**Fine Tuning Requirement**

**Requirements collected from different websites:  
  
1. User Registration and Authentication**

**Collected requirements from Hedera:**

1. Users must register and authenticate to use the smart contract.
2. Use multi-factor authentication (MFA) for extra security.
3. Store user credentials securely with encryption methods like public-private key pairs or OAuth.

**Collected requirements from Imaginovation:**

1. Ensure secure registration and authentication for all users (buyers, sellers, agents, lawyers).
2. Use strong authentication methods to prevent unauthorized access.

**Collected requirements from Light IT:**

1. Implement user registration and authentication processes.
2. Use advanced security measures for storing user credentials.
3. Support multi-factor authentication (MFA).

**Collected requirements from RisingMax:**

1. Secure registration process for all participants (buyers, sellers, agents, and lawyers).
2. Use advanced encryption techniques for storing user credentials.

**2. Property Listing Management**

**Collected requirements from Hedera:**

1. Sellers or agents can list properties, set prices, and define sale or lease terms.
2. Ability to update or remove listings by the owner or authorized agents.
3. Include property search functionality with filters like location and price range.

**Collected requirements from Imaginovation:**

1. Provide property listing management features for sellers and agents.
2. Allow updating and removing property listings.
3. Implement search and notification features for interested buyers.

**Collected requirements from Light IT:**

1. Enable property listing by sellers and agents with detailed terms.
2. Allow property listings to be updated or removed.
3. Include advanced search functionalities for buyers.

**Collected requirements from RisingMax:**

1. Implement secure and transparent property listing services.
2. Use decentralized property listings for better visibility and data integrity.

**3. Transaction Handling**

**Collected requirements from Hedera:**

1. Automate transactions when predefined conditions are met.
2. Support for escrow services to secure funds until all conditions are satisfied.
3. Clearly define conditions that trigger automated transactions.

**Collected requirements from Imaginovation:**

1. Facilitate automated transactions based on predefined conditions.
2. Implement escrow services for securing transaction funds.
3. Define triggers for transaction automation, such as payment and document verification.

**Collected requirements from Light IT:**

1. Automate transaction processes when conditions are met.
2. Provide escrow services to manage funds securely.
3. Specify conditions for triggering transactions.

**Collected requirements from RisingMax:**

1. Use blockchain technology to automate real estate transactions and improve cash flow management.
2. Implement smart contracts for secure and quick transaction processing.

**4. Document Handling and Verification**

**Collected requirements from Hedera:**

1. Upload and verify necessary legal documents like title deeds and inspection reports.
2. Ensure only authorized personnel can access sensitive documents.
3. Use digital signatures to ensure document authenticity.

**Collected requirements from Imaginovation:**

1. Support uploading and verification of legal documents.
2. Restrict access to sensitive documents to authorized users.
3. Implement digital signatures for document verification.

**Collected requirements from Light IT:**

1. Enable the upload and verification of legal documents.
2. Ensure document access is limited to authorized personnel.
3. Use digital signatures to verify documents.

**Collected requirements from RisingMax:**

1. Create digital entries for all paperwork and store them on a decentralized ledger.
2. Use blockchain for secure and transparent document handling.

**5. Payment Processing**

**Collected requirements from Hedera:**

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

**Collected requirements from Imaginovation:**

1. Process cryptocurrency payments and handle exchange rate adjustments.
2. Provide transaction receipts and maintain transaction history.

**Collected requirements from Light IT:**

1. Facilitate cryptocurrency transactions with exchange rate adjustments.
2. Generate and store receipts and transaction history.

**Collected requirements from RisingMax:**

1. Use smart contracts to automate financial transactions and ensure security.
2. Enable quick and secure payments through blockchain technology.

**6. Dispute Resolution**

**Collected requirements from Hedera:**

1. Mechanisms to handle disputes through third-party arbitration smart contracts or predefined rules.
2. Include an arbitration process with third-party involvement if necessary.
3. Clearly define steps and criteria for resolving disputes.

**Collected requirements from Imaginovation:**

1. Implement dispute resolution mechanisms, possibly involving third-party arbitration.
2. Define arbitration processes and criteria for resolving disputes.

**Collected requirements from Light IT:**

1. Provide dispute resolution mechanisms with third-party arbitration options.
2. Specify steps for arbitration and dispute resolution.

**Collected requirements from RisingMax:**

1. Use blockchain to ensure transparency and trust in the dispute resolution process.
2. Implement smart contracts for automated dispute handling.

**Non-functional Requirements**

**Security**

**Collected requirements from Hedera:**

1. Secure against common vulnerabilities like re-entrancy and overflow/underflow.
2. Regular security audits and updates to address new vulnerabilities.
3. Specify quarterly security audits and vulnerability assessments.

**Collected requirements from Imaginovation:**

1. Ensure security against common vulnerabilities.
2. Conduct regular security audits and updates.

**Collected requirements from Light IT:**

1. Protect against vulnerabilities.
2. Schedule regular security audits and updates.

**Collected requirements from RisingMax:**

1. Implement decentralized and transparent systems to prevent fraud.
2. Regular security updates and audits to maintain system integrity.

**Performance**

**Collected requirements from Hedera:**

1. Handle transactions quickly, even during high network congestion.
2. Optimized for minimal gas usage to reduce transaction costs.
3. Aim to process up to 1,000 transactions per minute.
4. Ensure transaction processing time remains under 2 seconds under normal conditions.

**Collected requirements from Imaginovation:**

1. Maintain high performance even under network congestion.
2. Minimize gas usage to reduce costs.

**Collected requirements from Light IT:**

1. Ensure quick transaction handling.
2. Optimize gas usage for cost efficiency.

**Collected requirements from RisingMax:**

1. Use blockchain technology to ensure fast and secure transactions.
2. Optimize system performance to handle large volumes of transactions efficiently.

**Scalability**

**Collected requirements from Hedera:**

1. Handle growing transaction volumes as the user base increases.
2. Support additional property types and transaction structures.
3. Handle a 10 percent increase in transaction volume per month without performance degradation.

**Collected requirements from Imaginovation:**

1. Ensure scalability to manage increasing transactions.
2. Support diverse property types and transaction structures.

**Collected requirements from Light IT:**

1. Scale to accommodate increasing transactions.
2. Support various transaction structures.

**Collected requirements from RisingMax:**

1. Implement scalable blockchain solutions to handle large transaction volumes.
2. Ensure system scalability to support future growth.

**Usability**

**Collected requirements from Hedera:**

1. Provide a user-friendly interface for interacting with the smart contract.
2. Maintain comprehensive logs and alerts for significant actions and changes.
3. Aim for a user satisfaction rating of 90 percent or higher.
4. Ensure logs include user actions, system events, and are retained for at least one year.

**Collected requirements from Imaginovation:**

1. Ensure the interface is user-friendly.
2. Maintain detailed logs and alerts for important actions.

**Collected requirements from Light IT:**

1. Develop a user-friendly interface.
2. Keep comprehensive logs and alerts.

**Collected requirements from RisingMax:**

1. Provide a user-friendly and intuitive interface for users.
2. Ensure the system maintains detailed logs and alerts for significant actions.

These combined requirements cover various aspects necessary for developing a robust and efficient smart contract solution for real estate transactions. For more detailed information, you can refer to the original sources:

[Hedera](https://hedera.com)

[Imaginovation](https://imaginovation.net)

[Light IT](https://light-it.net)

[RisingMax](https://risingmax.com)

[Apriorit](https://apriorit.com)

**Prompt:(GPT-4o)**  
I have collected different requirements from different website and also requirements from different LLM(GPT-4o & COPILOT) for smart contract project. I’ll give you both the requirements generate me a response which should include all the valid requirements from both the source(LLM & WEB) and generate final requirements which I can use it for designing and implementation of smart contract project for real estate.  
  
Requirements from website:  
 Given the requirement collected from web.

Requirements from LLM:

Given requirements from LLM.  
  
**Response:  
  
Final Requirement:**  
Final Requirements for Smart Contract Project

**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.

Enhancements:

* The blockchain-based solution enhances transparency, security, and efficiency over traditional systems. Blockchain provides an immutable ledger that ensures all transactions are transparent and secure, significantly reducing the risk of fraud. The smart contract automates processes, leading to faster and more efficient transactions.
* For instance, a typical property purchase involves listing a property, buyer offers, and escrow services—all facilitated automatically. Specific goals include achieving 100 percent transaction transparency, reducing transaction times by 50 percent, and ensuring data security through advanced encryption.
* Emphasize how blockchain ensures a decentralized approach to data storage, reducing the risk of data tampering and enhancing data integrity.
* Highlight potential cost savings through reduced need for intermediaries and faster transaction processing times.

**2. Overall Description**

User Needs

* Property buyers and 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.

Enhancements:

User Personas:

* Buyers: Seek reliable property data and secure transactions.
* Sellers: Need a platform to easily list and manage property sales.
* Agents: Require tools to manage multiple listings and client communications.
* Legal Entities: Need access to immutable transaction records for compliance.
* Dependencies and Contingencies: If property data is not accurately digitized, the system will include a verification step before listing. Additionally, if users lack compatible wallets, a guide will be provided to set up and use them.

- Include more detailed user personas, such as foreign investors needing multilingual support and currency conversion.

- Consider the need for integration with existing real estate databases and property management systems for seamless data exchange.

**3. Functional Requirements**

*1. User Registration and Authentication*

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

Enhancements:

* Utilize multi-factor authentication (MFA) and secure storage of user credentials using encryption methods such as public-private key pairs or OAuth.
* Incorporate biometric authentication options (e.g., fingerprint, facial recognition) for enhanced security.

*2. Property Listing Management*

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

Enhancements:

* Include property search functionality with filters (e.g., location, price range) and notifications for interested buyers.
* Implement advanced search features using AI to recommend properties based on buyer preferences and past interactions.

*3. Transaction Handling*

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

**Enhancements:**

* Clearly define conditions such as successful payment and document verification that trigger automated transactions.
* Support multi-signature wallets for escrow services to enhance security and trust.

*4. Document Handling and Verification*

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

Enhancements:

* Detail the role of authorized personnel (e.g., legal experts) in verifying documents. Use digital signatures to ensure authenticity.
* Implement AI-based document verification to automatically detect anomalies and ensure document integrity.

*5. Payment Processing*

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

Enhancements:

* Include support for multiple cryptocurrencies and fiat currency conversions.

6. Dispute Resolution

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

Enhancements:

* Include an arbitration process with third-party involvement if necessary. Clearly define the steps and criteria for resolving disputes.
* Provide an option for mediation before arbitration to resolve disputes amicably.

**4. 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.

Enhancements:

* Specify quarterly security audits and vulnerability assessments.
* Employ real-time monitoring and alerting systems to detect and respond to security threats promptly.

*2. Performance*

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

Enhancements:

- Aim to process up to 1,000 transactions per minute. Ensure that transaction processing time remains under 2 seconds under normal conditions.

- Use off-chain solutions like state channels or sidechains to improve transaction throughput and reduce costs.

*3. Scalability*

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

Enhancements:

* System should handle a 10 percent increase in transaction volume per month without performance degradation.
* Design the system to scale horizontally by adding more nodes as transaction volume grows.

*4. Usability*

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

Enhancements:

- Aim for a user satisfaction rating of 90 percent or higher. Ensure comprehensive logs include user actions, system events, and are retained for at least one year.

- Conduct regular user testing sessions to gather feedback and continuously improve the interface.

*5. 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.

Enhancements:

* Focus on consistency, simplicity, and responsiveness. Ensure compliance with accessibility standards such as WCAG 2.1.
* Implement multilingual support to cater to a global user base.

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.

Enhancements:

* Provide API endpoints for seamless integration with third-party applications.

**6. 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.

Enhancements:

* Implement a detailed plan for ongoing compliance monitoring and reporting. Regularly update the system to reflect changes in laws and regulations.
* Use encryption methods for data protection and strict access controls to ensure privacy. Regularly review and update data privacy policies.
* Establish partnerships with legal experts to ensure the system remains compliant with evolving regulations.

**Conclusion**

By incorporating these detailed suggestions from both sources, the requirements document will be more comprehensive, user-focused, and aligned with best practices for smart contract development. This will ensure a robust, secure, and efficient solution for managing real estate transactions.

**Comparison of output of the requirement from LLM and requirement from multiple resource.**

**Prompt:(GPT-4o)**

For this final requirement of smart contract project for real estate give me complete implementation in solidity code.  
  
**Response:**To implement the detailed requirements for a smart contract project for real estate using Solidity, we will create a series of contracts that handle user registration, property listing management, transaction handling, document handling, payment processing, and dispute resolution.

Here is a high-level implementation approach:

**1. User Registration and Authentication**

This smart contract handles user registration and authentication, incorporating multi-factor authentication and secure storage of credentials.

```solidity

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract UserRegistration {

enum Role { Buyer, Seller, Agent, Lawyer }

struct User {

string name;

address userAddress;

Role role;

bool registered;

}

mapping(address => User) public users;

event UserRegistered(address indexed userAddress, string name, Role role);

function registerUser(string memory \_name, Role \_role) public {

require(!users[msg.sender].registered, "User already registered");

users[msg.sender] = User(\_name, msg.sender, \_role, true);

emit UserRegistered(msg.sender, \_name, \_role);

}

}

```

**2. Property Listing Management**

This contract allows sellers and agents to list properties, update or remove listings, and includes search functionality.

```solidity

pragma solidity ^0.8.0;

contract PropertyListing {

struct Property {

uint id;

string name;

string location;

uint price;

address owner;

bool forSale;

}

mapping(uint => Property) public properties;

uint public propertyCount;

event PropertyListed(uint id, string name, string location, uint price, address owner);

event PropertyUpdated(uint id, string name, string location, uint price);

event PropertyRemoved(uint id);

function listProperty(string memory \_name, string memory \_location, uint \_price) public {

propertyCount++;

properties[propertyCount] = Property(propertyCount, \_name, \_location, \_price, msg.sender, true);

emit PropertyListed(propertyCount, \_name, \_location, \_price, msg.sender);

}

function updateProperty(uint \_id, string memory \_name, string memory \_location, uint \_price) public {

Property storage property = properties[\_id];

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

property.name = \_name;

property.location = \_location;

property.price = \_price;

emit PropertyUpdated(\_id, \_name, \_location, \_price);

}

function removeProperty(uint \_id) public {

Property storage property = properties[\_id];

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

delete properties[\_id];

emit PropertyRemoved(\_id);

}

}

```

**3. Transaction Handling**

This contract facilitates transactions between buyers and sellers, supporting escrow services.

```solidity

pragma solidity ^0.8.0;

contract PropertyTransaction {

enum Status { Pending, Completed, Cancelled }

struct Transaction {

uint id;

uint propertyId;

address buyer;

address seller;

uint amount;

Status status;

}

mapping(uint => Transaction) public transactions;

uint public transactionCount;

event TransactionInitiated(uint id, uint propertyId, address buyer, address seller, uint amount);

event TransactionCompleted(uint id);

event TransactionCancelled(uint id);

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

require(msg.value == \_amount, "Insufficient funds sent");

transactionCount++;

transactions[transactionCount] = Transaction(transactionCount, \_propertyId, msg.sender, \_seller, \_amount, Status.Pending);

emit TransactionInitiated(transactionCount, \_propertyId, msg.sender, \_seller, \_amount);

}

function completeTransaction(uint \_id) public {

Transaction storage transaction = transactions[\_id];

require(transaction.status == Status.Pending, "Transaction not pending");

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

transaction.status = Status.Completed;

payable(transaction.seller).transfer(transaction.amount);

emit TransactionCompleted(\_id);

}

function cancelTransaction(uint \_id) public {

Transaction storage transaction = transactions[\_id];

require(transaction.status == Status.Pending, "Transaction not pending");

require(transaction.buyer == msg.sender || transaction.seller == msg.sender, "Only buyer or seller can cancel the transaction");

transaction.status = Status.Cancelled;

payable(transaction.buyer).transfer(transaction.amount);

emit TransactionCancelled(\_id);

}

}

```

**4. Document Handling and Verification**

This contract manages the upload and verification of necessary legal documents.

```solidity

pragma solidity ^0.8.0;

contract DocumentHandling {

struct Document {

uint id;

string documentHash;

address uploader;

bool verified;

}

mapping(uint => Document) public documents;

uint public documentCount;

event DocumentUploaded(uint id, string documentHash, address uploader);

event DocumentVerified(uint id);

function uploadDocument(string memory \_documentHash) public {

documentCount++;

documents[documentCount] = Document(documentCount, \_documentHash, msg.sender, false);

emit DocumentUploaded(documentCount, \_documentHash, msg.sender);

}

function verifyDocument(uint \_id) public {

Document storage document = documents[\_id];

require(!document.verified, "Document already verified");

document.verified = true;

emit DocumentVerified(\_id);

}

}

```

**5. Payment Processing**

This contract handles cryptocurrency transfers and maintains transaction history.

```solidity

pragma solidity ^0.8.0;

contract PaymentProcessing {

struct Payment {

uint id;

address from;

address to;

uint amount;

uint timestamp;

}

mapping(uint => Payment) public payments;

uint public paymentCount;

event PaymentMade(uint id, address from, address to, uint amount, uint timestamp);

function makePayment(address \_to) public payable {

require(msg.value > 0, "No funds sent");

paymentCount++;

payments[paymentCount] = Payment(paymentCount, msg.sender, \_to, msg.value, block.timestamp);

emit PaymentMade(paymentCount, msg.sender, \_to, msg.value, block.timestamp);

payable(\_to).transfer(msg.value);

}

}

```

**6. Dispute Resolution**

This contract handles dispute resolution through arbitration or predefined rules.

```solidity

pragma solidity ^0.8.0;

contract DisputeResolution {

enum ResolutionStatus { Pending, Resolved, Rejected }

struct Dispute {

uint id;

uint transactionId;

address complainant;

string details;

ResolutionStatus status;

}

mapping(uint => Dispute) public disputes;

uint public disputeCount;

event DisputeFiled(uint id, uint transactionId, address complainant, string details);

event DisputeResolved(uint id);

event DisputeRejected(uint id);

function fileDispute(uint \_transactionId, string memory \_details) public {

disputeCount++;

disputes[disputeCount] = Dispute(disputeCount, \_transactionId, msg.sender, \_details, ResolutionStatus.Pending);

emit DisputeFiled(disputeCount, \_transactionId, msg.sender, \_details);

}

function resolveDispute(uint \_id) public {

Dispute storage dispute = disputes[\_id];

require(dispute.status == ResolutionStatus.Pending, "Dispute already resolved or rejected");

dispute.status = ResolutionStatus.Resolved;

emit DisputeResolved(\_id);

}

function rejectDispute(uint \_id) public {

Dispute storage dispute = disputes[\_id];

require(dispute.status == ResolutionStatus.Pending, "Dispute already resolved or rejected");

dispute.status = ResolutionStatus.Rejected;

emit DisputeRejected(\_id);

}

}

```

Compilation and Deployment

Ensure you have installed the necessary tools and dependencies, like Solidity and a development environment such as Remix IDE or Truffle. The smart contracts can be compiled and deployed to an Ethereum network or a compatible blockchain.

This implementation provides a basic structure for the smart contracts required to manage real estate transactions on the blockchain. Depending on your specific requirements, further enhancements and integrations can be added.

**Output of updated requirements smart contract:**

**Contract 1**

**User Registration and Authentication Record:**

**[vm]**

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

**to:** UserRegistration.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0xb8c...33d20

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xb8c01dac9f4f26778c12587868b7056a9de836eb6a3fa781e79d88de0b233d20 |
| **block hash** | 0x9e3c6cd3d4e4ee11a99475fd5762a460fad7c4c298cf1cd02bc6871246fd02bc |
| **block number** | 18 |
| **contract address** | 0x7b96aF9Bd211cBf6BA5b0dd53aa61Dc5806b6AcE |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | UserRegistration.(constructor) |
| **gas** | 742465 gas |
| **transaction cost** | 645621 gas |
| **execution cost** | 549779 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**Contract 2:  
Property Listing Management:**

**[vm]**

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

**to:** PropertyListing.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x623...e105d

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x62330505aec32d2f3237f39f5982b96a37468f1362479dc0f2eb3b64346e105d |
| **block hash** | 0x3b80261a313d5776140d70b0b0ef0f39f8b357b79c44b7175da316878c3553fd |
| **block number** | 20 |
| **contract address** | 0x5e17b14ADd6c386305A32928F985b29bbA34Eff5 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | PropertyListing.(constructor) |
| **gas** | 1073713 gas |
| **transaction cost** | 933663 gas |
| **execution cost** | 817449 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**Contract 3**

**Transaction Handling:**

**[vm]**

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

**to:** PropertyTransaction.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x2db...242f6

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x2db9c7676d70563874f4d4d6305a03ffbb84244c5f479adc07e7366d183242f6 |
| **block hash** | 0x02bfdbad2bf8ca0fbcb19cc8d992eb5dc026fe50727bc6c9b45a837d491ecf9f |
| **block number** | 21 |
| **contract address** | 0xe2899bddFD890e320e643044c6b95B9B0b84157A |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | PropertyTransaction.(constructor) |
| **gas** | 899383 gas |
| **transaction cost** | 782072 gas |
| **execution cost** | 677706 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**Contract 4:  
Document Handling and Verification**

**[vm]**

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

**to:** DocumentHandling.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x242...7a8cf

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x2429c6049a9dba8f68bb41985eda0aa158cb35c4499f902b4bb77f248de7a8cf |
| **block hash** | 0x0434e908cbbf46b1a28c778c055c1072fcc509d81e6b3506a260320e26c69b7c |
| **block number** | 22 |
| **contract address** | 0x1c91347f2A44538ce62453BEBd9Aa907C662b4bD |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | DocumentHandling.(constructor) |
| **gas** | 749360 gas |
| **transaction cost** | 651617 gas |
| **execution cost** | 555385 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**>**

**Contract 5:**

**Payment Processing**

**[vm]**

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

**to:** PaymentProcessing.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x07e...a70ec

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x07edf7fd42b2a19477a63d7c46ca0a4193c67161b91895d8814ff6108a4a70ec |
| **block hash** | 0xbdab5565574b47150e2a7b3e79fe5d6948bf91c75f4ae81759b3a2ce9c8d51e0 |
| **block number** | 23 |
| **contract address** | 0x93f8dddd876c7dBE3323723500e83E202A7C96CC |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | PaymentProcessing.(constructor) |
| **gas** | 423747 gas |
| **transaction cost** | 368475 gas |
| **execution cost** | 292729 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**Contract 6:  
Dispute Resolution**

**[vm]**

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

**to:** DisputeResolution.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0xecb...a0f8d

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xecbb9e65ae0b4e09017fb6582eeed604ea51d72802639644075636603e8a0f8d |
| **block hash** | 0xa48984b7adc24a3ef452368d7d668e5af0f9fbe8b78f4a2e96b416279036845f |
| **block number** | 24 |
| **contract address** | 0x5A86858aA3b595FD6663c2296741eF4cd8BC4d01 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | DisputeResolution.(constructor) |
| **gas** | 887506 gas |
| **transaction cost** | 771744 gas |
| **execution cost** | 667500 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**>**

**Observation:**

While comparing the updated requirement with the previous requirement:

For the previous requirement’s implementation we worked only with three different contracts:

1. USER
2. PROPERTY
3. TRANSACTION

Whereas for the updated requirement:

1. User Registration and Authentication.
2. Property Listing Management
3. Transaction Handling
4. Document Handling and Verification
5. Payment Processing
6. Dispute resolution.

**Comparison of the common contract’s transactions for both the requirement:**

1. User Contract:
2. **[vm]**
3. **from:** 0x5B3...eddC4
4. **to:** UserContract.(constructor)
5. **value:** 0 wei
6. **data:** 0x608...90033
7. **logs:** 0
8. **hash:** 0xc7f...df851
9. **Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xc7fffabf3f3a502264a3c85cce204de2001f91a934369bdfbb7311de1acdf851 |
| **block hash** | 0x05010b4ee80cf86a9855efb09c508ce111d721025be19f82593999d4f9590426 |
| **block number** | 25 |
| **contract address** | 0x406AB5033423Dcb6391Ac9eEEad73294FA82Cfbc |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | UserContract.(constructor) |
| **gas** | 869744 gas |
| **transaction cost** | 756299 gas |
| **execution cost** | 652281 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

User Registration:

**[vm]**

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

**to:** UserRegistration.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x0e2...df733

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x0e2c42672668dbee4b6733e00c112ebe060885b7c7209dee327b8499055df733 |
| **block hash** | 0x33cd382a74fb3730b16a60d9cf046b47e795f1e689949ea62f1cb27da7d0c9fc |
| **block number** | 26 |
| **contract address** | 0x4a9C121080f6D9250Fc0143f41B595fD172E31bf |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | UserRegistration.(constructor) |
| **gas** | 742465 gas |
| **transaction cost** | 645621 gas |
| **execution cost** | 549779 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

The `User Registration` contract is more gas-efficient, consuming less gas for deployment and execution, making it cost-effective. In contrast, the `User Contract` might offer richer functionality or greater complexity due to its higher gas usage. Choose `User Registration` for efficiency and `User Contract` for potential advanced features.

2.Property:

**[vm]**

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

**to:** UserContract.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x64b...30671

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x64ba9c80dc2350905cb918a4a2a9b09fad9df6d93f63c2b17fe411a181e30671 |
| **block hash** | 0x969995fd09e1bf38985dc8a61669f31955c0a1030ca3b40cf6e2f3840059c392 |
| **block number** | 27 |
| **contract address** | 0x540d7E428D5207B30EE03F2551Cbb5751D3c7569 |
| **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 listing management:

**[vm]**

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

**to:** PropertyListing.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0xdfb...b394c

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xdfb377b184e5399733b6f33f3f61822fb9ba161d35ae413cd904ab75fb8b394c |
| **block hash** | 0xb8a7f7c46b3dfd9d165ef877e93063fd09957dcef2c5579619f767623c86fc0c |
| **block number** | 28 |
| **contract address** | 0xEf9f1ACE83dfbB8f559Da621f4aEA72C6EB10eBf |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | PropertyListing.(constructor) |
| **gas** | 1073713 gas |
| **transaction cost** | 933663 gas |
| **execution cost** | 817449 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**>**

Output comparison:

The `User Contract` and `Property Listing` contract deployments both succeeded, with `User Contract` using 869744 gas for deployment and `Property Listing` using 1073713 gas, indicating the latter's greater complexity. The `User Contract` had a transaction cost of 756299 gas and an execution cost of 652281 gas, while `Property Listing` had higher costs at 933663 gas and 817449 gas, respectively. Both contracts were deployed from the same address, with no input, output, or logs recorded. Choose `User Contract` for more efficient gas usage, and `Property Listing` if additional features justify the higher gas costs.

Transaction contract:

**[vm]**

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

**to:** TransactionContract.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0xccf...5d825

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0xccf32cf762f70361693e9367baeabb85274cfe5cae5730281bcd1b864c95d825 |
| **block hash** | 0x88f471a75558e297a793693a733ef65e3614a73cfc95fd85e4b02aae72736a31 |
| **block number** | 29 |
| **contract address** | 0x0498B7c793D7432Cd9dB27fb02fc9cfdBAfA1Fd3 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | TransactionContract.(constructor) |
| **gas** | 769978 gas |
| **transaction cost** | 669546 gas |
| **execution cost** | 574004 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

Transaction Handling:

**[vm]**

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

**to:** PropertyTransaction.(constructor)

**value:** 0 wei

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

**logs:** 0

**hash:** 0x01e...989b7

**Debug**

|  |  |
| --- | --- |
| **status** | 0x1 Transaction mined and execution succeed |
| **transaction hash** | 0x01ed2b3815b215e06adaa601399e1c60ea673dff32453fd3c5ce3fb22fc989b7 |
| **block hash** | 0x9eb7c43377feb7b7a5cae276a2f5264e45859c78a9c24fd93fbaa5ff50c89b07 |
| **block number** | 30 |
| **contract address** | 0x38cB7800C3Fddb8dda074C1c650A155154924C73 |
| **from** | 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 |
| **to** | PropertyTransaction.(constructor) |
| **gas** | 899383 gas |
| **transaction cost** | 782072 gas |
| **execution cost** | 677706 gas |
| **input** | 0x608...90033 |
| **decoded input** | {} |
| **decoded output** | - |
| **logs** | [] |

**>**

Output Comparison:

The Property Transaction and Transaction Contract deployments both succeeded, with Property Transaction using more gas (899383 gas) compared to Transaction Contract (769978 gas). The transaction cost for Property Transaction was 782072 gas, while Transaction Contract had a lower transaction cost of 669546 gas. Similarly, the execution cost for Property Transaction was 677706 gas, higher than Transaction Contract’s 574004 gas. Both contracts were deployed from the same address, with no inputs, outputs, or logs recorded.

In summary, Transaction Contract is more gas-efficient, consuming less gas for deployment and execution, making it cost-effective. On the other hand, Property Transaction might offer additional features or complexity due to its higher gas usage. Choose Transaction Contract for efficiency and Property Transaction for potential advanced functionalities.