Project Report

Team ID	NM2023TMID03303
Project Name	Digital Asset Management on the Ethereum Blockchain

Submitted by

**TEAM LEADER : ** SARATHI R

TEAM MEMBER 1 : NAVEEN M

TEAM MEMBER 2 : VIGNESH N

TEAM MEMBER 3 : DHANUSH V

Team ID	NM2023TMID00438
Project Name	Digital Asset Management on the Ethereum Blockchain

1. INTRODUCTION

1.1 Project Overview

The proposed Ethereum-based Digital Asset Management (DAM) system will revolutionize the way individuals and organizations manage their digital assets.[1] Leveraging the power of Ethereum's blockchain, the project will enable users to create, register, and trade various digital assets, ranging from Non-Fungible Tokens (NFTs) to fungible tokens, in a trustless and decentralized environment.[2] Smart contracts will be at the core of this system, ensuring secure ownership management and streamlined asset transactions.[3]

Furthermore, the DAM system will feature an intuitive user interface that simplifies the asset management process, making it accessible to both experienced blockchain users and newcomers.[4] Interoperability with other blockchain networks will enhance the flexibility and value of digital assets. Compliance with relevant regulations will be a priority to ensure the system's legitimacy and adoption in various regions.[5] The project's scalability and performance optimizations will pave the way for a seamless experience as the ecosystem grows. Overall, this DAM
system holds the potential to disrupt traditional asset management by providing a secure, transparent, and user-/friendly solution on the Ethereum blockchain.[6]

1.2 Purpose

The primary purpose of developing a Digital Asset Management (DAM) system on the Ethereum blockchain is to create a secure and decentralized platform for individuals and businesses to effectively manage and trade digital assets. The blockchain's immutable ledger ensures the integrity of asset ownership, allowing users to confidently buy, sell, and trade assets without the need for intermediaries. This democratizes access to asset management, making it accessible to a global audience and fostering a peer-to-peer economy.

Additionally, the DAM system aligns with the broader trend of blockchain technology transforming traditional industries. It provides a solution for artists, content creators, collectors, and businesses to tokenize and manage their assets, including art, music, collectibles, and more, while ensuring traceability and provenance. This platform also unlocks new monetization opportunities, as asset creators can earn from transaction fees or minting their own NFTs. Ultimately, the system's purpose is to empower users, enhance asset liquidity, and bring the benefits of blockchain technology to the world of digital assets

2. LITERATURE SURVEY

2.1Existing problem

Market Volatility: Asset prices can be highly volatile, which can make it difficult to predict and manage risks effectively.
Lack of Diversification: Failing to diversify a portfolio can lead to concentrated risk and potential losses.
High Fees: Investment management fees can erode returns, especially if they are excessive.
Lack of Transparency: Some asset managers may not provide clear information about their investment strategies or portfolio holdings, making it hard for investors to make informed decisions.
Regulatory Compliance: Asset managers must adhere to complex regulatory requirements, which can be challenging to navigate.
Changing Market Conditions: Market dynamics change over time, and strategies that were once successful may become less effective.
Client Expectations: Meeting the diverse needs and expectations of clients, especially in a rapidly changing financial landscape, can be a complex task.
Technology Adoption: Keeping up with technological advancements and using them to gain an edge in asset management can be a struggle for some firms.
Environmental, Social, and Governance (ESG) Considerations: Incorporating ESG factors into investment strategies is becoming increasingly important, but it adds complexity to the decision-making process.
Behavioral Biases: Investors often make decisions based on emotions and cognitive biases, which can lead to suboptimal outcomes.
Performance Benchmarking: Evaluating the performance of asset managers against relevant benchmarks and justifying their fees can be challenging.
Macro-Economic Factors: Broader economic conditions can impact asset values and market trends, creating uncertainty.

2.2REFERENCES

[1] M. Barni and F. Bartolini, Watermarking Systems Engineering: EnablingDigital Assets Security and Other Applications. CRC Press, 2004
.
[2] U. W. Chohan, “Non-fungible tokens: Blockchains, scarcity, and value,”Critical Blockchain Research Initiative (CBRI) Working Papers, 2021.
[3] W. Ku and C.-H. Chi, “Survey on the technological aspects of digitalrights management,” in Proc. International Conference on InformationSecurity (ISC), 2004, pp. 391–403.

[4] H. R. Hasan and K. Salah, “Proof of delivery of digital assets usingblockchain and smart contracts,” IEEE Access, vol. 6, pp. 65439–65 448,2018

.[5] A. Garba, A. D. Dwivedi, M. Kamal, G. Srivastava, M. Tariq, M. A.Hasan, and Z. Chen, “A digital rights management system based on ascalable blockchain,” Peer-to-Peer Networking and Applications, vol. 14,no. 5, pp. 2665–2680, 2021.

[6] Y. Zhang, M. Yutaka, M. Sasabe, and S. Kasahara, “Attribute-basedaccess control for smart cities: A smart-contract-driven framework,”IEEE Internet of Things Journal, vol. 8, no. 8, pp. 6372–6384, 2020

2.3Problem Statement Definition

1.Who: Who is involved or affected by the situation or problem? This includes individuals, groups, or organizations.

Our organization, which relies on a wide range of digital assets such as images, videos, documents, and creative content, is directly affected by this problem. Key stakeholders include content creators, marketers, designers, and IT personnel responsible for managing these digital assets.

2.What: What is the issue or topic being discussed? What is the core problem, event, or subject of interest?

The current digital asset management (DAM) system in our organization is struggling to efficiently organize, store, retrieve, and distribute our digital assets. The metadata and tagging are inconsistent, leading to difficulties in searching and locating assets. This inconsistency is affecting various departments and teams, making it challenging for them to access and utilize digital assets effectively.

3.Where: Where is the situation or problem occurring? This can refer to a physical location, a specific context, or an environment.

This issue is pervasive across our organization, impacting all departments and locations that rely on digital assets. It influences marketing campaigns, content creation, product development, and other core functions.

4.When: When did the situation or problem occur, or when is it scheduled to happen? This includes the time frame, date, or historical context.

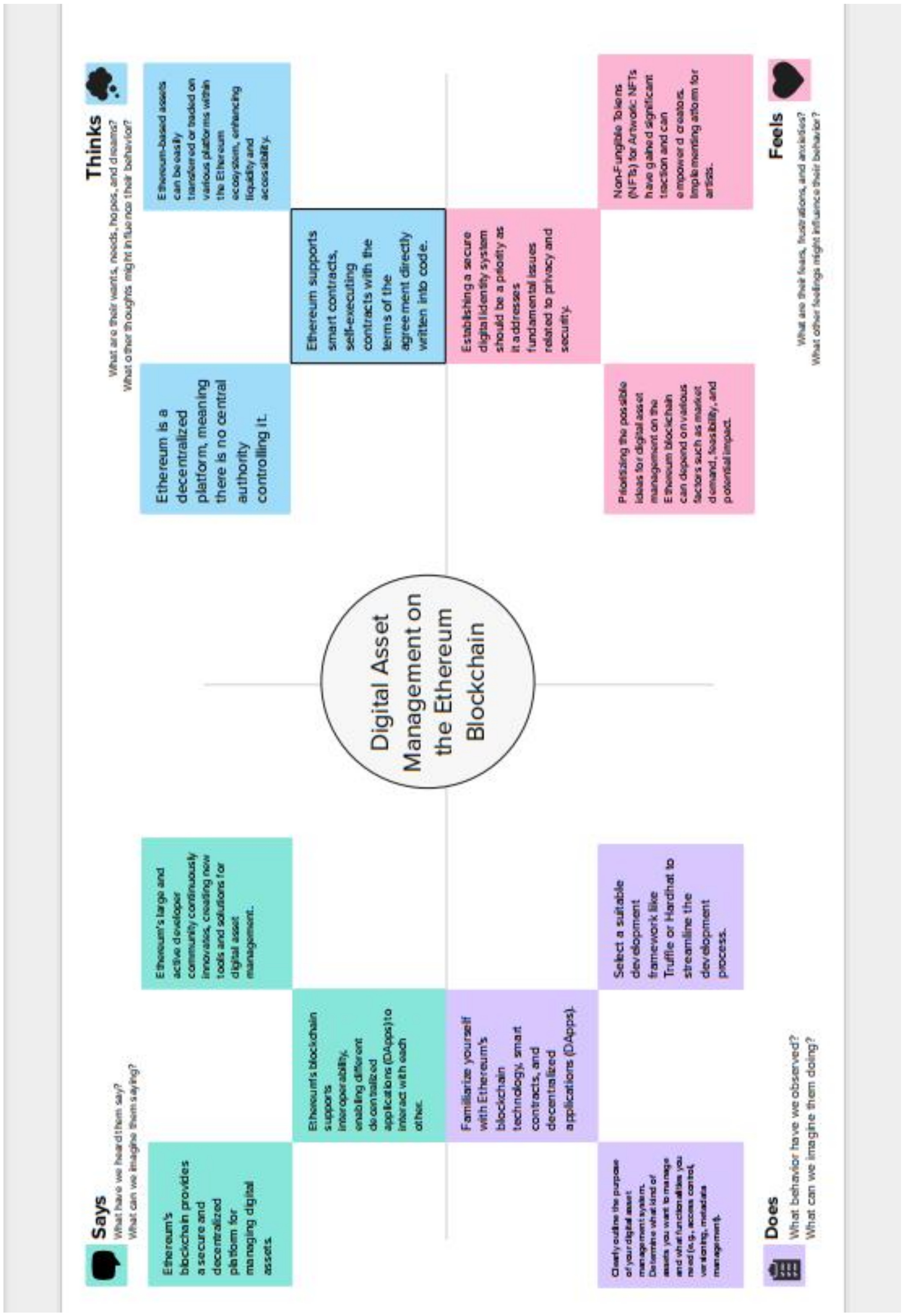
This issue is pervasive across our organization, impacting all departments and locations that rely on digital assets. It influences marketing campaigns, content creation, product development, and other core functions.

5.Why: Why is this situation or problem happening? What are the causes or motivations behind it? This is where you delve into the reasons or factors contributing to the issue.

The root causes of this problem include the absence of a centralized asset management system, inconsistent tracking processes, and limited visibility into the condition and location of assets. Inadequate maintenance scheduling and documentation exacerbate the problem. Without an efficient asset management solution, our organization is incurring unnecessary costs and experiencing operational inefficiencies, which directly impact our ability to serve our clients and achieve our business objectives.

3. IDEATION & PROPOSED SOLUTION

3.1Empathy Map Canvas


Says
What have we heard them say?
What can we imagine them saying?
Ethereum's blockchain provides a secure and decentralized platform for managing digital assets.
Ethereum's large and active developer community continuously innovates, creating new tools and solutions for digital asset management.
Ethereum's blockchain supports interoperability, enabling different decentralized applications (DApps) to interact with each other.
Thinks
What are their wants, needs, hopes, and dreams?
What other thoughts might influence their behavior?
Ethereum is a decentralized platform, meaning there is no central authority controlling it.
Ethereum-based assets can be easily transferred or traded on various platforms within the Ethereum ecosystem, enhancing liquidity and accessibility.
Ethereum supports smart contracts, self-executing contracts with the terms of the agreement directly written into code.
Digital Asset Management on the Ethereum Blockchain
Familiarize yourself with Ethereum's blockchain technology, smart contracts, and decentralized applications (DApps).
Clearly outline the purpose of your digital asset management system. Determine what kind of assets you want to manage and what functionalities you need (e.g., access control, versioning, metadata management).
Select a suitable development framework like Truffle or Hardhat to streamline the development process.
Does
What behavior have we observed?
What can we imagine them doing?
Establishing a secure digital identity system should be a priority as it addresses fundamental issues related to privacy and security.
Prioritizing the possible ideas for digital asset management on the Ethereum blockchain can depend on various factors such as market demand, feasibility, and potential impact.
Non-Fungible Tokens (NFTs) for Artwork: NFTs have gained significant traction and can empowerd creators. Implementing atform for artists.
Feels
What are their fears, frustrations, and anxieties?
What other feelings might influence their behavior?

3.2Ideation & Brainstorming

Dhanush V

- Artists can tokenize their artwork
- Tokens sent back or forward in time, altering.
- NFTs on Ethereum

Naveen M

- Decentralized Finance(DeFi) Platforms
- Infinite Token Supply
- Develop decentralized lending

Vignesh N

- Tokenized Real Estate
- Perfect Prediction Contracts
- actional ownership andeasier property transactions.

Sarathi R

- Supply Chain Management
- Universal Asset Conversion
- Digital Identity Management

Artists can tokenize their artwork

Tokenized Real Estate

NFTs on Ethereum

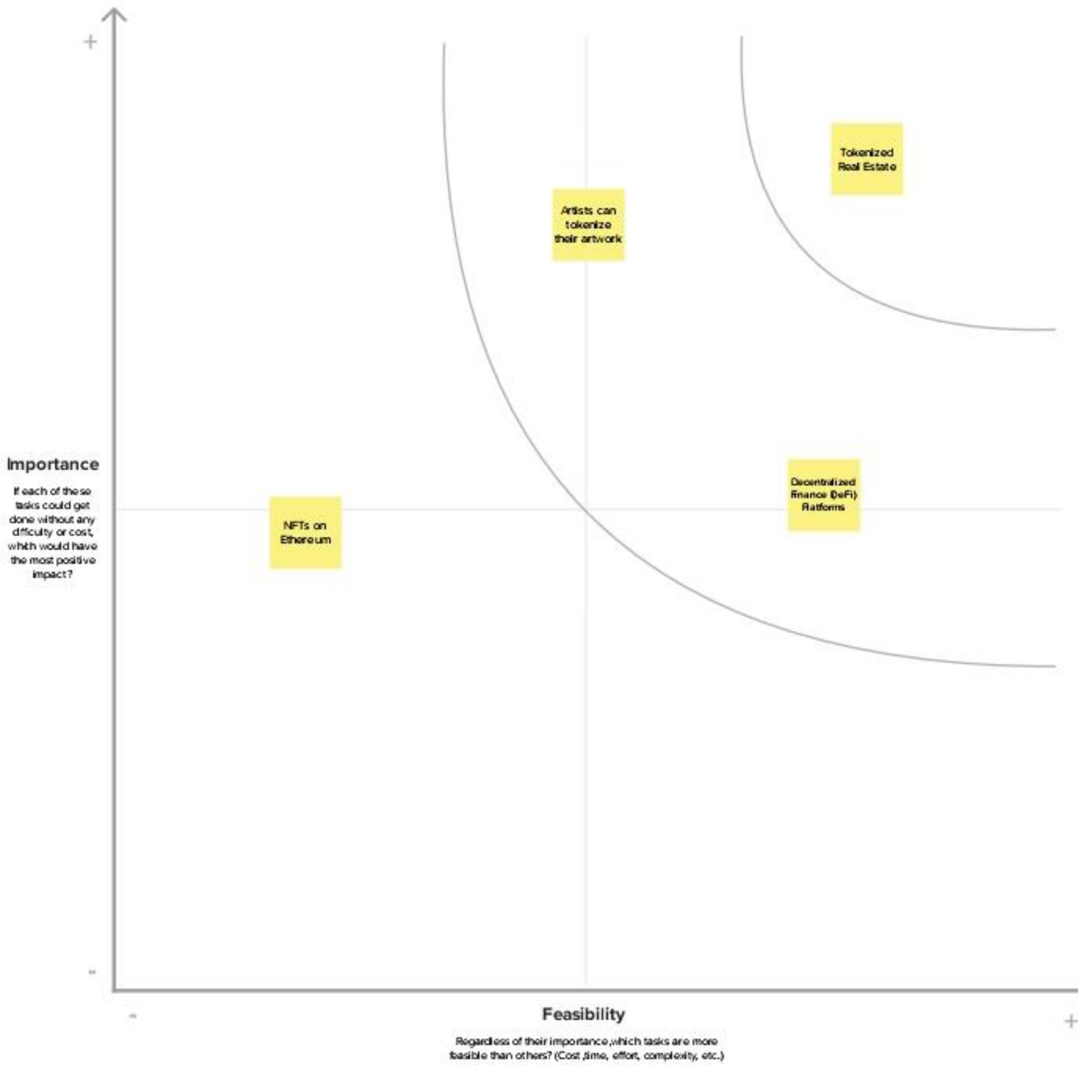
ractional ownership and easier property transactions.

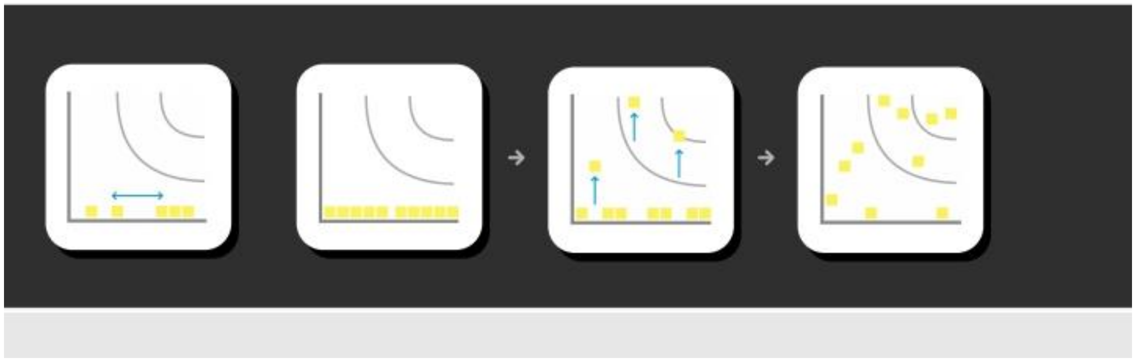
Decentralized Finance(DeFi) Platforms

Supply Chain Management

Develop decentralized lending

Digital Identity Management


+
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
-
Tokenized Real Estate
Artists can tokenize their artwork
Decentralized Finance (DeFi) Platforms
NFTs on Ethereum
-
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)
+



4. REQUIREMENT ANALYSIS

4.1Functional requirement

FR No.	Functional Requirement	
FR-1	Asset Creation and Registration	Users should be able to create and register digital assets on the Ethereum blockchain. Asset registration should include metadata such as title, description, author, date, and any other relevant information.
FR-2	Asset Storage and Encryption	Digital assets should be securely stored on the blockchain, with data encryption to protect their integrity and confidentiality.
FR-3	Asset Tracking and Metadata Management	Users should have the ability to update asset metadata, including tags, categories, and descriptions. The system should support searching and filtering assets based on metadata.
FR-4	Asset Ownership and Transfer	Assets should be associated with specific owners, and ownership should be transferable through blockchain transactions. Ownership transfers should be securely recorded on the blockchain.
FR-5	Access Control and Permissions	Define access control and permissions for asset viewing, editing, and transfer. Implement role-based access control (RBAC) for different users or user groups.
FR-6	Smart Contracts:	Utilize smart contracts for managing asset ownership, transfers, and permissions. Implement contract functionality for executing predefined rules and logic.
FR-7	Interoperability with Other Systems	Ensure interoperability with other digital asset management systems or blockchain platforms. Support importing and exporting assets and metadata to and from other systems.
FR-8	Content Preview and Playback	Provide the ability to preview or play digital assets directly within the system. Ensure compatibility with various file formats and viewers.
FR-9	Reporting and Analytics	Generate reports on asset usage, ownership changes, and access patterns. Provide analytics to help users understand asset performance and trends.
FR-10	Auditing and Compliance	Maintain an audit trail of all asset-related activities. Ensure compliance with relevant data protection and copyright regulations.

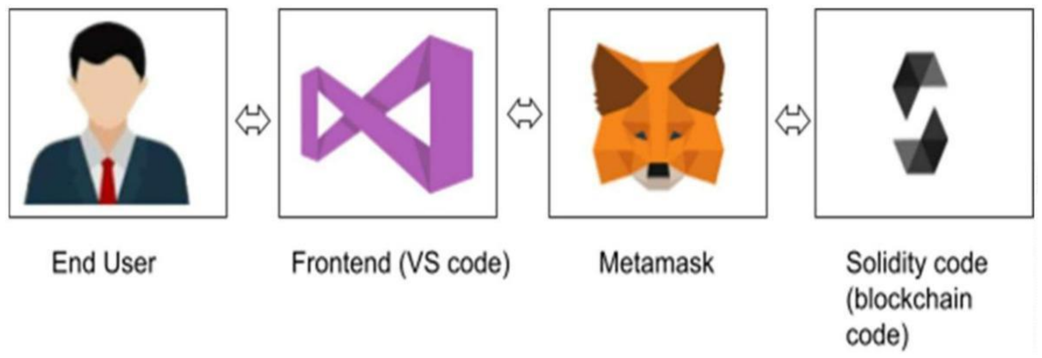
4.2 Non-Functional requirements

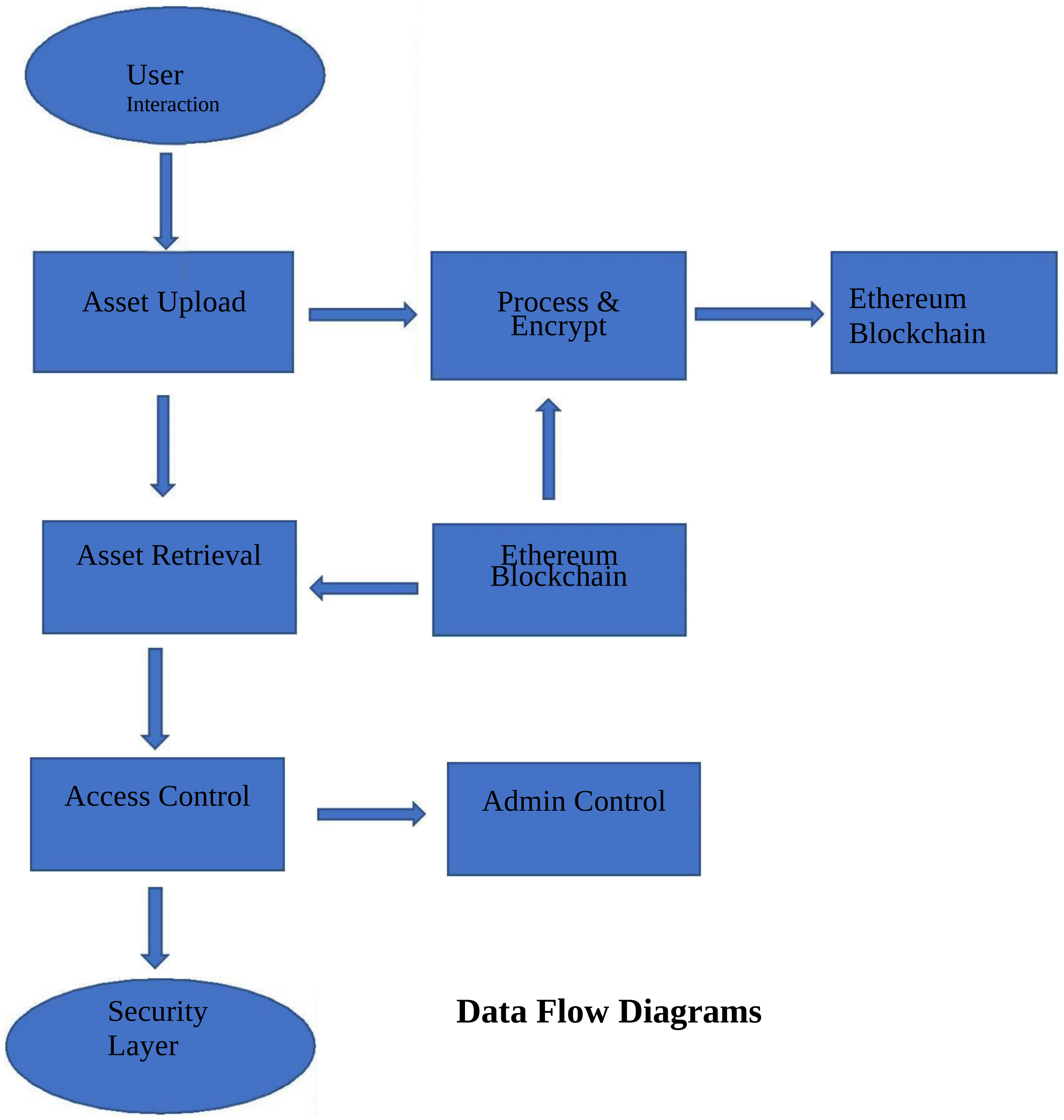
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	A user-friendly and intuitive interface is central to the usability of our Ethereum blockchain-based digital asset management system. With a clean and easy-to-navigate design, users can seamlessly upload, organize, and access their digital assets. We prioritize user onboarding through informative tutorials and tooltips to guide users. Powerful search and filtering options simplify asset discovery, while in-system preview and playback capabilities enhance the user experience. Clear access control settings, mobile responsiveness, and responsive customer support further contribute to a user-centric design, ensuring that our system aligns with diverse user needs and preferences. Usability remains a key focus, with regular feedback mechanisms in place to continually enhance the user experience
NFR-2	**Security**	The system leverages the Ethereum blockchain's immutability for asset data integrity and uses strong encryption for confidentiality. Access controls, authentication, and authorization mechanisms ensure only authorized users access and manage assets. Smart contracts automate secure asset transfers, and Ethereum's consensus mechanisms enhance transaction security. Regular security audits, regulatory compliance, incident response planning, and user education collectively create a robust security framework to safeguard digital assets and user data.
NFR-3	**Reliability**	the blockchain's immutable ledger, the system maintains unalterable records of digital asset ownership and transactions, ensuring data integrity. High availability minimizes system downtime, assuring users of consistent access. Regular data backups and a well-defined recovery plan provide reliability in safeguarding assets and data, allowing for swift restoration in the event of data loss or system failures. Strong security measures, including authentication and encryption, enhance both the security and reliability of the DAM system, instilling trust in its performance and the protection of digital assets.

NFR-4	**Performance**	Users expect swift interactions, and the system must deliver. Asset retrieval, uploads, and search queries should be optimized to ensure users can work efficiently. Scalability is essential, as the system must maintain usability as the volume of digital assets and users grows. Consistent response times are paramount, and well-defined benchmarks must be met to uphold usability standards. Performance testing and regular system optimization are essential to ensure that the DAM system operates smoothly, regardless of the scale or usage patterns.
NFR-5	**Availability**	Users rely on consistent access to their digital assets, and the system's high availability ensures they can do so without disruptions. Minimal downtime for maintenance or updates is a key requirement to prevent operational interruptions. Redundant servers, data backup mechanisms, and disaster recovery plans are integral to maintaining this level of availability, safeguarding assets and user data. Users can trust that their assets are accessible when needed, thanks to a reliable system with a robust availability framework. This confidence in consistent access underpins the system's integrity and user satisfaction.
NFR-6	**Scalability**	As the volume of digital assets and users grows, the system must gracefully adapt without compromising performance. Scalability is achieved through an architecture that can handle increased demands, such as asset uploads, downloads, and search queries, while maintaining response times. It ensures that the DAM system can scale up or out as necessary, accommodating user growth and asset expansion without significant degradation in usability. Scalability testing is a key practice to verify the system's ability to handle peak loads and maintain a seamless user experience. A scalable DAM system provides room for growth and enhances overall user satisfaction

5. PROJECT DESIGN

5.1Data Flow Diagrams & User Stories





Data Flow Diagrams

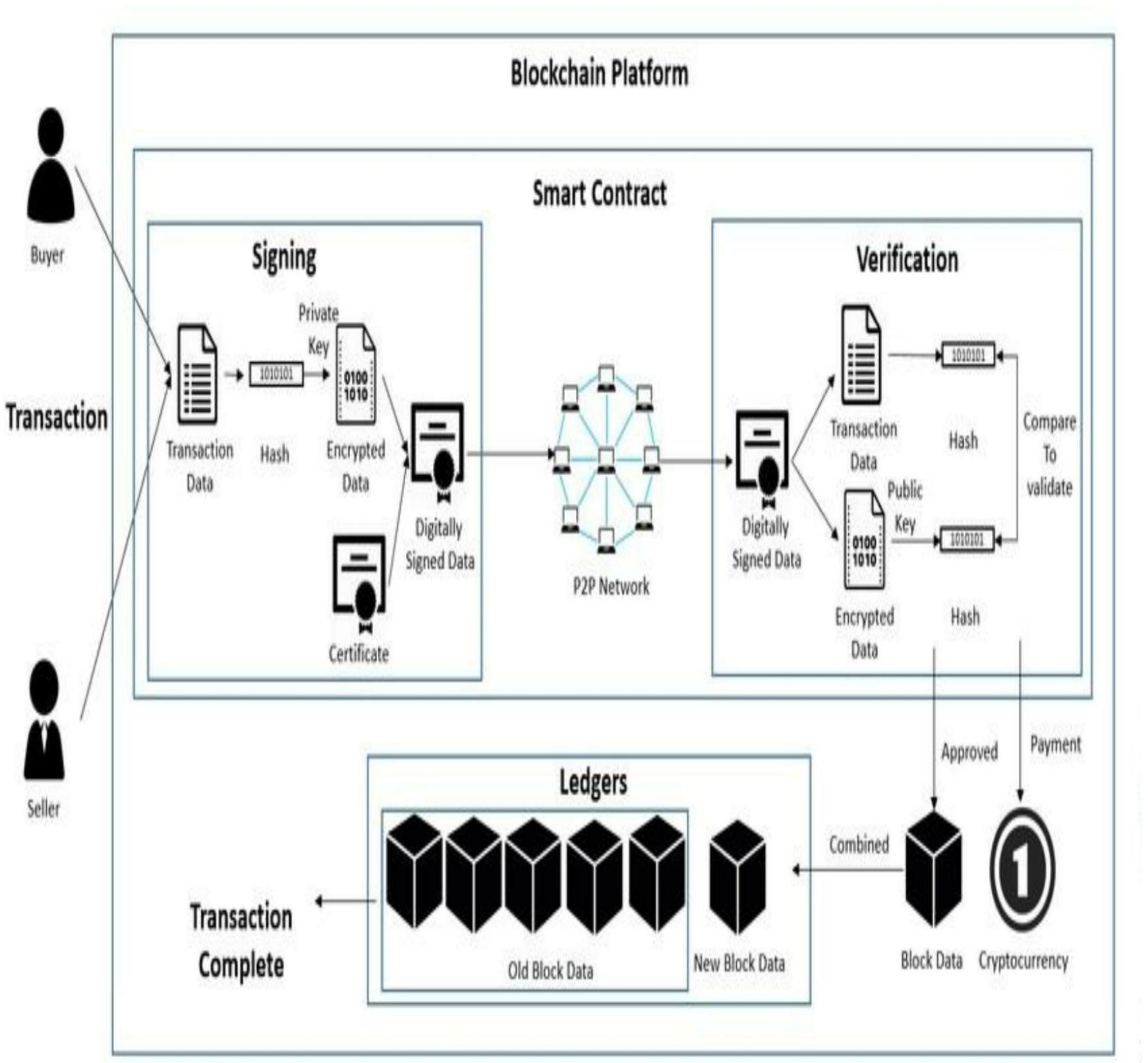
User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Team Member
Content Creator	Asset Upload and Management	USN-1	As a content creator, I want to upload multiple image and video assets to the DAM system with drag-and-drop functionality.	Users should be able to drag and drop multiple assets onto the DAM interface, and the system should process and upload them efficiently.	High	John (Frontend Developer)
Content Creator	Asset Upload and Management	USN-2	As a content creator, I need to add detailed metadata to my assets, including titles, descriptions, and copyright information, to keep them well-organized.	Metadata fields should be easily accessible and editable, and changes should be immediately reflected in asset information.	High	John (Frontend Developer)
Marketing Manager	Asset Organization and Access Control	USN-3	As a marketing manager, I want to create and assign tags to assets for easy categorization, facilitating efficient asset retrieval.	Tags should be customizable, and assets should be sortable and filterable by assigned tags.	Medium	Sarah (Product Owner)
Marketing Manager	Asset Organization and Access Control	USN-4	As a marketing manager, I need to restrict access to confidential assets to authorized team members only.	Access control settings should allow me to specify who can view, edit, and delete assets, with permissions easily adjustable.	Medium	Sarah (Product Owner
Administrator	System Management	USN-5	As an administrator, I want to monitor and manage asset access, user roles, and system performance.	The admin dashboard should provide insights into user activity, allow role assignments, and offer system performance metrics.	High	David (SysAdmin)
Administrator	System Management	USN-6	As an administrator, I need to set up automated data backups and a disaster recovery plan for data safety.discipline manner without	The system should regularly back up data and provide a documented	High	David (SysAdmin)

any issues

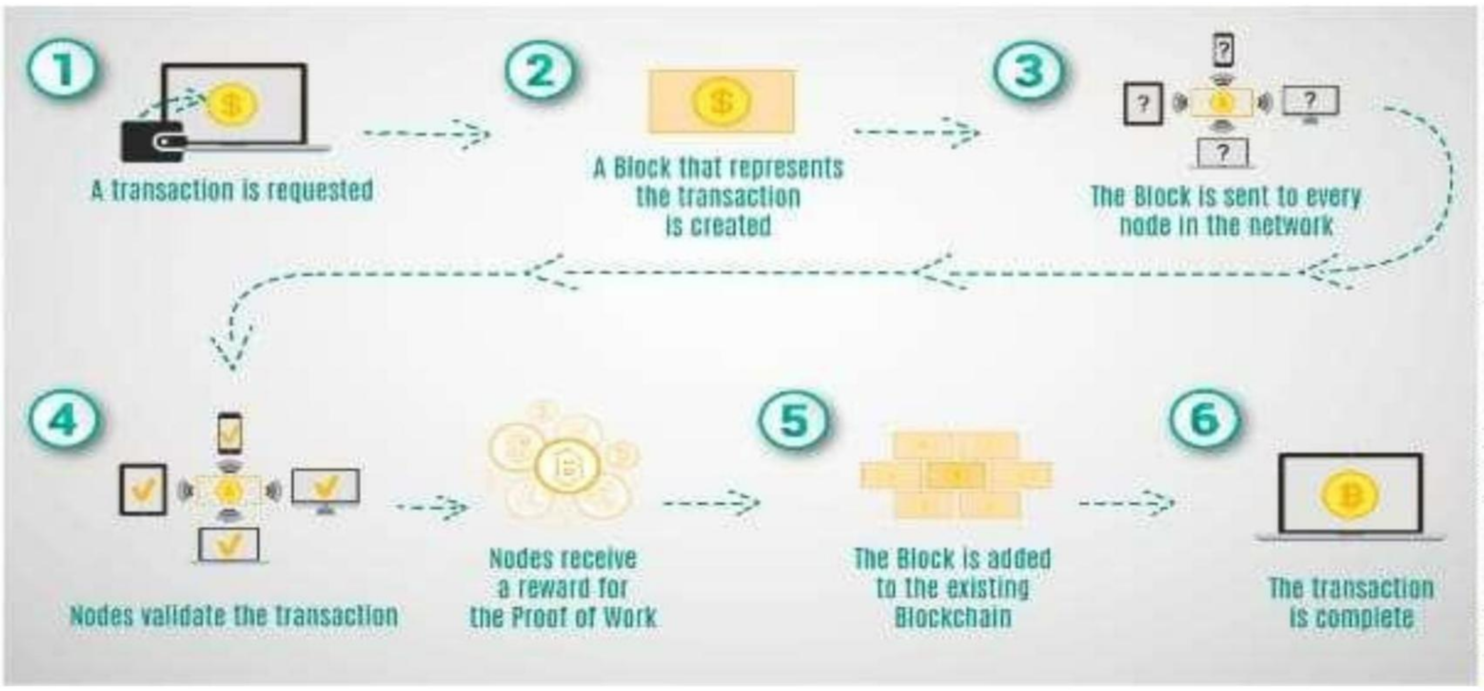
recovery plan to prevent data loss.

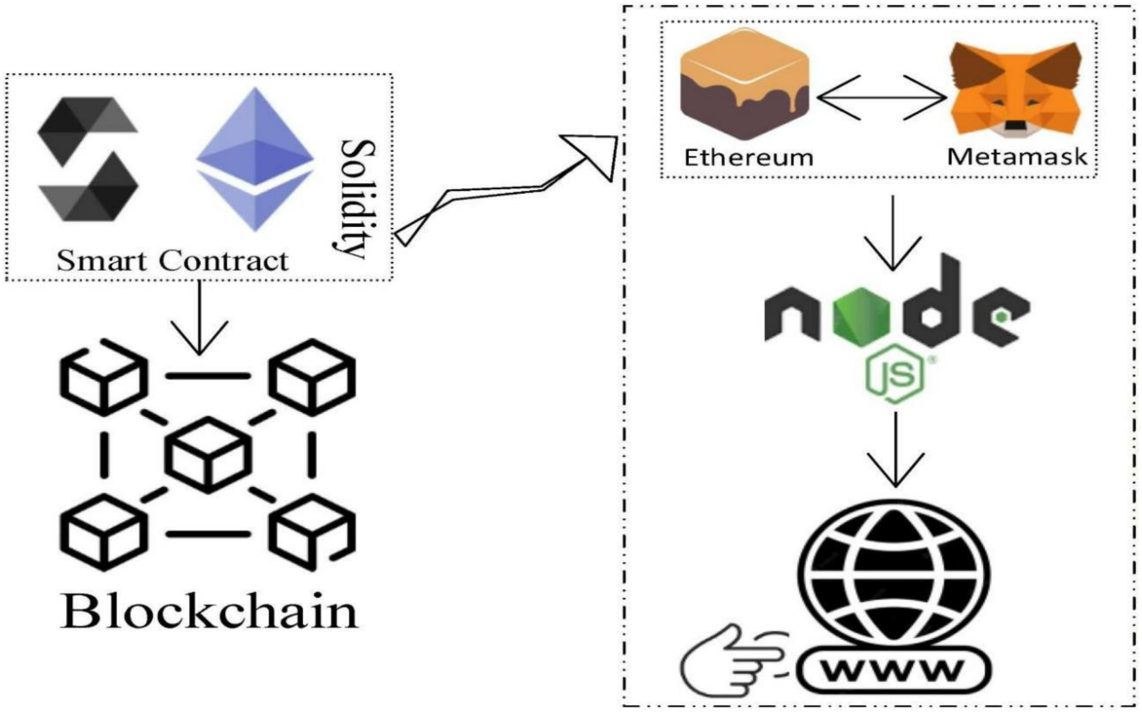
5.2Solution Architecture



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture


1
A transaction is requested
2
A Block that represents the transaction is created
3
The Block is sent to every node in the network
4
Nodes validate the transaction
Nodes receive a reward for the Proof of Work
5
The Block is added to the existing Blockchain
6
The transaction is complete



6.2 Sprint Planning & Estimation

1. User Story Backlog:

Start by creating a backlog of user stories. These stories should represent the features, enhancements, or tasks needed for your DAM system. Ensure they are well-defined, with clear acceptance criteria.

2. Prioritization:

Collaborate with stakeholders to prioritize the user stories based on their importance and impact. High-priority stories should be at the top of the backlog.

3. Sprint Planning Meeting:

Hold a sprint planning meeting with your development team. During this meeting, select a set of user stories from the backlog to work on during the upcoming sprint. Consider the team's capacity and the complexity of the stories.

4. Story Point Estimation:

Use a method like story point estimation to estimate the effort required for each user story. The team assigns relative points to stories to indicate their complexity. This helps in determining how many stories can be included in the sprint.

5. Sprint Goal:

Define a clear sprint goal, which should align with the project's objectives. The goal should provide a sense of purpose for the sprint.

6. Daily Stand-Ups:

Conduct daily stand-up meetings to keep the team updated on progress, discuss any challenges, and make necessary adjustments to the sprint plan.

7. Sprint Review:

At the end of the sprint, hold a sprint review meeting to showcase the completed work to stakeholders. Gather their feedback and insights.

8. Sprint Retrospective:

After the review, conduct a sprint retrospective to assess what went well and what could be improved. Use this feedback to make process enhancements for the next sprint.

9. Continuous Improvement:

Agile principles emphasize continuous improvement. Apply lessons learned from each sprint to refine the process, including better estimation and planning.

10. Blockchain Integration Considerations:

When estimating and planning, consider the complexities related to blockchain integration, such as smart contract development, security measures, and the use of Ethereum's capabilities.

6.3 Sprint Delivery Schedule

- Divide the Project into Sprints:
  Begin by dividing the overall DAM project into sprints. Sprints are time-bound iterations, usually lasting 2-4 weeks, during which specific sets of features or tasks are completed.

- Prioritize User Stories:
  Review the prioritized user stories from your backlog and select those that will be addressed in each sprint. Ensure that each sprint has a clear focus and goal.

- Define Sprint Durations:
  Decide on the duration of each sprint. Agile sprints are typically 2-4 weeks long, but you can choose the duration that works best for your team and project.

- Create a Sprint Backlog:
  For each sprint, create a sprint backlog that includes the user stories, tasks, and features that will be tackled during that sprint.

- Assign Story Points:
  Estimate the effort required for each user story in the sprint backlog using story points or other estimation methods. This helps in understanding the capacity of the sprint.

- Distribute Workload:
  Based on the team's capacity and story point estimates, distribute the workload evenly across the sprint backlog items. Ensure that the team can realistically complete the planned work during the sprint.

- Define Milestones:
  Within each sprint, set specific milestones or checkpoints for key tasks or features. This helps in tracking progress and ensuring that the team is on target.

- Adjust for Blockchain Integration:
  Consider the complexities of blockchain integration in your delivery schedule. Tasks related to smart contract development, security testing, and Ethereum-specific considerations should be accounted for.

- Iterative Development:
  Remember that in Agile development, work is delivered incrementally. At the end of each sprint, you should have a potentially shippable product increment.
- Continuous Review and Adaptation:
  After each sprint, hold sprint reviews and retrospectives to gather feedback, evaluate progress, and make necessary adjustments to the delivery schedule or project priorities.

- Release Planning:
  Based on the progress in each sprint and the feedback received, plan releases of the DAM system. These releases can be scheduled according to the completion of major features or project milestones.

- Maintain a Release Calendar:
  Maintain a release calendar that outlines when each sprint's deliverables or major releases are expected to be available to users or stakeholders. Share this calendar with the team and relevant parties.

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

7.1 Feature 1

Register Asset

- The "Asset" struct represents the properties of a digital asset, including title, description, IPFS hash, and the owner's Ethereum address.
- The "assets" array stores registered assets.
- The "registerAsset" function allows a user to register a new asset by providing the title, description, and the IPFS hash of the asset data. It also records the user's Ethereum address as the owner.
- You can emit an event to log the asset registration, providing information about the newly registered asset.

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 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);
    }
```

7.2Feature 2

<u>Trasfor Ownership</u>

- The "Asset" struct represents the properties of a digital asset, including title, description, IPFS hash, and the owner's Ethereum address.
- The "assets" array stores registered assets.
- The "registerAsset" function allows a user to register a new asset, which is owned by the user who registers it.
- The "transferOwnership" function lets the owner of an asset transfer ownership to another user by specifying the asset's index and the address of the new owner.

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 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);
    }
}
```

7.3 Database Schema (if Applicable)

The traditional database schema is replaced with smart contracts, which define the structure and behavior of assets and related data on the blockchain. However, certain off-chain data and metadata may be stored in traditional databases or decentralized storage systems for efficiency and scalability. Below, I'll provide a high-level overview of how the database schema for a DAM system on the Ethereum blockchain might look:

On-Chain Ethereum Smart Contracts:

AssetContract:

- Attributes:
  Asset title (string)
  Asset description (string)
  IPFS hash for asset data (string)
  Owner (address) Public status (boolean)

- Functions:
  Register asset
  Transfer ownership
  Toggle public status

Off-Chain Metadata Storage (Traditional Database or Decentralized Storage):

1.User Profiles:

- Attributes:
  User ID
  Ethereum address
  Username Email
  Other user-specific details

2.Audit Trail:

- Attributes:
  Asset ID

Action
User performing the action
Timestamp
Details of the action

3.Asset Metadata:

- Attributes:
  Asset ID
  Asset metadata
  Additional details

This off-chain storage can include a traditional relational database for user profiles and audit trail data or decentralized storage systems like IPFS for storing asset metadata and potentially even the asset files themselves.

It's important to note that the Ethereum blockchain is primarily used for storing critical asset ownership and transaction data, ensuring immutability and transparency. Off-chain storage is often used to handle less critical data and to optimize data access and retrieval times.

The database schema can vary significantly based on the specific requirements of the DAM system, and you may need to expand or modify the schema to suit your application's needs. The goal is to balance the benefits of blockchain's immutability with efficient data management and retrieval for users.

8. PERFORMANCE TESTING

8.1Performace Metrics

Asset Upload and Retrieval Speed:

Metric: Average time taken to upload and retrieve assets.
Importance: Measures the speed of asset management, ensuring quick access to assets.

Blockchain Transaction Throughput:

Metric: Transactions per second (TPS) on the Ethereum blockchain.
Importance: Indicates how well the system handles blockchain transactions, which is crucial for scalability.

Smart Contract Execution Time:

Metric: Average time taken for smart contract execution.
Importance: Evaluates the efficiency of the blockchain-based logic governing asset ownership and access.

Asset Metadata Search Time:

Metric: Time it takes to search for assets based on metadata.
Importance: Measures the responsiveness of the system's search functionality.

User Authorization Latency:

Metric: Time it takes to validate and authorize user access to assets.
Importance: Ensures that authorized users can access assets promptly while maintaining security.

Storage Space Usage:

Metric: Amount of blockchain storage used by assets and associated data.
Importance: Evaluates the cost and efficiency of storage on the blockchain.

Asset Accessibility Uptime:

Metric: Percentage of time assets are accessible.
Importance: Measures the system's reliability and availability for users.

Security Audit Findings:

Metric: Number and severity of security vulnerabilities discovered during audits.
Importance: Identifies potential risks and the need for security improvements.

User Feedback and Satisfaction:

Metric: User surveys or feedback on system usability and performance.
Importance: Provides insights into user satisfaction and areas for improvement.

Ethereum Network Gas Costs:

Metric: Total gas costs incurred for transactions and contract interactions.
Importance: Measures the cost-efficiency of system operations.

Scalability Metrics:

Metric: System's ability to handle an increasing number of users and assets.
Importance: Assesses how well the system can scale with growing demands.

Audit Trail Accuracy:

Metric: Accuracy and completeness of the audit trail for asset management.
Importance: Ensures a reliable record of asset history for compliance and accountability.

Data Backup and Recovery Time:

Metric: Time taken for data backup and recovery operations.
Importance: Evaluates the system's readiness for data recovery in case of failures.

9. RESULTS

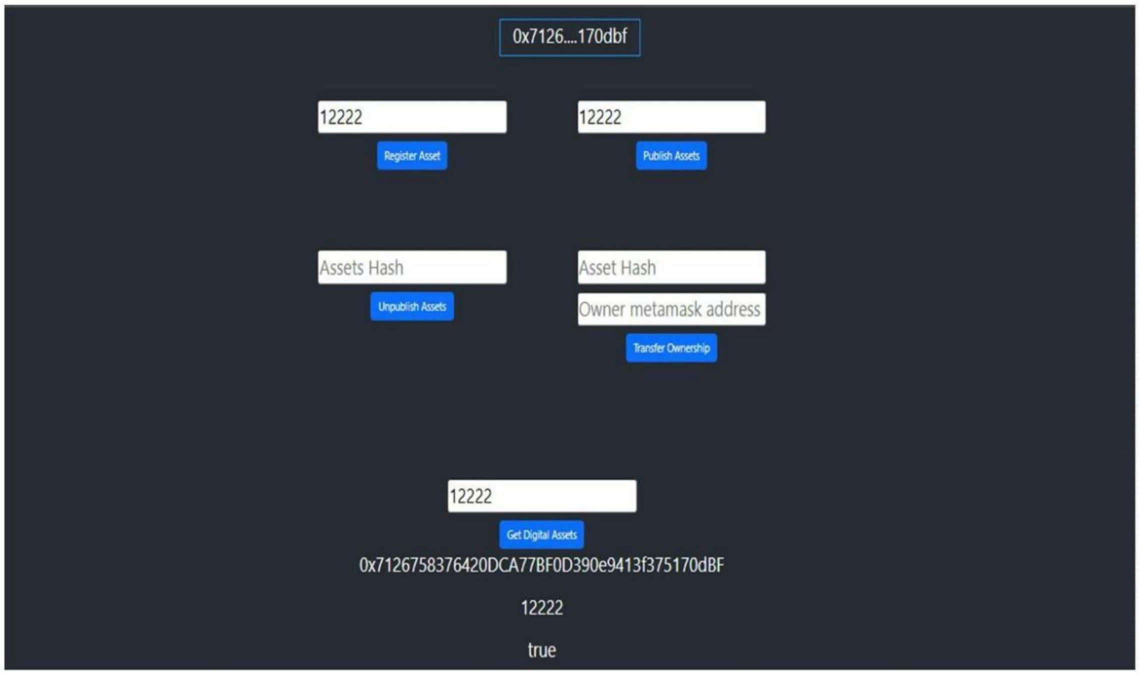
9.1Output Screenshots

The "register asset" feature allows users to officially record and store information about a digital asset on the Ethereum blockchain. This process establishes proof of ownership and a public record of the asset's existence.

"public asset" refers to a digital asset that is made publicly accessible to anyone without restrictions. It is marked as "public" and can be viewed or accessed by anyone on the Ethereum blockchain.

Get digital assets is showing a status of the assets (assets hash , owner address , publish or unpublish)

If system consist Publish assets output shows true

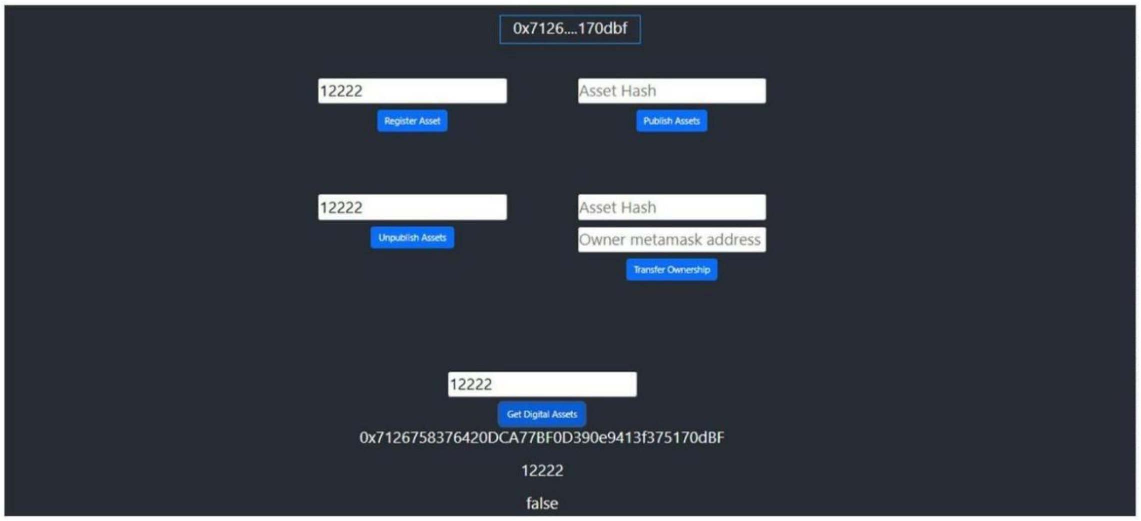


0x7126758376420DCA77BF0D390e9413f375170dBF

12222

true

If system consist “unpublish assets” output shows false

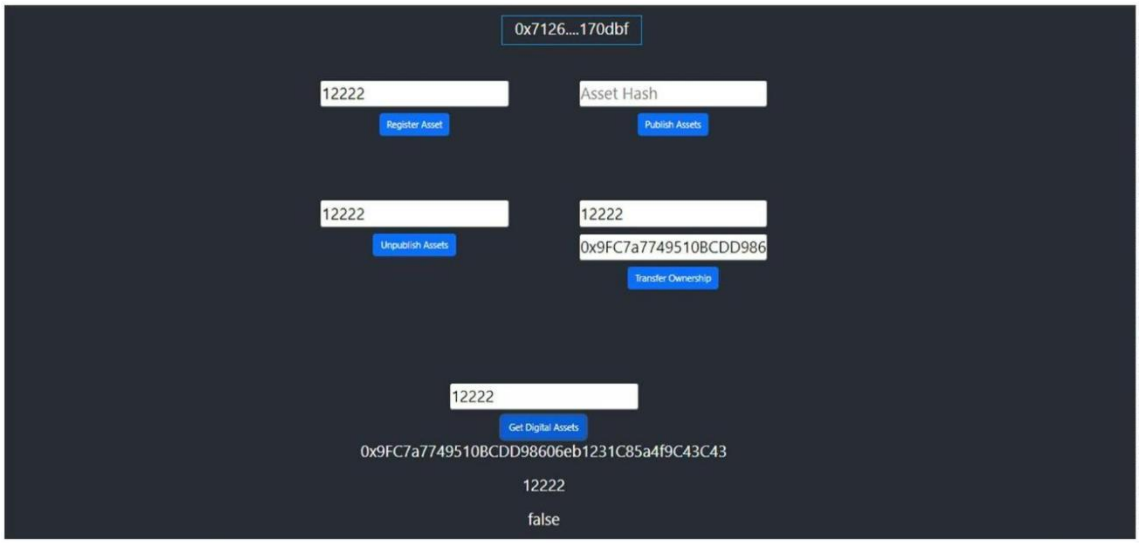


0x7126758376420DCA77BF0D390e9413f375170dBF

12222

false

"Transfer ownership" in the context of a digital asset management system on the Ethereum blockchain refers to the ability of the current owner of a digital asset to transfer ownership rights to another user. This feature allows asset owners to change the entity or individual that has control and ownership of a particular asset.



0x9FC7a7749510BCDD98606eb1231C85a4f9C43C43

12222

false

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- Immutability and Transparency: All asset transactions and ownership changes are recorded on the Ethereum blockchain, providing an immutable and transparent audit trail. This enhances trust and security.
- Ownership Verification: Ethereum's smart contracts allow for clear ownership verification, reducing disputes and proving the authenticity of digital assets.
- Decentralization: The Ethereum blockchain operates on a decentralized network, reducing the risk of single points of failure and enhancing security and availability.
- Security: Assets stored on the blockchain benefit from robust cryptographic security, making it difficult for unauthorized parties to tamper with or steal assets.
- Global Accessibility: Ethereum is a global network, making digital assets accessible to a worldwide audience without geographic restrictions.
- Interoperability: Ethereum's compatibility with various standards and protocols allows for easy integration with other blockchain-based systems.
- Cost-Effective: Smart contracts can automate asset management processes, reducing the need for intermediaries and saving costs.

DISADVANTAGES

- Scalability: Ethereum has faced challenges related to network congestion and scalability, making it less suitable for high-frequency asset management.

- Gas Costs: Every operation on the Ethereum blockchain consumes gas (transaction fees), which can make frequent asset management expensive.

- User Experience: Interacting with the blockchain can be complex for non-technical users, leading to usability challenges.

- Regulatory Concerns: Blockchain-based assets may raise regulatory questions, particularly if they represent real-world assets or securities.

- Data Storage: Storing large digital assets directly on the blockchain can be inefficient and costly. Many assets are stored off-chain or on decentralized storage systems like IPFS.

- Irreversible Transactions: Once a transaction is confirmed on the Ethereum blockchain, it is irreversible. Mistakes can be costly.

- Smart Contract Vulnerabilities: Poorly written smart contracts can lead to security vulnerabilities and hacks, resulting in loss of assets.

- Privacy: The Ethereum blockchain is public, which means that asset data is visible to anyone. For private assets, additional privacy measures are needed.

11. CONCLUSION

Implementing digital asset management on the Ethereum blockchain offers advantages such as immutability, transparency, and decentralized ownership, making it a secure and transparent solution for asset management. However, challenges related to scalability, transaction costs, and regulatory compliance must be carefully addressed to fully realize the potential of this technology. As blockchain technology continues to evolve, Ethereum-based digital asset management holds promise for revolutionizing how we manage and exchange digital assets in various industries.

12. FUTURE SCOPE

The future scope of digital asset management (DAM) on the Ethereum blockchain is exciting and holds considerable potential for innovation and growth.

- Interoperability with Other Blockchains: Future DAM systems may explore interoperability with other blockchain networks, enabling cross-chain asset transfers and interactions. This could expand the reach of digital assets and create a more interconnected blockchain ecosystem.
- DeFi Integration: Integration with decentralized finance (DeFi) platforms could allow for more advanced financial operations involving digital assets, such as lending, borrowing, and earning interest. This would add a financial dimension to digital asset management.
- NFT Enhancements: Non-fungible tokens (NFTs) are a key component of DAM on Ethereum. Future developments may focus on enhancing the functionality of NFTs, such as enabling fractional ownership, composite NFTs, and more interactive experiences.
- Cross-Platform Compatibility: DAM systems may offer seamless compatibility with various platforms and devices, ensuring a consistent user experience on web, mobile, and desktop applications.
- Decentralized Identity: The integration of decentralized identity solutions could enhance user authentication, privacy, and security in DAM systems. Users may have greater control over their identity and data.
- Scaling Solutions: Overcoming Ethereum's scalability challenges is crucial. The adoption of Layer 2 scaling solutions, like Optimistic Rollups and zk-Rollups, can significantly improve transaction throughput and reduce costs

13. APPENDIX

Source Code

Solidity coding :

```
// 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);
    }
}
```

Java script :

```
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": "", "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"
 },
 {
  "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"
 }
]

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 = "0xC52C1b299a6419CEFaee21e5B49fA808003911Ce"

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

<u>HTML coding</u> :

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the
build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>React App</title>
  </head> <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

GitHub :
https://github.com/SarathiRavichandran/Nasn-mudhalvanon-the-Ethereum-Blockchain

Project Video Demo Link

https://github.com/SarathiRavichandran/Nasn-mudhalvanon-the-Ethereum-Blockchain