September 02 2024

CS61065 Theory and Applications of Blockchain

Assignment 3: Ethereum Solidity

Date of Submission: **September 13, 2024 EOD**

In this assignment, you will get familiar with Ethereum Smart Contracts . You will learn how to compile, deploy and interact with smart contracts in Ethereum. Use the account created in Ethereum Sepolia Network in the previous assignment for the transactions.

Submission Instructions

You need to submit the assignment as a group of a maximum of 2 members. Only one member of each group should submit the assignment. In case of multiple submissions, the final one before the deadline will be considered.

Submit the answers to the questions in the following Google Forms link:

https://forms.gle/iw8yBPH9dR1kr9h47

Mark Distribution:

Part A: 5 Marks (Q1 - 1 Mark, Q2 - 2 Marks, Q3 - 2 Marks)
Part B: 30 Marks (The code should run in the demonstration without any error, and marks for each individual component are mentioned in Part C Question Description). Marks will be deducted if the smart contract address is not submitted to the form by the deadline.

Part A

Use the **Sepolia faucet (https://cloud.google.com/application/web3/faucet/ethereum)** to obtain some Ether in your account.

Use **web3.js** to query and execute the following smart contract:

Smart Contract Address: **0x61b0592cA07C420260f5D3d0d1ba7A0B7fDf7126**

The smart contract stores your name and roll number corresponding to your Ethereum account address. Generate the ABI from the contract code. The code of the contract is as follows:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.5.8;

contract NameRollRegistry {
    struct Info {
        string name;
        string roll;
    }

    mapping(address => Info) public info;

    // Function to update both name and roll number
    function update(string calldata newName, string calldata
newRoll) public {
        info[msg.sender].name = newName;
        info[msg.sender].roll = newRoll;
    }

    // Function to get name and roll number by address
    function get(address addr) public view returns (string
memory, string memory) {
        return (info[addr].name, info[addr].roll);
    }

    // Function to get your own name and roll number
    function getmine() public view returns (string memory,
string memory) {
        return (info[msg.sender].name, info[msg.sender].roll);
    }
}
```

B.1. Query the name and roll for the address:
0x328Ff6652cc4E79f69B165fC570e3A0F468fc903

B.2. Input your own roll through the **update** function.

I. **Submit your Ethereum account address from which you made the transaction,**
II. **The transaction id.**

Part B: BlockCart with Ethereum Smart Contract

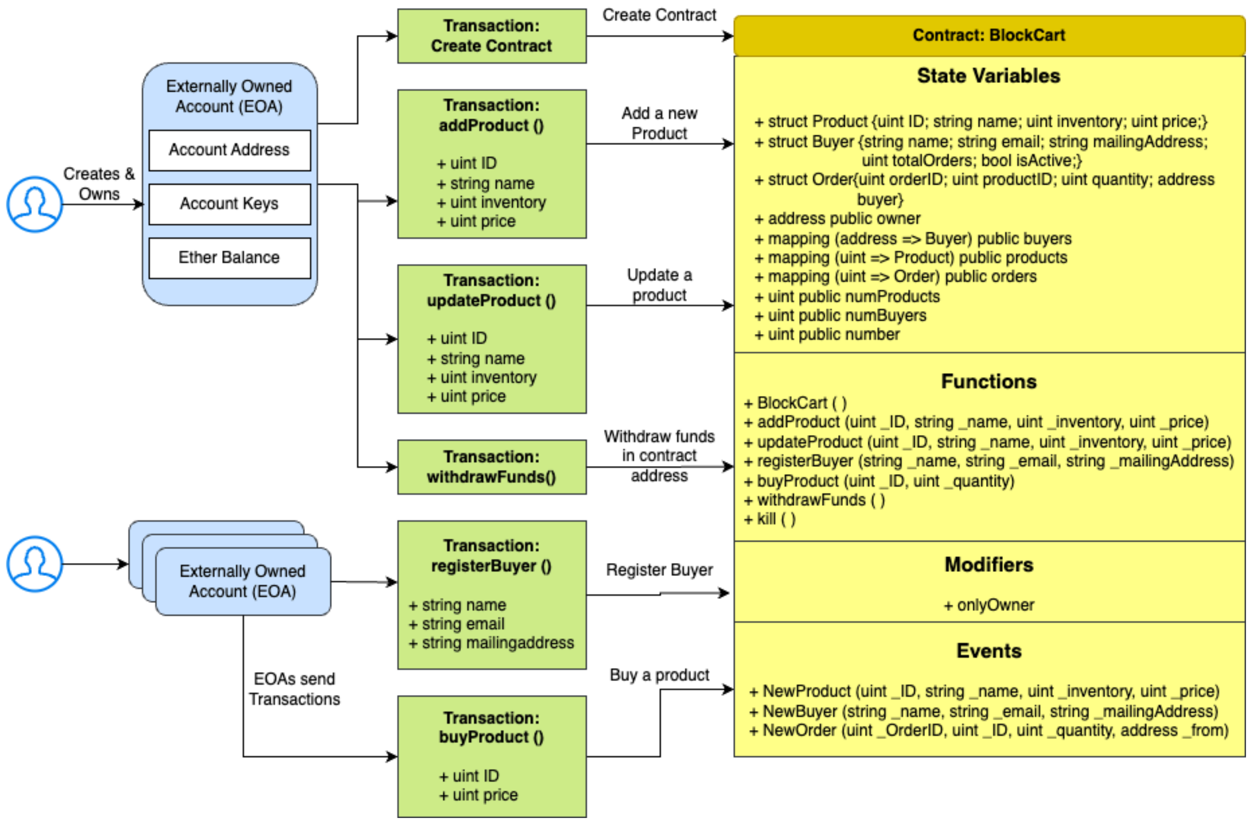
Only one member from each group should submit the assignment in Moodle. Clearly mention your group details in the submission. Deploy your smart contract in the Sepolia test network. You may use web3.js or the combination of Remix IDE and MetaMask extension to develop, test, and deploy your contract.

Write the address of the contract and the roll numbers of all group members as comments in the solidity file. Submit the smart contract address (one for each group) in the given google form. Submit your solidity code as a single file with .sol extension as <RollNo>.sol in Moodle.

In this assignment, you will get familiar with Ethereum smart contracts in solidity language. You will learn how you can use blockchain to enforce certain rules in a distributed setting in order to construct applications where the users do not necessarily trust each other.

Description:

Design a ***BlockCart*** Smart Contract for product sales. With the BlockCart contract, the owner of the contract can sell products to the buyers.


Creates &
Owns
Externally Owned
Account (EOA)
Account Address
Account Keys
Ether Balance
Transaction:
Create Contract
Create Contract
Transaction:
addProduct ()
+ uint ID
+ string name
+ uint inventory
+ uint price
Add a new
Product
Transaction:
updateProduct ()
+ uint ID
+ string name
+ uint inventory
+ uint price
Update a
product
Transaction:
withdrawFunds()
Withdraw funds
in contract
address
Externally Owned
Account (EOA)
Transaction:
registerBuyer ()
+ string name
+ string email
+ string mailingaddress
Register Buyer
EOAs send
Transactions
Transaction:
buyProduct ()
+ uint ID
+ uint price
Buy a product
Contract: BlockCart
State Variables
+ struct Product {uint ID; string name; uint inventory; uint price;}
+ struct Buyer {string name; string email; string mailingAddress;
uint totalOrders; bool isActive;}
+ struct Order{uint orderID; uint productID; uint quantity; address
buyer}
+ address public owner
+ mapping (address => Buyer) public buyers
+ mapping (uint => Product) public products
+ mapping (uint => Order) public orders
+ uint public numProducts
+ uint public numBuyers
+ uint public number
Functions
+ BlockCart ( )
+ addProduct (uint _ID, string _name, uint _inventory, uint _price)
+ updateProduct (uint _ID, string _name, uint _inventory, uint _price)
+ registerBuyer (string _name, string _email, string _mailingAddress)
+ buyProduct (uint _ID, uint _quantity)
+ withdrawFunds ( )
+ kill ( )
Modifiers
+ onlyOwner
Events
+ NewProduct (uint _ID, string _name, uint _inventory, uint _price)
+ NewBuyer (string _name, string _email, string _mailingAddress)
+ NewOrder (uint _OrderID, uint _ID, uint _quantity, address _from)

The owner can add a product for sale by sending a transaction with the product ID, product name, inventory, and price. The buyers who already have their Ethereum accounts created can register with the *BlockCart* contract by sending a transaction to the contract with their name, email, and mailing address. A registered buyer can then place an order for a product by sending a transaction to the contract with the product ID, quantity, and value equal to (or greater than) the order amount. If the buyer pays more than the order amount, the balance is refunded when the transaction for a new order is processed.

Write a constructor as follows: **(5 Marks)**

```
constructor() {

       owner = msg.sender;

       numBuyers = 0;

       numProducts = 0;

   }
```

In the smart contract, define a custom data type to represent a ***Product, Buyer, and Order***. **(3 Marks)**

The ***Buyer, Product, and Order*** variables maintain a mapping from the address of the buyers, number of products, and orders respectively. **(2 Marks)**

There are three events ***NewProduct, NewBuyer, and NewOrder.*** **(5 Marks)**

Only a seller can send a transaction to the addProduct function of the contract along with the product ID, product name, inventory, and price to add a product. Similarly ***updateProduct*** update the price of a product. **(10 Marks)**
The function ***addProduct*** is defined as follows:

```
function addProduct(uint _ID email, string _name, uint
_inventory, string _price){
. . .
}
```

The buyers who already have their Ethereum accounts created can register with the *BlockCart* contract by sending a transaction to the ***registerBuyer*** with their name, email, and mailing address. A registered buyer can then place an order for a product by sending a transaction to the function ***buyProduct*** with the product ID, quantity, and value equal to (or greater than) the order amount. **If the buyer pays more than the order amount, the balance is refunded when the transaction for a new order is processed**. **(10 Marks)**

The function ***buyProduct*** is defined as follows:

```
function buyProduct(uint _ID email, string _name, uint
_inventory, string _price) returns (uint newOrderID){
. . .
}
```

The amount paid by all the buyers for the products purchased is held in the contract account. The contract Seller (or beneficiary) can withdraw the amount from the contract account by sending a transaction to the "***withdrawFunds***" function. This function has a modifier ***"onlyOwner"*** to check if the transaction is sent by the owner. **(5 Marks)**

```
modifier onlyOwner() {
 . .
}
```

```
function withdrawFunds() public onlyOwner {
```

```
. . .
}
```

Follow the UML diagram for the detailed skeleton of the BlockCart Smart Contract.

Additional Functions:

In addition to the function, also write the following functions to debug and test the smart contract:

```
function getBuyerAmountPaid(address buyer) public view returns (uint)
```

Return the total amount paid by a buyer.

```
function kill() public onlyOwner
```

To kill the BlockCart buy event.

References

Solidity documentation: https://docs.soliditylang.org/en/v0.8.7/

Remix Editor: https://remix.ethereum.org/