

IT465 Assignment 2 - Report

Sachin Prasanna - 211IT058

September 4, 2024

1 Problem Statement

1) Create a smart contract for Crowdfunding smartcontract.

Crowdfunding smart contract creates contracts for fundraising purposes only and collects ETH. An investor can invest in a way that creates a transaction that sends an investor to this contract. The contract sets up the deadline and target amount for the fundraising activities and remits the collected ETH to the owner of the contract when the target value is achieved at the closing date. If the target value is not achieved, return ETH to the investor. There are two scenarios, Success and Failed of funding.

2) CryptoFlight and its smart contract.

The core idea with CryptoFlight is to provide a way for airlines to sell remaining tickets for a price as good as possible and for travelers to get left-over tickets as cheap as possible. An airline posts an offer with a departure and destination as well as their minimum acceptable bid. The travelers can list all available flights and make make sure that all deployed contracts actually uses your code. We will have a separate factory-contract. This contract is responsible for holding a list of deployed contracts as well as acting as a creator of new contract.

Use MetaMask as well as GANACHE Truffle Suite to Deploy a Smart Contract in Solidity

2 Solution to Question 1

2.1 Code

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.26;
3
4 contract Crowdfunding {
5     mapping(address => uint) investorBalanceMapping;
6     mapping(address => bool) hasInvestorFunded;
7     address[] investorAddresses;
8
9     address public ownerAddress;
10    uint public deadline;
```

```

11     uint public goalAmount;
12
13     constructor(uint _duration, uint _goalAmount) {
14         deadline = block.timestamp + _duration;
15         goalAmount = _goalAmount * 1 ether;
16         ownerAddress = msg.sender;
17     }
18
19     modifier onlyOwner() {
20         require(msg.sender == ownerAddress, "Only the owner can perform this
21             action");
22         _;
23     }
24
25     modifier onlyInvestor() {
26         require(msg.sender != ownerAddress, "Only an Investor can perform this
27             action");
28         _;
29     }
30
31     function fund() external payable onlyInvestor {
32         require(block.timestamp <= deadline, "Deadline Exceeded");
33         require(msg.value > 0, "ETH value should be greater than 0");
34
35         if (!hasInvestorFunded[msg.sender]) {
36             investorAddresses.push(msg.sender);
37             hasInvestorFunded[msg.sender] = true;
38         }
39
40         investorBalanceMapping[msg.sender] += msg.value;
41     }
42
43     function getInvestedAmount() view public onlyInvestor returns (uint) {
44         return investorBalanceMapping[msg.sender];
45     }
46
47     function checkFundedAmount() view external returns (uint) {
48         return address(this).balance;
49     }
50
51     function isGoalReached() external onlyOwner returns (string memory) {
52         require(block.timestamp > deadline, "Deadline not passed");
53         uint currentFundedAmount = address(this).balance;
54         require(currentFundedAmount > 0, "Money has been withdrawn");
55
56         if (currentFundedAmount >= goalAmount) {
57             payable(ownerAddress).transfer(currentFundedAmount);
58             return "Goal Successfully Reached. Money Transferred to Owners.";
59         }
60
61         else {
62             for (uint i = 0 ; i < investorAddresses.length; i++) {
63                 uint investorFundedAmount = investorBalanceMapping[
64                     investorAddresses[i]];

```

```

62         payable(investorAddresses[i]).transfer(investorFundedAmount);
63     }
64
65     return "Goal Not Reached. Money Transferred to Investors.";
66 }
67 }
68 }

```

2.2 Functionalities

The **CrowdFunding** smart contract manages a crowdfunding campaign, allowing investors to contribute ether towards a funding goal and facilitating the transfer of funds based on the campaign's success.

- **Funding Contributions:** Investors can contribute ether to the campaign by calling the `fund` function. The contract verifies that the contribution amount is positive and ensures that each investor is only recorded once in the `investorAddresses` array.
- **Investment Tracking:** The `getInvestedAmount` function allows investors to check the total amount they have contributed to the campaign.
- **Campaign Status Check:** The `checkFundedAmount` function provides the current total amount of ether accumulated in the contract.
- **Goal Achievement Verification:** The `isGoalReached` function, callable only by the contract owner, checks if the total amount raised meets or exceeds the funding goal. If the goal is reached, it transfers the entire balance to the owner. If not, it refunds all contributors by transferring their contributions back to them.
- **Access Control:** The contract uses access modifiers to restrict functionality. The `onlyOwner` modifier ensures that only the contract owner can perform actions related to checking the funding goal and handling fund transfers. The `onlyInvestor` modifier restricts certain actions to investors and prevents the owner from calling those functions.

2.3 Outputs

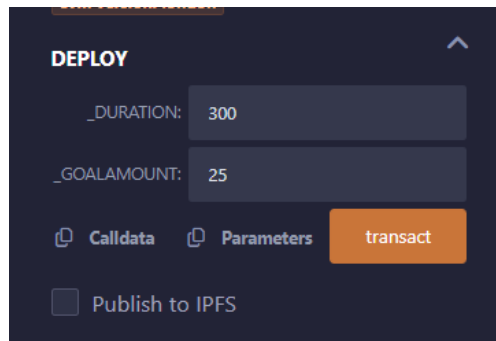


Figure 1: Deploying the contract with the parameters specified in the question

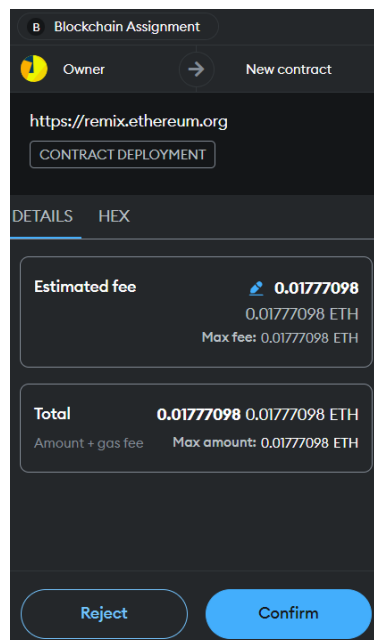


Figure 2: Corresponding MetaMask transaction

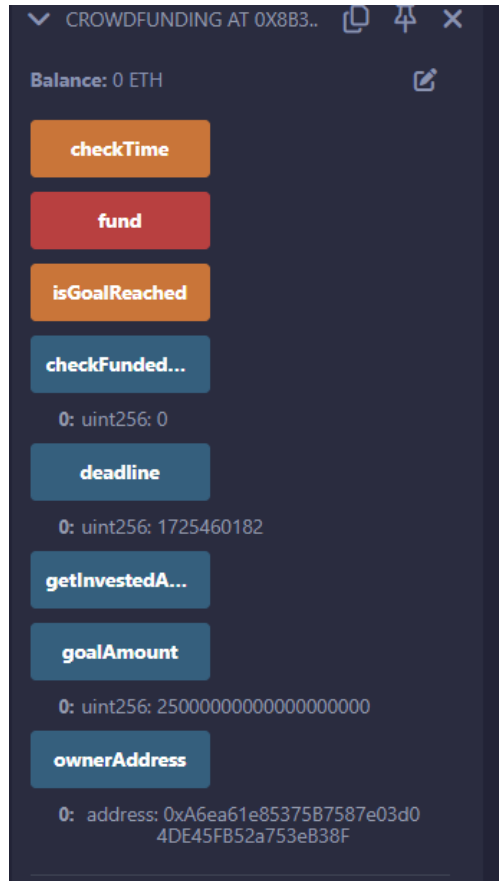


Figure 3: Functions and variables of the smart contract

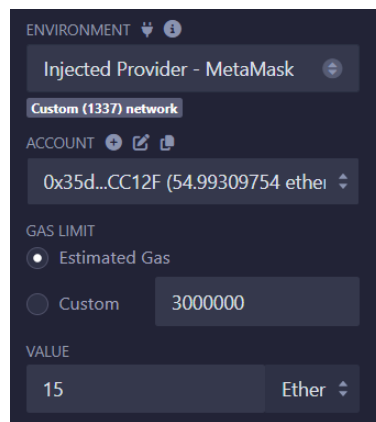


Figure 4: Transferring 15 Ether from Investor 1

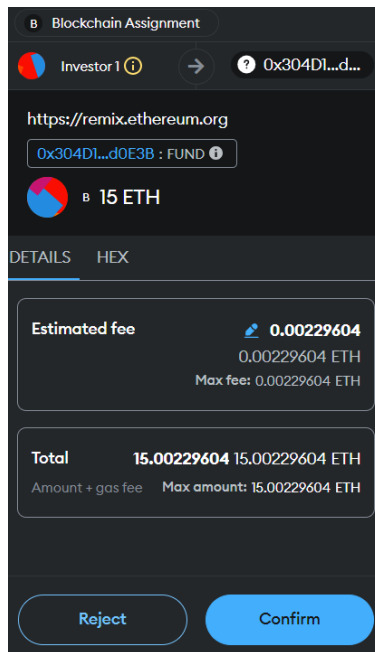


Figure 5: Corresponding MetaMask transaction

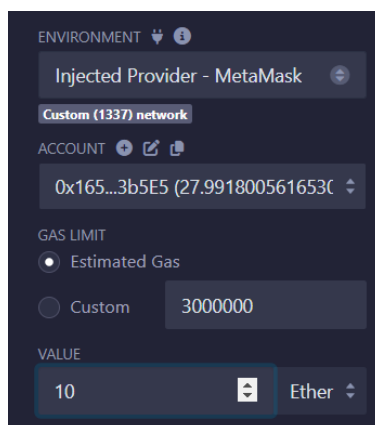


Figure 6: Transferring 10 Ether from Investor 2

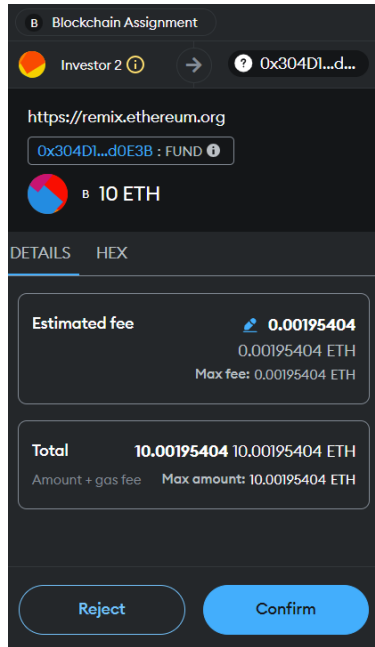


Figure 7: Corresponding MetaMask Transaction

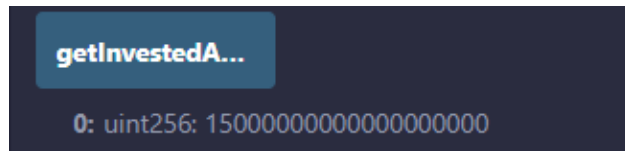


Figure 8: Total Wei (15 Ether) invested by Investor 1

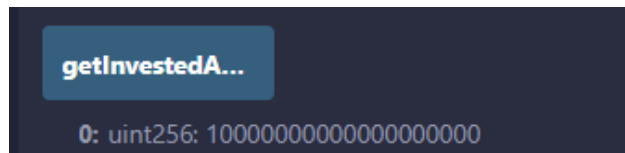


Figure 9: Total Wei (10 Ether) invested by Investor 2

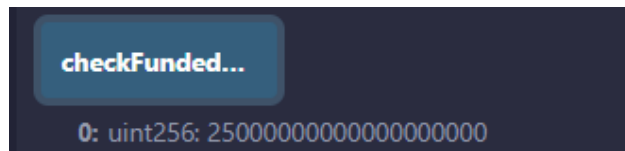


Figure 10: Total Wei (25 Ether) held by the smart contract or the fundraiser

ADDRESS	BALANCE	TX COUNT	INDEX	
0xA6ea61e85375B7587e03d04DE45F852a753eB38F	100.66 ETH	66	0	

Figure 11: Total Ether held by the owner

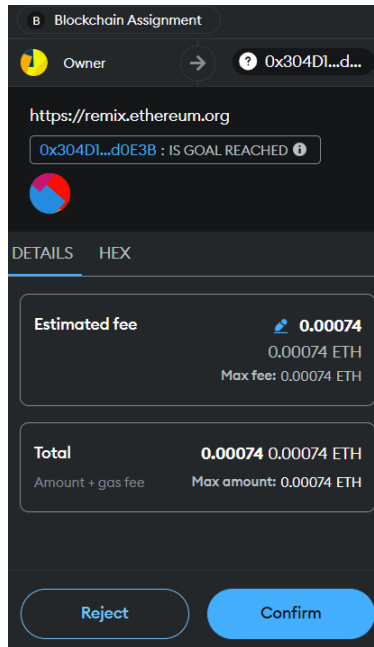


Figure 12: Corresponding MetaMask Transaction

ADDRESS	BALANCE	TX COUNT	INDEX	
0xA6ea61e85375B7587e03d04DE45FB52a753eB38F	125.66 ETH	68	0	

Figure 13: Total Ether held by the owner after isGoalReached function

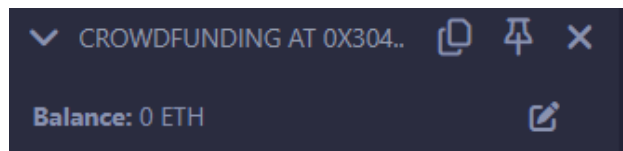


Figure 14: Ether held by the smart contract after the isGoalReached function

ADDRESS	BALANCE	TX COUNT	INDEX	
0xA6ea61e85375B7587e03d04DE45FB52a753eB38F	125.65 ETH	69	0	
0x16530558b401EB9232773D162D26F43F80F3b5E5	17.99 ETH	10	1	
0x35d38Af4A10395024019a88453b2fE116d1CC12F	39.99 ETH	5	2	

Figure 15: Ether held by all parties, Owner, Investor 1, Investor 2 in order

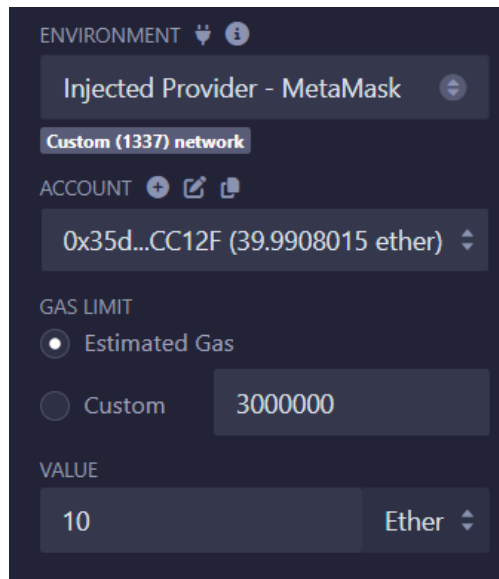


Figure 16: Beginning of second scenario: Transferring 10 Ether from Investor 1

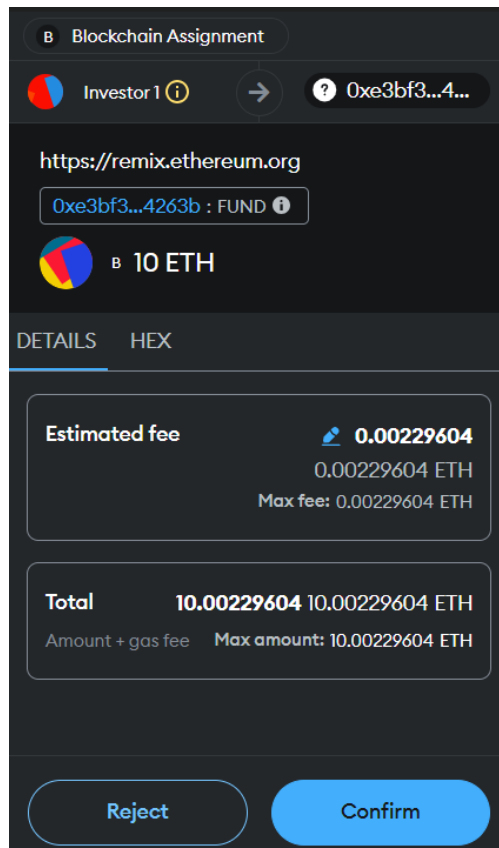


Figure 17: Corresponding MetaMask Transaction

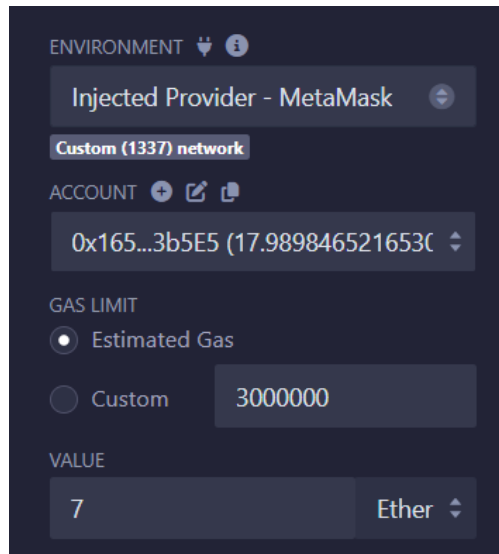


Figure 18: Transferring 7 Ether from Investor 2

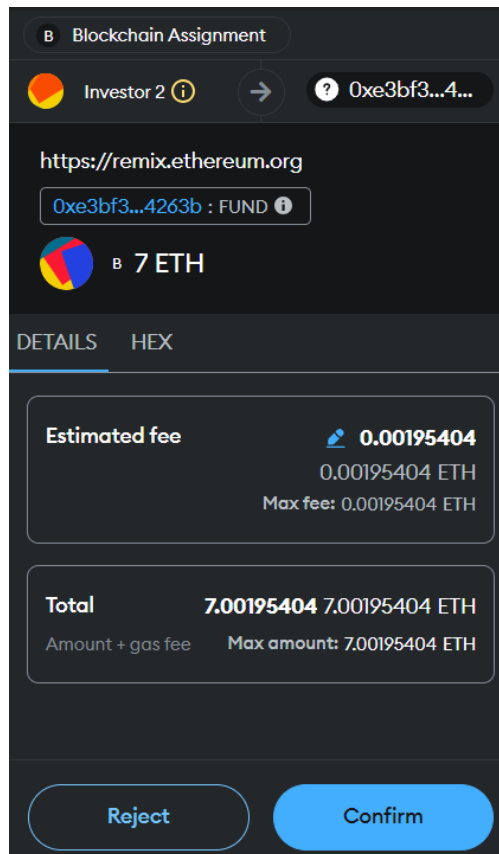


Figure 19: Corresponding MetaMask Transaction

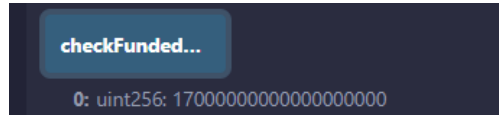


Figure 20: Total Wei (17 Ether) held by the smart contract or the fundraiser

ADDRESS 0xA6ea61e85375B7587e03d04DE45FB52a753eB38F	BALANCE 125.65 ETH	TX COUNT 69	INDEX 0	🔗
ADDRESS 0x16530558b401EB9232773D162D26F43F80F3b5E5	BALANCE 10.99 ETH	TX COUNT 11	INDEX 1	🔗
ADDRESS 0x35d38Af4A10395024019a88453b2fE116d1CC12F	BALANCE 29.99 ETH	TX COUNT 6	INDEX 2	🔗

Figure 21: Ether held by all parties, Owner, Investor 1, Investor 2 in order before isGoalReached function

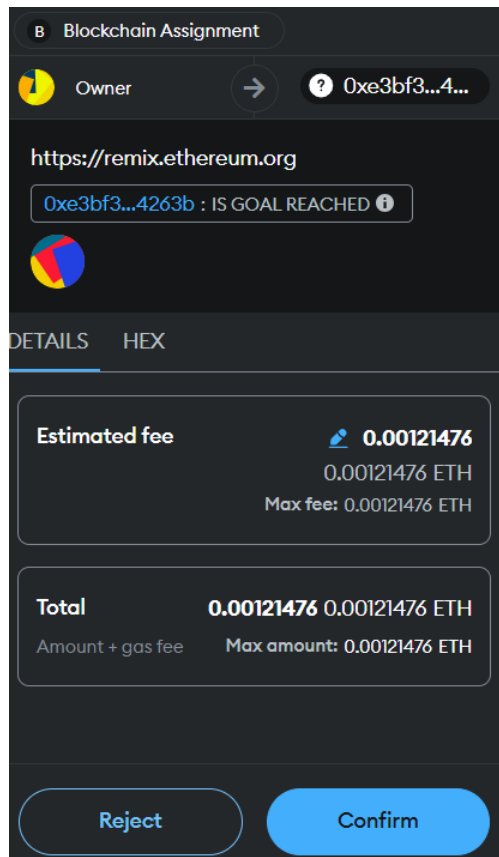


Figure 22: Corresponding MetaMask Transaction for isGoalReached function




ADDRESS 0xA6ea61e85375B7587e03d04DE45FB52a753eB38F	BALANCE 125.64 ETH	TX COUNT 70	INDEX 0	
ADDRESS 0x16530558b401EB9232773D162D26F43F80F3b5E5	BALANCE 17.99 ETH	TX COUNT 12	INDEX 1	
ADDRESS 0x35d38Af4A10395024019a88453b2fE116d1CC12F	BALANCE 39.99 ETH	TX COUNT 6	INDEX 2	

Figure 23: Ether held by all parties, Owner, Investor 1, Investor 2 in order, after isGoalReached function

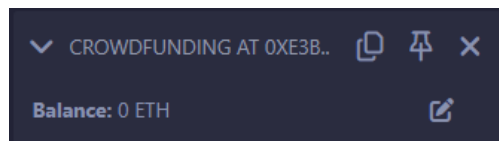


Figure 24: Ether held by the smart contract after the isGoalReached function

3 Solution for Question 2

3.1 Code

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.26;
3
4 contract FlightFactory {
5     Flight[] public flights;
6
7     function createFlight(uint256 minimumBid, string memory departure, string
8         memory destination) external {
9         Flight newFlight = new Flight(minimumBid, departure, destination, payable(
10             msg.sender));
11         flights.push(newFlight);
12     }
13
14     function getFlights() external view returns (Flight[] memory) {
15         return flights;
16     }
17 }
18
19 contract Flight {
20     struct Traveler {
21         uint256 bidAmount;
22         address payable account;
23         bool isApproved;
24     }
25
26     Traveler[] public travelers;
27     address payable public organizer;
28     uint256 public minimumBid;
29     string public departureLocation;
30     string public destinationLocation;
31     bool public isClosed;
32
33     modifier onlyOrganizer() {
34         require(msg.sender == organizer, "Not authorized");
35         _;
36     }
37
38     constructor(uint256 _minimumBid, string memory _departure, string memory
39         _destination, address payable _organizer) {
40         organizer = _organizer;
41         minimumBid = _minimumBid;
42         departureLocation = _departure;
43         destinationLocation = _destination;
44         isClosed = false;
45     }
46
47     function getFlightDetails() external view returns(address, uint256, string
48         memory, string memory, bool) {
```

```

45     return (organizer, minimumBid, departureLocation, destinationLocation,
46         isClosed);
47 }
48 function bidForFlight() external payable {
49     require(msg.value >= minimumBid, "Bid amount too low");
50
51     Traveler memory newTraveler = Traveler({
52         account: payable(msg.sender),
53         bidAmount: msg.value,
54         isApproved: false
55     });
56
57     travelers.push(newTraveler);
58 }
59
60 function finalizeFlight(uint256 seatsAvailable) external onlyOrganizer {
61     require(!isClosed, "Flight already finalized");
62
63     uint256 seatsAllocated = 0;
64
65     while (seatsAllocated < seatsAvailable && seatsAllocated < travelers.
66         length) {
67         uint256 highestBid = 0;
68         uint256 highestBidIndex = 0;
69
70         for (uint256 i = 0; i < travelers.length; i++) {
71             if (!travelers[i].isApproved && travelers[i].bidAmount > highestBid
72                 ) {
73                 highestBid = travelers[i].bidAmount;
74                 highestBidIndex = i;
75             }
76         }
77
78         if (highestBid > 0) {
79             travelers[highestBidIndex].isApproved = true;
80             seatsAllocated++;
81         } else {
82             break;
83         }
84     }
85
86     for (uint256 i = 0; i < travelers.length; i++) {
87         if (travelers[i].isApproved) {
88             organizer.transfer(travelers[i].bidAmount);
89         } else {
90             travelers[i].account.transfer(travelers[i].bidAmount);
91         }
92     }
93
94     isClosed = true;
95 }

```

3.2 Functionalities

The `FlightFactory` and `Flight` contracts provide the following functionalities:

- **Create Flight:** The `FlightFactory` contract allows users to create new flight contracts. This involves specifying a minimum bid amount, departure location, and destination. The flight contract is then initialized with these parameters and the address of the contract creator.
- **Retrieve Flights:** The `FlightFactory` contract provides a function to retrieve a list of all deployed flight contracts. This allows users to view and interact with the available flights.
- **Bid for Flight:** The `Flight` contract allows users to place bids for a flight by sending ether to the contract. The bid amount must be equal to or greater than the minimum bid set for the flight. Bids are recorded with the user's address and bid amount.
- **Get Flight Details:** The `Flight` contract provides a function to view the details of a flight. This includes the organizer's address, minimum bid amount, departure and destination locations, and whether the flight has been finalized.
- **Finalize Flight:** The `Flight` contract allows the flight organizer to finalize the flight. The function allocates available seats based on the highest bids. Approved travelers are marked, and the corresponding bid amounts are transferred to the organizer. Travelers who were not approved receive a refund of their bid amounts. The flight status is updated to closed after finalization.

3.3 Outputs

This subsection describes the scenario used for calculating and verifying the output of the deployed smart contracts. The corresponding outputs are then verified

- A `flight` contract is created with a minimum bid of 2 ETH.
- Six passengers place bids for the flight as follows:
 - Passenger 1 bids 4 ETH
 - Passenger 2 bids 5 ETH
 - Passenger 3 bids 6 ETH
 - Passenger 4 bids 7 ETH
 - Passenger 5 bids 8 ETH
 - Passenger 6 bids 9 ETH
- The airplane has 4 available seats.
- Based on the bids, the 4 highest bids will be selected. Consequently, the bids from Passenger 1 and Passenger 2 will not secure seats.

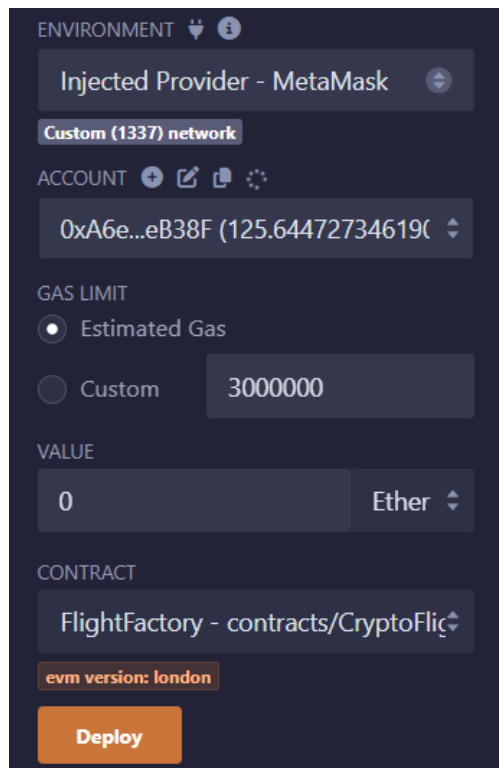


Figure 25: Deploying the FlightFactory Contract to make Flights

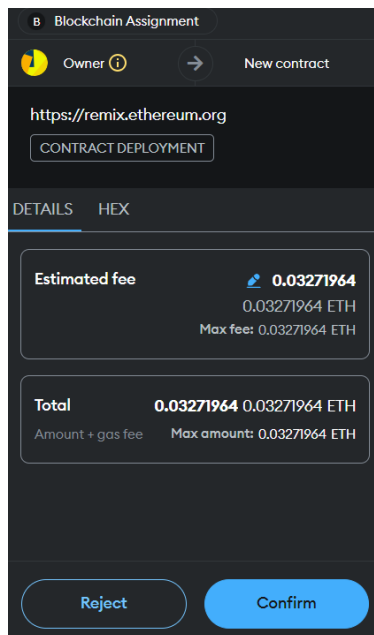


Figure 26: Corresponding MetaMask transaction



Figure 27: Functions and variables of the Flight Factory smart contract

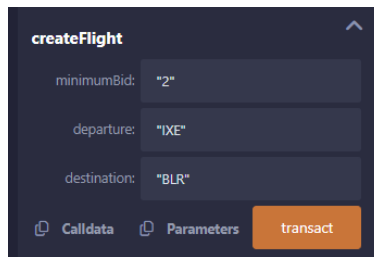


Figure 28: Creating a Flight from IXE to BLR with a minimum bid of 2

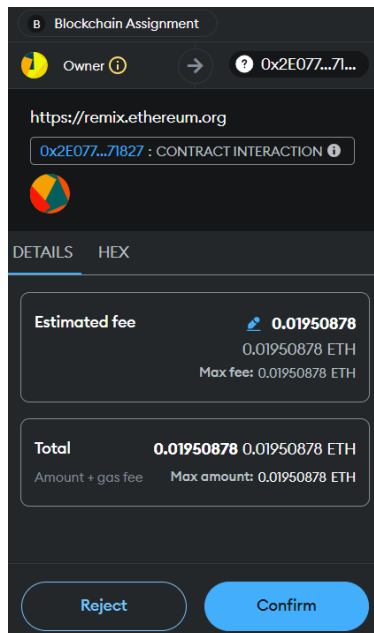


Figure 29: Corresponding MetaMask transaction

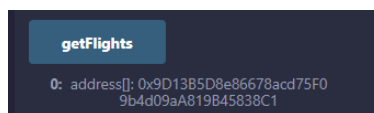


Figure 30: Address of the created Flight smart contract

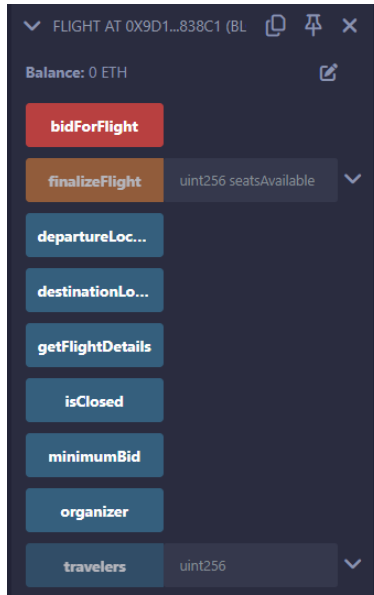


Figure 31: Functions and variables of the Flight smart contract

ADDRESS	BALANCE	TX COUNT	INDEX	
0xA6ea61e85375B7587e83d04DE45FB52a753eB38F	125.59 ETH	72	0	
0x16530558b401EB9232773D162D26F43F80F3b5E5	17.99 ETH	12	1	
0x35d38Af4A10395024019a88453b2fE116d1CC12F	39.99 ETH	6	2	
0xA82dce0ADb759FE32513E99E9f831E724873B39D	100.00 ETH	0	3	
0xB0e22a28775e0f655C0DDBC87A417249D0747526	100.00 ETH	0	4	
0xa279B7addB5ea50b72f5B43FFf5b0Ce0E3168564	100.00 ETH	0	5	
0x4409aFE2520174A07b499A581A1427797c41533	100.00 ETH	0	6	

Figure 32: Total Ether held by the owner till Passenger 6, in order as displayed

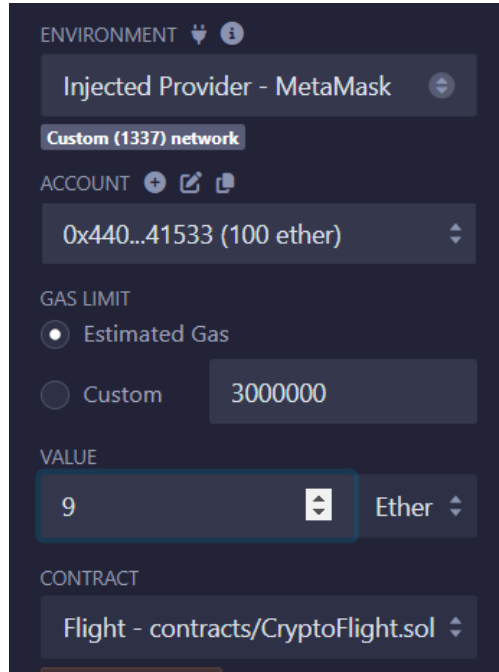


Figure 33: Total Ether bidded by Passenger 6 (Screenshots of other passengers are omitted because they are similar)

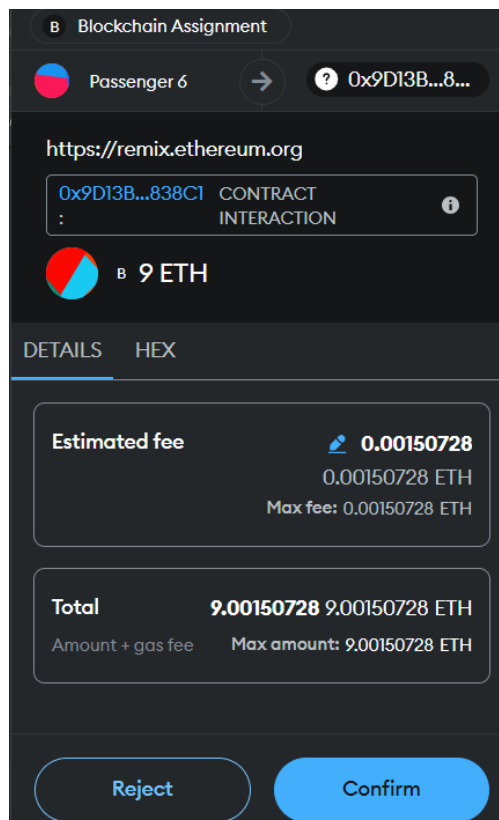


Figure 34: Corresponding MetaMask transaction

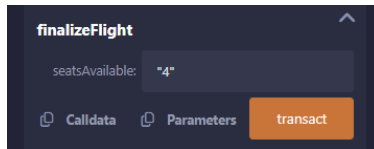


Figure 35: Setting the seats available to 4 as mentioned in the previous section and finalizing the Flight

ADDRESS	BALANCE	TX COUNT	INDEX	
0xA6ea61e85375B7587e03d04DE45FB52a753eB38F	125.59 ETH	72	0	🔗
0x16530558b401EB9232773D162D26F43F80F3b5E5	13.99 ETH	13	1	🔗
0x35d38Af4A10395024019a88453b2fE116d1CC12F	34.99 ETH	7	2	🔗
0xA82dce0ADb759FE32513E99E9f831E724873B39D	94.00 ETH	1	3	🔗
0xB0e22a28775e0f655c0DD8C87A417249D0747526	93.00 ETH	1	4	🔗
0xa279B7addB5ea50b72f5B43FFf5b0Ce0E3168564	92.00 ETH	1	5	🔗
0x4409aFE2520174A07b499A581A14277797c41533	91.00 ETH	1	6	🔗

Figure 36: Total Ether held by the owner till Passenger 6 after finalizeFlight function, in order as displayed

ADDRESS	BALANCE	TX COUNT	INDEX	
0xA6ea61e85375B7587e03d04DE45FB52a753eB38F	125.66 ETH	68	0	🔗

Figure 37: Total Ether held by the owner after the finalizeFlight function

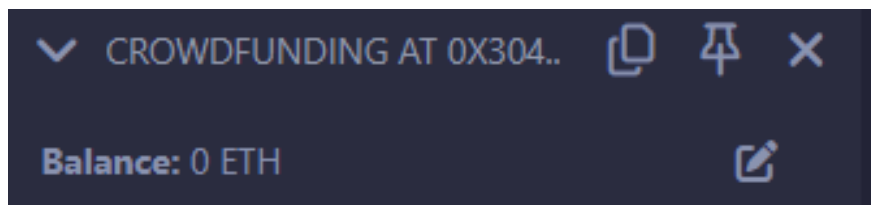


Figure 38: Ether held by the smart contract after finalizeFlight function