Fintech Final Project - 3

Ethereum Smart Contracts – For Mortgage Lending

May 14th,2024

Nikhil Karol - Prerna Mehta - Sobi Iyver - Amir Nizam

Introduction

OBJECTIVE :The project aims to create a Decentralized mortgage system using Ethereum smart contracts.

SMART CONTRACT: Developed a Solidity smart contract called " Mortgage" to manage mortgage loans.

FUNCTIONALITY : The smart contract allows users to submit loan requests , approve or decline loans, make payments and withdraw remaining balances.

CONTRACT FEATURES: Tracks loan details such as loan amount, monthly payments, remaining balance , and homeowner's address

Executive Summary

MINIMUM INCOME REQUIREMENT: Implemented a requirement for a minimum monthly income for loan approval.

INTEREST CALCULATION: Added an interest calculator to calculate interest on the loan amount.

LATE FEE OPTION: Included a late fee option for borrowers to pay late fees along with their payments.

STREAMLIT INTERFACE: Developed a Streamlit web interface for users to interact with the smart contract.

DATA VISUALIZATION: Displayed contract data such as homeowner address and loan amount using the Streamlit app.

ETHEREUM INTEGRATION: Connected the Streamlit app to a local Ethereum node to interact with the smart contract. Handled Ethereum transactions for loan submissions, approvals, declines, and payments.

TESTING: Tested the functionality of the smart contract and the Streamlit interface to ensure proper operation.

Structure of Solidity Program – Introduction about Json

main.py | {} Mortgage.json

build > contracts > {} Mortgage.json > [ ] abi > {} 1


```
{
  "contractName": "Mortgage",
  "abi": [
    {
      "inputs": [],
      "stateMutability": "nonpayable",
      "type": "constructor"
    },
    {
      "anonymous": false,
      "inputs": [
        {
          "indexed": false,
          "internalType": "uint256",
          "name": "loanAmount",
          "type": "uint256"
        },
        {
          "indexed": false,
          "internalType": "uint256",
          "name": "monthlyPayment",
          "type": "uint256"
        }
      ],
      "name": "LoanApproved",
      "type": "event"
    },
```


We have used ABI standardized way to interact with Ethereum smart contracts, allowing other contracts to understand how to call the functions within the contract. Various functions as follows:

- Contract name
- Constructor
- Loan approved
- Loan amount
- Monthly payment
- Remaining Balance

Mortgage. Sol

main.py Mortgage.sol

contracts > Mortgage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8;

contract Mortgage {
    address payable public homeownerAddress;
    address payable public mortgageHolder;
    uint256 public loanAmount;
    uint256 public monthlyPayment;
    uint256 public remainingBalance;
    bool public loanApproved;
    uint256 public totalMortgages;
    uint256 public requiredMonthlyIncome; // Required monthly income for loan approval

    event LoanRequested(uint256 loanAmount, uint256 monthlyPayment, address indexed homeowner);
    event LoanApproved(uint256 loanAmount, uint256 monthlyPayment);
    event LoanDeclined(uint256 loanAmount, address indexed homeowner);
    event PaymentReceived(uint256 amount);
    event LateFeeApplied(uint256 lateFee);
```

It basically defines the smart contract named Mortgage, which facilitates the process of granting mortgages between two parties: the mortgage holder (the entity that deploys the contract) and the homeowner (the borrower). Key components as follows:

- Homeowners address
- Mortgage holder
- Loan amount and etc..

```
function submitLoan(uint256 _loanAmount, uint256 _monthlyPayment, address payable _homeowner, uint256 _interestRate) external onlyMortgageHolder
    require(getMonthlyIncome(_homeowner) >= requiredMonthlyIncome, "Insufficient monthly income for loan approval");

    // Calculate total loan amount with interest
    loanAmount = _loanAmount + (_loanAmount * _interestRate / 100);

    monthlyPayment = _monthlyPayment;
    homeownerAddress = _homeowner;
    remainingBalance = loanAmount;
    loanApproved = true; // Loan automatically approved if monthly income requirement is met
    totalMortgages++; // Increment total mortgages when a new loan is submitted
    emit LoanRequested(loanAmount, monthlyPayment, homeownerAddress);
    emit LoanApproved(loanAmount, monthlyPayment);
}
```

This code represents a function in a smart contract written in Solidity, a language used for programming smart contracts.
Key Components functions as follows:

- Get monthly income
- Loan Approval/Decline and etc..

Mortgage. Sol

Mortgage Holder Functions are as follows:

- SubmitLoan: Initiates a loan request for a homeowner. Calculates the total loan amount with interest and sets loan details. Automatically approves the loan if the homeowner meets the income requirement. Emits events for loan requested and approved.
- DeclineLoan: Rejects a pending loan request. Emits a loan declined event.
- MakePayment: Allows the mortgage holder to receive payments from the homeowner. Applies late fees if applicable. Updates remaining balance and loan status. Emits a payment received event.

Migration. Sol

main.py | Mortgage.sol | *Migrations.sol*

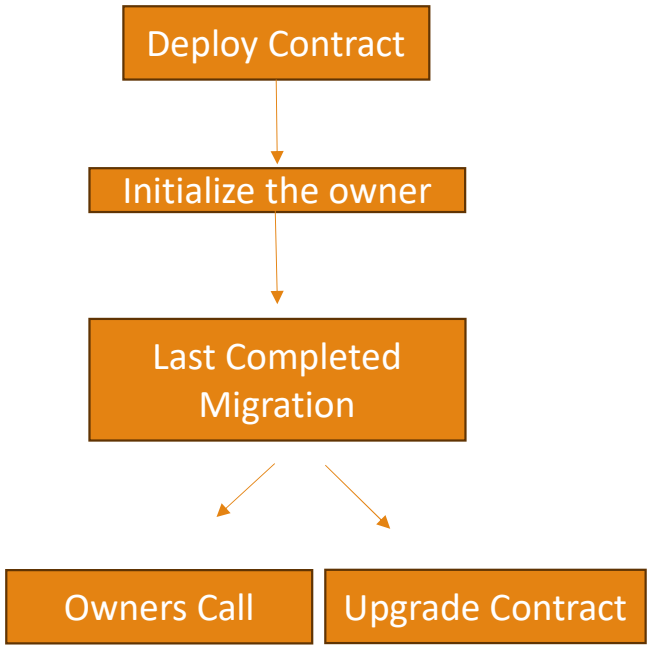
migrations > Migrations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8;

contract Migrations {
  address public owner;
  uint public last_completed_migration;

  modifier restricted() {
    if (msg.sender == owner) _;
  }

  constructor() {
    owner = msg.sender;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }

  function upgrade(address new_address) public restricted {
    Migrations upgraded = Migrations(new_address);
    upgraded.setCompleted(last_completed_migration);
  }
}
```

The code accomplish the following:

- Track of an owner address and index the last completed migration
- Restricts other to access
- On deployment sender become the owner.
- Allows owner to update the index
- Enables owner to upgrade the contact to new address and transferring the last completed migration index to new contract

Migration. Sol



Overall, this contract provides functionality for managing migrations and upgrading the contract to a new version while ensuring that only the owner can perform these actions. It's commonly used in the context of smart contract development, especially when managing the deployment and upgrading of contracts.

Main.py – 1 (Input and special functions)

main.py

```python
import streamlit as st
from web3 import Web3

# Define ABI and contract address
abi = [
    {
        "inputs": [],
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "inputs": [],
        "name": "homeowner",
        "outputs": [
            {
                "internalType": "address payable",
                "name": "",
                "type": "address"
            }
        ],
        "stateMutability": "view",
        "type": "function"
    },
    {
        "inputs": [],
        "name": "loanAmount",
        "outputs": [
            {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
```

- Streamlit imported as st
- Web3 from the web3
- We have defined the ABI to interact with smart contract, which includes the details of the functions names as input represent a special functions like homeowner, loan amount, monthly payment , interest rate, make payment, approve & decline the loan.
- These inputs represents the submit loan functions of the contract and takes four input parameters example (a) loan amount (b) monthly payment (c) interest rate
- Overall this code defines, the ABI for interacting with smart contract likely to handle the loan, this includes functions for the submitting the loan

Main.py – 2 (Contract Address)

```
contract_address = "0x0A8C94B9ACa641790174E42F6583A8Ba53e25744"  # Updated with correct contract address from Ganache

# Minimum monthly income required for loan approval has been set
MINIMUM_MONTHLY_INCOME = 5000

# Streamlit app title
st.title("Mortgage Lending - Smart Contracts")

# Connect to a local Ethereum node
node_url = "HTTP://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(node_url))

# Check if connected to the Ethereum node
if web3.is_connected():
    st.write("Connected to Ethereum node successfully!")
    contract = web3.eth.contract(address=contract_address, abi=abi)
```

- This represents the Ethereum address of a deployed smart contract.
- We have set the minimum monthly income to qualify for the loan.
- Set the title for the streamlit application as Mortgage Lending – Smart Contracts.
- This URL pointing to the local node

Main.py – 3 (web3 instance connected to an ETH node)

```
# Check if connected to the Ethereum node
if web3.is_connected():
    st.write("Connected to Ethereum node successfully!")
    contract = web3.eth.contract(address=contract_address, abi=abi)

    try:
        # Display homeowner's address
        homeowner = contract.functions.homeowner().call()
        st.write("Homeowner Address:", homeowner)

        # Display loan amount
        loan_amount = contract.functions.loanAmount().call()
        st.write("Loan Amount:", loan_amount)
    except Exception as e:
        st.error(f"Error fetching contract data: {str(e)}")
```

This represents the web3 instance is connected to Ethereum node, retrieves data from a deployed smart particularly the homeowner and loan amount. If any error occurs the error message will prompt to address the issue.

Main.py - 4

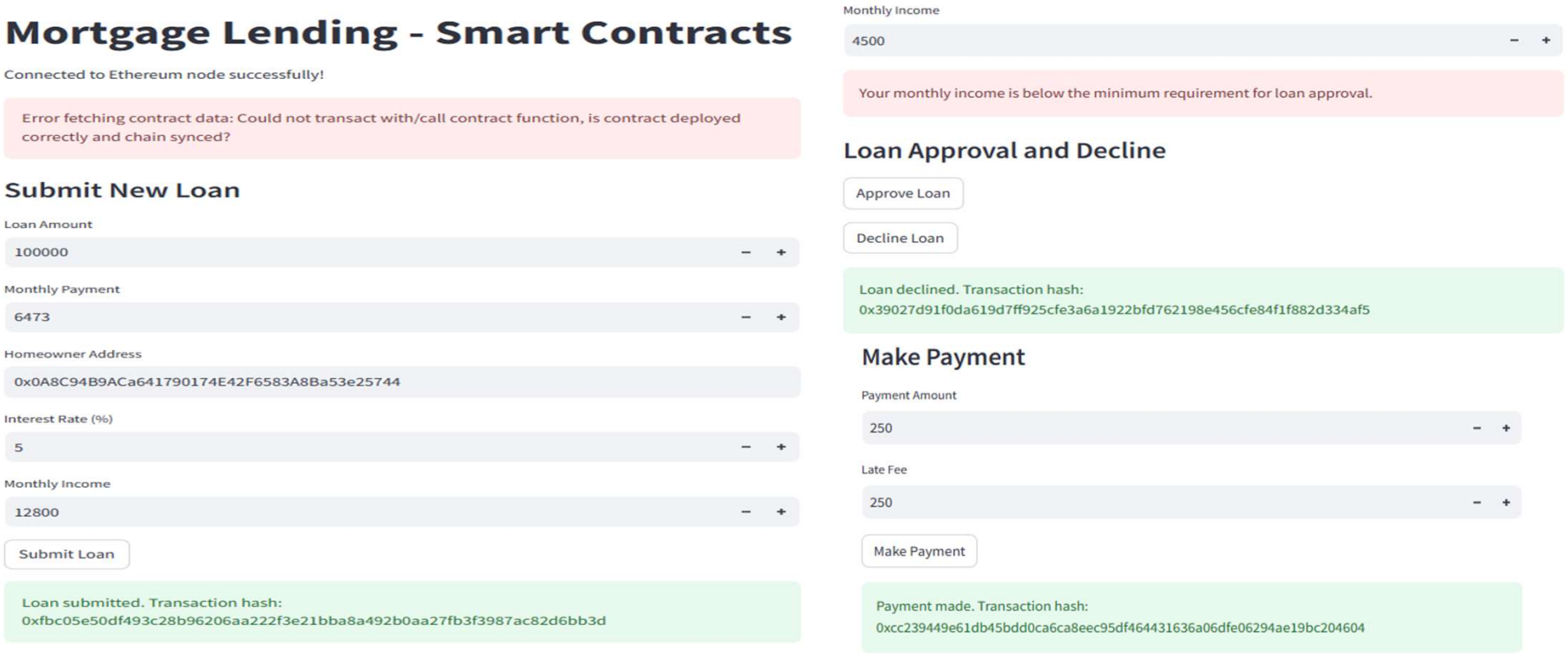
```python
# Submit new loan form
st.subheader("Submit New Loan")
loan_amount_input = st.number_input("Loan Amount", min_value=1)
monthly_payment_input = st.number_input("Monthly Payment", min_value=1)
homeowner_address_input = st.text_input("Homeowner Address")
interest_rate_input = st.number_input("Interest Rate (%)", min_value=0, max_value=100)
```

```python
# Loan approval and decline buttons
st.subheader("Loan Approval and Decline")
if st.button("Approve Loan"):
    try:
        account = web3.eth.accounts[0]
        tx_hash = contract.functions.approveLoan().transact({"from": account})
        st.success(f"Loan approved. Transaction hash: {tx_hash.hex()}")
    except Exception as e:
        st.error(f"Error approving loan: {str(e)}")
```

```python
    # Make payment
    st.subheader("Make Payment")
    payment_amount_input = st.number_input("Payment Amount", min_value=1)
    late_fee_input = st.number_input("Late Fee", min_value=0)  # Add late fee input box

    if st.button("Make Payment"):
        try:
            # Used the sender address from the connected web3 instance
            sender_address = web3.eth.accounts[0]  # Assuming the sender is the first account

            # Call the makePayment function and specify the sender address, payment amount, and late fee
            tx_hash = contract.functions.makePayment().transact(
                {"from": sender_address, "value": payment_amount_input + late_fee_input})

            st.success(f"Payment made. Transaction hash: {tx_hash.hex()}")
        except Exception as e:
            st.error(f"Error making payment: {str(e)}")
else:
    st.error("Failed to connect to Ethereum node. Please check your connection.")
```

This represents the code segment users with a form to submit a new loan application, verifies if their monthly income meets the minimum requirement of loan approval, and allows users to know the loan status whether it is approved or declined. Additionally, if the loan approved, this functions will allow the user to make payments towards their loans.

Mortgage Lending – Local Host (main · Streamlit)



Potential Improvements or Additional Features:

- Implementing more robust access control mechanisms.
- Adding events or functions for tracking loan history and payments.
- Introducing interest calculations for more realistic mortgage scenarios.
- Implementing mechanisms for handling missed payments or late fees.
- Integrating with external sources for credit checks or loan approval processes.
- Adding additional security features, optimizing gas usage, focusing on user experience.

Conclusion

The main.py serves as the backbone of a user-friendly interface for managing the mortgage lending activities on the blockchain. By streamlining the process of submitting loan applications, facilitating efficient loan approval and declines and enabling seamless payment management, the application empowers users to navigate the complexities of mortgage transactions and allow users about the status of their actions.


THANK YOU