

COLEGIUL NAȚIONAL DE INFORMATICĂ „GR. MOISIL” BRAȘOV

**LUCRARE PENTRU DOBÂNDIREA COMPETENȚELOR
PROFESIONALE**

BlockFund

Elev: Bleotu Alexandru-David

Clasa a XII-a D

Profesor îndrumător: Trandabur Alexandra/

Șerban Manuela

Brașov – Mai 2025

Cuprins

1. Motivația alegerii temei lucrării	2
1.1 Context și elemente fundamentale	2
1.2 Probleme identificate în soluțiile clasice	2
1.3 Avantajele unei platforme de crowdfunding pe blockchain	2
2. Utilitatea aplicației	3
2.1 Scenarii de utilizare (User Stories).....	3
2.2 Avantaje pentru utilizatori	4
2.3 Beneficii pentru dezvoltare	4
3. Structura aplicației	5
3.1 Arhitectura generală	5
3.2 Organizarea codului	5
3.3 Fișiere de configurare.....	6
4. Detalii de implementare.....	8
4.1 Smart Contract (Hardhat + Solidity).....	8
4.2 Front-end (React + Tailwind + Ethers.js)	9

1. Motivația alegerii temei lucrării

1.1 Context și elemente fundamentale

- **Dezvoltarea Web3:** În ultimii ani, arhitectura descentralizată a Web3 (blockchain, smart contracts) a câștigat tot mai mult teren în fața soluțiilor centralizate. Nevoia de transparență și de eliminare a intermediarilor a stimulat apariția unor aplicații care se bazează exclusiv pe cod public și validat de rețea.
- **Popularitatea crowdfunding-ului:** Crowdfunding-ul a făcut mult mai simplă strângerea de bani pentru afaceri noi, proiecte sociale sau artistice. Însă platformele clasice iau comisioane între 5% și 10% și cer verificări de identitate care pot îngreuna participarea celor din țările în dezvoltare.

1.2 Probleme identificate în soluțiile clasice

- **Lipsă de transparență:** Donatorii nu pot verifica direct modul în care sunt gestionate fondurile depind de rapoartele proprietarilor de campanie sau de auditurile periodice ale platformelor.
- **Comisioane ridicate:** Taxele de procesare și de platformă pot ajunge la 15–20% din suma strânsă.
- **Acces limitat:** Utilizatorii din țări cu sisteme bancare subdezvoltate sau sancțiuni internaționale întâmpină bariere la transferuri de fonduri.
- **Dependență de infrastructură centrală:** Întreruperile de serviciu, atacurile DDoS sau blocarea conturilor pot împiedica campaniile să-și atingă obiectivele.

1.3 Avantajele unei platforme de crowdfunding pe blockchain

- **Transparență totală:** Toate tranzacțiile sunt publice, inspectabile oricând pe exploratorul de blocuri (Etherscan).
- **Costuri reduse:** Smart contract-ul rulează independent, eliminând majoritatea comisioanelor de intermediere.
- **Acces global și instantaneu:** Orice persoană cu MetaMask și Ether poate contribui imediat, fără KYC extins.
- **Rezistență la cenzură:** Campaniile nu pot fi închise unilateral de către o autoritate centrală, atâta timp cât smart contract-ul este activ pe rețea.

2. Utilitatea aplicației

2.1 Scenarii de utilizare (User Stories)

1. Creator de campanie

- Context: Un utilizator dorește să își promoveze proiectul și să strângă fonduri.
- Flux de lucru:
 - Completează, printr-un formular intuitiv din interfața web, titlul, descrierea, obiectivul financiar (suma țintă), data de încheiere și categoria campaniei, apoi încarcă imagini reprezentative.
 - La lansarea campaniei, tranzacția este trimisă către smart contract-ul de pe rețeaua Mainnet, iar utilizatorul achită taxa de gas aferentă.
 - Campania creată apare imediat în lista de proiecte, afișând atât datele on-chain, cât și elementele off-chain.
- Rezultat: Oricine poate lansa o campanie în câțiva pași simpli.

2. Contribuția la campanii

- Context: Un susținător dorește să ofere suport financiar.
- Flux de lucru:
 - Navighează pe pagina de campanii și selectează proiectul la care vrea să contribuie.
 - Apasă „Contribuie”, introduce suma dorită și confirmă tranzacția în MetaMask.
 - După validare, totalul on-chain se actualizează automat.
 - Trimite un mesaj direct creatorului campaniei pentru a pune întrebări suplimentare.
- Rezultat: Orice utilizator cu un portofel MetaMask poate trimite fonduri rapid și transparent.

3. Vizualizare fără conectare

- Context: Un potențial susținător sau vizitator pur și simplu explorează campaniile disponibile.
- Flux de lucru:
 - Accesează pagina „Explore” fără a conecta un wallet.
 - Consultă detaliile campaniilor (titlu, descriere, nivel de finanțare) generate prin interogări read-only la smart contract.
- Rezultat: Informațiile sunt disponibile tuturor, chiar dacă nu dețin sau nu au conectat un portofel.

4. Administrare și întreținere

- Context: Administratorul trebuie să verifice și să reseteze starea campaniilor sau să schimbe rețeaua Ethereum utilizată de BlockFund.
- Flux de lucru:
 - Accesează interfața de administrare Supabase, unde găsește tabele cu campanii, utilizatori sau mesaje.

- Rulează, dacă este necesar, scriptul Hardhat pentru resetarea contractului pe rețeaua locală.
- Rezultat: Monitorizarea și întreținerea aplicației se fac rapid, cu instrumentele deja existente.

2.2 Avantaje pentru utilizatori

Caracteristică	Platformă tradițională	BlockFund
Comisioane	5 – 15% per tranzacție	Gas fee + 2.5% la retragere
Timp de procesare	1 – 3 zile (plăți bancare)	Minute (confirmare blockchain)
Transparență	Rapoarte manual	Tranzacții publice
Acces global	KYC/AML, limitări geografice	Orice wallet MetaMask
Reziliență	Dependent de servere	Imuabilitate a contractului

- **Economii:** Eliminarea comisiunilor mari de platformă lasă mai mulți bani în mâna creatorilor și a susținătorilor.
- **Experiență fluidă:** Toate operațiunile se fac direct în browser, fără redirectionări către procese de plată externe.
- **Încredere sporită:** Oricine poate urmări în timp real cum sunt gestionate fondurile.

2.3 Beneficii pentru dezvoltare

1. Arhitectură modulară

- Front-end React + Tailwind, smart contracts în Solidity gestionate cu Hardhat și back-office pe Supabase.
- Fiecare funcție a contractului are propriul test și poate fi updatată independent.

2. Ciclu de dezvoltare rapid

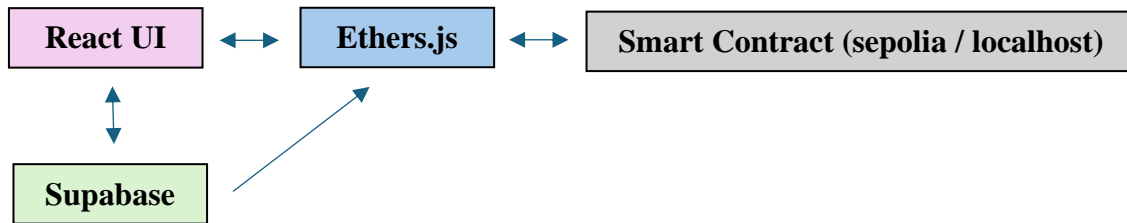
- Hot-reload prin Vite pentru front-end și rețea locală Hardhat pentru iterații imediate.
- Deploy direct pe Sepolia cu un singur comandă, în vederea demo-urilor.

3. Costuri reduse de infrastructură

- Folosirea testnet-ului Sepolia minimizează cheltuielile cu gas în faza de dezvoltare.
- Planul gratuit Supabase acoperă nevoile de stocare off-chain pentru MVP.

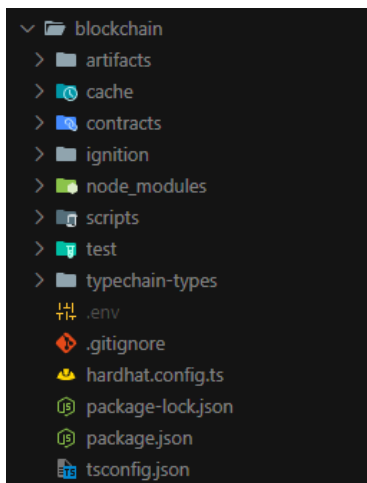
3. Structura aplicației

3.1 Arhitectura generală

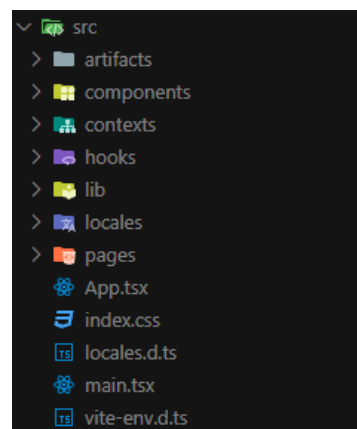
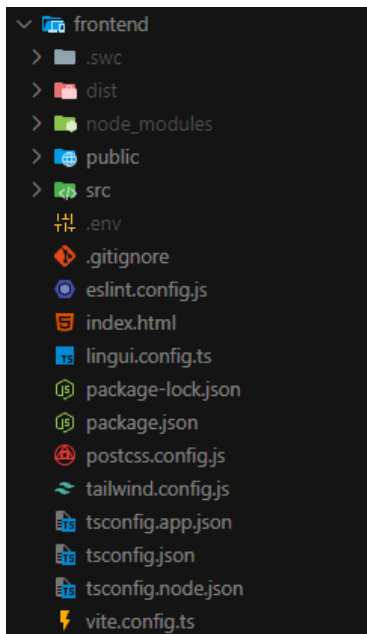


3.2 Organizarea codului

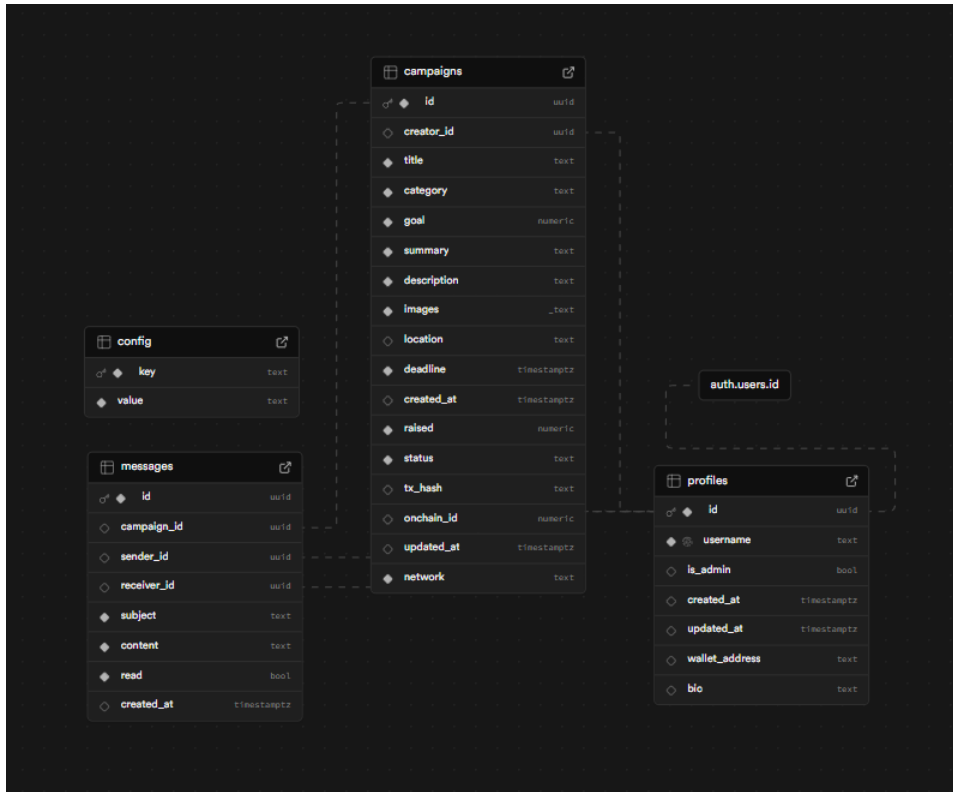
1. Structura directorului **blockchain**



2. Structura directorului **frontend**



3. Structura bazei de date de pe supabase



3.3 Fișiere de configurare

- hardhat.config.js

```
1 import "@nomicfoundation/hardhat-toolbox";
2 import * as dotenv from "dotenv";
3 import { HardhatUserConfig } from "hardhat/config";
4
5 dotenv.config();
6
7 const { ALCHEMY_API_KEY, WALLET_PRIVATE_KEY, ETHERSCAN_API_KEY } = process.env;
8
9 const config: HardhatUserConfig = {
10   solidity: "0.8.28",
11   networks: {
12     sepolia: {
13       url: `https://eth-sepolia.g.alchemy.com/v2/${ALCHEMY_API_KEY}`,
14       accounts: WALLET_PRIVATE_KEY ? [WALLET_PRIVATE_KEY] : [],
15     },
16     mainnet: {
17       url: `https://eth-mainnet.g.alchemy.com/v2/${ALCHEMY_API_KEY}`,
18       accounts: WALLET_PRIVATE_KEY ? [WALLET_PRIVATE_KEY] : [],
19     },
20   },
21   etherscan: {
22     apiKey: ETHERSCAN_API_KEY,
23   },
24 };
25
26 export default config;
```

- vite.config.js

```
1 import { lingui } from "@lingui/vite-plugin";
2 import react from "@vitejs/plugin-react-swc";
3 import { defineConfig } from "vite";
4
5 export default defineConfig({
6   plugins: [
7     react({
8       plugins: [["@lingui/swc-plugin", {}]],
9     }),
10    lingui(),
11  ],
12  optimizeDeps: {
13    exclude: ["lucide-react"],
14  },
15 });
16
```

- package.json (front-end)

```
1 {
2   "name": "project",
3   "private": true,
4   "version": "0.0.0",
5   "type": "module",
6   "scripts": {
7     "dev": "vite --host",
8     "build": "tsc && vite build",
9     "lint": "eslint . --ext ts,tsx --report-unused-disable-directives --max-warnings 0",
10    "preview": "vite preview"
11  },
12  "dependencies": {
13    "@emailjs/browser": "^4.4.1",
14    "@lingui/cli": "^5.3.1",
15    "@lingui/core": "^5.3.1",
16    "@lingui/core/macro": "^5.3.1",
17    "@lingui/react": "^5.3.1",
18    "@lingui/swc-plugin": "^5.5.2",
19    "@supabase/supabase-js": "^2.39.3",
20    "@vitejs/plugin-react-swc": "^3.9.0",
21    "date-fns": "^4.1.0",
22    "ethers": "^6.9.0",
23    "framer-motion": "^11.0.3",
24    "lucide-react": "^0.309.0",
25    "react": "^18.2.0",
26    "react-dom": "^18.2.0",
27    "react-hot-toast": "^2.4.1",
28    "react-router-dom": "^7.5.2"
29  },
30  "devDependencies": {
31    "@lingui/vite-plugin": "^5.3.1",
32    "@types/react": "^18.2.43",
33    "@types/react-dom": "^18.2.17",
34    "@typescript-eslint/eslint-plugin": "^6.14.0",
35    "@typescript-eslint/parser": "^6.14.0",
36    "@vitejs/plugin-react": "^4.2.1",
37    "autoprefixer": "^10.4.17",
38    "eslint": "^8.55.0",
39    "eslint-plugin-react-hooks": "^4.6.0",
40    "eslint-plugin-react-refresh": "^0.4.5",
41    "postcss": "^8.4.33",
42    "tailwindcss": "^3.4.1",
43    "typescript": "^5.2.2",
44    "vite": "^5.0.8"
45  }
46 }
47
```


4. Detalii de implementare

4.1 Smart Contract (Hardhat + Solidity)

1. Campaign.sol

- Funcții cheie:
 - function createCampaign(uint256 _goal, uint256 _deadline, string calldata _metadataCID)
 - function contribute(uint256 _campaignId)
 - function closeCampaign(uint256 _campaignId)
 - function withdraw(uint256 _campaignId)
 - function collectFees(uint256 _campaignId)
 - function getCampaign(uint256 _campaignId)
 - function updateCampaign(uint256 _campaignId, uint256 _newGoal, uint256 _newDeadline, string calldata _newMetadataCID)
 - function getCampaignCount()

```
1  function contribute(uint256 _campaignId) external payable {
2      CampaignData storage campaignData = campaigns[_campaignId];
3      require(campaignData.status != CampaignStatus.CLOSED, "Campaign is closed");
4      require(block.timestamp < campaignData.deadline, "Campaign has ended");
5      require(msg.value > 0, "No ETH sent");
6      require(msg.sender != campaignData.creator, "Creator cannot fund their own campaign");
7
8      campaignData.totalFunded += msg.value;
9      contributions[_campaignId][msg.sender] += msg.value;
10     emit ContributionMade(_campaignId, msg.sender, msg.value);
11 }
```

2. Testare smart contract

```
Campaign Contract
✓ Should create a new campaign
✓ Should not allow the creator to fund their own campaign
✓ Should allow contributions
✓ Should ensure totalContributions is equal to totalFunded
✓ Should retain totalContributions after withdrawal
✓ Should allow the creator to close the campaign explicitly
✓ Should allow the creator to withdraw funds after closing the campaign explicitly
✓ Should not allow withdrawal if totalFunded is 0
✓ Should allow the creator to update the campaign
✓ Should not allow non-creators to update the campaign
✓ Should not allow non-creators to withdraw funds
✓ Should not allow contributions after the deadline
✓ Should not allow non-creators to close a campaign
✓ Should allow contributions exceeding the campaign goal
✓ Should hold 2.5% fee on withdraw and allow feeReceiver to collect it

15 passing (1s)
```

```

1 it("Should allow contributions", async function () {
2   const goal = ethers.parseEther("5");
3   const deadline = Math.floor(Date.now() / 1000) + 3600;
4   const metadataCID = "QmExampleCID";
5
6   await campaign.createCampaign(goal, deadline, metadataCID);
7
8   const contribution1 = ethers.parseEther("2");
9   await campaign.connect(addr1).contribute(1, { value: contribution1 });
10
11   const campaignData = await campaign.getCampaign(1);
12   expect(campaignData.totalFunded).toEqual(contribution1);
13
14   const contribution2 = ethers.parseEther("1");
15   await campaign.connect(addr2).contribute(1, { value: contribution2 });
16
17   const updatedData = await campaign.getCampaign(1);
18   expect(updatedData.totalFunded).toEqual(ethers.parseEther("3"));
19 });

```

4.2 Front-end (React + Tailwind + Ethers.js)

1. Configurare proiect
 - vite.config.js
 - tailwind.config.js
 - postcss.config.js
 - lingui.config.ts
2. Conectare la wallet prin MetaMask
 - Hook personalizat – **useWallet.ts**

```

1 if (!(window as any).ethereum) {
2   throw new Error('MetaMask is not installed');
3 }
4
5 const [newAddress] = (await (window as any).ethereum.request({
6   method: "eth_requestAccounts",
7   })) as string[];
8
9 if (!newAddress) {
10   throw new Error('No account found');
11 }
12
13 const { data, error: updateErr } = await supabase
14   .from("profiles")
15   .update({ wallet_address: newAddress })
16   .eq("id", user.id)
17   .select();

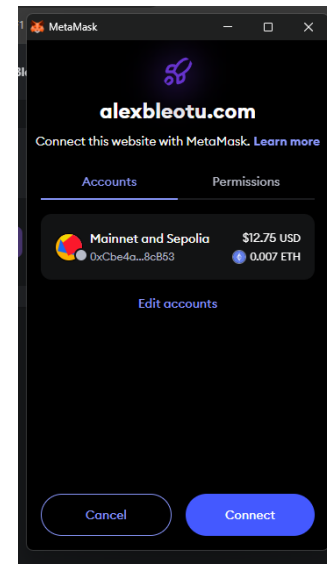
```

- Hook personalizat – **useMetaMask.ts**

```

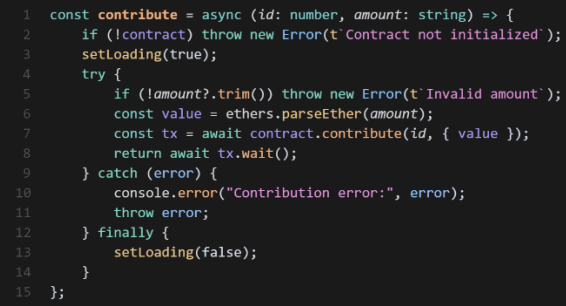
1 try {
2   if (!localStorage.getItem("walletAddress")) {
3     await connectWallet();
4
5     if (localStorage.getItem("walletAddress")) {
6       setIsConnected(true);
7       setIsLocked(false);
8       setError(null);
9       window.location.reload();
10    }
11   } else {
12     setIsConnected(true);
13     setIsLocked(false);
14     setError(null);
15   }
16 } catch (err: any) {
17   setError(err.message || 'Failed to connect to MetaMask');
18 }

```



3. Comunicare cu smart contract prin Ethers.js

- Hook personalizat – **useCampaignContract.ts**

A screenshot of a code editor with a dark background and light-colored text. The code is written in TypeScript and defines an asynchronous function named 'contribute'. The function takes two parameters: 'id' of type 'number' and 'amount' of type 'string'. It starts with a check for 'contract' and throws an error if it's not initialized. Then, it sets 'loading' to true and enters a try block. Inside the try block, it checks if 'amount' is empty and throws an error if so. It then parses the 'amount' using 'ethers.parseEther', creates a transaction 'tx' by calling 'contract.contribute', and returns the result of 'tx.wait()'. A catch block handles any errors by logging them to the console and re-throwing them. Finally, it sets 'loading' back to false. The function is exported as 'contribute' from the module.

```
1  const contribute = async (id: number, amount: string) => {  
2    if (!contract) throw new Error('Contract not initialized');  
3    setLoading(true);  
4    try {  
5      if (!amount?.trim()) throw new Error('Invalid amount');  
6      const value = ethers.parseEther(amount);  
7      const tx = await contract.contribute(id, { value });  
8      return await tx.wait();  
9    } catch (error) {  
10     console.error("Contribution error:", error);  
11     throw error;  
12   } finally {  
13     setLoading(false);  
14   }  
15 }
```