Voting System

Blockchain project

▶ Cuprins

Introducere

Acest proiect cuprinde un sistem de votare implementat folosind tehnologia Blockchain și este creat de: Ana-Maria Comorașu, Alexandru Țifui, Laura Tender.

Proiectul a fost dezvoltat în cadrul cursului Blockchain din anul al III-lea la Facultatea de Matematică și Informatică, Universitatea din București.

Sistemul de votare permite unui proprietar să își creeze o alegere cu scop sau un titlu și să adauge candidați. Proprietarul este cel care deschide și închide alegerile și cel care trebuie să autorizeze persoanele care participă la vot. Sistemul nostru de votare cuprinde și o interfață pentru a arată progresul alegerilor.

Tehnologii

- Solidity
- Javascript
- Truffle
- Ganache
- MetaMask
- Web3.js
- Nodejs

Usage

Trebuie să instalăm următoarele: truffle, ganache, metamask.

Comenzile de care avem nevoie pentru a porni aplicația:

```
conda activate blockchain
ganache -p 7545 -i 5777
truffle migrate --reset all
node client/src/script.js
```

Apoi intrăm pe localhost:3000 și ne conectăm cu MetaMask.

Pentru a porni alegerile, adminul folosește truffle console astfel:

```
truffle console
ballot = await Ballot.deployed()
ballot.grantVoter(adresaUser)
ballot.startBallot()
```

Putem verifica că alegerile au fost pornite astfel:

```
ballot.electionState()
>in_progress
```

Pentru a închide alegerile:

```
ballot.endBallot()
```

Implementarea sistemului de votare

Implementarea sistemului de votare poate fi găsită în fișierul `Ballot.sol`.

- Crearea, deschiderea și închiderea alegerilor

Această parte este realizată de *Ana Comorașu* și cuprinde crearea clasei Ballot, constructorul, deschiderea, închiderea și starea alegerilor.

În constructor, alegerea noastră cuprinde un titlu, un proprietar, o listă de candidați și starea votării este marcată ca registered.

```
constructor(string memory _title, bytes32[] memory _candidateList){
    title = _title;
    chairperson = msg.sender;
    for(uint i = 0; i < _candidateList.length; i++){
        candidatesList.push(Candidate({
            name: _candidateList[i],
            numVotes: 0
        }));
    }
    state = VotingState.Registered;
}
```

Votarea poate avea trei stări: nu a început (registered), în desfășurare, încheiată. Verificarea stărilor are loc prin modifierii: `inProgress`, `notStarted`, `finished`. Funcția `startBallot` deschide alegerile, iar funcția `endBallot` le încheie.

```
  function startBallot() public notStarted isChairperson {
      state = VotingState.Progress;
  }

  function endBallot() public inProgress {
```

```
        state = VotingState.Completed;
    }
```

- Verificarea dreptului de proprietar și autorizarea votanților

Această parte este realizată de *Laura Tender*.

Verificarea dreptului de proprietar este implementată cu ajutorul modifierului `isChairperson`, prin care ne asigurăm că senderul este persoană care a creat alegerile. Aceast modifier este folosit pentru deschiderea și închiderea alegerilor, cât și pentru autorizarea votanților.

```
    modifier isChairperson(){
        require(msg.sender == chairperson, "Only the chairperson is
allowed.");
        _;
    }
```

Un votant se află în una dintre stările: neautorizat (`NotGranted`), autorizat(`Granted`) și care a votat deja (`Voted`). Votanții sunt reținuți într-un dicționar în care cheia este adresa lor, iar valoarea este starea sa. Autorizarea unui votant are loc în funcția grantVoter prin care votantul este adăugat în dicționar cu starea granted.

```
    function grantVoter(address _voter) public notStarted isChairperson {
        alreadyVoted[_voter] = VoterState.Granted;
    }
```

- Votarea și rezultatele alegerilor

Această parte este realizată de *Alexandru Țifui*.

Votarea are loc prin funcția `vote`, dacă alegerile sunt încă în desfășurare și dacă persoană are drept de vot. Odată ce este condiții sunt îndeplinite, verificăm că opțiunea de vot este una validă, creștem cu un vot numărul de voturi pentru acest candidat și schimbăm starea votantului din `Granted` în `Voted`.

```
    function vote(uint _choice) public inProgress canVote {
        // * check if the choice is in the candidacy list
        require(_choice < candidatesList.length, "There is no choice for
this");
        // * add vote
        candidatesList[_choice].numVotes++;
        alreadyVoted[msg.sender] = VoterState.Voted;
    }
```

Funcțiile `nthCandidate` și `nthNoVotes` returnează numele, respectiv numarul de voturi ale candidatului cu indexul `n`. Acestea verfică dacă candidatul cu indexul `n` există.

```
  function nthCandidate(uint _id) public view finished returns (string
memory) {
      require(_id < candidatesList.length, "Candidate index out of
range");
      return candidatesList[_id].name;
  }
```

Client

- Migrații

Proiectul cuprinde 2 migrații: `1_initial_migration.js` și `2_deploy_ballot.js`.

Prima migrație face deploy la contractul `Migrations` (care poate fi găsit în `contracts/Migrations.sol`).

A doua migrație face deploy la contactul Ballot (are poate fi găsit în `contracts/Ballot.sol`). Astfel sunt create alegeri cu titlul "USA Election 1904" și lista de candidați "Candidate 1", "Candidate 2", ..., "Candidate 10".

- Legarea front end-ului cu smart contractul

În fișierul `utils.js` am creat funcțiile:

- `getWeb3`: pentru a instanția un obiect de tip web3. Am folosit window.ethereum ca provider și window.ethereum.request pentru a cere permisiunea pentru a accesa conturile.
- `getContract`: pentru a crea o instanță a contractului.

- Interacționarea cu smart contractul

Fișierul `api-calls.js` conține câteva funcții ajutătoare pentru a apela metodele din smart contract.

Funcțiile `getElectionState`, `callVote`, `getCandidatesNo`, `getNthCandidate` și `getNthResult` apelează metodele corespunzătoare și tratează excepțiile.

Fișierul `index.js` conține funcțiile `constructErrorCard`, `connectToContract`, `vote`, `getVotes`, `getResults` și `createCandidateResult` și face legătura dintre html și funcționalități.