Project 3: Decentralized Real Estate Marketplace

Charles Panagopoulos, Alex Toenshoff, Samirah Djachechi

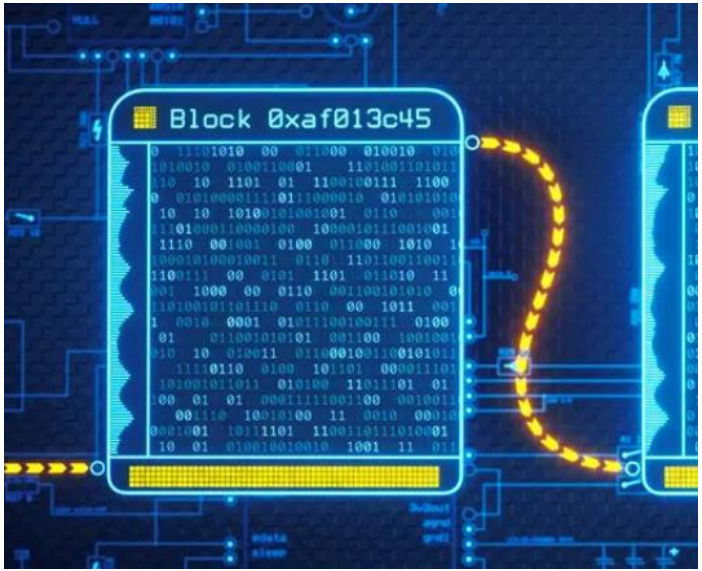
Project Summary

Real Estate has made more millionaires than any other industry. However, it is not accessible to everyone. This is why we built a decentralized application to tokenize and invest in real estate.

Decentralization provides more investment opportunities to more people at a lower cost. As blockchain is rapidly spreading across industries, we believe decentralizing real estate is both valuable for long term decentralization.

Being one of the biggest industries in the world, we are confident that creating a DApp for real estate would be a great investment long term.

Tokenizing Real Estate Using Smart Contracts



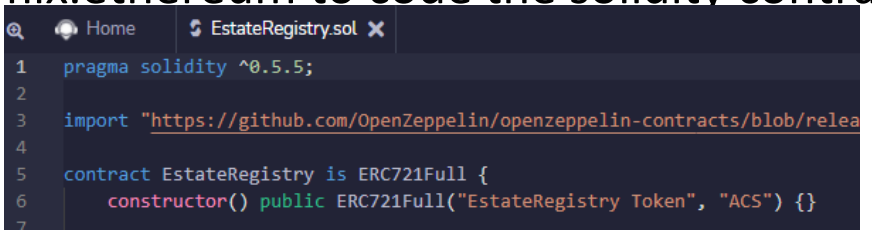
- Through our decentralized REIT, we have allowed people to buy shares of real estate, to take advantage of its appreciation during periods of high inflation.
- By utilizing smart contracts, we look to simplify ownership disputes across an immutable blockchain ledger.
- Smart contracts will also alleviate transaction and legal fees that are notorious in the real estate industry.

Application Core Functionality

- Register Properties
- Tokenize Property Listings
- Appraise & Property Appraisal History
- Invest in Properties by Purchasing Shares of Ownership

Tools Used for Development

remix.ethereum to code the solidity contract



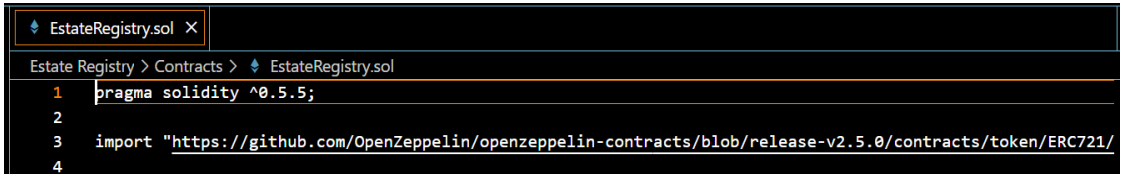
VScode to house app function using python scripting

```
# Set the contract address (this is the address of the deployed contract)
contract_address = os.getenv("SMART_CONTRACT_ADDRESS")

# Get the contract
contract = w3.eth.contract(
    address=contract_address,
    abi=EstateRegistry_abi
)
```

Backend:
Solidity Development

Solidity Contract

Compiler Versioning & OpenZepplin Import



Estate Registry ERC 721 Contract Created

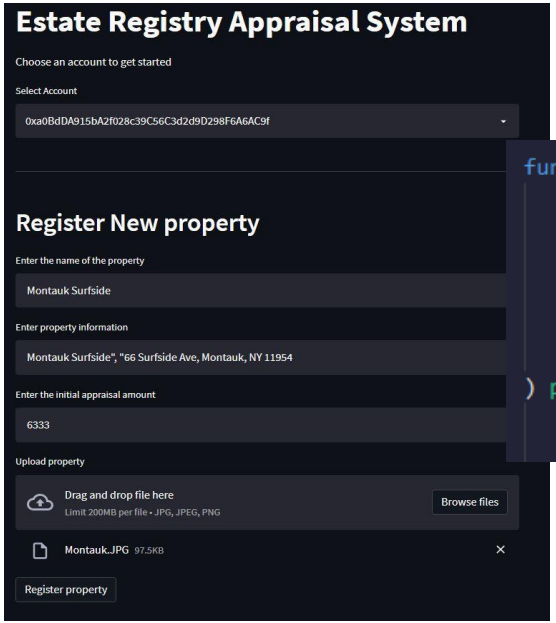
- Constructor Function Initialized for Public Token
- Property Struct Created to Define Data Types for Property Tokenization

```
4
5  contract EstateRegistry is ERC721Full {
6      constructor() public ERC721Full("EstateRegistry Token", "ACS") {}
7
8      struct Property {
9          string name;
10         string Location;
11         string PropertyHistory;
12         uint256 appraisalValue;
13     }
14
```

Mapping Property Portfolio & Appraisal Event Creation

```
14
15      mapping(uint256 => Property) public Portfolio;
16
17      event Appraisal(uint256 tokenId, uint256 appraisalValue, string reportURI);
```

Contract Function and Token Minting




```
function registerProperty(
    address owner,
    string memory name,
    string memory PropertyHistory,
    string memory Location,
    uint256 initialAppraisalValue,
    string memory tokenURI
) public returns (uint256) {
    uint256 tokenId = totalSupply();
```

Property Registration defined:

Owner

Property Name

Location

Appraisal Value

Token URI

Token Minting & ID Defined

- Token minting and token identification defined including token URI
- Property tokenization defined

```
) public returns (uint256) {
    uint256 tokenId = totalSupply();

    _mint(owner, tokenId);
    _setTokenURI(tokenId, tokenURI);

    Portfolio[tokenId] = Property(name, PropertyHistory, Location, initialAppraisalValue);

    return tokenId;
}
```

Property Appraisal Function Defined

```
function newAppraisal(
    uint256 tokenId,
    uint256 newAppraisalValue,
    string memory reportURI
) public returns (uint256) {
    Portfolio[tokenId].appraisalValue = newAppraisalValue;

    emit Appraisal(tokenId, newAppraisalValue, reportURI);

    return Portfolio[tokenId].appraisalValue;
}
```

Frontend:
Python Development

Property App Script: Functions & UI (Streamlit)

Import Dependencies & Libraries

```
##############################################################################
# Imports
import streamlit as st
from dataclasses import dataclass
from typing import Any, List
from web3 import Web3
w3 = Web3(Web3.HTTPProvider('HTTP://127.0.0.1:7545'))
##############################################################################
```

Import Wallet Functions

```
# From `crypto_wallet.py import the functions generate_account, get_balance,
#  and send_transaction

from crypto_wallet import generate_account, get_balance, send_transaction
```

Investment Properties Database Defined

```
################################################################################
# properties Information

# Database of properties including their name, digital address, detail and  cost per Ether.
# A single Ether is currently valued at $3000
properties_database = {
    "Montauk Surfside": ["Montauk Surfside", "0xaC8eB8B2ed5C4a0fC41a84Ee4950F417f67029F0", "66 Surfside Av
    "The Baldhead House": ["The Baldhead House", "0x2422858F9C4480c2724A309D58Ffd7Ac8bF65396", "218 Statio
    "Cliff House": ["Cliff House", "0x8fD00f170FDf3772C5ebdCD90bF257316c69BA45", "62 Sols Cliff Rd, Bar Ha
    "The Delaware Farm": ["The Delaware Farm", "0x8fD00f170FDf3772C5ebdCD90bF257316c69BA45", "23556 Sloan

}

# A list of the properties
properties = ["Montauk Surfside", "The Baldhead House", "Cliff House", "The Delaware Farm"]
```

Property Listings

Available Investments

Name: Montauk Surfside

Ethereum Address: 0xaC8eB8B2ed5C4a0fC41a84Ee4950F417f67029F0

Property Address: 66 Surfside Ave, Montauk, NY 11954

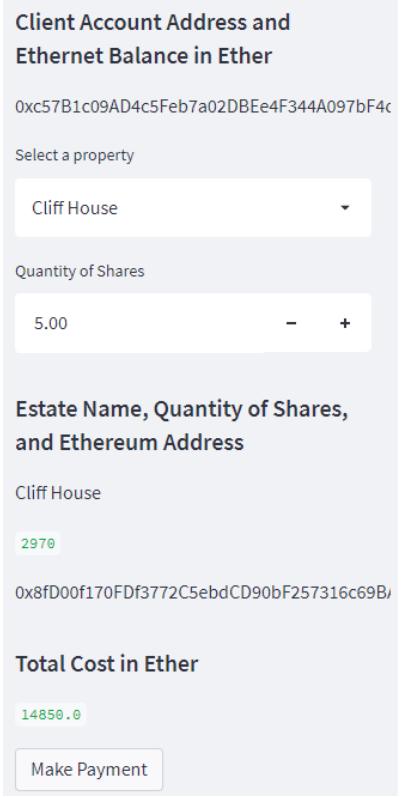
Ethereum Price : `6333` eth

get_properties Function Calls Database Values

```
40
41  def get_properties():
42      """Display the database of properties information."""
43      db_list = list(properties_database.values())
44
45      for number in range(len(properties)):
46          st.image(db_list[number][4], width=400)
47          st.write("Name: ", db_list[number][0])
48          st.write("Ethereum  Address: ", db_list[number][1])
49          st.write("Property Address: ", db_list[number][2])
50          st.write("Ethereum Price : ", db_list[number][3], "eth")
51          st.text(" \n")
52
```

Properties Defined & Written to Streamlit Sidebar

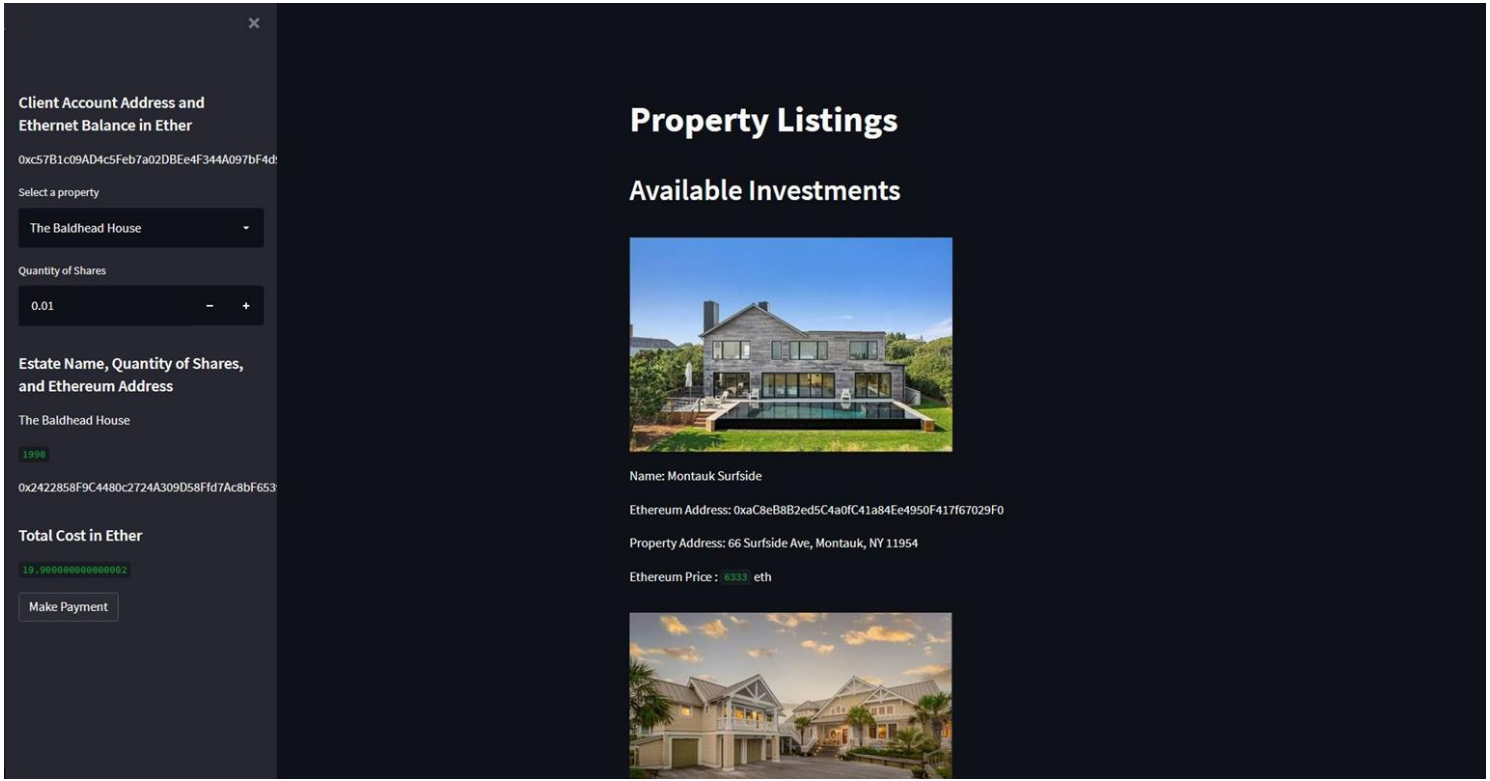
```
# Create a select box to chose a property
properties = st.sidebar.selectbox('Select a property', properties)

# Create a input field to record the number of Shares the properties worked
Shares = st.sidebar.number_input("Quantity  of Shares")

st.sidebar.markdown("## Estate Name, Quantity of Shares, and Ethereum Address")

# Identify the property
properties = properties_database[properties][0]

# Write the property   name to the sidebar
st.sidebar.write(properties)

# Identify the property  fractional share
fractional_share = properties_database[properties][3]
```



TX Hashing & Payment

```
if st.sidebar.button("Make Payment"):


    # Call the `send_transaction` function and pass it 3 parameters:
    # Your `account`, the `properties_address`, and the `Cost` as parameters
    # Save the returned transaction hash as a variable named `transaction_hash`
    transaction_hash = send_transaction(w3,account, properties_address, Cost)

    # Markdown for the transaction hash
    st.sidebar.markdown("#### Validated Transaction Hash")

    # Write the returned transaction hash to the screen
    st.sidebar.write(transaction_hash)

    # Celebrate your successful payment
    st.balloons()

# The function that starts the Streamlit application
# Writes properties propertiess to the Streamlit page
get_properties()
```

Final Product: Property Listing



Video Demo

Share Purchase Confirmation on Testnet (Truffle via Ganache)

App Script: Property Image Functions w/ Pinata

Dependency & Library Imports

```
import os
import json
from web3 import Web3
from pathlib import Path
from dotenv import load_dotenv
import streamlit as st

from pinata import pin_file_to_ipfs, pin_json_to_ipfs, convert_data_to_json

load_dotenv()

# Define and connect a new Web3 provider
w3 = Web3(Web3.HTTPProvider(os.getenv("WEB3_PROVIDER_URI")))
```

Smart Contract Load Function Defined

```python
@st.cache(allow_output_mutation=True)
def load_contract():

    # Load the contract ABI
    with open(Path('./EstateRegistry_abi.json')) as f:
        EstateRegistry_abi = json.load(f)

    # Set the contract address (this is the address of the deployed contract)
    contract_address = os.getenv("SMART_CONTRACT_ADDRESS")

    # Get the contract
    contract = w3.eth.contract(
        address=contract_address,
        abi=EstateRegistry_abi
    )

    return contract

# Load the contract
contract = load_contract()
```

Pin Property Function to Hash & Convert JSON Data to IPFS

```python
def pin_property(property_name, property_file):
    # Pin the file to IPFS with Pinata
    ipfs_file_hash = pin_file_to_ipfs(property_file.getvalue())

    # Build a token metadata file for the property
    token_json = {
        "name": property_name,
        "image": ipfs_file_hash
    }
    json_data = convert_data_to_json(token_json)

    # Pin the json to IPFS with Pinata
    json_ipfs_hash = pin_json_to_ipfs(json_data)

    return json_ipfs_hash
```

Used Pinata for Property Registration

```
if st.button("Register property"):
    # Use the `pin_property` helper function to pin the file to IPFS
    property_ipfs_hash = pin_property(property_name, file)

    property_uri = f"ipfs://{property_ipfs_hash}"

    tx_hash = contract.functions.registerProperty(
        address,
        property_name,
        property_information,
        int(initial_appraisal_value),
        property_uri
    ).transact({'from': address, 'gas': 1000000})
    receipt = w3.eth.waitForTransactionReceipt(tx_hash)
    st.write("Transaction receipt mined:")
    st.write(dict(receipt))
    st.write("You can view the pinned metadata file with the following IPFS Gateway Link")
    st.markdown(f"[property IPFS Gateway Link](https://ipfs.io/ipfs/{property_ipfs_hash})")
st.markdown("---")
```

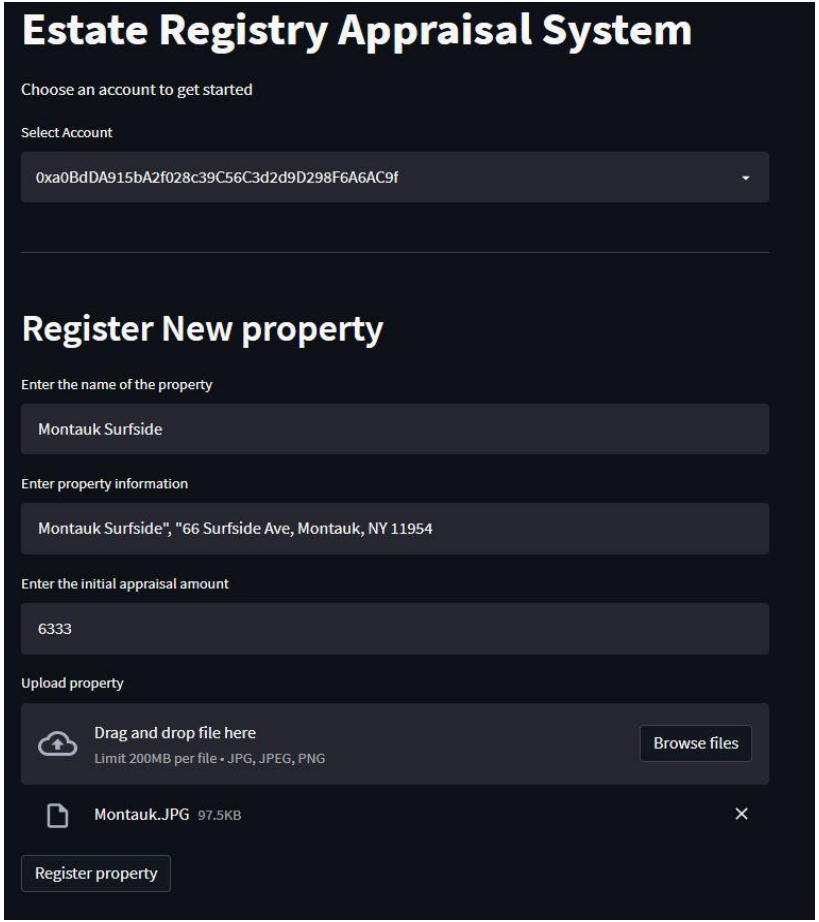
Property Appraisal

- Reporting hash defined using IPFS
- URI Report Defined
- TX Hash Defined
- Web3 used to write receipt of TXs

```
if st.button("Appraise property"):

    # Use Pinata to pin an appraisal report for the report content
    appraisal_report_ipfs_hash =  pin_appraisal_report(appraisal_report_content)

    # Copy and save the URI to this report for later use as the smart contract's
    report_uri = f"ipfs://{appraisal_report_ipfs_hash}"

    tx_hash = contract.functions.newAppraisal(
        token_id,
        int(new_appraisal_value),
        report_uri
    ).transact({"from": w3.eth.accounts[0]})
    receipt = w3.eth.waitForTransactionReceipt(tx_hash)
    st.write(receipt)
st.markdown("---")
```

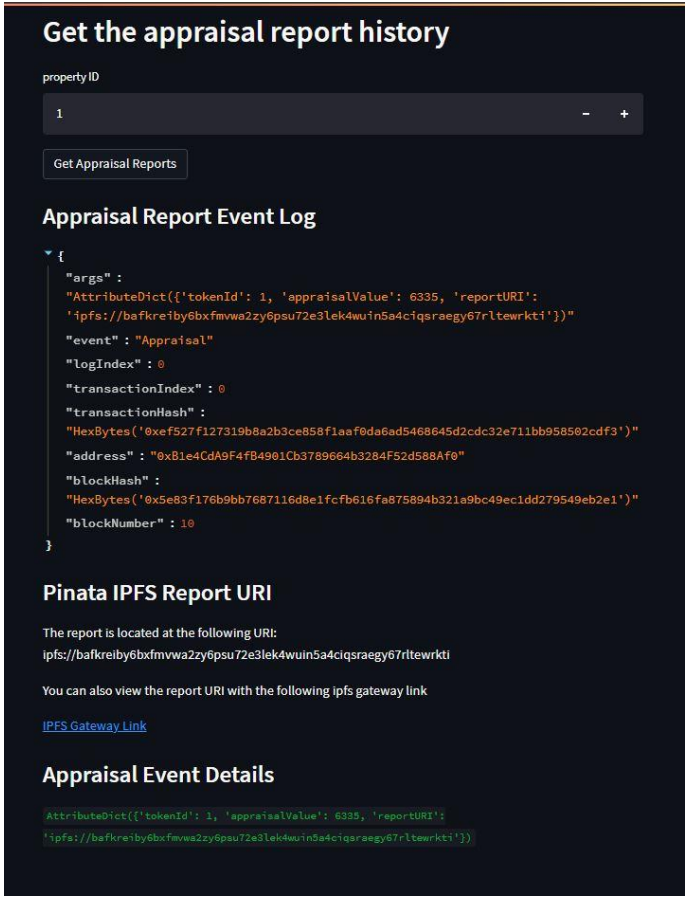
Final Product: Estate Registry System

- Allows new properties to be added to the application and shares can be fractionalized so owners can buy and sell shares.
- Owners enter an appraised value of the house and can offer portions of the property for sale.



Final Product: Appraisal History Report

Appraisal History allows investors to see how the property has been appraised in the past, and see if they want to invest in a given property



Video Demo

Thank You