**REPORT**

**STEP1:**

**Prompt:** Give me detailed requirement of smart contract project for handling real estate transactions and business.

Response:

Creating a detailed requirements specification for a smart contract intended for handling real estate transactions involves outlining the functional and non-functional requirements, security considerations, and key stakeholders interactions. Below is a comprehensive approach to structuring this specification.

1. **Introduction**

**Purpose**

* The purpose of this document is to provide a detailed overview of a blockchain-based smart contract designed for managing real estate transactions, including buying, selling, and leasing properties.

**Scope**

* The smart contract will automate transaction validations, payments, and record-keeping related to real estate transactions to ensure transparency, security, and efficiency.

2. Overall Description

**User Needs**

* Property buyers/sellers need a secure and transparent system to conduct transactions.
* Real estate agents require a platform to manage listings and client interactions effectively.
* Legal entities need access to immutable records for verification and compliance purposes.

**Assumptions and Dependencies**

All users have access to a cryptocurrency wallet compatible with the blockchain.

Property data is accurately digitized and accessible.

3. **System Features and Requirements**

Functional Requirements

**1. User Registration and Authentication**

* Users (buyers, sellers, agents, lawyers) must register and authenticate to interact with the smart contract.

**2. Property Listing Management**

* Sellers or agents can list properties, set prices, and define sale/lease terms.
* Ability to update or remove listings by the owner or authorized agents.

**3. Transaction Handling**

* Automated facilitation of transactions when predefined conditions are met.
* Support for escrow services to secure funds until all conditions are satisfied.

4. Document Handling and Verification

* Upload and verify necessary legal documents (title deeds, inspection reports).
* Smart contract should ensure only authorized personnel can access sensitive documents.

5. Payment Processing

* Handle transfers of cryptocurrency and automatically adjust to current exchange rates if necessary.
* Generate receipts and transaction history for both parties.

6.Dispute Resolution

* Mechanisms to handle disputes, possibly through third-party arbitration smart contracts or pre-defined rules within the contract.

**Non-functional Requirements:**

1. Security

* The smart contract must be secure against common vulnerabilities (e.g., re-entrancy, overflow/underflow).
* Regular security audits and updates to address new vulnerabilities.

2. Performance

* The smart contract should handle transactions quickly, even during high network congestion.
* Optimized for minimal gas usage to reduce transaction costs.

3. Scalability

* Capable of handling a growing amount of transactions as user base increases.
* Support for additional property types and transaction structures.

4. Usability

* User-friendly interface for interacting with the smart contract.
* Comprehensive logs and alerts for significant actions and changes.

4. **External Interface Requirements**

User Interfaces

* Web interface accessible via standard web browsers, optimized for both desktop and mobile.
* Integration with blockchain wallets (e.g., MetaMask) for transactions.

Hardware Interfaces

* No specific hardware requirements beyond a standard computing device with internet access.

Software Interfaces

* Blockchain platform (e.g., Ethereum).
* IPFS for decentralized file storage.

Communications Interfaces

* HTTPS for secure communication.
* Integration with real estate platforms for data exchange.

**5. Other Requirements**

Regulatory Compliance

* Comply with local and international real estate laws and regulations.
* Ensure data privacy laws are adhered to, especially when handling personal user data.
* This specification provides a foundation for developing a smart contract tailored to the complexities of real estate transactions, ensuring that the solution is both robust and adaptable to the needs of various stakeholders involved in real estate.

**Prompt 2:** Based on the provided requirement give me a detailed design of smart contract project for handling real estate transactions and business.

**Response 2:**

**1. System Architecture**

Smart Contract Components

**User Contract**: Handles user registration, authentication, and management of user profiles.

**Property Contract**: Manages property listings, updates, and the storage of property details.

**Transaction Contract**: Facilitates buying, selling, and leasing processes including escrow and payment functionalities.

**Document Contract**: Manages uploading, validation, and retrieval of documents related to real estate transactions.

**Dispute Resolution Contract**: A separate contract or a set of functions within the Transaction Contract for handling disputes.

**External Integrations**

IPFS (InterPlanetary File System): For storing and accessing large documents in a decentralized manner.

Oracles: External services to provide real-time data like currency exchange rates or property valuation data.

**2. Data Flow Diagram**

High-Level Flow

**1.User Interaction:**

Users interact with the system via a web interface that communicates with the smart contracts on the blockchain.

Authentication is verified through blockchain transactions for every action requiring user validation.

**2.Property Listing:**

Sellers or agents submit property details and documents, which are stored on IPFS with references stored in the blockchain.

Listings are made available to potential buyers through the Property Contract.

**3.Transaction Processing:**

When a transaction is initiated, the Transaction Contract locks the agreed amount in an escrow.

The Document Contract checks the validity of required documents.

Upon all conditions being met, the contract executes the payment and updates the property ownership.

**4.Dispute Handling:**

In case of disputes, the Dispute Resolution Contract is invoked to arbitrate or enforce pre-agreed rules.

**3. Security Design**

**Smart Contract Security:**

Use established patterns to avoid common issues like reentrancy attacks.

Implement checks-effects-interactions pattern to prevent unauthorized actions.

**Audit and Compliance:**

Regularly audit smart contracts both internally and externally.

Ensure all actions are logged and traceable for compliance with real estate regulations.

**4. User Interface Design**

Web Interface:

User-friendly web application integrated with MetaMask for handling blockchain transactions.

Separate views for buyers, sellers, and agents, providing customized functionalities to each type of user.

**5. Technology Stack**

Blockchain: Ethereum for deploying smart contracts.

Smart Contract Language: Solidity.

**Frontend:** React.js integrated with Web3.js for interacting with Ethereum smart contracts.

**Backend:** (if necessary for additional processing or off-chain operations): Node.js.

**File Storage**: IPFS for decentralized storage of transaction-related documents.

**Oracle Services**: Chainlink or Band Protocol for real-world data integration.

**6. Scalability and Performance**

**Gas Optimization**: Ensure smart contracts are optimized for minimal gas usage to reduce transaction costs.

**Scalable Architecture**: Design contracts to handle an increasing number of transactions and users without significant drops in performance.

**7. Deployment Strategy**

**Testnet Deployment**: Initially deploy on Ethereum testnets (like Rinkeby or Ropsten) to test functionality.

**Mainnet Launch**: After thorough testing and auditing, deploy on Ethereum Mainnet.

**8. Maintenance and Upgrades**

**Upgradeable Contracts:** Use proxy contracts or Ethereum's new beacon chain approach to allow for smart contract upgrades without losing state or data.

This design outlines a comprehensive framework for a blockchain-based real estate transaction system, ensuring robust functionality, security, and user-friendly interactions. The design leverages modern technologies and best practices in smart contract development to create a secure, efficient, and scalable system.

**Prompt 3:** For the provided design and requirements give me the tests to test the function correctness, security, privacy, fairness, and transparency or decentralization.

**Response 3:**

**Solidity Testing:**

You can also write tests in Solidity itself. This method is especially useful for internal function testing and when JavaScript cannot emulate certain behaviours.

Create a Solidity Test Contract: Write a new Solidity contract that tests another contract.

**Example of a Solidity test contract:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "remix\_tests.sol"; // this import is automatically injected by Remix.

import "../contracts/Wallet.sol"; // Adjust the path to your contract

contract WalletTest {

Wallet wallet;

function beforeAll() public {

wallet = new Wallet();

}

function checkInitialOwner() public {

Assert.equal(wallet.owner(), msg.sender, "Owner should be the deployer");

}

}

**Prompt 3(Updated with requirement):**

Okay let me provide you clear instruction I have three different contracts named "UserContract","PropertyContract","TransactionContract",

Here are the solidity code:

**User Contract:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract UserContract {

struct User {

uint id;

string name;

string role; // Buyer, Seller, Agent

}

mapping(address => User) public users;

uint public userCount;

function registerUser(string memory name, string memory role) public {

require(bytes(users[msg.sender].name).length == 0, "User already registered");

userCount++;

users[msg.sender] = User(userCount, name, role);

}

function getUser(address userAddress) public view returns (User memory) {

return users[userAddress];

}

}

PropertyContract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract PropertyContract {

struct Property {

uint id;

string title;

string location;

uint price;

address owner;

bool isListed;

}

mapping(uint => Property) public properties;

uint public propertyCount;

function listProperty(string memory title, string memory location, uint price) public {

propertyCount++;

properties[propertyCount] = Property(propertyCount, title, location, price, msg.sender, true);

}

function updateProperty(uint propertyId, uint newPrice) public {

Property storage property = properties[propertyId];

require(msg.sender == property.owner, "Only property owner can update the listing");

property.price = newPrice;

}

}

TransactionContract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract TransactionContract {

enum State { Created, Locked, Inactive }

State public state;

address payable public buyer;

address payable public seller;

uint public value;

uint public propertyId;

modifier inState(State \_state) {

require(state == \_state, "Invalid state.");

\_;

}

function initiateTransaction(uint \_propertyId, address payable \_seller) public payable {

buyer = payable(msg.sender);

seller = \_seller;

propertyId = \_propertyId;

value = msg.value;

state = State.Created;

}

function confirmTransaction() public inState(State.Created) {

require(msg.sender == buyer, "Only buyer can confirm the transaction.");

state = State.Locked;

}

function completeTransaction() public inState(State.Locked) {

require(msg.sender == buyer, "Only buyer can complete the transaction.");

seller.transfer(value);

state = State.Inactive;

}

function abortTransaction() public inState(State.Created) {

require(msg.sender == buyer, "Only buyer can abort the transaction.");

buyer.transfer(address(this).balance);

state = State.Inactive;

}

}

These are the different solidity code write the give me code for testing for each contract I am using Remix IDE to execute this give me code for testing in solidity.  
  
**Summary of prompt 3**: I have provided all my solidity code for three different contract from which I have asked GPT to give me code to test the contracts and I have mentioned to give code in solidity .   
  
  
**Response 3:**

**“Testing UserContract”:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "remix\_tests.sol"; // this import is automatically injected by Remix.

import "./UserContract.sol"; // Adjust the path to your UserContract.

contract TestUserContract {

UserContract userContract;

function beforeAll() public {

userContract = new UserContract();

}

/// Test if user registration works

function testUserRegistration() public {

userContract.registerUser("Alice", "Buyer");

UserContract.User memory user = userContract.getUser(msg.sender);

Assert.equal(user.name, "Alice", "User name should be Alice");

Assert.equal(user.role, "Buyer", "User role should be Buyer");

}

/// Test duplicate registration prevention

function testFailRegisterSameUser() public {

userContract.registerUser("Bob", "Seller");

bool result = true;

try {

userContract.registerUser("Bob", "Agent");

} catch {

result = false;

}

Assert.equal(result, false, "Should not allow re-registration");

}

}

**Testing PropertyContract:**// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "remix\_tests.sol"; // automatically injected by Remix.

import "./PropertyContract.sol"; // Adjust the path to your PropertyContract.

contract TestPropertyContract {

PropertyContract propertyContract;

function beforeAll() public {

propertyContract = new PropertyContract();

}

/// Test property listing

function testPropertyListing() public {

propertyContract.listProperty("Dream Home", "123 Main St", 100000);

PropertyContract.Property memory property = propertyContract.properties(1);

Assert.equal(property.title, "Dream Home", "Property title should be 'Dream Home'");

Assert.equal(property.location, "123 Main St", "Property location should be '123 Main St'");

Assert.equal(property.price, 100000, "Property price should be 100000");

Assert.isTrue(property.isListed, "Property should be listed");

}

/// Test updating property price

function testUpdatePropertyPrice() public {

propertyContract.listProperty("Lake House", "456 Lake Rd", 150000);

propertyContract.updateProperty(2, 160000);

PropertyContract.Property memory updatedProperty = propertyContract.properties(2);

Assert.equal(updatedProperty.price, 160000, "Property price should be updated to 160000");

}

}

**Testing TransactionContract:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "remix\_tests.sol"; // automatically injected by Remix.

import "./TransactionContract.sol"; // Adjust the path to your TransactionContract.

contract TestTransactionContract {

TransactionContract transactionContract;

function beforeAll() public {

transactionContract = new TransactionContract();

}

/// Test initiating a transaction

function testInitiateTransaction() public {

transactionContract.initiateTransaction{value: 100000}(1, payable(address(0x123)));

Assert.equal(uint(transactionContract.state()), uint(TransactionContract.State.Created), "State should be 'Created'");

Assert.equal(transactionContract.value(), 100000, "Transaction value should be 100000");

}

/// Test confirming a transaction

function testConfirmTransaction() public {

transactionContract.confirmTransaction();

Assert.equal(uint(transactionContract.state()), uint(TransactionContract.State.Locked), "State should be 'Locked'");

}

/// Test completing a transaction

function testCompleteTransaction() public {

transactionContract.completeTransaction();

Assert.equal(uint(transactionContract.state()), uint(TransactionContract.State.Inactive), "State should be 'Inactive'");

}

}  
  
Observation(Testing Response):  
  
In the provided code after the changing the file path of the respective contracts there are some issues in the provided code.  
  
I have addressed the assertion issue in property contract testing part and also the provided code has a try catch part for one of the contract in which the try catch part is trying to address the external functional call since in the provided code has using the try catch block just to address the issue within the same contract it didn’t work properly.

Result of all contracts:

User Contract:  
**[vm]**

**from:** 0x5B3...eddC4

**to:** UserContract.(constructor)

**value:** 0 wei

**data:** 0x608...90033

**logs:** 0

**hash:** 0x137...84e05

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x1372971c44443371ab1b92e38338721264b82196f4168fd25d1f7fa250384e05 |
| **block hash** | 0xecc38b48930472f2ba20ddc361698e1d7adc993b946232cd3ec57764ce7e9093 |
| **block number** | 5 |
| **contract address** | 0x7EF2e0048f5bAeDe046f6BF797943daF4ED8CB47 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | UserContract.(constructor) |
| **gas** | 869744 gas |
| **transaction cost** | 756299 gas |
| **execution cost** | 652281 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

Property Contract:  
  
**[vm]**

**from:** 0x5B3...eddC4

**to:** PropertyContract.(constructor)

**value:** 0 wei

**data:** 0x608...90033

**logs:** 0

**hash:** 0x043...3ad7b

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x04322e5bd7620e85ebcad7931331ba3b1e08bb6f5f92b7c3c2421a796743ad7b |
| **block hash** | 0x0d458102652affaf332aa438a772814cd20f0e5019e67aecbc1f4622fdd86716 |
| **block number** | 6 |
| **contract address** | 0xDA0bab807633f07f013f94DD0E6A4F96F8742B53 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | PropertyContract.(constructor) |
| **gas** | 1034333 gas |
| **transaction cost** | 899420 gas |
| **execution cost** | 785016 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

Transaction Contract:  
**[vm]**

**from:** 0x5B3...eddC4

**to:** TransactionContract.(constructor)

**value:** 0 wei

**data:** 0x608...90033

**logs:** 0

**hash:** 0xee0...84f1c

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xee01ec9d8c4df8cd9b839e5042b063ce631131037345d19a410c0a6e4e184f1c |
| **block hash** | 0x72e6529e86fd05383ce5c7fe72e7d073f701fcfb2fe86266d0d603910c43913c |
| **block number** | 7 |
| **contract address** | 0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | TransactionContract.(constructor) |
| **gas** | 769978 gas |
| **transaction cost** | 669546 gas |
| **execution cost** | 574004 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**Testing Part:**

**User Contract:**

**[vm]**

**from:** 0x5B3...eddC4

**to:** Assert.(constructor)

**value:** 0 wei

**data:** 0x611...90033

**logs:** 0

**hash:** 0x3c2...de71f

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x3c25908085c6964da8df106b13302c8f6a2f26591cb73a67c565c2fb411de71f |
| **block hash** | 0x07145b0bd26daf80e18659f04022373c39f6687428d0ae357a3f162b0bfd0bd9 |
| **block number** | 16 |
| **contract address** | 0xD4Fc541236927E2EAf8F27606bD7309C1Fc2cbee |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | Assert.(constructor) |
| **gas** | 1989966 gas |
| **transaction cost** | 1730405 gas |
| **execution cost** | 1554047 gas |
| **input** | 0x611...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

creation of TestUserContract pending...

**[vm]**

**from:** 0x5B3...eddC4

**to:** TestUserContract.(constructor)

**value:** 0 wei

**data:** 0x608...90033

**logs:** 0

**hash:** 0x768...72e92

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x768960bb6264f4a5259757e2ea5c6f09786618529e0a59a3d9c4f2c921472e92 |
| **block hash** | 0x235db6f5e287240d2440af6f55b16e02ef85dc7e67c1a8b5cfdb4d5e4e56ba93 |
| **block number** | 17 |
| **contract address** | 0x5FD6eB55D12E759a21C09eF703fe0CBa1DC9d88D |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | TestUserContract.(constructor) |
| **gas** | 1721446 gas |
| **transaction cost** | 1496909 gas |
| **execution cost** | 1340595 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**>**

**Testing Property contract:**

**[vm]**

**from:** 0x5B3...eddC4

**to:** Assert.(constructor)

**value:** 0 wei

**data:** 0x611...90033

**logs:** 0

**hash:** 0xdb3...81c3b

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xdb3b4b47fba903c089a7e1b4371662ec3234a967e16150a377bb924314781c3b |
| **block hash** | 0x9251727c5518b7555284ecfeadf64da30c4720fb9b360413c11074ec91b5dc52 |
| **block number** | 14 |
| **contract address** | 0xaE036c65C649172b43ef7156b009c6221B596B8b |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | Assert.(constructor) |
| **gas** | 1989966 gas |
| **transaction cost** | 1730405 gas |
| **execution cost** | 1554047 gas |
| **input** | 0x611...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

creation of TestPropertyContract pending...

**[vm]**

**from:** 0x5B3...eddC4

**to:** TestPropertyContract.(constructor)

**value:** 0 wei

**data:** 0x608...90033

**logs:** 0

**hash:** 0x178...4d4d1

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x178b4c79a8deea54421e567006bf29bf3bcff2652b2da79cb081e8d38914d4d1 |
| **block hash** | 0xe8862d713d26a5ffe88dd74c07024164535b146ed3002838a64e129dd8d985b7 |
| **block number** | 15 |
| **contract address** | 0x9d83e140330758a8fFD07F8Bd73e86ebcA8a5692 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | TestPropertyContract.(constructor) |
| **gas** | 2112682 gas |
| **transaction cost** | 1837114 gas |
| **execution cost** | 1655534 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**Testing Transaction Contract:**

**[vm]**

**from:** 0x5B3...eddC4

**to:** Assert.(constructor)

**value:** 0 wei

**data:** 0x611...90033

**logs:** 0

**hash:** 0x5a6...3612a

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x5a6cbffce7d814b57ed74237d6eaae5ece289a9c0b81cce311830d9fcad3612a |
| **block hash** | 0x914ddb64e3e0b15922394316bbddf4e8b3a718891f252b9d8d8886eab9c83676 |
| **block number** | 12 |
| **contract address** | 0xb27A31f1b0AF2946B7F582768f03239b1eC07c2c |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | Assert.(constructor) |
| **gas** | 1989966 gas |
| **transaction cost** | 1730405 gas |
| **execution cost** | 1554047 gas |
| **input** | 0x611...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

creation of TestTransactionContract pending...

**[vm]**

**from:** 0x5B3...eddC4

**to:** TestTransactionContract.(constructor)

**value:** 0 wei

**data:** 0x608...90033

**logs:** 0

**hash:** 0x61a...81554

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x61a4f5b7d73beb64c0930b44ddf767059fab2dfe91f8836e34dec35d45f81554 |
| **block hash** | 0x61e87eb2bafc3c45d88b2e69ad425f821eed1f373cc0ece6c13e1f1a197c8b6d |
| **block number** | 13 |
| **contract address** | 0xcD6a42782d230D7c13A74ddec5dD140e55499Df9 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | TestTransactionContract.(constructor) |
| **gas** | 1514377 gas |
| **transaction cost** | 1316849 gas |
| **execution cost** | 1174419 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |