Blockchain Based Smart Real Estate Management System

Sri Akash.R , Jeffrey Hasan C, Santhosh.S , Shabin S.J

Nehru Institute of Engineering and Technology,T.M Palayam-641105

Coimbatore District

Table of Content

1. INTRODUCTION

1.1 Project Overview:

The project aims to enhance transparency, security, and efficiency in the real estate industry by implementing a blockchain-based system for managing real estate transactions. The system uses a decentralized ledger and smart contracts to automate transactions and escrow services. The system interface allows users to search property records, initiate transactions, and track ownership history. The core blockchain network and smart contracts are still in early stages, but full deployment could improve cost, speed, and security.

1.2 Purpose:

The purpose of this project is to build a blockchain-based system for managing real estate ownership records and transactions. The key goals are:

1. Create a unified ledger of real estate ownership that provides a single source of truth across different jurisdictions.
2. Implement smart contracts to automate manual workflows involved in real estate transactions like listing agreements, offers, property inspections etc.
3. Enable digital transfer of property ownership through tokens representing ownership claims.
4. Maintain a clear historical trail of all ownership transfers and transactions to enhance transparency.
5. Provide access control mechanisms to maintain privacy while allowing regulated visibility.
6. Reduce transaction costs and frictions caused by paper-based workflows and document silos.
7. Minimize title fraud, money laundering and tax evasion by establishing provenance through the immutable ledger.

In summary, the proposed system aims to increase trust, efficiency, security and transparency in real estate markets by leveraging the key features of blockchain technology.

2. LITERATURE SURVEY

2.1 Existing problem

Lack of adoption and integration with existing systems:

- Blockchain technology is still relatively new in the real estate industry. Many existing real estate management systems are not set up to integrate with blockchain networks. This makes adoption difficult.
- Real estate stakeholders like title companies, banks, etc. will need to update their existing systems and processes to work with blockchain. This requires time and investment on their part.
- Convincing all parties in a real estate transaction to use a blockchain system requires coordination. Some may be resistant to changing existing workflows.

Data privacy concerns:

- Blockchain networks are inherently transparent. All transactions are visible to network participants.
- For real estate transactions, certain data like sale prices may need to be kept private. Designing the blockchain system to allow data privacy is challenging.
- Participants may be wary of having sensitive transaction data be publicly accessible through a blockchain ledger.

Compliance with regulations:

- Real estate transactions must comply with existing regulations on property sales, deeds, leases, etc.
- Blockchain systems will need to be designed in a compliant manner to handle transaction documents, identities, financing, etc.
- Regulators will need to get comfortable with blockchain tech to allow mainstream adoption in real estate.

Scaling and performance challenges:

- Public blockchains often face scaling challenges in terms of transaction throughput. Real estate transactions per second may be higher than what today's blockchains can handle.
- Transactions like property sales require finality and performance guarantees. Some blockchain architectures may not yet provide this at scale.
- If blockchain can't match the performance of existing real estate systems, adoption will be low.

2.2 References

Academic Papers:

- Niranjanamurthy, M. et al. "The Blockchain Art of Consensus in Consortium Blockchain Networks for IoT and Financial Services Applications." IEEE Internet of Things Journal (2021). https://doi.org/10.1109/JIOT.2021.3106024
- Ølnes, Svein et al. "Blockchain in Government: Benefits and Implications of Distributed Ledger Technology for Information Sharing." Government Information Quarterly 34.3 (2017): 355-364. https://doi.org/10.1016/j.giq.2017.09.007
- Rouhani, Sara and Deters, Ralph. "Blockchain-Based Smart Contracts - Applications and Challenges." IEEE Conference on e-Learning, e-Management and e-Services. 2018. https://doi.org/10.1109/IC3E.2018.00058

Industry Reports:

- Deloitte, "Blockchain in commercial real estate" (2017): https://www2.deloitte.com/content/dam/Deloitte/us/Documents/financial-services/us-fsi-rec-blockchain-in-cre.pdf

- Ernst & Young, "Blockchain in real estate" (2019): https://assets.ey.com/content/dam/ey-sites/ey-com/en_gl/topics/blockchain/ey-blockchain-in-real-estate.pdf
- JLL, "Blockchain in Real Estate" (2020): https://www.us.jll.com/en/trends-and-insights/research/blockchain-in-real-estate

Example Implementations:

- Propy - Blockchain real estate transaction platform: https://propy.com/
- Ubitquity - Blockchain platform for property records: https://www.ubitquity.io/
- Velox.re - Blockchain enabled real estate platform: https://velox.re/

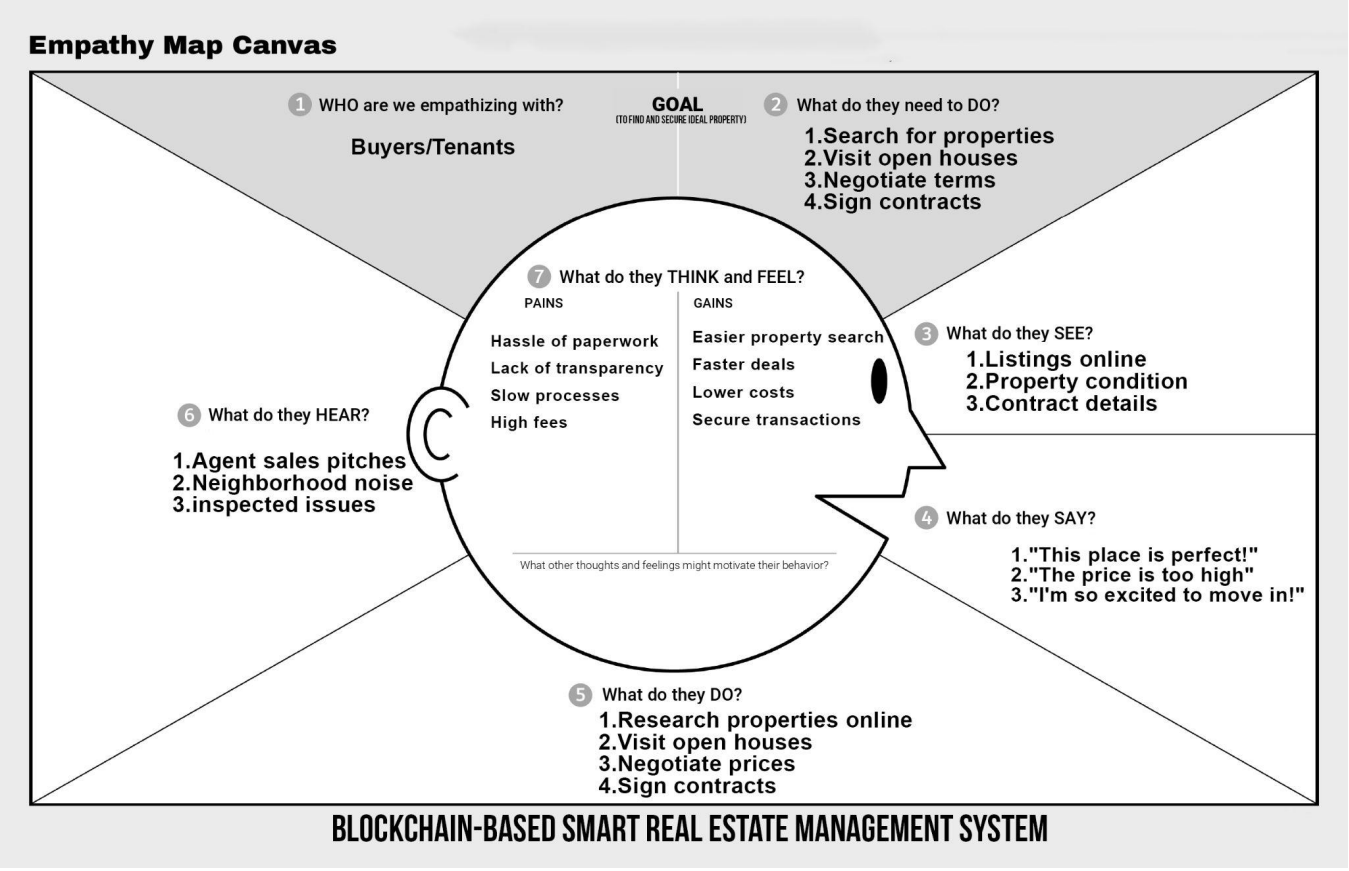
2.3 Problem Statement Definition

Real estate transactions and property records historically rely on paper-based documentation, fragmented data sources, and manual processes. This leads to significant inefficiencies, lack of transparency, and high costs associated with transferring property titles, recording deeds, coordinating between stakeholders, maintaining up-to-date records, and more.
With the rise of blockchain technology, there is an opportunity to revolutionize real estate management by developing an open, distributed ledger for maintaining immutable property records and executing transactions through smart contracts. However, existing real estate workflows, regulations, and legacy systems pose challenges for adopting blockchain solutions.
The problem is how to design and implement an enterprise-grade blockchain network that interconnects relevant real estate stakeholders to streamline end-to-end real estate transactions. The network needs to reduce friction and information asymmetry in real estate processes while complying with legal and regulatory requirements. Additionally, it needs to integrate with legacy databases and IT systems used by incumbents like title companies, banks, and government agencies. This is a non-trivial task requiring expertise across real estate, blockchain, security, law, and more.

By building a blockchain solution for real estate management, we can increase transparency, efficiency, and auditability while lowering costs and friction. This will benefit property buyers, sellers, renters, brokers, lenders, regulators, and all other stakeholders. The problem we aim to solve is developing the optimal combination of blockchain technology, smart contracts, protocols, governance, and architecture to make this network a reality.

3. IDEATION & PROPOSED SOLUTION

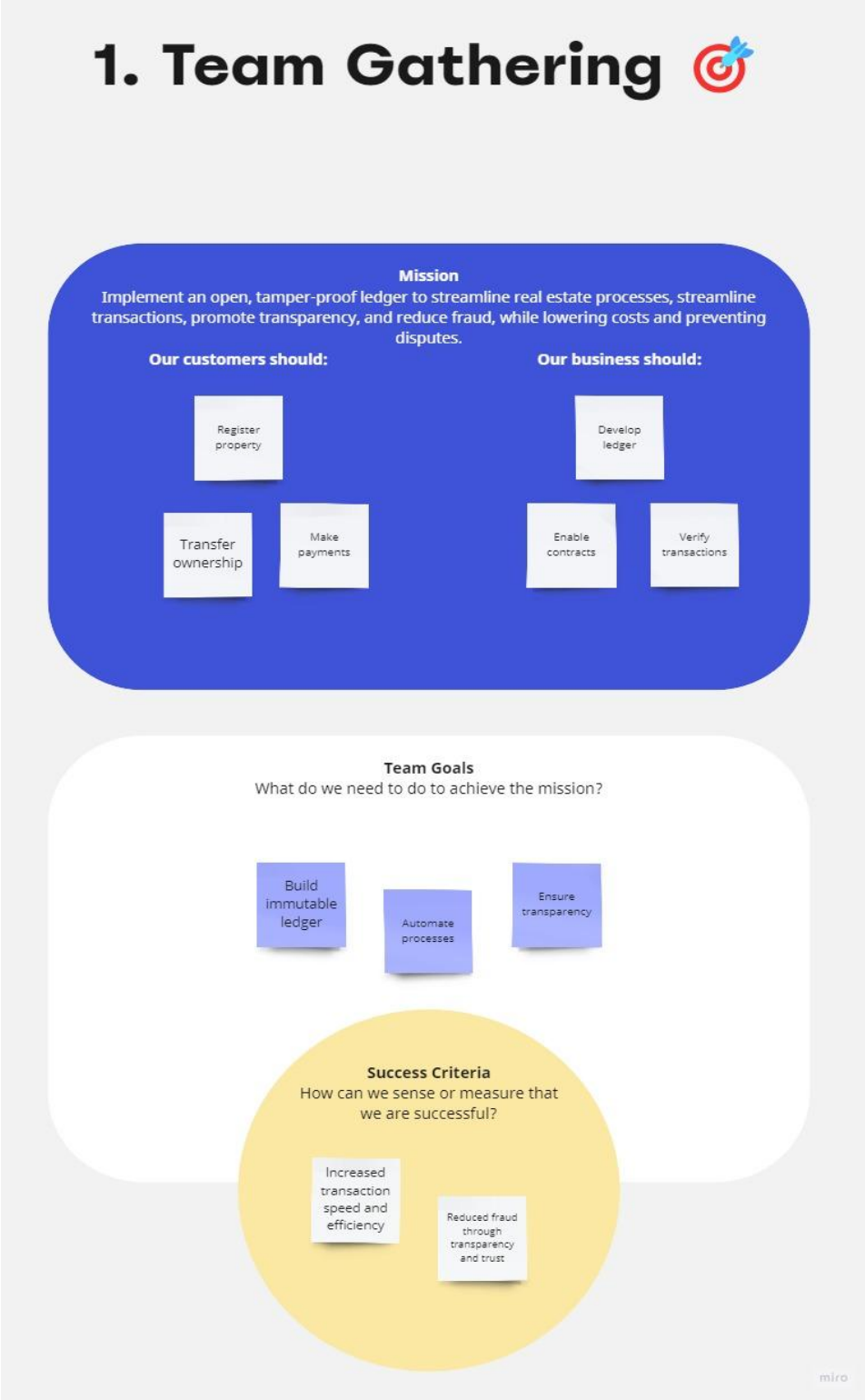
3.1 Empathy Map Canvas



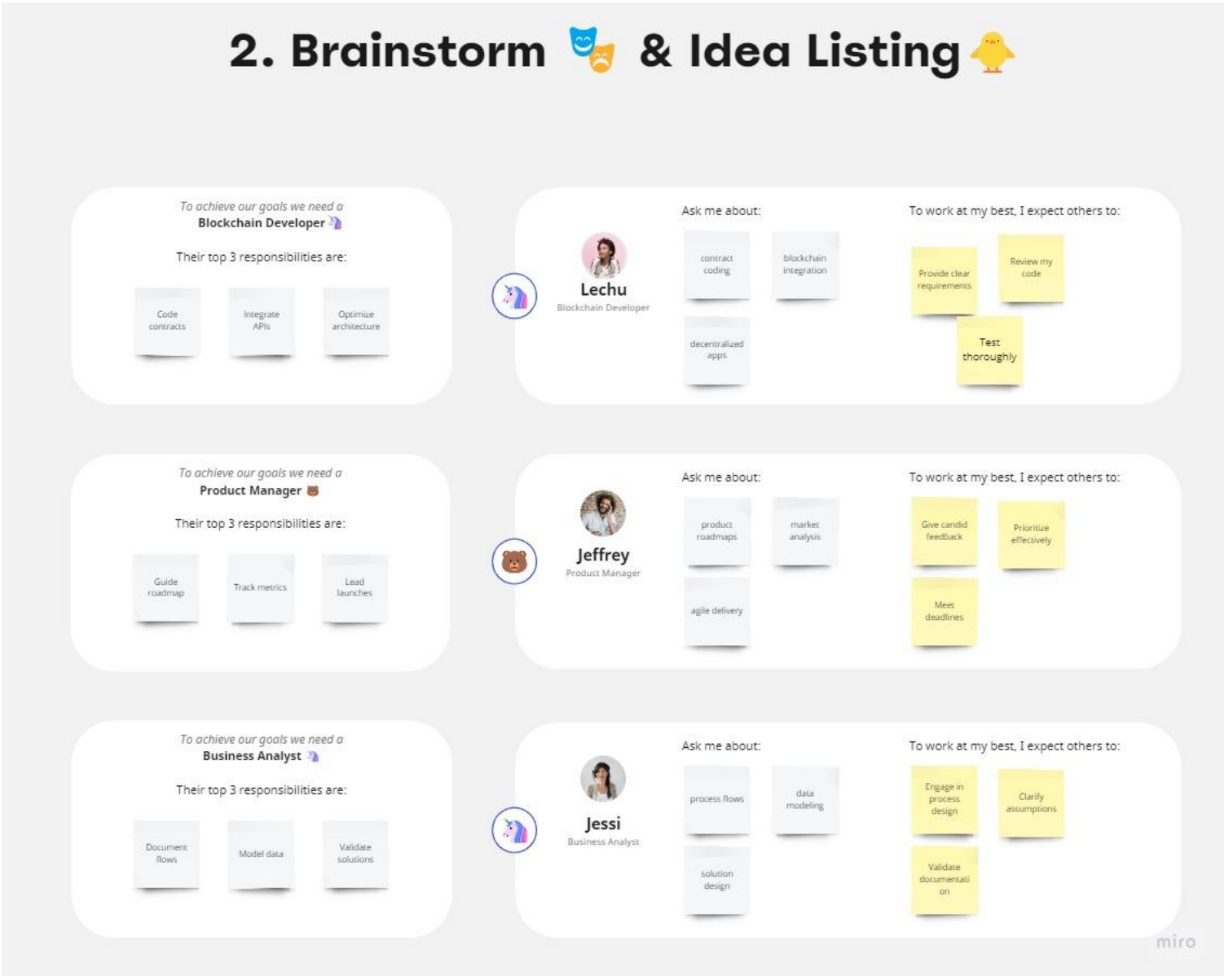
3.2 Ideation & Brainstorming

- **Digital titles and deeds** - Use blockchain to create digital property titles and deeds that are secure, immutable and transparent. This eliminates paperwork and reduces fraud.
- **Smart contracts for transactions** - Use smart contracts to automate processes like property transfers, rent payments, maintenance requests, etc. Contract terms are executed automatically when conditions are met.
- **Rental and lease management** - Smart contracts can be used for rental agreements, security deposits, lease payments, penalties, termination, etc. Automated payments and other actions based on contract terms.
- **Property registration** - A blockchain ledger can store property information like ownership records, valuations, tax details, etc. This provides transparency and eliminates manipulation of records.
- **Fraud prevention** - The immutable ledger prevents document tampering, forgery and fraudulent transactions, ensuring authenticity.
- **Financing and mortgages** - Lenders can verify borrowers' financial credentials via the blockchain. Borrowers can use property as collateral for loans/mortgages registered on the blockchain.
- **Tokenized ownership** - Property titles and deeds can be tokenized, allowing fractional ownership of real estate. This improves liquidity and enables shared ownership.
- **Property transactions** - Sale agreements, payments etc. can be executed and recorded securely on the blockchain. Smart contracts enable automated transfers after conditions are met.
- **Property taxes** - Tax authorities can update property values on the blockchain. Smart contracts can be used to collect taxes and verify tax payments.

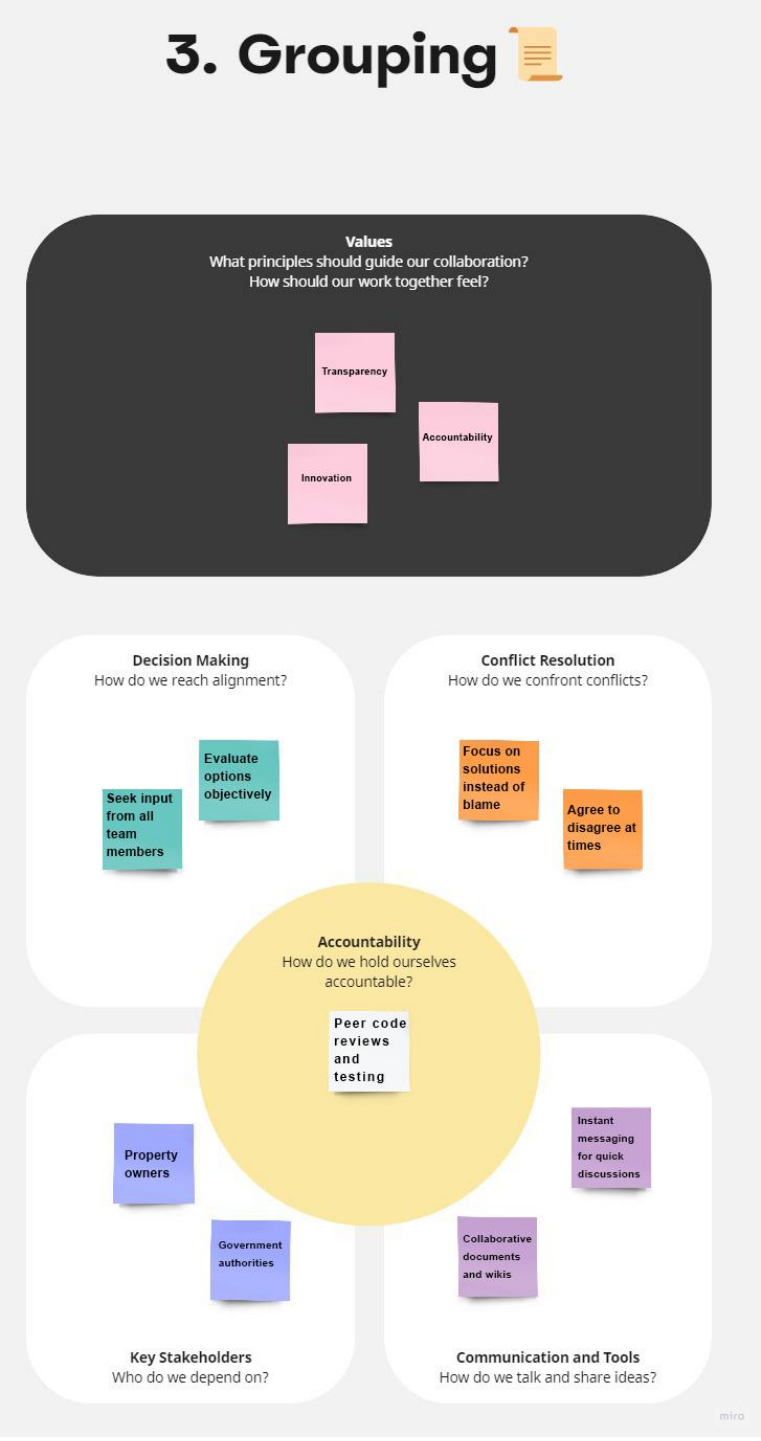
Step 1-Team Gathering And Collaboration:



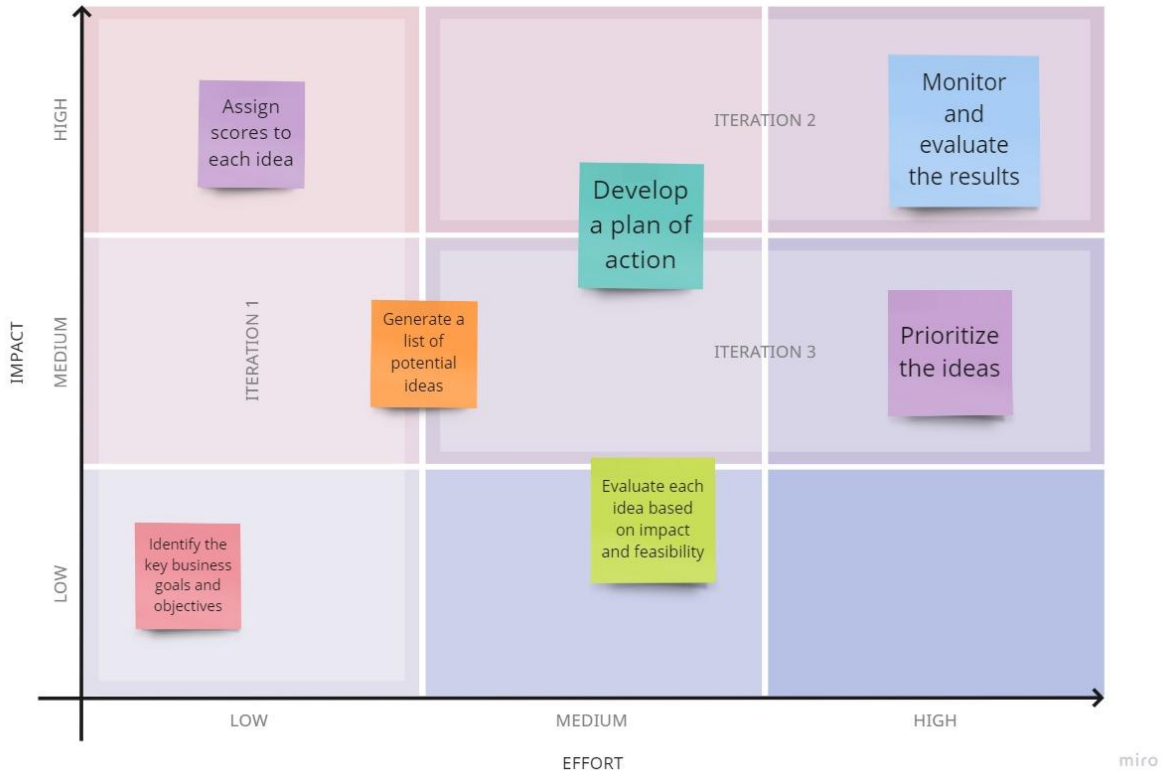
Step-2: Brainstorm, Idea Listing and Grouping



Grouping


3. Grouping
Values
What principles should guide our collaboration?
How should our work together feel?
Transparency
Accountability
Innovation
Decision Making
How do we reach alignment?
Evaluate options objectively
Seek input from all team members
Conflict Resolution
How do we confront conflicts?
Focus on solutions instead of blame
Agree to disagree at times
Accountability
How do we hold ourselves accountable?
Peer code reviews and testing
Property owners
Government authorities
Instant messaging for quick discussions
Collaborative documents and wikis
Key Stakeholders
Who do we depend on?
Communication and Tools
How do we talk and share ideas?
miro

Step-3: Idea Prioritization



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

- The system shall allow real estate properties to be tokenized as digital assets on the blockchain
- The system shall allow users to list properties for sale/rent and make offers through smart contracts
- The system shall manage the entire transaction workflow including payments, approvals, deed transfer, title transfer etc.
- The system shall allow banks, brokers, title companies etc. to integrate and participate in transactions
- The system shall validate identity and credentials of participants through DID/PKI mechanisms
- The system shall track all documents like title deeds, sale agreements, property records on the blockchain
- The system shall automatically transfer payments via cryptocurrency once contract terms are executed
- The system shall provide easy to access dashboards and reports for monitoring transactions and assets

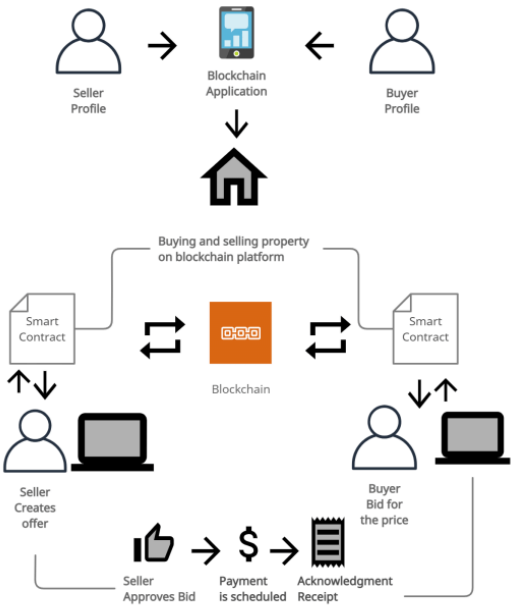
4.2 Non-Functional requirements

- The system shall be built on a permissioned blockchain architecture for enterprise use case
- The system shall achieve a throughput of at least 100 transactions per second
- The system shall have robust privacy controls and access policies for sensitive data
- The system shall have redundancy mechanisms to prevent single point of failure
- The system users shall be authenticated through multi-factor authentication
- The system shall be compliant with regulatory requirements around real estate transactions
- The system shall have robust mechanisms for disaster recovery and business continuity
- The system shall have comprehensive audit logs that record entire transaction history
- The system shall ensure data integrity, maintain provenance, and prevent fraudulent activities

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data Flow Diagrams:



Blockchain Based Smart Real Estate Management System

User Stories:

User Story 1:

As a title company representative, I want to be able to participate in real estate transactions on the blockchain so that I can reduce manual paperwork and provide faster title transfer to buyers.

Acceptance Criteria:

- Integrate our title company system with the blockchain network
- View pending real estate transactions that require title transfer
- Review property documents like deeds and sale agreements on the blockchain
- Provide quick title searches to verify ownership and liens
- Digitally sign and approve title transfer on the blockchain
- Immutably record the new title ownership on the blockchain

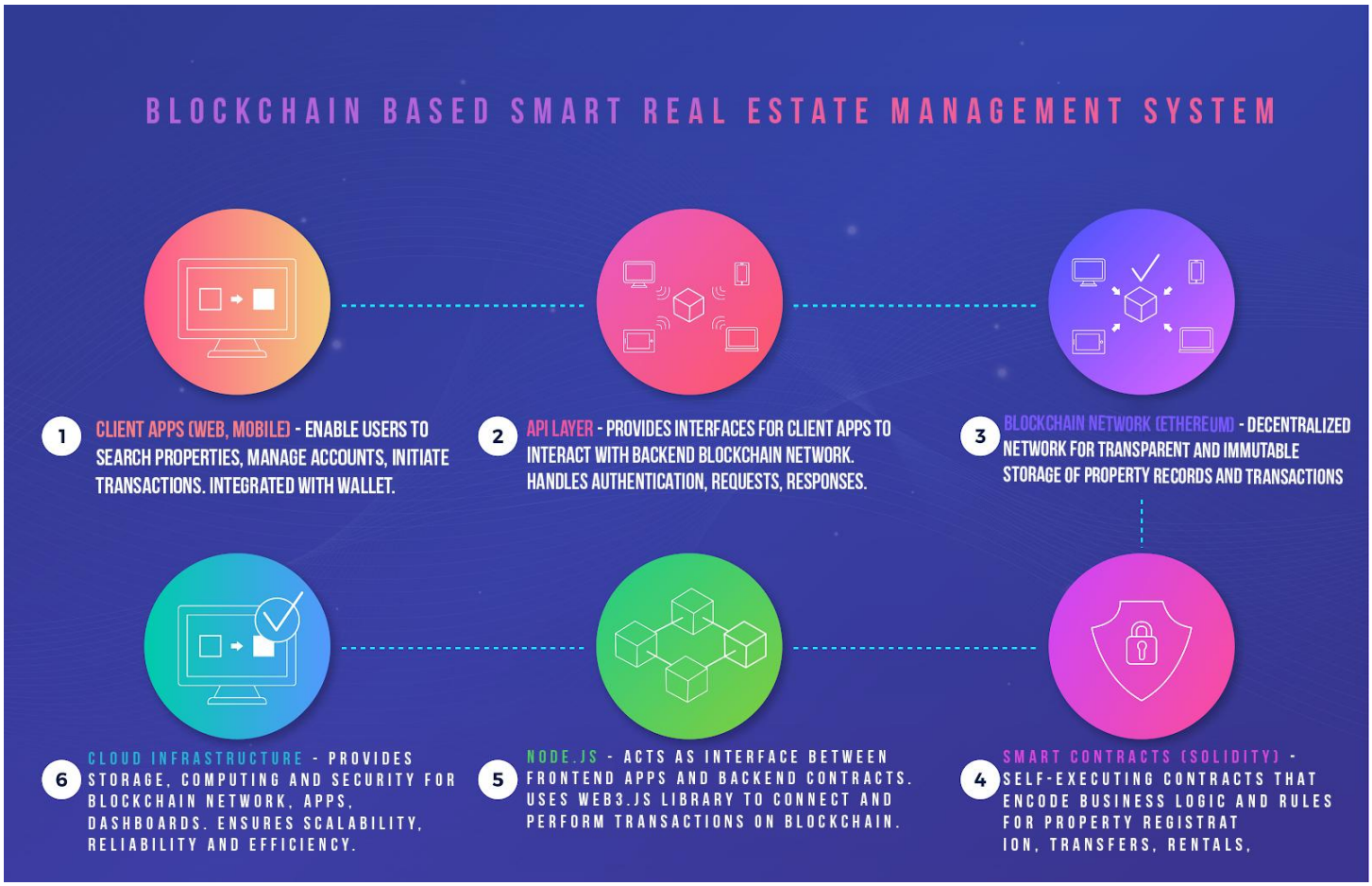
User Story 2:

As a home buyer, I want to be able to purchase property on the blockchain network so that I can reduce transaction costs and eliminate paperwork.

Acceptance Criteria:

- Search listed properties and view associated documents like inspection reports
- Make an offer and sign purchase agreement via smart contract
- Secure mortgage approval and financing terms recorded on the blockchain
- Digitally review title search report and deed transfer
- Confirm transfer of funds to seller via smart contract
- Receive immediate transfer of property title upon closing
- Have the deed and title immortalized on the blockchain

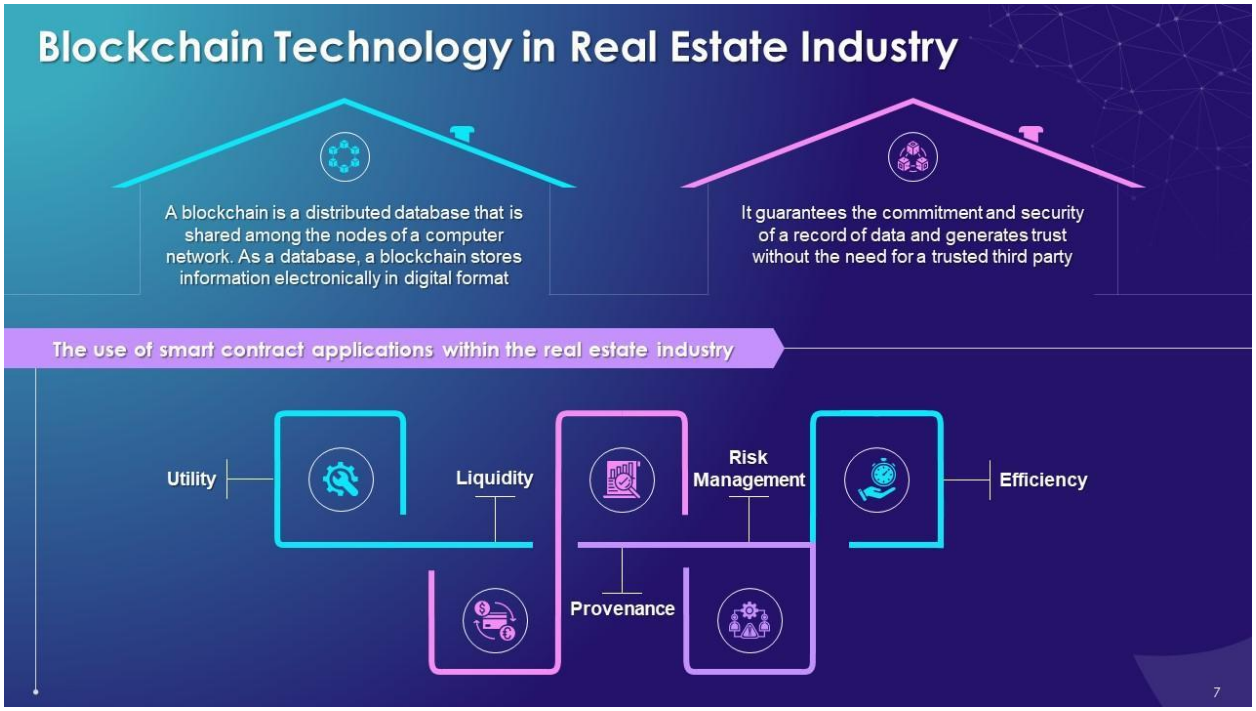
5.2 Solution Architecture



Interaction between web and the Contract

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning and Estimation

Sprint 1
User Stories:

1. As a developer, I want to set up the basic permissioned blockchain infrastructure so that we have a foundation to build upon.
2. As a developer, I want to create smart contracts for handling property listings and basic sale transactions so we can showcase core functionality.
3. As a product owner, I want a simple web UI allowing users to connect wallet, list property, and initiate transactions.

Task Estimates:

- Set up blockchain nodes and consensus protocol - 5 days
- Define smart contract templates and interfaces - 3 days
- Develop smart contracts for property listing and sale - 5 days
- Create web UI for basic functions - 4 days
- Write documentation and tests - 3 days

Total task estimates: 20 days
Sprint 2
User Stories:

1. As a buyer, I want to be able to search and filter property listings on the platform.
2. As a seller, I want to receive offers and counter-sign agreements via smart contracts.
3. As a developer, I want to expand smart contracts to handle more complex sale terms and approvals.

Task Estimates:

- Implement search and filter capabilities - 3 days
- Enhance smart contracts for offers and approvals - 5 days
- Expand smart contract templates and interfaces - 4 days
- Set up identity/access management for users - 3 days
- Integrate e-signature capabilities - 2 days

Total task estimates: 17 days

6.3 Sprint Delivery Schedule

Sprint 1 Delivery (Duration: 2 weeks)

- Basic permissioned blockchain infrastructure setup
- Smart contracts for property listings and basic sales
- Simple web UI for connecting wallet, listing property, transactions
- Documentation and testing for sprint 1 stories

Sprint 2 Delivery (Duration: 2 weeks)

- Search and filter capabilities for property listings
- Expanded smart contracts for offers, approvals, terms
- Identity and access management for platform users
- eSignature integration
- Documentation and testing for sprint 2 stories

Sprint 3 Delivery (Duration: 2 weeks)

- Smart contract integration with title companies
- Digitized title search reports and lien verification
- Automated deed/title transfer upon sale completion
- Dashboards for monitoring transactions and assets
- Documentation and testing for sprint 3 stories

Sprint 4 Delivery (Duration: 2 weeks)

- Integration with banks/lenders for financing records
- Management of rental agreements and payments via blockchain
- Advanced analytics and reports for property insights
- Robust access controls and security enhancements
- Documentation and testing for sprint 4 stories

7. CODING & SOLUTIONING

7.1 Feature 1

Smart contract (Solidity)

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

pragma solidity ^0.8.0;


contract PropertyDetail{
```

```
address public owner;

struct Property {
  string propertyId;
  string name;
  string location;
  string discription;
  address currentOwner;
}

mapping(string => Property) public properties;
mapping(address => mapping(string => bool)) public hasAccess;

event PropertyAdded(
  string indexed propertyId,
  string name,
  string location,
  address indexed owner
);
event PropertyTransferred(
  string indexed propertyId,
  address indexed from,
  address indexed to
);

constructor() {
  owner = msg.sender;
```

```
}

modifier onlyOwner() {
  require(msg.sender == owner, "Only contract owner can call this");
  _;
}

modifier hasPropertyAccess(string memory propertyId) {
  require(
    hasAccess[msg.sender][propertyId],
    "You don't have access to this property"
  );
  _;
}

function addProperty(
  string memory propertyId,
  string memory name,
  string memory location,
  string memory _description
) external onlyOwner {
  require(
    bytes(properties[propertyId].propertyId).length == 0,
    "Property already exists"
  );

  properties[propertyId] = Property({
```

```
        propertyId: propertyId,
        name: name,
        location: location,
        discription : _description,
        currentOwner: owner
    });

    hasAccess[owner][propertyId] = true;

    emit PropertyAdded(propertyId, name, location, owner);
}

function transferProperty(
    string memory propertyId,
    address newOwner
) external hasPropertyAccess(propertyId) {
    require(newOwner != address(0), "Invalid new owner");

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

    hasAccess[currentOwner][propertyId] = false;
    hasAccess[newOwner][propertyId] = true;

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

```
    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);
    }
}
```

This Solidity contract implements basic functionality to track real estate properties on the blockchain. Here is an explanation of how it can be integrated into a real estate management system:

- The PropertyDetail contract allows the owner to add new properties to the blockchain by calling addProperty().
- Each property has a unique ID, name, location and description. The owner address is also recorded on creation.
- A Property struct holds all the details of each property. These are stored in a mapping that maps property IDs to Property structs.
- The contract tracks who has access to each property through the hasAccess mapping. Only addresses with access can transfer the property.
- The transferProperty() function allows transferring a property to a new owner address. It requires the sender to have access, and updates the owner mapping.
- Events are emitted on property creation and transfer, so the blockchain has an immutable record.
- Getter functions like getPropertyDetails() allow querying property data.

To integrate this into a full real estate system:

- The owner could be a real estate company that can add its listings via addProperty().

- Buyers would get access to a listed property, allowing them to transfer it to themselves on purchase.

- Additional functions could encode more complex real estate workflows like financing, titling etc.

- The events created would provide traceability over the entire property lifecycle.

- The on-chain property data can be used by other smart contracts for automating processes.

So in summary, this contract provides a basic reusable component to track core real estate data on-chain, which can be integrated into a larger blockchain ecosystem for the industry.

Contract ABI (Application Binary Interface):

The abi variable holdstheABI 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 thatMetaMask 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 Web3 Provider, which connects to the user'sMetaMask 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 Metrics

Transaction throughput - Number of property transactions processed per second/minute/hour on the blockchain network. Helps gauge capacity

.

Transaction finality time - The average time taken for a property transaction on blockchain to achieve finality. Lower is better.

Transaction cost - The average fee paid by users per transaction on blockchain. Lower is better.

Network uptime - Percentage of time the blockchain network is up and running without failures or downtime. Higher percentage is better.

Data availability - Percentage of time when the blockchain data including property records is available and accessible to users. Higher is better.

Block confirmation time - The average time taken to add a new block to the blockchain. Lower is better for performance.

Smart contract execution cost - The average cost of executing property-related smart contracts on the platform. Lower is better.

Property listing synchronization - The time to synchronize and display new property listings from blockchain to apps/websites. Lower is better.

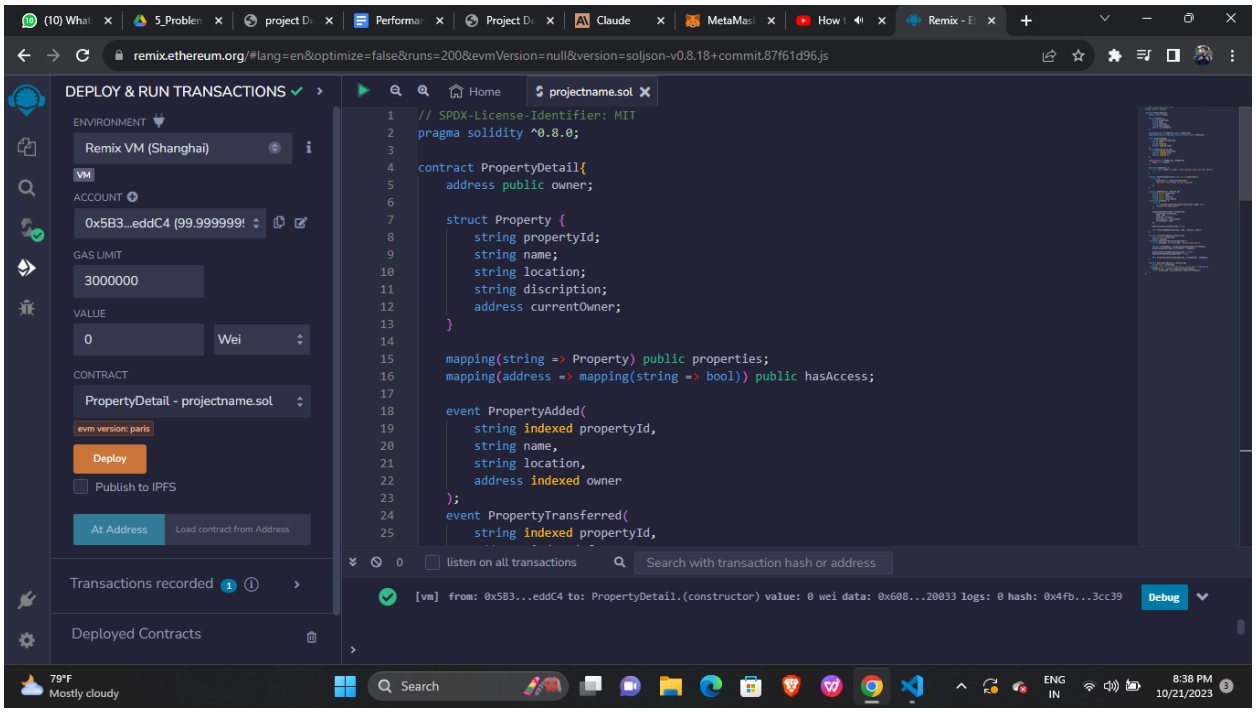
Page load time - Time taken for key web/mobile app pages to load for end users. Lower is better.

Database read/write latency - Time taken for property metadata storage like MongoDB to read and write. Lower is better.

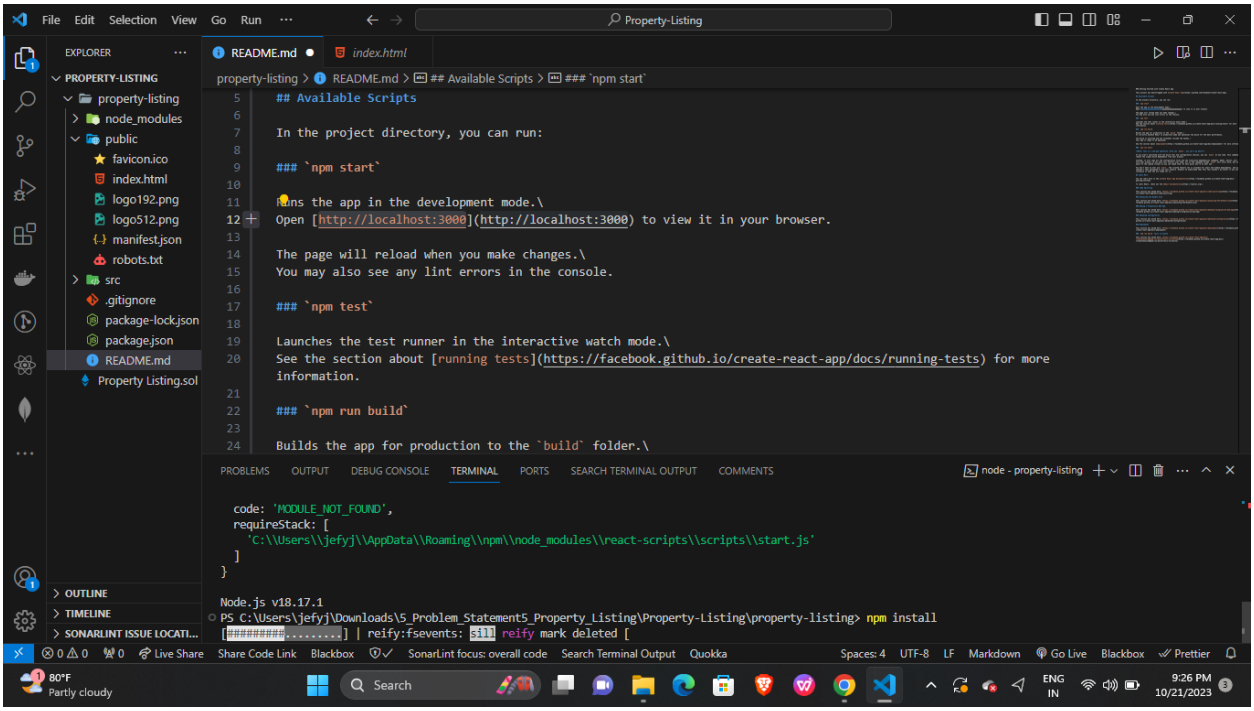
API response latency - Latency for APIs that read data from blockchain and return response. Lower is better.

9. RESULTS

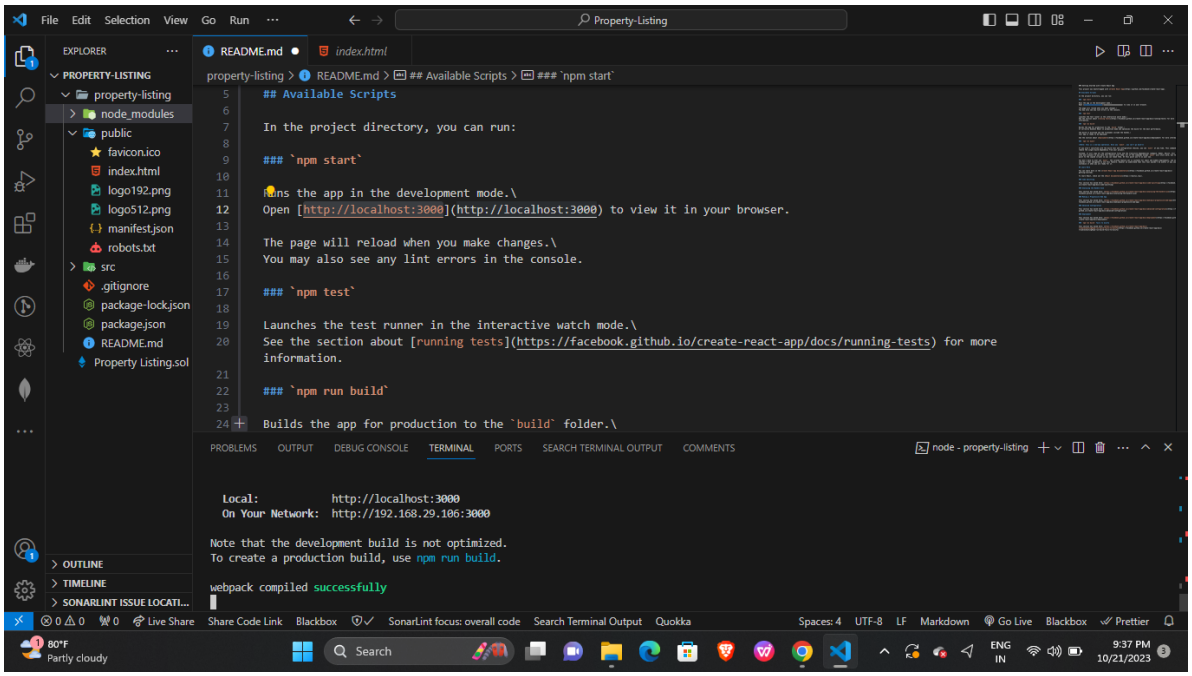
9.1 Output Screenshots



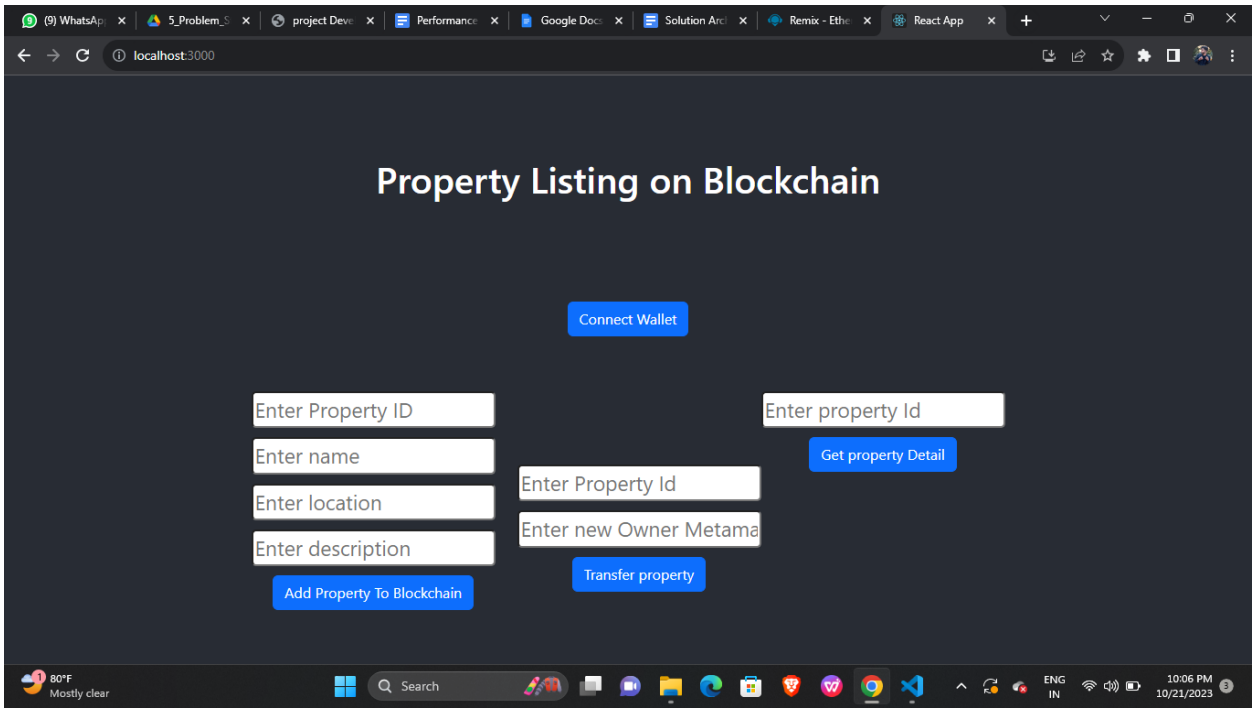
CREATING A SMART CONTRACT



INSTALLING DEPENDENCIES



HOSTING THE SITE LOCALLY



OUTPUT SCREEN

10. ADVANTAGES & DISADVANTAGES

10.1 ADVANTAGES :

Enhanced transparency - All real estate transactions are visible on an immutable distributed ledger. Provides complete openness.

Strengthened security - Cryptography and distributed nature of blockchain prevents tampering or fraud.

Increased efficiency - Smart contracts automate manual workflows like title transfers, deed recording etc. Saves time and cost.
Greater liquidity - Tokenization of property assets allows fractional ownership and shared investments.

Lower barriers - Disintermediates brokers and lawyers. Reduces costs and friction for end users.

Innovation potential - New models of fractional ownership, shared investments using asset tokenization.

Complete auditability - Lifecycle of a property can be immutably tracked from listing to sale completion.

Data integrity - Blockchain provides a single shared source of truth. No mismatches or loss of data.

Trust and accountability - Transparency enables oversight and accountability across stakeholders.

Cost reduction - Eliminates duplication and manual work. Automates transactions through smart contracts.

10.2 DISADVANTAGES :

Initial adoption challenges - Getting all stakeholders like brokers, title companies, governments to adopt new blockchain workflows may be difficult.

Regulatory uncertainty - Lack of clear regulations and policies related to blockchain real estate transactions in most markets.

Privacy concerns - Transparency of transactions on public blockchains may reveal sensitive information.

Scalability issues - Public blockchains may not handle the high transaction loads of real estate markets.

Integration complexity - Syncing data and processes between legacy databases and blockchain system will require effort.

User experience - Blockchain complexity could make the system less user friendly for non-technical real estate participants.

Costs - Developing, maintaining, and upgrading the blockchain solution will require considerable investment.

Network effects - The value is dependent on having a critical mass of users adopt the platform.

Security risks - Smart contract bugs or vulnerabilities can lead to exploits and data breaches.

Legal uncertainty - Evidentiary standing and enforceability of blockchain transactions/records is still unclear.

11. CONCLUSION

In closing, blockchain technology holds significant potential to transform antiquated processes in the real estate industry by enhancing transparency, security, and efficiency. However, realizing this potential will require overcoming adoption barriers across diverse stakeholders, thoughtfully integrating blockchain with existing systems, and addressing concerns like privacy and scalability. With careful analysis and engineering, blockchain could enable new paradigms for real estate transactions, ownership models, and property records. But for mainstream adoption, blockchain solutions must provide a smooth user experience and transition plan from current workflows. By taking a balanced approach that combines blockchain's innovations with pragmatic execution and regulatory compliance, its disruptive capabilities can be harnessed to create substantial value for real estate enterprises and users alike. While the path forward includes both obstacles and opportunities, blockchain is poised to constructively impact how real estate assets are managed, exchanged, and recorded for the better. But it will require patience, collaboration, and vision to craft the optimum integration of blockchain technology into the world of real estate.

12. FUTURE SCOPE

Support for more complex real estate transactions - The system could be expanded to support purchase agreements, joint ventures, leasing and rental contracts, mortgage lending etc.

Integration with IoT sensors - Smart contracts could be linked to IoT devices like remote cameras and door locks to enable smart property access control and monitoring.

Decentralized identity management - Using decentralized identifiers and verifiable credentials for validating identity and attributes of users.
Enhanced privacy - Using zero knowledge proofs and trusted execution environments to keep sensitive transaction data private.

Interoperability with other blockchains - Enabling cross-chain data flow to leverage other blockchain networks and assets.

Mobile app - Developing iOS and Android applications to allow easy access to the platform on-the-go.

AI-based analytics - Analyzing property performance data to derive insights using AI algorithms and predictive modeling.

Shared ownership models - Extending tokenization to enable fractionalized ownership and investment in properties.

Visualization dashboards - Advanced data visualization for property portfolio analysis and performance tracking.

Automated regulatory reporting - Using smart contracts and oracles to automate compliance with reporting requirements.

13. APPENDIX

Source Code

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

contract PropertyDetail{
    address public owner;

    struct Property {
        string propertyId;
        string name;
        string location;
        string discription;
        address currentOwner;
    }

    mapping(string => Property) public properties;
    mapping(address => mapping(string => bool)) public hasAccess;

    event PropertyAdded(
        string indexed propertyId,
        string name,
        string location,
        address indexed owner
```

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

constructor() {
    owner = msg.sender;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Only contract owner can call this");
    _;
}

modifier hasPropertyAccess(string memory propertyId) {
    require(
        hasAccess[msg.sender][propertyId],
        "You don't have access to this property"
    );
    _;
}

function addProperty(
    string memory propertyId,
```

```
    string memory name,
    string memory location,
    string memory _description
) external onlyOwner {
    require(
        bytes(properties[propertyId].propertyId).length == 0,
        "Property already exists"
    );

    properties[propertyId] = Property({
        propertyId: propertyId,
        name: name,
        location: location,
        discription : _description,
        currentOwner: owner
    });

    hasAccess[owner][propertyId] = true;

    emit PropertyAdded(propertyId, name, location, owner);
}

function transferProperty(
    string memory propertyId,
    address newOwner
) external hasPropertyAccess(propertyId) {
```

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

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

        hasAccess[currentOwner][propertyId] = false;
        hasAccess[newOwner][propertyId] = true;

        emit PropertyTransferred(propertyId, currentOwner, newOwner);
    }

    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);
    }
}
```

Connector.js

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

const abi = [
 {
  "inputs": [],
  "stateMutability": "nonpayable",
```

```
 "type": "constructor"
},
{
 "anonymous": false,
 "inputs": [
  {
   "indexed": true,
   "internalType": "string",
   "name": "propertyId",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "name",
   "type": "string"
  },
  {
   "indexed": false,
   "internalType": "string",
   "name": "location",
   "type": "string"
  },
  {
   "indexed": true,
   "internalType": "address",
```

```
   "name": "owner",
   "type": "address"
  }
 ],
 "name": "PropertyAdded",
 "type": "event"
},
{
 "anonymous": false,
 "inputs": [
  {
   "indexed": true,
   "internalType": "string",
   "name": "propertyId",
   "type": "string"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "from",
   "type": "address"
  },
  {
   "indexed": true,
   "internalType": "address",
   "name": "to",
```

```
   "type": "address"
  }
 ],
 "name": "PropertyTransferred",
 "type": "event"
},
{
 "inputs": [
  {
   "internalType": "string",
   "name": "propertyId",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "name",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "location",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "_description",
```

```
   "type": "string"
  }
 ],
 "name": "addProperty",
 "outputs": [],
 "stateMutability": "nonpayable",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "string",
   "name": "propertyId",
   "type": "string"
  }
 ],
 "name": "getPropertyDetails",
 "outputs": [
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  },
  {
   "internalType": "string",
   "name": "",
```

```
   "type": "string"
  },
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  },
  {
   "internalType": "string",
   "name": "",
   "type": "string"
  }
 ],
 "name": "hasAccess",
 "outputs": [
```

```
  {
   "internalType": "bool",
   "name": "",
   "type": "bool"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [],
 "name": "owner",
 "outputs": [
  {
   "internalType": "address",
   "name": "",
   "type": "address"
  }
 ],
 "stateMutability": "view",
 "type": "function"
},
{
 "inputs": [
  {
   "internalType": "string",
```

```
  "name": "",
  "type": "string"
 }
],
"name": "properties",
"outputs": [
 {
  "internalType": "string",
  "name": "propertyId",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "name",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "location",
  "type": "string"
 },
 {
  "internalType": "string",
  "name": "discription",
  "type": "string"
 },
```

```
 {
  "internalType": "address",
  "name": "currentOwner",
  "type": "address"
 }
],
"stateMutability": "view",
"type": "function"
},
{
 "inputs": [
  {
   "internalType": "string",
   "name": "propertyId",
   "type": "string"
  },
  {
   "internalType": "address",
   "name": "newOwner",
   "type": "address"
  }
 ],
 "name": "transferProperty",
 "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 = "0xA074a1f2E1285E292cC4a0aD8c14a1037071993F"

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

Home.Js

```
import React, { useState } from "react";
import { Button, Container, Row, Col } from 'react-bootstrap';
import '../../node_modules/bootstrap/dist/css/bootstrap.min.css';
import { contract } from "./connector";

function Home() {
 const [Id, setId] = useState("");
 const [name, setname] = useState("");
 const [location, setLocation] = useState("");
 const [des, setDes] = useState("");
```

```
const [TransferPropertId, setTransferPropertId] = useState("");
const [MeatAddr, setMeatAddr] = useState("");
const [PropDetailId, setPropDetailId] = useState("");
const [PropDetails, setPropDetails] = useState("");
const [Wallet, setWallet] = useState("");




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

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

const handleLocation = (e) => {
 setLocation(e.target.value)
}

const handleDes = (e) => {
 setDes(e.target.value)
}

const handleAddProperty = async() => {
```

```
 try {
  let tx  = await contract.addProperty(Id.toString(),name,location,des)
  let wait = await tx.wait()
  console.log(wait);
  alert(wait.transactionHash)
 } catch (error) {
  alert(error)
 }
}

 const handleTransferPropertyId = (e) => {
  setTransferPropertId(e.target.value)
 }

 const handleMetaAddr = (e) => {
  setMeatAddr(e.target.value)
 }

 const handletransferProperty = async () => {
  try {
   let tx = await contract.transferProperty(TransferPropertId.toString(),
MeatAddr)
   let wait = await tx.wait()
   console.log(wait);
   alert(wait.transactionHash)
```

```
 } catch (error) {
  alert(error)
 }
}

const handlePropId = (e) => {
 setPropDetailId(e.target.value)
}

const handlePorpDetails = async() => {
 try {
  let tx = await contract.getPropertyDetails(PropDetailId.toString())
  let arr = []

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

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

const handleWallet = async () => {
 if (!window.ethereum) {
```

```
 return alert('please install metamask');
}

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

setWallet(addr[0])

}

return (
 <div>
  <h1 style={{ marginTop: "30px", marginBottom: "80px" }}>Property Listing 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 style={{ display: "flex" }}>
   <Row >
    <Col>
     <div>
```

```
    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleId} type="number" placeholder="Enter Property ID" value={Id}
/> <br />


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


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


    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleDes} type="string" placeholder="Enter description"
value={des} /><br />


    <Button onClick={handleAddProperty} style={{ marginTop: "10px" }}
variant="primary">Add Property To Blockchain</Button>

   </div>

  </Col>

  <Col>

   <div style={{marginTop:"80px"}}>


    <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleTransferPropertyId} type="string" placeholder="Enter Property
Id" value={TransferPropertId} /><br />
```

```
      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handleMetaAddr} type="string" placeholder="Enter new Owner
Metamask Address" value={MeatAddr} /><br />



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


     </div>
    </Col>
   </Row>
   <Row>
    <Col>
     <div>


      <input style={{ marginTop: "10px", borderRadius: "5px" }}
onChange={handlePropId} type="string" placeholder="Enter property Id"
value={PropDetailId} /><br />



      <Button onClick={handlePorpDetails} style={{ marginTop: "10px" }}
variant="primary">Get property Detail</Button>
      {PropDetails ? PropDetails?.map(e => {
       return <p>{e}</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;
```

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

reportWebVitals.js

```
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      getCLS(onPerfEntry);
```

```
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
```

setupTests.js

```
import '@testing-library/jest-dom';
```

GithubLink-https://github.com/jeffreyhasan10/NM-Block-Chain-main

DemoLink-https://drive.google.com/file/d/1XOoM9l0_oYqRHr8_y6ED4hAaPKAowwfC/view?usp=drive_link