Configuration Manual

MSc Research Project

FinTech

TING YI LIU
Student ID: X21177899

School of Computing

National College of Ireland
Supervisor: Theo Mendonca

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	TING YI LIU		
Student ID:	X21177899		
Programme:	MSC FINTECH	**Year:**	2022/23
Module:	MSC Research Project		
Supervisor:	Theo Mendonca		
Submission Due Date:	August 14, 2023		
Project Title:	Innovative User Incentive Models in Blockchain-Based Deckles Bike Sharing System		
Word Count:	1757	**Page Count** 15	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	TING YI LIU
Date:	14/08/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ting Yi Liu
x21177899

1. Introduction

This user configuration manual outlines in detail how to set up and configure the Bike Sharing Smart Contract an Ethereum-based decentralized application (Dapp). Through a reward and penalty structure, the smart contract promises to change the bike sharing sector by motivating responsible user behavior and boosting active involvement.

2. System Requirements

2.1. Hardware

- MacBook Pro (Retina, 13-inch, Early 2015)
- Mac OS Big Sur version 11.7
- Memory - 8 GB 1867 MHz DDR3
- Storage 128 GB SSD

2.2. Software

- Install Ganache: Download and install Ganache from the official website.
- Remix IDE: Open your web browser and go to the Remix IDE website (https://remix.ethereum.org/). Remix is a browser-based IDE, so no additional installation is needed.

3. Configuration Steps

3.1. Step 1 - Launch Ganache

After installation, launch the Ganache program. It will create a local blockchain network with predefined accounts and 100 test Ether each for transactions.

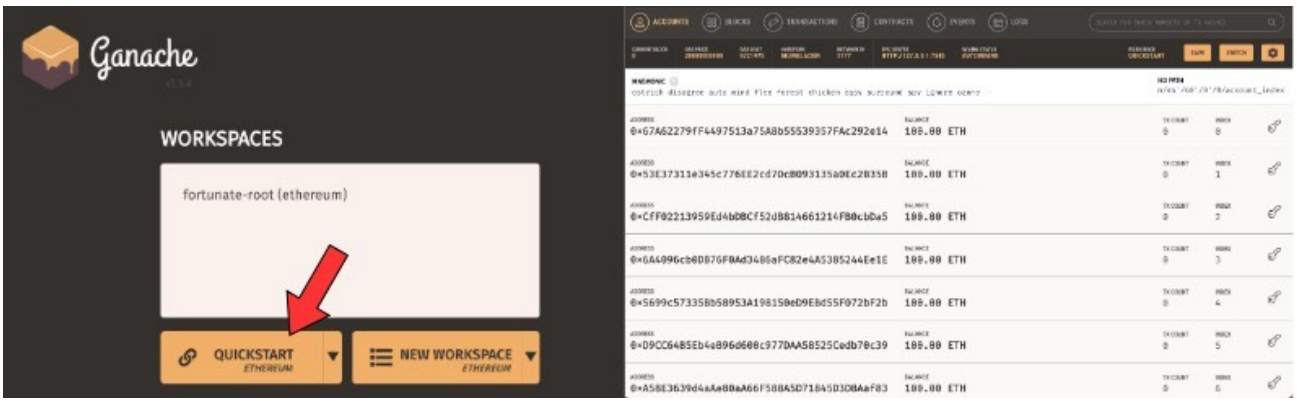


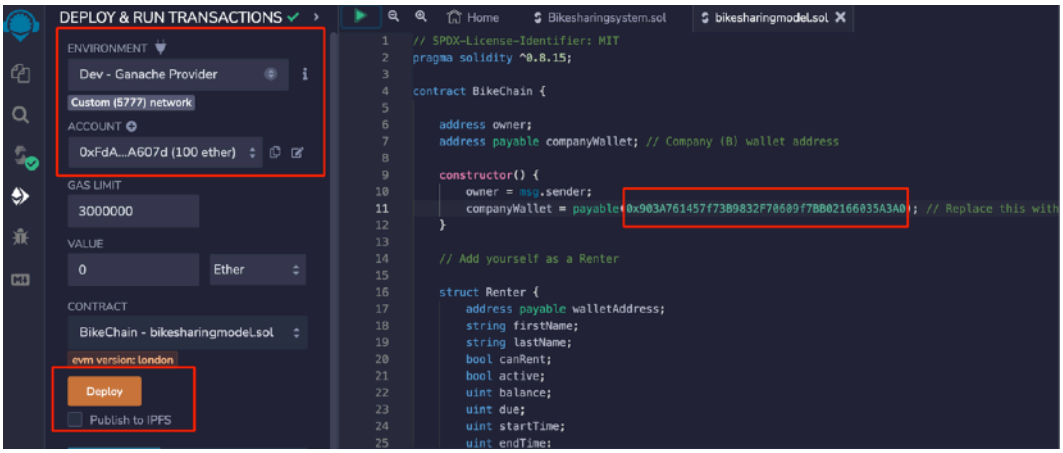
Figure 6: Ganache set up

3.2. Step 2 - Deploy the Smart Contract

Compile and deploy the Bike Sharing Smart Contract to the Ganache network using the Remix IDE development environment. Go to the “Deploy & Run Transactions” tab on the left-hand side. Under the “Environment” section, select “Dev - Ganache Provider” as the environment and enter the connection URL for Ganache (http://127.0.0.1:7545).

Connect Remx to Ganache

In Remix IDE, go to the "Solidity Compiler" tab and select the version of Solidity used in your smart contract. Then, navigate to the "Deploy & Run Transactions" tab again. Click "Deploy" to deploy the contract to the Ganache network.



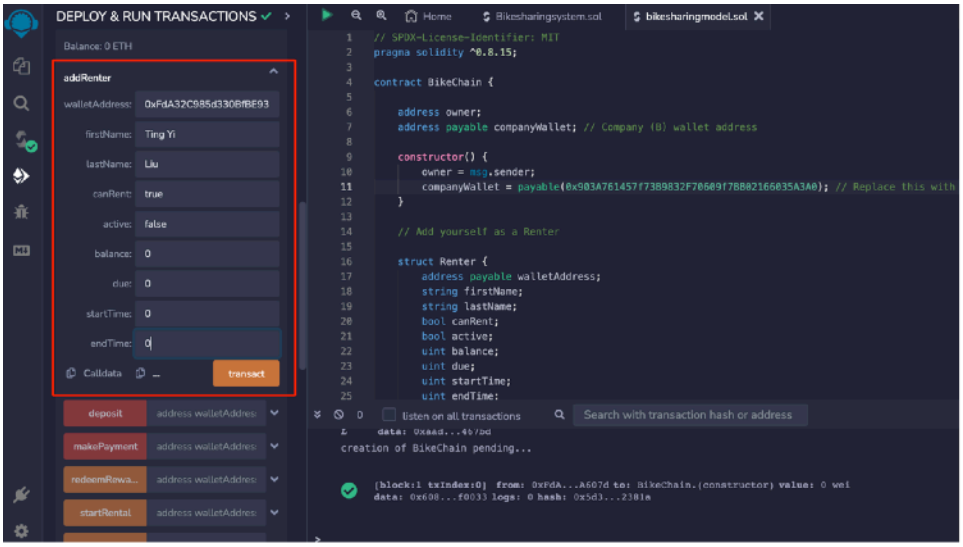
Deploy the smart contract

3.3. Step 3 - Interact with the Smart Contract

Users can start interacting with the smart contract using MetaMask to complete transactions and run functions once it has been deployed. Through the smart contract's functionalities, users may do operations such as user registration, starting and ending rentals, making payments, and redeeming rewards.

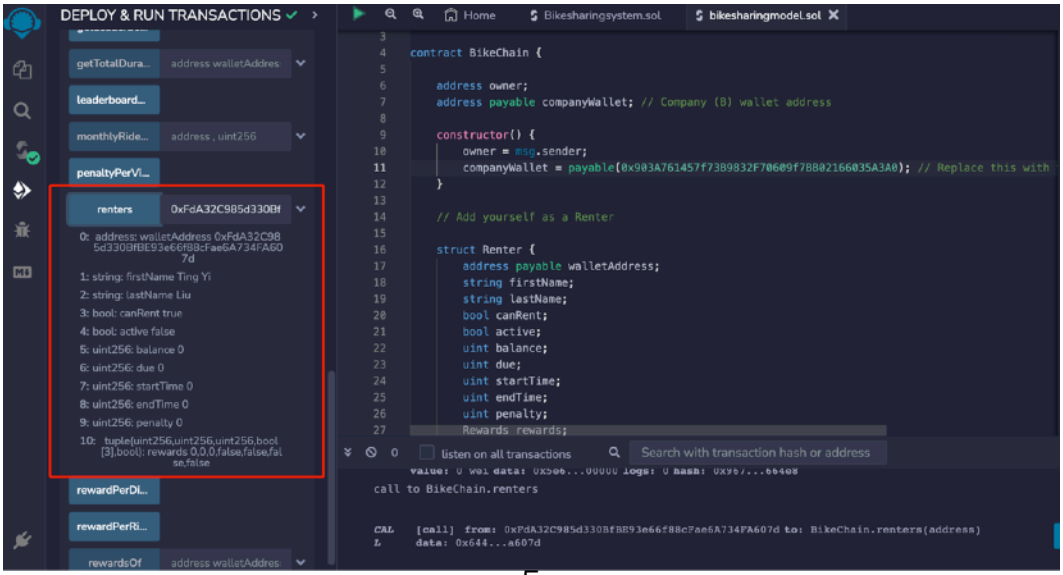
3.3.1. Add a Renter

To add a new renter, fill in the required parameters (walletAddress, firstName, lastName, canRent, active, balance, due, startTime, endTime) and click on the function button.



Add a Renter

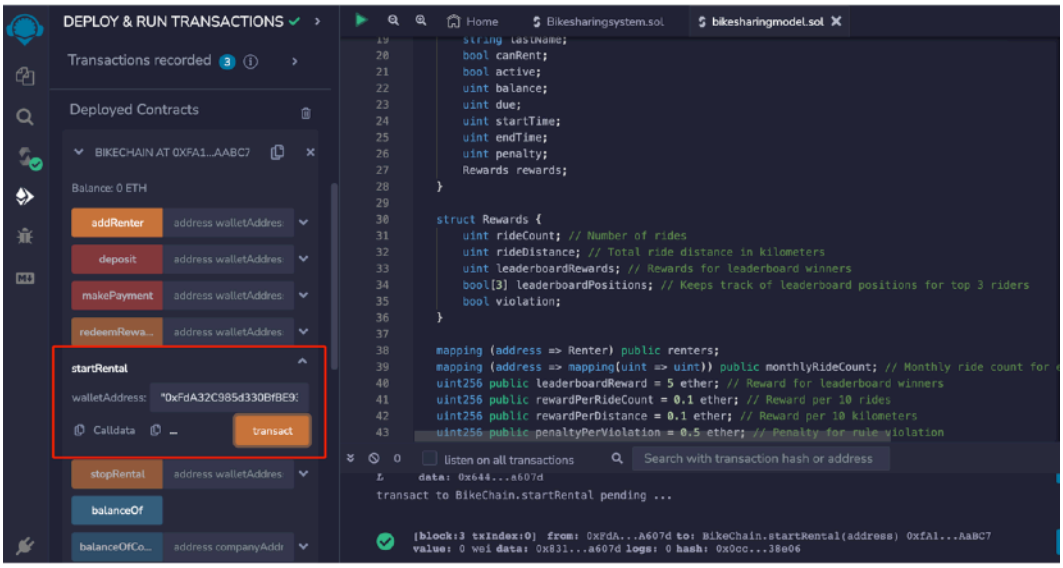
You will get the renter's information stored in the renters mapping. This includes the firstName, lastName, canRent, active, balance, due, startTime, endTime, penalty, and the Rewards struct associated with that walletAddress. The renter's name is "Ting Yi Liu," and they have the ability to rent a bike (canRent: true). However, currently, they are not actively renting a bike (active: false) and have no pending balance (due: 0). There are no previous rental records, so the startTime, endTime, and penalty are all set to 0. Additionally, the renter has not completed any rides or earned any rewards yet (rewards: rideCount: 0, rideDistance: 0, leaderboardRewards: 0, leaderboardPositions: [false, false, false]).



Retrieve the details

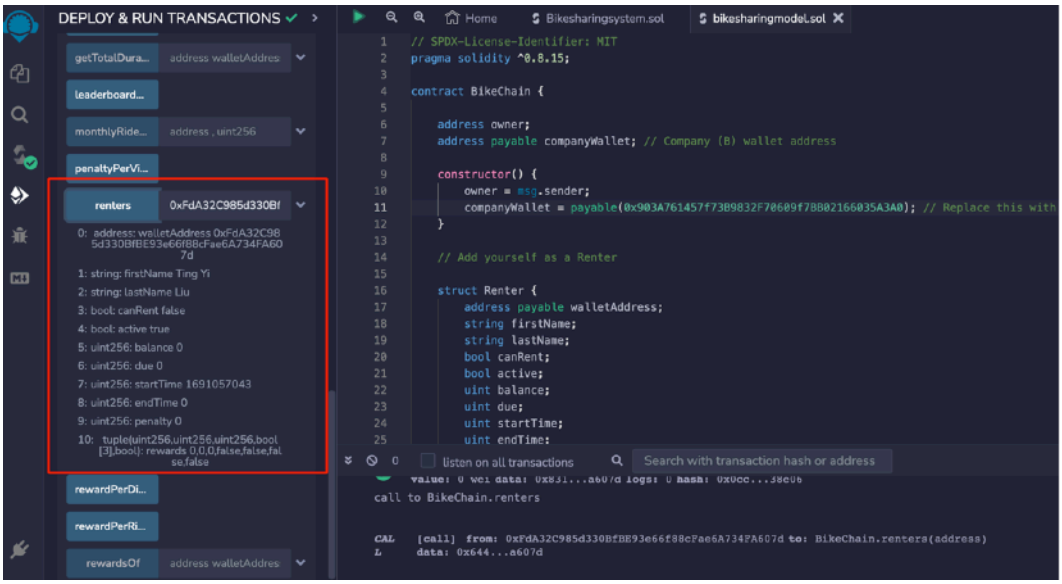
3.3.2.Start Renting a Bike

To start renting a bike, provide the renter's wallet address as a parameter in the "startRental" function and click the function button.



Start Renting a Bike

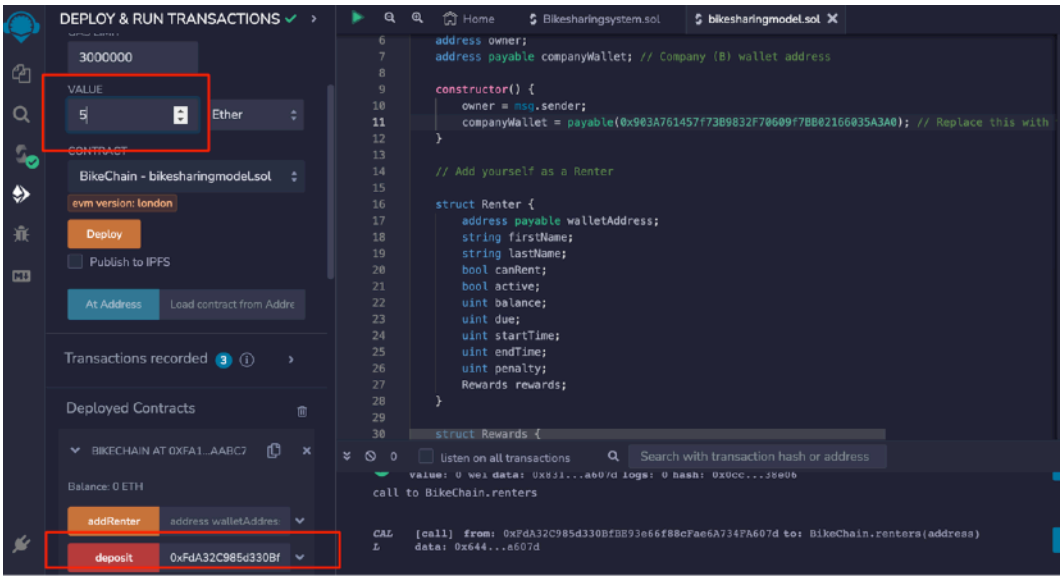
After start a rental. we can check the information. The renter currently cannot rent a bike (canRent: false). They have an ongoing rental session (active: true) that started at startTime: 1691057043 (Unix timestamp). The renter has no pending balance (due: 0), no penalty (penalty: 0), and has not completed any rides or earned rewards yet (rewards: rideCount: 0, rideDistance: 0, leaderboardRewards: 0, leaderboardPositions: [false, false, false]).



“Renters” information after start renting a bike

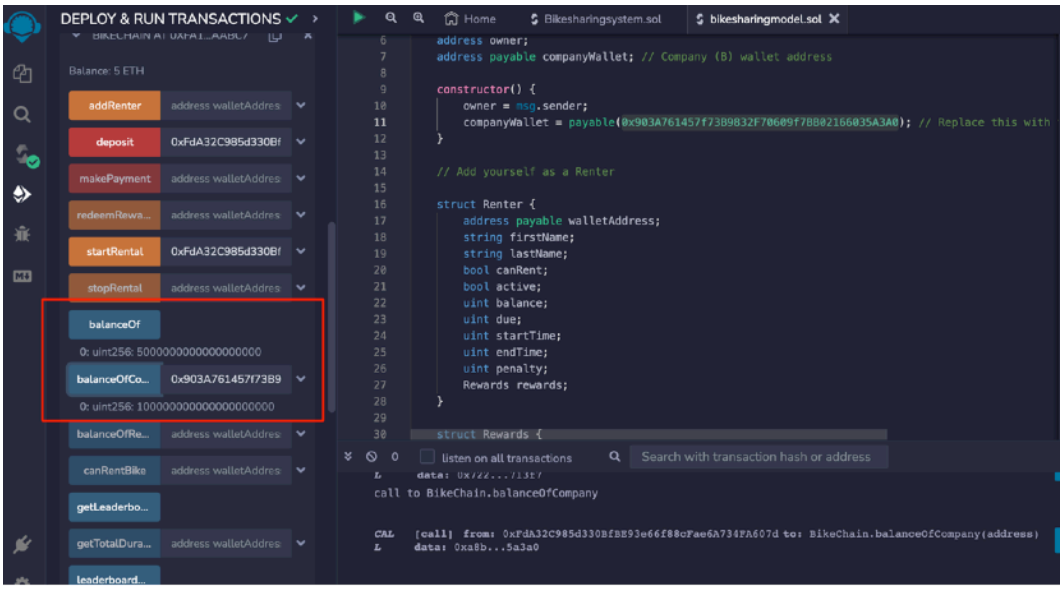
3.3.3.Deposit

The deposit function allows users (renters) to add funds to their account balance in the smart contract by sending Ether with a transaction. The deposited amount is added to the user's existing balance, which can be used to cover pending dues or future bike rentals. As shown below, we added 5 ETH to deposit.



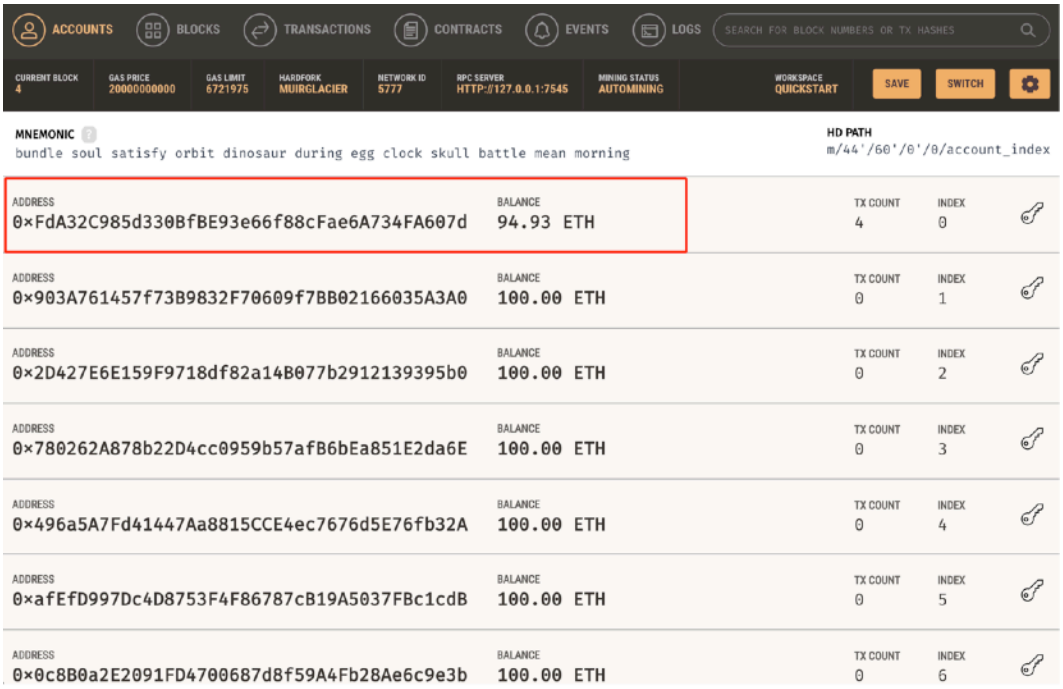
Deposit

The balanceOfRenter function allows the smart contract to retrieve and display the balance of a specific renter's wallet address within the bike sharing system. And the balanceOfCompany function retrieves the balance of the company's wallet address in the smart contract. As figure below, we can see balance of renter is 5 ETH and balance of company is 100 ETH.

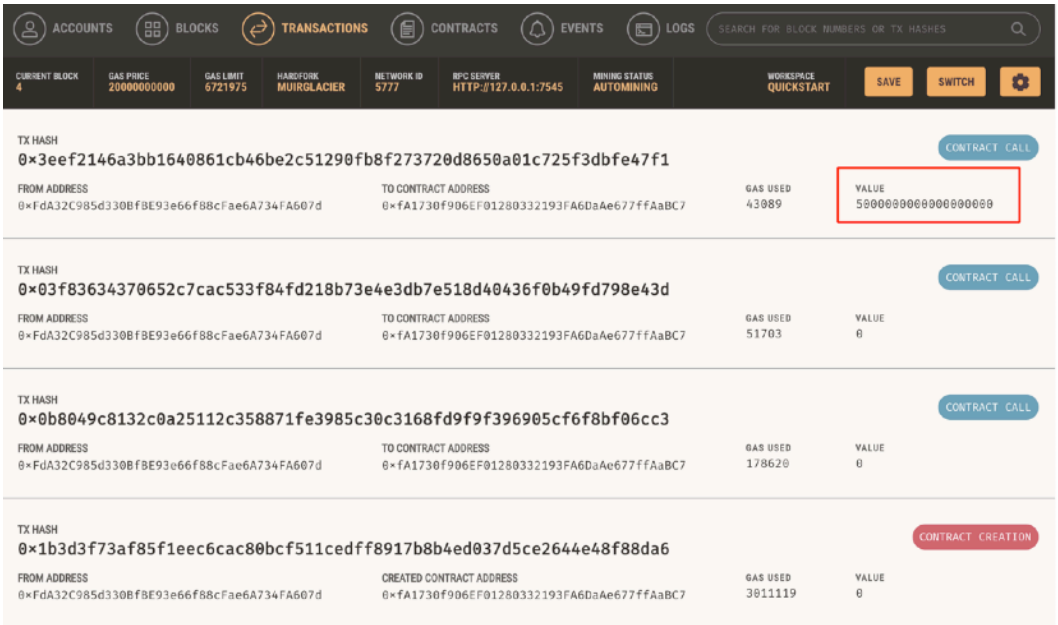


Balance

In Ganache, we can see the account balance dropped 5 ETH.



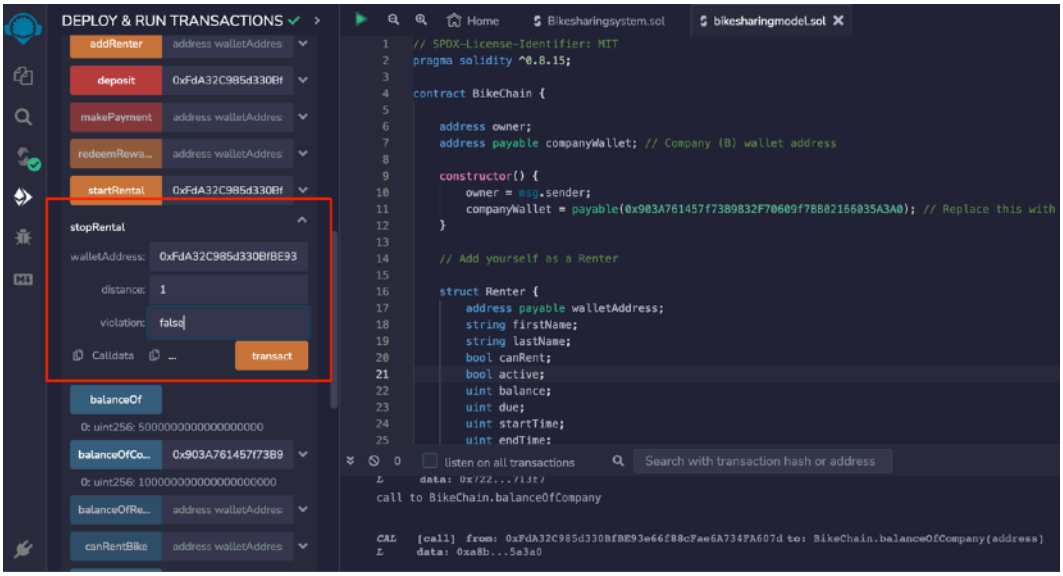
Ganache account balance



Ganache transaction detail

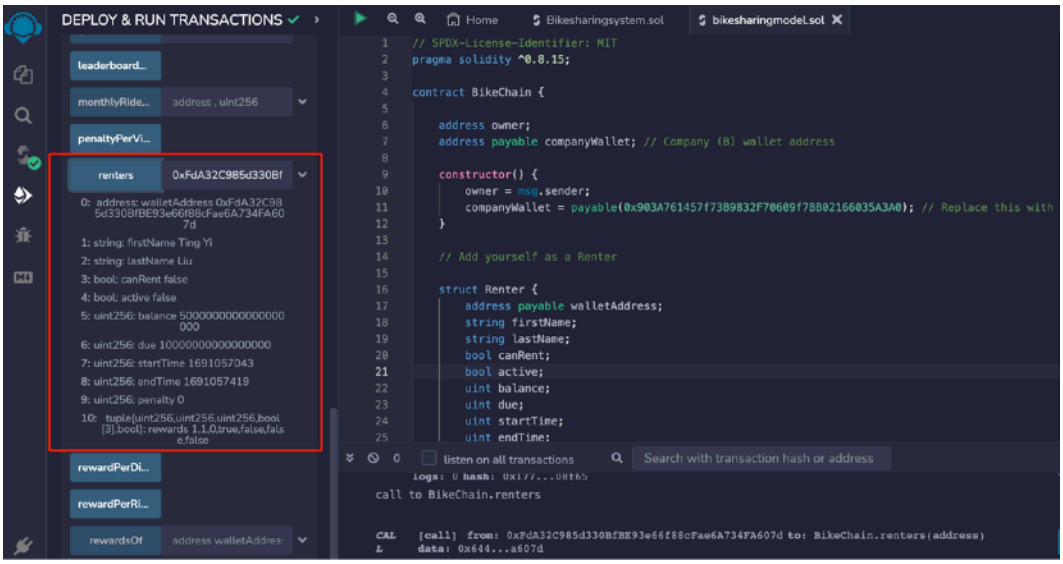
3.3.4.Stop Renting a Bike

To stop renting a bike, provide the renter's wallet address, the distance traveled, and whether there was a violation as parameters in the "stopRental" function. Then, click the function button.



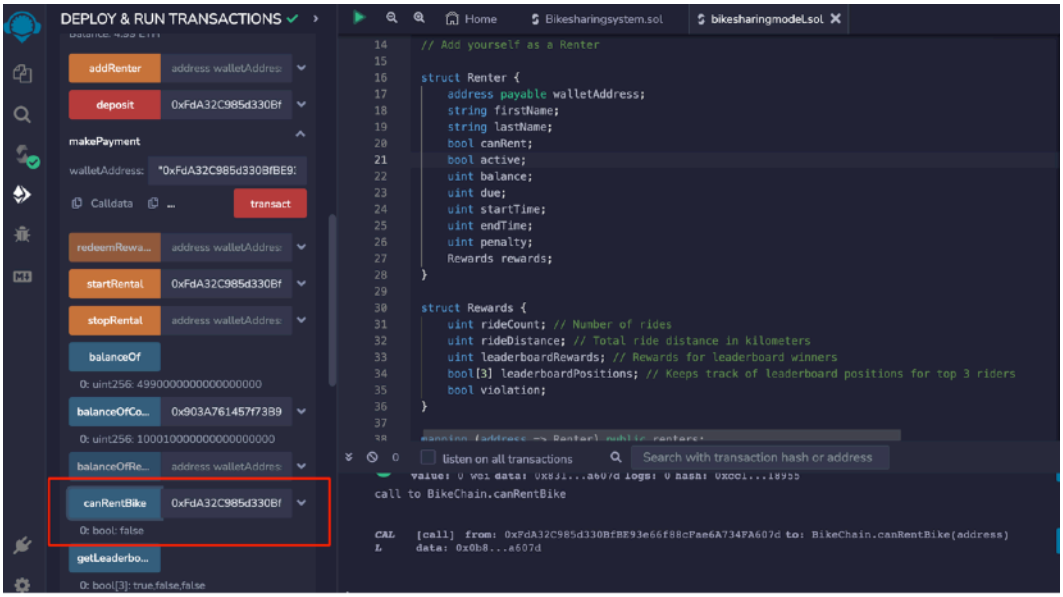
Stop renting a bike

The renter's name is Ting Yi Liu. Currently, the renter cannot rent a bike (canRent is false) and is not actively using a bike (active is false). The renter's account balance is 5 Ether. The renter has a pending due amount of 0.01 Ether. The startTime indicates that the renter started a bike rental at a specific timestamp, while endTime shows the end time of the rental. The penalty value is currently 0, indicating no penalty has been applied. The rewards tuple shows that the renter has completed 1 ride (rideCount: 1) and covered a total ride distance of 1 kilometer (rideDistance: 1). The renter has earned 0 leaderboard rewards and holds the first position on the leaderboard (leaderboardPositions: [true, false, false]). Additionally, the renter has no violated the rules (violation: false).



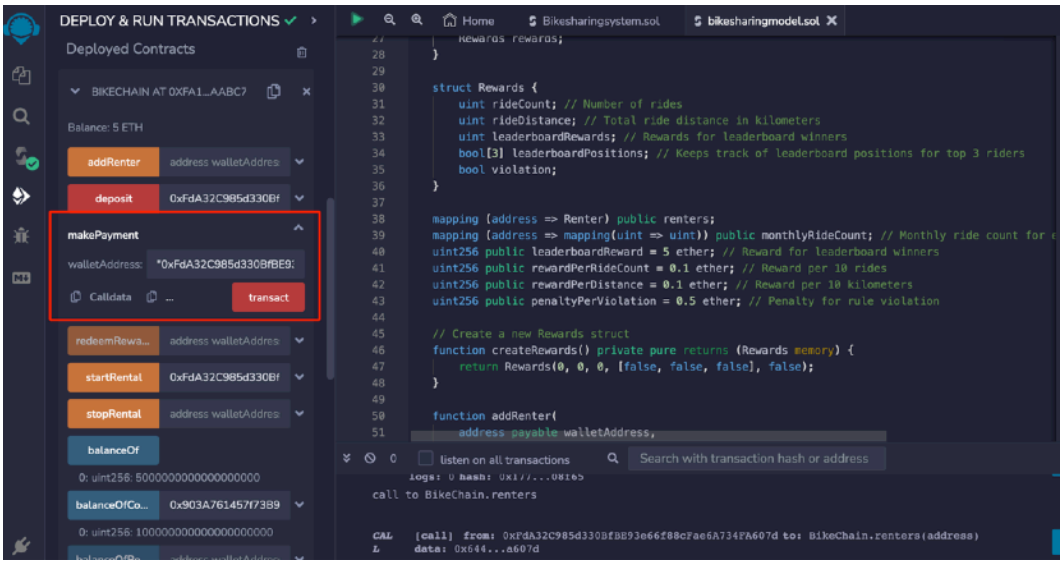
“renters” information after stop renting a bike

canRentBike function is false.



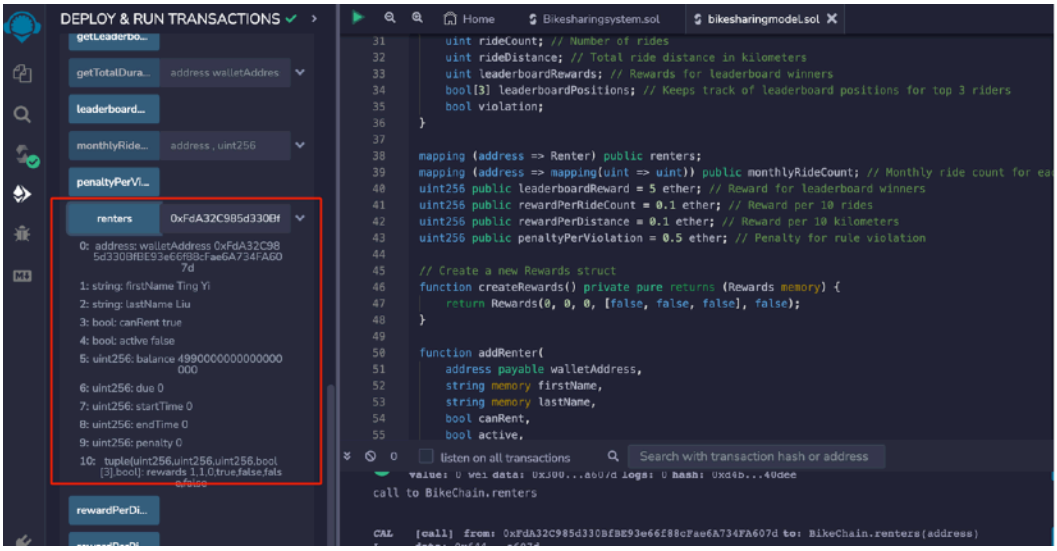
canRentBike function

Make payment



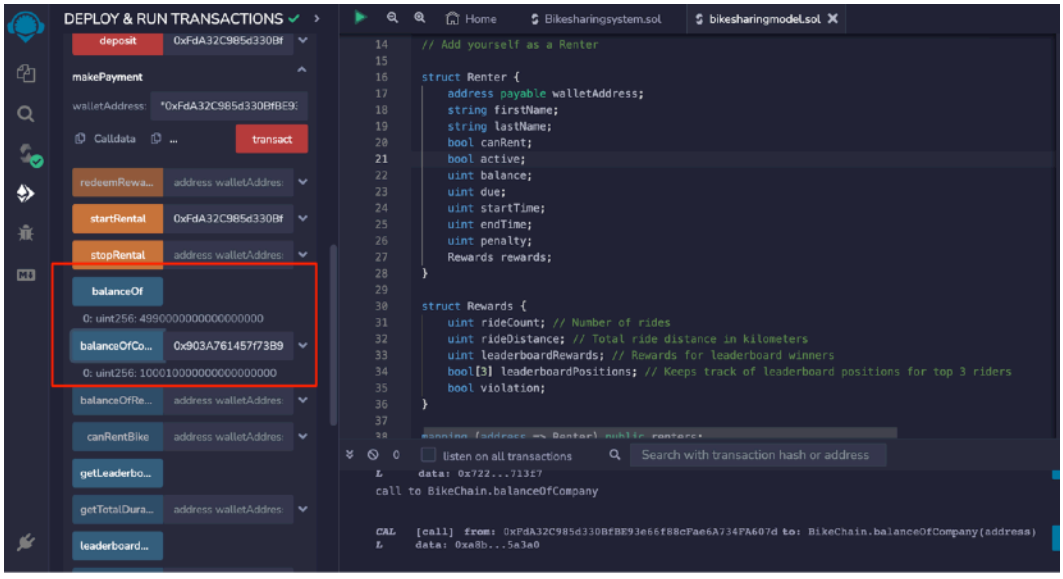
make payment function

after making payment

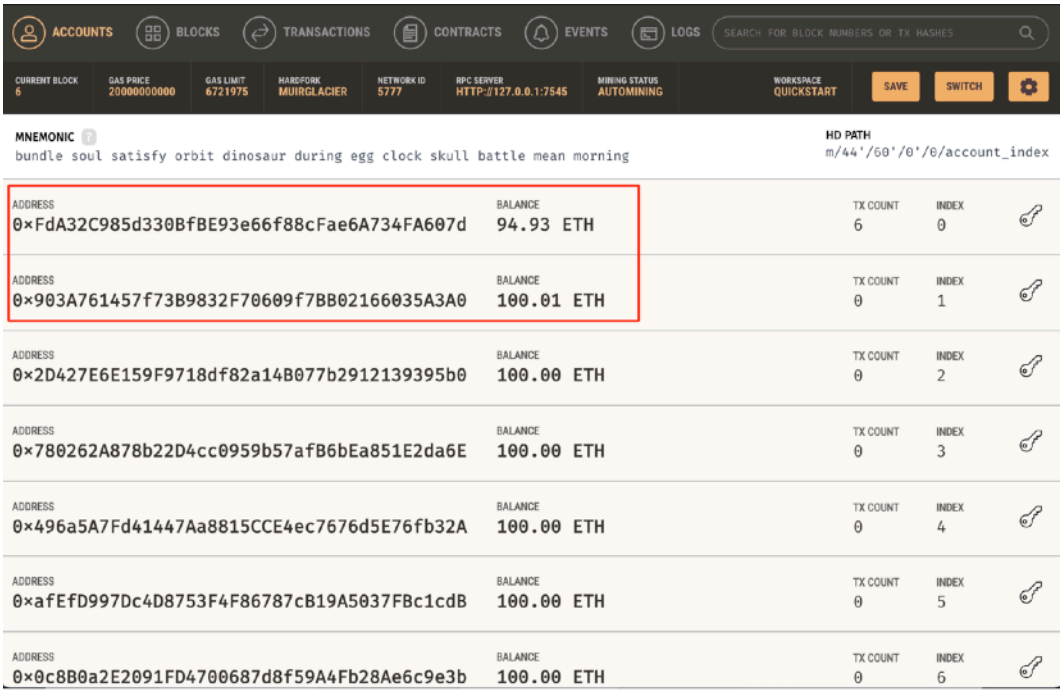


“renters” information after making a payment

The balance of renter and company

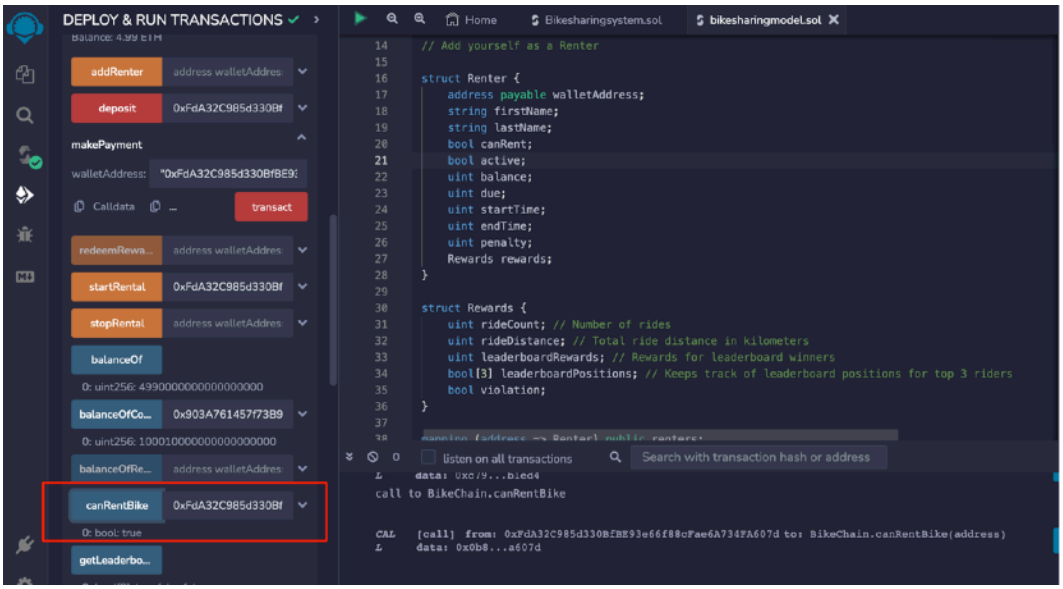


Balance of renter and company



Ganache account balance

can rent true



canRentBike function

4. Smart Contract Sodility Code:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;

contract BikeChain {

    address owner;
    address payable companyWallet; // Company (B) wallet address

    constructor() {
        owner = msg.sender;
         companyWallet = payable(0x903A761457f73B9832F70609f7BB02166035A3A0); // Replace this with
the actual company wallet address
    }

    // Add yourself as a Renter

    struct Renter {
        address payable walletAddress;
        string firstName;
        string lastName;
        bool canRent;
        bool active;
        uint balance;
        uint due;
        uint startTime;
        uint endTime;
        uint penalty;
        Rewards rewards;
    }

    struct Rewards {
        uint rideCount; // Number of rides
        uint rideDistance; // Total ride distance in kilometers
        uint leaderboardRewards; // Rewards for leaderboard winners
        bool[3] leaderboardPositions; // Keeps track of leaderboard positions for top 3 riders
        bool violation;
    }

    mapping (address => Renter) public renters;
      mapping (address => mapping(uint => uint)) public monthlyRideCount; // Monthly ride count for each
user
    uint256 public leaderboardReward = 5 ether; // Reward for leaderboard winners
    uint256 public rewardPerRideCount = 0.1 ether; // Reward per 10 rides
    uint256 public rewardPerDistance = 0.1 ether; // Reward per 10 kilometers
    uint256 public penaltyPerViolation = 0.5 ether; // Penalty for rule violation

    // Create a new Rewards struct
    function createRewards() private pure returns (Rewards memory) {
        return Rewards(0, 0, 0, [false, false, false], false);
    }

    function addRenter(
        address payable walletAddress,
        string memory firstName,
        string memory lastName,
        bool canRent,
        bool active,
        uint balance,
```

```
    uint due,
    uint startTime,
    uint endTime
    ) public {
    renters[walletAddress] = Renter(
        walletAddress,
        firstName,
        lastName,
        canRent,
        active,
        balance,
        due,
        startTime,
        endTime,
        0,
        createRewards() // Use the createRewards function to initialize the Rewards struct
        );
}

// Start rent a bike
function startRental(address walletAddress) public {
    require(renters[walletAddress].due == 0, "You have a pending balance.");
    require(renters[walletAddress].canRent == true, "You cannot rent at this time.");
    renters[walletAddress].active = true;
    renters[walletAddress].startTime = block.timestamp;
    renters[walletAddress].canRent = false;
}

// Stop rent a bike
function stopRental(address walletAddress, uint256 distance, bool violation) public {
    require(renters[walletAddress].active == true, "Please start rent a bike first.");
    renters[walletAddress].active = false;
    renters[walletAddress].endTime = block.timestamp;
    setDue(walletAddress);
    updateRewards(walletAddress, distance, violation); // Call updateRewards function when return bike
    updateMonthlyRideCount(walletAddress); // Update monthly ride count
    updateLeaderboard(walletAddress); // Update leaderboard positions
    updatePenaltyViolation(walletAddress, violation); // Update penalty violation
}

// Get total duration of bike use
function renterTimespan(uint startTime, uint endTime) internal pure returns(uint) {
    return endTime - startTime;
}

function getTotalDuration(address walletAddress) public view returns(uint) {
    require(renters[walletAddress].active == false, "Bike is currently rented.");
    uint timespan = renterTimespan(renters[walletAddress].startTime, renters[walletAddress].endTime);
    uint timespanInMinutes = timespan / 60;
    return timespanInMinutes;
}

// Get Contract balance
function balanceOf() view public returns(uint) {
    return address(this).balance;
}

// Get Renter's balance
function balanceOfRenter(address walletAddress) public view returns(uint) {
    return renters[walletAddress].balance;
```

```
    }

    // Get the balance of the company's wallet
    function balanceOfCompany(address companyAddress) public view returns (uint) {
        return companyAddress.balance;
    }

    // Set Due amount
    function setDue(address walletAddress) internal {
        uint timespanMinutes = getTotalDuration(walletAddress);
        uint fiveMinuteIncrements = timespanMinutes / 5;
        renters[walletAddress].due = fiveMinuteIncrements * 0.01 ether;
    }

    function canRentBike(address walletAddress) public view returns(bool) {
        return renters[walletAddress].canRent;
    }

    // Deposit
    function deposit(address walletAddress) payable public {
        renters[walletAddress].balance += msg.value;
    }

    // Make Payment
    function makePayment(address walletAddress) payable public {
        require(renters[walletAddress].due > 0, "You do not have anything due at this time.");
         require(renters[walletAddress].balance > msg.value, "You do not have enough funds to cover payment.
Please make a deposit.");

        // Transfer the due amount to the company wallet
        companyWallet.transfer(renters[walletAddress].due);

        renters[walletAddress].balance -= renters[walletAddress].due;
        renters[walletAddress].canRent = true;
        renters[walletAddress].due = 0;
        renters[walletAddress].startTime = 0;
        renters[walletAddress].endTime = 0;
    }

    // Update rewards
    function updateRewards(address walletAddress, uint256 distance, bool violation) internal {
        renters[walletAddress].rewards.rideCount++;
        renters[walletAddress].rewards.rideDistance += distance;
        uint256 rideCountRewards = (renters[walletAddress].rewards.rideCount / 10) * rewardPerRideCount;
        uint256 distanceRewards = (renters[walletAddress].rewards.rideDistance / 10) * rewardPerDistance;
        uint256 penalty = violation ? penaltyPerViolation : 0;
        renters[walletAddress].balance += rideCountRewards + distanceRewards - penalty;
    }

    // Update monthly ride count
    function updateMonthlyRideCount(address walletAddress) internal {
        uint256 currentMonth = (block.timestamp / 1 days) / 30; // Assuming 30 days per month
        monthlyRideCount[walletAddress][currentMonth]++;
    }

    // Update leaderboard positions
    function updateLeaderboard(address walletAddress) internal {
        bool[3] memory positions = renters[walletAddress].rewards.leaderboardPositions;
        uint256 currentMonth = (block.timestamp / 1 days) / 30; // Assuming 30 days per month
```

```
        // Check if the user has entered top 3 positions
        for (uint256 i = 0; i < 3; i++) {
            if (!positions[i]) {
                                if (i == 0 || (i > 0 && monthlyRideCount[walletAddress][currentMonth] >
monthlyRideCount[renters[walletAddress].walletAddress][currentMonth])) {
                for (uint256 j = 2; j > i; j--) {
                                    renters[renters[walletAddress].walletAddress].rewards.leaderboardPositions[j] =
renters[renters[walletAddress].walletAddress].rewards.leaderboardPositions[j - 1];
                }
                renters[renters[walletAddress].walletAddress].rewards.leaderboardPositions[i] = true;
                break;
            }
            }
        }

        // Check if the user has won a leaderboard reward
        if (positions[0] && renters[walletAddress].rewards.leaderboardRewards < leaderboardReward) {
            renters[walletAddress].rewards.leaderboardRewards = leaderboardReward;
            renters[walletAddress].balance += leaderboardReward;
        }
    }

    // Update penalty violation
    function updatePenaltyViolation(address walletAddress, bool violation) internal {
    if (violation) {
        renters[walletAddress].penalty += penaltyPerViolation;
        renters[walletAddress].balance -= penaltyPerViolation;
        }
    }

    // Get rewards balance
    function rewardsOf(address walletAddress) public view returns(uint) {
        Renter memory renter = renters[walletAddress];
        uint256 rideCountRewards = (renter.rewards.rideCount / 10) * rewardPerRideCount;
        uint256 distanceRewards = (renter.rewards.rideDistance / 10) * rewardPerDistance;
                        return rideCountRewards + distanceRewards + renter.rewards.leaderboardRewards -
penaltyPerViolation;
    }

    // Redeem rewards
    function redeemRewards(address walletAddress) public {
        uint rewards = rewardsOf(walletAddress);
        require(rewards > 0, "No rewards available");
        renters[walletAddress].balance = 0;
        payable(walletAddress).transfer(rewards);
        renters[walletAddress].canRent = true;
    }

    // Update reward per ride count
    // function updateRewardPerRideCount(uint256 _reward) public {
    //     require(msg.sender == owner, "Only owner can update reward per ride count.");
    //     rewardPerRideCount = _reward;
    // }

    // Update reward per distance
    // function updateRewardPerDistance(uint256 _reward) public {
    //     require(msg.sender == owner, "Only owner can update reward per distance.");
    //     rewardPerDistance = _reward;
    // }
```

```
    // Update leaderboard reward
    // function updateLeaderboardReward(uint256 _reward) public {
    //    require(msg.sender == owner, "Only owner can update leaderboard reward.");
    //    leaderboardReward = _reward;
    // }

    // Update penalty
    // function updatePenaltyPerViolation(uint256 penalty) public {
    //     require(msg.sender == owner, "Only owner can update penalty.");
    //     penaltyPerViolation = _penalty;
    // }

    // Get current leaderboard positions
    function getLeaderboardPositions() public view returns(bool[3] memory) {
        return renters[msg.sender].rewards.leaderboardPositions;
    }
}
```