UNIVERSITATEA DIN BUCURESTI

AUCTION PLATFORM

Bejan Topse Lucian Cosmin, Olaru Elena, Trache Andrei

Facultatea de Matematică și Informatică
12 aprilie 2022

Abstract

Licitația online este considerată cel mai util mijloc de schimb cu cea mai rapidă creștere care a apărut din tehnologia comerțului electronic. Acestea revoluționează modul în care desfășurăm afaceri online.

Motivația în alegerea temei proiectului este evidențiată prin avantajele pe care putem avea prin aucțiunile online, printre care putem aminti:

- **Mai mulți cumpărători**: din cauza conflictelor, a distanței, a pandemiei, există un public mult mai larg dispus să participe la o licitație online. Acei cumpărători sunt diferiți (persoane mai tinere,care stau acasă, etc.) și au șanse mai mari să liciteze online.
- **Acoperire mai mare**: licitațiile online ajung la cumpărători din întreaga lume. Astfel, licitațiile online oferă posibilitatea de a găsi cumpărătorul potrivit pentru fiecare articol.
- **Comoditate**: cumpărătorii pot licita când și unde doresc și pot licita pentru mai multe licitații în aceeași zi.
- **Fără transport**: NFT-urile sunt achiziționate din aceeași locație, online. Sunt mai mulți bani în buzunar pe care i-ai fi cheltuit pe costurile de transport.

Cuprins

1 Tehnologii folosite **4**
1.1 Solidity 4
1.2 Hardhat 4
1.3 Angular 4
1.4 Ether.js 5
1.5 Pinata IPFS 5

2 Tema Proiectului **6**

3 Etape Implementare **7**
3.1 Hardhat 7
3.2 Solidity 10
3.2.1 Auction.sol 10
3.2.2 AuctionFactory.sol 14
3.2.3 AuctionNFT.sol 14
3.3 Angular 17
3.4 Pinata IPFS 20

Bibliografie **21**

Capitolul 1

Tehnologii folosite

1.1 Solidity

Este un limbaj orientat spre contract, care este conceput pentru a viza mașina virtuală Ethereum (EVM). Codul este tastat static, acceptă moștenire, biblioteci și tipuri complexe definite de utilizator, printre alte caracteristici.[5]

Deoarece contractul nostru se va baza pe Ethereum, vom folosi Solidity, un limbaj de programare folosit pentru crearea de contracte inteligente.

1.2 Hardhat

Este un mediu de dezvoltare Ethereum care îi ajută pe dezvoltatori să gestioneze și să automatizeze sarcinile recurente inerente construirii de contracte inteligente și DApps.[3]

Am ales să îl folosim ca mediu de dezvoltare pentru Solidity, pentru a facilita implementarea contractelor într-o rețea locală.

1.3 Angular

Este un framework structural JavaScript open-source pentru aplicatiile web prezentate ca o singura pagina dinamica. [1]

Proiectul stă la baza framework-ului Angular ca parte de front-end, prin simplitatea în crearea componentelor, a paginilor, facilitând de asemenea și o legătură mai ușoară cu partea de backend.

1.4 Ether.js

Biblioteca ethers.js își propune să fie o bibliotecă completă și compactă pentru interacțiunea cu Ethereum Blockchain și ecosistemul acestuia.[2]

Folosim Biblioteca client web Ethereum pentru a realiza legătura dintre Angular și Solidity, fiind un mod prin care vom interacționa cu contractul inteligent .

1.5 Pinata IPFS

Sistemul de fișiere interplanetare este o rețea de stocare distribuită formată din „noduri" sau computere din întreaga lume, unde oamenii și aplicațiile stochează și partajează date.[4]

Blockchain-ul nu se poate descurca cu stocarea unor cantități mari de date, deoarece devine extrem de costisitor să reproducă cantități mari de date pe acele mii de noduri. Pentru a stoca o imagine pe blockchain-ul Ethereum ar costa probabil zeci de mii de dolari. Din acest motiv, majoritatea datelor NFT-urilor trebuie să fie stocate în afara lanțului și trebuie să securizăm și aceste date.

Din acest considerent am ales Pinata IPFS pentru a stoca NFT-urile din licitații (folosind API-urile lor), fiind util pentru partajarea și stocarea fișierelor, după cum spune și definiția.

Capitolul 2

Tema Proiectului

Lucrarea constă într-o aplicație de licitații online, ce are la bază NFT-urile. Un user poate incărca o poză, să o transforme in NFT și să o vandă altui user, prin intermediul unei licitații, desfășurată într-o perioadă de timp determinată de cel care initiază aucțiunea.

Contractul nostru de licitație va avea o interfață simplă, realizată folosind Solidity, care permite utilizatorilor să plaseze oferte și, după finalizarea licitației, să își retragă fondurile, în cazul în care nu câștigă licitația. În privința proprietarului unei licitații, acesta crează aucțiunea, plecând de la un bid ales și o perioadă de timp determinată și reușește cu succes să își desfășoare licitația.

Când un utilizator pune la vânzare un NFT, dreptul de proprietate asupra articolului va fi transferat de la creator la contractul de piață. Când un utilizator cumpără un NFT, prețul de achiziție va fi transferat de la cumpărător la vânzător, iar articolul va fi transferat de pe piață la cumpărător.

Capitolul 3

Etape Implementare

3.1 Hardhat

După cum am precizat ulterior, noi am ales Hardhat pentru a rula un blockchain local și pentru a face deploy contractelor noastre.

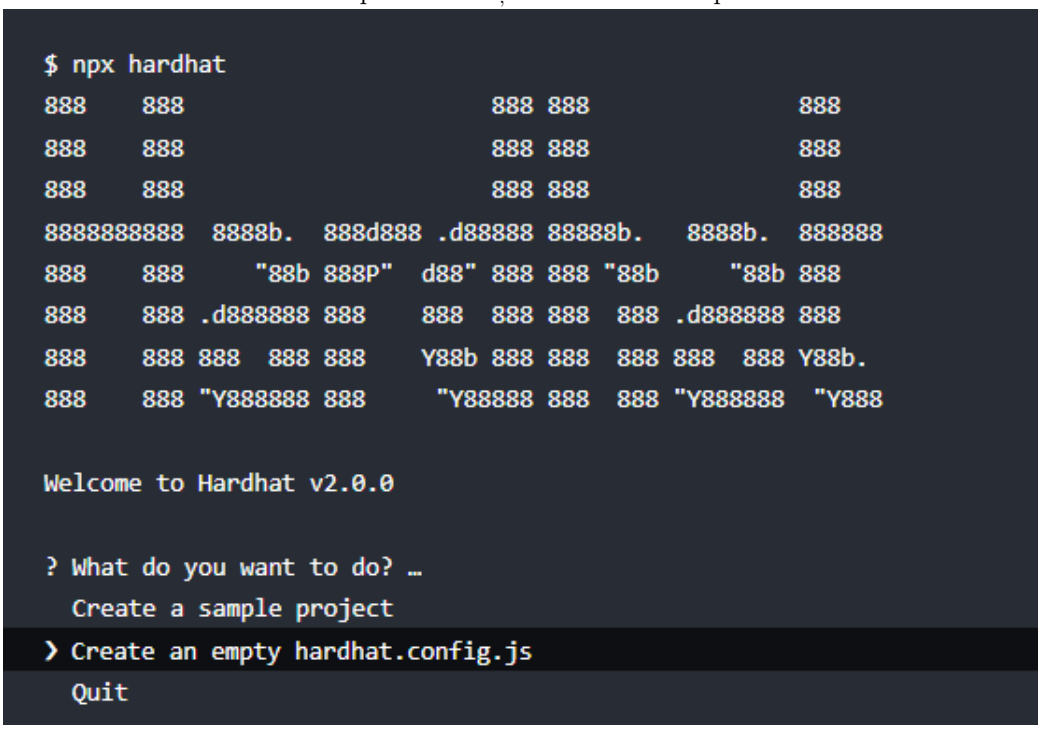
În acest scop, am implementat următorii pași:

- Am instalat Node.js (>=12.0).
- Am inițializat proiectul folosind Node.js și am instalat Hardhat.

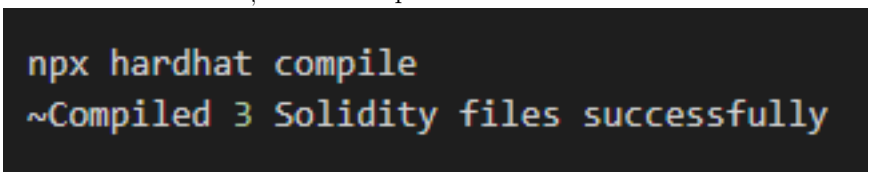
```
mkdir auction-dapp
cd auction-dapp
npm init
npm install --save-dev hardhat

npx hardhat
```

- Am rulat comanda următoare pentru a inițializa Hardhat în proiect.



- Am scris contractele și le-am compilat.



- Am pornit un blockchain local folosind Hardhat.

- Am adăugat un network entry în Hardhat.config.js, variabilele secrete reprezentând parametrii blockchain-ului local.

```
const secrets = require("./environment/secrets.json");


module.exports = {
  solidity: "0.7.1",
  paths:{
    sources:"./blockchain/contracts",
    tests:"./blockchain/test",
    cache:"./blockchain/cache",
    artifacts:"./blockchain/artifacts"
  },
  networks:{
    local:{
      url: secrets.localnode,
      accounts: [secrets.privateKeyLocalAccount]
    }
  }
};
```

- Am dat deploy pe network-ul local contractelor compilate și am salvat adresele lor într-un fișier json.

```
npx hardhat run ./blockchain/scripts/deploy.js --network local
```

```
const fs = require('fs');
const { ethers } = require('hardhat');

async function main(){

    const AuctionFactoryContract = await ethers.getContractFactory("AuctionFactory");
    const AuctionNFT = await ethers.getContractFactory("AuctionNFT");

    const auctionFactory = await AuctionFactoryContract.deploy();
    const auctionNFT = await AuctionNFT.deploy();

    await auctionFactory.deployed();
    await auctionNFT.deployed();

    console.log("The auction factory contract was deployed to: " + auctionFactory.address);
    console.log("The auction nft contract was deployed to: " + auctionNFT.address);

    let addresses = {
        "auctionFactoryContract": auctionFactory.address,
        "auctionNFTContract":auctionNFT.address
    };
    let addressesJSON = JSON.stringify(addresses);
    fs.writeFileSync("environment/contract-address.json", addressesJSON);

}

main()
.then(() => {
    process.exit(0);
})
.catch((error) => {
    console.error(error);
    process.exit(1);
})
```

3.2 Solidity

Am implementat 3 contracte folosind limbajul Solidity.

3.2.1 Auction.sol

Acest contract conține următoarele variabile:

- **beneficiary**: adresa beneficiarului (vânzătorul);
- **nftId**: id-ul nft-ului pus la licitație;

- **endTime**: timpul la care licitația se incheie (în secunde).
- **itemName**: numele itemului.
- **highestBidder**: utilizatorul care a licitat cel mai mult;
- **highestBid**: cel mai mare bid licitat;
- **initialBid**: bidul initial, setat de licitator.
- **timesUp**: ne indică dacă licitația este incă în desfășurare sau nu;
- **pendingReturns**: salvează pentru fiecare licitator suma pe care o poate retrage;

Funcții implicate:

- **bid()**

```
function bid() public payable {
    require(block.timestamp <= endTime, "Auction is over.");
    require(msg.value > highestBid, "Bidd is too low.");
    require(
        msg.sender != beneficiary,
        "You can't bid on your own auction."
    );
    require(msg.value > initialBid, "Bidd lower than the initial bidd.");

    if (highestBid != 0) {
        pendingReturns[highestBidder] += highestBid;
    }

    highestBidder = msg.sender;
    highestBid = msg.value;

    emit BidIncreased(highestBidder, highestBid);
}
```

Funcție apelată când un utilizator vrea să liciteze. Sunt verificate mai multe condiții, highestBidder și highestBid sunt actualizate și pendingReturns este înnoit pentru fostul highestBidder.

- **withdraw()**

```
function withdraw() public returns (bool) {
    require(msg.sender != highestBidder, "You are the highest Bidder.");
    require(pendingReturns[msg.sender] > 0, "Your pending returns are 0.");
    uint256 amount = pendingReturns[msg.sender];
    if (amount > 0) {
        pendingReturns[msg.sender] = 0;
        if (!payable(msg.sender).send(amount)) {
            pendingReturns[msg.sender] = amount;
            return false;
        }
    }
    return true;
}
```

Funcție apelată când un highestBidder anterior vrea să-și retragă banii.

- **auctionEnd()**

```
function auctionEnd() external {
    require(block.timestamp >= endTime, "Auction not yet finished.");
    require(!timesUp, "AuctionEnd already called.");
    require(highestBid > 0, "There is no highest bidd.");
    require(
        msg.sender == beneficiary,
        "Only the seller can end the auction."
    );

    timesUp = true;

    beneficiary.transfer(highestBid);
}
```

Funcție apelată de vânzător pentru a încheia licitația.

- **Mai multe funcții view pentru a returna informații necesare pe frontend**

```
function getTotalBalanceOfContract() public view returns (uint256) {
    return address(this).balance;
}

function auctionEnded() public view returns (bool) {
    return timesUp;
}

function fHighestBid() public view returns (uint256) {
    return highestBid;
}

function getInitialBid() public view returns (uint256) {
    return initialBid;
}

function fHighestBiddder() public view returns (address) {
    return highestBidder;
}

function senderPendingReturns() public view returns (uint256) {
    return pendingReturns[msg.sender];
}

function getItemName() public view returns (string memory) {
    return itemName;
}

function getItemEndTime() public view returns (uint256) {
    return endTime;
}

function getNFTId() public view returns (uint256) {
    return nftId;
}

function getBeneficiary() public view returns (address) {
    return beneficiary;
}
```

3.2.2 AuctionFactory.sol

Există o singură variabilă, o listă cu adresele acțiunilor create.

- **auctionsAdresses**

```
address[] public auctionsAddresses;
```

Funcții implicate:

- **newAuction**

```
function newAuction(
    uint256 nftId,
    string memory name,
    uint256 initialBid,
    uint256 time
) public returns (Auction newContract) {
    Auction auc = new Auction(nftId, msg.sender, time, name, initialBid);
    auctionsAddresses.push(address(auc));
    return auc;
}
```

Este creat un nou obiect de tip Auction, iar adresa lui este salvată în listă

- **O funcție view** pentru a returna adresele acțiunilor create.

```
function getAllAuctionsAddresses()
    public
    view
    returns (address[] memory listOfAuctions)
{
    return auctionsAddresses;
}
```

3.2.3 AuctionNFT.sol

Acest contract implementează token-ul ERC721 din @openZeppelin.
Constructorul care implementează ERC721.

```
constructor() public ERC721("AuctionNFT", "ANFT") {
}
```

Variabile implicate:

- **tokenIDs**: Variabilă Counter pentru inițializarea ID-urilor unice.
- **userOwnedTokens**: Un mapping de la adresă la listă de int-uri, care salvează pentru fiecare user ID-urile NFT-urilor proprii.
- **tokenIsAtIndex**: Un mapping de la int la int în care pentru fiecare NFT se salvează index-ul pe care se află în lista din mapping-ul anterior.
- **nftIds**: O listă cu ID-urile tuturor NFT-urilor

```
using Counters for Counters.Counter;
Counters.Counter private _tokenIds;
mapping(address => uint256[]) public userOwnedTokens;
mapping(uint256 => uint256) public tokenIsAtIndex;
uint256[] public nftIds;
```

Funcții implicate:

- **mintNFT()**: primește URI-ul unui token, crează un ID, face modificările necesare in variabilele din contract și apelează _mint și _setTokenURI din contractul din @openZeppelin.

```
function mintNFT(string memory tokenURI)
    public
    returns (uint256)
{
    _tokenIds.increment();

    uint256 newItemId = _tokenIds.current();
    nftIds.push(newItemId);
    userOwnedTokens[msg.sender].push(newItemId);
    uint256 arrayLength = userOwnedTokens[msg.sender].length;
    tokenIsAtIndex[newItemId] = arrayLength - 1;

    _mint(msg.sender, newItemId);
    _setTokenURI(newItemId, tokenURI);
    return newItemId;
}
```

- **transferNFT() & deleteNftFromIndex()**: primește adresa de la care se transferă NFT-ul, adresa la care se transferă NFT-ul și ID-ul NFT-ului. În primă fază se va apela approve() pentru noul owner, apoi i se va transfera NFT-ul apelând funcția safeTransferFrom(), iar într-un final se va apela deleteNftFromIndex pentru a se face modificările necesare în variabilele din contract.

```
function deleteNftFromIndex(uint index, address owner) private{
    uint256[] storage array = userOwnedTokens[owner];
    array[index] = array[array.length-1];
    uint256 nftIdLocal = array[array.length -1];
    array.pop();

    tokenIsAtIndex[nftIdLocal] = index;
    userOwnedTokens[owner] = array;
}

function transferNft(address from, address to, uint256 nftId) public {
    approve(to,nftId);
    safeTransferFrom(from,to,nftId);
    uint index = tokenIsAtIndex[nftId];
    deleteNftFromIndex(index, from);

    userOwnedTokens[to].push(nftId);
    uint256 arrayLength = userOwnedTokens[to].length;
    tokenIsAtIndex[nftId] = arrayLength;
}
```

- **Funcții view**

```
function getNFTIdsForUser()
    public
    view
    returns (uint256[] memory)
{
    return userOwnedTokens[msg.sender];
}

function getNftIds() public view returns (uint256[] memory) {
    return nftIds;
}
```

3.3 Angular

Implementarea contractelor în Angular am realizat-o folosind librăria Ethers.js. Astfel, am urmat următorii pași:

- Instalare pachete Node.js:

```
npm install --save ethers
```

- Instalăm Angular și creăm un proiect pe care îl rulăm:

```
ng new auction dapp
ng serve
```

- În proiectul Angular creăm un nou fișier de servicii în care vom implementa tot ce este necesar din libraria Ethers.js.

  – Importăm librăria ethers.js si declarăm variabila "window" în care metamask injectează anumite informații necesare nouă in cod.

```
declare let window: any;
import { ethers } from "ethers";
```

  – Prima funcție apelată la încărcarea paginii este **checkIfMetamaskUserIsValid()** în care preluăm adresa user-ului logat în MetaMask.

```
async checkIfMetamaskUserIsValid(): Promise<boolean> {
  if (window.ethereum) {
    console.log({ window, eth: window.ethereum })
    console.log("Exista user metamask");
    try {
      const accounts = await window.ethereum.request({
        method: "eth_requestAccounts",
      });
      console.log("Accountul din metamask:", accounts);
      this.account = accounts[0];
      return true;
    } catch (error) {
      console.log("Error...")
    }
  } else {
    console.log("Nu exista user metamask");
  }
  return false;
}
```

– În continuare ne vom crea un provider (conexiunea cu network-ul Ethereum) și un signer (obiectul folosit pentru a semna tranzacții.) . Tot aici verificăm și dacă pagina este pe network-ul potrivit, în caz contrar dând reload, și, de asemenea, dacă signer-ul are chainid-ul network-ului nostru.

```
const checkIfMetamaskUserIsValid = await this.checkIfMetamaskUserIsValid();
if (checkIfMetamaskUserIsValid) {
  const provider = new ethers.providers.Web3Provider(window.ethereum);
  console.log("Provider:", provider);

  provider.on("network", (newNetwork: any, oldNetwork: any) => {
    if (oldNetwork) {
      window.location.reload();
    }
  });

  this.signer = provider.getSigner();
  console.log("Signer:", this.signer);

  if (await this.signer.getChainId() !== 31337) {
    alert("Wrong network");
  }
```

– Vom instanția contractele AuctionFactory și AuctionNft, folosind addresele salvate pe local, ABI-ul rezultat după compilarea contractelor folosind Hardhat și signer-ul declarat anterior.

```
this.auctionFactoryContract = new ethers.Contract(addresses.auctionFactoryContract, AuctionFactory.abi, this.signer);
this.nftContract = new ethers.Contract(addresses.auctionNFTContract, AuctionNFT.abi, this.signer);
```

- În continuare declarăm funcții getter pentru fiecare element important din fișierul de servicii: AuctionFactoryContract, AuctionNFTContract, Account(contul conectat pe metamask), Signer; pentru ca acestea sa fie vizibile in toate componentele din proiectul Angular.

Astfel, putem interacționa cu contractele scrise in Solidity. În ethers.js un apel spre o funcție din contract se face in felul următor:

```
async bidOnAuction() {
  const account = this.metamaskService.getAccount();
  if (this.bidValue != 0) {
    this.auction.bid({ from: account, value: ethers.utils.parseEther(this.bidValue.toString()) }).then(
      (responseBid: any) => {
        this.notifier.notify("success","Bid succesfully.");
        console.log("Bid placed succesfully.", responseBid);
        responseBid.wait().then(() => {
          this.itemHighestBidd = this.bidValue.toString();
        })
      }
    ).catch(
      (error: any) => {
        this.notifier.notify("error","Bidul nu a fost creat");
        console.log(error);
      }
    );
  }
}
```

Aceste apeluri crează obiecte de tip Promise din Javascript, dupa care practic trebuie să așteptăm să ruleze. Acesta este apelul spre funcția bid() din Auction.sol, care nu are parametrii direcți, în schimb are 2 parametrii ”speciali”, unul notat cu ”from” care asociază valoarea trimisă lui ”msg.sender” din solidity, și unul notat cu ”value” care asociază valoarea trimisă lui ”msg.value”. În ”.then()” se va intra în cazul în care apelul functiei are un răspuns pozitiv, în caz contrar (daca nu se trece de un require() din Solidity) se va intra în ”.catch()”.

În această manieră se apelează toate funcțiile din contractele noastre.

3.4 Pinata IPFS

Pentru a stoca imaginile folosite ca NFT-uri, am ales să folosim API-ul de la Pinata IPFS. Pentru a salva o imagine, este nevoie să facem un request de tip POST:

```
async pinFileToIPFS(imgBuffer: any,) {
    let formData = new FormData();
    let endPoint = secrets.pinataApiEndpoint + 'pinning/pinFileToIPFS';

    formData.append('file', imgBuffer);

    const res = await axios
        .post(endPoint, formData, {
            maxBodyLength: Infinity, //this is needed to prevent axios from erroring out with large files
            headers: {
                'Content-Type': `multipart/form-data;`,
                pinata_api_key: secrets.pinataApiKey,
                pinata_secret_api_key: secrets.pinataApiSecret
            }
        });

    if (res) {
        return {
            succes: true,
            image: 'https://gateway.pinata.cloud/ipfs/' + res.data.IpfsHash
        }
    } else {
        return {};
    }
}
```

În urma request-ului primim un URL pe care a fost salvată poza. Pe acesta îl com pune ulterior in URI-ul token-ului cand apelăm funcția de mint.

Bibliografie

[1] *Angular*. URL: https://angular.io/docs. accesat: 04.04.2022.

[2] *Ether.js*. URL: https://docs.ethers.io/v5/. accesat: 04.04.2022.

[3] *HardHat*. URL: https://hardhat.org/getting-started/. accesat: 04.04.2022.

[4] *Pinata IPFS*. URL: https://docs.pinata.cloud. accesat: 04.04.2022.

[5] *Solidity*. URL: https://docs.soliditylang.org/en/v0.8.13/. accesat: 04.04.2022.