const { ethers } = require("hardhat");

const { expect } = require("chai");

describe("CarRental contract", async function () {

const cost = 1e17;

let carRental;

let BUYER;

let RENTER;

let CarRental;

beforeEach(async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const [BUYER, RENTER] = await ethers.getSigners();

const carRental = await CarRental.deploy();

await carRental.deployed();

});

it("should deploy smart contract properly", async function () {

const BUYER = await ethers.getSigners();

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.deployed();

console.log(carRental.address);

expect(carRental.address !== "");

});

it("should provide owner name, unique car ID, and car location", async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

const name = await carRental.getownerName(carNum);

const location = await carRental.getcarLocation(carNum);

expect(name != "" && location != "");

});

it("should provide customer name, valid age, License number and car ID", async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

await carRental.createRental("sara", 20, 123444, carNum, {

from: BUYER,

value: ethers.utils.parseUnits("1", 17),

});

const name = await carRental.getcustomerName();

const age = await carRental.getcustomerAge();

const ID = await carRental.getlicenseID();

expect(name != "" && age >= 18 && ID > 0);

expect(carRental.getorderValidity == true);

expect(carRental.getcarAvailability(0) == true);

});

it("should be a valid rental to confirm", async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

const validityStatus = await carRental.getorderValidity();

expect(validityStatus == true);

});

it("should be an unconfirmed order to be able to cancel", async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

const confirmStatus = await carRental.getorderConfirmation();

expect(confirmStatus == false);

});

it("should confirm/cancel existing order before creating new order", async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

await carRental.createRental("sara", 20, 123444, carNum, {

from: BUYER,

value: ethers.utils.parseUnits("1", 17),

});

const validityStatus = await carRental.getorderValidity();

const confirmStatus = await carRental.getorderConfirmation();

expect(validityStatus == true && confirmStatus == false);

});

it("customer should have enough ETHERS", async () => {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

expect(ethers.provider.getBalance(BUYER) >= cost);

});

it("rental can be done if everything is ok", async () => {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

await carRental.createRental("sara", 20, 123444, carNum, {

from: BUYER,

value: ethers.utils.parseUnits("1", 17),

});

expect(carRental.getorderValidity == true);

});

it("rental can be confirmed if everything is ok", async () => {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

await carRental.createRental("sara", 20, 123444, carNum, {

from: BUYER,

value: ethers.utils.parseUnits("1", 17),

});

await carRental.confirmRental();

expect(carRental.getorderConfirmation == true);

});

it("should have correct balance of ETHERS", async () => {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

const scBalance = await carRental.getBalanceofSC();

expect(scBalance >= cost);

});

it("rental can be returned if everything is ok", async () => {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

await carRental.createRental("sara", 20, 123444, carNum, {

from: BUYER,

value: ethers.utils.parseUnits("1", 17),

});

await carRental.confirmRental();

await carRental.returnCar(0);

const returnStatus = carRental.getorderReturn();

expect(returnStatus == true);

});

it("customer receive the money after order cancellation", async function () {

const CarRental = await ethers.getContractFactory("CarRental");

const carRental = await CarRental.deploy();

await carRental.addCar("sara", "kelowna");

const carNum = carRental.totalCarNum();

await carRental.createRental("sara", 20, 123444, carNum, {

from: BUYER,

value: ethers.utils.parseUnits("1", 17),

});

await carRental.cancelRental();

expect(ethers.provider.getBalance(BUYER) == cost);

});

});