

# Project Report

## 1. INTRODUCTION

### 1.1 Project Overview

The Blockchain-Based Smart Real Estate Management System is a forward-looking technological solution designed to streamline and enhance various aspects of the real estate industry. By leveraging blockchain technology, this project aims to provide transparency, efficiency, and trust in property transactions, property management, and ownership.

### 1.2 Purpose

The purpose of the Blockchain-Based Smart Real Estate Management System is to revolutionize and modernize the real estate industry by leveraging blockchain technology and smart contracts. The Blockchain-Based Smart Real Estate Management System has the overarching purpose of transforming the real estate industry by making it more transparent, efficient, and accessible while enhancing security and trust in property transactions and management. It aligns with the technological advancements of the digital age and the evolving demands of real estate stakeholders.

## 2. LITERATURE SURVEY

### 2.1 Existing problem

The existing problems in the traditional real estate industry provide the impetus for the development of a Blockchain-Based Smart Real Estate Management System.

**Lack of Transparency:** Real estate transactions often lack transparency, making it difficult for buyers, sellers, and investors to access complete and accurate information about a property's history, condition, and ownership.

**High Transaction Costs:** Real estate transactions involve significant transaction costs, including fees for brokers, legal services, and other intermediaries. These costs can be a barrier to entry for some buyers and investors.

**Slow and Complex Processes:** Traditional real estate transactions are often slow and complex, involving extensive paperwork, negotiations, and third-party approvals. This can lead to delays and frustration for all parties involved.

**Intermediary Dependence:** Real estate transactions depend heavily on intermediaries, such as brokers, agents, and escrow services. This reliance on intermediaries can lead to increased costs and potential conflicts of interest.

**Inefficient Property Management:** Property management processes, including rent collection, maintenance requests, and communication between property owners and tenants, can be inefficient and error-prone.

**Data Privacy and Security:** Sensitive property and owner data are vulnerable to data breaches and unauthorized access. Data privacy and security are major concerns in the traditional system.

**Limited Accessibility:** Access to the real estate market can be limited due to factors such as high investment requirements, geographical restrictions, and regulatory constraints.

## **2.2 References**

Blockchain Technology and Real Estate:

M. Beconi, J. Gilbert, and L. Wright, "Blockchain in Commercial Real Estate," RICS Research, 2017.

Real Estate Tokenization:

A. Zevenhoven and S. Luger, "Real Estate Tokenization," European Journal of Business and Management, Vol.11, No.22, 2019.

Smart Contracts in Real Estate:

L. Fort, "The Role of Smart Contracts in Real Estate," International Conference on e-Democracy and Open Government, 2017.

Blockchain and Property Ownership:

M. Mougayar, "Property Ownership with Blockchain," Startup Management, 2018.

Decentralized Finance (DeFi) and Real Estate:

S. M. Podraza, "Real Estate and DeFi: A Blockchain Solution for Property Investment," IEEE Open Journal of the Computer Society, Vol.2, 2021.

Data Security and Privacy in Blockchain:

N. Kshetri, "Can Blockchain Strengthen the Internet of Things?," IT Professional, Vol.19, No.4, 2017.

Regulatory Considerations for Blockchain in Real Estate:

J. Wüst and G. Gervais, "Do You Need a Blockchain?," Financial Cryptography and Data Security, 2018.

Real Estate Management Software:

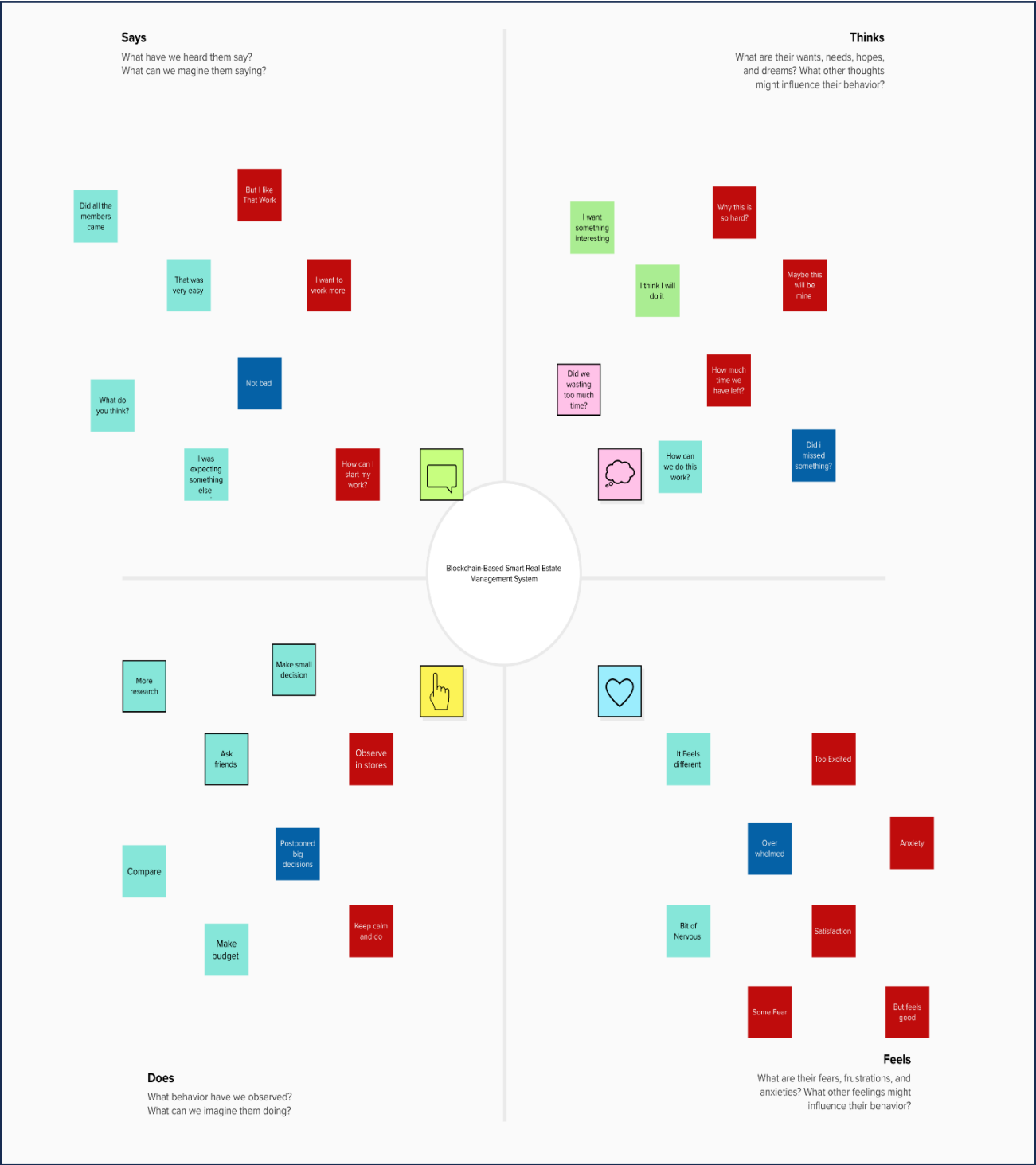
J. L. Gu, "A Comprehensive Analysis of Real Estate Management Software," The Open Construction and Building Technology Journal, Vol.14, 2020.

## **2.3 Problem Statement Definition**

Traditional real estate management systems are burdened with inefficiencies, lack transparency, and are vulnerable to fraud and errors. There is a pressing need for a Blockchain-Based Smart Real Estate Management System that can revolutionize the industry by offering secure, automated, and transparent solutions for property ownership, rentals, maintenance, and investment. This system will address the challenges of fragmented data, manual processes, and lack of trust, ultimately providing a seamless and trustable ecosystem for all stakeholders involved in real estate, ensuring smoother transactions, and maximizing the value of properties

### 3. IDEATION & PROPOSED SOLUTION


#### 3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming

Step-1: Team Gathering, Collaboration and Select the Problem Statement

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.

10 minutes to prepare

1 hour to collaborate

2-8 people recommended

**Before you collaborate**

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A

**Team gathering**

Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B

**Set the goal**

Think about the problem you'll be focusing on solving in the brainstorming session.

C

**Learn how to use the facilitation tools**

Use the Facilitation Superpowers to run a happy and productive session.

Open article →

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.

5 minutes

**Blockchain-Based Smart Real Estate Management System**

**PROBLEM**

Traditional real estate management systems are burdened with inefficiencies, lack transparency, and are vulnerable to fraud and errors. There is a pressing need for a Blockchain-Based Smart Real Estate Management System that can revolutionize the industry by offering secure, automated, and transparent solutions for property.

**Key rules of brainstorming**

To run a smooth and productive session

Stay in topic.

Encourage wild ideas.

Defer judgment.

Listen to others.

Go for volume.

If possible, be visual.

Share template feedback



## 4. REQUIREMENT ANALYSIS

### 4.1 Functional requirement

#### 1.Vs Code

Visual Studio Code (VS Code) is a versatile and popular code editor that can be used for developing blockchain-based smart real estate management systems. Here are some steps and tips for using VS Code in such a project:

##### **Installation:**

Download and install Visual Studio Code from the official website (<https://code.visualstudio.com/>).

##### **Extensions:**

Install relevant extensions for blockchain and smart contract development. Some commonly used extensions for Ethereum and Solidity development include "Solidity" and "Truffle."

##### **Smart Contract Development:**

Create a new folder for your project and open it in VS Code.

Write, test, and deploy your smart contracts using the Solidity extension.

Use Truffle, a development framework for Ethereum, to help you with smart contract development.

##### **Front-End Development:**

Develop the front-end of your real estate management system using web development technologies like HTML, CSS, and JavaScript.

Ensure that your front-end code interacts with your smart contracts on the blockchain.

##### **Testing:**

Write tests for your smart contracts and front-end code. VS Code can be used to run and manage test suites.

##### **Deployment:**

Deploy your smart contracts and front-end applications to the desired blockchain network and hosting platforms.

#### 2. Nodejs

Node.js is a popular runtime environment for JavaScript, and it can be used in the development of a Blockchain-Based Smart Real Estate Management System, particularly for building the backend server, handling API requests, and interacting with the blockchain. Here's how Node.js can be integrated into the development process:

##### **Backend Development:**

Node.js is commonly used to build the backend server of web applications. You can create an Express.js server to handle API requests from the front end and interact with the blockchain network (e.g., Ethereum).

### **Smart Contract Deployment:**

You can use Node.js scripts to automate the deployment of smart contracts to the blockchain network during the development and testing phases.

### **Authentication and Authorization:**

Implement user authentication and authorization using Node.js middleware, such as Passport.js, to secure **your application and control access to certain features.**

### **Real-Time Features:**

For real-time features like property bidding, chat, or notifications, you can use Node.js with WebSocket libraries like Socket.io to provide instant updates to users.

## **4.2 Non-Functional requirements**

### **1. Metamask**

MetaMask is a popular cryptocurrency wallet and decentralized application (dApp) browser extension that allows users to interact with the Ethereum blockchain. It serves as both a cryptocurrency wallet and a gateway to the world of decentralized applications.

MetaMask has gained popularity for its role in enabling users to access and participate in the decentralized finance (DeFi) ecosystem, interact with non-fungible tokens (NFTs), and securely manage their Ethereum-based assets

Integrating MetaMask into a Blockchain-Based Smart Real Estate Management System can offer several benefits, especially when dealing with Ethereum-based tokens, smart contracts, and decentralized applications (dApps).

**User Wallets:** Each user, including property owners, buyers, and investors, can have their own MetaMask wallet. This wallet allows them to securely store and manage property tokens, Ether (ETH), or any other cryptocurrencies used within the system.

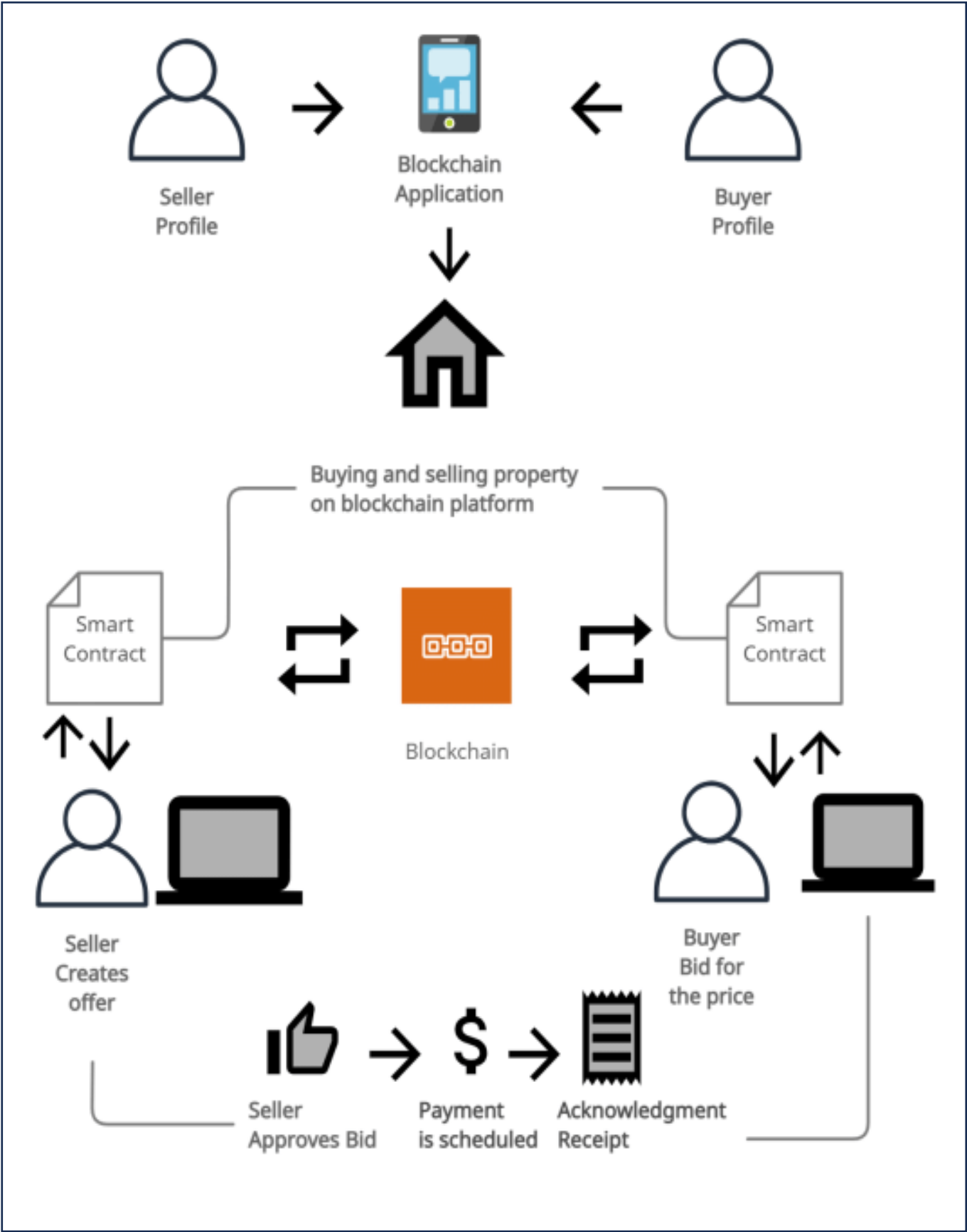
**Access to dApps:** MetaMask serves as a bridge to decentralized applications. Users can access the real estate management system's dApp through MetaMask, enabling them to initiate property transactions, view property details, and perform various management tasks.

**Transaction Execution:** Users can use MetaMask to interact with the smart contracts that govern property transactions. For example, they can initiate property purchases, sales, and transfers directly through their MetaMask wallets.

**Token Management:** Property tokens, which represent fractional ownership, can be managed within MetaMask. Users can view their token holdings, transfer tokens to other users, and participate in property investment activities.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User & Stories Solution Architecture





## **6. PROJECT PLANNING & SCHEDULING**

### **6.1 Technical Architecture**

The technical architecture of a Blockchain-Based Smart Real Estate Management System is crucial for its functionality, security, and efficiency. The technical architecture of the Blockchain-Based Smart Real Estate Management System is a complex, interconnected system that leverages blockchain technology, smart contracts, user interfaces, and security measures to provide a transparent, efficient, and secure real estate management solution. Careful design and continuous development and maintenance are essential to ensure its success and effectiveness.

### **6.2 Sprint Planning & Estimation**

Sprint planning and estimation are crucial components of an agile development process for a Blockchain-Based Smart Real Estate Management System. They help the development team organize work, set priorities, and estimate the effort required for each task within a sprint. Here's how sprint planning and estimation can be approached:

#### **1. Product Backlog:**

Begin with a well-defined product backlog. This backlog includes all the features, user stories, and tasks that need to be implemented in the system. It should be maintained and prioritized by the product owner.

#### **2. Sprint Goal:**

Determine the sprint goal for the upcoming sprint. This goal should be aligned with the overall project objectives and should specify what the team intends to achieve in the sprint.

#### **3. Sprint Length:**

Decide on the duration of the sprint. Common sprint lengths are 2 weeks, 3 weeks, or 4 weeks. Choose a sprint length that suits the team's work capacity and project complexity.

#### **4. Sprint Planning Meeting:**

Conduct a sprint planning meeting before the start of each sprint. This meeting involves the product owner, development team, and the scrum master. During the meeting, the team reviews the prioritized backlog items and selects the ones to be worked on in the upcoming sprint.

Sprint planning and estimation are iterative processes, and the team should continuously improve their estimation accuracy and sprint planning based on past performance and feedback. Effective sprint planning and estimation ensure that the Blockchain-Based Smart Real Estate Management System is developed efficiently and with a focus on delivering value to users.

### **6.3 Sprint Delivery Schedule**

The sprint delivery schedule for a Blockchain-Based Smart Real Estate Management System depends on the sprint length and the scope of work for each sprint. Typically, agile development teams follow a sprint schedule with fixed sprint durations (e.g., 2 weeks, 3 weeks, or 4 weeks). Here's an example of a sprint delivery schedule for a 2-week sprint:

Sprint 1:

Sprint Duration: Week 1 and Week 2

Sprint Planning: Day 1 of Week 1

Daily Standup Meetings: Held daily throughout the sprint

Mid-Sprint Review: Day 5 of Week 1

Sprint Review and Demo: Day 5 of Week 2

Sprint Retrospective: Day 5 of Week 2

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

### **7.1 Feature 1**

```
// Mapping to store property details with propertyId as the key
```

```
mapping(string => Property) public properties;
```

```
// Mapping to manage access control for properties, mapping addresses to propertyIds
```

```
mapping(address => mapping(string => bool)) public hasAccess;
```

```
// Events for logging property-related actions
```

```
event PropertyAdded(
```

```
    string indexed propertyId,
```

```
    string name,
```

```
    string location,
```

```
    address indexed owner
```

```
);
```

```
event PropertyTransferred(
```

```
    string indexed propertyId,
```

```
    address indexed from,
```

```
    address indexed to
```

```
);
```

```
// Modifier to check if the caller has access to a specific property
```

```
modifier hasPropertyAccess(string memory propertyId) {
```

```
    require(
```

```

        hasAccess[msg.sender][propertyId],
        "You don't have access to this property"
    );
    _;
}

```

```

// Function to add a new property to the system, accessible only by the owner
function addProperty(
    string memory propertyId,
    string memory name,
    string memory location,
    string memory _description

```

## 7.2 Feature 2

```

external onlyOwner {
    // Check if the property with the given propertyId doesn't already exist
    require(
        bytes(properties[propertyId].propertyId).length == 0,
        "Property already exists"
    );

    // Create a new Property structure and initialize its values
    properties[propertyId] = Property({
        propertyId: propertyId,
        name: name,
        location: location,
        discription: _description,
        currentOwner: owner
    });
    // Grant the owner access to the property

```

```
hasAccess[owner][propertyId] = true;
// Emit an event to log the addition of the property
emit PropertyAdded(propertyId, name, location, owner);
}
```

```
// Function to transfer ownership of a property to a new owner
```

```
function transferProperty(
```

```
    string memory propertyId,
```

```
    address newOwner
```

```
) external hasPropertyAccess(propertyId) {
```

```
    // Check if the new owner's address is valid
```

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

```
    // Get the current owner's address
```

```
    address currentOwner = properties[propertyId].currentOwner;
```

```
    // Update the currentOwner of the property to the new owner
```

```
    properties[propertyId].currentOwner = newOwner;
```

```
    // Update access control to revoke access from the current owner and grant access to the
new owner
```

```
    hasAccess[currentOwner][propertyId] = false;
```

```
    hasAccess[newOwner][propertyId] = true;
```

```
    // Emit an event to log the property transfer
```

```
    emit PropertyTransferred(propertyId, currentOwner, newOwner);
```

```
}
```

```
// Function to retrieve property details (name, location, and current owner) by propertyId
```

```
function getPropertyDetails(
```

```
    string memory propertyId
```

```
) external view returns (string memory, string memory, address) {
```

```
Property memory prop = properties[propertyId];  
return (prop.name, prop.location, prop.currentOwner);  
}  
}
```

## **8. PERFORMANCE TESTING**

### **8.1 Performace Metrics**

When evaluating the performance of a Blockchain-Based Smart Real Estate Management System, it's important to consider various metrics to assess its effectiveness and efficiency. These metrics can help gauge the system's impact, user experience, and overall success. Transaction Throughput: Measure the number of real estate transactions processed per second or per minute. Higher throughput indicates better system performance.

Transaction Confirmation Time: Calculate the time it takes for a real estate transaction to be confirmed on the blockchain. Lower confirmation times are favorable, as they reduce delays in property transactions.

Scalability: Assess how well the system can handle an increasing number of users and transactions. Scalability metrics should indicate that the system can grow to accommodate a larger user base without a significant drop in performance.

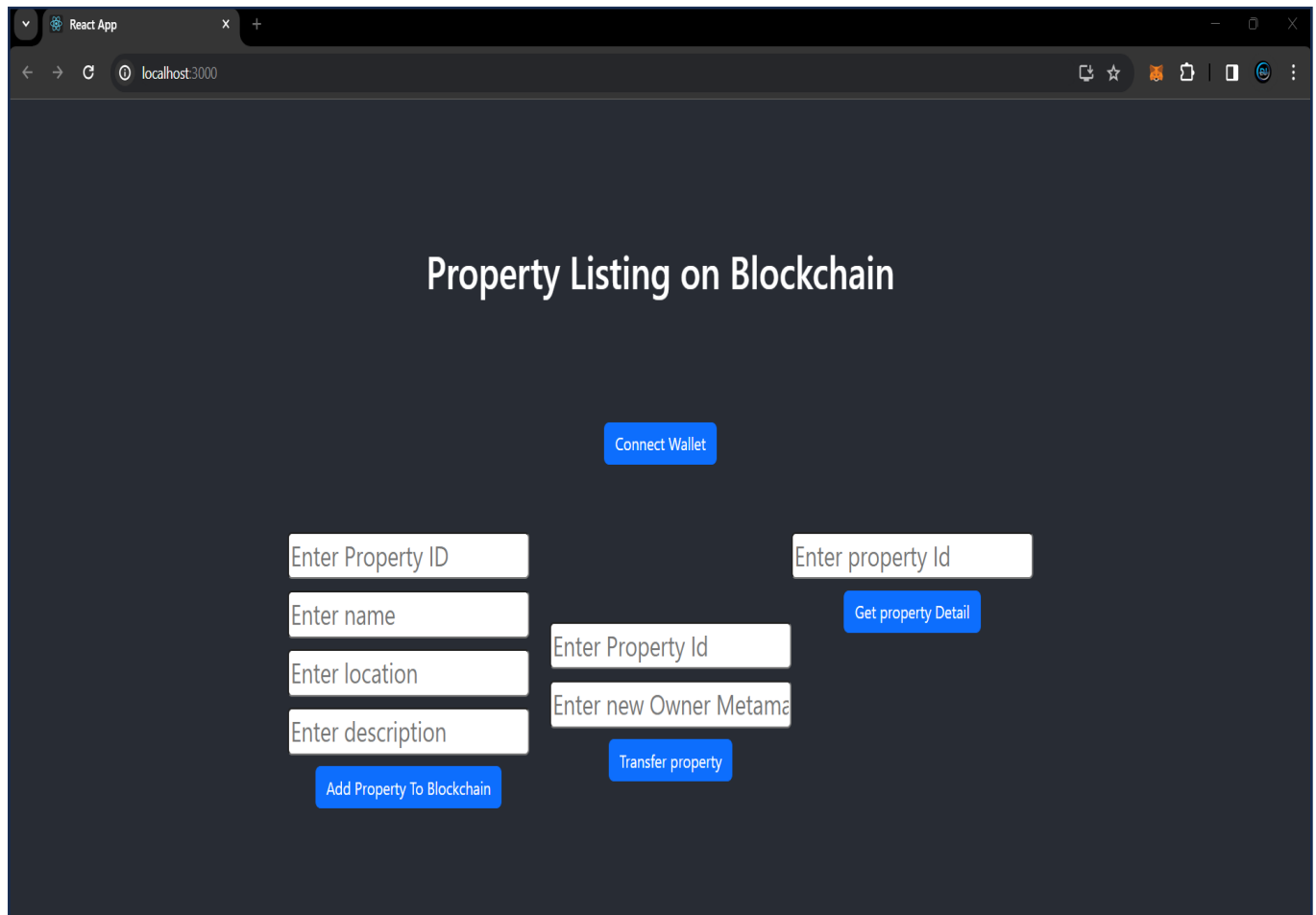
Cost Reduction: Evaluate the cost savings achieved through the elimination of intermediaries and streamlining of processes. Compare the cost of using the blockchain system to traditional real estate transaction costs.

Smart Contract Efficiency: Evaluate the efficiency of smart contracts in automating real estate transactions. Measure the number of successful, error-free smart contract executions.

Property Record Accuracy: Assess the accuracy of property records stored on the blockchain. Measure the number of disputes or discrepancies related to property ownership or history.

## 9. RESULTS

### 9.1 Output Screenshots



## 10. ADVANTAGES & DISADVANTAGES

### Advantages:

Transparency and Trust:

Blockchain provides a transparent and immutable ledger, reducing the potential for fraud and disputes in real estate transactions.

Reduced Intermediaries:

Eliminating intermediaries like brokers and escrow services can reduce costs and streamline the buying and selling process.

Automated Contracts:

Smart contracts automate property transactions, reducing paperwork and the need for third-party legal services.

#### Efficiency:

Real-time data updates, automated processes, and streamlined workflows enhance operational efficiency.

#### Security:

Strong encryption and decentralization make blockchain-based systems highly secure, reducing the risk of data breaches and fraud.

#### Global Accessibility:

Blockchain systems are accessible from anywhere, enabling international real estate investments and management.

#### Immutable Records:

Immutable records of property history, including renovations and maintenance, provide a clear picture of property conditions.

#### Cost Savings:

Reduced administrative costs, lower fees, and the elimination of paper-based processes lead to cost savings.

#### Sustainability:

Blockchain can support eco-friendly practices by tracking energy consumption and sustainability measures in real estate properties.

### **Disadvantages:**

#### **Scalability:**

Scalability issues can arise as the blockchain network grows, leading to slower transaction times and higher costs.

#### **Technical Complexity:**

Developing and maintaining blockchain-based systems requires specialized technical expertise, which can be expensive and in short supply.

#### **Dependency on the Internet:**

A blockchain-based system relies on the internet and can be disrupted by outages or cyberattacks.

## 11. CONCLUSION

In conclusion, a Blockchain-Based Smart Real Estate Management System holds great promise for revolutionizing the real estate industry by leveraging blockchain technology and smart contracts. This innovative approach offers transparency, efficiency, and security in property transactions and management.

In spite of these challenges, the advantages of a Blockchain-Based Smart Real Estate Management System are significant. It offers transparency, trust, efficiency, and reduced costs. Property tokenization allows fractional ownership and liquidity, and smart contracts automate transactions, reducing paperwork and reliance on intermediaries.

To successfully implement a Blockchain-Based Smart Real Estate Management System, collaboration between the technology and real estate sectors is essential. It's also crucial to work closely with legal experts to navigate regulatory complexities.

While the path to widespread adoption may be challenging, the potential for positive disruption and transformation in the real estate industry is undeniable. As blockchain technology matures and the industry becomes more accustomed to its benefits, it's likely that we will see more innovative and efficient approaches to real estate management and transactions.

## 12. FUTURE SCOPE

The future scope of a Blockchain-Based Smart Real Estate Management System is both promising and expansive. As the real estate industry continues to evolve, this technology offers various opportunities and developments:

**Widespread Adoption:** Increased awareness and acceptance of blockchain technology in the real estate sector will likely lead to widespread adoption. As more real estate professionals, property owners, and investors recognize the benefits of blockchain, it will become a standard part of the industry.

**Global Real Estate Markets:** Blockchain can facilitate international real estate transactions and investments. The ability to purchase property across borders using cryptocurrencies and smart contracts will open up new opportunities for investors.

**Fractional Ownership:** Property tokenization allows for fractional ownership, making it easier for individuals to invest in real estate without buying an entire property. This can democratize real estate investment and offer liquidity in traditionally illiquid assets.

**Smart Contracts for Rental Agreements:** Smart contracts can automate rental agreements, security deposits, and rent payments. Tenants and landlords can experience more seamless, transparent, and trustless rental processes.

**Property Records and Verification:** Blockchain can improve property record management and verification. Immutable records ensure the accuracy of property history, which can reduce title disputes and fraud.



## 13. APPENDIX

### Source Code

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

pragma solidity ^0.8.0;

contract PropertyDetail {

    // State variable to store the owner of the contract
    address public owner;

    // Struct to represent a real estate property
    struct Property {
        string propertyId; // Unique identifier for the property
        string name; // Name of the property
        string location; // Location of the property
        string discription; // Description of the property
        address currentOwner; // Current owner of the property
    }

    // Mapping to store property details with propertyId as the key
    mapping(string => Property) public properties;

    // Mapping to manage access control for properties, mapping addresses to propertyIds
    mapping(address => mapping(string => bool)) public hasAccess;

    // Events for logging property-related actions
    event PropertyAdded(
        string indexed propertyId,
        string name,
        string location,
        address indexed owner
    );
```

```

event PropertyTransferred(
    string indexed propertyId,
    address indexed from,
    address indexed to
);

// Constructor to set the contract owner as the deployer of the contract
constructor() {
    owner = msg.sender;
}

// Modifier to restrict function access to the contract owner
modifier onlyOwner() {
    require(msg.sender == owner, "Only contract owner can call this");
    _;
}

// Modifier to check if the caller has access to a specific property
modifier hasPropertyAccess(string memory propertyId) {
    require(
        hasAccess[msg.sender][propertyId],
        "You don't have access to this property"
    );
    _;
}

// Function to add a new property to the system, accessible only by the owner
function addProperty(
    string memory propertyId,
    string memory name,
    string memory location,
    string memory _description

```

```

) external onlyOwner {

    // Check if the property with the given propertyId doesn't already exist
    require(

        bytes(properties[propertyId].propertyId).length == 0,

        "Property already exists"

    );

    // Create a new Property structure and initialize its values
    properties[propertyId] = Property({

        propertyId: propertyId,

        name: name,

        location: location,

        discription: _description,

        currentOwner: owner

    });

    // Grant the owner access to the property
    hasAccess[owner][propertyId] = true;

    // Emit an event to log the addition of the property
    emit PropertyAdded(propertyId, name, location, owner);
}

// Function to transfer ownership of a property to a new owner
function transferProperty(

    string memory propertyId,

    address newOwner

) external hasPropertyAccess(propertyId) {

    // Check if the new owner's address is valid
    require(newOwner != address(0), "Invalid new owner");

    // Get the current owner's address

```

```

address currentOwner = properties[propertyId].currentOwner;

// Update the currentOwner of the property to the new owner
properties[propertyId].currentOwner = newOwner;

// Update access control to revoke access from the current owner and grant access to the new owner
hasAccess[currentOwner][propertyId] = false;
hasAccess[newOwner][propertyId] = true;

// Emit an event to log the property transfer
emit PropertyTransferred(propertyId, currentOwner, newOwner);
}

// Function to retrieve property details (name, location, and current owner) by propertyId
function getPropertyDetails(
    string memory propertyId
) external view returns (string memory, string memory, address) {
    Property memory prop = properties[propertyId];
    return (prop.name, prop.location, prop.currentOwner);
}
}

```

## GitHub & Project Demo Link

Github link : <https://github.com/Aathianbu/NM2023TMID03676>

Project Demo link : <https://youtu.be/uR5dz4ncSNY>