2024

Intelligent System for Passenger Seat in Vessels – I4ET

Smart Contract

GROUP 4
MUHAMMAD OMER FAROOQ ZAHID
MILAD JALILZADEHKHATOUNI
URFAN AGHAYEV

1 Motivation

The motivation behind developing the Intelligent System for Passenger Seats in Vessels stems from the need to enhance passenger comfort and convenience during ferry travel. Ferries are often subject to significant motions caused by water waves, which can result in discomfort for passengers. These motions include pitching, rolling, and yawing, which can cause seasickness and discomfort.

To mitigate these issues, we have developed a system that incorporates accelerometers as sensors to dampen these motions. Additionally, temperature and humidity sensors are integrated into the system to provide a comfortable environment for passengers. This system will provide an experience similar to the in-flight entertainment systems found on airplanes, where passengers can control various aspects of their immediate environment.

The system is implemented using a smart contract on the Ethereum blockchain, leveraging blockchain technology for its transparency, security, and immutability. The smart contract includes functions for logging in, adjusting temperature and humidity, activating the damped seat feature, and purchasing tokens. It also includes a feature to tell random jokes for passengers' entertainment.

This project showcases the potential of smart contracts and blockchain technology to improve passenger experiences in transportation systems, setting a precedent for future developments in marine travel comfort and convenience.

2 Use Case definition.

The Intelligent System for Passenger Seats in Vessels is designed to cater to the needs of passengers during ferry travel, focusing on comfort and convenience. The system's use cases and objectives are defined as follows:

2.1 Passenger Comfort and Environment Control:

Passengers can log into the system using their name and travel ID, granting them access to various features. They can adjust the temperature and humidity levels within predefined ranges to create a comfortable environment suited to their preferences. This capability ensures that passengers can enjoy a pleasant journey regardless of external conditions.

2.2 Motion Damping Feature Activation:

Passengers have the option to activate the damped seat feature using tokens allocated upon their first login. This feature utilizes accelerometer data to reduce vibrations and motions caused by water waves, thereby minimizing discomfort due to vessel movements. By using tokens, passengers can enhance their travel experience and mitigate the effects of sea motions, making their journey more enjoyable.

2.3 Owner Management and Control:

The system allows the owner, identified by a specific travel ID, to access administrative functions. The owner can add or remove customers, manage temperature and humidity settings, and activate the damped seat feature without the need to purchase tokens. This use case ensures that the owner has full control over the system and can maintain operational efficiency and customer satisfaction.

3 Design Aspects and Methodology

The design of the Intelligent System for Passenger Seats in Vessels encompasses the following key aspects:

- Sensor Integration:
    - Accelerometers: used to measure vessel motions & passenger seat motions
    - Temperature and Humidity Sensors: Monitor & allow passengers to change.
- Smart Contract on Blockchain:
    - Ethereum Blockchain
    - Token System
- User Interface:
    - In-flight Entertainment System: Tablet-like interface for passengers to log in, adjust settings (temperature, humidity), activate features (damped seat), and access entertainment (jokes).
    - Owner Control: Separate login for the owner to manage system settings and customer data.
- Token-Based System:
    - Token Allocation: Passengers receive tokens upon their first login for free WiFi and system features.
    - Token Purchase: Passengers can purchase additional tokens with Ether at a predefined exchange rate.

- Security and Access Control:
    - Customer Authentication: Log in with name and travel ID.
    - Owner Privileges: Access to administrative functions including customer management and system settings.

Now we will explain what we have done in solidify code.

3.1 Contract Declaration and State Variables

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract FerryService {
  // Struct to represent a customer
  struct Customer {
    string fullName;
    uint256 travelId;
    bool firstTime;
    uint256 tokenBalance;
    bool dampedSeatActivated;
    int temperature;
    uint256 humidity;
  }
  // Mapping to store customers
  mapping(address => Customer) private customers;
  // Mapping to track logged-in status
  mapping(address => bool) private loggedIn;
  // Address of the owner
  address public owner;
  // Address of the company's account to receive Ether
  address payable private companyAccount;
  // Free WiFi code
  string private freeWifiCode = "FREE_WIFI_CODE";
  // Exchange rate: 10 tokens per 1 Ether
  uint256 private tokensPerEther = 10;
  // Initial values for temperature and humidity
  int private initialTemperature = 25;
  uint256 private initialHumidity = 50;
  // Array of jokes for entertainment
  string[] private jokes = [
    "Why don't scientists trust atoms? Because they make up everything!",
    "Parallel lines have so much in common. It’s a shame they’ll never meet.",
    "I told my wife she should embrace her mistakes. She gave me a hug.",
    "I'm reading a book on anti-gravity. It's impossible to put down!"
  ];
```

Explanation of the above code:

- Contract Declaration and SPDX License: The contract is named FerryService and is licensed under the MIT license.
- Struct Definition: Customer struct defines the structure of each customer's data, including their name, travel ID, token balance, and settings.
- Mappings: customers and logged in mappings are used to store customer data and track their logged-in status.
- State Variables: Various state variables are declared to store owner address, company account address, free Wi-Fi code, token exchange rate, initial temperature and humidity values, and an array of jokes for entertainment.

3.2 Constructor

```
// Constructor to initialize the contract
constructor(address _owner, address payable _companyAccount) {
    owner = _owner;
    companyAccount = _companyAccount;
    // Pre-loading customer database with initial values
    customers[0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2] = Customer("Milad", 69039, true, 10, false, initialTemperature, initialHumidity);
    customers[0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2] = Customer("Urfan", 39069, true, 10, false, initialTemperature, initialHumidity);
}
```

Here we are initializing the contract with the owner's address and the company's account address. It also pre-loads the customer database with initial values for Milad and Urfan, setting them as first-time customers with 10 tokens, damped seat deactivated, and initial temperature and humidity values.

3.3 Modifiers

```
// Modifier: Check if customer is logged in
modifier onlyLoggedIn(address _customer) {
    require(loggedIn[_customer], "You must be logged in to perform this action.");
    _;
}
// Modifier: Check if caller is the owner
modifier onlyOwner() {
    require(msg.sender == owner, "Only the owner can perform this action.");
```

```
    _;
  }
```

onlyLoggedIn Modifier checks if the customer is logged in before allowing the function to execute. onlyOwner ensures that only the contract owner can perform specific actions, such as managing customers or settings.

3.4 Customer Login and Logout

```
  // Function: Customer login
  function login(string memory _fullName, uint256 _travelId) public returns (string memory) {
    require(!loggedIn[msg.sender], "You are already logged in.");

    // Check if owner is logging in
    if (msg.sender == owner) {
      require(
        keccak256(abi.encodePacked("Omer Farooq")) == keccak256(abi.encodePacked(_fullName)) &&
        _travelId == 637042012,
        "Invalid name or travel ID."
      );
      loggedIn[msg.sender] = true;
      return "Welcome owner Omer Farooq!";
    } else {
      Customer storage customer = customers[msg.sender];
      require(
        keccak256(abi.encodePacked(customer.fullName)) == keccak256(abi.encodePacked(_fullName)) &&
        customer.travelId == _travelId,
        "Invalid name or travel ID."
      );
      loggedIn[msg.sender] = true;

      if (customer.firstTime) {
        customer.firstTime = false;
        customer.tokenBalance += 10;
        return string(abi.encodePacked("Welcome ", customer.fullName, "! Since it's your first time travelling
with us, we are giving you free WiFi (", freeWifiCode, ") and 10 tokens."));
      } else {
        return string(abi.encodePacked("Welcome back, ", customer.fullName, "!"));
      }
    }
  }
  // Function: Customer logout
  function logout() public onlyLoggedIn(msg.sender) {
```

```
    loggedIn[msg.sender] = false;
}
```

The first function, Login, allows customers to log in using their name and travel ID. Owners can also log in using their specific name and travel ID. First-time customers receive free Wi-Fi and tokens. Returns a personalized greeting based on whether it's their first login or a return visit. The second function Logout allows logged-in customers to log out, updating their logged-in status to false.

3.5 Customer Token Management and Recharge

```
// Function: Get customer's token balance
function getTokenBalance() public view onlyLoggedIn(msg.sender) returns (uint256) {
    return customers[msg.sender].tokenBalance;
}
// Function: Recharge tokens by purchasing with Ether
function rechargeTokens(uint256 _amount) public payable onlyLoggedIn {
    require(msg.value == _amount * 1 ether, "Incorrect amount of Ether sent.");
    customers[msg.sender].tokenBalance += _amount;
}
```

The first function Get Token Balance Function returns the token balance of the logged-in customer. The second function Recharge Tokens Function allows customers to recharge their tokens by purchasing them with Ether. Ensures that the amount of Ether sent matches the amount of tokens requested.

3.6 Customer Environment Control

```
// Function: Set temperature
function setTemperature(int _temperature) public onlyLoggedIn returns (string memory) {
    require(_temperature >= 16 && _temperature <= 30, "Temperature must be between 16 and 30 Celsius.");
    customers[msg.sender].temperature = _temperature;
    customers[msg.sender].tokenBalance -= 2;
    return "Temperature set successfully.";
}
// Function: Set humidity
function setHumidity(uint256 _humidity) public onlyLoggedIn returns (string memory) {
    require(_humidity >= 30 && _humidity <= 60, "Humidity must be between 30 and 60.");
    customers[msg.sender].humidity = _humidity;
    customers[msg.sender].tokenBalance -= 2;
```

```
    return "Humidity set successfully.";
}
```

The first function Set Temperature Function allows customers to set the cabin temperature within a specified range. Deducts 2 tokens from the customer's balance for each adjustment. The second function Set Humidity Function allows customers to set the cabin humidity within a specified range. Deducts 2 tokens from the customer's balance for each adjustment.

3.7 Damped Seat Feature

```
// Function: Activate damped seat feature
function activateDampedSeat() public onlyLoggedIn returns (string memory) {
    require(!customers[msg.sender].dampedSeatActivated, "Damped seat feature is already activated.");
    customers[msg.sender].dampedSeatActivated = true;
    customers[msg.sender].tokenBalance -= 5;
    return "Damped seat feature activated successfully.";
}
```

This function allows customers to activate the damped seat feature by deducting 5 tokens from their balance. Ensures that the feature is only activated once.

3.8 Owner Functions

```
// Function: Add a new customer (only owner)
function addCustomer(address _address, string memory _fullName, uint256 _travelId) public onlyOwner {
    customers[_address] = Customer(_fullName, _travelId, true, 0, false, initialTemperature, initialHumidity);
}
// Function: Remove a customer (only owner)
function removeCustomer(address _address) public onlyOwner {
    delete customers[_address];
}
```

The first function allows the owner to add a new customer to the system, initializing their details. The second function allows the owner to remove a customer from the system.

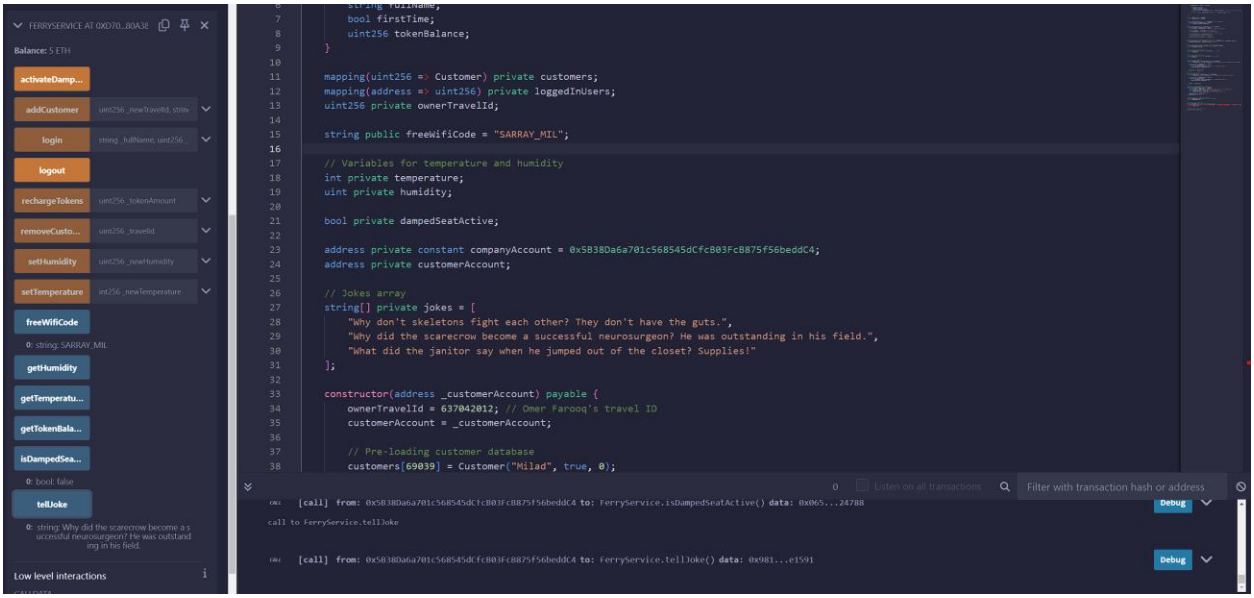
3.9 Jokes and Entertainment

```
// Function: Tell a joke
function tellJoke() public onlyLoggedIn returns (string memory) {
    uint256 index = uint256(keccak256(abi.encodePacked(block.timestamp, block.difficulty, msg.sender))) %
jokes.length;
    return jokes[index];
}
```

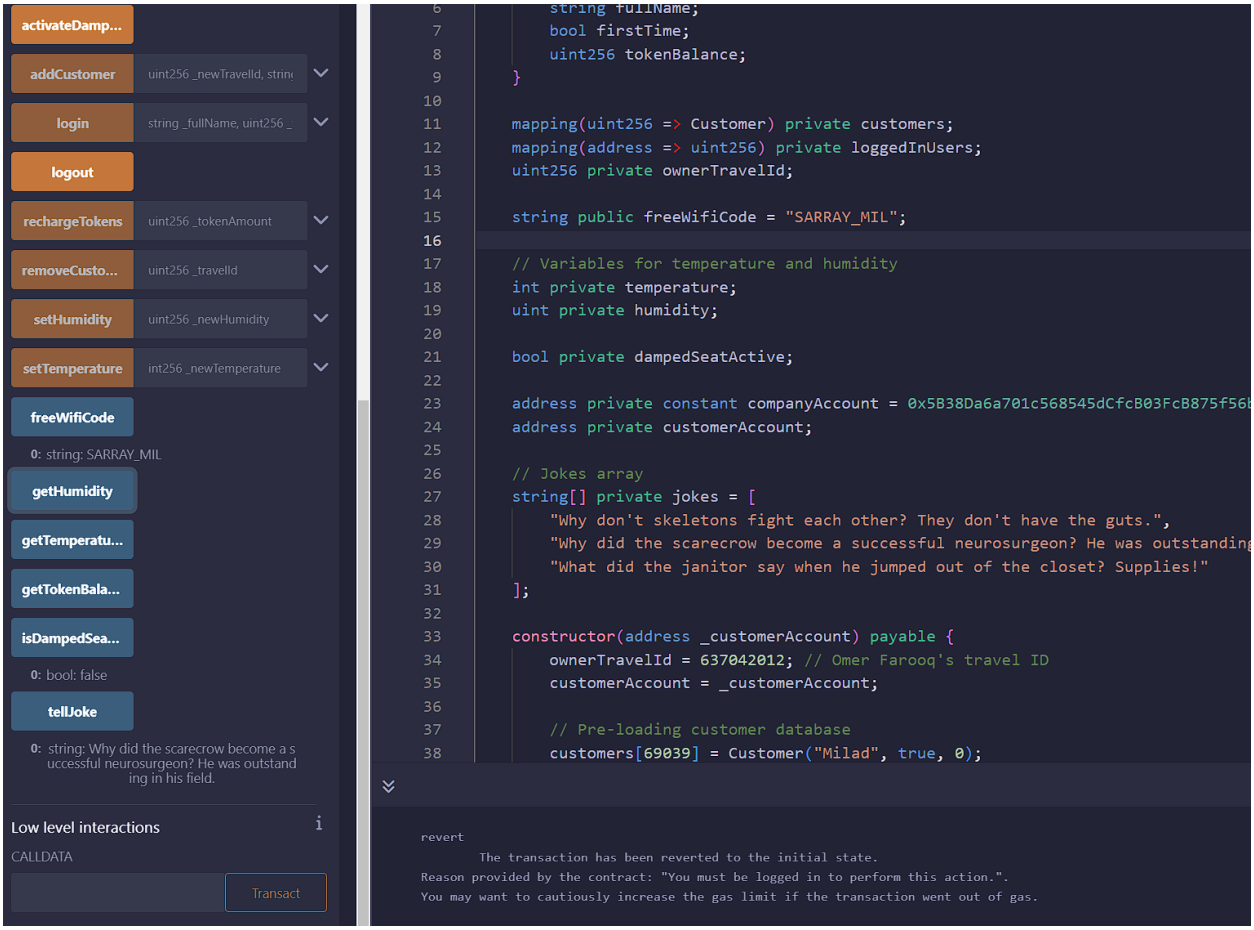
This function returns a random joke from a predefined array of jokes. Only logged-in customers can access this function for entertainment.

4 Results

Check the video attached (named smart contract working) to see the full working of this smart contract, but below are the screen shots of some important results.

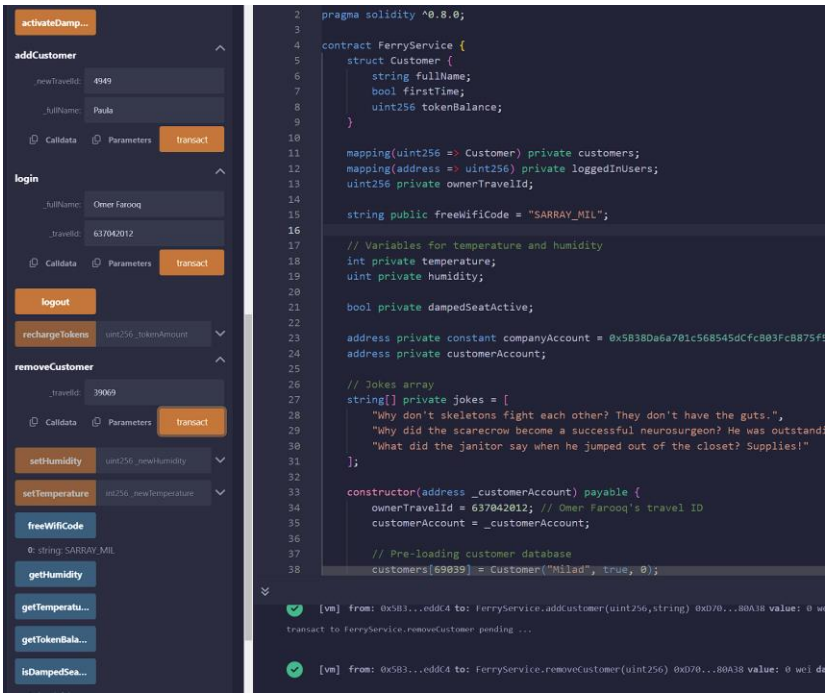


We are getting the free wi-fi code, a function name isDampedSeat returning value false, telling us that the seat is not damped yet. tellJoke function telling us some randomly chosen joke.

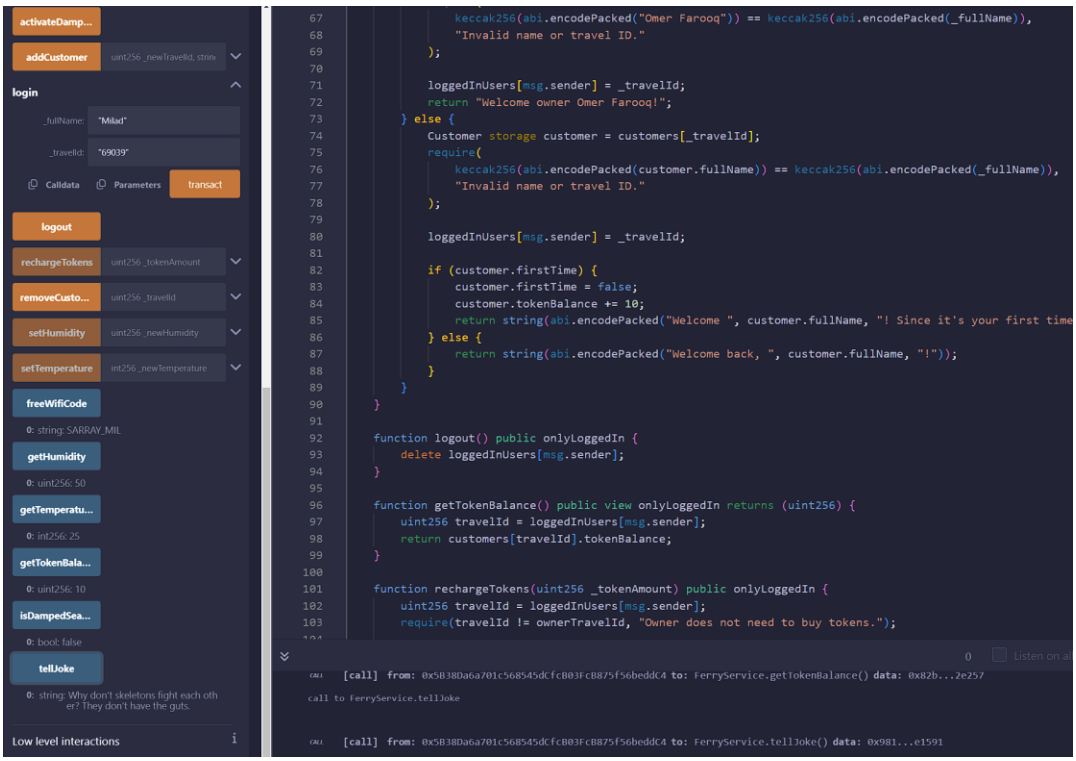


I can’t call other functions, as they will return error that you must be logged in to do this (check above image)

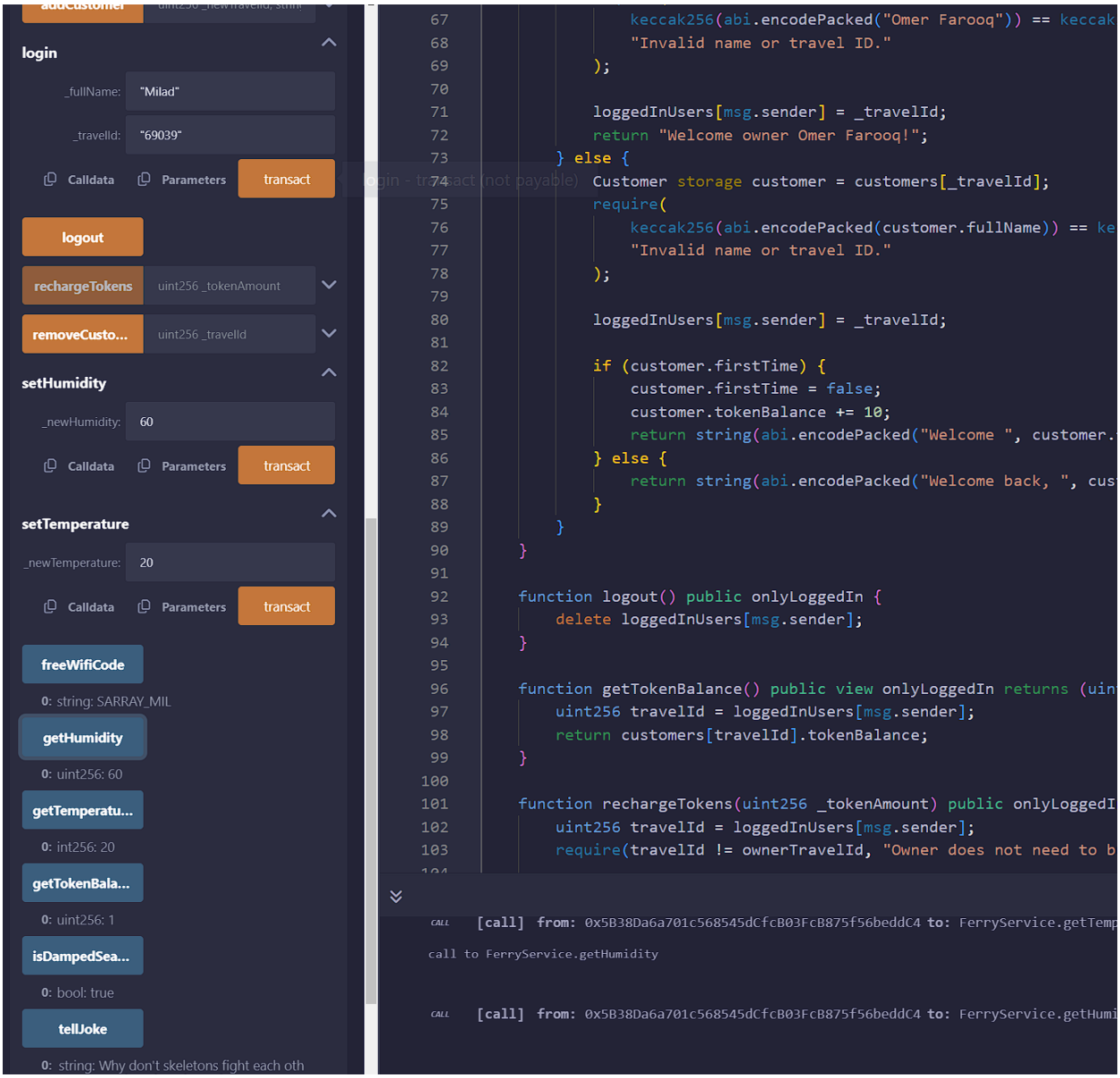
Firstly, we will login as the owner. Using the owner’s privileges, we will add a new customer. And remove already defined customer Urfan from database.



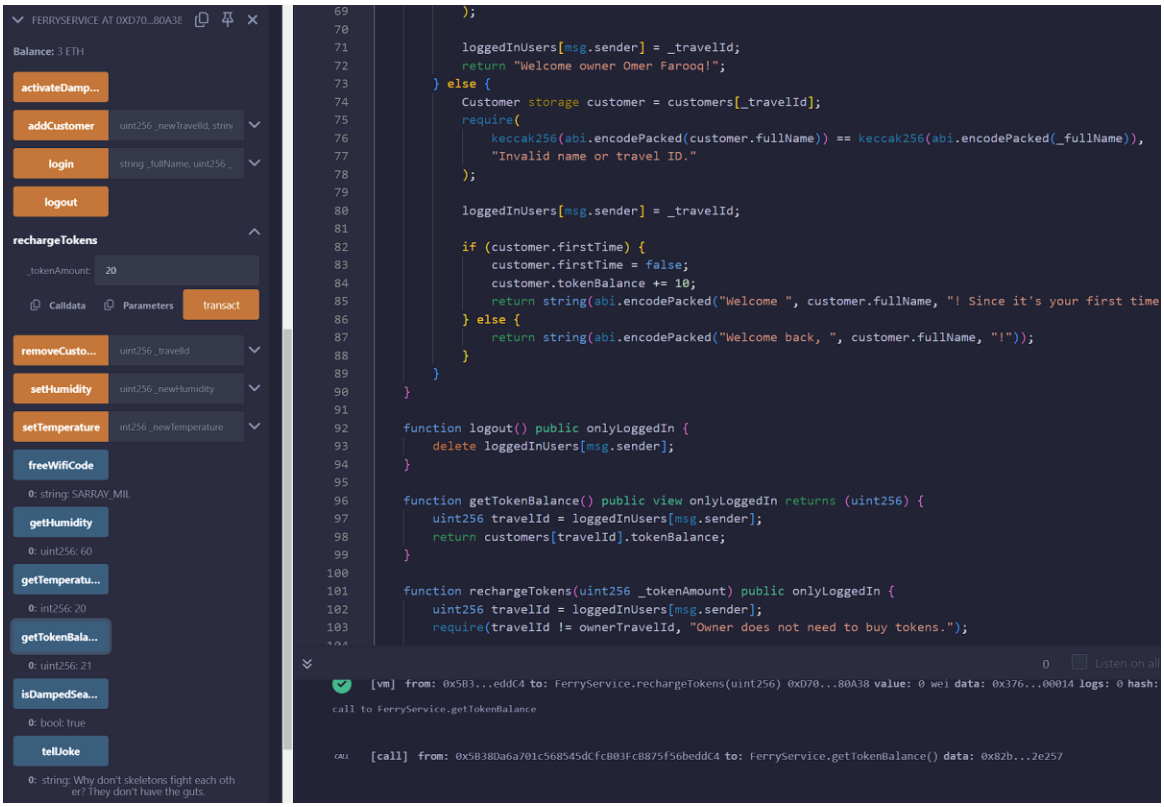
Now, we will login as customer Milad and check initial values for temperature, humidity, token balance. We have 10 tokens as the ferry company is giving 10 free tokens on first time logging in.



Now, we will change the temperature and humidity values. Each one will cost 2 tokens. Then we will also activate the seat damping which will cost 5 tokens. After this new function values can be seen in the picture below.



Finally, if the customer wants, he can buy more tokens. For one ETH the customer can buy 10 tokens. In the picture below the customer has bought 20 tokens worth 2 ETH. His new ETH balance and token balance have changed, as can be seen.



5 Conclusion and Lesson Learned

The Intelligent System for Passenger Seats in Vessels project successfully integrates modern technology to enhance passenger comfort and convenience. By utilizing blockchain technology for secure and transparent transactions, along with IoT sensors to monitor and adjust environmental conditions, we created a comprehensive solution that addresses passenger needs during travel. The smart contract, deployed on the Ethereum blockchain, allows for seamless user interactions, including the management of tokens for services and entertainment, ensuring an enjoyable and controlled travel experience.

Throughout the project, we learned the importance of integrating blockchain for secure and transparent transactions, and the challenges involved in smart contract development on Ethereum using Solidity. We gained insights into robust user authentication and management, ensuring differentiated access for customers and owners. Additionally, the integration of IoT sensors for environmental control highlighted the potential of IoT in enhancing passenger comfort. Finally, we realized the significance of user experience enhancements, such as adding entertainment features, to improve overall customer satisfaction.