```solidity
pragma solidity >=0.8.0 < 0.9.0;


contract RealEstate{



    address payable mainowner=payable(msg.sender);
    address payable user;
    uint256 _total;
    uint256 amount;
    uint256 depositTime;
    uint256 tokens;


    uint256 withdraw_amount;


    uint256 public accumulation;
    uint256 public Token_Price;
    uint256 public Rent_per_Second;
    uint256  public StartTime;
    uint public decimals;



    string public apartment_name;
    string public apartment_symbol;

    mapping(address=>uint256) public balances;
    mapping(address=>uint256) public income;
```

```
  event Deposit(address issuer,uint amount,uint time);
  event Withdraw(address _recipient,uint _amount,uint time);






constructor(string memory _propertyID, string memory _propertysymbol, uint256 total) {
      balances[msg.sender]=total;
      apartment_name=_propertyID;
      apartment_symbol=_propertysymbol;
      decimals = 18;
      _total=total;
      income[msg.sender]=0;

  }


  function offerprice (uint256 _price) public{                    //This must be in WEI
      require(msg.sender==mainowner);
      Token_Price=_price;

  }

  function buytokens() public payable{                         //This is in WEI
      tokens=msg.value/Token_Price;
      balances[msg.sender]+=tokens;
      balances[mainowner] -= tokens;
      mainowner.transfer(msg.value);
```

```
}

function rent_per_second(uint256 price) public{          //has to be set in WEI
    require(msg.sender==mainowner);
    Rent_per_Second=price;
    StartTime=block.timestamp;
}

function getTime () public view returns(uint256 time){
    return block.timestamp;
}



function _topay() public view returns(uint256 time){
    return (block.timestamp - StartTime)*Rent_per_Second;
}

function payrent() public payable{
    uint minAmount = (block.timestamp - StartTime)*Rent_per_Second;
    require (msg.value >= minAmount);

    uint moneyToReturn = msg.value - minAmount;
    if(moneyToReturn > 0){
    user=payable(msg.sender);
    user.transfer(moneyToReturn);
    }
    accumulation += msg.value;

    emit Deposit(msg.sender,msg.value,depositTime);
```

```
  }

  function withdraw(address payable _to) public{
     require(block.timestamp>depositTime);
     if (balances[_to]>0)
     withdraw_amount=balances[_to]*accumulation/_total;
     income[_to]+=withdraw_amount;
     _to.transfer(withdraw_amount);
     accumulation -= withdraw_amount;
     emit Withdraw(_to,withdraw_amount,block.timestamp);

  }

}
```