//SPDX-License-Identifier: Unlicense

pragma solidity ^0.8.0;

// import "hardhat/console.sol";

contract CarRental {

// Define state variables

address payable rentalOwner;

address payable carOwner;

uint256 constant public price = 1e17; // 0.1 ETH

uint256 public totalCarNum = 0;

uint256 private totalUserNumber = 0;

mapping(address => uint256) public balances;

struct CustomerInfo {

bool isValidRental;

string customerName;

uint customerAge;

uint256 licenseID;

uint rentDuration;

bool hasReturned;

bool hasConfirmed;

}

struct CarInfo {

bool isAvailable;

string carLocation;

uint256 carID;

uint256 price;

string ownerName;

}

struct User {

string userName;

string password;

uint256 userID;

address payable userAddress;

bool isUserLoggedIn;

}

mapping(address => CustomerInfo) public rentals;

mapping(uint256 => CarInfo) public cars;

mapping(address => User) private users;

//mapping(uint256 => address) private caridtocarowner;

//CarInfo[] carInformation;

//CustomerInfo storage customer = rentals[rentalOwner];

constructor() {

rentalOwner = payable(msg.sender);

carOwner = payable(address(this));

}

// Account creation events

event accountCreated(

User user

);

event LoginDone(User user);

event LoginFail(string fail);

// Car Info Events

event carAdded(

uint256 indexed carID,

//address payable indexed carOwner,

CarInfo car

);

// Rental Events

event rentalPlaced(

string customerName,

uint customerAge,

uint256 licenseID,

uint256 carID

);

event rentalConfirmed(

string customerName,

uint customerAge,

uint256 licenseID

//uint256 carID

);

event rentalCanceled(

string customerName,

uint customerAge,

uint256 licenseID

//uint256 carID

);

event carReturned(

string customerName,

uint customerAge,

uint256 licenseID

//uint256 carID

);

// Modifiers

modifier onlyOwner() {

require(msg.sender == rentalOwner, 'only order owner can call this function');

\_;

}

// modifier onlyOneRental() {

// require(rentalCount <= 1, 'only one order at a time');

// \_;

// }

modifier existingRental() {

require(rentals[rentalOwner].isValidRental == true, 'Please confirm/cancel existing order');

\_;

}

modifier unconfirmedRental() {

require(rentals[rentalOwner].hasConfirmed == false, 'Cannot cancel confirmed rental');

\_;

}

modifier validInfo(string memory \_customerName, uint \_age, uint256 \_licenseID) {

require(bytes(\_customerName).length > 0 && \_age > 0 && \_licenseID >0, 'Please enter name, age and license');

require(\_age >= 18, 'Invalid age');

\_;

}

modifier allowNewRental() {

require(rentals[rentalOwner].isValidRental == false, 'Please start new rental');

\_;

}

modifier enoughAmount(){

require(msg.value >= price, 'No enough Ethers');

\_;

}

// modifier carAvailable(uint256 \_carID){

// require(cars[\_carID].isAvailable == true, 'Car is not available.');

// \_;

// }

// Account functions

function SignUp(address payable \_Account, string memory \_userName, string memory \_password) public returns (bool) {

require(\_Account != address(0), "Account can not be empty");

require(users[\_Account].userID == 0, "Account already exists");

require(bytes(\_userName).length > 0, "Please enter a name");

require(bytes(\_password).length > 0, "Please enter a password");

User memory \_user;

totalUserNumber++;

\_user.userID = totalUserNumber;

\_user.userAddress = \_Account;

\_user.userName = \_userName;

\_user.password = \_password;

\_user.isUserLoggedIn = false;

users[\_Account] = \_user;

emit accountCreated(\_user);

return true;

}

function Login(address \_address, string memory \_password) public returns (bool) {

User memory \_user = users[\_address];

if (keccak256(abi.encodePacked(\_user.password)) ==

keccak256(abi.encodePacked(\_password))) {

\_user.isUserLoggedIn = true;

emit LoginDone(\_user);

return \_user.isUserLoggedIn;

} else {

emit LoginFail("Incorrect Password");

return false;

}

}

function logout(address \_address) public {

users[\_address].isUserLoggedIn = false;

}

// Rentee functions

function addCar (string memory \_carOwner, string memory \_carLocation) public returns (bool) {

// for (uint i = 0; i < carInformation.length; i++) {

// require(\_carID != carInformation[i].carID && \_carID > 0, 'Please choose a unique carID' );

// }

require(bytes(\_carOwner).length > 0 && bytes(\_carLocation).length > 0 , 'Please enter name and car location');

CarInfo memory \_car;

totalCarNum++;

\_car.ownerName = \_carOwner;

\_car.carLocation = \_carLocation;

\_car.carID = totalCarNum;

\_car.isAvailable = true;

cars[totalCarNum] = \_car;

//carInformation.push(\_car);

//cars[carOwner].push(information);

emit carAdded(totalCarNum, \_car);

return true;

}

// Rental main functions

function createRental (string memory \_customerName, uint \_age, uint256 \_licenseID, uint256 \_carID) public payable allowNewRental() enoughAmount() validInfo(\_customerName, \_age, \_licenseID) {

CarInfo memory \_car = cars[\_carID];

require(\_car.isAvailable == true, 'Car is not available.');

rentals[rentalOwner].isValidRental = true;

rentals[rentalOwner].customerName = \_customerName;

rentals[rentalOwner].customerAge = \_age;

rentals[rentalOwner].licenseID = \_licenseID;

rentals[rentalOwner].hasConfirmed = false;

rentals[rentalOwner].hasReturned = false;

\_car.isAvailable = false;

//wallet.transfer(msg.value);

emit rentalPlaced(rentals[rentalOwner].customerName, rentals[rentalOwner].customerAge, rentals[rentalOwner].licenseID, \_carID);

}

function confirmRental() public existingRental() {

rentals[rentalOwner].hasConfirmed = true;

// wallet.transfer(msg.value);

emit rentalConfirmed(rentals[rentalOwner].customerName, rentals[rentalOwner].customerAge, rentals[rentalOwner].licenseID);

}

function cancelRental() public payable existingRental() unconfirmedRental() {

rentals[rentalOwner].hasConfirmed = false;

rentals[rentalOwner].isValidRental = false;

rentalOwner.transfer(price);

emit rentalCanceled(rentals[rentalOwner].customerName, rentals[rentalOwner].customerAge, rentals[rentalOwner].licenseID);

}

function returnCar(uint256 \_carID) public existingRental(){

rentals[rentalOwner].hasConfirmed = false;

rentals[rentalOwner].isValidRental = false;

rentals[rentalOwner].hasReturned = true;

cars[\_carID].isAvailable = true;

emit carReturned(rentals[rentalOwner].customerName, rentals[rentalOwner].customerAge, rentals[rentalOwner].licenseID);

}

// Functions for testing

function getcarAvailability(uint256 \_carID) public view returns (bool) {

return cars[\_carID].isAvailable;

}

function getorderValidity () public view returns (bool) {

return rentals[rentalOwner].isValidRental;

}

function getorderConfirmation () public view returns (bool) {

return rentals[rentalOwner].hasConfirmed;

}

function getorderReturn() public view returns (bool) {

return rentals[rentalOwner].hasReturned;

}

function getcustomerName () public view returns (string memory) {

return rentals[rentalOwner].customerName;

}

function getcustomerAge() public view returns (uint) {

return rentals[rentalOwner].customerAge;

}

function getlicenseID() public view returns (uint256) {

return rentals[rentalOwner].licenseID;

}

function getownerName(uint256 \_carID) public view returns (string memory) {

return cars[\_carID].ownerName;

}

function getcarLocation(uint256 \_carID) public view returns (string memory) {

return cars[\_carID].carLocation;

}

function getcarID(uint256 \_carID) public view returns (uint256) {

return cars[\_carID].carID;

}

function getBalanceofSC() public view returns(uint256) {

return carOwner.balance;

}

function getBalanceofOwner() public view returns(uint256) {

return balances[rentalOwner];

}

}