-: Property Transactions :-

Team memebrs:

1:Gurajapu Esther Rani

2:Shaik Sheefa Anjum

3:Reddymalla Pravallika

4:Patibandla Bhargavi

Block Chain : Solidity code

- Program for Property Transactions:

Smart Contracts:

1:PropertyTransaction

2:PropertyVerification

3:PropertySale

Property Transaction

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

pragma solidity ^0.8.0;

contract Property {
    enum UserRole { PropertyOwner, PropertyVerifier, Buyer }

    struct PropertyForSale {
        uint256 propertyId;
        address payable seller;
        uint256 price;
        bool isVerified;
    }

    struct User {
        address payable addr;
        UserRole role;
    }

    mapping(uint256 => PropertyForSale) public propertiesForSale;
    mapping(address => User) public users;
    uint256 public propertyIdCounter;
```

continue

```
 event PropertyForSaleAdded(uint256 indexed propertyId, address indexed seller, uint256 price);
    event PropertySold(uint256 indexed propertyId, address indexed buyer, uint256 price);

    function addPropertyForSale(uint256 _propertyId, uint256 _price, bool _isVerified) public {
        require(users[msg.sender].role == UserRole.PropertyOwner, "Only property owner can add property for sale.");

        propertiesForSale[_propertyId] = PropertyForSale({
            propertyId: _propertyId,
            seller: payable(msg.sender),
            price: _price,
            isVerified: _isVerified
        });

        emit PropertyForSaleAdded(_propertyId, msg.sender, _price);
    }

    function buyProperty(uint256 _propertyId) public payable {
        PropertyForSale storage property = propertiesForSale[_propertyId];
        require(property.propertyId != 0, "Property does not exist.");
        require(msg.value >= property.price, "Not enough ether sent.");
        require(users[msg.sender].role == UserRole.Buyer, "Only buyer can buy property.");

        property.seller.transfer(msg.value);
        property.propertyId = 0;

        emit PropertySold(_propertyId, msg.sender, msg.value);
    }
}
```

Property verification

- entities

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract propertyVerification{
    uint PropertyID;
    string CurrentOwner;
    string PreviousOwner;
    uint   Price;
    string GovtAuthorizations;
    mapping(uint => propertyVerification) public verify;
    function verified(uint _PropertyID,string memory _CurrentOwner,string memory _NewOwner,string memory _lawyer,uint _cost) public {
        PropertyID = _PropertyID;
        CurrentOwner = _CurrentOwner;
        PreviousOwner = _NewOwner;
        GovtAuthorizations= _lawyer;
        Price = _cost;
    }
}
```

Main code

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

 pragma solidity ^0.8.10;

contract PropertyVerification {
    struct Property {
        string place;
        string description;
        uint256 price;
    }

    struct Transaction {
        uint256 id;
        Property property;
        address buyer;
        address seller;
        bool isVerified;
    }

    uint256 public transactionId;
    Transaction[] public transactions;
```

continue

```
event TransactionAdded(uint256 id, Property property, address buyer, address seller);
    event TransactionVerified(uint256 id,bool isVerified);

    function addTransaction(Property memory _property, address _seller) public {
        transactionId++;
        transactions.push(Transaction(transactionId, _property, msg.sender, _seller, false));
        emit TransactionAdded(transactionId, _property, msg.sender, _seller);
    }

    function verifyTransaction(uint256 _id) public {
        for (uint256 i = 0; i < transactions.length; i++) {
            if (transactions[i].id == _id) {
                transactions[i].isVerified = true;
                emit TransactionVerified(_id, true);
                break;
            }
        }
    }
}
```

Property Sale

- Entities// SPDX-License-Identifier: MIT

```
pragma solidity >=0.8.12 <0.9.0;
contract PropertySale{
    string SellerName;
    string SellerParentsname;
    string SellerGender;
    uint SellerContractno;
    uint SellerAadharno;
    address SellerPanNumber;
    mapping(uint => PropertySale) public Seller;
    function saled(string memory _Name,string memory _Parentsname,string memory _Gender,uint _Contractno,uint _Aadharno,address _PanNumber)
public {
        SellerName = _Name;
        SellerParentsname = _Parentsname;
        SellerGender = _Gender;
        SellerContractno = _Contractno;
        SellerAadharno = _Aadharno;
        SellerPanNumber = _PanNumber;   }
    string BuyerName;
    string BuyerParentsname;
    string BuyerGender;
    uint BuyerContractno;
    uint BuyerAadharno;
    uint BuyerPanNumber;
    mapping(uint => PropertySale) public Buyer;
    function saled(string memory _Name,string memory _Parentsname,string memory _Gender,uint  _Contractno,uint _Aadharno,uint _PanNumber)
public {
```

```
BuyerName = _Name;
          BuyerParentsname = _Parentsname;
          BuyerGender = _Gender;
          BuyerContractno = _Contractno;
          BuyerAadharno = _Aadharno;
          BuyerPanNumber = _PanNumber;
      }}
```

- Main code
- //< SPDX-License-Identifier: MIT
-

  pragma solidity >=0.8.12 <0.9.0;

```
//< SPDX-License-Identifier: MIT

pragma solidity >=0.8.12 <0.9.0;

contract PropertySale {
    address payable public buyer;
    address payable public seller;
    uint public price;
    bool public sold;

    constructor(address payable _seller, uint _price) {
        seller = _seller;
        price = _price;
        sold = false;
    }


    modifier onlyBuyer() {
        require(msg.sender == buyer);
        _; }
modifier onlyUnsold() {
        require(!sold);
        _;
    }
    modifier onlySeller() {
    require(msg.sender == seller);
    _;
}

    function buy() public payable onlyUnsold {
        require(msg.value == price);
        buyer =payable (msg.sender);
        seller.transfer(price);
        sold = true;
    }

    function changePrice(uint _newPrice) public onlySeller onlyUnsold
{
        price = _newPrice;
    }

    function withdraw() public onlySeller {
        require(sold);
        seller.transfer(address(this).balance);
    }
}
```

Aspirants

Born To Achieve

THANK YOU