Raport privind Utilizarea Inteligenței Artificiale în Dezvoltarea și Testarea Platformei de Abonamente NFT

Scopul Utilizării AI

Inteligența Artificială (AI) a fost folosită ca instrument de suport pentru accelerarea procesului de dezvoltare, validare și documentare a platformei blockchain pentru abonamente reprezentate prin NFT-uri. AI-ul nu a înlocuit deciziile de design sau testarea manuală, ci a acționat ca un asistent inteligent în fazele critice ale proiectului.

Zone în care AI-ul a fost Util

1. Asistență în Scrierea Contractelor Inteligente

Tool folosit: ChatGPT (OpenAI)

Prompt exemplu:

> *Scrie un contract ERC721 numit AbonamentNFT care include: data de expirare, un emitent, și o funcție de verificare dacă abonamentul este activ.*

Răspuns:

```
contract AbonamentNFT is ERC721 {
    struct Abonament {
        address emitent;
        uint256 dataExpirare;
    }
    mapping(uint256 => Abonament) public abonamente;
    function esteActiv(uint256 tokenId) public view returns (bool) {
        return block.timestamp < abonamente[tokenId].dataExpirare;
    }
}
```

Codul generat a oferit o bază bună, dar a fost ulterior adaptat manual pentru: verificări suplimentare de ownership, compatibilitate cu Ownable, Pausable și ERC721URIStorage, validări specifice platformei.

2. Propuneri pentru Testare

Tool folosit: ChatGPT, GitHub Copilot

Prompt exemplu:

> ***Scrie un test pentru funcția esteActiv din contractul AbonamentNFT folosind Hardhat și Chai.***

Răspuns:

```
it("ar trebui să returneze true dacă data expirării este în viitor",
async function () {
    const inViitor = Math.floor(Date.now() / 1000) + 3600;
    await abonamentNFT.connect(owner).mint(addr1.address, inViitor);
    expect(await abonamentNFT.esteActiv(1)).to.be.true;
});
```

Comparativ cu testele scrise manual:

Criteriu	AI	Manual	Observații
Acoperire logică	Simplă	Complexă	AI-ul oferă bază, dar lipsesc edge-case-uri
Timp dezvoltare	Rapid	Mediu	AI eficientizează începutul
Complexitate	Scăzută	Ridicată	Testele proprii includ fluxuri reale
Tool-uri avansate	Sugerează fast-check	Include fast-check și benchmark	AI oferă direcții

3. Redactarea Documentației

AI-ul a ajutat la: organizarea documentației, explicarea interacțiunii dintre contracte și generarea secțiunilor de exemplu.

Prompt exemplu:

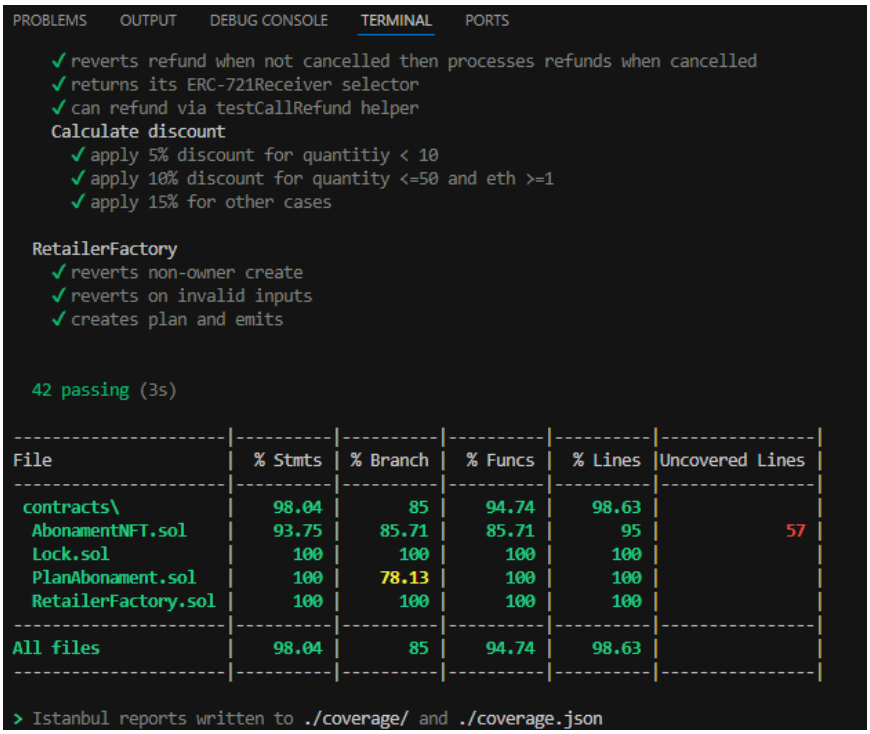
> ***Explică pe scurt cum funcționează interacțiunea dintre un contract RetailerFactory și PlanAbonament într-o platformă de abonamente NFT.***

Răspuns:

Contractul RetailerFactory creează instanțe de PlanAbonament pentru fiecare comerciant. Fiecare plan conține logica de emitere a NFT-urilor prin AbonamentNFT, iar abonamentele pot fi verificate sau reînnoite prin metodele publice ale acestora.

4. Analiza Acoperirii Testelor

Testele generate cu AI au contribuit la creșterea acoperirii codului, dar unele funcții au necesitat completări manuale.



Tabelul de acoperire evidențiază faptul că AbonamentNFT.sol are linii neacoperite și necesită atenție suplimentară.

5. Comparație între Testele Proprii și cele Autogenerate cu AI

Pentru a evalua eficiența și aplicabilitatea testelor generate cu ajutorul AI, acestea au fost comparate cu suita completă de teste proprii, scrise manual. Comparația s-a realizat pe baza mai multor criterii.

Criteriu	Teste Autogenerate (AI)	Teste Scrise Manual	Observații
Acoperire logică	Bazată pe exemple simple și generice	Include edge-case-uri, fluxuri complexe	AI-ul oferă o bază, dar necesită completări
Timp de dezvoltare	Foarte scurt (secunde/minute)	Mediu-ridicat (zeci de minute/oră)	AI-ul accelerează prototiparea

Calitate cod	Uneori inconsistent, lipsă comentarii	Uniform, comentat, integrat în framework	Codul generat trebuie adaptat
Complexitate testată	Validări directe, simple	Fluxuri de abonare, refund, validări ERC721	Testele proprii oferă profunzime
Testare avansată	Sugerează fast-check, dar minimă	Include fuzzing, benchmark gaz	Testele proprii adaugă valoare în scenarii reale

Prompt exemplu:

> ***Scrie un test pentru funcția `esteActiv` din contractul AbonamentNFT folosind Hardhat și Chai.***

Răspuns:

```
it("ar trebui să returneze true dacă data expirării este în viitor",
async function () {
  const inViitor = Math.floor(Date.now() / 1000) + 3600;
  await abonamentNFT.connect(owner).mint(addr1.address, inViitor);
  expect(await abonamentNFT.esteActiv(1)).to.be.true;
});
```

Interpretare Rezultate

Testele generate cu AI au fost utile în special pentru cazurile de bază. Totuși, suita finală completă a fost obținută prin rafinare manuală, adăugare de fluxuri reale, manipulare de stări și verificări ERC721.

6. Exemple Detaliate din Testele Proprii

6.1 Teste Unitare – Contractul AbonamentNFT

Teste proprii pentru validarea implementării ERC-721 cu funcționalități specifice de abonament:

```
it("mint and transfer ETH to plan", async function () {
  await expect(
    nft.connect(other).cumparaAbonament(other.address, { value:
price })
  )
    .to.emit(nft, "AbonamentCumparat")
```

```
    .withArgs(other.address, 1);
  expect(await nft.ownerOf(1)).to.equal(other.address);
});



it("reverts on wrong ETH amount", async function () {
  await expect(
    nft.connect(other).cumparaAbonament(other.address, { value:
price - 1n })
  ).to.be.revertedWith("Suma ETH gresita");
});
```

6.2 Teste de Integrare – Flux Complet

Test de integrare între RetailerFactory, PlanAbonament și AbonamentNFT:

```
it("should allow user to subscribe and mint NFT", async function
() {
  await plan.connect(addr1).cumparaSubscriptie({ value:
ethers.parseEther("0.1") });
  const balance = await nft.balanceOf(addr1.address);
  expect(balance).to.equal(1);
});



it("should enforce max supply limit", async function () {
  for (let i = 0; i < initialSupply; i++) {
    await plan.connect(owner).cumparaSubscriptie({ value:
ethers.parseEther("0.1") });
  }
  await expect(
    plan.connect(addr1).cumparaSubscriptie({ value:
ethers.parseEther("0.1") })
  ).to.be.revertedWith("No subscriptii available");
});
```

6.3 Alte Teste Specifice – Reduceri și Refund

Exemple din PlanAbonament pentru logica de discount și refund:

```
it("apply 10% discount for quantity <=50 and eth >=1", async ()
```

```
=> {
  const net = await
plan.calculeazaDiscount(ethers.parseEther("2"), 20);
  expect(net).to.equal(ethers.parseEther("36"));
});
```

```
it("reverts refund when not cancelled then processes refunds when
cancelled", async function () {
  await plan.connect(subscriber).cumparaSubscriptie({ value:
price });
  await expect(
    plan.connect(subscriber).refundSubscriptie(1)
  ).to.be.revertedWith("Plan is not cancelled");
  await plan.connect(owner).cancelPlan();
  await expect(
    plan.connect(subscriber).refundSubscriptie(1)
  ).to.emit(plan, 'RefundRequested');
});
```

Limitări Observate ale AI-ului

- Nu a putut anticipa edge cases sau probleme logice profunde.

- Sugestiile uneori conțineau redundanțe sau greșeli minore.

- Nu a putut optimiza codul pentru gaz fără ajustări manuale.

Concluzii

Utilizarea AI-ului în dezvoltarea platformei NFT a redus semnificativ timpul de scriere a codului și a documentației. Totuși, validarea logicii, optimizarea și securizarea codului au necesitat intervenție manuală și testare riguroasă.

Bibliografie

1. OpenAI. (2024). ChatGPT Model Documentation. https://platform.openai.com/docs

2. GitHub Copilot. (2024). Your AI pair programmer. https://github.com/features/copilot

3. OpenZeppelin Docs. (2024). Secure Smart Contract Framework. https://docs.openzeppelin.com

4. Hardhat Documentation. (2024). Ethereum Development Environment. https://hardhat.org