PROGETTO SDC 21/22

Gabriele Benedetti, Emanuele Izzo,
Giulia Pascale, Vlad Cristinel Simon

Introduzione

Lo smart contract

Analisi dei requisiti

Parte di Vlad e Emanuele

Tecnologie utilizzate

Esempi di utilizzo

INTRODUZIONE

Lo scopo di questo progetto è quello di sviluppare un sistema distribuito che fornisca uno strumento per la creazione e la gestione di carte fedeltà per la raccolta punti di alcune catene di negozi. L'idea è quella di facilitare le transazioni dei punti delle carte che appartengono ad un singolo negozio o ad un consorzio di negozi da parte del personale.

INTRODUZIONE

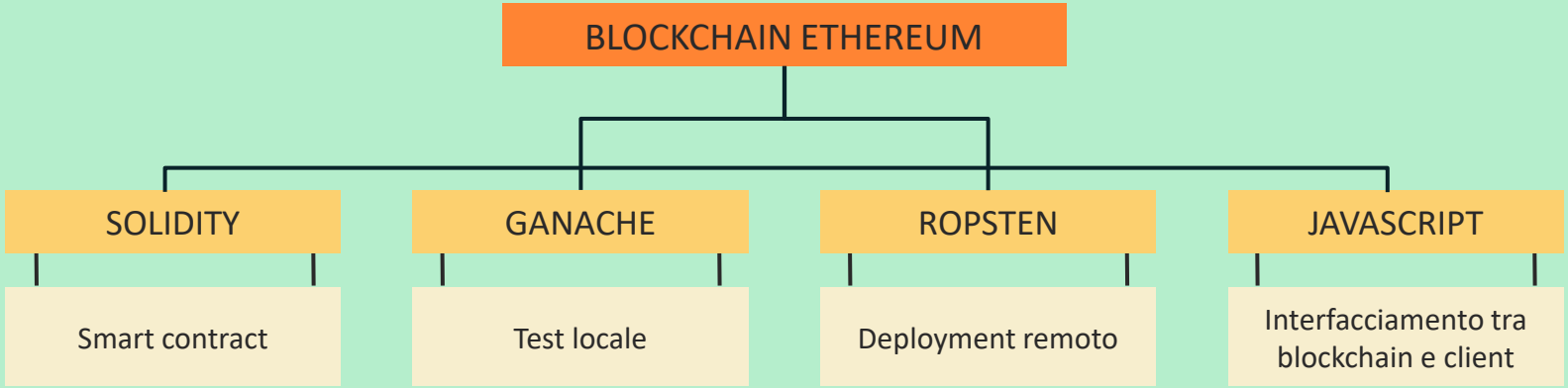
Abbiamo utilizzato una blockchain piuttosto che un database centralizzato perché dovevamo fare il progetto di SDC.

Analisi dei requisiti

Il sistema deve:

- Permettere al dipendente di un negozio di creare una nuova carta fedeltà per un cliente.
- Determinare l’appartenenza di una carta ad un negozio o ad un consorzio di negozi.
- Permettere al dipendente di un negozio di gestire i punti se la carta appartiene al medesimo negozio del dipendente o al medesimo consorzio di quel negozio.
- Mostrare al cliente i suoi punti.
- Creare un nuovo consorzio ed aggiungere e rimuovere negozi da questo.

TECNOLOGIE UTILIZZATE



LO SMART CONTRACT

FUNZIONI PRINCIPALI

- **addCliente**

```
/**
* @dev Funzione che crea un nuovo cliente e gli assegna un codice univoco
* Questa funzione può essere invocata dal proprietario o da un funzionario
* @param nome nome del cliente
* @param cognome cognome del cliente
* @param data_nascita data di nascita del cliente
* @return ID generato
*/
function addCliente(string memory nome, string memory cognome, string memory data_nascita) public onlyFunzionarioOrOwner returns(bytes32){
    bytes32 ID = keccak256(abi.encodePacked(nome, cognome, data_nascita));
    Cliente memory c = Cliente(ID, nome, cognome, data_nascita, "null");
    clienti[c.id] = c;
    emit NuovoCliente(ID, nome, cognome, data_nascita);
    return ID;
}
```

- **addFunzionario**

```
/**
* @dev Funzione che crea un nuovo funzionario e gli assegna un codice univoco
* Questa funzione può essere invocata solo dal proprietario
* @param id address del funzionario
* @param n negozio nel quale opera il funzionario
*/
function addFunzionario(address id, Negozio n) public onlyOwner{
    Funzionario memory f = Funzionario(address(id), Negozio (n));
    funzionari[id] = f;
    ruoli[Ruolo.Funzionario].push(f.id);
    negozio_funzionario[f.id] = n;
}
```

- addFunzionario

LO SMART CONTRACT

FUNZIONI PRINCIPALI

- addCarta

```
/**
* @dev Funzione che crea una nuova carta per un cliente e gli assegna un codice univoco
* Questa funzione può essere invocata dal proprietario o da un funzionario
* @param c cliente per il quale va creata la carta
*/
function addCarta(Cliente memory c) public onlyFunzionarioOrOwner returns(bytes32) {
    uint date = block.timestamp;
    bytes32 ID = keccak256(abi.encodePacked(c.nome, c.cognome, c.dataDiNascita, date));
    Carta memory i = Carta(ID, c.id, 0, funzionari[msg.sender].negozio);
    carte[ID] = i;
    emit NuovaCarta(ID, c.id);
    return ID;
}
```

- addConsorzio

```
function addConsorzio(string memory nome) public onlyOwner {
    bytes32 ID = keccak256(abi.encodePacked(nome));
    Negozio[] memory n;
    Consorzio memory c = Consorzio(ID, nome, n);
    consorzi[ID] = c;
    emit NuovoConsorzio(ID, nome);
}
```

- addNegozioConsorzio

```
function addNegozioConsorzio(Negozio negozio, bytes32 id_consorzio) public onlyOwner{
    consorzioNegozi[negozio] = id_consorzio;
    consorzi[id_consorzio].negozi.push(negozio);
}
```

LO SMART CONTRACT

FUNZIONI PRINCIPALI

- **checkAuthority**

```
/**
*@dev Funzione che controlla se c'è l'autorità di operare
* [1] la carta deve essere dello stesso negozio in cui opera il funzionario
* Oppure
* [2] la carta deve appartenere allo stesso consorzio di negozi in cui opera il funzionario (carta di un negozio diverso, stesso consorzio)
*@param id_carta id univoco della carta su cui si vuole effettuare l'operazione
*/
function checkAuthority(bytes32 id_carta) public view returns (bool){
    bytes32 id_consorzio_carta = consorzioNegozi[carte[id_carta].negozio];
    bytes32 id_consorzio_funzionario = consorzioNegozi[funzionari[msg.sender].negozio];
    return (carte[id_carta].negozio == funzionari[msg.sender].negozio || id_consorzio_carta == id_consorzio_funzionario);
}
```

- **addPunti**

```
/**
* @dev Funzione che aggiunge nuovi punti ad una carta
* @param id_carta codice univoco della carta
* @param numeroPunti numero dei punti da aggiungere alla carta
*/
function addPunti(bytes32 id_carta, uint numeroPunti) public {
    if(checkAuthority(id_carta) == true)
        carte[id_carta].punti += numeroPunti;
}
```

- **getPunti**

```
function getPunti(bytes32 id_carta) public view returns(uint){
    return carte[id_carta].punti;
}
```

LO SMART CONTRACT

FUNZIONI PRINCIPALI

- **addCliente**