



DECENTRALIZED MARKETPLACE SMART CONTRACT

Empowering Decentralized Buying and Selling

SUBMITTED BY:
CHINMAYEE SHARMA

Table of Contents

- Overview of Problem Statement
- Objective
- Code snippets
- Solution Overview
- Testing
- Conclusion

Overview of Problem Statement

Challenges in Traditional Marketplaces

- **Centralization Issues:**

Traditional marketplaces are often centralized, leading to a lack of transparency and increased vulnerability to fraud.

- **Intermediaries:**

The involvement of intermediaries may result in higher fees and slower transaction processes.

- **Limited Security:**

Security concerns arise due to centralized control, making data and transactions susceptible to hacks.

The Need for a Decentralized Solution

- **Reduced Fees:**

With no middlemen, users can transact directly, saving money on commissions.

- **Transparency:**

All transactions and agreements are recorded on the blockchain, making them transparent and tamper-proof.

- **Censorship Resistance:**

Users have full control over their transactions, reducing the risk of censorship.

- **Global Access:**

Anyone with an internet connection can participate in the marketplace.

Objectives

Clear Objectives of the Smart Contract

- **Decentralization:** Enable decentralized buying and selling to eliminate the need for centralized control.
- **Transparency:** Provide transparency in transactions by leveraging blockchain technology.
- **Security:** Enhance security by securing ownership and transaction details on the blockchain.
- **Efficiency:** Automate and streamline transactions through the use of smart contracts.
- **User Empowerment:** Empower users by reducing reliance on intermediaries and providing direct control over transactions.

Addressing Key Requirements.

- **Seller Listings:** Allow sellers to list items with names and prices.
- **Buyer Transactions:** Enable buyers to purchase items using Ether equal to the listed price.
- **Secure Ether Transfer:** Ensure secure transfer of Ether from buyers to sellers.
- **Prevent Further Sales:** Implement functionality to mark items as sold upon purchase, preventing further sales.

Enhancing Transparency and Security in Transactions

- **Smart Contracts:** Utilize smart contracts for automated and secure transactions.
- **Blockchain Technology:** Leverage blockchain to create a tamper-resistant and transparent ledger.

```
Decentralised Marketplace.sol X
1 /*
2 Problem Statement 1: Decentralized Marketplace
3
4 Objective: Develop a Decentralized Marketplace contract where users can list items for sale and others
5 can purchase them using ether.
6
7 Sellers should list items with a name and price.
8 Buyers can purchase items by sending Ether equal to the listed price.
9 Ensure that the Ether sent by buyers is transferred to the seller.
10 Implement functionality to mark items as sold upon purchase to prevent further sales.
11 */
12
13 // SPDX-License-Identifier: MIT
14 pragma solidity ^0.8.0;
15
16 contract DecentralizedMarketplace {
17
18     struct Product {
19         uint256 productId;
20         string name;
21         uint256 price;
22         address seller;
23         bool sold;
24         string details;
25     }
26
27     address public owner;
28
29     //mapping productId with Product
30     mapping(uint256 => Product) public products;
31
32     //total number of products in a list
33     uint256 public productCount;
34
35     // Mapping to keep track of authorized sellers
36     mapping(address => bool) public authorizedSellers;
37
38     modifier authOwner() {
39         require(msg.sender == owner, "You are not the owner");
40         _;
41     }
42
43     modifier isAuthorizedSeller() {
44         require(authorizedSellers[msg.sender], "Real ID se aao seller bhai");
45         _;
46     }
47
48     modifier productAvailable(uint256 productId) {
49         require(productId < productCount, "Product is out of Stock");
50         _;
51     }
52
53     modifier notSold(uint256 productId) {
54         require(!products[productId].sold, "Sold out");
55         _;
56     }
57
58     constructor() {
59         owner = msg.sender;
60         productCount = 0; //Initially no. of products set to 0
61     }
62
63     // Allowing the owner to authorize sellers
64     function authorizeSeller(address _seller) external authOwner {
65         authorizedSellers[_seller] = true;
66     }
67
68     // Allowing authorized sellers to add products to the list
69     function addProduct(string memory _name, uint256 _price, string memory _details) external isAuthorizedSeller {
70
71         products[productCount] = Product({
72             productId: productCount,
73             name: _name,
74             price: _price,
75             seller: msg.sender,
76             sold: false,
77             details: _details
78         });
79
80         productCount++;
81     }
82
83     //Buyers can purchase items by sending ether equal to the listed price. ....
84
85     function buyProduct(uint256 productId) external payable productAvailable(productId) notSold(productId) {
86
87         Product storage product = products[productId];
88
89         //the ether send by buyer is sufficient or not
90         require(msg.value == product.price, "Pure pese de bhai");
91
92         // Ensuring that the Ether sent by buyers is transferred to the seller.
93         payable(product.seller).transfer(msg.value);
94         product.sold = true;
95     }
96
97     // Implement functionality to mark item as sold upon purchase to prevent further sales.
98     function isProductSold(uint256 productId) external view returns (bool) {
99         return products[productId].sold;
100     }
101
102     // New function to get the product ID by providing the name
103     function getProductIdByName(string memory productName) external view returns (uint256) {
104         for (uint256 i = 0; i < productCount; i++) {
105             if (keccak256(abi.encodePacked(products[i].name)) == keccak256(abi.encodePacked(productName))) {
106                 return i;
107             }
108         }
109         revert("Product not found");
110     }
111 }
112
```

```
Decentralised Marketplace.sol X
58 constructor() {
59     owner=msg.sender;
60     productCount=0; //Initially no. of products set to 0
61 }
62
63 // Allowing the owner to authorize sellers
64 function authorizeSeller(address _seller) external authOwner {
65     authorizedSellers[_seller] = true;
66 }
67
68 // Allowing authorized sellers to add products to the list
69 function addProduct(string memory _name, uint256 _price, string memory _details) external isAuthorizedSeller {
70
71     products[productCount] = Product({
72         productId: productCount,
73         name: _name,
74         price: _price,
75         seller: msg.sender,
76         sold: false,
77         details: _details
78     });
79
80     productCount++;
81 }
82
83 //Buyers can purchase items by sending ether equal to the listed price. ....
84
85 function buyProduct(uint256 productId) external payable productAvailable(productId) notSold(productId) {
86
87     Product storage product = products[productId];
88
89     //the ether send by buyer is sufficient or not
90     require(msg.value == product.price, "Pure pese de bhai");
91
92     // Ensuring that the Ether sent by buyers is transferred to the seller.
93     payable(product.seller).transfer(msg.value);
94     product.sold = true;
95 }
96
97 // Implement functionality to mark item as sold upon purchase to prevent further sales.
98 function isProductSold(uint256 productId) external view returns (bool) {
99     return products[productId].sold;
100 }
101
102 // New function to get the product ID by providing the name
103 function getProductIdByName(string memory productName) external view returns (uint256) {
104     for (uint256 i = 0; i < productCount; i++) {
105         if (keccak256(abi.encodePacked(products[i].name)) == keccak256(abi.encodePacked(productName))) {
106             return i;
107         }
108     }
109     revert("Product not found");
110 }
111 }
112
```

Solution Overview

- **Decentralized Marketplace Contract:**

Empower users to buy and sell products without central authority.

- **Key Components:**

1. **Product Struct:** Defines the structure for product data, including name, price, seller, and transaction status.

2. **Functions:**

- ✓ **authorizeSeller:** Owner authorizes sellers
- ✓ **addProduct:** Sellers add products
- ✓ **buyProduct:** Buyers purchase products
- ✓ **isProductSold:** Checks product status
- ✓ **getProductIdByName:** Retrieves product ID by name

3. **Modifiers:**

- ❑ **authOwner:** Ensures owner authentication
- ❑ **isAuthorizedSeller:** Ensures authorized seller authentication
- ❑ **productAvailable:** Ensures product availability
- ❑ **notSold:** Ensures the product is not sold

- **Secure and Transparent Transactions**

Utilises blockchain technology and smart contracts for enhanced security and transparency.

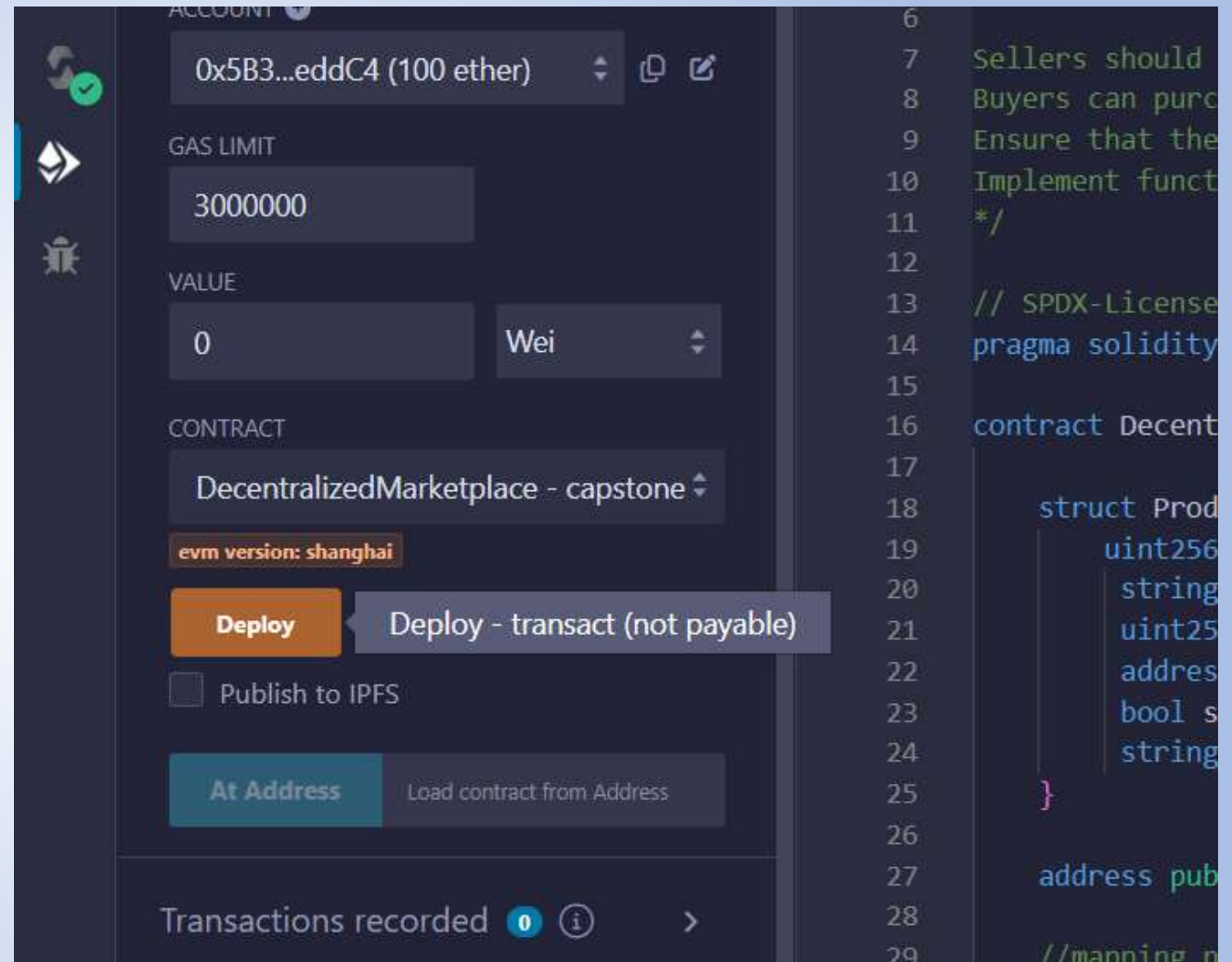
- **Objective**

Provide a decentralized and secure platform for transparent buying and selling.

Testing of the decentralized marketplace contract

Step 1: Deploy the Contract

1. Open Remix at <https://remix.ethereum.org/>.
2. Paste the modified Solidity code into the editor.
3. Choose a Solidity compiler version (e.g., 0.8.0) in the "Solidity" tab.
4. Click the "Compile" button to compile the contract.
5. Switch to the "Deploy & Run Transactions" tab.
6. In the "Environment" section, select "Remix VM Shanghai" as the environment.
7. Click the "Deploy" button to deploy the contract.



Step 2: Authorize Sellers (Owner's Action) and Add Products

1. Use the **authorizeSeller** function to authorize some addresses as sellers.
2. Use the **addProduct** function to add the products in the marketplace.
3. Example: Adding a product with the name Laptop, price 2 ether, and details "HP Intel Core i5".
4. Add more product with different details.

The screenshot displays a web application for a decentralized marketplace. The main panel shows the 'Deployed Contracts' section for 'DECENTRALIZEDMARKETPLACE AT'. The balance is 0 ETH. The 'addProduct' function is being interacted with, showing fields for name, price, and details. The 'transact' button is highlighted, and a tooltip indicates 'addProduct - transact (not payable)'. The right panel shows the Solidity code for the contract.

```
4 Objective: Develop a
5 can purchase them us
6
7 Sellers should list
8 Buyers can purchase
9 Ensure that the Ethe
10 Implement functional
11 */
12
13 // SPDX-License-Iden
14 pragma solidity ^0.8
15
16 contract Decentraliz
17
18
19 uint256 prod
20 string name
21 uint256 pri
22 address sel
23 bool sold;
24 string deta
25 }
26
27 address public c
28
```

Step 3: Buy Products

1. Use the **buyProduct** function to simulate the buyers purchasing products.
2. Example: buying the product with **productId 0** by sending 2 ether.
3. Check if the product is marked as sold by calling the **isProductSold** function.

The screenshot shows a web interface for a blockchain application. The top navigation bar has three tabs: 'Calldata', 'Parameters', and 'transact'. The 'transact' tab is selected. Below the tabs, there is a section for the 'buyProduct' function. The 'productId' is set to '0'. A red 'transact' button is highlighted, and a tooltip shows 'buyProduct - transact (payable)'. Below the 'buyProduct' section, there are three other functions: 'authorizedSel...', 'getProductId...', and 'isProductSold'. The 'isProductSold' function is selected, and its parameters are shown as 'uint256 productId'. The right side of the screen displays Solidity code, including comments and function definitions.

```
9 Ensure that th
10 Implement func
11 */
12
13 // SPDX-Licens
14 pragma solidit
15
16 contract Decer
17
18 struct Pro
19     uint25
20     strin
21     uint2
22     addre
23     bool
24     strin
25 }
26
```

CONCLUSION

Summary of the Smart Contract's Capabilities

- **Decentralization:** Provides a decentralized platform for buying and selling.
- **Transparency:** Utilizes blockchain for transparent and tamper-resistant transactions.
- **Security:** Enhances security through smart contracts and secure ownership.
- **Efficiency:** Automates transactions, eliminating the need for intermediaries.
- **User Empowerment:** Direct control over transactions, reducing reliance on third parties.

Contributions to Decentralized Marketplaces

- **Elimination of Intermediaries:** Reduces fees and accelerates transaction processes.
- **Secure Transactions:** Ensures secure ownership and reduces fraud risk.
- **Efficient Marketplace:** Streamlines buying and selling processes.

Benefits of Transparency and Security

- **Tamper-Resistant Transactions:** Transactions recorded on the blockchain are resistant to tampering.
- **Trust and Confidence:** Builds trust and confidence among users through transparent processes.
- **Future of Decentralized Commerce:** Paves the way for the future of decentralized commerce.