|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**Experiment No. 08 PBLE**

|  |  |
| --- | --- |
| Semester | B.E. Semester VII – Computer Engineering |
| Subject | Blockchain Lab (CSDL7022) |
| Subject Professor In-charge | Prof. Swapnil S. Sonawane |
| Academic Year | 2024-25 |

|  |  |
| --- | --- |
| Student Name | Deep Salunkhe |
| Roll Number | 21102A0014 |

**Title:** Tracking Supply Chain Transactions

**Code:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract SupplyChain {

    enum ItemState { Created, Paid, Shipped, Delivered }

    struct Item {

        uint id;

        string name;

        uint price;

        ItemState state;

        address payable seller;

        address payable buyer;

        string description;

    }

    // State variables

    mapping(uint => Item) public items;

    uint public itemCount;

    // Events

    event ItemCreated(uint id, string name, uint price, string description);

    event ItemPurchased(uint id, address buyer,uint price);

    event ItemShipped(uint id);

    event ItemDelivered(uint id);

    // Modifiers

    modifier onlySeller(uint \_id) {

        require(msg.sender == items[\_id].seller, "Only seller can perform this action");

        \_;

    }

    modifier onlyBuyer(uint \_id) {

        require(msg.sender == items[\_id].buyer, "Only buyer can perform this action");

        \_;

    }

    modifier itemExists(uint \_id) {

        require(\_id > 0 && \_id <= itemCount, "Item does not exist");

        \_;

    }

    // Create a new item

    function createItem(string memory \_name, uint \_price, string memory \_description) public {

        require(\_price > 0, "Price must be greater than zero");

        itemCount++;

        items[itemCount] = Item({

            id: itemCount,

            name: \_name,

            price: \_price,

            state: ItemState.Created,

            seller: payable(msg.sender),

            buyer: payable(address(0)),

            description: \_description

        });

        emit ItemCreated(itemCount, \_name, \_price, \_description);

    }

    // Get item details

    function getItem(uint \_id) public view itemExists(\_id) returns (

        uint id,

        string memory name,

        uint price,

        ItemState state,

        address seller,

        address buyer,

        string memory description

    ) {

        Item storage item = items[\_id];

        return (

            item.id,

            item.name,

            item.price,

            item.state,

            item.seller,

            item.buyer,

            item.description

        );

    }

    // Purchase an item

    function purchaseItem(uint \_id,uint \_price) public payable itemExists(\_id) {

        Item storage item = items[\_id];

        require(item.state == ItemState.Created, "Item is not available for purchase");

        require(\_price == item.price, "Incorrect payment amount");

        require(msg.sender != item.seller, "Seller cannot buy their own item");

        item.buyer = payable(msg.sender);

        item.state = ItemState.Paid;

        emit ItemPurchased(\_id, msg.sender,\_price);

    }

    // Ship an item

    function shipItem(uint \_id) public itemExists(\_id) onlySeller(\_id) {

        Item storage item = items[\_id];

        require(item.state == ItemState.Paid, "Item must be paid before shipping");

        item.state = ItemState.Shipped;

        emit ItemShipped(\_id);

    }

    // Confirm delivery

    function confirmDelivery(uint \_id) public itemExists(\_id) onlyBuyer(\_id) {

        Item storage item = items[\_id];

        require(item.state == ItemState.Shipped, "Item must be shipped before delivery confirmation");

        item.state = ItemState.Delivered;

        // Transfer payment to seller

        item.seller.transfer(item.price);

        emit ItemDelivered(\_id);

    }

    // Get all items for sale

    function getItemsForSale() public view returns (uint[] memory) {

        uint[] memory itemIds = new uint[](itemCount);

        uint numberOfItems = 0;

        for (uint i = 1; i <= itemCount; i++) {

            if (items[i].state == ItemState.Created) {

                itemIds[numberOfItems] = i;

                numberOfItems++;

            }

        }

        // Create a properly sized array

        uint[] memory forSale = new uint[](numberOfItems);

        for (uint i = 0; i < numberOfItems; i++) {

            forSale[i] = itemIds[i];

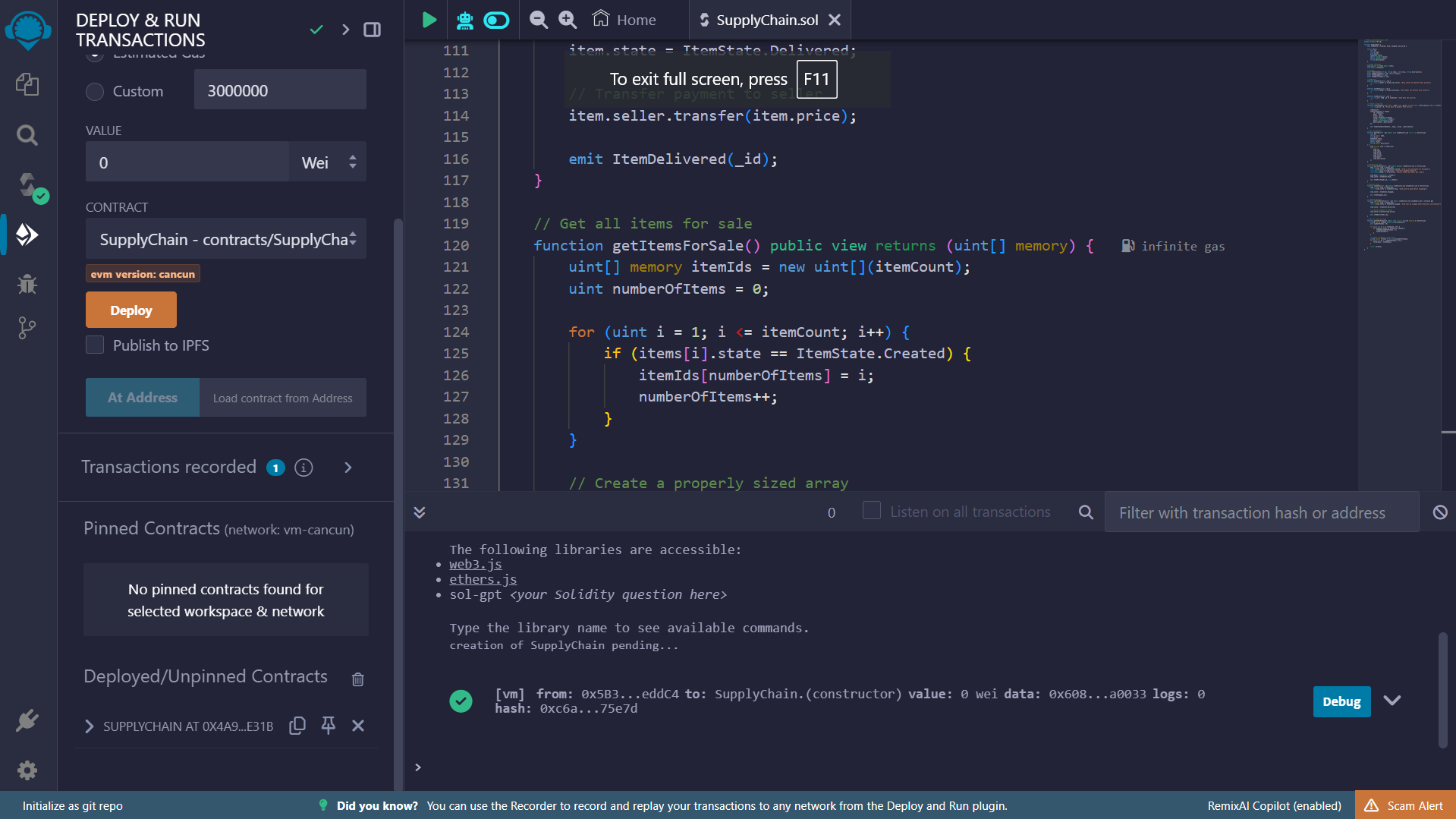
        }

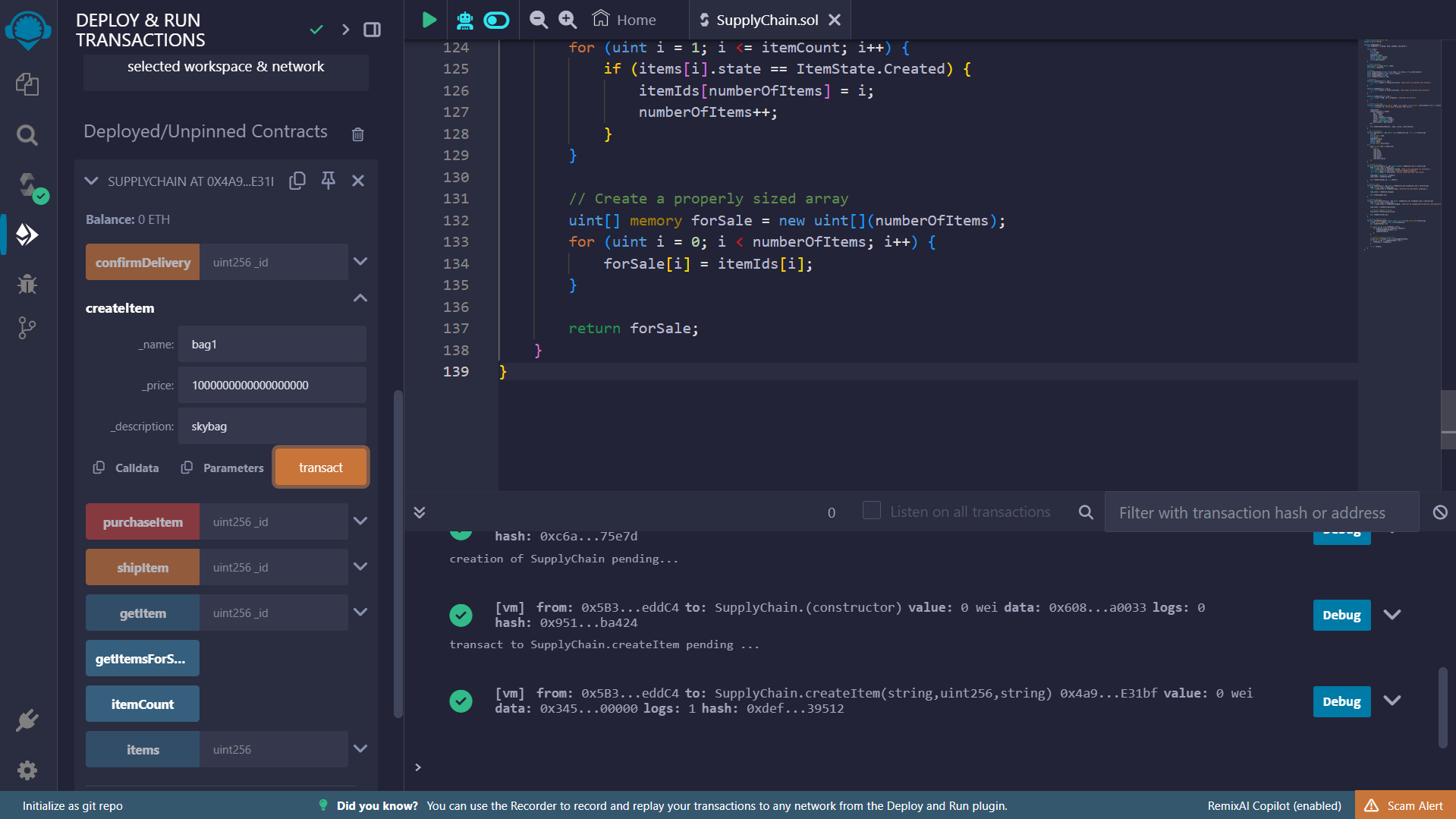
        return forSale;

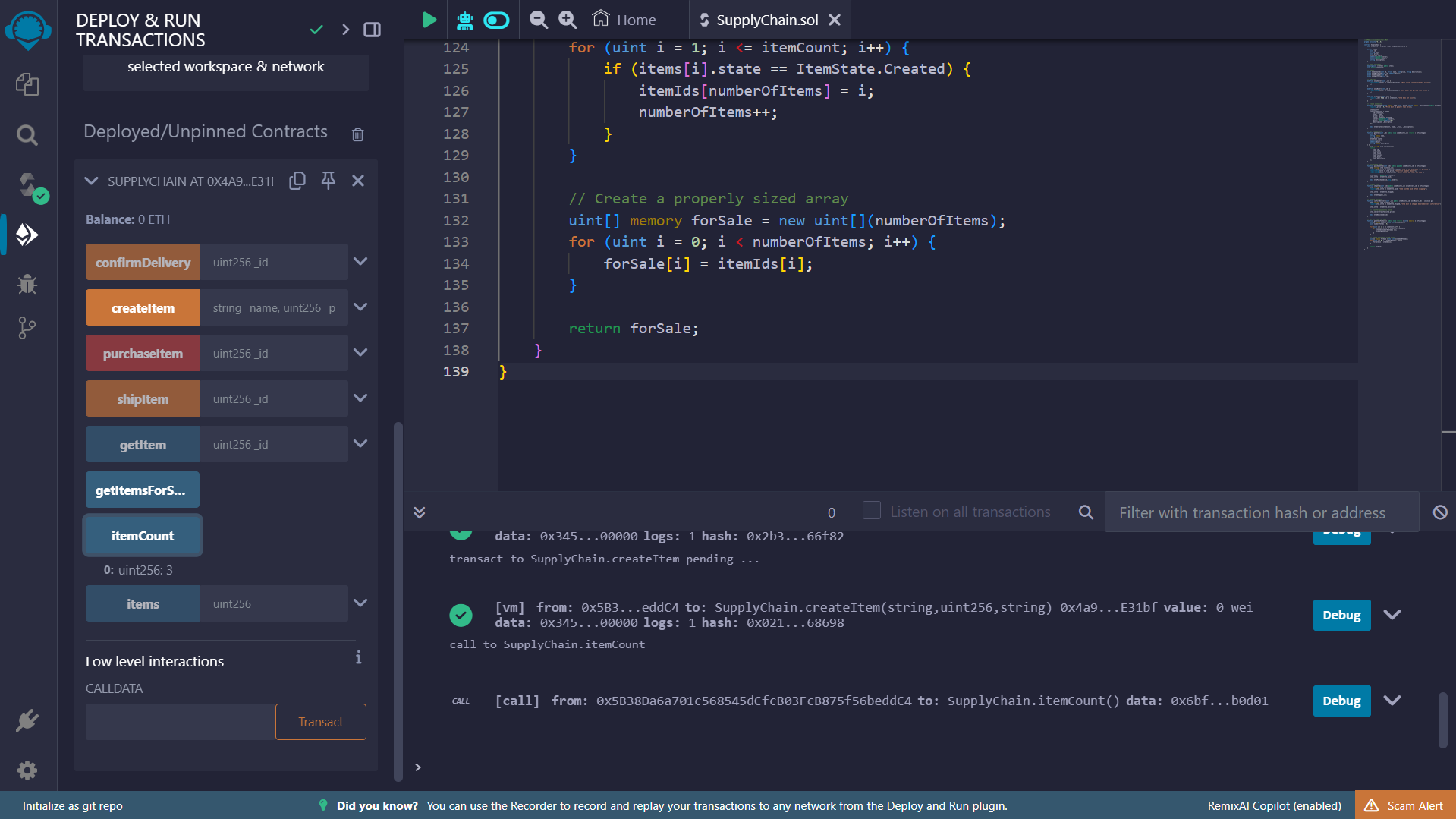
    }

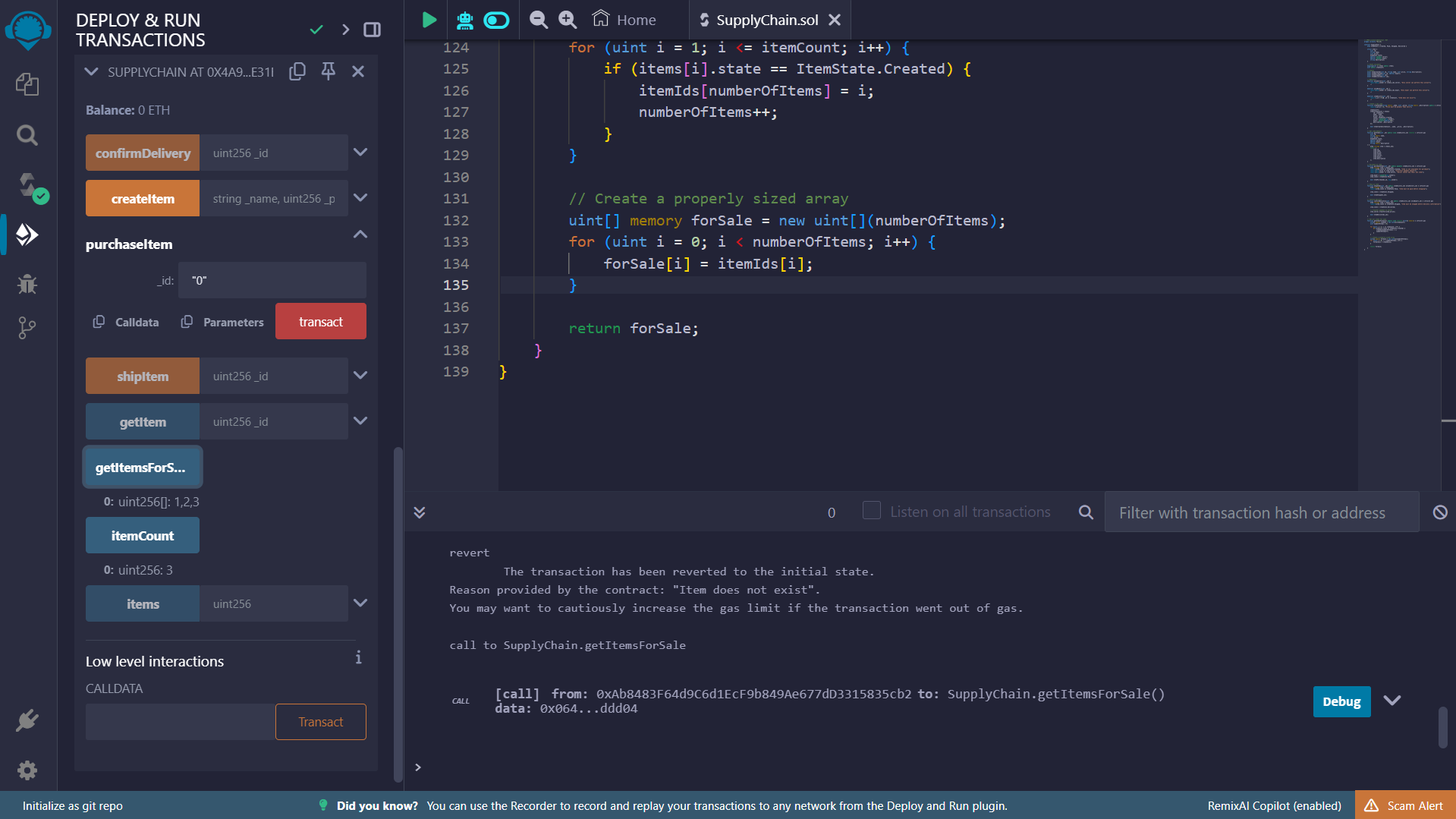
}

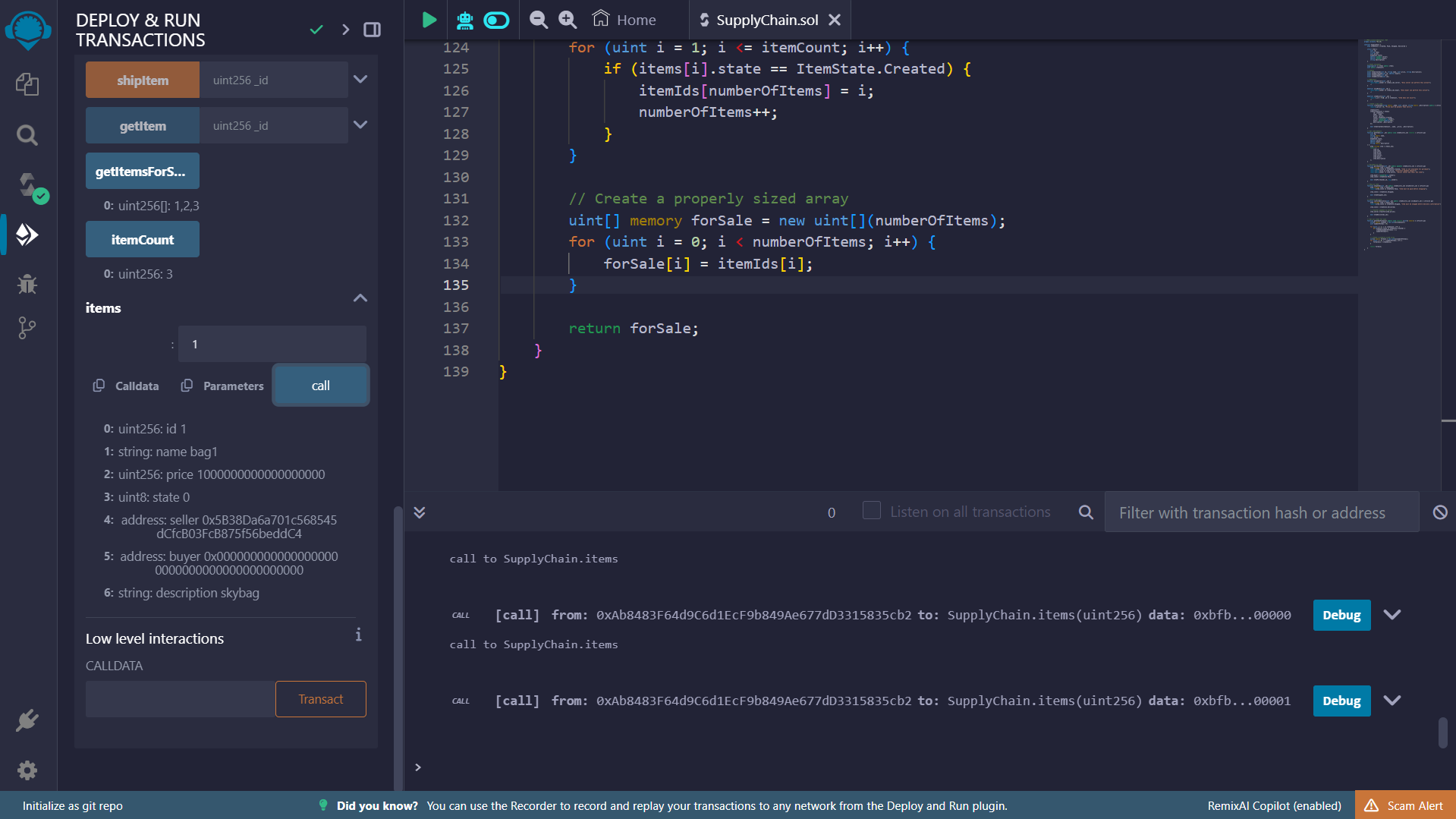
**Output:**

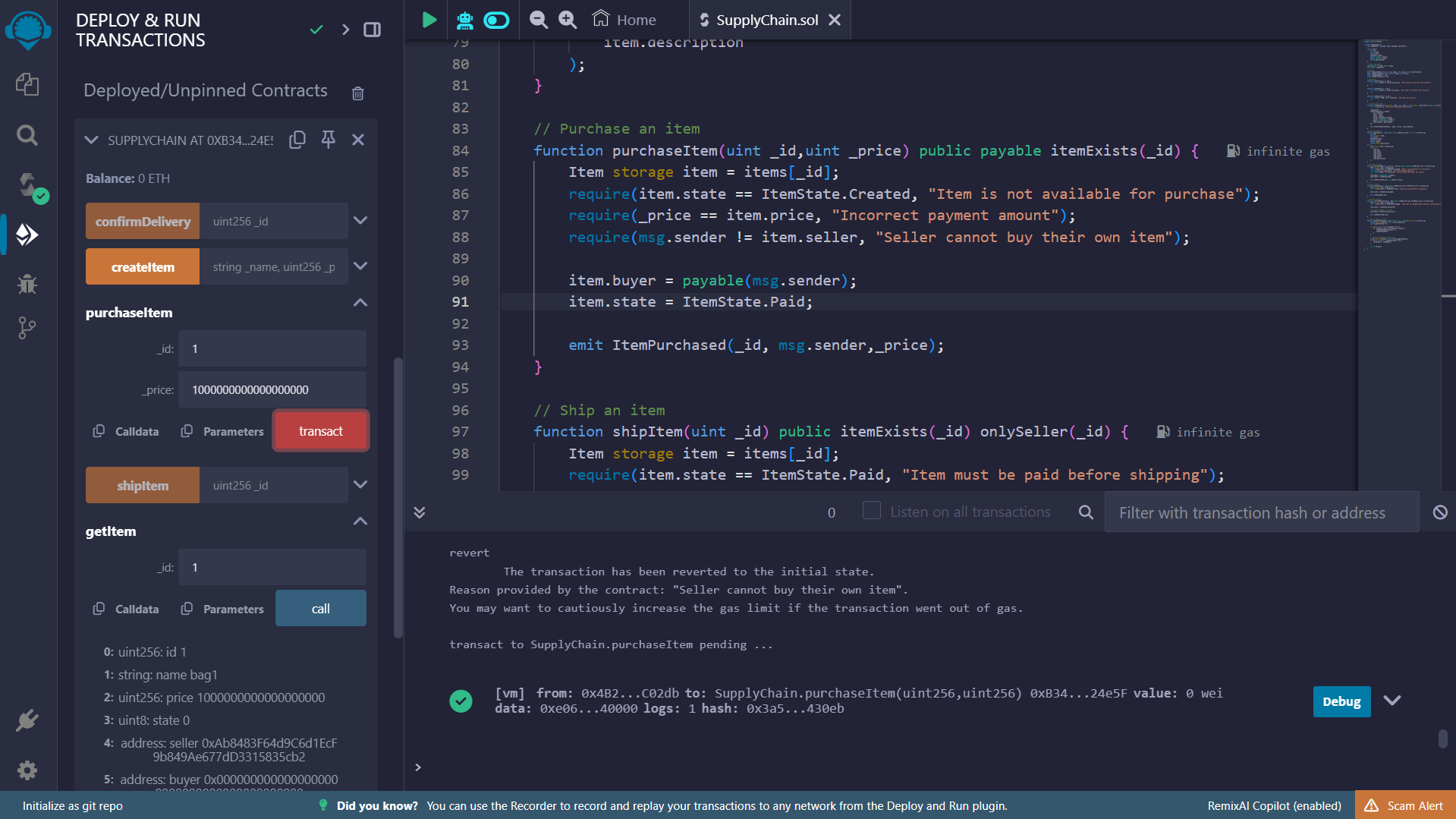
****

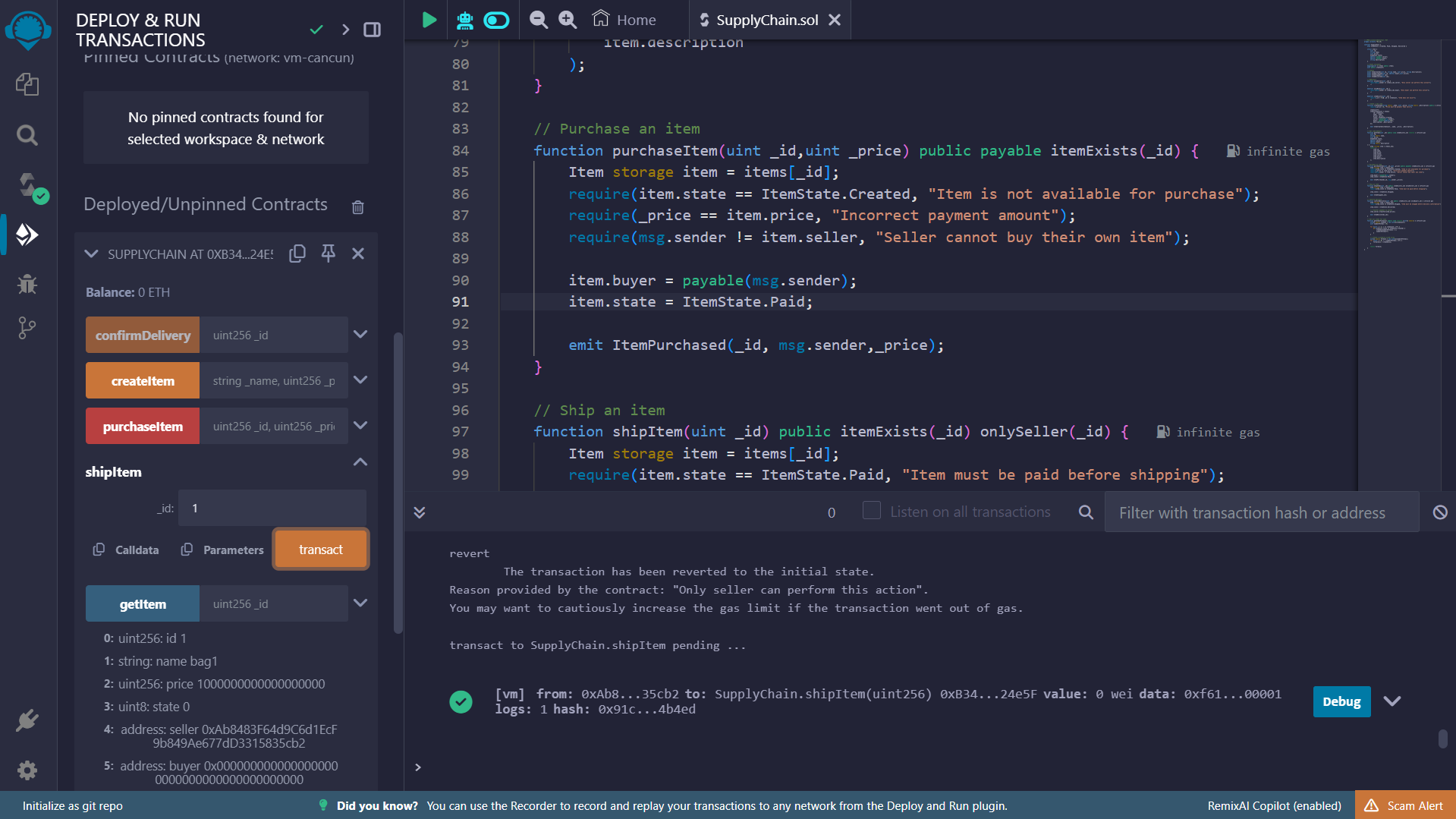
****

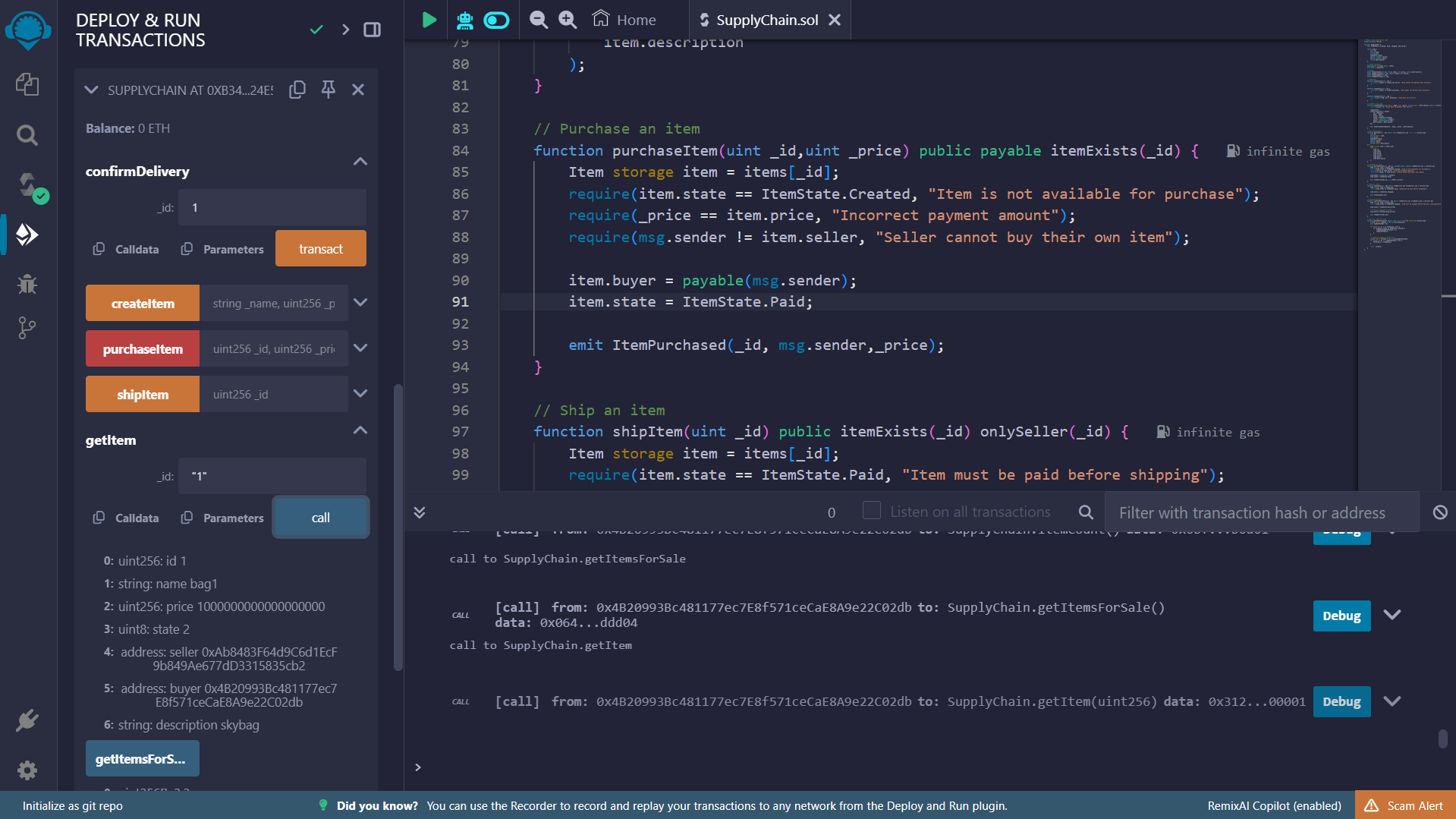
****

****

****

****

****

****