TABLE OF CONTENTS

01 Problem Definition
02 Project Objectives
03 Methodology
04 Demo
05 Challenges
06 Project Schedule
07 Conclusion
Existing Land Registration System Challenges in Hong Kong:

- **01** Legal uncertainties and complexities in property transactions
- **02** Inconvenience and Higher costs
- **03** Difficulty in secure storage of paper deeds

Opportunities with Blockchain Technology

- **01** Reliable, decentralized, transparent and secure land registration system.
- **02** Adoption of smart contracts on public blockchain platforms
- **03** Potential for enhancing security and maintenance of land data
Literature Review

An overview of the key points discussed in “A Systematic Literature Review on Blockchain for Real Estate Transactions: Benefits, Challenges, Enablers, and Inhibitors” (https://doi.org/10.21203/rs.3.rs-2823844/v1)

Note: Articles were selected from four sources, EBSCO Discovery Service (816), Web of Science (190), IEEE (90) and Springer (261), till May 2022. Data was collected by an exhaustive reading of the 75 articles, published mostly in journals and conferences.

Challenges in Real Estate Transactions
- Inefficiency, high costs, and complex processes.
- Corruption, fraud, and outdated technology.
- Lack of trust, transparency, and certainty.

Benefits of Blockchain in Real Estate
- Increased transparency and security.
- Reduction of intermediaries, fraud, time, and costs.
- Accelerated execution and traceable transactions.

Challenges to overcome
- Lack of regulation and legal enforcement.
- Poor data quality and incomplete information.
- Compliance with data protection regulations.
PROJECT OBJECTIVES

ESTABLISHING A BLOCKCHAIN-BASED LAND REGISTRY

AUTOMATING PROPERTY TRANSACTIONS

ENSURING DATA SECURITY AND PRIVACY
METHODOLOGY

01
System Architectural Design
How the system works as a whole?

02
Smart Contract Design
Initial Implementation of Smart Contracts
System Architectural Design

1. Ethereum Blockchain:

   • Digital Ledger:

     o Ethereum blockchain is a decentralized and distributed ledger that securely records transactions across a network of computers. Each block in the chain contains a list of transactions, and these blocks are linked together using cryptographic hashes, forming a secure and transparent chain of data.
System Architectural Design

1. Ethereum Blockchain:

   • Digital Ledger:
     
     o Ethereum blockchain is a decentralized and distributed ledger that securely records transactions across a network of computers. Each block in the chain contains a list of transactions, and these blocks are linked together using cryptographic hashes, forming a secure and transparent chain of data.

   • Land Transactions:
     
     o In the context of land transactions, each entry on the Ethereum blockchain represents a unique transaction related to real estate, such as buying, selling, or transferring ownership.
2. Smart Contracts (Solidity):

• Digital Agreements:

  o Smart contracts are self-executing contracts with the terms of the agreement directly written into code. Solidity is the programming language we are using to create these smart contracts on the Ethereum platform.
System Architectural Design

2. Smart Contracts (Solidity):

• Digital Agreements:
  
  o Smart contracts are self-executing contracts with the terms of the agreement directly written into code. Solidity is the programming language we are using to create these smart contracts on the Ethereum platform.

• Automation:
  
  o Smart contracts automate processes by executing predefined actions when specific conditions are met. For example, in the context of land transactions, a smart contract could automatically transfer ownership of a piece of land to the buyer once the payment is confirmed, eliminating the need for intermediaries.
System Architectural Design

3. User Interface:

1. **Face of the System:** It provides a user-friendly way for individuals to engage with the underlying technology without needing to understand the complex coding or cryptographic concepts.

2. **Functionality:** In our land transactions scenario, the UI offers features such as a

   - Registration portal for new land transactions,
   - A dashboard for checking ownership details, and
   - Options for initiating and tracking the progress of transactions.
SMART CONTRACT Design

The 4 Main Use Cases:

REGISTER
For Land Registration

VERIFY / VIEW
To verify the authenticity of records

TRANSFER
Land Transfer

SELL
For Property Sales
- Class Overview

Shows how data and functions interact with each other.
Interactions

How Different Functions and Events in our system interact.
`registerLand()` allows an address to register a new parcel of land with its location and a unique parcel ID. Emits a `LandRegistered` event when land is registered.
• **registerLand()** allows an address to register a new parcel of land with its location and a unique parcel ID. Emits a LandRegistered event when land is registered.

• **transferLand()** allows the current owner of a land parcel to transfer it to a new owner. Emits a LandTransferred event when land is transferred.

• **sellLand()** allows a landowner to sell their land to a buyer. This also involves the transfer of digital currency from the buyer to the seller, representing the payment for the land.
- **registerLand()** allows an address to register a new parcel of land with its location and a unique parcel ID. Emits a LandRegistered event when land is registered.

- **transferLand()** allows the current owner of a land parcel to transfer it to a new owner. Emits a LandTransferred event when land is transferred.

- **sellLand()** allows a landowner to sell their land to a buyer. This also involves the transfer of digital currency from the buyer to the seller, representing the payment for the land.

- **verifyLand()** allows anyone to verify the details of a land parcel.

- **listLandsByOwner()** allows for querying all land parcels owned by a specific owner, making it easier to get an overview of one's properties.
Real Estate Blockchain - Land Registration / Tokenization

RealEstateBlockchain is a decentralized application built on a blockchain network designed to revolutionize the real estate industry. By leveraging the power of blockchain technology, RealEstateBlockchain enhances transparency, efficiency, and security in real estate transactions.

Website to our Notion Page: [https://www.notion.so/FYP-2023-2024-d6b32d8](https://www.notion.so/FYP-2023-2024-d6b32d8)

Table of Contents

<table>
<thead>
<tr>
<th>Name</th>
<th>Last commit message</th>
<th>Last commit date</th>
</tr>
</thead>
<tbody>
<tr>
<td>contracts</td>
<td>Add files and scripts for Real Estate Blockchain Registry</td>
<td>last week</td>
</tr>
<tr>
<td>scripts</td>
<td>Add files and scripts for Real Estate Blockchain Registry</td>
<td>last week</td>
</tr>
<tr>
<td>.gitattributes</td>
<td>Add files and scripts for Real Estate Blockchain Registry</td>
<td>last week</td>
</tr>
<tr>
<td>.gitignore</td>
<td>Add files and scripts for Real Estate Blockchain Registry</td>
<td>last week</td>
</tr>
<tr>
<td>brownie-config.yaml</td>
<td>Add .DS_Store to .gitignore</td>
<td>last week</td>
</tr>
</tbody>
</table>
DEMO!

Video Link
Components

Solidity Smart Contract:
- LandRegistry.sol

Metamask Wallet

Sepolia Testnet

Infura EVM

Python Scripts (using brownie framework):
- deploy.py
- register_land.py
- verify_land.py
- list_lands_by_owner.py
- transfer_land.py
- initiate_purchase.py
- confirm_sale.py
MetaMask Accounts

0.9775 SepoliaETH

Buy & Sell  | Send  | Swap  | Bridge  | Portfolio

Tokens

SepoliaETH

0.9775 SepoliaETH

Import tokens
Refresh list
MetaMask support
Infura EVM

real-estate-blockchain-fyp
Last updated on 2024-01-08

Active Endpoints

- **Ethereum** Sepolia

  https://sepolia.infura.io/v3/e6d463bba4864a0db8e779c93b754905

Quick Links
Get help from our community or support team with these links
Solidity Smart Contract

Structs

```
struct Land {
    address payable owner;
    string location;
    string parcelID;
    uint256 price;
    // Uncomment this if you want to include a flag indicating whether a
    // parcel of land is registered or not.
    // bool isRegistered;
}
```
Solidity Smart Contract

Mappings

mapping(string => Land) public lands;
mapping(address => string[]) public ownerToLands;
mapping(string => address payable) public pendingPurchases;
Solidity Smart Contract

Events

event LandRegistered(address indexed owner, string parcelID);

event PurchasePending(
    address indexed buyer,
    string parcelID,
    uint256 price
);

event LandSold(
    address indexed seller,
    address indexed buyer,
    string parcelID,
    uint256 price
);

event LandTransferred(
    address indexed oldOwner,
    address indexed newOwner,
    string parcelID
);
Solidity Smart Contract

Modifiers

```solidity
modifier onlyLandOwner(string memory _parcelID) {
    require(
        lands[_parcelID].owner == msg.sender,
        "Only the current owner can perform this operation."
    );
}
```
function registerLand(
    string memory _location,
    string memory _parcelID,
    uint256 _price
) public {
    // Check if the land parcel is already registered
    require(
        lands[_parcelID].owner == address(0),
        "This land parcel is already registered."
    );

    // Create a new Land struct and store it in the lands mapping
    lands[_parcelID] = Land(
        payable(msg.sender),
        _location,
        _parcelID,
        _price
    );

    // Add the parcelID to the list of lands owned by the sender
    ownerToLands[msg.sender].push(_parcelID);

    // Emit the event
    emit LandRegistered(msg.sender, _parcelID);
}
function transferLand(
    address payable _newOwner,
    string memory _parcelID
) public onlyLandOwner(_parcelID) {
    // Transfer the land to the new owner
    address oldOwner = lands[_parcelID].owner;
    lands[_parcelID].owner = _newOwner;

    // Update the ownerToLands mapping for the old owner
    removeLandFromOwner(oldOwner, _parcelID);

    // Update the ownerToLands mapping for the new owner
    ownerToLands[_newOwner].push(_parcelID);

    // Emit the event
    emit LandTransferred(oldOwner, _newOwner, _parcelID);
}
function initiatePurchase(string memory _parcelID) public payable {
    uint256 landPrice = lands[_parcelID].price;
    require(msg.value == landPrice, "Incorrect Ether value.");
    require(lands[_parcelID].owner != address(0), "Land not registered.");

    pendingPurchases[_parcelID] = payable(msg.sender);
    emit PurchasePending(msg.sender, _parcelID, msg.value);
}
function confirmSale(
    string memory _parcelID
) public onlyLandOwner(_parcelID) {
    address payable buyer = pendingPurchases[_parcelID];
    require(buyer != address(0), "No pending purchase for this land.");

    // Transfer the land to the new owner
    address payable oldOwner = lands[_parcelID].owner;
    lands[_parcelID].owner = buyer;
    ownerToLands[oldOwner].push(_parcelID);
    ownerToLands[buyer].push(_parcelID);

    // Transfer the funds to the previous owner
    uint256 price = lands[_parcelID].price;
    oldOwner.transfer(price);

    // Clear the pending purchase
    delete pendingPurchases[_parcelID];

    emit LandSold(oldOwner, buyer, _parcelID, price);
}
function removeLandFromOwner(
    address _owner,
    string memory _parcelID
) internal {
    uint256 len = ownerToLands[_owner].length;
    for (uint256 i = 0; i < len; i++) {
        if {
            keccak256(bytes(ownerToLands[_owner][i])) ==
            keccak256(bytes(_parcelID))
        } {
            ownerToLands[_owner][i] = ownerToLands[_owner][len - 1];
            ownerToLands[_owner].pop();
            break;
        }
    }
function verifyLand(
  string memory _parcelID
) public view returns (address, string memory, string memory, uint256) {
  // Check if the land parcel is registered
  require(
    lands[_parcelID].owner != address(0),
    "This land parcel is not registered."
  );

  // Return the land details
  return (  
    lands[_parcelID].owner,
    lands[_parcelID].location,
    lands[_parcelID].parcelID,
    lands[_parcelID].price
  );
}
function listLandsByOwner(  
    address _owner  
) public view returns (string[] memory) {
    return ownerToLands[_owner];
}
Real Estate Blockchain Registry

```bash
# Click here to ask Blackbox to help you code faster
1  export PRIVATE_KEY_REGISTRAR=0x049db6e655d056dbde0e004acf241b2a3dff69a656d914a83730bca07ac6d12b
2  export PRIVATE_KEY_BUYER=0xefb55c200007a1da16e9d1ad87d85517caf1d956044b9d9d61f694f27f976adf
3  export WEB3_INFURA_PROJECT_ID=e6d463bba486494db8b779c93b754905
```
# brownie-config.yaml File

```yaml
Real Estate Blockchain Registry  >  🚀 brownie-config.yaml  >  {} networks  >  {} sepolia  >  🌐 host

You, 5 days ago | 1 author (You) | 👨‍💻 Click here to ask Blackbox to help you code faster

dotenv: .env

networks:
  default: sepolia
  sepolia:
    host: https://sepolia.infura.io/v3/${WEB3_INFURA_PROJECT_ID}
    chain_id: 11155111
    explorer: https://sepolia.etherscan.io

wallets:
  from_key_registrar: ${PRIVATE_KEY_REGISTRAR}
  from_key_buyer: ${PRIVATE_KEY_BUYER}
```
def deploy_land_registry():
    account = get_account()
    print(f"Deploying from account: {account}\n"
    land_registry = LandRegistry.deploy("false", account) # 7
    print(f"LandRegistry deployed at: {land_registry.address}\n"

    def get_account():
        active_network = network.show_active()
        print(f"Active network: {active_network}\n"
        if active_network == "development":
            return accounts[0]
        else:
            return accounts.add(config[\"wallets\"])[\"from_key_registrar\"]

    def main():
        deploy_land_registry()
Confirm contract deployment on Etherscan
def register_land(account, _location, _parcelID, _price):
    land_registry = LandRegistry[1]
    tx = land_registry.registerLand(_location, _parcelID, _price, {'from': account})
    tx.wait()
    print(f"Land with parcel ID {_parcelID} registered by {account.address}")
    return tx

main()
def register_land(account, location, parcelID, price):
    land_registry = LandRegistry.get()
    tx = land_registry.register_land(location, parcelID, price, {'from': account})
    tx.wait()
    print(f"Land with parcel ID {parcelID} registered by {account.address}")
    return tx

# Use register account
register_account = accounts.add(config['wallets']['from_key_registrar'])

# Dummy data for demonstration
# location = "% Long Walk Rd, Kennedy Town"
# parcelID = "kennedytown"
# price = 2000000000000000000000 # 0.82 ETH in wei

# Dummy data for demonstration
# location = "SS Sasson Road, Pok Fu Lam"
# parcelID = "pokfulam"
# price = 1000000000000000000000 # 0.82 ETH in wei

register_land(register_account, location, parcelID, price)
```python
# Python Scripts

# verify_land.py

import LandRegistry

def verify_land(parcelID):
    land_registry = LandRegistry()
    return land_registry.verify_land(parcelID)

if __name__ == '__main__':
    # Dummy data for demonstration
    parcelID = "pokfulam"
    landInfo = verify_land(parcelID)
    print("Land Info: Owner: {} Location: {} Parcel ID: {} Prices: {}".format(landInfo[0], landInfo[1], landInfo[2], landInfo[3]))
```

```python
# Python Scripts
# list_lands_by_owner.py

Real Estate Blockchain Registry > scripts > list_lands_by_owner.py > main
You, 1 second ago | author (You)! Click here to ask Blockbox to help you code faster!

```
import LandRegistry, accounts, config

def transfer_land(from_account, new_owner, parcel_id):
    land_registry = LandRegistry[cfg]
    tx = land_registry.transferLand(new_owner, parcel_id, {'from': from_account})
    tx.wait()  # Wait for the transaction to be mined
    print("Land with parcel ID {} transferred to {}.").format(parcel_id, new_owner)
    return tx

if __name__ == '__main__':
    new_owner = accounts.add(config['wallets'][1]['from_key_buyer']).address
    parcel_id = "kennedytown"  # Dummy data for demonstration
    transfer_land(transferer_account, new_owner, parcel_id)

LandRegistry.sol
register_land.py
verify_land.py
list_lands_by_owner.py
transfer_land.py
initiate_purchase.py
confirm_sale.py
real_estate_blockchain.py
Real Estate Blockchain Registry > scripts > transfer_land.py
Python Scripts

initiate_purchase.py

```python
from brownie import LandRegistry, accounts, config, network

def initiate_purchase(buyer, parcel_id, price):
    land_registry = LandRegistry[-1]  # Fetch the latest deployed LandRegistry contract

    # The buyer initiates the land purchase
    tx = land_registry.initiatePurchase(parcel_id, ('from': buyer, 'value': price), tx.wait(1))

    print(f"Purchased for land with parcel ID {parcel_id} initiated by {buyer}.")
    return tx

def main():
    # Use buyer account
    buyer = accounts.add(config['wallets']['from_key_buyer'])

    # Dummy data for demonstration
    parcel_id = 'pokhara103'
    land_registry = LandRegistry[-1]

    # Access the land's details from the contract and then get its price
    land_details = land_registry.lands(parcel_id)
    land_price = land_details[3]

    # Call the initiate purchase function
    initiate_purchase(buyer, parcel_id, land_price)
```

Real Estate Blockchain Registry > scripts > initiate_purchase.py > ...
```bash
python initiate_purchase.py --network sepolia
```

Purchase for land with parcel ID pokhara103 initiated by 0x8798cc3277591434698972B5734C80d1a98B2B83.

def confirm_sale(seller, parcel_id):
    land_registry = LandRegistry[1]  # Fetch the latest deployed LandRegistry contract
    # The seller confirms the sale
    tx = land_registry.confirmSale(parcel_id, {'from': seller})
    tx.wait()
    print("Sale for land with parcel ID (parcel_id) confirmed by {seller}.")
    return tx

def main():
    # Use seller account
    seller = accounts.add(config['wallets']['key_registrar'])
    # Dummy data for demonstration
    parcel_id = "pk"  
    # Call the confirm sale function
    confirm_sale(seller, parcel_id)

if __name__ == '__main__':
    main()
CHALLENGES

INTEGRATING LAND REGISTRATION DATA

LEGAL AND TECHNOLOGICAL COMPLEXITIES

SMART CONTRACT COMPLEXITIES

SECURITY
**PROJECT SCHEDULE**

**Phase 1 Deliverables**
- Project Plan (October 1, 2023)

**Phase 2**
- **Phase 2 Deliverables**
  - First Presentation (January 8-12th, 2024)
  - Interim Project Report (January 21st)

**Phase 3**
- **Key Milestones**
  - Back testing and optimisations
  - Performance testing
  - Final documentation

**Phase 4 Deliverables**
- Final Project Report
- Final Project Presentation (Due date: April 15-19th, 2023)
CONCLUSION

**Blockchain Revolution**
Property Transaction Transformation

**Milestone Achievements**
Data Integration and Implementation of Smart Contracts

**Next Steps**
Further Implementation and Testing

**Future Outlook**
Total Asset Digitalization
THANKS!

Do you have any questions?