digital assets, making it an ideal solution for long-term storage and record-keeping. Ethereum is a public blockchain, which means that anyone with an internet connection can access and interact with it. This open accessibility fosters transparency and inclusivity, allowing a wide range of users to engage with your digital assets. Ethereum's compatibility with other blockchains and platforms allows for seamless integration with various decentralized applications (dApps) and services. This interoperability facilitates the creation of complex digital asset management solutions and ecosystems. Ethereum's smart contracts enable the easy creation of tokens that represent ownership in real-world or digital assets. This allows for the fractional ownership of high-value assets, making investments more accessible to a wider range of people. Ethereum operates on a decentralized network of nodes, reducing the risk of a single point of failure. This distributed nature ensures that your digital assets are not controlled by a single entity, providing greater resilience against censorship and downtime. Ethereum is accessible to a global audience, making it easy to manage digital assets across borders. It eliminates the need for intermediaries and facilitates cross-border transactions and asset management. All transactions and operations on the Ethereum blockchain are publicly recorded and verifiable. This transparency enhances trust and accountability in digital asset management.

10.2 DISADVANTAGES

Ethereum has faced scalability challenges, leading to slow transaction processing times and high fees during periods of network congestion. This can make managing digital assets on the platform costly and inefficient. The regulatory environment for cryptocurrencies and blockchain technology is still evolving. Digital asset managers on the Ethereum blockchain may face legal and compliance issues as governments worldwide establish rules and regulations for this technology. While the Ethereum blockchain is generally secure, there is always a risk of vulnerabilities, hacks, and smart contract bugs. If a digital asset manager doesn't have strong security measures in place, assets can be at risk. Ethereum's blockchain is known for its transparency, which can be a disadvantage when it comes to privacy. Managing digital assets on a public blockchain means that transactions and asset holdings can be easily traced and monitored by anyone. The user experience for managing digital assets on the Ethereum blockchain may not be as user-friendly as traditional financial systems. Handling private keys, gas fees, and smart contracts can be complex for the average user. Ethereum, like many blockchain platforms, relies on energy-intensive consensus mechanisms, such as proof-of-work (although Ethereum is transitioning to proof-of-stake). This can lead to environmental concerns due to high energy consumption. During periods of high demand or network congestion, Ethereum can experience delays and high transaction fees. This can be frustrating for digital asset managers, as it can impact the timeliness and cost-effectiveness of their operations.

11 CONCLUSION

Digital Asset Management on the Ethereum blockchain offers a revolutionary approach to securely and transparently manage a wide range of digital assets. By leveraging the power of blockchain technology, Ethereum provides a decentralized and tamper-resistant platform for tracking, trading, and storing digital assets, from cryptocurrencies to non-fungible tokens (NFTs). This technology has the potential to disrupt traditional asset management systems and enable new possibilities in finance, supply chain management, and various other industries. However, it's important to note that challenges like scalability, regulatory compliance, and user adoption must be addressed for the full potential of digital asset management on Ethereum to be realized. Nonetheless, it represents a promising future for the digitization and management of assets in the digital age.

By utilizing smart contracts and decentralized applications (dApps), users can securely store, transfer, and trade digital assets, eliminating the need for traditional intermediaries and reducing the risk of fraud and unauthorized access. This technology not only provides transparency and traceability but also enhances accessibility and interoperability across different platforms and applications.

However, it's important to note that there are still challenges to overcome, such as scalability issues and regulatory concerns. As the technology continues to evolve, it is vital for stakeholders to collaborate on solutions that can address these issues and foster mainstream adoption.

Digital Asset Management on the Ethereum blockchain is a promising innovation that has the potential to revolutionize how we handle and exchange digital assets. As the ecosystem matures and develops, it holds the promise of unlocking new opportunities and reshaping the future of digital finance and ownership.

Digital Asset Management on the Ethereum blockchain represents a significant advancement in the way we handle and safeguard various types of digital assets. Leveraging the security, transparency, and decentralization features of the Ethereum blockchain, this technology offers a robust and efficient solution for managing a wide range of assets, including cryptocurrencies, non-fungible tokens (NFTs), and more.

12.FUTURE SCOPE

The future scope for Digital Asset Management on the Ethereum blockchain is promising and filled with potential. As of my last knowledge update in September 2021, Ethereum has been a pioneering platform for decentralized applications, smart contracts, and tokenization. Here are some key points to consider when looking at the future of Digital Asset Management on the Ethereum blockchain:

Tokenization of Real-World Assets: Ethereum's blockchain can be used to tokenize real-world assets like real estate, art, and commodities. This can enable more efficient and accessible trading of such assets, increasing liquidity and lowering barriers to entry for investors.

DeFi Integration: The integration of decentralized finance (DeFi) applications with Digital Asset Management is expected to grow. This can lead to innovative financial products, like yield-generating assets, lending, and borrowing of digital assets, all managed securely on the Ethereum blockchain.

Interoperability: As blockchain technology evolves, we may see improved interoperability between different blockchain networks. Ethereum's ERC-20 and ERC-721 standards are already widely adopted, and cross-chain solutions are being developed. This could enable assets to move seamlessly between different blockchains.

Regulatory Frameworks: The regulatory environment around digital assets is evolving. Governments and regulators are working on establishing clearer rules and guidelines for the management and trading of digital assets. The future of Digital Asset Management will likely involve compliance with these regulations, providing greater trust for institutional and retail investors.

Security and Scalability: Ethereum is actively working on improving its scalability and security through upgrades like Ethereum 2.0. As these enhancements roll out, Digital Asset Management platforms on Ethereum can offer more efficient, secure, and cost-effective solutions.

Decentralized Autonomous Organizations (DAOs): DAOs, which are smart contract-based organizations governed by token holders, can play a significant role in Digital Asset Management. They can allow asset holders to have a say in the management and decision-making processes.

N**FTs and Digital Collectibles:** Non-fungible tokens (NFTs) are already popular on Ethereum, and they have potential applications in asset management. This can include ownership and transfer of digital collectibles, intellectual property rights, and more.

Data Management: Efficient data management is crucial for asset management. Decentralized solutions on Ethereum can offer secure and transparent ways to manage and verify the data associated with digital assets.

Digital Asset Management on the Ethereum blockchain is likely to be shaped by technological advancements, regulatory developments, and the growing demand for decentralized, efficient, and secure solutions for asset management. As the blockchain ecosystem continues to mature, we can expect exciting developments in this field. However, it's important to note that the landscape is continually evolving, and staying updated with the latest trends and innovations is crucial for anyone involved in digital asset management on Ethereum or any blockchain platform.

13.APPENDIX

SOURCE CODE

Digitalassest.sol

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

pragma solidity ^0.8.0;


contract DigitalAssetRegistry {

  struct DigitalAsset {

    address owner;

    string assetHash;

    bool isPublished;

  }


  mapping(string => DigitalAsset) public digitalAssets;


  event AssetRegistered(address owner, string assetHash);

  event AssetPublished(string assetHash);

  event AssetUnpublished(string assetHash);

  event AssetOwnershipTransferred(string assetHash, address newOwner);


  modifier onlyOwner(string memory _assetHash) {

    require(digitalAssets[_assetHash].owner == msg.sender, "Only the owner can perform this action");

    _;

  }


  modifier onlyNotPublished(string memory _assetHash) {
        require(!digitalAssets[_assetHash].isPublished, "Asset is already published");

        _;

    }


    function registerAsset(string memory _assetHash) external {

      // require(digitalAssets[_assetHash].owner != address(0), "Asset already registered");

        digitalAssets[_assetHash] = DigitalAsset(msg.sender, _assetHash, false);

        emit AssetRegistered(msg.sender, _assetHash);

    }


    function publishAsset(string memory _assetHash) external onlyOwner(_assetHash)
onlyNotPublished(_assetHash) {

        digitalAssets[_assetHash].isPublished = true;

        emit AssetPublished(_assetHash);

    }


    function unpublishAsset(string memory _assetHash) external onlyOwner(_assetHash) {

        digitalAssets[_assetHash].isPublished = false;

        emit AssetUnpublished(_assetHash);

    }


    function transferOwnership(string memory _assetHash, address _newOwner) external
onlyOwner(_assetHash) {

        require(_newOwner != address(0), "Invalid new owner address");

        digitalAssets[_assetHash].owner = _newOwner;

        emit AssetOwnershipTransferred(_assetHash, _newOwner);

    }

}
```

Connector.js

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


const abi =[
	{
		"anonymous": false,
		"inputs": [
			{
				"indexed": false,
				"internalType": "string",
				"name": "assetHash",
				"type": "string"
			},
			{
				"indexed": false,
				"internalType": "address",
				"name": "newOwner",
				"type": "address"
			}
		],
		"name": "AssetOwnershipTransferred",
		"type": "event"
	},
	{
		"anonymous": false,
		"inputs": [
			{
				"indexed": false,
					"internalType": "string",
					"name": "assetHash",
					"type": "string"
				}
			],
			"name": "AssetPublished",
			"type": "event"
		},
		{
			"anonymous": false,
			"inputs": [
				{
					"indexed": false,
					"internalType": "address",
					"name": "owner",
					"type": "address"
				},
				{
					"indexed": false,
					"internalType": "string",
					"name": "assetHash",
					"type": "string"
				}
			],
			"name": "AssetRegistered",
			"type": "event"
		},
		{
```

```
		"anonymous": false,
		"inputs": [
			{
				"indexed": false,
				"internalType": "string",
				"name": "assetHash",
				"type": "string"
			}
		],
		"name": "AssetUnpublished",
		"type": "event"
	},
	{
		"inputs": [
			{
				"internalType": "string",
				"name": "_assetHash",
				"type": "string"
			}
		],
		"name": "publishAsset",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "string",
				"name": "_assetHash",
				"type": "string"
			}
		],
		"name": "registerAsset",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	},
	{
		"inputs": [
			{
				"internalType": "string",
				"name": "_assetHash",
				"type": "string"
			},
			{
				"internalType": "address",
				"name": "_newOwner",
				"type": "address"
			}
		],
		"name": "transferOwnership",
		"outputs": [],
		"stateMutability": "nonpayable",
		"type": "function"
	},
{
	"inputs": [
		{
			"internalType": "string",
			"name": "_assetHash",
			"type": "string"
		}
	],
	"name": "unpublishAsset",
	"outputs": [],
	"stateMutability": "nonpayable",
	"type": "function"
},
{
	"inputs": [
		{
			"internalType": "string",
			"name": "",
			"type": "string"
		}
	],
	"name": "digitalAssets",
	"outputs": [
		{
			"internalType": "address",
			"name": "owner",
			"type": "address"
		},
                    {
                              "internalType": "string",
                              "name": "assetHash",
                              "type": "string"
                    },
                    {
                              "internalType": "bool",
                              "name": "isPublished",
                              "type": "bool"
                    }
          ],
          "stateMutability": "view",
          "type": "function"
     }
]

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

export const provider = new ethers.providers.Web3Provider(window.ethereum);
export const signer = provider.getSigner();
export const address = "0xd0a80D9Faf79d7a5fbcBa3FA00f8A9Cb016FDA6b"

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

Home.js

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';

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

import { contract } from "./connector";


function Home() {

  const [RegHash, setRegHash] = useState("");


  const [PubHash, setPubHash] = useState("");




  const [UnPubHash, setUnPubHash] = useState("");





  const [TransferHash, setTransferHash] = useState("");

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

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







  const [gId, setGIds] = useState("");

  const [Details, setDetails] = useState("");




  // const handlePolicyNumber = (e) => {

  //    setNumber(e.target.value)

  // }


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

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



const handleRegAsset = async () => {
  try {
    let tx = await contract.registerAsset(RegHash)
    let wait = await tx.wait()
    alert(wait.transactionHash)
    console.log(wait);
  } catch (error) {
    alert(error)
  }
}



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



const handlePublish = async () => {
  try {
    let tx = await contract.publishAsset(PubHash)
    let wait = await tx.wait()
    console.log(wait);
    alert(wait.transactionHash)
  } catch (error) {
      alert(error)
    }
  }

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



  const handleUnPublish = async () => {
    try {
      let tx = await contract.unpublishAsset(UnPubHash)
      let wait = await tx.wait()
      console.log(wait);
      alert(wait.transactionHash)
    } catch (error) {
      alert(error)
    }
  }

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

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

```
const handleTransfer = async () => {
  try {
    let tx = await contract.transferOwnership(TransferHash, Addr)
    let wait = await tx.wait()
    console.log(wait);
    alert(wait.transactionHash)
  } catch (error) {
    alert(error)
  }
}




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

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

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

    console.log(tx);
```

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

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

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

      setWallet(addr[0])

    }

  return (
   <div>
    <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Digital Asset Registry 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={handlePolicyNumber}
type="string" placeholder="Policy number" value={number} /> <br /> */}

     <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleRegHash} type="string"
placeholder="Assets Hash" value={RegHash} /> <br />

      <Button onClick={handleRegAsset} style={{ marginTop: "10px" }} variant="primary">Register
Asset</Button>

     </div>

    </Col>



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

       <div>

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



         <Button onClick={handlePublish} style={{ marginTop: "10px" }} variant="primary"> Publish
Assets</Button>

       </div>

     </Col>
```

```
 </Row>

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

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

          <div>

            <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleUnPublishHash}
type="string" placeholder="Assets Hash" value={UnPubHash} /> <br />


            <Button onClick={handleUnPublish} style={{ marginTop: "10px" }} variant="primary">
Unpublish Assets</Button>

          </div>

        </Col>



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

          <div>

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

            <input style={{ marginTop: "10px", borderRadius: "5px" }} onChange={handleAddr}
type="string" placeholder="Owner metamask address" value={Addr} /> <br />



            <Button onClick={handleTransfer} style={{ marginTop: "10px" }} variant="primary"> Transfer
Ownership</Button>

          </div>

        </Col>



    </Row>

      <Row style={{ marginTop: "50px" }}>

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

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

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

  </div>
 )
}

export default Home;
App.js
import './App.css';
import Home from './Page/Home'

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <Home />
      </header>
    </div>
  );
}

export default App;
```

GITHUB_LINK:https://github.com/Kaviya-csbs/Digital-Assest-Management-on-the-Ethereum-BlockChain

DEMO_VIDEO_LINK:https://drive.google.com/file/d/1bD0zMdC_xfqbwGt_8Vm1aLlphacLucRu/view?usp=drivesdk