// SPDX-License-Identifier: MIT LICENSE

pragma solidity ^0.5.4;

contract shoppy {

    address payable public owner;

    constructor() public {

       owner=msg.sender;

          }

             }

}

I have created a Public address variable **owner** and made a deployer address as owner in the constructor. ( Msg sender of address ‘ person who's currently connecting with the contract’). The owner’s address is able to receive ether from the Seller. so I made it **payable**

// SPDX-License-Identifier: MIT LICENSE

pragma solidity ^0.5.4;

contract shoppy {

    address payable public owner;

    constructor() public {

       owner=msg.sender;

          }

   struct seller {

     string name;

     address addr;

     uint bankGuaraantee;

     bool bgPaid;

     }

    mapping(address=> seller) public sellers;

}

In order to Store each individual used address mappings pointing to our struct **seller.**Here**seller**struct consists of all Required details about the seller.

// SPDX-License-Identifier: MIT LICENSE

pragma solidity ^0.5.4;

contract shoppy {

    address payable public owner;

    constructor() public {

       owner=msg.sender;

          }

     uint id;

     uint purchaseId;

   struct seller {

     string name;

     address addr;

     uint bankGuaraantee;

     bool bgPaid;

     }

    mapping(address=> seller) public sellers;

function sellerSignUp(string memory \_name) public payable{

        require(!sellers[msg.sender].bgPaid, "You are Already Registered");

        require(msg.value==5 ether, "Bank Guarantee of 5ETH Required");

        owner.transfer(msg.value);

        sellers[msg.sender].name= \_name;

        sellers[msg.sender].addr= msg.sender;

        sellers[msg.sender].bankGuaraantee = msg.value;

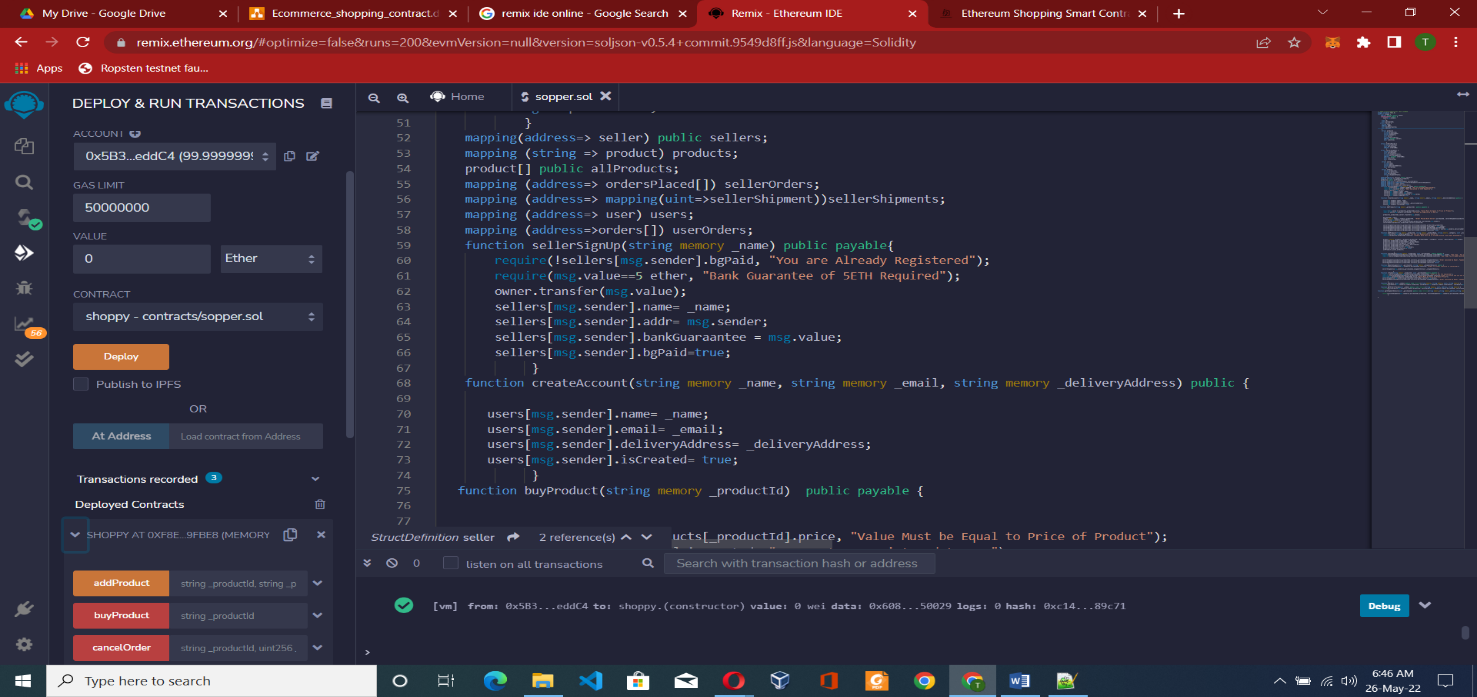
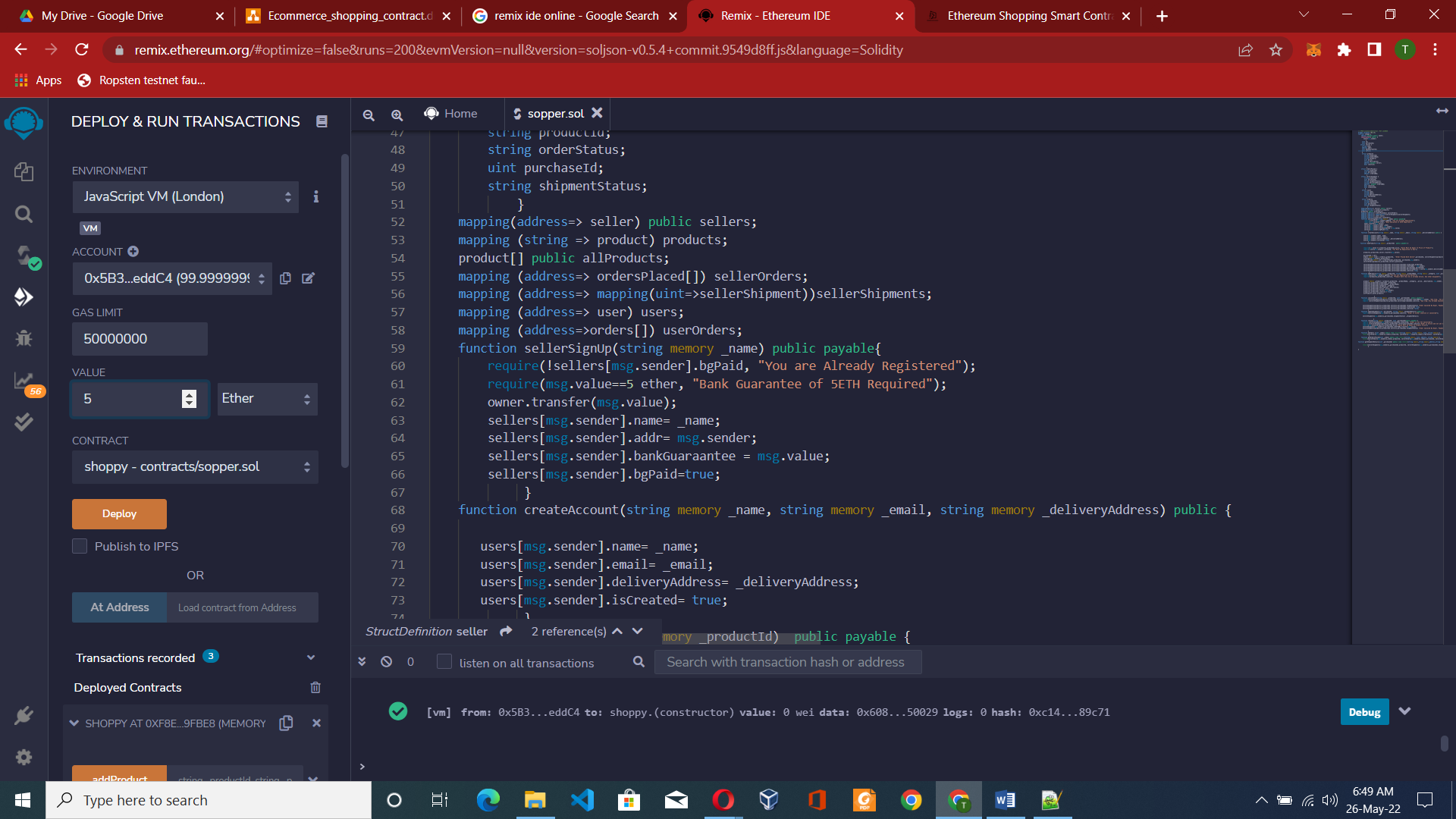
        sellers[msg.sender].bgPaid=true;

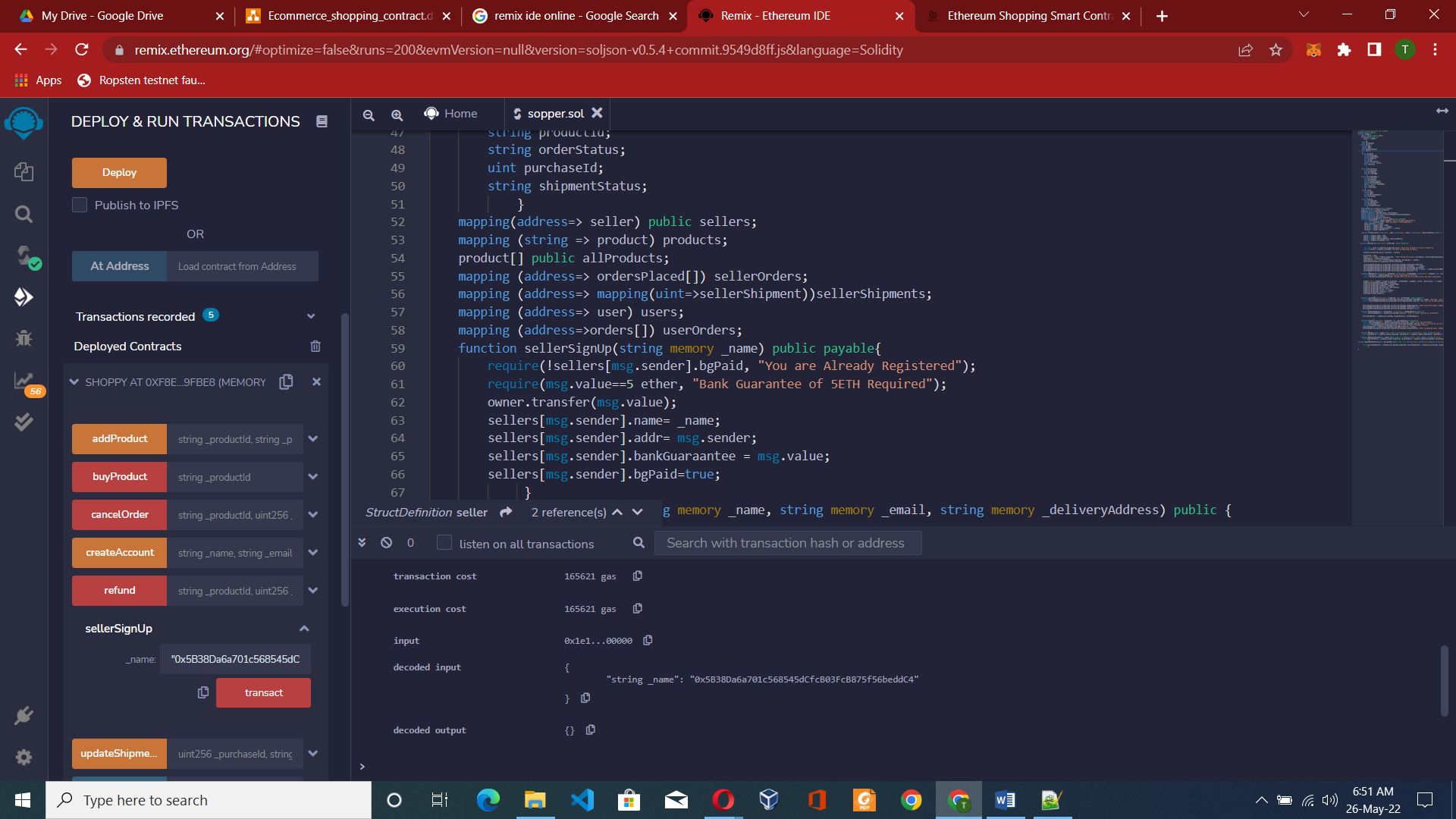
             }

}

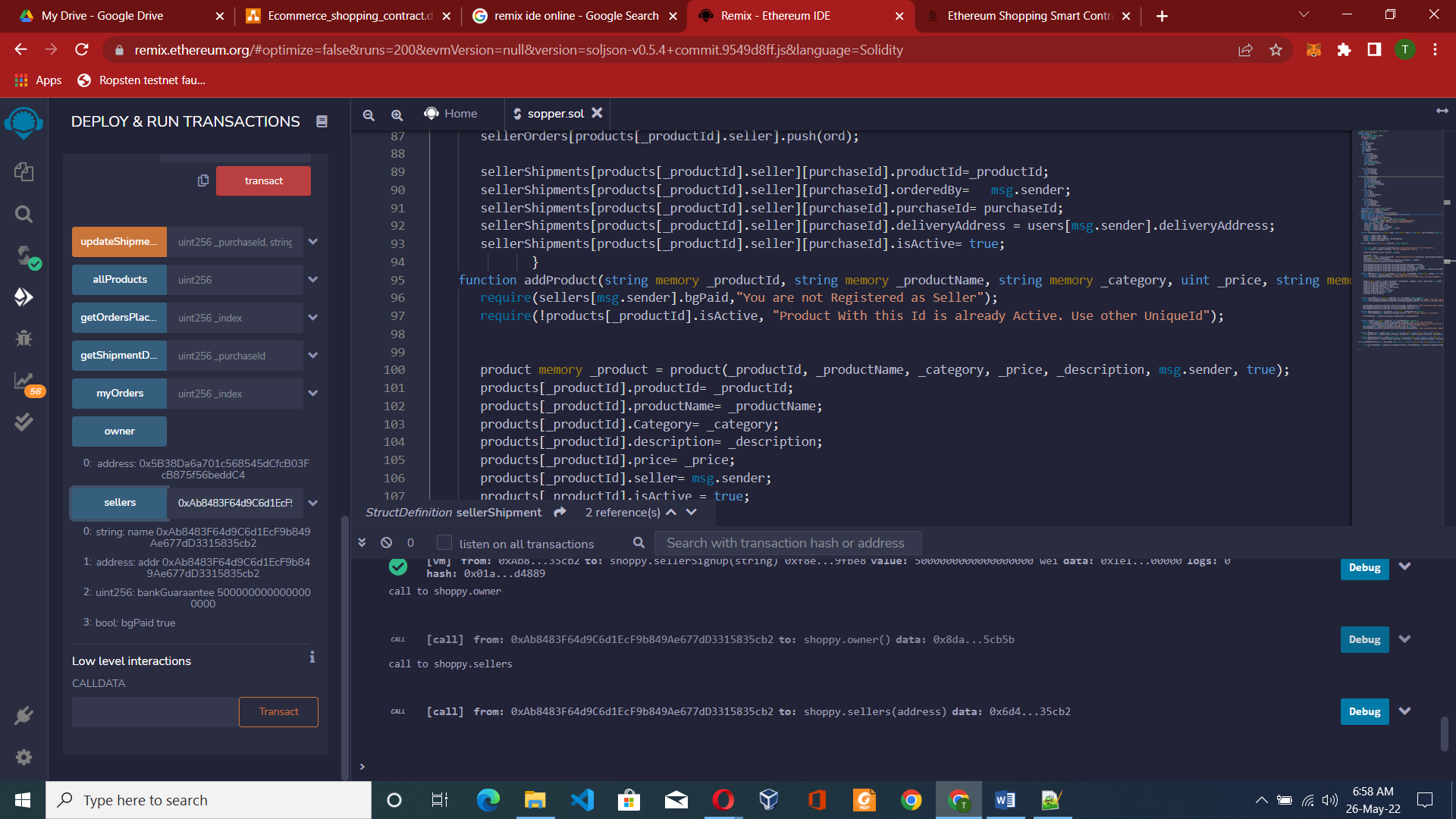
Added a function **sellerSignup**with require statements. Here is What they’ll do,

1)Checks if seller is already registered, if registered condition will be false and revert back to initial stage.

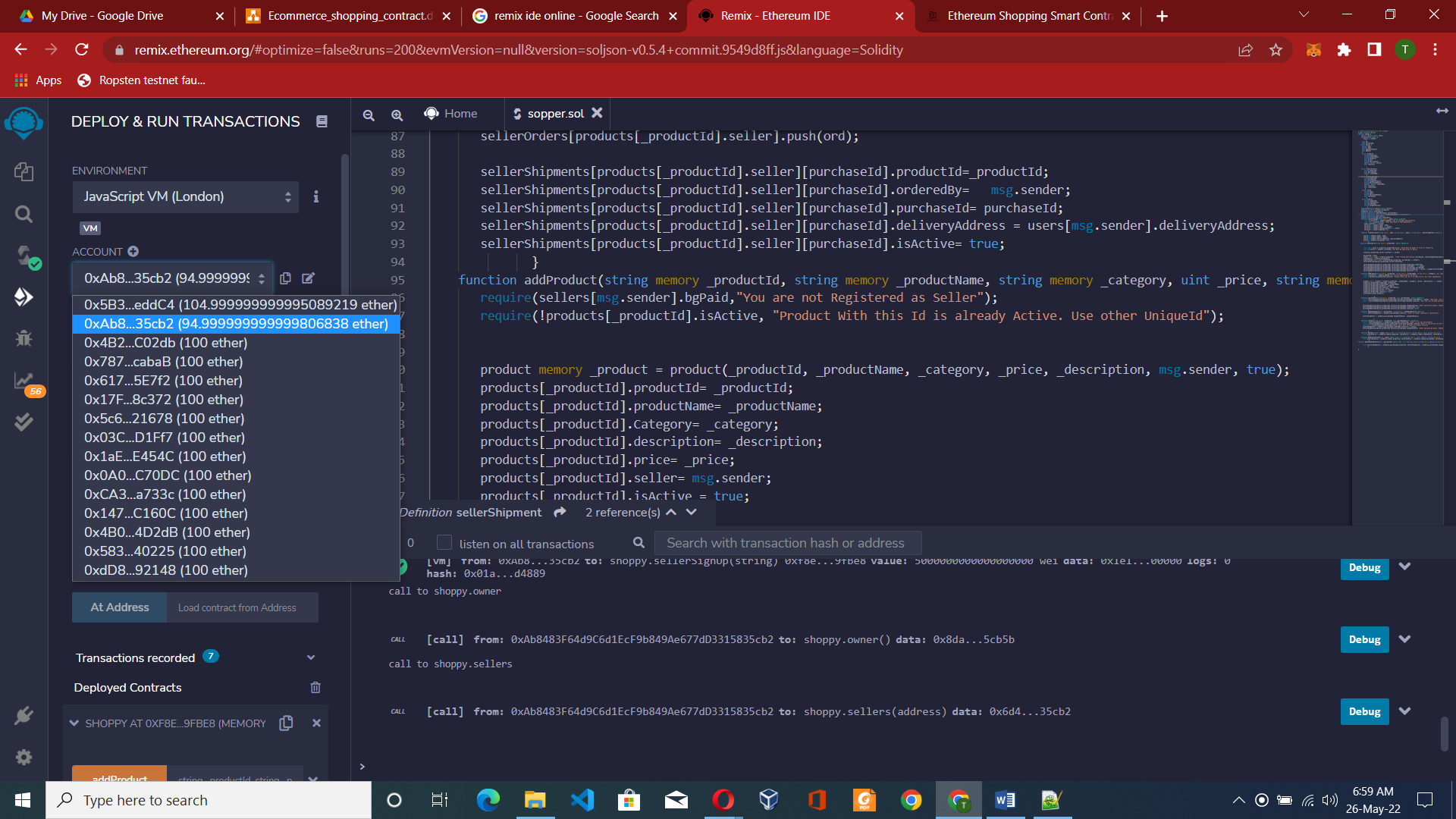
2)Check in stage msg value = 5 ether will create account of seller as register seller. After deployment. 



We mapped owner as seller and seller who paid 5 ether to himself as he is the owner also. Lets map a another seller using another address



Signed up another seller account 2 who paid 5 eth to the owner and become registered seller.



// SPDX-License-Identifier: MIT LICENSE

pragma solidity ^0.5.4;

contract shoppy {

    address payable public owner;

    constructor() public {

       owner=msg.sender;

          }

     uint id;

     uint purchaseId;

   struct seller {

     string name;

     address addr;

     uint bankGuaraantee;

     bool bgPaid;

     }

     struct product{

        string productId;

        string productName;

        string Category;

        uint price;

        string description;

        address payable seller;

        bool isActive;

           }

    mapping(address=> seller) public sellers;

    mapping (string => product) products;

    product[] public allProducts;

    function sellerSignUp(string memory \_name) public payable{

        require(!sellers[msg.sender].bgPaid, "You are Already Registered");

        require(msg.value==5 ether, "Bank Guarantee of 5ETH Required");

        owner.transfer(msg.value);

        sellers[msg.sender].name= \_name;

        sellers[msg.sender].addr= msg.sender;

        sellers[msg.sender].bankGuaraantee = msg.value;

        sellers[msg.sender].bgPaid=true;

             }

       product memory \_product = product(\_productId, \_productName, \_category, \_price, \_description, msg.sender, true);

       products[\_productId].productId= \_productId;

       products[\_productId].productName= \_productName;

       products[\_productId].Category= \_category;

       products[\_productId].description= \_description;

       products[\_productId].price= \_price;

       products[\_productId].seller= msg.sender;

       products[\_productId].isActive = true;

       allProducts.push(\_product);

                     }

    function addProduct(string memory \_productId, string memory \_productName, string memory \_category, uint \_price, string memory \_description) public {

       require(sellers[msg.sender].bgPaid,"You are not Registered as Seller");

       require(!products[\_productId].isActive, "Product With this Id is already Active. Use other UniqueId");

}

In Above Code product is created and with function add product created list of product category. It requires sellers to add product not other user . with unique id of product is in use , use another id not same id to register product.

pragma solidity ^0.5.4;

contract shoppy {

   address payable public owner;

   constructor() public {

       owner=msg.sender;

          }

   struct seller {

     string name;

     address addr;

     uint bankGuaraantee;

     bool bgPaid;

          }

     struct product{

        string productId;

        string productName;

        string Category;

        uint price;

        string description;

        address payable seller;

        bool isActive;

           }

    struct ordersPlaced {

        string productId;

        uint purchaseId;

        address orderedBy;

           }

    struct sellerShipment {

        string productId;

        uint purchaseId;

        string shipmentStatus;

        string deliveryAddress;

        address  payable orderedBy;

        bool isActive;

        bool isCanceled;

         }

 mapping(address=> seller) public sellers;

 mapping (string => product) products;

 product[] public allProducts;

 mapping (address=> ordersPlaced[]) sellerOrders;

 mapping (address=> mapping(uint=>sellerShipment))sellerShipments;

  function sellerSignUp(string memory \_name) public payable{

    require(!sellers[msg.sender].bgPaid);

        require(msg.value==5 ether);

        owner.transfer(msg.value);

        sellers[msg.sender].name= \_name;

        sellers[msg.sender].addr= msg.sender;

        sellers[msg.sender].bankGuaraantee = msg.value;

        sellers[msg.sender].bgPaid=true;

    }

 function addProduct(string memory \_productId, string memory \_productName, string memory \_category, uint \_price, string memory \_description) public {

       require(!products[\_productId].isActive);

       require(sellers[msg.sender].bgPaid);

       product memory \_product = product(\_productId, \_productName, \_category, \_price, \_description, msg.sender, true);

       products[\_productId].productId= \_productId;

       products[\_productId].productName= \_productName;

       products[\_productId].Category= \_category;

       products[\_productId].description= \_description;

       products[\_productId].price= \_price;

       products[\_productId].seller= msg.sender;

       products[\_productId].isActive = true;

       allProducts.push(\_product);

             }

function getOrdersPlaced(uint \_index) public view returns(string memory, uint, address, string memory) { return(sellerOrders[msg.sender][\_index].productId, sellerOrders[msg.sender][\_index].purchaseId, sellerOrders[msg.sender][\_index].orderedBy, sellerShipments[msg.sender][sellerOrders[msg.sender][\_index].purchaseId].shipmentStatus);

}

function getShipmentDetails(uint \_purchaseId) public view returns(string memory,string memory,address,string memory) {

return(sellerShipments[msg.sender][\_purchaseId].productId, sellerShipments[msg.sender][\_purchaseId].shipmentStatus, sellerShipments[msg.sender][\_purchaseId].orderedBy,sellerShipments[msg.sender][\_purchaseId].deliveryAddress);

    }

}

Above code mapping when order is placed with shipment details by unit id and purchased id. of owner. Until it not cancel to shipment all details are here. upto delivery order is tracked.

// SPDX-License-Identifier: MIT LICENSE

pragma solidity ^0.5.4;

contract shoppy {

    address payable public owner;

    constructor() public {

       owner=msg.sender;

          }

     uint id;

     uint purchaseId;

   struct seller {

     string name;

     address addr;

     uint bankGuaraantee;

     bool bgPaid;

     }

     struct product{

        string productId;

        string productName;

        string Category;

        uint price;

        string description;

        address payable seller;

        bool isActive;

           }

    struct ordersPlaced {

        string productId;

        uint purchaseId;

        address orderedBy;

           }

    struct sellerShipment {

        string productId;

        uint purchaseId;

        string shipmentStatus;

        string deliveryAddress;

        address  payable orderedBy;

        bool isActive;

        bool isCanceled;

         }

     struct user{

        string name;

        string email;

        string deliveryAddress;

        bool isCreated;

           }

     struct orders{

        string productId;

        string orderStatus;

        uint purchaseId;

        string shipmentStatus;

            }

    mapping(address=> seller) public sellers;

    mapping (string => product) products;

    product[] public allProducts;

    mapping (address=> ordersPlaced[]) sellerOrders;

    mapping (address=> mapping(uint=>sellerShipment))sellerShipments;

    mapping (address=> user) users;

    mapping (address=>orders[]) userOrders;

    function sellerSignUp(string memory \_name) public payable{

        require(!sellers[msg.sender].bgPaid, "You are Already Registered");

        require(msg.value==5 ether, "Bank Guarantee of 5ETH Required");

        owner.transfer(msg.value);

        sellers[msg.sender].name= \_name;

        sellers[msg.sender].addr= msg.sender;

        sellers[msg.sender].bankGuaraantee = msg.value;

        sellers[msg.sender].bgPaid=true;

             }

    function createAccount(string memory \_name, string memory \_email, string memory \_deliveryAddress) public {

       users[msg.sender].name= \_name;

       users[msg.sender].email= \_email;

       users[msg.sender].deliveryAddress= \_deliveryAddress;

       users[msg.sender].isCreated= true;

             }

   function buyProduct(string memory \_productId)  public payable {

       require(msg.value == products[\_productId].price, "Value Must be Equal to Price of Product");

       require( users[msg.sender].isCreated, "You Must Be Registered to Buy");

       products[\_productId].seller.transfer(msg.value);

       purchaseId = id++;

       orders memory order = orders(\_productId,  "Order Placed With Seller",purchaseId, sellerShipments[products[\_productId].seller][purchaseId].shipmentStatus);

       userOrders[msg.sender].push(order);

       ordersPlaced memory ord = ordersPlaced(\_productId, purchaseId,msg.sender);

       sellerOrders[products[\_productId].seller].push(ord);

       sellerShipments[products[\_productId].seller][purchaseId].productId=\_productId;

       sellerShipments[products[\_productId].seller][purchaseId].orderedBy=   msg.sender;

       sellerShipments[products[\_productId].seller][purchaseId].purchaseId= purchaseId;

       sellerShipments[products[\_productId].seller][purchaseId].deliveryAddress = users[msg.sender].deliveryAddress;

       sellerShipments[products[\_productId].seller][purchaseId].isActive= true;

              }

    function addProduct(string memory \_productId, string memory \_productName, string memory \_category, uint \_price, string memory \_description) public {

       require(sellers[msg.sender].bgPaid,"You are not Registered as Seller");

       require(!products[\_productId].isActive, "Product With this Id is already Active. Use other UniqueId");

       product memory \_product = product(\_productId, \_productName, \_category, \_price, \_description, msg.sender, true);

       products[\_productId].productId= \_productId;

       products[\_productId].productName= \_productName;

       products[\_productId].Category= \_category;

       products[\_productId].description= \_description;

       products[\_productId].price= \_price;

       products[\_productId].seller= msg.sender;

       products[\_productId].isActive = true;

       allProducts.push(\_product);

                     }

    function cancelOrder(string memory \_productId, uint \_purchaseId)  public payable {

      require(sellerShipments[products[\_productId].seller][\_purchaseId].orderedBy==msg.sender, "Aww Crap.. You are not Authorized to This Product PurchaseId");

      require (sellerShipments[products[\_productId].seller][purchaseId].isActive, "Aww crap..You Already Canceled This order");

      sellerShipments[products[\_productId].seller][\_purchaseId].shipmentStatus= "Order Canceled By Buyer, Payment will Be  Refunded";

      sellerShipments[products[\_productId].seller][\_purchaseId].isCanceled= true;

      sellerShipments[products[\_productId].seller][\_purchaseId].isActive= false;

             }

    function updateShipment(uint \_purchaseId, string memory \_shipmentDetails) public {

      require(sellerShipments[msg.sender][\_purchaseId].isActive, "Order is either inActive or cancelled");

      sellerShipments[msg.sender][\_purchaseId].shipmentStatus= \_shipmentDetails;

                    }

    function refund(string memory \_productId, uint \_purchaseId)public payable {

      require (sellerShipments[msg.sender][\_purchaseId].isCanceled, "Order is not Yet Cancelled");

      require (!sellerShipments[products[\_productId].seller][purchaseId].isActive,"Order is Active and not yet Cancelled");

      require(msg.value==products[\_productId].price,"Value Must be Equal to Product Price");

      sellerShipments[msg.sender][\_purchaseId].orderedBy.transfer(msg.value);

      sellerShipments[products[\_productId].seller][\_purchaseId].shipmentStatus= "Order Canceled By Buyer, Payment Refunded";

             }

    function myOrders (uint \_index) public view returns(string memory, string memory, uint, string memory) {

      return(userOrders[msg.sender][\_index].productId, userOrders[msg.sender][\_index].orderStatus, userOrders[msg.sender][\_index].purchaseId, sellerShipments[products[userOrders[msg.sender][\_index].productId].seller][userOrders[msg.sender][\_index].purchaseId].shipmentStatus);

              }

    function getOrdersPlaced(uint \_index) public view returns(string memory, uint, address, string memory) {

      return(sellerOrders[msg.sender][\_index].productId, sellerOrders[msg.sender][\_index].purchaseId, sellerOrders[msg.sender][\_index].orderedBy, sellerShipments[msg.sender][sellerOrders[msg.sender][\_index].purchaseId].shipmentStatus);

              }

function getShipmentDetails(uint \_purchaseId) public view returns(string memory,string memory,address,string memory) {

      return(sellerShipments[msg.sender][\_purchaseId].productId, sellerShipments[msg.sender][\_purchaseId].shipmentStatus, sellerShipments[msg.sender][\_purchaseId].orderedBy,sellerShipments[msg.sender][\_purchaseId].deliveryAddress);

             }

}

User struct is created and userorder struct is created created account of user buyer sign up and track buy related order from shipment delivery and refund et.