**Raport de folosire a chatGPT in testarea software**

**Introducere**

Analiza următoare se referă la testarea unei implementări a standardului de smartcontracte ERC20. Contractul este implementat in fisierul ERC20.sol . Am ales sa utilizam testarea functionala pentru acest contract, abordand tehnica *Partitionarii pe categorii.* Am detaliat aceasta abordate in Documentatie1.docx. Teste scrise, folosind aceasta abordare se găsesc in fisierul erc20Tests.js . I-am cerut lui chatGPT, sa scrie teste, utilizand testarea functionala, abordand tehnica *Partitionarii pe categorii.*

**Utilizarea AI**

Prompt-ul: “Scrie teste functionale care sa testeze dupa principiul partitionarii pe categorii, urmatoarea functie. Testele v-or fii scrise in JavaScript, folosind "hardhat" si "chai". – contract code – ”

Conversatia se regaseste si la adresa: <https://chat.openai.com/share/cd282cb3-4bda-4043-9906-513a3cebfeb6>

Raspunsul lui chatGPT a fost:

Pentru a scrie teste funcționale care să testeze funcția `ERC20`, putem folosi principiul partitionării pe categorii pentru a acoperi diferite cazuri. Iată câteva exemple de teste folosind framework-ul Hardhat și chai:

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

describe("ERC20", function() {

let ERC20, erc20, owner, addr1, addr2;

beforeEach(async function() {

ERC20 = await ethers.getContractFactory("ERC20");

[owner, addr1, addr2] = await ethers.getSigners();

erc20 = await ERC20.deploy(1000, "Test Token", "TT");

await erc20.deployed();

});

it("Should return the correct total supply", async function() {

expect(await erc20.totalSupply()).to.equal(1000);

});

it("Should return the correct balance for owner", async function() {

expect(await erc20.balanceOf(owner.address)).to.equal(1000);

});

it("Should transfer tokens between accounts", async function() {

await erc20.transfer(addr1.address, 100);

expect(await erc20.balanceOf(addr1.address)).to.equal(100);

expect(await erc20.balanceOf(owner.address)).to.equal(900);

});

it("Should emit Transfer event when tokens are transferred", async function() {

await expect(erc20.transfer(addr1.address, 100))

.to.emit(erc20, "Transfer")

.withArgs(owner.address, addr1.address, 100);

});

it("Should approve spender and allow them to transfer tokens", async function() {

await erc20.approve(addr1.address, 100);

expect(await erc20.allowance(owner.address, addr1.address)).to.equal(100);

await erc20.connect(addr1).transferFrom(owner.address, addr2.address, 50);

expect(await erc20.balanceOf(addr2.address)).to.equal(50);

});

it("Should emit Approval event when allowance is set", async function() {

await expect(erc20.approve(addr1.address, 100))

.to.emit(erc20, "Approval")

.withArgs(owner.address, addr1.address, 100);

});

it("Should not allow transfer if balance is insufficient", async function() {

await expect(erc20.transfer(addr1.address, 150)).to.be.revertedWith("Insufficient funds!");

});

it("Should not allow transfer if allowance is insufficient", async function() {

await erc20.approve(addr1.address, 50);

await expect(erc20.connect(addr1).transferFrom(owner.address, addr2.address, 100)).to.be.revertedWith("Insufficient allowance!");

});

});

Aceste teste acoperă mai multe scenarii:

1. Verifică totalul de aprovizionare.

2. Verifică soldurile contului inițial.

3. Verifică transferul de tokenuri între conturi.

4. Verifică emiterea evenimentului `Transfer` la transferul de tokenuri.

5. Verifică aprobarea unui cheltuitor și transferul de tokenuri de către acesta.

6. Verifică emiterea evenimentului `Approval` la stabilirea unei aprobări.

7. Verifică dacă transferul nu este permis dacă fondurile sunt insuficiente.

8. Verifică dacă transferul nu este permis dacă aprobarea este insuficientă.

Acestea sunt doar câteva teste de bază. Puteți adăuga și mai multe teste pentru a acoperi mai multe scenarii și edge case-uri.

**Comparatii**

Comparatia facuta de chatGPT intre cele 2 seturi de teste:

Ambele seturi de teste sunt bine scrise și acoperă o gamă variată de scenarii pentru contractul ERC20. În general, ambele folosesc un stil de testare similar, bazat pe biblioteca Chai și pe framework-ul Mocha.

Există câteva diferențe între cele două seturi de teste:

1. Structura testelor: Setul de teste scris de tine pare să fie organizat în funcție de funcționalitatea contractului, grupând teste pentru diferite funcții ale contractului ERC20. Pe de altă parte, setul de teste scris de ChatGPT pare să fie organizat în funcție de comportamentul așteptat al contractului în diverse scenarii.
2. Utilizarea beforeEach (în setul de teste ChatGPT): În setul de teste ChatGPT, se utilizează beforeEach pentru a inițializa variabilele și a desfășura acțiuni comune înainte de fiecare test. Aceasta poate fi o practică bună pentru a evita repetiția codului.
3. Capturarea și tratarea excepțiilor: Am observat că ambele seturi de teste gestionează capturarea și tratarea excepțiilor în mod corespunzător pentru a verifica că anumite operațiuni eșuează așa cum este de așteptat.
4. Utilizarea expect și aserțiuni: Ambele seturi de teste folosesc expect și aserțiuni pentru a verifica comportamentul contractului în diferite scenarii.
5. Utilizarea variabilelor: Ambele seturi de teste folosesc variabile pentru a stoca diferite adrese și obiecte de contract, ceea ce face codul mai ușor de înțeles și de întreținut.

În concluzie, ambele seturi de teste sunt bine scrise și își ating scopul de a asigura că contractul ERC20 se comportă conform așteptărilor în diverse scenarii. Este întotdeauna important să menții teste comprehensive și bine structurate pentru a asigura calitatea și robustețea codului.

Comparatia facuta de mine, intre cele 2 seturi de teste:

1. chatGPT utilizeaza ‘beforeEach’ in timp ce noi folosim
2. const {
3. loadFixture,
4. } = require("@nomicfoundation/hardhat-toolbox/network-helpers");

care este mai eficient.

2. chatGPT nu a facut teste pentru cazurile de fail

3. chatGPT nu a facut teste legate de limitele maxime ale variabilelor, sau legate de argumente transmise gresit

1. Testele facute de chatGPT, sunt mai putine, si incomplete