Project 3: Decentralized Real Estate Marketplace

Samirah, Charles, Alex

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.

Tools Used for Development

- We used remix.ethereum to code the solidity contract

```
Home    EstateRegistry.sol
pragma solidity ^0.5.5;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/relea

contract EstateRegistry is ERC721Full {
    constructor() public ERC721Full("EstateRegistry Token", "ACS") {}
```

- We used VScode to house app function 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
)
```

Application Core Functionality

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

Backend:
Solidity Development

Solidity Contract

Compiler Versioning & OpenZepplin Import

EstateRegistry.sol

Estate Registry > Contracts > EstateRegistry.sol

```
pragma solidity ^0.5.5;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC721/

```

Estate Registry ERC 721 Contract Created

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

```
contract EstateRegistry is ERC721Full {
    constructor() public ERC721Full("EstateRegistry Token", "ACS") {}

    struct Property {
        string name;
        string Location;
        string PropertyHistory;
        uint256 appraisalValue;
    }

```

Mapping Property Portfolio & Appraisal Event Creation

```
mapping(uint256 => Property) public Portfolio;

event Appraisal(uint256 tokenId, uint256 appraisalValue, string reportURI);
```

Contract Function and Token Minting

- Property Register function defined:
  - Owner
  - Property Name
  - Location
  - Appraisal Value
  - Token URI

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

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"]
```

get_properties Function Calls Database Values

```python

def get_properties():
    """Display the database of properties information."""
    db_list = list(properties_database.values())

    for number in range(len(properties)):
        st.image(db_list[number][4], width=400)
        st.write("Name: ", db_list[number][0])
        st.write("Ethereum  Address: ", db_list[number][1])
        st.write("Property Address: ", db_list[number][2])
        st.write("Ethereum Price : ", db_list[number][3], "eth")
        st.text(" \n")

```

Streamlit Code

```
# Streamlit application headings
st.markdown("# Property Listings")
st.markdown("## Available Investments ")
st.text(" \n")

##############################################################################
# Streamlit Sidebar Code - Start

st.sidebar.markdown("## Client Account Address and Ethernet Balance in Ether")

#########################################

#  Call the `generate_account` function and save it as the variable `account`
account = generate_account(w3)

#########################################

# Write the client's Ethereum account address to the sidebar
st.sidebar.write(account.address)
```

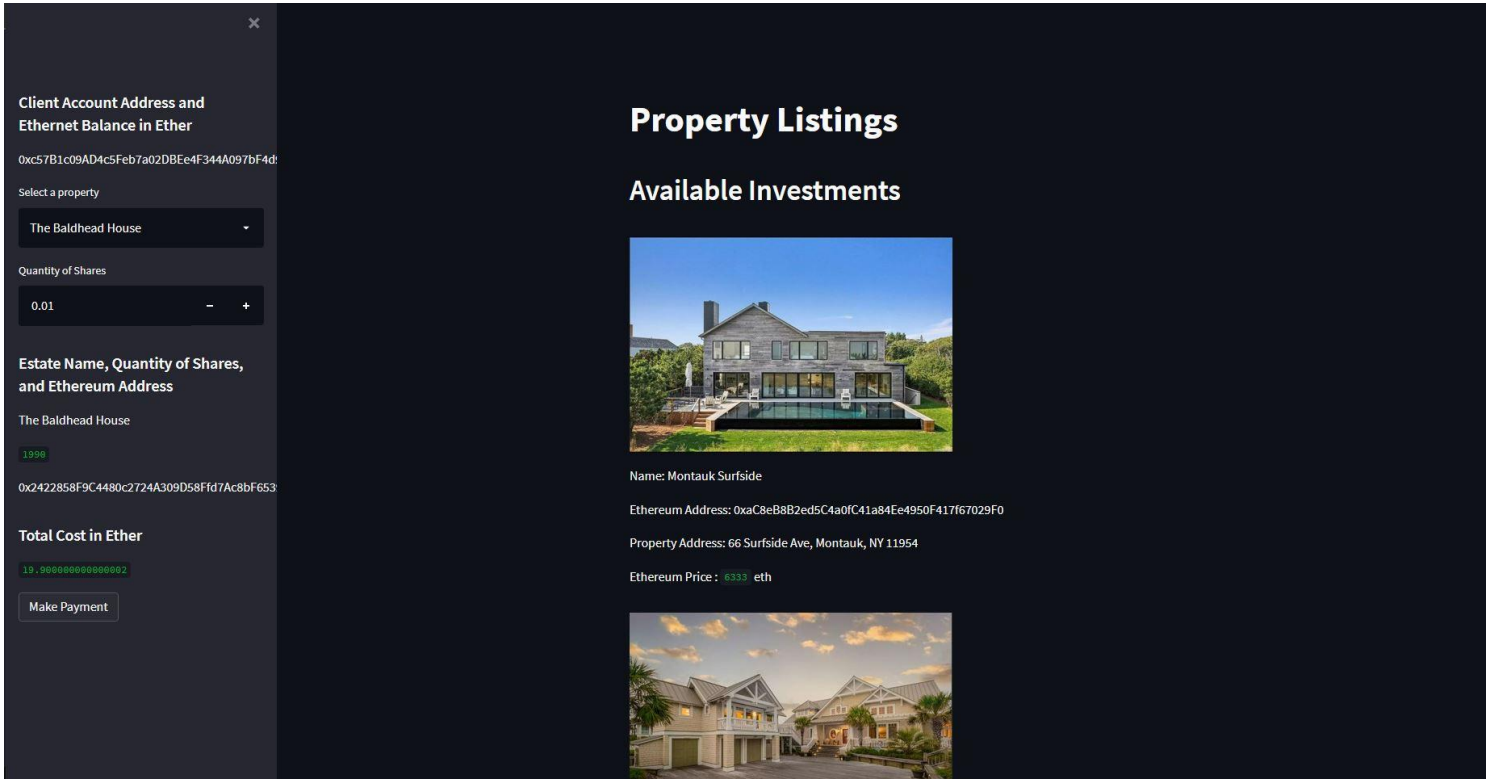
Properties Defined & Written to Streamlit Sidebar

```
# 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

```python
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)

TX HASH

0×622e23477f5a35c2f1b1d7f98ba146c71f9b54795d643a9fea3e12ff4cdba1e4

CONTRACT CALL

FROM ADDRESS	TO CONTRACT ADDRESS	GAS USED	VALUE
0×c57B1c09AD4c5Feb7a02DBEe4F344A097bF4d912	0×8fD00f170FDf3772C5ebdCD90bF257316c69BA45	21000	2970000000000000000

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("---")
```

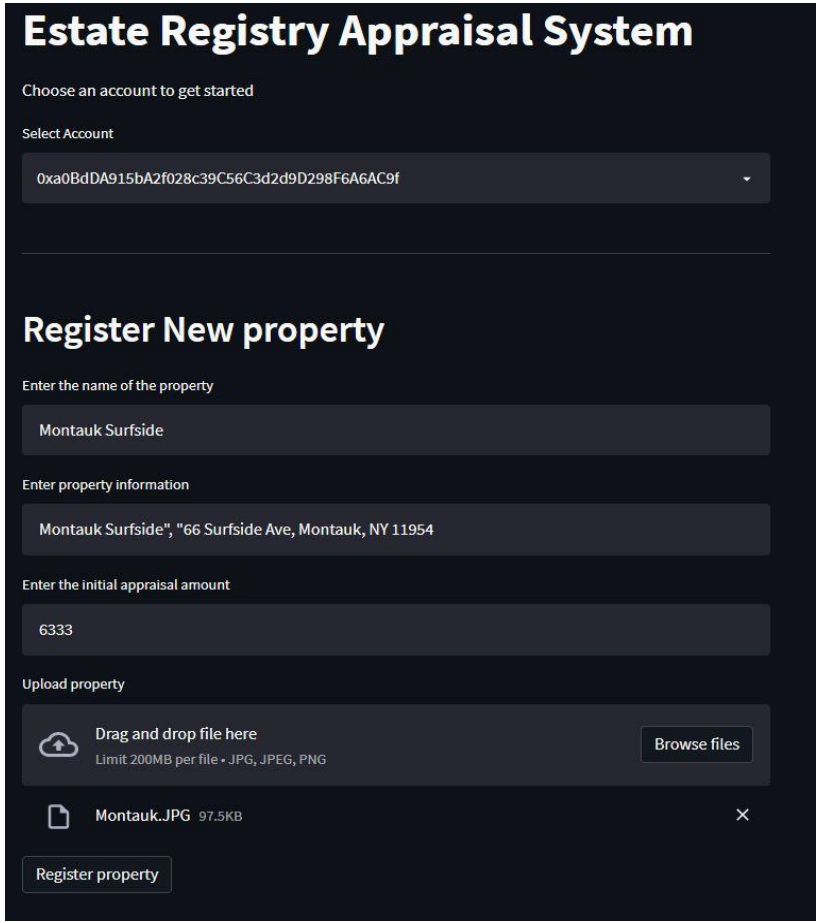
Property Appraisal

- 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

Get the appraisal report history

property ID

1 - +

Get Appraisal Reports

Appraisal Report Event Log


```
{
  "args" :
  "AttributeDict({'tokenId': 1, 'appraisalValue': 6335, 'reportURI':
  'ipfs://bafkreiby6bxfmvwa2zy6psu72e3lek4wuin5a4ciqsraegy67rltewrkti'})"
  "event" : "Appraisal"
  "logIndex" : 0
  "transactionIndex" : 0
  "transactionHash" :
  "HexBytes('0xef527f127319b8a2b3ce858f1aaf0da6ad5468645d2cdc32e711bb958502cdf3')"
  "address" : "0xB1e4CdA9F4fB4901Cb3789664b3284F52d588Af0"
  "blockHash" :
  "HexBytes('0x5e83f176b9bb7687116d8e1fcfb616fa875894b321a9bc49ec1dd279549eb2e1')"
  "blockNumber" : 10
}
```


Pinata IPFS Report URI

The report is located at the following URI:
ipfs://bafkreiby6bxfmvwa2zy6psu72e3lek4wuin5a4ciqsraegy67rltewrkti

You can also view the report URI with the following ipfs gateway link

IPFS Gateway Link

Appraisal Event Details


```
AttributeDict({'tokenId': 1, 'appraisalValue': 6335, 'reportURI':
'ipfs://bafkreiby6bxfmvwa2zy6psu72e3lek4wuin5a4ciqsraegy67rltewrkti'})
```

Video Demo

Thank You